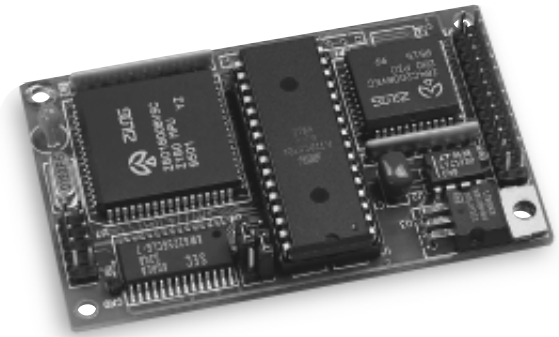


BL1400

C-Programmable Controller

User's Manual

Revision 2

Z-World • BL1400

User's Manual • Part Number 019-0017-02
Revision 2 • 021-0026-02 • Printed in U.S.A.
Last revised by RPB/TI on September 21, 1998

Copyright

Trademarks

- Dynamic C® is a registered trademark of Z-World
- PLCBus™ is a trademark of Z-World
- Windows® is a registered trademark of Microsoft Corporation.
- Hayes Smart Modem™ is a 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.

Company Address

Z-World
2900 Spafford Street
Davis, California 95616-6800 USA

Telephone: (530) 757-3737
Facsimile: (530) 753-5141
24-Hour FaxBack: (530) 753-0618
Web Site: http://www.zworld.com
E-Mail: zworld@zworld.com

TABLE OF CONTENTS

About This Manual vii

BL1400 Overview 1-1

BL1400 Overview 1-2
BL1400 Features 1-3
Developer's Kit 1-4
Networking 1-4
Software Development and Evaluation Tools 1-5

Getting Started 2-1

Initial BL1400 Setup 2-2
Connecting the BL1400 to a Host PC 2-2
Via Development Board 2-2
Via Flash EPROM 2-5
Via Regular EPROM 2-7
Running Dynamic C 2-8
Test the Communication Line 2-8
Selecting Communications Rate, Port, and Protocol 2-8
Running a Sample Program 2-8
Power Options 2-9

BL1400 Subsystems 3-1

Operating Modes 3-2
Changing Baud Rate on the BL1400 3-2
Power Connections 3-2
Interface Description 3-3
Parallel Input/Output (PIO) Chip 3-4
PIO Operation Modes 3-6
Control Register Byte Sequence 3-7
Header H2 Signals (RS-232 Port) 3-9
Header H3 Signals (PIO Ports) 3-9
Serial Communication 3-10
RS-232 Communication 3-10
Receive and Transmit Buffers 3-10
Echo Option 3-11
CTS/RTS Control 3-11
XMODEM File Transfer 3-11
Modem Communication 3-11

RS-485 Communication 3-12
Developing an RS-485 Network 3-12
Hardware Connection 3-13
Regulator 3-14
Supervisor 3-14
Real-Time Clock (RTC) 3-15
555 Timer 3-16
Using Thermistors 3-18
Memory 3-19
Direct Programming of the Serial Ports 3-20
Attainable Baud Rates 3-21
Z180 Serial Ports 3-21
Asynchronous Serial Communication Interface 3-23
ASCI Status Registers 3-23
ASCI Control Register A 3-25
ASCI Control Register B 3-26
SS (Source/Speed Select) 3-26
DR (Divide Ratio) 3-26
PEO (Parity Even/Odd) 3-26
–CTS/PS (Clear to Send/Prescaler) 3-26
MP (Multiprocessor Mode) 3-27
MPBT (Multiprocessor Bit Transmit) 3-27

Software Reference 4-1

Software Development Options 4-2
Dynamic C Development Software 4-2
Dynamic C Manuals 4-2
Programmable Input/Output 4-3
PIO Ports A and B 4-3
Shadow Registers 4-3
Function Prototypes 4-4
Real-Time Clock Integrated Circuit 4-5
Global Time and Date Structure 4-5
555 Timer 4-6
Flash EPROM 4-7
Serial Communication Software 4-8
Interrupt Handling for Z180 Port 0 4-8
RS-232 Software Support 4-9
XMODEM Commands 4-11
Miscellaneous Functions 4-12
Master-Slave Networking 4-14
Miscellaneous Functions 4-15

Support Libraries and Sample Programs 4-16
Sample Programs 4-16

Appendix A: Troubleshooting A-1
Out of the Box A-2
Dynamic C Will Not Start A-2
Dynamic C Loses Serial Link A-3
BL1400 Resets Repeatedly A-3
Input/Output Problems A-3
Power-Supply Problems A-3
Common Programming Errors A-4

Appendix B: Specifications B-1
Electrical and Mechanical Specifications B-2
BL1400 ..... B-2
Development Board B-4
Prototyping Board B-5
Base Plate B-6
Connectors B-7
Temperature B-7
Power B-7
Headers and Jumpers B-8
BL1400 ..... B-8
Development Board B-9

Appendix C: Power Management C-1
Direct Current Input C-2
Power Regulator C-2
Maximum Power Dissipation C-2
Heat Dissipation with the BL1400 Base Plate C-3
Heat Dissipation without the Base Plate C-4

Appendix D: Prototyping Board D-1
Prototyping Board D-2
Installing the Prototyping Board D-4
Sample Circuits D-5
LEDs D-5
Switches D-5
Jumpers D-5
Buzzer D-6
RC Filter D-6
Thermistor D-6

Appendix E: PLCBus **E-1**
PLCBus Overview E-2
Allocation of Devices on the Bus E-6
4-Bit Devices E-6
8-Bit Devices E-7
Expansion Bus Software E-7

Appendix F: Simulated PLCBus Connection **F-1**
PIO Port Connections F-2
Expansion Boards F-2
Liquid Crystal Displays and Keypads F-3
Software Drivers F-5
Using Expansion Boards with PIO Port A F-5
Using an LCD with PIO Port A F-8
Using a Keypad with PIO Ports A and B F-9

Appendix G: Input/Output Maps and Interrupt Vectors **G-1**
Memory Map G-2
Internal Input/Output Registers G-2
Other Input/Output Addresses G-4
Real-Time Clock (RTC) Registers G-4
Interrupt Vectors G-6
Interrupt Priorities G-7

Appendix H: Simulated EEPROM **H-1**

Index

ABOUT THIS MANUAL

This manual provides instructions for installing, testing, configuring, and interconnecting Z-World's BL1400 series controllers.

The term "BL1400" will be used generically throughout this manual when referring to any controller in the BL1400 series. Where information applies to a specific controller, the model number will be specified. The models currently covered by this manual include the BL1400 and the BL1410.

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 a target system that is controlled by a controller with relay expansion boards.
- 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
  C: A Reference Manual by Harbison and Steel

- 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 is a list of acronyms that may be used in this manual.

Table 1. Acronyms

Acronym	Meaning
EPROM	Erasable Programmable Read Only Memory
EEPROM	Electronically Erasable Programmable Read Only Memory
NMI	Nonmaskable Interrupt
PIO	Parallel Input / Output Circuit (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

Conventions

Table 2 lists and defines typographical conventions that may be used in this manual.

Table 2. Typographical Conventions

Example	Description
`while`	Courier font (bold) indicates a program, a fragment of a program, or a Dynamic C keyword or phrase.
`// IN-01…`	Program comments are written in Courier font, plain face.
Italics	Indicates that something should be typed instead of the italicized words (e.g., in place of *filename*, type a file's name).
Edit	Sans serif font (bold) signifies a menu or menu selection.
. . .	An ellipsis indicates that (1) irrelevant program text is omitted for brevity or that (2) preceding program text may be repeated indefinitely.
[ ]	Brackets in a C function's definition or program segment indicate that the enclosed directive is optional.
< >	Angle brackets occasionally enclose classes of terms.
a | b | c	A vertical bar indicates that a choice should be made from among the items listed.

Programming Pseudo Types

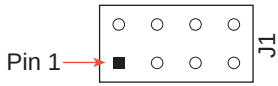
For convenience, this manual uses the following pseudo types.

- **`uint`** means **`unsigned integer`**
- **`ulong`** means **`unsigned long`**

These pseudo types are not standard C keywords; therefore, they will not function in an application unless first declared with **`typedef`** or **`#define`**.

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.

Icons

Table 3 displays and defines icons that may be used in this manual.

Table 3. Icons

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

BL1400 OVERVIEW

Chapter 1 provides an overview and a brief description of the BL1400 features and options.

BL1400 Overview

The BL1400 is a compact and powerful programmable controller. Its small size, low cost, and versatility make it ideal for a wide range of embedded control applications. The BL1400 accepts Z-World's 4-bit expansion boards through its PIO ports.

Figure 1-1 illustrates the BL1400 board layout.

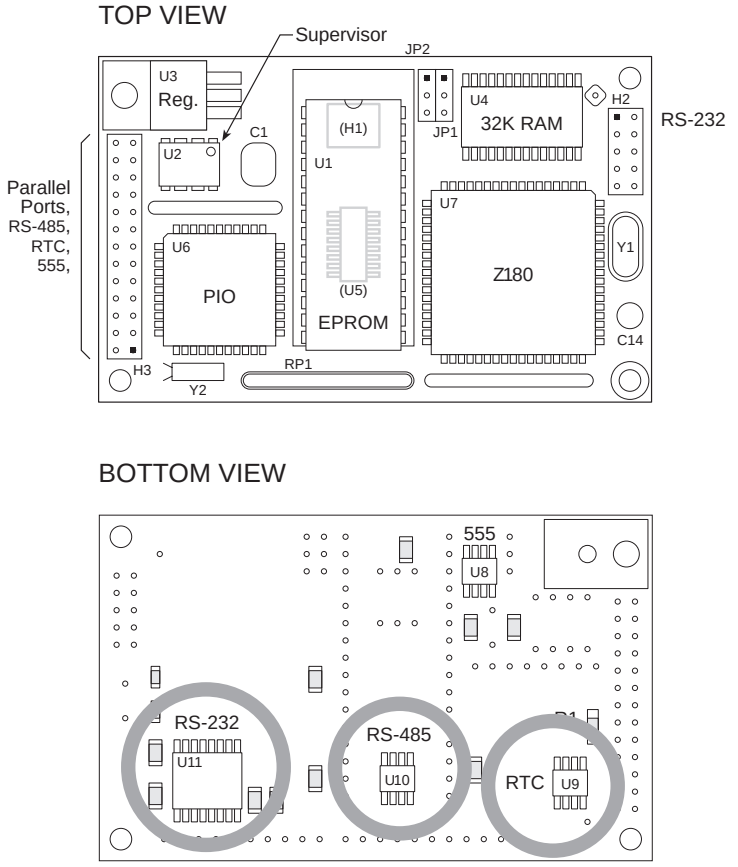


Figure 1-1. BL1400 Board Layout

BL1400 Features

- Z180 microprocessor running at 6.144 MHz.
- Zilog PIO with 12 configurable parallel I/O channels.
- 32K SRAM, surface-mounted.
- EPROM socket for up to 512K EPROM or 256K flash EPROM. The BL1400 comes with a 32K EPROM installed; 128K or 256K factory-installed flash EPROM options are available.
- DS1232 supervisor with watchdog timer and power-failure reset.
- Linear regulator (5 V).
- RS-232 serial channel.
- RS-485 serial channel.
- 555 timer, configured as an analog input channel.
- DS1302 real-time clock (RTC) with 31-byte scratchpad RAM.
- Optional base plate for mounting and additional heat sinking.

Table 1-1 lists the versions of the BL1400 Series that are available.

Table 1-1. BL1400 Series Features

Model	Features
BL1400	Standard full-featured model.
BL1410	BL1400 without RS-485 channel or real-time clock. Has two additional PIO channels
BL1470	EasyStart™ version of BL1410 with 128K flash EPROM and base plate.

See the ***BL1470 User's Manual*** for further information on the BL1470 and EasyStart™.

Appendix B, "Specifications," provides a complete list of the BL1400's specifications.

Developer's Kit

The BL1400 developer's kit contains the following items.

- Manual with schematics
- Development board
- Prototyping board for BL1400 expansion
- Aluminum base plate/heat sink
- 128K flash EPROM
- AC power supply
- Programming cable

Networking

The BL1400's RS-485 capability allows system developers to build a network of controllers with links up to several kilometers apart.

The BL1400 was designed to allow users to build and add their own piggyback expansion boards. The developer's kit contains a prototyping board for this purpose.

Refer to Appendix D, "The Prototyping Board," for more information.

A BL1400 system may be extended with Z-World's standard 4-bit PLCBus expansion boards. Software for these purposes, which uses the PIO ports, is included in BL14_15.LIB and is described in Chapter 5, "Software Reference."

See Appendix E, "PLCBus," and Appendix F, "Simulated PLCBus Connection," for more information on using Z-World's expansion boards with the BL1400.

Software Development and Evaluation Tools

Dynamic C, Z-World's Windows-based real-time C language development system, is used to develop software for the BL1400. Dynamic C is an integrated editor-compiler-debugger that runs under Windows on IBM-compatible PCs.

As a program compiles, it is downloaded directly to the BL1400, which is connected to one of the COM ports on the host PC. Serial communication is normally at 19,200 bps, but may be as high as 57,600 bps. The controller normally remains connected to the PC while a program is undergoing development. Once program development is complete, the completed program may be compiled for EPROM. An EPROM can be burned in a separate process and installed in the EPROM socket on the BL1400.

On power-up, the Dynamic C BIOS (in the BL1400's EPROM) checks for the presence of the development board first, then for the presence of flash EPROM. If neither of these is present, Dynamic C reverts to the EPROM. If Dynamic C detects a program in the EPROM, it executes that program.

Z-World's Dynamic C reference manuals provide complete software descriptions and programming instructions.

For ordering information, or for more details about the various options and prices, call your Z-World Sales Representative at (530) 757-3737.

GETTING STARTED

Chapter 2 provides instructions for connecting the BL1400 to a host PC and running a sample program.

Initial BL1400 Setup

The developer's kit contains the items necessary for BL1400 software and hardware development. The kit includes the following items.

- Development Board for programming the BL1400, with development RAM up to 512K
- Prototyping Board for prototyping BL1400 expansion circuits and powering the board during development
- Aluminum base plate/heat sink that adds support and allows the BL1400 to dissipate up to 3.5 W at an ambient temperatures of 50°C
- Manual with schematics
- 128K flash EPROM
- AC adapter
- Serial programming cable
- Miscellaneous small hardware such as 26-pin cable connectors, screws, and standoffs.

Connecting the BL1400 to a Host PC

The BL1400 can be developed in three ways: using the Development Board, using flash EPROM, or using regular EPROM. Each method is described below.

On power-up, the Dynamic C BIOS (in the BL1400's EPROM) checks for the presence of the Development Board first, then for the presence of flash EPROM. If neither is present, Dynamic C searches for a program in the EPROM. If Dynamic C detects a program in the EPROM, it executes that program.

The BL1400 operates either in a program mode or in run mode. Select the run/program mode as described below either through jumpers JP3 and JP4 on the Development Board or through header H2 on the Protoyping Board, depending on which development method is being used.

Via Development Board

The Development Board plugs into the EPROM socket of the BL1400. The Development Board has its own RS-232 port, and communicates with the host PC. It emulates the EPROM normally installed on the BL1400, providing up to 504K of program space in addition to the 32K RAM on the BL1400 (used as data space).

1. Remove the EPROM installed in socket U1 on the BL1400 board.
2. Jumper pins 2–3 on BL1400 headers JP1 and JP2.
3. Make sure that the Dynamic C development EPROM supplied with the Development Board is installed at U3 on the development board, as shown in Figure 2-1.

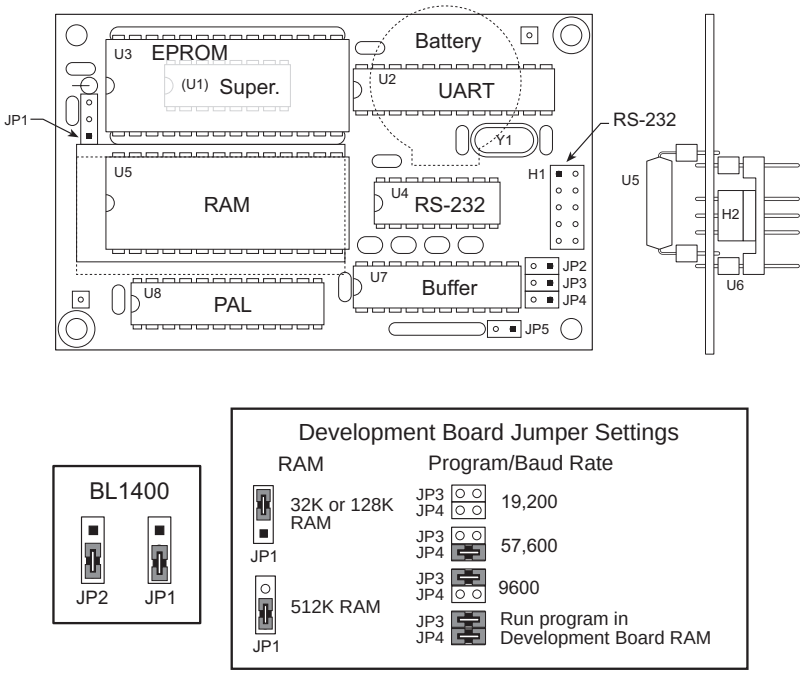


Figure 2-1. Developing Applications with BL1400 Development Board

4. Set the jumpers on the Development Board as shown in Figure 2-1 for RAM size and run/program mode. Figure 2-2 shows how to seat a 28-pin RAM chip (32K or 128K) on the Development Board.

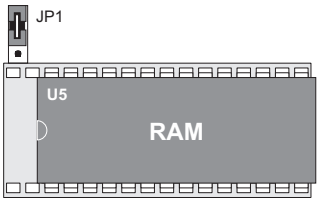


Figure 2-2. Placing 28-pin Development Board RAM in 32-pin Socket

5. Plug the Development Board into BL1400 socket U1 and H1, as shown in Figure 2-3. U6 and H2 on the under side of the Development Board match the BL1400 U1 and H1 sockets exactly.

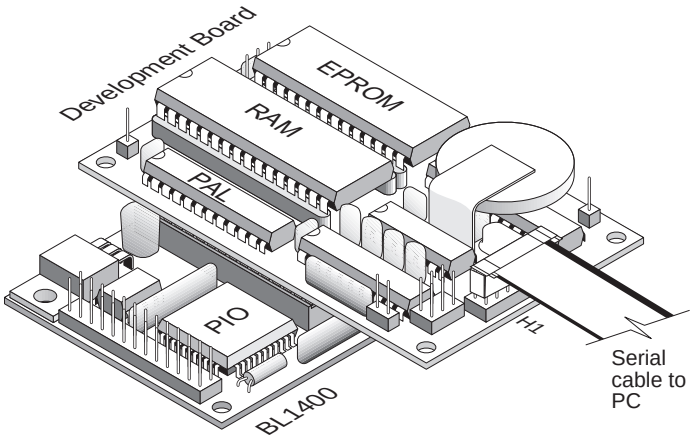


Figure 2-3. Attaching Development Board to BL1400

6. Connect the serial cable as shown in Figure 2-4. Connect the 10-pin end to header H1 on the Development Board, taking care to match the arrow on the 10-pin connector to pin 1 on H1. Connect the end with the DB9 connector to the COM port of your PC.
6. Connect the Prototyping Board to the BL1400 by connecting header H1 on the underside of the Prototyping Board to H3 on the BL1400 as shown in Figure 2-4. Make sure the Prototyping Board is oriented as shown. Most of the Prototyping Board extends beyond the BL1400.
7. Connect the DC plug from the AC adapter to the DC input jack of the Prototyping Board. Plug in the AC adapter. The BL1400 is now ready for programming.

Dynamic C is used to create the application, which is then compiled by selecting **Compile to File** in the **Compile** menu. Essential library routines are uploaded from the Dynamic C development EPROM on the Development Board and are linked to the application being developed. The Dynamic C output is a binary file (or an Intel hex format file) that can be used to burn an EPROM for use in the BL1400.

After the Development Board is removed, the BL1400 will "remember" the last setting established by the Development Board. Be sure to power the BL1400 up in run mode (with JP3 and JP4 on the Development Board connected as shown in Figure 2-1) before disconnecting the development board. Place the newly burned EPROM in socket U1.

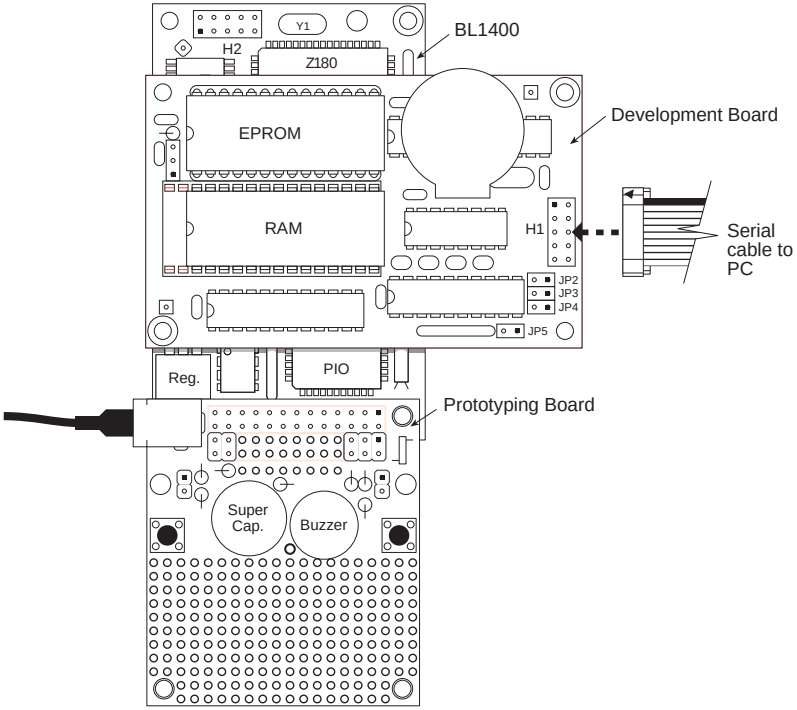


Figure 2-4. Attaching Prototyping Board to BL1400

Via Flash EPROM

Using the flash EPROM provides up to 256K programming space (in flash EPROM) and 32K in RAM (on the BL1400). The RS-232 port on the BL1400 is used for programming.

1. If necessary, plug a flash EPROM into socket U1 on the BL1400 board.
2. Jumper pins 2–3 on BL1400 header JP1 and pins 1–2 on BL1400 header JP2, which indicates flash EPROM.
3. Connect the Prototyping Board to the BL1400 by connecting header H1 on the underside of the prototyping board to H3 on the BL1400 as shown in Figure 2-4. Make sure the Prototyping Board is oriented as shown. The Development Board is not used with this development method. Most of the Prototyping Board extends beyond the BL1400.

4. Connect the DC plug from the AC adapter to the DC input jack of the Prototyping Board. Plug in the AC adapter.
5. Configure the jumpers on the BL1400 and on the Protoyping Board as shown in Figure 2-5.

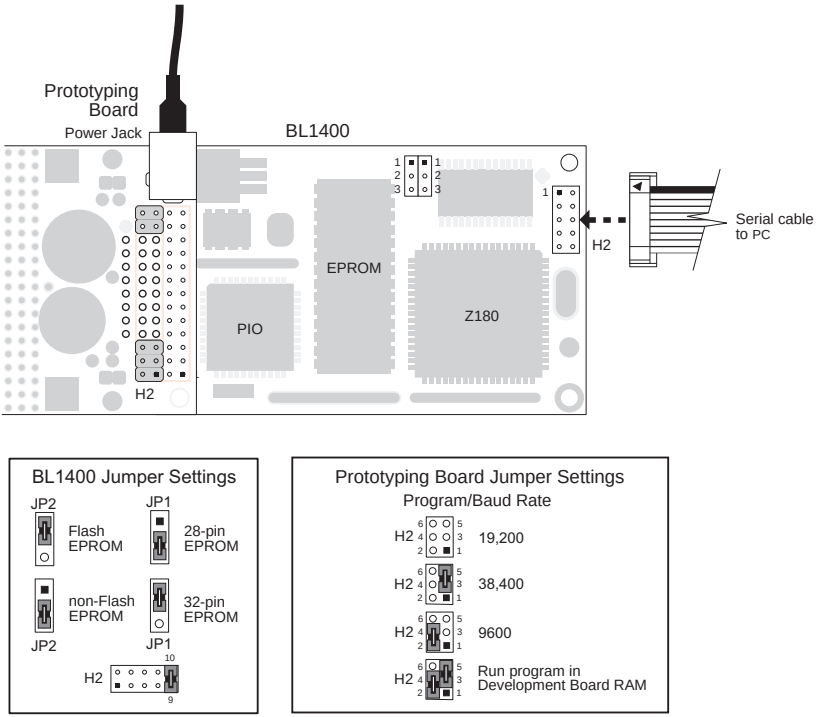


Figure 2-5. Developing Applications with BL1400 Prototyping Board

6. Do a power reset by disconnecting the AC adapter and reconnecting it.
7. Remove the jumpers from H2 on the BL1400 and from H2 on the Prototyping Board.
8. Connect the serial cable as shown in Figure 2-5. Connect the 10-pin end to header H2 on the BL1400, taking care to match the arrow on the 10-pin connector to pin 1 on H2. Connect the end with the DB9 connector to the COM port of your PC.
9. The BL1400 is now ready for programming.

The BL1400 will "remember" the operating mode specified in Step 5 until Steps 5–7 are repeated.

Via Regular EPROM

Using regular EPROM provides up to 32K code and data space, in RAM, on the BL1400. The RS-232 port on the BL1400 is used for programming.

1. Check the BL1400 jumpers. Connect pins 1–2 on header JP1 for a 28-pin EPROM at U1, or connect pins 2–3 on JP1 for a 32-pin EPROM. Connect pins 2–3 on header JP2 for non-flash EPROM. These jumper configurations are summarized in Figure 2-6.
2. Plug the 28-pin or 32-pin EPROM into the socket at U1 on the BL1400. Figure 2-6 illustrates how to seat a 28-pin EPROM.

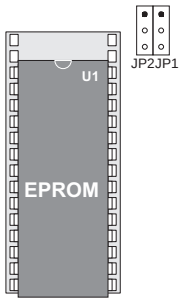


Figure 2-6. Placing 28-pin EPROM in 32-pin Socket

3. Connect the Prototyping Board to the BL1400 by connecting header H1 on the underside of the Prototyping Board to H3 on the BL1400 as shown in Figure 2-5. Make sure the Prototyping Board is oriented as shown. Most of the Prototyping Board extends beyond the BL1400.
4. Connect the DC plug from the AC adapter to the DC input jack of the prototyping board. Plug in the AC adapter.
5. Configure the jumpers on the BL1400 and on the Protoyping Board as shown in Figure 2-5.
6. Do a power reset by disconnecting the AC adapter and reconnecting it.
7. Remove the jumpers from H2 on the BL1400 and from H2 on the Prototyping Board.
8. Connect the serial cable as shown in Figure 2-5. Connect the 10-pin end to header H2 on the BL1400, taking care to match the arrow on the 10-pin connector to pin 1 on H2. Connect the end with the DB9 connector to the COM port of your PC.
9. The BL1400 is now ready for programming.

The BL1400 will “remember” the operating mode specified in Step 5 until Steps 5–7 are repeated.

Running Dynamic C

Test the Communication Line

Double-click the Dynamic C icon to start the software. Note that the PC attempts to communicate with the BL1400 each time Dynamic C is started. No error messages are displayed once communication is established.

See Appendix A, Troubleshooting, if an error message such as **Target Not Responding** or **Communication Error** appears.

Once the necessary changes have been made to establish communication between the host PC and the BL1400, use the Dynamic C shortcut **Ctrl Y** to reset the controller and initiate communication.

Selecting Communications Rate, Port, and Protocol

The communication rate, port, and protocol are all selected by choosing **Serial Options** from Dynamic C's **OPTIONS** menu.

The BL1400's factory-default communication rate is 19,200 bps. However, the Dynamic C software may be initialized for a different rate. To begin with, use the communications rate of 19,200 bps.

Make sure that the PC serial port used to connect the serial cable (COM1 or COM2) is the one selected in the Dynamic C **OPTIONS** menu. Select the 1-stop-bit protocol.

Running a Sample Program

1. Open the sample program **MGDEMORT.C** located in the Dynamic C **SAMPLES\BL14_15** directory.
2. Compile the program by pressing **F3** or by choosing **Compile** from the **Compile** menu. Dynamic C compiles and downloads the program into the BL1400's memory. During compilation, Dynamic C rapidly displays several messages in the **STDIO** window. This condition is normal.
3. Run the program by pressing **F9** or by choosing **Run** from the **Run** menu.
4. Press **Ctrl Z** to stop execution of the program.
5. If needed, press **F9** to restart execution of the program.

Power Options

During software development, the BL1400 draws its power from the Prototyping Board, which is connected to the 9 V DC output of the AC adapter.

After development is complete, connect the 9 V to 12 V power leads from the power source to pins 25 (DCIN) and 26 (GND) of header H3 of the BL1400. A 26-pin cable connector in the developer's kit may be used to build a cable to connect the power source (and other signals).

Customer piggyback expansion boards are normally normally connected directly to header H3 on the BL1400. The expansion boards must have their own power connection.

BL1400 SUBSYSTEMS

Sections in Chapter 3 discuss the following topics.

- Operating Modes
- Interface Overview
- Parallel Input/Output Chip
- H2 Signals
- H3 Signals
- Serial Communication
- Regulator
- Supervisor
- Real-Time Clock
- 555 Precision Timer
- Memory

Operating Modes

The BL1400 operates either in a program mode or in run mode. Select the run/program mode either through jumpers JP3 and JP4 on the Development Board, or through header H2 on the Protoyping Board and header H2 on the BL1400, depending on which development method is being used. The BL1400 will remember the latest settings after the Development Board or the Prototyping Boards have been removed.

When in run mode, the BL1400 will look for and then execute the program in EPROM upon power-up.

Changing Baud Rate on the BL1400

The baud rate may be changed by connecting the appropriate pins on the Development Board or the Prototyping Board, depending on which development method is being used. The baud rate may be changed after the application has been developed. The BL1400 will remember the latest settings after the Development Board or the Prototyping Board has been removed.

Power Connections

After development is complete, connect the 9 V to 12 V power leads from the power source to pins 25 (DCIN) and 26 (GND) of header H3 of the BL1400. A 26-pin cable connector in the developer's kit may be used to build a cable to connect the power source (and other signals).

See Chapter 2, "Getting Started," for complete details on configuring the BL1400 for programming or standalone operation.

Interface Description

This section describes the various interfaces of the BL1400. The BL1400 has byte- and bit-wise parallel inputs/outputs (I/O), an RS-232 serial port, an RS-485 port, a real-time clock (RTC), and a 555 timer that measures resistance. Figure 3-1 shows a block diagram of the BL1400.

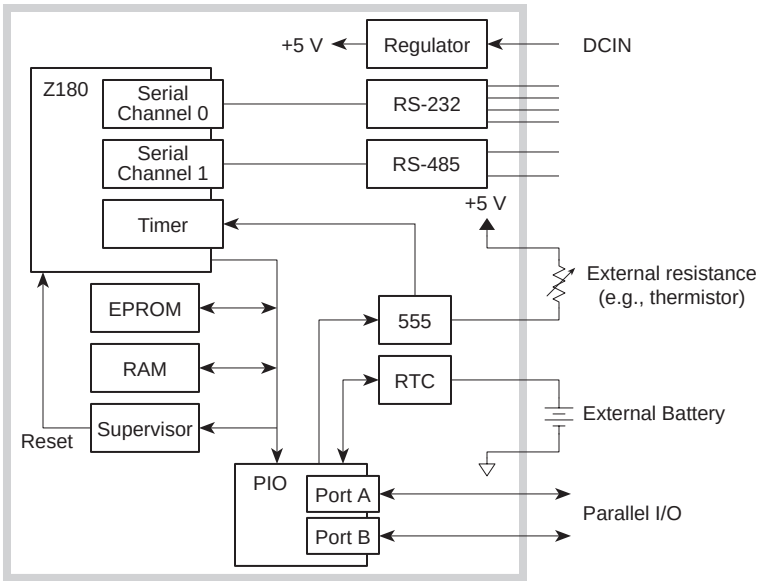


Figure 3-1. BL1400 Block Diagram

The I/O interface consists of headers H2 and H3, as shown in Figure 3-2.

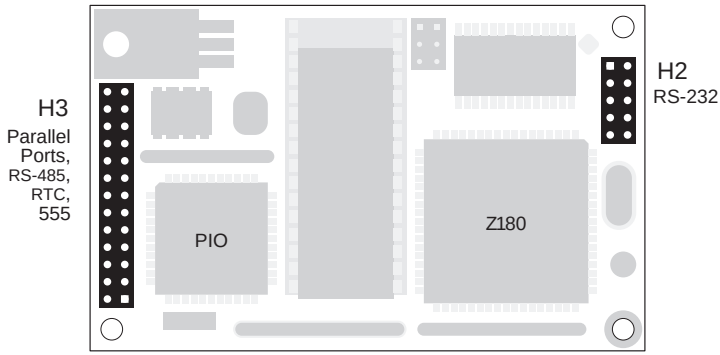


Figure 3-2. BL1400 I/O Headers

The EPROM socket (U1) and the 6-pin header H1 are used to interface to the development board. Socket U1 connects with U6 of the development board, and H1 connects with H2 of the development board.

Parallel Input/Output (PIO) Chip

The BL1400 has a Zilog PIO (Parallel Input/Output) chip. The Zilog PIO chip (U6) is a 44-pin chip that provides two 8-bit parallel I/O ports, as shown in Figure 3-3.

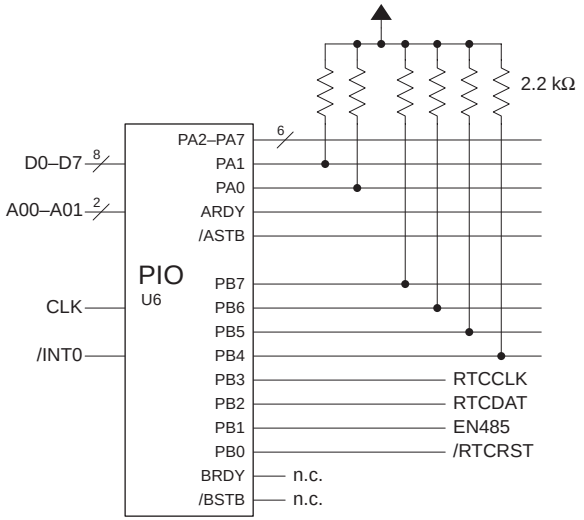


Figure 3-3. PIO Pin Map

Lines PA0–PA7 are considered to be "Port A" and lines PB0–PB7 are considered to be "Port B." Each line can serve either as an input or as an output in different modes. The lines may also be used to detect transitions and cause an interrupt in the microprocessor in various ways. There are two handshaking lines, a ready line and a strobe line, for each port, but handshaking lines BRDY and /BSTB are not used.

Lines PA0–PA7 have TTL-compatible logic levels. Lines PB0–PB7 can supply up to 1.5 mA at 1.5 V to drive Darlington transistors. Depending on the application, the user may need to add current-limiting resistors, noise filters, or transient voltage suppression circuitry for either port.

The impedance of the PIO is approximately 80 Ω for sinking current and 160 Ω for sourcing current. Voltages below ground or above VCC should not be applied to the PIO.

The PIO is very flexible and has a number of modes of operation. The two ports are controlled by the four registers shown in Table 3-1.

Table 3-1. PIO Registers

Register	Port
00C0H (PIODA)	PIO Port A (data)
00C1H (PIODB)	PIO Port B (data)
00C2H (PIOCA)	PIO Port A (command)
00C3H (PIOCB)	PIO Port B, (command)

BL1400 address logic uses only address bits A15, A7, and A6 to decode these addresses. The PIO decodes the lower two bits of the address itself. Thus, many phantom decodes, which are aliases of these addresses, recur throughout the I/O address space. Note that the development board also has two I/O addresses reserved: F000 for the UART (serial I/O controller), and E000 for the memory map and configuration jumpers. Address bits A15 through A12 are used to decode these addresses.

Each register pair controls one of the 8-bit ports and the two handshaking lines associated with each port. There are four modes of operation for PIO Port A and PIO Port B, as shown in Table 3-2.

Table 3-2. PIO Operation Modes

Mode	Result
Mode 0	Strobed byte output
Mode 1	Strobed byte input
Mode 2	Bidirectional data transfer
Mode 3	Bitwise I/O, input/output selectable per bit

On the BL1400, Port A may be programmed to operate in Modes 0, 1, and 3, but not Mode 2. Port B is available in Mode 3 only. Port B's handshaking lines are not connected.

Four of port B's lines are preassigned, as shown in Table 3-3 (see also Figure 3-3).

Table 3-3. Preasssigned PIO Lines

PIO Port	Pin Signal	Pin Function
PB3	RTCCLK	DS1302 RTC serial clock
PB2	RTCDAT	DS1302 RTC serial data
PB1	EN485	RS485 transmit enable
PB0	/RTCRST	555 trigger and DS1302 RTC reset

PB3 and PB2 can be used as two additional parallel I/O if the real-time clock is not being used.

The PIO uses /INT0 of the Z180. The PIO can be programmed to interrupt the Z180 upon the transfer of each byte of data during handshake mode, or upon the change of state of individual bits during bit I/O mode.

Refer to Zilog's ***Z80180/Z180 MPU User's Manual*** for complete details.

PIO Operation Modes

Mode 0 (Strobed Byte Output)

When the microprocessor stores a byte in a port's data register, the eight associated output lines change their level according to how each bit is set: to high for a 1 and low for a 0. The ready handshake line goes high. When an external device pulses the strobe line (low), the ready line is reset. If interrupts are enabled for the port, a PIO interrupt is requested. This allows for interrupt-driven parallel output.

Mode 1 (Strobed Byte Input)

The PIO latches eight bits into a register upon the strobe signal from an external device. The strobe signal also causes the ready line to go low. An interrupt is then requested. After the microprocessor reads the register, the ready line is raised to indicate that the port is ready for another byte.

Mode 2 (Bidirectional Data Transfer)

Mode 2 is not available on the BL1400.

Mode 3 (Bitwise I/O)

This is a general-purpose I/O mode. Each bit can be individually specified as input or output. In this mode, the input lines can also serve as interrupt request lines. Either transition to high or transition to low can be specified for the interrupt request. Interrupts for specific input lines are controlled with a mask and by specifying an AND or an OR function for the masked lines. Interrupts on PIO ports are edge-triggered.

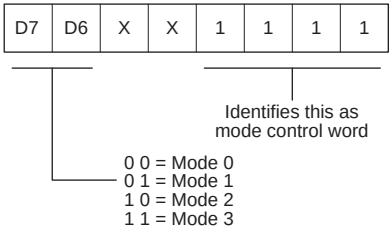
Control Register Byte Sequence

To set up a port for I/O, first write a sequence of bytes to its control register. Then read, or write, its data register to transfer data.

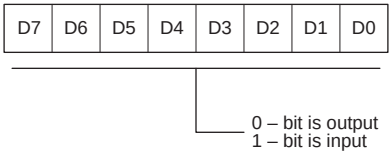
The control register byte sequence is shown below.

mode control word
I/O register control word (only if mode 3)
interrupt vector word
interrupt control word
mask control word
interrupt disable word

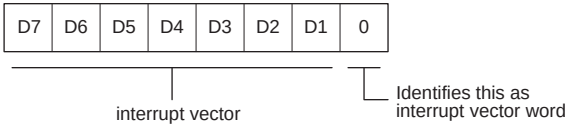
The ***mode control word*** specifies the mode for the port.



The ***I/O register control word*** must immediately follow the mode control word, but only in Mode 3 (bitwise I/O). This byte specifies which bits are inputs and which bits are outputs for bitwise I/O.



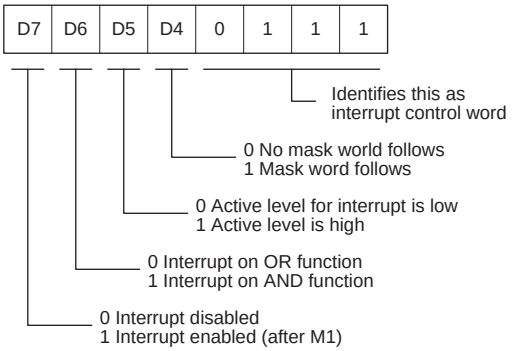
The ***interrupt vector word*** specifies the interrupt vector for the particular PIO channel.



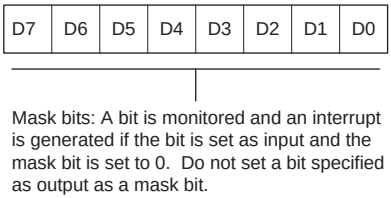
The vectors for the PIO ports are as follows.

0x12	(**PIOA_VEC**)	PIO Port A
0x14	(**PIOB_VEC**)	PIO Port B

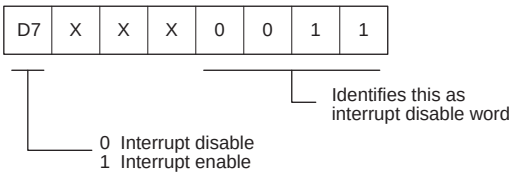
The ***interrupt control word*** specifies the conditions under which an interrupt is generated.



The ***mask control word*** must immediately follow the interrupt control word if bit D4 of the interrupt control word is set.



The ***interrupt disable word*** is used to enable and disable an interrupt for a port that is already defined by an interrupt control word. This byte can also be used to disable interrupts on an unconfigured port.



Header H2 Signals (RS-232 Port)

Header H2 provides one 5-wire RS-232 interface. Header H2 is also the programming port if the development board is not used for programming. When header H2 is used as the programming port, it cannot be used as a communication port by the application.

Figure 3-4 illustrates the signals on header H2.

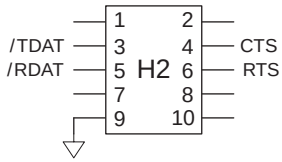


Figure 3-4. Header H2 Signals

Refer to Chapter 4, “System Development,” for further details.

Header H3 Signals (PIO Ports)

Header H3 carries all I/O signals, except for the RS-232 signals, as shown in Figure 3-5.

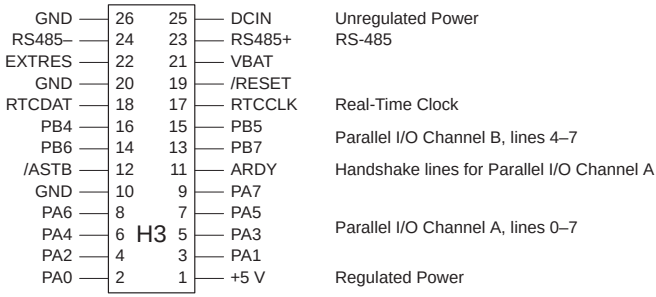


Figure 3-5. Header H3 Signals

Header H3 is also the expansion “bus,” intended for engineers who wish to develop their own “piggyback” expansion boards for the BL1400. These signals are described later in this chapter.

Serial Communication

Two Z180 serial ports support asynchronous communication at baud rates from 300 bps to 38,400 bits per second. Z180 Port 0 is a 5-wire RS-232 port (with RTS and CTS). Port 1 is a half-duplex RS-485 port, providing half-duplex asynchronous communication over twisted-pair wires.

Figure 3-6 illustrates the configuration of Z180 Port 0 and Z180 Port 1.

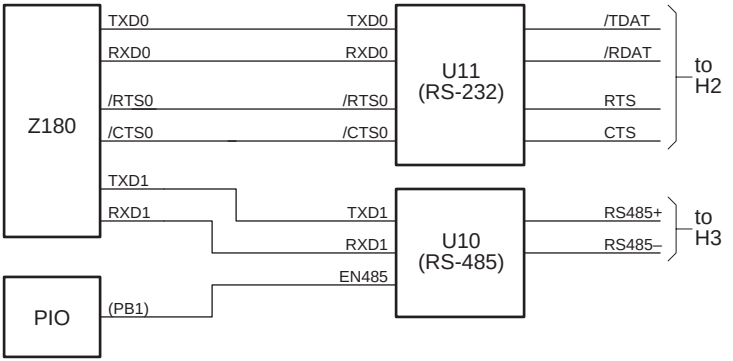


Figure 3-6. Z180 Serial Channel Configuration

RS-232 Communication

Z-World has AASC RS-232 support libraries for Z180 Port 0. The support for serial communication includes the following functions.

- Initialization of the serial ports
- Monitoring and reading a circular receive buffer
- Monitoring and writing to a circular transmit buffer
- An echo option
- CTS (clear to send) and RTS (request to send) control
- XMODEM protocol for downloading and uploading data

Receive and Transmit Buffers

Serial communication is made easier with a background interrupt routine that updates the receive and transmit buffers. Every time a port receives another character, the interrupt routine places it into the receive buffer. A program can read the data one character at a time or as a stream of characters terminated by a special character.

A program sends data by writing characters into the transmit buffer. If the serial port is not already transmitting, the write functions will automatically initiate transmission. Once the last character of the buffer is sent, the transmit interrupt is turned off. Data can be written one character at a time or as a stream of characters.

Echo Option

If the echo option is turned on with **Dinit_z0** during initialization of the serial port, any character received is automatically echoed back (transmitted out). This feature is ideal for use with a dumb terminal and also for checking the characters received.

CTS/RTS Control

Z180 Port 0 is constrained by hardware to have the CTS (clear to send) pulled low by the RS-232 device to which it is talking.

If the CTS/RTS option is chosen, the support software will pull the RTS (request to send) line high when the receive buffer has reached 80% of its capacity. Thus, the transmitting device (if its CTS is enabled) will stop transmitting. The RTS line is pulled low again when the receive buffer has gone below 20% of its capacity.

If the device with which the BL1400 is communicating does not support CTS and RTS, the CTS and RTS lines on the BL1400 side must be tied together to make communication possible.

The CTS line must be pulled up when not in use.

XMODEM File Transfer

The BL1400 supports the XMODEM protocol for downloading and uploading data. Currently, the library supports downloading an array of data whose size is a multiple of 128 bytes.

Uploaded data are written to a specified area in RAM. The targeted area for writing should not conflict with the current resident program or data.

Character echo is automatically suspended during XMODEM transfer.

Modem Communication

Modems and telephone lines enable RS-232 communication across great distances. If the modem option is chosen, character streams that are read from the receive buffer are automatically scanned for modem commands. When a modem command is found, the software takes appropriate action. Normally, the communication package would be in COMMAND mode while waiting for valid modem commands or messages. Once a link is established, the communication is in DATA mode (regular RS-232). However, the software continues to monitor the modem for a **NO_CARRIER** message.

RS-485 Communication

Header H3 on the BL1400 provides a half-duplex RS-485 interface, as shown in Figure 3-7. An RS-485 serial communication channel can be used to create a network of BL1400 (or other) controllers with links spanning several kilometers.

The RS-485 signals are on pins 23 and 24 of H3, and also on pins 23 and 24 of H2 on the prototyping board.

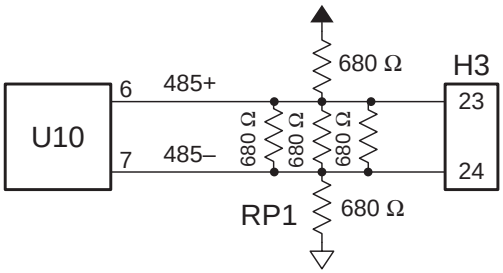


Figure 3-7. RS-485 Signal

Developing an RS-485 Network

Z-World has Dynamic C library functions in **NETWORK.LIB** for master-slave two-wire half-duplex RS-485 ninth-bit binary communication. This protocol is supported only on the Z180 Port 1, which can be configured for RS-485 communication on the BL1400 (and on many of Z-World's other controllers).

Functional support for master-slave serial communication follows this scheme.

1. Initialize Z180 Port 1 for RS-485 communication.
2. The master sends an inquiry and waits for a response from a slave.
3. Slaves monitor for their address during the transmission of the ninth bit. The targeted slave replies to the master.

The binary command message protocol adopted is similar to that used for the new opto 22 binary protocol. A master message is composed as follows.

```
[slave id] [len] [ ] [ ]…[ ] [CRC hi][CRC lo]
```

The slave's response is composed as follows.

```
[len] [ ] [ ]...[ ] [CRC hi] [CRC lo]
```

The term **len** is the length of the message that follows.

Refer to the Dynamic C manuals for more details on master-slave networking.

Hardware Connection

Figure 3-8 shows the connections for a two-wire RS-485 network. On all networked controllers, connect RS-485+ to RS-485+ and RS-485- to RS-485- using single twisted pair wires (nonstranded, tinned).

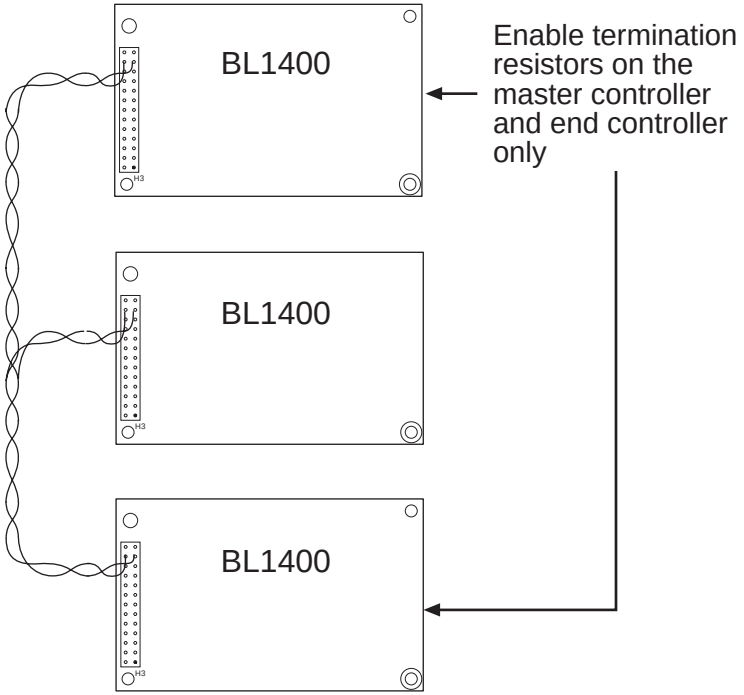


Figure 3-8. BL1400 Multidrop RS-485 Network

Any one of Z-World's controllers can be a master or a slave. While there can only be one master, there can be up to 255 slaves. The master should have a board identification address of 0. Slaves should each have their own distinct identification address from 1 to 255.

Termination and bias resistors are required in a multidrop network to minimize reflections (echoing), and to keep the network line active in an idle state. Typically, 120 Ω termination resistors are installed across the master node and across the physical end node of an RS-485 network.

Contact Z-World Technical Support at (530) 757-3737 for assistance with large-scale network design.

Regulator

An on-board +5 V linear regulator (U3) accepts 9 V to 12 V DC. This is DCIN. The regulator has some excess capacity to power expansion boards or external loads.

The maximum power dissipation is 1.0 W at 70°C without additional heat sinking. The BL1400 itself draws 120 mA, dissipating about 0.85 W with 12 V input. Additional heat sinking can be obtained by mounting the BL1400 on the optional aluminum base plate, using aluminum standoffs.

See Appendix C, “Power Management,” for more information.

Supervisor

The DS1232 supervisor (U2) provides a watchdog timer that guards against system or software faults by resetting the Z180 microprocessor if the software does not “hit” the timer at least every 1.2 seconds.by making a call to **`hitwd`**. The supervisor also provides the reset signal for the microprocessor and the rest of the system on power-up or when VCC (normally +5 V) falls below 4.62 V.

The negative-going reset signal (/RESET) (shown in Figure 3-8) is active for at least 250 ms after VCC reaches 4.62 V. /RESET is an open-drain signal, pulled up by a 2.2 kΩ resistor

An external reset switch can be added as shown in Figure 3-9. A small capacitor can also be added if debouncing is needed.

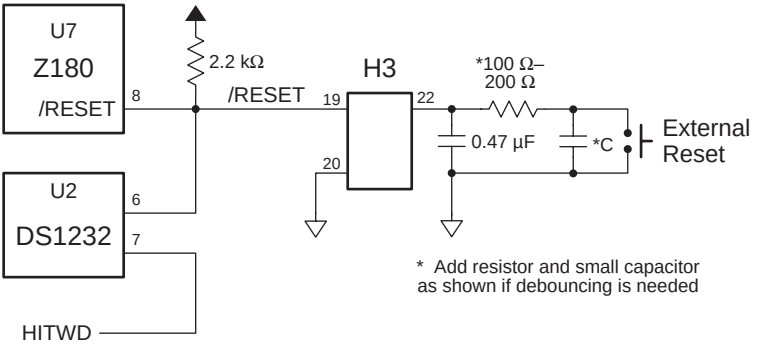


Figure 3-9. Reset Circuit

Real-Time Clock (RTC)

The DS1302 real-time clock (U9) on the BL1400 provides time and date functions, plus 31 bytes of scratchpad RAM.

The RTC automatically adjusts the last date of the month for the number of days in a month, and accounts for leap years. Time is provided in 24-hour or 12-hour format (with AM/PM indicator).

An external battery can be added to retain RTC data when power is disconnected, as shown in Figure 3-10.

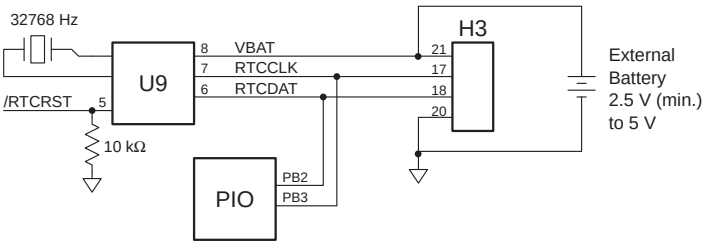


Figure 3-10. External RTC Battery

Data are clocked into and out of the RTC with signals RTCCLK (pin 17) and RTCDAT (pin 18). Information may be read from, or written to, the time/date registers or the scratchpad RAM of the RTC. The RTC is reset or idled when /RTCRST is low. /RTCRST must be raised at the start of a transfer to enable the DS1302. It must be lowered at the end of the transfer to terminate the transfer. Z-World software drivers take care of these details.

If the RTC is not present, the RTCCLK (PB3) and RTCDAT (PB2) lines may be used for other purposes.

/RTCRST has a second function. The falling edge of /RTCRST triggers the 555 timer. 555 activity has no effect on the RTC if the RTCCLK and RTCDAT lines are not active at the same time.

The DS1302 contains a trickle charge circuit to charge rechargeable batteries or super capacitors. The charge circuit, controlled by the trickle charge register, is disabled at power-up.

Do *not* enable the trickle charge circuit if you are using a nonrechargeable battery to back up the DS1302. Enabling the trickle-charge circuit could cause the battery to explode.

Refer to Appendix G, "Input/Output Maps and Interrupt Vectors, for more information on the DS1302 registers and their addresses.

555 Timer

A 555 timer (U8) is available to measure an external resistance, such as a thermistor, control potentiometer, or position sensor. Figure 3-11 shows the connections to measure the external resistance.

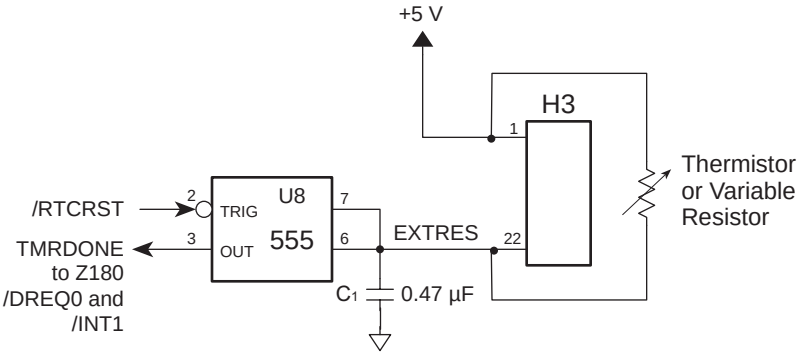


Figure 3-11. Use of 555 Timer Circuit to Measure External Resistance

When used as an astable osillator, the 555 measures resistance according to the formula

$$\Delta T = 1.1\ RC_1\ , \qquad (3\text{-}1)$$

where ΔT is the length of the 555 timeout cycle measured by the software, C_1 is the 555 timing capacitor, C_1 (0.47 µF), and R is the external resistor connected to +5 V.

R divided by ΔT gives ohms per unit time; for the 0.47 µF capacitor, this becomes 1.93 Ω/µs.

Optimum results are obtained when measuring resistances up to about 2 kΩ to 4 kΩ when C_1 is 0.47 µF. At these resistances, the values of the PRT count obtained are repeatable over the short term (temperature may affect long-term stability). For example, using the 555 time delay equation given above,

- at 2 kΩ the 555 timeout period is 1.034 ms, corresponding to a PRT time interval of 317 clock periods;
- at 4 kΩ the 555 timeout period would be 2.068 ms, corresponding to a PRT time interval of 634 clock periods.

Larger resistances can be measured, but accuracy (or more correctly, repeatability) is reduced. This reduction is cuased by noise effects combined with the flattening of the capacitor charging waveform, which is fed to the 555 voltage comparator. Additionally, conversion times are proportionately longer.

The initial accuracies and temperature stability of the primary components listed below contribute to the overall accuracy of 555 resistance measurements.

- The Z180 crystal. The initial accuracy of the crystal is 0.005%, or 50 parts per million.
- The 555 timer. The data sheet for the Phillips NE555D specifies a temperature stability of 0.005% per degree Celcius, or 50 parts per million, referenced to 25°C. The initial deviation of the NE555D is specified as 1% typical, 3% maximum. This represents a temperature stability of 0.225% over the operating temperature range of the BL1400 (0°C to 70°C).
- The onboard timing capacitor C_1. For C_1, the value is 10%. Timing capacitor C_1 has an X7R temperature coefficient that results in a temperature stability of approximately 2% to 3%, varying somewhat with the manufacturer, over the operating temperature range of the BL1400.
- The external resistor. The user must determine the initial accuracy of the external resistor.

When measuring temperature with a thermistor, the temperature coefficient of the thermistor, discussed in the next section, is the primary consideration. When measuring an external variable resistance, be aware of its temperature coefficient and the effect the temperature coefficient has on the overall accuracy of the measurement.

For maximum accuracy, connect the external resistance to the +5 V supply of the BL1400, as shown in Figure 3-10. Do not return the external resistor to another power supply in the system.

Phillips and other manufacturers have published several useful application notes that describe in detail how to use the 555 effectively. These notes include important precautions as well as suggestions for ensuring reliable results.

See Chapter 5, “Software Reference,” for functions that read the 555.

Using Thermistors

Negative temperature coefficient (NTC) thermistors are frequently used as inexpensive and reasonably accurate temperature sensors. Units with initial accuracies of 1% are available.

This section presents a brief summary of the use of thermistors as temperature sensors.

Refer to a specific manufacturer's literature for additional detail.

The resistance of an NTC thermistor varies inversely with temperature according to Equation (3-2).

$$R = Ae^{ß/T} \quad \text{(T in kelvins)} \tag{3-2}$$

Two parameters are typically specified for a thermistor: its resistance at 25°C and its temperature coefficient α (alpha), expressed as the percent change of resistance per degree Celcius at 25°C. ß (beta) in the temperature equation is computed from α, or is sometimes given by the manufacturer:

$$ß = -T^2\alpha \quad (T = 25°C, \text{ or } 298.15 \text{ K}) \cdot$$

α (alpha) is defined as $1/R \times dR/dT$, and equals $-\beta/T^2$ from the resistance equation.

Once β is known, A can be computed from the resistance at 25°C.

For example: assume a thermistor with $\alpha = -4.40\%$ per degree Celcius and R, at 25°C, of 10 kΩ. Then,

$$ß = -(298.15 \text{ K} \times 298.15 \text{ K}) \times (-0.044 \text{ K}^{-1})$$
$$= 3911 \text{ K} \ .$$

Since the resistance at 25°C is 10 kΩ,

$$10{,}000\ \Omega = Ae^{(3911 \text{ K}/298.15 \text{ K})}$$
$$= A \times 497{,}600 \ .$$

Therefore,

$$A = 10{,}000\ \Omega/497{,}600 = 0.0200\ \Omega \ , \text{ and}$$
$$\ln A = -3.907 \ .$$

Using the resistance equation and expressing T as a function of R,

$$T \text{ (in Kelvins)} = \frac{\beta}{(\ln R - \ln A)}$$
$$= \frac{3911}{[\ln R - (-3.907)]} \ .$$

Memory

The BL1400 has 32K of RAM and can have from 32K to 512K of EPROM. EPROM occupies the lower half of the Z180 address space. RAM addresses start at 80 000_{HEX}, as shown in Figure 3-12.

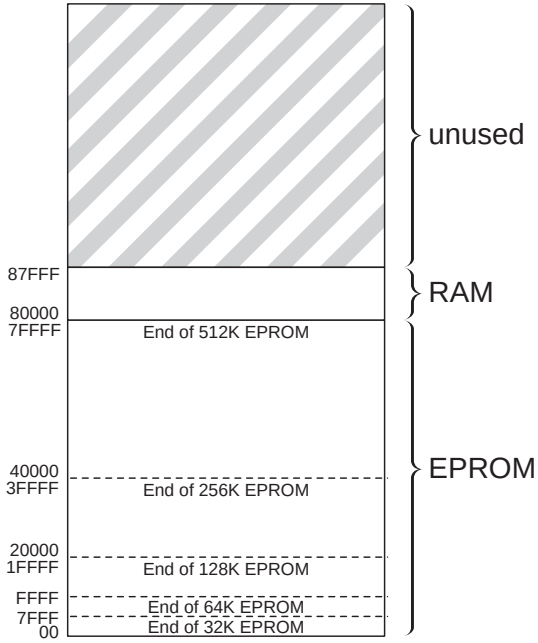


Figure 3-12. BL1400 Memory Addresses

Direct Programming of the Serial Ports

If you are planning extensive use of the serial ports, or intend to use synchronous communication, Z-World recommends that you obtain copies of the Zilog technical manuals. You will need the ***Z180 MPU User's Manual*** and the ***Z180 Microprocessor Family User's Manual*** (which describes the CPU and CTC, DMA, PIO and SIO functions). Z-World provides just a few low-level utility functions.

- **`int sysclock()`**

  Returns the clock frequency in multiples of 1200 bps, as read from RAM. The fixed clock frequency is stored at location 108H at the factory.

- **`int z180baud( int clock, int baud )`**

  Returns the byte to be stored in CNTLB0 or CNTLB1, considering only the bits needed to set the baud rate. The clock and baud rate must be supplied in multiples of 1200 Hz. Thus, a 6.144 MHz clock is expressed by 5120 and 19,200 bps is expressed by 16. The return value is –1 if the value for the baud rate cannot be derived from the given clock frequency.

Each serial port appears to the CPU as a set of registers. Each port can be directly accessed with the **`inport`** and **`outport`** library functions, using the symbolic constants shown in Table 3-4.

Table 3-4. Z180 Serial Port Registers

Address	Name	Description
00	CNTLA0	Control Register A, Serial Channel 0
01	CNTLA1	Control Register A, Serial Channel 1
02	CNTLB0	Control Register B, Serial Channel 0
03	CNTLB1	Control Register B, Serial Channel 1
04	STAT0	Status Register, Serial Channel 0
05	STAT1	Status Register, Serial Channel 1
06	TDR0	Transmit Data Register, Serial Channel 0
07	TDR1	Transmit Data Register, Serial Channel 1
08	RDR0	Receive Data Register, Serial Channel 0
09	RDR1	Receive Data Register, Serial Channel 1

For example, use the following code to read and write from Serial Port 0.

```
char ch;
ch = inport( RDR0 );
outport( TDR0, ch );
```

Ports may be either polled or interrupt-driven. The interrupt vectors are shown in Table 3-5.

Table 3-5. Serial Port Interrupt Vectors

Address	Name	Description
0E	**SER0_VEC**	Z180 Serial Port 0 (higher priority)
10	**SER1_VEC**	Z180 Serial Port 1

Attainable Baud Rates

The serial ports built into the Z180 generate the standard baud rates of 150 bps to 38,400 bps (see Table 3-7) for the 6.144 MHz clock in the BL1400. (The crystal is 12.288 MHz.)

Z180 Serial Ports

The Z180 has two independent, full-duplex asynchronous serial channels, with a separate baud-rate generator for each channel. The baud rate can be divided down from the microprocessor clock or from an external clock.

The serial ports have a multiprocessor communications feature. When enabled, this feature includes an extra bit in the transmitted character (where the parity bit would normally go). Receiving Z180s can be programmed to ignore all received characters except those with the extra multiprocessing bits enabled. This provides a 1-byte attention message that can be used to wake up a processor without the processor having to intelligently monitor all traffic on a shared communication link.

The block diagram in Figure 3-13 shows Serial Channel 0. Serial Channel 1 is similar, but modem control lines for /RTS and /DCD do not exist. The five unshaded registers shown in Figure 3-14 are directly accessible as internal registers.

The serial ports can either be polled or interrupt-driven.

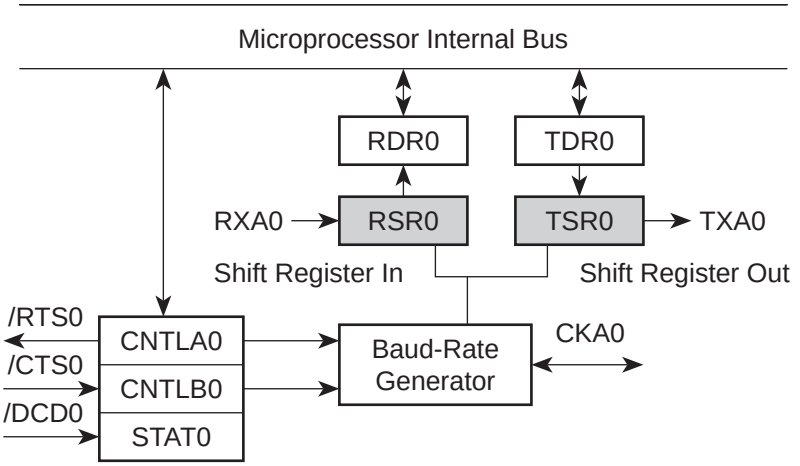


Figure 3-13. Z180 Serial Channel 0

A *polling* driver tests the ready flags (TDRE and RDRF) until a ready condition appears (transmitter data register empty or receiver data register full). If an error condition occurs on receive, the routine must clear the error flags and take appropriate action, if any. If the /CTS line is used for flow control, transmission of data is automatically stopped when /CTS goes high because the TDRE flag is disabled. This prevents the driver from transmitting more characters because it thinks the transmitter is not ready. The transmitter will still function with /CTS high, but exercise care since TDRE is not available to synchronize loading the data register (TDR) properly.

An *interrupt-driven* driver works as follows. The program enables the receiver interrupt as long as it wants to receive characters. The transmitter interrupt is enabled only while characters are waiting in the output buffer. When an interrupt occurs, the interrupt routine must determine the cause: receiver data register full, transmitter data register empty, receiver error, or /DCD0 pin high (channel 0 only). None of these interrupts is edge-triggered. Another interrupt will occur immediately if interrupts are re-enabled without disabling the condition causing the interrupt. The signal /DCD0 is grounded on the BL1400.

Asynchronous Serial Communication Interface

The Z180 incorporates an asynchronous serial communication interface (ASCI) that supports two independent full-duplex channels.

ASCI Status Registers

A status register for each channel provides information about the state of each channel and allows interrupts to be enabled and disabled.

STAT0 (04H)

7	6	5	4	3	2	1	0
RDRF	OVRN	PE	FE	RIE	/DCD0	TDRE	TIE
R	R	R	R	R / W	R	R	R / W

STAT1 (05H)

7	6	5	4	3	2	1	0
RDRF	OVRN	PE	FE	RIE	CTS1E	TDRE	TIE
R	R	R	R	R / W	R	R	R / W

/DCD0 (Data Carrier Detect)

This bit echoes the state of the /DCD0 input pin for Channel 0. However, when the input to the pin switches from high to low, the data bit switches low only after STAT0 has been read. The receiver is held reset as long as the input pin is held high. This function is not generally useful because an interrupt is requested as long as /DCD0 is a 1. This forces the programmer to disable the receiver interrupts to avoid endless interrupts. A better design would cause an interrupt only when the state of the pin changes. In the BL1400, this pin is tied to ground.

TIE (Transmitter Interrupt Enable)

This bit masks the transmitter interrupt. If set to 1, an interrupt is requested whenever TDRE is 1. The interrupt is not edge-triggered. Set this bit 0 to stop sending. Otherwise, interrupts will be requested continuously as soon as the transmitter data register is empty.

TDRE (Transmitter Data Register Empty)

A 1 means that the channel is ready to accept another character. A high level on the /CTS pin forces this bit to 0 even though the transmitter is ready.

CTS1E (CTS Enable, Channel 1)

The signals RXS and CTS1 are multiplexed on the same pin. A 1 stored in this bit makes the pin serve the CTS1 function. A 0 selects the RXS function. (The pin RXS is the CSIO data receive pin.) When RXS is selected, the CTS line has no effect. It is not advisable to use the CTS1 function on the BL1400 because the RXS line is needed to control several other devices on the board.

RIE (Receiver Interrupt Enable)

A 1 enables receiver interrupts and 0 disables them. A receiver interrupt is requested under any of the following conditions: /DCD0 (channel 0 only), RDRF (read data register full), OVRN (overrun), PE (parity error), or FE (framing error). The condition causing the interrupt must be removed before the interrupts are re-enabled, or another interrupt will occur. Reading the receiver data register (RDR) clears the RDRF flag. The EFR bit in CNTLA is used to clear the other error flags.

FE (Framing Error)

A stop bit was missing, indicating scrambled data. This bit is cleared by the EFR bit in CNTLA.

PE (Parity Error)

Parity is tested only if MOD1 in CNTLA is set. This bit is cleared by the EFR bit in CNTLA.

OVRN (Overrun Error)

Overrun occurs when bytes arrive faster than they can be read from the receiver data register. The receiver shift register (RSR) and receiver data register (RDR) are both full.

RDRF (Receiver Data Register Full)

This bit is set when data is transferred from the receiver shift register to the receiver data register. It is set even when one of the error flags is set, in which case defective data is still loaded to RDR. The bit is cleared when the receiver data register is read, when the /DCD0 input pin is high, and by RESET and IOSTOP.

ASCI Control Register A

Control register A affects various aspects of the asynchronous channel operation.

CNTLA0 (00H)

7	6	5	4	3	2	1	0
MPE	RE	TE	/RTS0	MPBR/ EFR	MOD2	MOD1	MOD0
R / W	R / W	R / W	R / W	R / W	R / W	R / W	R / W

CNTLA1 (01H)

7	6	5	4	3	2	1	0
MPE	RE	TE	CKA1D	MPBR/ EFR	MOD2	MOD1	MOD0
R / W	R / W	R / W	R / W	R / W	R / W	R / W	R / W

MOD0–MOD2 (Data Format Mode Bits)

MOD0 controls stop bits: 0 ⇒ 1 stop bit, 1 ⇒ 2 stop bits. If 2 stop bits are expected, then 2 stop bits must be supplied.

MOD1 controls parity: 0 ⇒ parity disabled, 1 ⇒ parity enabled. (See PEO in control registers B for even/odd parity control.)

MOD2 controls data bits: 0 ⇒ 7 data bits, 1 ⇒ 8 data bits.

MPBR/EFR (Multiprocessor Bit Receive/Error Flag Reset)

Reads and writes on this bit are unrelated. Storing a byte when this bit is 0 clears all the error flags (OVRN, FE, PE). Reading this bit obtains the value of the MPB bit for the last read operation when multiprocessor mode is enabled.

/RTS0 (Request to Send, Channel 0)

Store a 1 in this bit to set the RTS0 line from the Z180 high. This line is further inverted by the output driver. This bit is essentially a 1-bit output port without other side effects.

CKA1D (CKA1 Disable)

This bit controls the function assigned to the multiplexed pin (CKA1/–$TEND_0$): 1 ⇒ –TEND0 (a DMA function) and 0 ⇒ CKA1 (external clock I/O for Channel 1 serial port).

TE (Transmitter Enable)

This bit controls the transmitter: 1 ⇒ transmitter enabled, 0 ⇒ transmitter disabled. When this bit is cleared, the processor aborts the operation in progress, but does not disturb TDR or TDRE.

RE (Receiver Enable)

This bit controls the receiver: 1 ⇒ enabled, 0 ⇒ disabled. When this bit is cleared, the processor aborts the operation in progress, but does not disturb RDRF or the error flags.

MPE (Multiprocessor Enable)

This bit (1 ⇒ enabled, 0 ⇒ disabled) controls multiprocessor communication mode, which uses an extra bit for selective communication when a number of processors share a common serial bus. This bit has effect only when MP in Control Register B is set to 1. When this bit is 1, only bytes with the MP bit on will be detected. Others are ignored. If this bit is 0, all bytes received are processed. Ignored bytes do not affect the error flags or RDRF.

ASCI Control Register B

Control Register B for each channel configures multiprocessor mode, parity and baud rate selection.

CNTLB0 (02H) and CNTLB1 (03H)

7	6	5	4	3	2	1	0
MPBT	MP	/CTS PS	PEO	DR	SS2	SS1	SS0
R / W	R / W	R / W	R / W	R / W	R / W	R / W	R / W

SS (Source/Speed Select)

Coupled with the prescaler (PS) and the divide ratio (DR), the SS bits select the source (internal or external clock) and the baud rate divider, as shown in Table 4-2:

DR (Divide Ratio)

This bit controls one stage of frequency division in the baud-rate generator. If 1, then divide by 64. If 0, then divide by 16. This is the only control bit that affects the external clock frequency.

PEO (Parity Even/Odd)

This bit affects parity: 0 ⇒ even parity, 1 ⇒ odd parity. It is effective only if MOD1 is set in CNTLA (parity enabled).

–CTS/PS (Clear to Send/Prescaler)

When read, this bit gives the state of external pin /CTS: 0 ⇒ low, 1 ⇒ high. When /CTS pin is high, RDRF is inhibited so that incoming receive characters are ignored. When written, this bit has an entirely different function. If a 0 is written, the baud rate prescaler is set to divide by 10. If a 1 is written, it is set to divide by 30.

MP (Multiprocessor Mode)

When this bit is set to 1, multiprocessor mode is enabled. The multiprocessor bit (MPB) is included in transmitted data:

start bit, data bits, MPB, stop bits.

The MPB is 1 when MPBT is 1 and 0 when MPBT is 0.

MPBT (Multiprocessor Bit Transmit)

This bit controls the multiprocessor bit (MPB). When the MPB is 1, transmitted bytes will get the attention of other units listening only for bytes with MPB set.

The prescaler (PS), the divide ratio (DR), and the SS bits form a baud-rate generator, as shown in Figure 3-14.

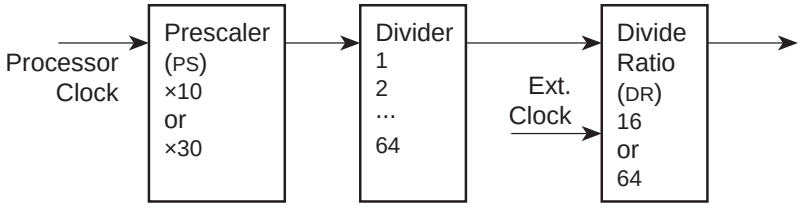


Figure 3-14. Z180 Baud-Rate Generator

Table 3-6 relates ASCI Control Register B to the baud rate. The Z180 in the BL1400 has a 6.144 MHz clock.

Table 3-6. Baud Rate Dependence on ASCI Control Register B

ASCI Control Register B Value	Baud Rate for 6.144 MHz Clock (bps)
00	38,400
01	19,200
02 or 08	9,600
03 or 09	4,800
04 or 0A	2,400
05 or 0B	1,200
06 or 0C	600
0D	300
0E	150

SOFTWARE REFERENCE

Chapter 4 describes the software functions available for the BL1400. Most functions are in the **BL14_15.LIB**; others are found in the Dynamic C BIOS in the BL1400's on-board EPROM.

Sections in this chapter include the following topics.

- Software Development Options
- Programmable Input/Output
- Real-Time Clock Integrated Circuit
- Serial Communication
- Master-Slave Networking
- Support Libraries and Sample Programs

Software Development Options

Dynamic C Development Software

Two versions of Z-World's Dynamic C development software, which runs under Microsoft Windows, are currently available for the BL1400.

- Standard: Maximum 80 Kbytes of code.

  This version of Dynamic C is suitable for programs up to 80K with limited access to extended memory (data items cannot be declared in extended memory).

- Deluxe: Maximum 512K of code, 512 Kb of data.

  This version supports programs with full access to extended memory.

Dynamic C Manuals

Z-World offers three Dynamic C manuals for programming reference.

- ***Dynamic C Technical Reference***
- ***Dynamic C Application Frameworks***
- ***Dynamic C Function Reference***

Programmable Input/Output

The functions described in this section operate the I/O interfaces of the BL1400. Most of these functions are to be found in **BL14_15.LIB**. Some are in the Dynamic C BIOS (in EPROM).

PIO Ports A and B

PIO Port A is available, all 8 bits, in PIO Modes 0, 1, and 3. PIO Port B is available in Mode 3 (bit I/O) only. Bits 4–7 of port B are always available; bits 3 (RTCCLK) and 2 (RTCDAT) are available if not the real-time clock is not used. Bits 0 and 1 of port B are dedicated lines, EN485 and /RTCRST respectively.

On power-up, the Dynamic C BIOS configures the PIO Ports A and B to Mode 3, with all bits as inputs.. There are four registers for Ports A and B, as shown in Table 4-1.

Table 4-1. PIO Register Names

Register Name	Function
PIODA	PIO Port A, data
PIOCA	PIO Port A, control
PIODB	PIO Port B, data
PIOCB	PIO Port B, control

Shadow Registers

Dynamic C maintains and uses shadow registers, **PIOCAShadow** and **PIOCBShadow**, for both ports. Shadow registers are a standard embedded-systems programming technique. Some of the functions that manipulate the hardware I/O ports automatically keep a copy of each register's current configuration. If writing routines (especially interrupt-service routines) that temporarily change the configuration of any hardware I/O port, the interrupt routines first save the appropriate shadow registers before changing the ports, and then use the saved shadow-register contents to restore the ports to their original configuration before exiting.

Function Prototypes

The following function descriptions follow standard C notation for "function prototypes." Function prototypes are not examples of how to use a given function. Instead, they modify the names, parameters, and arguments of functions with a C-type specifier such as **`void`**, **`byte`**, and **`int`**. In addition, some function prototypes use dummy arguments for parameters. When using these functions in an application program, do not include the type specifiers; instead replace the applicable dummy arguments with your own defined arguments.

- **`void setPIOCA( char mask )`**

  Active bits (1s) of **`mask`** are set in **`PIOCAShadow`**. That result is then sent to PIOCA. Active bits becomes ***input*** bits.

  PIOCA ← PIOCAShadow ← PIOCAShadow OR **`mask`**

- **`void resPIOCA( char mask )`**

  Active bits (1s) of **`mask`** are reset in **`PIOCAShadow`**. That result is then sent to PIOCA. Active bits becomes ***output*** bits.

  PIOCA ← PIOCAShadow ← PIOCAShadow AND NOT **`mask`**

- **`void setPIODA( char mask )`**

  Active bits (1s) of **`mask`** are set in the current output of PIODA.

  PIODA ← PIODA OR **`mask`**

- **`void resPIODA( char mask )`**

  Active bits (1s) of **`mask`** are reset in the current output of PIODA.

  PIODA ← PIODA AND NOT **`mask`**

- **`void setPIOCB( char mask )`**

  Active bits (1s) of **`mask`** are set in **`PIOCBShadow`**. That result is then sent to PIOCB. Active bits becomes ***input*** bits.

  PIOCB ← PIOCBShadow ← PIOCBShadow OR **`mask`**

- **`void resPIOCB( char mask )`**

  Active bits (1s) of **`mask`** are reset in **`PIOCBShadow`**. That result is then sent to PIOCB. Active bits becomes ***output*** bits.

  PIOCB ← PIOCBShadow ← PIOCBShadow AND NOT **`mask`**

- **`void setPIODB( char mask )`**

  Active bits (1s) of **`mask`** are set in the current output of PIODB.

  PIODB ← PIODB OR **`mask`**

- **`void resPIODB( char mask )`**

  Active bits (1s) of **`mask`** are reset in the current output of PIODB.

  PIODB ← PIODB AND NOT **`mask`**

Real-Time Clock Integrated Circuit

The real-time clock (RTC), a DS1302 trickle charge timekeeper, Real-Time Clock Integrated Circuit, has a clock/calendar smart enough to account for leap year and the different number of days in a month. It also has a 31-byte scratchpad RAM, and can recharge its own backup battery (if it has one). Both the clock/calendar and the RAM can be written in burst mode.

Global Time and Date Structure

Dynamic C automatically creates the following global structure to hold the time and date.

```
struct tm {
  char tm_sec;        // 0-59
  char tm_min;        // 0-59
  char tm_hour;       // 0-23
  char tm_mday;       // 1-31
  char tm_mon;        // 1-12
  char tm_year;       // 0-150 (1900-2050)
  char tm_wday;       // 0-6 where 0 means Sunday
};
```

Dynamic C creates this structure automatically, and so it does not have to be declared in an application.

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

  Reads the RTC into the structure **`*t`**. If the RTC is stopped, this function stores the value for midnight, 1 JAN 1980 in **`*t`**, and returns -1. Otherwise the function stores the current time in **`*t`** and returns 0.

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

  Writes the values in the structure **`*t`** to the RTC. The function returns 0 if successful; it returns –1 if it is unable to start the RTC.

- **`int WriteRam1302( int ram_loc, byte data )`**

  Writes data to any of the 31 (0–30) RAM locations of the DS1302. The function returns 1 if successful, or –1 if **`ram_loc`** is out of range.

- **`int ReadRam1302( int ram_loc )`**

  Reads data from any of the 31 (0–30) RAM locations of the DS1302. The function result is the data value, or –1 if **`ram_loc`** is out of range.

- **`void WriteBurst1302( void* pdata, int count )`**

  Writes **`count`** bytes, in burst mode, to the DS1302, starting at RAM location 0. The term **`pdata`** points to the data.

- **`void ReadBurst1302( void* pdata, int count )`**

  Reads **`count`** bytes, in burst mode, from the DS1302, starting at RAM location 0. The term **`pdata`** points to a storage area for the data.

- **`void Write1302( int reg, byte data )`**

  Writes **`data`** to a specified register of the DS1302.

- **`int Read1302( int reg )`**

  Reads data from a specified register of the DS1302. The function returns the data value.

- **`Charger1302( int on_off, int diode, int resistor )`**

  Turns the trickle charger on the DS1302 on (when **`on_off`** is non-zero) or off (**`on_off`** is 0). The term **`diode`** is 1 or 2 for the number of diodes on the charging circuit. The term **`resistor`** is 2, 4 or 8 for the resistance (in kilohms) of the circuit. If a supercapacitor or a rechargeable battery is connected across VBAT, and GND is charged, then the RTC continues to run and RAM data are not lost when power is removed from the BL1400.

Do *not* enable the trickle charge circuit if a nonrechargeable battery is used to back up the DS1302. The battery could explode.

555 Timer

The 555 timer, used as an analog input channel to measures resistance according to the formula

$$\Delta T = 1.1\ RC_1$$

Software starts the 555 measurement process, and simultaneously starts one of the Z180 timers. When the 555 finishes its measurement (i.e., after ΔT), it trips the timer. The timer value is proportional to R. C_1 is 0.47 µF.

There is a certain amount of overhead in the software, including these functions, which will cause some imprecision in readings. The resistor and capacitor will vary from their nominal values within a certain tolerance. The finished system may introduce other unpredictable factors. Therefore, Z-World recommends that resistance measurement software be calibrated, on site, if accuracy is important.

- **`void Set555( uint max_count )`**

  Loads Z180 Timer 0 with **`max_count`**, setting it to generate one interrupt. The function prepares DMA Channel 0 to receive data from TMRD0L and prepares INT1 and DREQ0 to receive a done signal from the 555 chip. Lastly, the function triggers the 555 timer chip.

- **`int Read555( uint *lapse_count )`**

  Reads Z180 Timer 0 for the number of counts that occurred during ΔT ($1.1\ RC_1$). The timer count is returned in **`*lapse_count`**. The 555 timer must be started with a call to **`Set555`**.

  This function returns 0 if Timer 0 has not yet timed out and the 555 has not reached $t = 1.1\ RC_1$. It returns 1 if the 555 has reached $t = 1.1\ RC_1$ and has generated an interrupt on INT1 and DREQ0. It returns –1 if Timer 0 has reached **`maxcount`** and the 555 has not yet reached $t = 1.1\ RC_1$.

  There is no calibration built into this function.

Flash EPROM

The top-most 512 bytes of the flash EPROM are available as nonvolatile memory. This storage area simulates an EEPROM. Its logical addresses are 0–511. Locations 0 and 1 are reserved for the Dynamic C BIOS to store the operation mode and the baud rate, respectively.

- **`int ee_rd( int address )`**

  Reads and returns data from flash EPROM storage location **`address`**. The function returns –1 if a non-flash EPROM is used.

- **`int ee_wr( int address, int data )`**

  Writes data to flash EPROM" storage location **`address`**. The function returns –1 if a non-flash EPROM is used.

- **`int WriteFlash( ulong addr, byte buf, int num )`**

  Writes **`num`** bytes from **`buf`** to flash EPROM, starting at **`addr`**. The term **`addr`** is an absolute physical address.

  To do this, allocate flash data in the Dynamic C program by declaring ***initialized*** variables or arrays, or initialized **`xdata`**. For **`xdata`**, the data name must be passed directly to the following function.

```
xdata my_data { 0, 0xFF, 0x08 };
...
WriteFlash( my_data, my_buffer, my_count );
```

For normal data, the physical address of the data must be passed to the following function.

```
char xxx[] = { 0, 0xFF, 0x08 };
...
WriteFlash( phy_adr(xxx), my_buffer, my_count );
```

RETURN VALUE:

0 if the operation was successful

–1 if no flash EPROM is present

–2 if a physical address is within the BIOS area (low 8K)

–3 if a physical address is within the symbol table

–4 if the write times out.

If some form of initialization is not included when data are declared, the data will be placed in RAM, not ROM, and this function will not work.

It may seem contradictory to write to "read-only" memory. But flash EPROM, despite its name, is not really read-only memory. Writing to flash EPROM, in essence, treats the flash memory as read/write non-volatile memory.

Flash EPROM is rated for 10,000 writes. In practice, flash EPROM has performed for up to 100,000 writes. Z-World recommends that any writes to the flash EPROM be made by the programmer rather than automatically by the software to maximize the life of the flash EPROM.

Serial Communication Software

This section describes functions for Port 0 of the Z180.

Interrupt Handling for Z180 Port 0

Normally, a serial interrupt service routine is declared with the following compiler directive.

```
#INT_VEC SER0_VEC routine
```

However, if the same serial port is used for Dyamic C programming, the program has to be downloaded first with Dynamic C before the address of the serial interrupt service routine is loaded into the interrupt vector table. That is, the service routine must be loaded at run-time. The function

```
reload_vec( int vector, int(*serv_function)() )
```

will load the address of the service function into the specified location in the interrupt vector table. In this case, do not the **#INT_VEC** directive. Once the service routine has taken over, the program cannot be debuged in Dynamic C.

When executable programs are generated for EPROM or for download to RAM, there will be no need for communication with Dynamic C. The compile-time directive **#INT_VEC** can then be used freely.

RS-232 Software Support

- **int Dinit_z0(void *rbuf, void *tbuf,**
  int rsize, int tsize,
  byte mode, byte baud,
  byte modem, byte echo);

  Initializes Z180 Port 0 for communication.

  PARAMATERS: **rbuf** is a pointer to the receive buffer.

  tbuf is a pointer to the transmit buffer.

  rsize is the size of the receive buffer.

  tsize is the size of the transmit buffer.

  mode selects the operation mode as follows.

bit 0	0	1 stop bit
	1	2 stop bits
bit 1	0	no parity
	1	with parity
bit 2	0	7 data bits
	1	8 data bits
bit 3	0	even parity
	1	odd parity
bit 4	0	CTS, RTS disabled
	1	CTS, RTS enabled

  baud is the baud rate in multiples of 1200 bps (e.g., specify 8 for 9600 bps).

  modem If 1, modem is supported. If 0, there is no modem.

  echo If 1, every character is echoed. If 0, there is no echoing.

  If CTS/RTS handshaking is selected, transmission from the sender is disabled (by raising RTS) when the receive buffer is 80 % full. The software lowers RTS (enabling the sender to transmit) when the receive buffer falls below 20 % of capacity. In a similar manner, a remote system can prevent transmission of data by the Z180 Port 0 by asserting its RTS (connected to the Z180 Port 0 CTS).

- **`int Dread_z01ch( char* ch )`**

  Reads a character from the circular receive buffer into character **`ch`**.

  RETURN VALUE:

  0 buffer is empty

  1 byte has been successfully extracted from buffer

- **`int Dwrite_z01ch( char ch )`**

  Places a character in the transmit buffer. If not already transmitting, the function initiates transmission.

  RETURN VALUE:

  0 transmit buffer did not have space for **`ch`**

  1 write was successful

- **`int Dread_z0( char* buffer, char terminate )`**

  Checks the receive buffer for a message terminated with the character **`terminate`**. The message is copied to **`buffer`** and terminated with a null character according to the C convention.

  RETURN VALUE:

  0 no message found with the specified terminating character

  1 message has been successfully extracted from buffer

- **`int Dwrite_z0( char* buffer, int count )`**

  Copies **`count`** bytes from **`buffer`** to the transmit buffer. If not already transmitting, the function initiates transmission.

  RETURN VALUE:

  0 transmit buffer did not have space for **`count`** bytes

  1 write is successful.

- **`void Dz0send_prompt()`**

  Places CR, LF and > in the transmit buffer.

- **`void Dreset_z0rbuf()`**

  Resets the receive buffer.

- **`void Dreset_z0tbuf()`**

  Resets the transmit buffer and stops transmission.

- **`void Dkill_z0()`**

  Turns off Z180 Port 0.

- **`void z0binaryreset()`**

  Sets the serial communication mode to regular ASCII mode. This means that the backspace character is tracked.

- **`void z0binaryset()`**

  Sets the serial communication mode to binary. This means that all data received are placed directly to the receive buffer without preprocessing.

XMODEM Commands

- **`int Dxmodem_z0down( char* buffer, int count )`**

  Sends (downloads) **`count`** 128-byte blocks in **`buffer`** using XMODEM protocol.

  RETURN VALUE:

  0 timed-out (no transfer)
  1 successful transfer
  2 canceled transfer (canceled by receiver side)

- **`int Dxmodem_z0up( ulong address, int* pages, int dest, int(*parser)() )`**

  Receives (uploads) a file using XMODEM protocol.

  PARAMETERS: **`address`** is the physical address in RAM where the received data are to be stored. If the receive buffer is allocated by **`xdata`** (a Dynamic C keyword to allocate extended memory data), then the name of the array may be used for the **`address`** argument. If, however, the data area is allocated using "normal" C, the logical address of the buffer must first be converted to a physical address using the library function **`phy_adr`**.

  `pages` is the number of 4K blocks of data that have been transferred.

  `dest` If an RS-485 master-slave network is set up, specify **`dest`** = 0 when the upload is intended for the master. If **`dest`** is non-zero, the upload is intended for the designated slave.

  `parser` is the function that handles parsing of the uploaded data.

  RETURN VALUE:

  0 timed-out (no transfer)
  1 successful transfer
  2 canceled transfer (canceled by sender side)

- **`int z0modemset()`**

  Returns information about modem selection. The function returns 1 if the modem option is selected (with **`Dinit_z0`**) and 0 if not.

- **`int z0modemstat()`**

  Returns the status of the modem. The function returns 1 if the modem is in command mode and 0 if it is in data mode.

Miscellaneous Functions

- **`int Dget_modem_command()`**

  Deciphers Hayes-compatible modem commands. The function returns –1 if no modem command is matched.

  The modem commands are given in Table 4-2.

Table 4-2. Modem Commands

Mode	Command	Action
0	`"\nOK"`	"OK" response
1	`"\nCONNECT"`	Connect at 300 bps
2	`"\nRING"`	Ring detected
3	`"\nNO CARRIER"`	No carrier
4	`"\nERROR"`	Command error
5	`"\nCONNECT 1200"`	Connect at 1200 bps
6	`"\nNO DIALTONE"`	No dial tone
7	`"\nBUSY"`	Line busy
8	`"\nNO ANSWER"`	No answer
9	`"\nCONNECT 2400"`	Connect at 2400 bps
10	`"\n"`	Line feed

A Hayes Smart Modem (or compatible) is recommended. A NULL modem is needed between the BL1400 and the modem.

Some modems may require that the RTS(4), CTS(5), and DTR(20) lines on the modem side be tied together.

- **`void Drestart_z0modem()`**

  Restarts the modem (during start of program or abnormal operation).

- **`void Dz0modem_chk( char* buffer )`**

  Checks the **`buffer`** for valid modem commands. The function takes the appropriate response to the modem command if it finds a valid modem command.

  RETURN VALUE:

  0 valid modem command

  –1 invalid modem command

- **`void Dz0_circ_int()`**

  This is an interrupt service routine for Z180 Port 0.

- **`void Ddelay_1sec()`**

  Creates adelay of approximately 1 second. **`suspend(40)`** is used if **`RUNKERNEL`** is defined.

- **`void Ddelay_100ms()`**

  Creates a delay of approximately 100 ms.

- **`void reload_vec( int vector, int(*function)() )`**

  Loads the address of a function into the interrupt vector table.

  This function is only useful during program development, when Z180 Port 0 is also used as the Dynamic C programming port. The compile-time interrupt directive can load the serial service function's address in the interrupt vector table to generate the executable code for EPROM or for download to RAM.

  PARAMETERS: **`vector`** is the offset for the specific interrupt.

  `function` is a pointer to the interrupt service function.

- **`int getcrc( char* buffer, byte count, int accum )`**

  Computes the CRC (cyclic redundancy check, or check sum) for data in **`buffer`**. Calls to **`getcrc`** can be "concatenated" to compute the CRC for a large buffer.

  PARAMETERS: **`buffer`** represents the characters for which to compute the CRC.

  `count` is the number of characters in **`buffer`**, limited to 255, for this function.

  `accum` is the accumulated CRC value from previous computation.

  RETURN VALUE: The function returns the integer CRC value.

- **`void resetZ180int()`**

  This is a general reset function that resets, or disables, interrupts for the DMA channels, Z180 serial channels 0 and 1, the PRT timers, the CSIO, INT1, and INT2.

Master-Slave Networking

- **`void op_init_z1( char baud, char* rbuf, byte address )`**

  Initializes Z180 Port 1 for RS-485 9th-bit binary communication. The data format defaults to 8 bits, no parity, 1 stop bit.

  PARAMETERS: **`baud`** selects the baud rate in multiples of 1200 bps (specify 16 for 19,200 bps).

  `rbuf` is the receive buffer.

  `address` is the network address of the board: 0 for the master board, 1–255 for slaves.

- **`int check_opto_command()`**

  Checks for a valid and completed command or reply in the receive buffer.

  RETURN VALUE:

  0 if there is no completed command or message available

  –1 if there is a completed command or reply available

  –2 if the completed command or reply has a bad CRC check

- **`int sendOp22( byte dest, char* message, byte len, int delays )`**

  The master sends a message to the slave and waits for a reply. The function puts the message in the following format.

  `[slave id] [len+2] [ ] [ ]...[ ] [CRC hi][CRC lo]`

  PARAMETERS: **`dest`** is the slave destination (1–255).

  `message` is the message.

  `len` is the length of the message. The maximum message length is 251 bytes.

  `delays` is the number of delays to wait for the slave reply. Each delay is ~50 ms if the RTK is used. If the RTK is not used, the software-generated delay is approximately 50 ms.

  RETURN VALUE:

  –1 if there is no reply from the slave

  –2 if a completed reply has a bad CRC

  1 if there is a completed reply with a proper CRC

  The slave's reply is stored in the receive buffer initialized with **`op_init_z1()`**.

- **`void replyOpto22( char* reply, byte count, int delays )`**

  The slave replies to the master's inquiry. The function puts the reply in the following format.

  `[count+2] [ ] [ ]...[ ] [CRC hi] [CRC lo]`

  PARAMETERS: **`reply`** is the slave's reply string.

  `count` is the length of the reply. Because two CRC bytes are appended at the end, the longest reply is 252 bytes.

  `delays` is the number of delays before the message is transmitted back. Each delay is ~50 ms when the RTK is used. If the RTK is not in use, the software-generated delay is approximately 50 ms.

  The delay is implemented with **`suspend()`** if the real-time kernel is being used. Otherwise, the delay is a software countdown delay.

Miscellaneous Functions

- **`void misticware( char* tbuf, char count )`**

  This is the gateway for RS-485 9th-bit binary communication. The receive buffer and the transmit buffer must have been previously set up. Interrupt-driven transmission must be enabled.

  PARAMETERS: **`tbuf`** is the transmit buffer. Data in the buffer should already be in the correct format

  `count` is the number of bytes to be transmitted

- **`void optodelay()`**

  Produces a delay of ~50 ms. The delay is implemented with **`suspend()`** if the real-time kernel is being used. Otherwise, it is a software countdown delay.

- **`int rbuf_there()`**

  Monitors the receive buffer for a completed command or reply.

  RETURN VALUE:

  1 if a completed command or reply is available
  0 if a completed command or reply is not available

- **`void op_send_z1( char* tbuf, byte count )`**

  This function is called by **`misticware()`** to initiate transmission of data.

- **`void op_rec_z1()`**

  This function is called by **`misticware()`** to reset and to ready the receiver to receive data.

- **`void op_kill_z1()`**

  Turns off Z180 Port 1. The RS-485 driver is also disabled.

- **`void z1_op_int()`**

  This is an interrupt service routine for the Z180 Port 1 used in master-slave networking.

Support Libraries and Sample Programs

Dynamic C provides libraries and software samples. These software libraries and sample programs are included on the Dynamic C distribution disk.

`DRIVERS.LIB` and **`PLC_EXP.LIB`** apply to most Z-World controllers. The libraries listed in Table 4-3 apply speciifically to the BL1400.

Table 4-3. Dynamic C Libraries

Library	Description
`Z0232.LIB`	RS-232 library for Z180 Port 0
`BL14_15.LIB`	Low-level drivers for the BL1400
`MODEM232.LIB`	Miscellaneous functions common to the other communication libraries
`NETWORK.LIB`	RS-485 9-bit binary half-duplex support for Z180 Port 1

Sample Programs

The sample programs listed in Table 4-4 can be found in the **`SAMPLES\BL14_15`** subdirectory in the Dynamic C directory.

Table 4-4. BL1400 Sample Programs

Program	Description
MG485MST.C	RS-485 master program that communicates to another BL1400 running **MG485SLV.C**
MG485SLV.C	RS-485 slave program that communicates to another BL1400 running **MG485MST.C**
MG555.C	Uses Timer0, INT1 and DMA0 to read back the elapsed time when the 555 times out
MG555PIO.C	Uses the 555 to time square waves generated out of the PIODA bits
MGCHRGER.C	Charges a supercapacitor or rechargeable battery with the DS1302
MGDEMO_RT.C	Shows the real-time kernel (RTK) for the BL1400
MGDSRAM.C	Writes and reads data from the RAM space of the DS1302
MGFLASH.C	Stores data to flash EPROM “initialized data space”
MGFLSHEE.C	Writes and reads data from simulated EEPROM of the flash EPROM
MGLCD.C	Uses PIOA to drive an LCD
MGLCDCG.C	Loads and uses special LCD characters
MGLCDKEY.C	Uses PIOA and PIOB to drive keypad and print to LCD
MGPIODAC.C	Uses bit 2 of PIOA as a simple 10-bit DAC.
MGPIOM0.C	Uses PIOA in Mode 0
MGPIOM1.C	Uses PIOA in Mode 1
MGPIOM3A.C	Uses PIOA in Mode 3
MGPIOM3B.C	Uses PIOB in Mode 3
MGPLCDAC.C	Uses PIOA to drive an XP8600
MGPLCRLY.C	Uses PIOA to drive an XP8300 or an XP8400
MGPLCUIO.C	Uses PIOA to drive an XP8200
MGPRTPIO.C	Uses PRT0 to time square waves generated out of the PIODA
MGRS-232.C	Uses RS-232 port on Z180 Port 0
MGRTC.C	Reads and displays time from the real-time clock

APPENDIX A: TROUBLESHOOTING

Appendix A provides procedures for troubleshooting system hardware and software.

Out of the Box

Check the items listed below before starting development. Rechecking may help to solve problems found during development.

- Verify that the BL1400 runs in standalone mode before connecting any expansion boards or I/O devices.
- Verify that the entire system has good, low-impedance, separate grounds for analog and digital signals. The BL1400 is often connected between the host PC and another device. Any differences in ground potential can cause serious problems that are hard to diagnose.
- Do not connect analog ground to digital ground anywhere.
- Verify that the host PC's COM port works by connecting a known-good serial device to the COM port. Remember that a PC's COM1/COM3 and COM2/COM4 share interrupts. User shells and mouse software, in particular, often interfere with proper COM-port operation. For example, a mouse running on COM1 can preclude running Dynamic C on COM3.
- Use the Z-World power supply supplied with the developer's kit. If another power supply must be used, verify that it has enough capacity and filtering to support the BL1400.
- Use the Z-World cables supplied.
- Experiment with each peripheral device connected to the BL1400 in order to determine how it appears to the BL1400 when powered up, powered down, and/or when its connecting wiring is open or shorted.

Dynamic C Will Not Start

If Dynamic C will not start, an error message on the Dynamic C screen (for example, **Target Not Responding** or **Communication Error**), announces a communication failure:

- *Wrong Baud Rate* — Either Dynamic C's baud rate is not set correctly or the BL1400's baud rate is not set correctly. Both baud rates must be identical. The baud rate is stored in the flash EPROM. Chapter 2 described how to change this rate. Dynamic C's baud rate is set by the **Serial Options** command in the **OPTIONS** menu.
- *Wrong Communication Mode* — Both the PC and the BL1400 must be using the same protocol: RS-232. Use Dynamic C's **Serial Options** command in the **OPTIONS** menu to check and alter the protocol for the PC.

- *Wrong COM Port* — A PC generally has two serial ports, COM1 and COM2. Specify the one used in the Dynamic C **Serial Options** command in the **OPTIONS** menu. Use trial and error, if necessary.
- *Wrong Operating Mode* — Communication with Dynamic C is lost if the jumpers are set for standalone operation. Reconfigure the board for programming as explained in Chapter 2.
- *Wrong Memory Size* — Jumpers JP1 and JP2 of the BL1400 specify the EPROM size. The size of the development board RAM is specified with jumper JP1 on the development board.
- If all else fails, connect the serial cable to the BL1400 after power up. If the PC's RS-232 port supplies a large current (most commonly on portable and industrial PCs), some RS-232 level converter ICs go into a nondestructive latch-up. Connecting the RS-232 cable after power up alleviates this problem.

Dynamic C Loses Serial Link

Dynamic C will lose its link if the program disables interrupts for more than 50 ms. If a communication method is used that is not driven by the nonmaskable interrupt (NMI), make sure that interrupts are not disabled for more than 50 ms. This is not a concern if a communication method driven by NMI is used.

BL1400 Resets Repeatedly

A system reset will occur every 1.2 s if the watchdog timer is not "hit." Dynamic C "hits" the timer, but a user program must include a call to **`hitwd`** to make sure the watchdog timer is hit periodically.

Input/Output Problems

A strobe is needed to move data in PIO modes 0 and 1. The strobe lines are connected to H5 and H6. Use Mode 3 for static input. Mode 1 may appear to work, but will be erratic because the strobe line floats.

Power-Supply Problems

If the external power supply does not have sufficient capacity, an additional load such as an LED can trigger a power-fail interrupt, initiating a hardware reset. The reset triggers the load to be turned off, but then the computer restarts and turns the load back on. The oscillation can be corrected by increasing the size of the power supply.

Common Programming Errors

- Values for constants or variables out of range.

Type	Range
`int`	–32,768 (-2^{15}) to +32,767 ($2^{15}-1$)
`long int`	–2,147,483,648 (-2^{31}) to +2147483647 ($2^{31}-1$)
`float`	-6.805646×10^{38} to $+6.805646 \times 10^{38}$

- Counting up from, or down to, one instead of zero. In the software world, ordinal series often begin or terminate with zero, not one.
- Confusing a function's definition with an instance of its use in a listing.
- Not ending statements with semicolons.
- Not inserting commas as required in functions' parameter lists.
- Leaving out an ASCII space character between characters forming a different legal—but unwanted—operator.
- Confusing similar-looking operators such as **&&** with **&**, == with =, // with /, etc.
- Inadvertently inserting ASCII nonprinting characters into a source-code file.

APPENDIX B: ***SPECIFICATIONS***

Appendix B provides the dimensions and specifications for the BL1400 controller.

Electrical and Mechanical Specifications

BL1400

Figure B-1 shows the dimensions and the locations of the mounting holes for the BL1400.

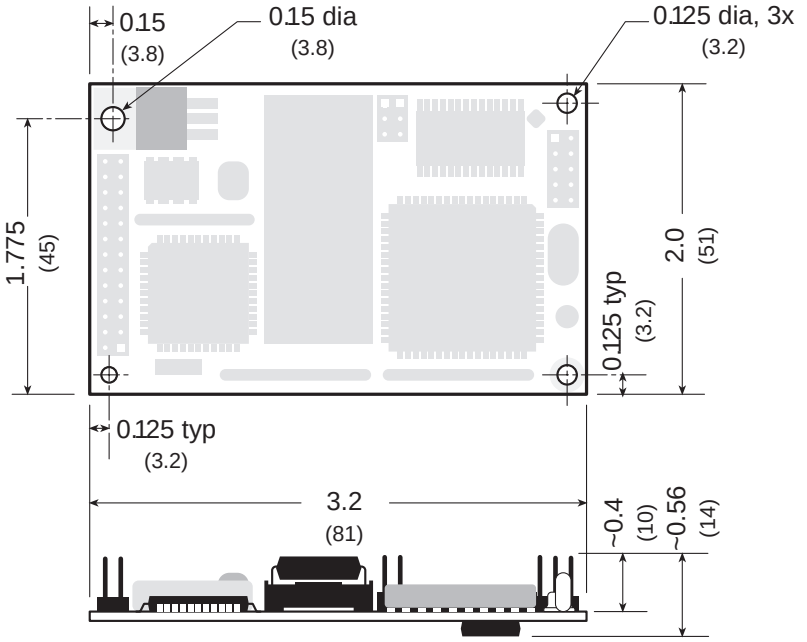


Figure B-1. BL1400 Dimensions

Table B-1 lists the electrical, mechanical, and environmental specifications for the BL1400.

Table B-1. BL1400 General Specifications

Parameter	Specification
Board Size	3.2″ × 2.0″ × 0.56″ (81 mm × 51 mm × 14 mm)
Operating Temperature	-40°C to +70°C
Humidity	5 % to 95 %, noncondensing
Input Voltage, Current	9 V to 12 V DC, 120 mA, linear regulator
Configurable I/O	12, TTL- and CMOS-compatible*
Digital Inputs	See configurable I/O above
Digital Outputs	See configurable I/O above
Analog Inputs	No
Analog Outputs	No
Resistance Measurement Input	One, 0 kΩ to 405 kΩ
Processor	Z180
Clock	6.144 MHz
SRAM	32K, surface mount
EPROM	32K
Flash EPROM	Supports up to 256K**
EEPROM	No
Counters	Software-implementable
Serial Ports	One RS-232 (with RTS/CTS handshake) and one RS-485 (two-wire)
Serial Rate	Up to 38,400 bps
Watchdog/Supervisor	Yes
Time/Date Clock	Yes
Backup Battery	Connections for user-supplied battery (backup time/date only)
Keypad and LCD	Supported on digital PIO lines
PLCBus Port	No, limited expansion board support on PIO lines

* I/O lines are software-selectable as either inputs or outputs

** Flash EPROM replaces standard EPROM

Development Board

Figure B-2 shows the dimensions and mounting hole locations of the BL1400 development board.

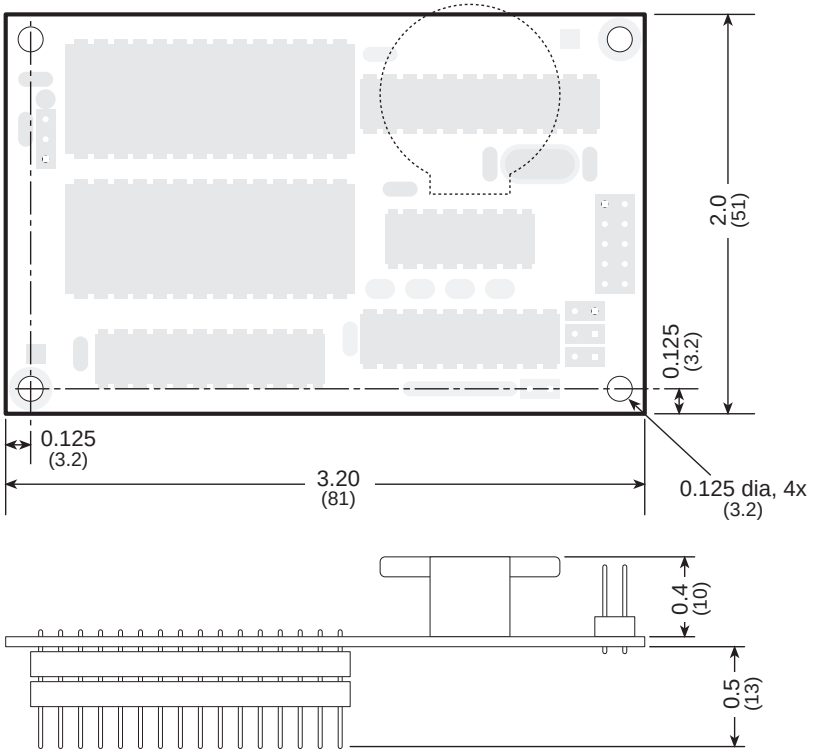


Figure B-2. BL1400 Development Board Dimensions

Prototyping Board

Figure B-3 shows the dimensions and mounting hole locations of the BL1400 prototyping board.

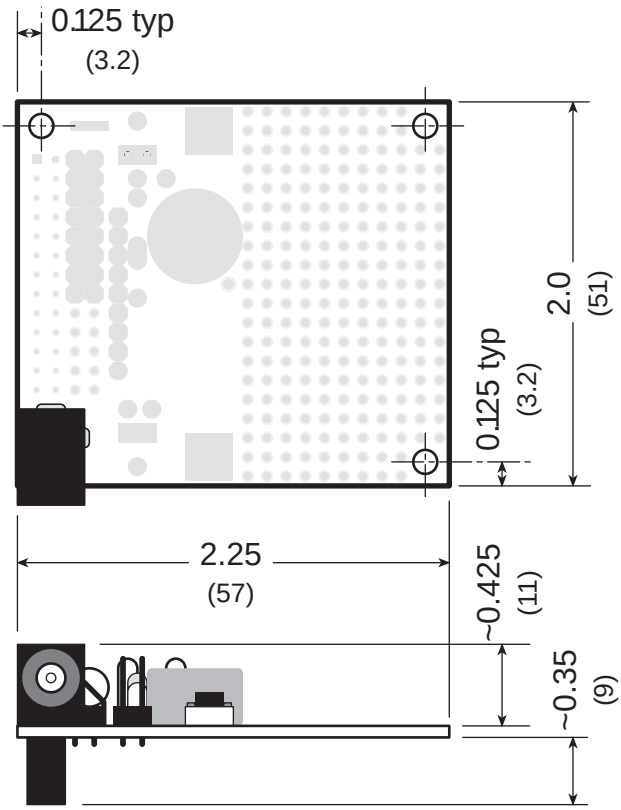


Figure B-3. BL1400 Prototyping Board Dimensions

Base Plate

Figure B-4 shows the dimensions and mounting hole locations of the BL1400 base plate. The base plate is made from 0.060" aluminum.

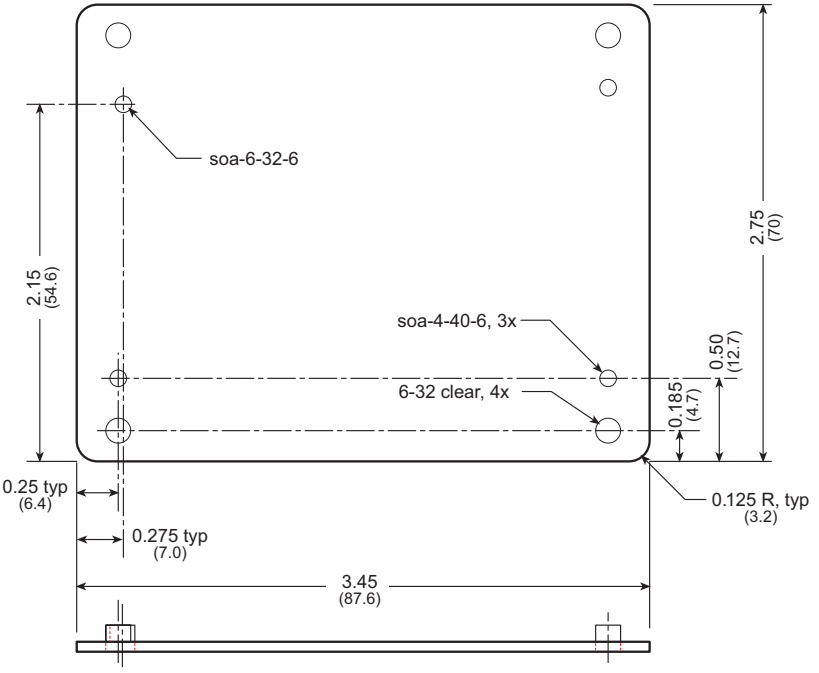


Figure B-4. BL1400 Base Plate Dimensions

Connectors

BL1400 connectors are not polarized or keyed. Carefully observe connector orientation and pin alignment before making a connection and before applying power.

Temperature

The BL1400 operates at ambient temperatures from –40°C to +70°C. The BL1400 may be stored at temperatures from –55°C to +85°C.

Power

The unregulated input is 9 V to 12 V DC. The maximum power dissipation without additional heat sinking is 1.0 W at 70°C. The BL1400 itself draws 120 mA, dissipating about 0.85 W for a 12 V input. Additional heat sinking can be obtained by mounting the BL1400 on the optional aluminum base plate, or by using aluminum standoffs.

See Appendix C, “Power Management,” for more information.

Headers and Jumpers

BL1400

Figure B-5 shows the location of the BL1400's headers.

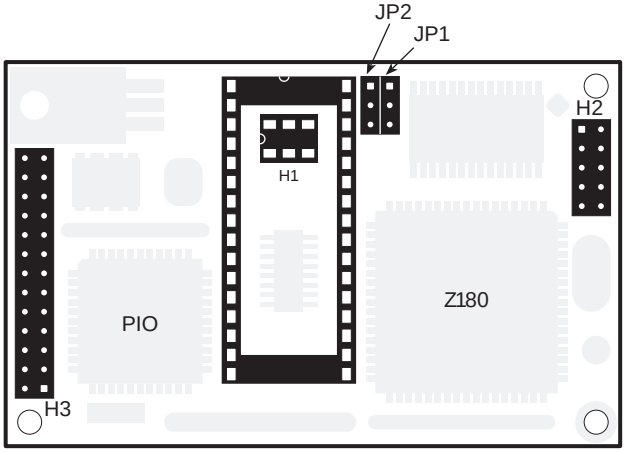


Figure B-5. Location of BL1400 Headers and Jumpers

Table B-2 lists the functions of the BL1400's headers and their jumper settings.

Table B-2. BL1400 Headers and Jumpers

Header	Pins	Description
H1	—	Connects to Development Board header H2 to develop applications. This header has no I/O.
H2	—	RS-232 port, may be used to develop application with Prototyping Board.
H3	—	PIO port signals.
JP1	1–2	Connect for 28-pin EPROM (factory default).
	2–3	Connect for 32-pin EPROM or when using Development Board.
JP2	1–2	Connect for flash EPROM.
	2–3	Connect for regular EPROM (factory default), or when using Development Board.

Development Board

Figure B-6 shows the location of the Development Board's headers.

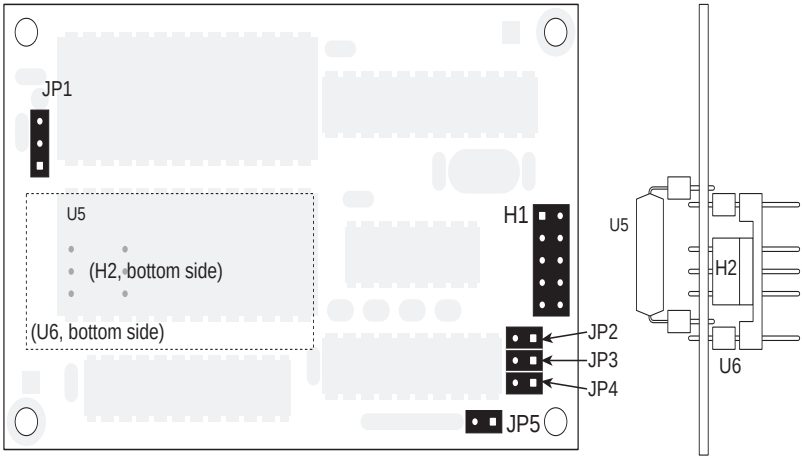


Figure B-6. BL1400 Development Board Header Locations

Table B-3 lists the functions of the Development Board's headers and their jumper settings.

Table B-3. BL1400 Development Board Headers and Jumpers

Header	Pins	Description
H1	—	RS-232 programming port.
H2	—	Connects to H1 on the BL1400 main board. This header has no I/O.
H3	—	PIO port signals.
JP1	1–2	Connect for 512K SRAM on the Development Board.
	2–3	Connect for 32K or 128K SRAM on the Development Board.
JP2	—	Not used.
JP3 JP4	No connections	Program at 19,200 bps.
	JP3 connected, JP4 not connected	Program at 9600 bps.
	JP3 not connected, JP4 connected	Program at 57,600 bps.
	JP3 connected, JP4 connected	Run program in Development Board RAM.
JP2	—	Not used.
U6	—	Connects to socket U1 on the BL1400.

APPENDIX C: POWER MANAGEMENT

Appendix C provides detailed information on power systems and sources. Sections include the following topics.

- Direct Current Input
- Power Regulator

Direct Current Input

During software development, power (9 V to 12 V DC) comes through the DC input jack of the prototyping board. In the absence of this board (for example, when system development has been completed) power for the BL1400 must be applied to pin 25 on header H3.

Power Regulator

The BL1400 has an onboard +5 V linear regulator that accepts an input voltage of 9 V to 12 V DC. The BL1400 itself draws approximately 120 mA. The regulator has some excess capacity to power expansion boards or external loads. The regulator package is a TO-220 rated for 1.0 W at an ambient temperature of 70°C without additional heat sinking.

Heat sinking can be enhanced by mounting the BL1400 so that the regulator makes contact with a mounting stud and a metal chassis or plate. Forced-air cooling, if available, will dissipate additional power.

Maximum Power Dissipation

The maximum allowable power dissipation of the onboard 5 V regulator at any ambient temperature is a function of the maximum junction temperature at which the regulator is operating. If the maximum power dissipation is exceeded, the regulator's junction temperature will rise above T_{JMAX} and the electrical specifications will not apply. If the die temperature rises above 150°C, the regulator will go into thermal shutdown. The LM340 used by the BL1400 has a maximum junction temperature of 125°C.

The maximum power dissipation can be computed as follows.

$$P_{MAX} = \frac{\left(T_{J_{MAX}} - T_A\right)}{\theta_{TOTAL}} \qquad \text{(C-1)}$$

where

P_{MAX} is the maximum power dissipation in watts

T_{JMAX} is the maximum junction temperature in degrees Celsius

T_A is the ambient temperature in degrees Celsius

θ_{TOTAL} is the thermal resistance from the junction to the ambient air, °C/W.

Thus,

$$P_{MAX} = \frac{(125°C - 70°C)}{54°C/W}$$
$$= 1.019 \text{ W at } 70°C.$$

In terms of the input voltage and current consumption of the board,

$$P_{MAX} = (V_{IN} - 5) \times I \qquad \text{(C-2)}$$

where

V_{IN} is the input dc voltage to the board (9 V to 12 V), and

I is the current consumption of the board.

Heat Dissipation with the BL1400 Base Plate

The junction-to-case thermal resistance, θ_{JC}, of the LM340 is 4°C/W. The thermal resistance (θ_{CS}) from the case to the top surface of the printed circuit board (PCB) is about 1°C/W.

From the top surface to the bottom surface, the three heat paths are the plating on the two 0.150-inch-diamter holes and through the PCB. Assuming a solder lamination of 0.001″, the heat resistance across one hole is approximated as follows.

$$\theta_{HOLE} = \frac{0.06}{(2.54 \times 0.15 \times 0.001 \times \pi \times 3.46)}$$
$$\approx 14.5°C/W.$$

The thermal resistance across the 0.6″ × 0.4″ PCB board is approximated as follows.

$$\theta_{PCB} = \frac{0.06}{(2.54 \times 0.6 \times 0.4 \times 0.0016)}$$
$$\approx 62°C/W.$$

The contact thermal resistance from the bottom of the PCB to the 6-32 PEM is approximately 1°C/W. The thermal resistance across a 3-16 long aluminum PEM is approximately

$$\theta_{PEM} = \frac{0.1875}{(2.54 \times 0.016 \times 2.595)}$$
$$\approx 1.8°C/W.$$

The base plate has a thermal resistance of approximately 7.5°C/W. Therefore,

$$\theta_{TOTAL} = \theta_{JC} + \theta_{CS} + (\theta_{HOLE} \parallel \theta_{HOLE} \parallel \theta_{PCB}) + \theta_{SP} + \theta_{PEM} + \theta_{BASE}$$
$$= 4 + 1 + (14.5 \parallel 14.5 \parallel 62) + 1 + 1.8 + 7.5$$
$$= 4 + 1 + 6.5 + 1.8 + 7.5$$
$$= 21°C/W.$$

At an ambient temperature of 50°C, the maximum power dissipation is

$$P_{MAX} = (125 - 50) / 21 \cong 3.57 \text{ W}.$$

With V_{IN} equal to 13 V, the maximum current draw is

$$I = 3.57 / (13 - 5) \cong 446 \text{ mA}.$$

Heat Dissipation without the Base Plate

If the BL1400 is to be mounted on a heat-conducting surface, Z-World recommends the following mounting scheme. Use a 6-32 steel screw to hold the regulator to a ¼" 6-32 aluminum standoff mounted on a large heat sink. A conservative estimate of the thermal resistance is as follows.

The thermal resistance across a ¼-6-32 aluminum standoff of ¼" height is

$$\theta_{AL} = \frac{0.25}{2.54 \times 0.035 \times 2.595} = 1°\text{C/W}.$$

Assuming a basis of 5 °C/W, the total thermal resistance becomes

$$\begin{aligned}\theta_{TOTAL2} &= \theta_{JC} + \theta_{CS} + \left(\theta_{HOLE} \parallel \theta_{HOLE} \parallel \theta_{PCB}\right) + \theta_{SP} + \theta_{PEM} + \theta_{BASE} \\ &= 4 + 1 + \left(14.5 \parallel 14.5 \parallel 62\right) + 1 + 1 + 5 \\ &= 4 + 1 + 6.5 + 1 + 5 \\ &= 17.5°\text{C/W}.\end{aligned}$$

At an ambient temperature of 50°C, the maximum power dissipation is

$$P_{MAX} = (125 - 50) / 17.5 = 4.28 \text{ W}.$$

With V_{IN} equal to 13 V, the maximum current draw is

$$I = 4.28 / (13 - 5) = 535 \text{ mA}.$$

APPENDIX D: PROTOTYPING BOARD

Appendix D describes the BL1400 Prototyping Board.

Prototyping Board

The BL1400 Prototyping Board was designed to allow the user to experiment with the BL1400 and to prototype small circuits that can "piggyback" on the BL1400. The Prototyping Board has several pre-built subcircuits that can be tested or used as part of a custom design. An array of uncommitted pads facilitates prototyping. Power rails bring +5 V and GND to these pads.

The prototyping board has the following features:

- **Direct Header Connections**

Header H1 on the bottom side of the Prototyping Board connects directly to header H3 of the BL1400.

- **Array of Pads**

An array of pads for dual in-line packages (DIPs) and discrete components is provided. Pads and power rails are arranged and interconnected so that it is easy to place and connect 300-mil or 600-mil DIPs.

- **Extra Pads**

Extra pads bring out signals from header H1 oo the Prototyping Board for easier soldering.

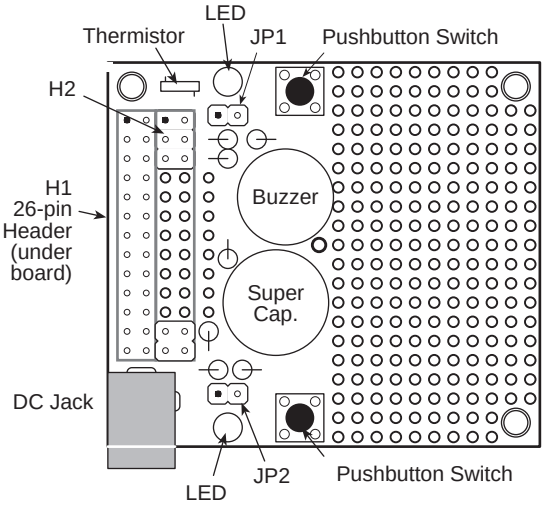


Figure D-1. BL1400 Prototyping Board

The following list notes some important points about the BL1400:

- The prototyping board measures 2.0″ × 2.25″ (51 mm × 57 mm).
- The mounting holes are 0.125″ in diameter. Their centers are inset 0.125″ from the edges of the board.
- Signals on header H1 are identical to those on header H3 of BL1400.
- Power (9 V to 12 V DC) comes in the jack, through pin 25 of H1, then to the 5 V regulator on the BL1400, and back to the Prototyping Board on pin 1 (+5 V) of H1.
- H2 on the Prototyping Board is nearly identical to H1, but most of the signal positions on H2 are pads for easy solder connections. The remaining positions on H2 are jumper pins.
- A set of nine "test points" or pads (shown in Figure D-2) make signals from the demonstration circuits available.

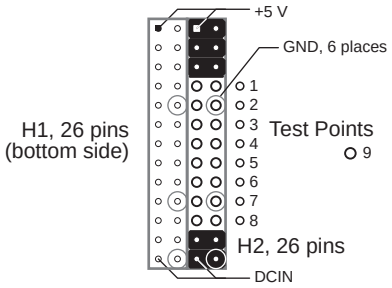


Figure D-2. Prototyping Board Signals

- Power rails bring +5 V and GND to the array of pads (see Figure D-3).

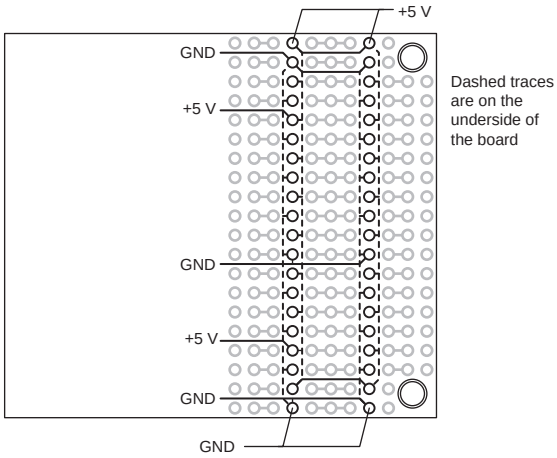


Figure D-3. Prototyping Board+5 V and GND Connections

- The array is 11×20 overall, with pads on 0.1″ centers.
- Certain adjacent pads are connected, making it easy to place 300-mil (or 600-mil) DIPs and solder wires to nearby pads, rather than to the pins of the DIPs.

Installing the Prototyping Board

Connect the prototyping board to the BL1400 by pressing the header block (H1) underneath the Prototyping Board onto header H3 of the BL1400. Observe the correct orientation for the prototyping board: when installed, the main body of the prototyping board extends out away from the BL1400 (see Figure D-4). Pin 1 of H1 must connect with pin 1 of H3 on the BL1400. Otherwise, both boards will be damaged.

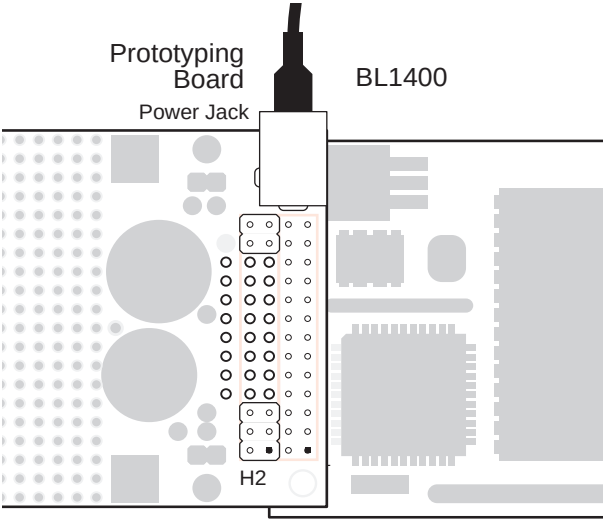


Figure D-4. Connecting the Prototyping Board to BL1400

When the Prototyping Board is installed, power for software development normally comes through the DC input jack. In the absence of a prototyping board (forexample, when system development has been completed), power for the BL1400 comes through H3, pin 25, of the BL1400.

Sample Circuits

Several demonstration circuits are included on the Prototyping Board.

LEDs

LEDs D1 and D2 are turned on when their respective test pads (TP8, TP1) are driven with a low logic level.

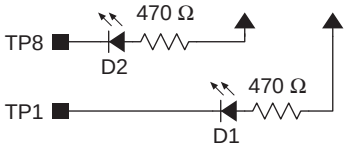


Figure D-5. Prototyping Board LEDs

Switches

Pushbutton switches SW1 and SW2 are tied to pull-up resistors. Their respective test pads (TP3, TP6) are pulled to ground when the switches are pressed.

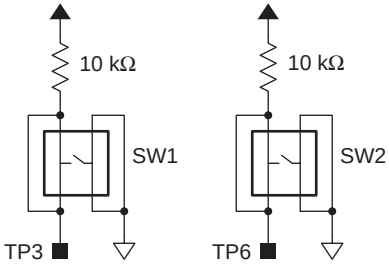


Figure D-6. Prototyping Board Switches

Jumpers

Jumpers J1 and J2 are tied to pull-up resistors. Their respective test pads (TP2, TP7) are pulled to ground when connectors are installed.

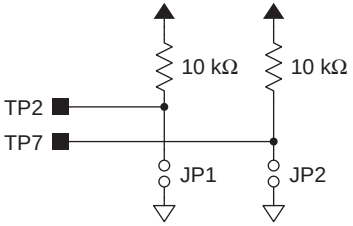


Figure D-7. Prototyping Board Jumpers

Buzzer

The buzzer, BZ1, is turned on when TP4 is brought to ground level.

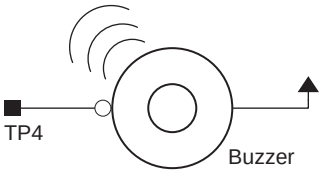


Figure D-8. Prototyping Board Buzzer

RC Filter

The RC low-pass filter consists of a 10 kΩ series resistor and a 4.7 µF capacitor to ground. A variable analog voltage is developed at TP9 by driving TP5 with a pulse-width modulated (PWM) signal.

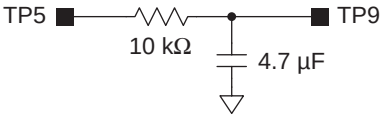


Figure D-9. Prototyping Board RC Filter

Thermistor

A thermistor is hardwired on the Prototyping Board. It is accessible through line EXTRES on the BL1400, and by using functions that read the 555 timer.

See Chapter 4, “Software Reference,” for more information.

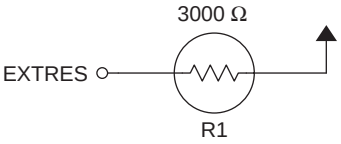


Figure D-10. Sample Thermistor Circuit

The "Using Thermistors" section in Chapter 3, "Interface Overview," discusses the resistance equations and temperature coefficient of a hypothetical thermistor. The thermistor used on the Prototyping Board has a resistance of 3000 Ω at 25°C and a temperature coefficient of -4.49% per degree Celsius. (This is Keystone part RL2007-1723-103-SA, or Digikey part KC004E-ND.)

For $\alpha = -4.49\%$, $ß = -(298.15 \times 298.15) \times -0.0449 = 3991$.

The resistance at 25°C is 3000 Ω, and so

$$3000 = A\,e^{3991/298.15}$$
$$= A \times 650{,}740$$

or

$$A = \frac{3000}{650{,}740} = 0.00461 \quad .$$

Therefore,

$$\ln A = -5.379$$

and so

$$T = \frac{\beta}{(\ln R - \ln A)}$$
$$= \frac{3991}{(\ln R + 5.379)} \quad .$$

APPENDIX E: PLCBUS

Appendix E provides the pin assignments for the PLCBus, describes the registers, and lists the software drivers.

PLCBus Overview

The PLCBus is a general-purpose expansion bus for Z-World controllers. The PLCBus is available on the BL1200, BL1600, BL1700, PK2100, and PK2200 controllers. The BL1000, BL1100, BL1300, BL1400, and BL1500 controllers support the XP8300, XP8400, XP8600, and XP8900 expansion boards using the controller's parallel input/output port. The BL1400 and BL1500 also support the XP8200 and XP8500 expansion boards. The ZB4100's PLCBus supports most expansion boards, except for the XP8700 and the XP8800. The SE1100 adds expansion capability to boards with or without a PLCBus interface.

Table E-1 lists Z-World's expansion devices that are supported on the PLCBus.

Table E-1. Z-World PLCBus Expansion Devices

Device	Description
EXP-A/D12	Eight channels of 12-bit A/D converters
SE1100	Four SPDT relays for use with all Z-World controllers
XP8100 Series	32 digital inputs/outputs
XP8200	"Universal Input/Output Board" —16 universal inputs, 6 high-current digital outputs
XP8300	Two high-power SPDT and four high-power SPST relays
XP8400	Eight low-power SPST DIP relays
XP8500	11 channels of 12-bit A/D converters
XP8600	Two channels of 12-bit D/A converters
XP8700	One full-duplex asynchronous RS-232 port
XP8800	One-axis stepper motor control
XP8900	Eight channels of 12-bit D/A converters

Multiple expansion boards may be linked together and connected to a Z-World controller to form an extended system.

Figure E-1 shows the pin layout for the PLCBus connector.

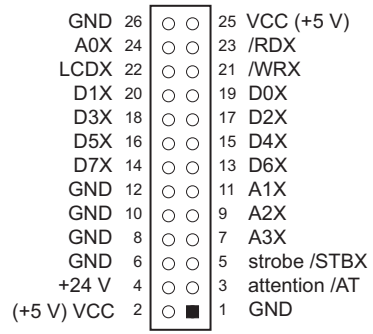


Figure E-1. PLCBus Pin Diagram

Two independent buses, the LCD bus and the PLCBus, exist on the single connector.

The LCD bus consists of the following lines.

- LCDX—positive-going strobe.
- /RDX—negative-going strobe for read.
- /WRX—negative-going strobe for write.
- A0X—address line for LCD register selection.
- D0X-D7X—bidirectional data lines (shared with expansion bus).

The LCD bus is used to connect Z-World's OP6000 series interfaces or to drive certain small liquid crystal displays directly. Figure A-2 illustrates the connection of an OP6000 interface to a BL1200 controller.

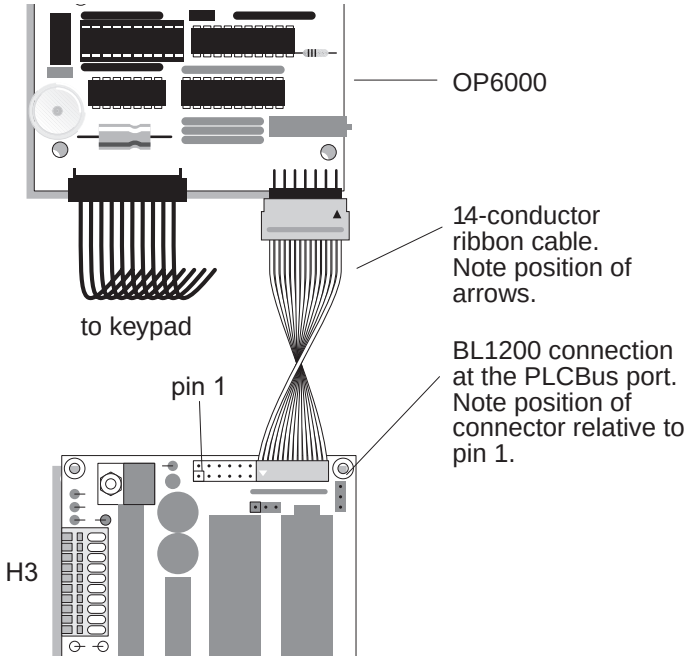


Figure E-2. OP6000 Connection to BL1200 PLCBus Port

The PLCBus consists of the following lines.

- /STBX—negative-going strobe.
- A1X–A3X—three control lines for selecting bus operation.
- D0X–D3X—four bidirectional data lines used for 4-bit operations.
- D4X–D7X—four additional data lines for 8-bit operations.
- /AT—attention line (open drain) that may be pulled low by any device, causing an interrupt.

The PLCBus may be used as a 4-bit bus (D0X–D3X) or as an 8-bit bus (D0X–D7X). Whether it is used as a 4-bit bus or an 8-bit bus depends on the encoding of the address placed on the bus. Some PLCBus expansion cards require 4-bit addressing and others (such as the XP8700) require 8-bit addressing. These devices may be mixed on a single bus.

There are eight registers corresponding to the modes determined by bus lines A1X, A2X, and A3X. The registers are listed in Table E-2.

Table E-2. PLCBus Registers

Register	Address	A3	A2	A1	Meaning
BUSRD0	C0	0	0	0	Read data, one way
BUSRD1	C2	0	0	1	Read data, another way
BUSRD2	C4	0	1	0	Spare, or read data
BUSRESET	C6	0	1	1	Read this register to reset the PLCBus
BUSADR0	C8	1	0	0	First address nibble or byte
BUSADR1	CA	1	0	1	Second address nibble or byte
BUSADR2	CC	1	1	0	Third address nibble or byte
BUSWR	CE	1	1	1	Write data

Writing or reading one of these registers takes care of all the bus details. Functions are available in Z-World's software libraries to read from or write to expansion bus devices.

To communicate with a device on the expansion bus, first select a register associated with the device. Then read or write from/to the register. The register is selected by placing its address on the bus. Each device recognizes its own address and latches itself internally.

A typical device has three internal latches corresponding to the three address bytes. The first is latched when a matching BUSADR0 is detected. The second is latched when the first is latched and a matching BUSADR1 is detected. The third is latched if the first two are latched and a matching BUSADR2 is detected. If 4-bit addressing is used, then there are three 4-bit address nibbles, giving 12-bit addresses. In addition, a special register address is reserved for address expansion. This address, if ever used, would provide an additional four bits of addressing when using the 4-bit convention.

If eight data lines are used, then the addressing possibilities of the bus become much greater—more than 256 million addresses according to the conventions established for the bus.

Place an address on the bus by writing (bytes) to BUSADR0, BUSADR1 and BUSADR2 in succession. Since 4-bit and 8-bit addressing modes must coexist, the lower four bits of the first address byte (written to BUSADR0) identify addressing categories, and distinguish 4-bit and 8-bit modes from each other.

There are 16 address categories, as listed in Table E-3. An "x" indicates that the address bit may be a "1" or a "0."

Table E-3. First-Level PLCBus Address Coding

First Byte	Mode	Addresses	Full Address Encoding
– – – – 0 0 0 0	4 bits × 3	256	0000 xxxx xxxx
– – – – 0 0 0 1		256	0001 xxxx xxxx
– – – – 0 0 1 0		256	0010 xxxx xxxx
– – – – 0 0 1 1		256	0011 xxxx xxxx
– – – x 0 1 0 0	5 bits × 3	2,048	x0100 xxxxx xxxxx
– – – x 0 1 0 1		2,048	x0101 xxxxx xxxxx
– – – x 0 1 1 0		2,048	x0110 xxxxx xxxxx
– – – x 0 1 1 1		2,048	x0111 xxxxx xxxxx
– – x x 1 0 0 0	6 bits × 3	16,384	xx1000 xxxxxx xxxxxx
– – x x 1 0 0 1		16,384	xx1001 xxxxxx xxxxxx
– – x x 1 0 1 0	6 bits × 1	4	xx1010
– – – – 1 0 1 1	4 bits × 1	1	1011 (expansion register)
x x x x 1 1 0 0	8 bits × 2	4,096	xxxx1100 xxxxxxxx
x x x x 1 1 0 1	8 bits × 3	1M	xxxx1101 xxxxxxxx xxxxxxxx
x x x x 1 1 1 0	8 bits × 1	16	xxxx1110
x x x x 1 1 1 1	8 bits × 1	16	xxxx1111

This scheme uses less than the full addressing space. The mode notation indicates how many bus address cycles must take place and how many bits are placed on the bus during each cycle. For example, the 5 × 3 mode means three bus cycles with five address bits each time to yield 15-bit addresses, not 24-bit addresses, since the bus uses only the lower five bits of the three address bytes.

Z-World provides software drivers that access the PLCBus. To allow access to bus devices in a multiprocessing environment, the expansion register and the address registers are shadowed with memory locations known as *shadow registers*. The 4-byte shadow registers, which are saved at predefined memory addresses, are as follows.

SHBUS0	SHBUS0+1	SHBUS1 SHBUS0+2	SHBUS1+1 SHBUS0+3
Bus expansion	BUSADR0	BUSADR1	BUSADR2

Before the new addresses or expansion register values are output to the bus, their values are stored in the shadow registers. All interrupts that use the bus save the four shadow registers on the stack. Then, when exiting the interrupt routine, they restore the shadow registers and output the three address registers and the expansion registers to the bus. This allows an interrupt routine to access the bus without disturbing the activity of a background routine that also accesses the bus.

To work reliably, bus devices must be designed according to the following rules.

1. The device must not rely on critical timing such as a minimum delay between two successive register accesses.
2. The device must be capable of being selected and deselected without adversely affecting the internal operation of the controller.

Allocation of Devices on the Bus

4-Bit Devices

Table E-4 provides the address allocations for the registers of 4-bit devices.

Table E-4. Allocation of Registers

A1	A2	A3	Meaning
000j	000j	xxxj	digital output registers, 64 registers 64 × 8 = 512 1-bit registers
000j	001j	xxxj	analog output modules, 64 registers
000j	01xj	xxxj	digital input registers, 128 registers 128 × 4 = 512 input bits
000j	10xj	xxxj	analog input modules, 128 registers
000j	11xj	xxxj	128 spare registers (customer)
001j	xxxj	xxxj	512 spare registers (Z-World)

j controlled by board jumper
x controlled by PAL

Digital output devices, such as relay drivers, should be addressed with three 4-bit addresses followed by a 4-bit data write to the control register. The control registers are configured as follows

bit 3	bit 2	bit 1	bit 0
A2	A1	A0	D

The three address lines determine which output bit is to be written. The output is set as either 1 or 0, according to D. If the device exists on the bus, reading the register drives bit 0 low. Otherwise bit 0 is a 1.

For digital input, each register (BUSRD0) returns four bits. The read register, BUSRD1, drives bit 0 low if the device exists on the bus.

8-Bit Devices

Z-World's XP8700 and XP8800 expansion boards use 8-bit addressing. Refer to the ***XP8700 and XP8800*** manual.

Expansion Bus Software

The expansion bus provides a convenient way to interface Z-World's controllers with expansion boards or other specially designed boards. The expansion bus may be accessed by using input functions. Follow the suggested protocol. The software drivers are easier to use, but are less efficient in some cases. Table E-5 lists the libraries.

Table E-5. Dynamic C PLCBus Libraries

Library Needed	Controller
DRIVERS.LIB	All controllers
EZIOTGPL.LIB	BL1000
EZIOLGPL.LIB	BL1100
EZIOMGPL.LIB	BL1400, BL1500
EZIOPLC.LIB	BL1200, BL1600, PK2100, PK2200, ZB4100
EZIOPLC2.LIB	BL1700
EZIOBL17.LIB	BL1700
PBUS_TG.LIB	BL1000
PBUS_LG.LIB	BL1100, BL1300
PLC_EXP.LIB	BL1200, BL1600, PK2100, PK2200

There are 4-bit and 8-bit drivers. The 4-bit drivers employ the following calls.

- **`void eioResetPlcBus()`**

  Resets all expansion boards on the PLCBus. When using this call, make sure there is sufficient delay between this call and the first access to an expansion board.

  LIBRARY: **`EZIOPLC.LIB`**, **`EZIOPLC2.LIB`**, **`EZIOMGPL.LIB`**.

- **`void eioPlcAdr12( unsigned addr )`**

  Specifies the address to be written to the PLCBus using cycles BUSADR0, BUSADR1, and BUSADR2.

  PARAMETER: **`addr`** is broken into three nibbles, and one nibble is written in each BUSADR*x* cycle.

  LIBRARY: **`EZIOPLC.LIB`**, **`EZIOPLC2.LIB`**, **`EZIOMGPL.LIB`**.

- **`void set16adr( int adr )`**

  Sets the current address for the PLCBus. All read and write operations access this address until a new address is set.

  PARAMETER: **`adr`** is a 16-bit physical address. The high-order nibble contains the value for the expansion register, and the remaining three 4-bit nibbles form a 12-bit address (the first and last nibbles must be swapped).

  LIBRARY: **`DRIVERS.LIB`**.

- **`void set12adr( int adr )`**

  Sets the current address for the PLCBus. All read and write operations access this address until a new address is set.

  PARAMETER: **`adr`** is a 12-bit physical address (three 4-bit nibbles) with the first and third nibbles swapped.

  LIBRARY: **`DRIVERS.LIB`**.

- **`void eioPlcAdr4( unsigned addr )`**

  Specifies the address to be written to the PLCBus using only cycle BUSADR2.

  PARAMETER: **`addr`** is the nibble corresponding to BUSADR2.

  LIBRARY: **`EZIOPLC.LIB`**, **`EZIOPLC2.LIB`**, **`EZIOMGPL.LIB`**.

- **`void set4adr( int adr )`**

  Sets the current address for the PLCBus. All read and write operations access this address until a new address is set.

  A 12-bit address may be passed to this function, but only the last four bits will be set. Call this function only if the first eight bits of the address are the same as the address in the previous call to set12adr.

  PARAMETER: **`adr`** contains the last four bits (bits 8–11) of the physical address.

  LIBRARY: **`DRIVERS.LIB`**.

- **`char _eioReadD0( )`**

  Reads the data on the PLCBus in the BUSADR0 cycle.

  RETURN VALUE: the byte read on the PLCBus in the BUSADR0 cycle.

  LIBRARY: **`EZIOPLC.LIB`**, **`EZIOPLC2.LIB`**, **`EZIOMGPL.LIB`**.

- **`char _eioReadD1( )`**

  Reads the data on the PLCBus in the BUSADR1 cycle.

  RETURN VALUE: the byte read on the PLCBus in the BUSADR1 cycle.

  LIBRARY: **`EZIOPLC.LIB`**, **`EZIOPLC2.LIB`**, **`EZIOMGPL.LIB`**.

- **`char _eioReadD2( )`**

  Reads the data on the PLCBus in the BUSADR2 cycle.

  RETURN VALUE: the byte read on the PLCBus in the BUSADR2 cycle.

  LIBRARY: **`EZIOPLC.LIB`**, **`EZIOPLC2.LIB`**, **`EZIOMGPL.LIB`**.

- **`char read12data( int adr )`**

  Sets the current PLCBus address using the 12-bit **`adr`**, then reads four bits of data from the PLCBus with BUSADR0 cycle.

  RETURN VALUE: PLCBus data in the lower four bits; the upper bits are undefined.

  LIBRARY: **`DRIVERS.LIB`**.

- **`char read4data( int adr )`**

  Sets the last four bits of the current PLCBus address using adr bits 8–11, then reads four bits of data from the bus with BUSADR0 cycle.

  PARAMETER: adr bits 8–11 specifies the address to read.

  RETURN VALUE: PLCBus data in the lower four bits; the upper bits are undefined.

  LIBRARY: **`DRIVERS.LIB`**.

- **`void _eioWriteWR( char ch)`**

  Writes information to the PLCBus during the BUSWR cycle.

  PARAMETER: ch is the character to be written to the PLCBus.

  LIBRARY: **`EZIOPLC.LIB`**, **`EZIOPLC2.LIB`**, **`EZIOMGPL.LIB`**.

- **`void write12data( int adr, char dat )`**

  Sets the current PLCBus address, then writes four bits of data to the PLCBus.

  PARAMETER: **`adr`** is the 12-bit address to which the PLCBus is set.

  `dat` (bits 0–3) specifies the data to write to the PLCBus.

  LIBRARY: **`DRIVERS.LIB`**.

- **`void write4data( int address, char data )`**

  Sets the last four bits of the current PLCBus address, then writes four bits of data to the PLCBus.

  PARAMETER: **`adr`** contains the last four bits of the physical address (bits 8–11).

  `dat` (bits 0–3) specifies the data to write to the PLCBus.

  LIBRARY: **`DRIVERS.LIB`**.

The 8-bit drivers employ the following calls.

- **`void set24adr( long address )`**

  Sets a 24-bit address (three 8-bit nibbles) on the PLCBus. All read and write operations will access this address until a new address is set.

  PARAMETER: **`address`** is a 24-bit physical address (for 8-bit bus) with the first and third bytes swapped (low byte most significant).

  LIBRARY: **`DRIVERS.LIB`**.

- **`void set8adr( long address )`**

  Sets the current address on the PLCBus. All read and write operations will access this address until a new address is set.

  PARAMETER: **`address`** contains the last eight bits of the physical address in bits 16–23. A 24-bit address may be passed to this function, but only the last eight bits will be set. Call this function only if the first 16 bits of the address are the same as the address in the previous call to **`set24adr`**.

  LIBRARY: **`DRIVERS.LIB`**.

- **`int read24data0( long address )`**

  Sets the current PLCBus address using the 24-bit address, then reads eight bits of data from the PLCBus with a BUSRD0 cycle.

  RETURN VALUE: PLCBus data in lower eight bits (upper bits 0).

  LIBRARY: **`DRIVERS.LIB`**.

- **`int read8data0( long address )`**

  Sets the last eight bits of the current PLCBus address using address bits 16–23, then reads eight bits of data from the PLCBus with a BUSRD0 cycle.

  PARAMETER: **`address`** bits 16–23 are read.

  RETURN VALUE: PLCBus data in lower eight bits (upper bits 0).

  LIBRARY: **`DRIVERS.LIB`**.

- **`void write24data( long address, char data )`**

  Sets the current PLCBus address using the 24-bit address, then writes eight bits of data to the PLCBus.

  PARAMETERS: **`address`** is 24-bit address to write to.

  `data` is data to write to the PLCBus.

  LIBRARY: **`DRIVERS.LIB`**.

- **`void write8data( long address, char data )`**

  Sets the last eight bits of the current PLCBus address using address bits 16–23, then writes eight bits of data to the PLCBus.

  PARAMETERS: **`address`** bits 16–23 are the address of the PLCBus to write.

  `data` is data to write to the PLCBus.

  LIBRARY: **`DRIVERS.LIB`**.

APPENDIX F:
SIMULATED PLCBUS CONNECTION

PIO Port Connections

Expansion Boards

Expansion boards may be connected to header H3 on the BL1400. To add expansion boards, the user must either make a custom cable or use an adapter board (Z-World part number 101-0050). To assist with making the connection via a custom-made ribbon cable, Table F-1 maps the signals from the controller's PIO to the expansion boards PLCBus header. Dynamic C's **`EZIOMGPL.LIB`** (for XP8900 Series expansion Boards) or **`BL14_15.LIB`** library may be used for programming.

Table F-1. PIO to PLCBus Signal Map

BL1400		Expansion Board	
H3 Pin No.	**PIO Port Signal**	**Pin No.**	**PLCBus Signal**
1	VCC (+5 V)	2	VCC (+5 V)
2	PA0	5	/STBX
3	PA1	19	D0X
4	PA2	20	D1X
5	PA3	17	D2X
6	PA4	18	D3X
7	PA5	11	A1X
8	PA6	9	A2X
9	PA7	7	A3X
10	GND	10	GND

The adapter board (Figure F-1) provides an easy way to add an expansion board to BL1400 series controllers. Power is supplied to the controller via the power jack and to the expansion board via a screw terminal. The expansion boards receive +12 V or +24 V from header J5 as shown when pins 1–2 on the adapter board's header J6 are not connected. Connect pins 1–2 on the adapter board's header J6 to power the expansion boards from DCIN, the BL1400 power supply.

Do not apply power to header J5 on the adapter board when pins 1–2 on the adpater board's header J6 are connected.

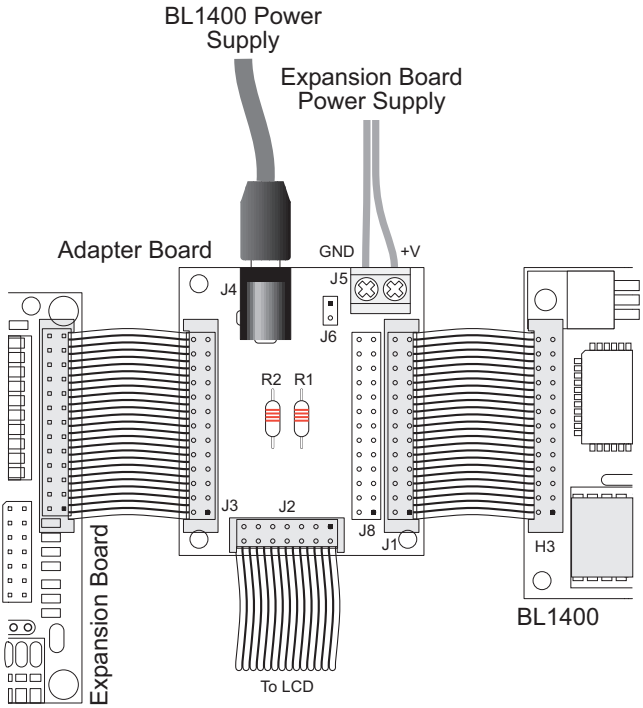


Figure F-1. Adapter Board Connections

Liquid Crystal Displays and Keypads

Liquid crystal displays (LCDs) can also be connected to the BL1400's PIO port on header H3 using the adapter board. The LCD must be hardwired for writing only. Table F-2 lists the wiring connections. for LCDs.

Only the LCD's EN line is exclusive to the LCD. RS and the other data lines can be shared with other devices. The LCD busy flag is never checked. Instead, a 40 µs software delay ensures that the LCD is not busy before writing to the LCD.

Table F-3 lists the wiring connections for keypads.

If either J2 or J3 on the adapter board is used, some of the signals on J8 are inaccesible. J8's RS-485 and RMI signals remain available.

Table F-2. PIO to LCD Signal Map

BL1400		2 × 20 LCD	
H3 Pin No.	**PIO Port Signal**	**Pin No.**	**Signal**
1	VCC (+5 V)	2	VDD
4	PA2	4	RS
5	PA3	6	EN
—	—	7	DB0
—	—	8	DB1
—	—	9	DB2
—	—	10	DB3
6	PA4	11	DB4
7	PA5	12	DB5
8	PA6	13	DB6
9	PA7	14	DB7
10	GND	3	VO*
26	GND	5	RD/WR
		1	VSS

* To improve contrast, connect VO to +0.3 V by adding a resistor between V0 and GND instead of connecting VO to GND.

Table F-3. PIO to Keypad Signal Map

BL1400		Keypad Signal
H3 Pin No.	**PIO Port Signal**	
2	PA0	KeyRead 1
3	PA1	KeyRead 2
6	PA4	DriveLine 1
7	PA5	DriveLine 2
8	PA6	DriveLine 3
9	PA7	DriveLine 4
13	PB7	KeyRead 6
14	PB6	KeyRead 5
15	PB5	KeyRead 4
16	PB4	KeyRead 3

Software Drivers

Using Expansion Boards with PIO Port A

A PLCBus driver is implemented using the 8-bit PIO Port A (PIOA). With the BL1400, the developer is limited to 4-bit PLCBus peripherals. An attention line (/AT) is not available. The wiring connections shown in Table F-1 are used.

For multiply threaded programs, be sure to save and to restore the state of the PLCBus when entering a thread that can preempt other threads that also use the PLCBus. The function **`mgsave_pbus`** saves PLCBus state to the stack and **`mgrestore_pbus`** restores it from the stack.

- **`void mgset12adr( int addr )`**

  Sets the current address of the PLCBus. A subsequent read or write of the PLCBus will access the expansion board with this address. The address remains in effect until a new address is set. The term **`addr`** is the 12-bit physical address of the PLCBus expansion board. The lowest 4-bit nibble is transmitted last (as BUSADR2). The third nibble is transmitted first (as BUSADR0).

- **`void mgset12data( int addr, int data )`**

  Writes data to the expansion board at **`addr`**. Only the lowest four bits of **`data`** are useable (for BUSWR).

- **`int mgread12data0( int addr )`**

  Reads data (with BUSRD0) from the expansion board at **`addr`**. The function result holds the data.

- **`int mgread12data1( int addr )`**

  Reads data (with BUSRD1) from the expansion board at **`addr`**. The function result holds the data.

- **`int mgread12data2( int addr )`**

  Reads data (with BUSRD2) from the expansion board at **`addr`**. The function result holds the data.

- **`void mgwrite4data( int value )`**

  Writes the low 4 bits of **`value`** (with BUSWR) to an expansion board. This function assumes that the expansion board's address has been placed on the PLCBus (with **`mgset12adr`**).

- **`void mgreset_pbus()`**

  Initializes PIO Port A to communicate with expansion boards. The function also resets all expansion boards.

- **`int mgplc_poll_node( int addr )`**

  Polls for the presence of an expansion board with the given board address. The function returns 1 if the board is found and 0 if the board is not found.

- **`void mgsave_pbus()`**

  Saves the current state of the PLCBus to the stack. This function should only be called in tandem with **`mgrestore_pbus`**. Otherwise, the stack will become unbalanced.

- **`void mgrestore_pbus()`**

  Restores the current state of the PLCBus from the stack. This function should only be called in tandem with **`mgsave_pbus`**. Otherwise, the stack will become unbalanced.

- **`void mgplc_set_relay( int number, int relay, int state )`**

  Turns a relay on an expansion board on or off. The relay board must be a Z-World XP8300 or XP8400 expansion board and its **`number`** must be from 0 to 63. The term **`relay`** selects the relay on the selected board (0–5 for XP8300 boards and 0–7 for XP8400 boards. The term **`state`** is 1 to turn on the relay board and 0 to turn it off.

Refer to the ***XP8300, XP8400 and SE1100 User's Manual*** for details regarding devices and device numbering schemes.

- **`int mgplcrly_board( int number )`**

  Computes the physical address of a relay board from its board **`number`**. The number must be from 0 to 63. (Board number 0 corresponds to address 0x000; board number 63 corresponds to address 0x11F.) The return value has the third and the first nibbles interchanged.

Refer to the ***XP8300, XP8400 and SE1100 User's Manual*** for details regarding devices and device numbering schemes.

- **`int mgplcuio_board( int number )`**

  Computes the physical address of an XP8200 expansion board from its board **number**. The number must be from 0 to 15. (Board number 0 corresponds to address 0x040. Board number 15 corresponds to address 0x04F.) The return value has the third and the first nibbles interchanged.

Refer to the ***XP8100 and XP8200 User's Manual*** for details regarding devices and device numbering schemes.

- **`int mgplc_dac_board( int number )`**

  Computes the physical address of an XP8600 or XP8900 expansion board from its board **`number`**. The number must be from 0 to 63. (Board number 0 corresponds to address 0x020; board number 63 corresponds to address 0x13F.) The return value has the third and the first nibbles interchanged.

  Refer to the ***XP8600 and XP8900 User's Manual*** for details regarding devices and device numbering schemes.

- **`void mginit_dac()`**

  Initializes an XP8600 or XP8900 expansion board on the PLCBus. This function assumes that the board's address has been placed on the bus (with **`mgset12adr`**).

- **`void mgwrite_dac1( int value )`**

  Writes the 12-bit integer **`value`** to Register A of DAC 1 of an XP8600 or XP8900 expansion board on the PLCBus. This function assumes that the board's address has been placed on the bus (with **`mgset12adr`**). The an XP8600 or XP8900 expansion board does not produce a new conversion value until a call to **`mglatch_dac1`** is executed.

- **`void mglatch_dac1()`**

  Moves Register A data to Register B of DAC 1 of an XP8600 or XP8900 expansion board on the PLCBus. The actual DAC 1 output is converted from Register B. This function assumes that the board's address has been placed on the bus (with **`mgset12adr`**). Be sure that Register A contains valid data. See **`mgwrite_dac1`** above.

- **`void mgset_dac1( int value )`**

  Writes a 12-bit integer **`value`** to Register A, then moves the data from Register A to Register B of DAC 1 of a selected XP8600 or XP8900 expansion board. This function assumes that the board's address has been placed on the bus (with **`mgset12adr`**). It combines the effect of **`mgwrite_dac1`** and **`mglatch_dac1`**.

- **`void mgwrite_dac2( int value )`**

  Writes the 12-bit integer **`value`** to Register A of DAC 2 of an XP8600 or XP8900 expansion board on the PLCBus. This function assumes that the board's address has been placed on the bus (with **`mgset12adr`**). The XP8600 or XP8900 expansion board does not produce a new conversion value until a call to **`mglatch_dac2`** is executed.

- **`void mglatch_dac2()`**

  Moves Register A data to Register B of DAC 2 of an XP8600 or XP8900 expansion board on the PLCBus. The actual DAC 2 output is converted from Register B. This function assumes that the board's address has been placed on the bus (with **`mgset12adr`**). Be sure that Register A contains valid data. See **`mgwrite_dac2`** above.

- **`void mgset_dac2( int value )`**

  Writes a 12-bit integer **`value`** to Register A, then moves the data from Register A to Register B of DAC 2 of a selected XP8600 or XP8900 expansion board. This function assumes that the board's address has been placed on the bus (with **`mgset12adr`**). It combines the effect of **`mgwrite_dac2`** and **`mglatch_dac2`**.

Using an LCD with PIO Port A

A 4-bit LCD (liquid crystal display) driver is implemented through six bits of PIO Port A.

The LCD drivers are compatible with the keypad drivers in the following section. That is, both an LCD and a keypad can be driven with this software and a BL1400.

- **`void lc_char( byte data )`**

  Writes a character to the LCD.

- **`void lc_ctrl( byte cmd )`**

  Writes a control command to the LCD.

- **`void lc_init()`**

  Initialize the LCD and accessory variables. The LCD uses PIO Port A.

- **`void lc_cgram( void* ptr )`**

  Loads up to eight special characters to the character generator of the LCD from the byte array **`*p`**. The first byte in the array is the number of bytes to store (at 8 bytes per character), with a maximum value of 64 for 8 characters. The character codes for the special characters are 0, 1, 2, 3, 4, 5, 6, and 7.

- **`void lc_printf( char* format, ... )`**

  This function works like **`printf`**, but for the LCD. The escape sequences shown in Table F-4 are also implemented.

  The escape character code is **`0x1B`**.

Table F-4. LCD **printf** *Escape Sequences*

Command	Result
<ESC> 1	Turn cursor on
<ESC> 0	Turn cursor off
<ESC> c	Erase from cursor to end of line
<ESC> b	Enable cursor blinking
<ESC> n	Disable cursor blinking
<ESC> e	Erase display and home cursor
<ESC> p n mm	Position cursor at line n, column mm

Using a Keypad with PIO Ports A and B

A keypad driver is implemented using PIO Ports A and B of the BL1400.

The keys are scanned by forcing the driver lines low, one at a time, and sensing the keyread lines. A low level on a keyread line indicates a keypress. Call the function **lc_keyscan** periodically to scan the keypad.

"Debouncing" is done by making sure a key is pressed for **DebounceCount** consecutive calls to **lc_keyscan**. The debouncing number can be changed by redefining **DebounceCount**:

```
#define DebounceCount nn
```

If not redefined, **DebounceCount** defaults to 20. If **lc_keyscan** is called every 25 ms and **DebounceCount** is 20, then a key has to be pressed for 20 × 25 ms = 500 ms to be valid.

The keypad drivers are compatible with the preceding LCD drivers. That is, both an LCD and a keypad can be driven with this software on a BL1400.

- **lc_kxinit()**

  Initializes the keypad driver and accessory variables. Define **KEY4x6** somewhere at the start of the code if using a 4 × 6 keypad.

  ```
  #define KEY4x6
  ```

  Otherwise, the driver defaults to a 2×6 keypad.

- **int lc_kxget(int mode)**

  Obtains the key value from the FIFO keypad buffer. If **mode** = 0, the key value is removed from the buffer. Otherwise, the key value is left in the buffer. In either case, the function returns the key value, or –1 if the keypad buffer is empty.

- **`void lc_keyscan()`**

  Scans the 4 × 6 or 2 × 6 keypad. A valid key has to be persistent for **`DebounceCount`** calls to **`lc_keyscan`**. The function puts valid keypresses into the keypad FIFO buffer. The software will access these keypresses using **`lc_kxget`**.

For more information or assistance, call a Z-World Technical Support Representative at (530) 757-3737.

APPENDIX G: INPUT/OUTPUT MAPS AND INTERRUPT VECTORS

Appendix G provides information on the BL1400's input/output maps and interrupt vectors.

Memory Map

Internal Input/Output Registers

The internal registers for the I/O devices built into to the Z180 processor occupy the first 40 (hex) addresses of the I/O space. Table G-1 lists addresses 00-3F of the Z180's internal I/O registers.

Table G-1. Z180 Addresses 00-3F

Address	Name	Description
00	CNTLA0	Control Register A, Serial Channel 0
01	CNTLA1	Control Register A, Serial Channel 1
02	CNTLB0	Control Register B, Serial Channel 0
03	CNTLB1	Control Register B, Serial Channel 1
04	STAT0	Status Register, Serial Channel 0
05	STAT1	Status Register, Serial Channel 1
06	TDR0	Transmit Data Register, Serial Channel 0
07	TDR1	Transmit Data Register, Serial Channel 1
08	RDR0	Receive Data Register, Serial Channel 0
09	RDR1	Receive Data Register, Serial Channel 1
0A	CNTR	Clocked Serial Control Register
0B	TRDR	Clocked Serial Data Register
0C	TMDR0L	Timer Data Register, Channel 0, least
0D	TMDR0H	Timer Data Register, Channel 0, most
0E	RLDR0L	Timer Reload Register, Channel 0, least
0F	RLDR0H	Timer Reload Register, Channel 0, most
10	TCR	Timer Control Register
11–13	—	*Reserved*
14	TMDR1L	Timer Data Register, Channel 1, least
15	TMDR1H	Timer Data Register, Channel 1, most
16	RLDR1L	Timer Reload Register, Channel 1, least
17	RLDR1H	Timer Reload Register, Channel 1, most
18	FRC	Free Running Counter
19–1F	—	*Reserved*

continued…

Table G-1. Z180 Addresses 00-3F (concluded)

Address	Name	Description
20	SAR0L	DMA Source Address, Channel 0, least
21	SAR0H	DMA Source Address, Channel 0, most
22	SAR0B	DMA Source Address, Channel 0, extra bits
23	DAR0L	DMA Destination Address, Channel 0, least
24	DAR0H	DMA Destination Address, Channel 0, most
25	DAR0B	DMA Destination Address, Channel 0, extra bits
26	BCR0L	DMA Byte Count Register, Channel 0, least
27	BCR0H	DMA Byte Count Register, Channel 0, most
28	MAR1L	DMA Memory Address Register, Channel 1, least
29	MAR1H	DMA Memory Address Register, Channel 1, most
2A	MAR1B	DMA Memory Address Register, Channel 1, extra bits
2B	IAR1L	DMA I/O Address Register, Channel 1, least
2C	IAR1H	DMA I/O Address Register, Channel 1, most
2D	—	*Reserved*
2E	BCR1L	DMA Byte Count Register, Channel 1, least
2F	BCR1H	DMA Byte Count Register, Channel 1, most
30	DSTAT	DMA Status Register
31	DMODE	DMA Mode Register
32	DCNTL	DMA/WAIT Control Register
33	IL	Interrupt Vector Low Register
34	ITC	Interrupt/Trap Control Register
35	—	*Rreserved*

Other Input/Output Addresses

The I/O addresses listed in Table G-2 control the I/O devices that are external to the Z180 processor.

Table G-2. External I/O Device Addresses

Address	Data bits	Symbol	Function
0xC0	D0-D7	PIODA	PIO Channel A, Data Register
0xC1	D0-D7	PIODB	PIO Channel B, Data Register
0xC2	D0-D7	PIOCA	PIO Channel A, Control Register
0xC3	D0-D7	PIOCB	PIO Channel B, Control Register
0xE000	D0-D7	MEMMAP	(Write) memory map state for the Development Board.
		CONFIG	(Read) configuration jumpers on Development Board
0xF000	D0-D7	—	UART on the Development Board

Real-Time Clock (RTC) Registers

Table G-3 lists the addresses for the DS1302 RTC. The addresses are internal to the RTC, and are not for the Z180 inputs/outputs or memory.

Table G-3. DS1302 Real-Time Clock Registers

Register		Command		Data							
		Write	Read	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Clock	SEC	80	81	CH	10 seconds			seconds			
	MIN	82	83	0	10 minutes			minutes			
	HR	84	85	12/24	0	A/P	10 hours	hours			
	DATE	86	87	0	0	10 days		days			
	MONTH	88	89	0	0	0	10 months	months			
	DAY	8A	8B	0	0	0	0	0	weekday		
	YEAR	8C	8D	10 years				years			
	CONTROL	8E	8F	WP	0	0	0	0	0	0	0
	TRICKLE CHARGER	90	91	1010 = charger selected other = charger off				diode select		resistor select	
	CLOCK BURST	BE	BF								
RAM	RAM 0	C0	C1	RAM DATA 0							
	…	…	…	…							
	RAM 30	FC	FD	RAM DATA 30							
	RAM BURST	FE	FF								

Register Fields

CH = clock halt: 1 = halt; 0 = run
10 seconds, seconds: 00-59
10 minutes, minutes: 00-59
12/24 mode: 1 = 12 hour; 0 = 24 hour
A/P: 1 = PM; 2 = AM
10 hours, hours: 01-12 or 00-23
10 days, days: 01-28, 29, 30, 31
10 months, month: 01-12
weekday: 01-07
10 years, years: 00-99

WP: 1 = write-protect enabled; 0 = write-protect disabled
diode select: 11 = off; 10 = 2 diodes; 01 = 1 diode; 00 = off
resistor select: 11 = 8 kΩ; 10 = 4 kΩ; 01 = 2 kΩ; 00 = off
clock burst: consecutive bytes starting at address 80
RAM burst: consecutive bytes starting at address C0

Interrupt Vectors

Table G-4 shows a suggested interrupt vector map. Most of these interrupt vectors can be altered under program control. The addresses are given are 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 set by the boot code in the Dynamic C EPROM.

Table G-4. Interrupt Vectors for Z180 Internal Devices

Address	Name	Description
0x00	**INT1_VEC**	Expansion bus attention INT1 vector
0x02	**INT2_VEC**	INT2 vector
0x04	**PRT0_VEC**	PRT Timer Channel 0
0x06	**PRT1_VEC**	PRT 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**	Asynchronous Serial Port Channel 0
0x10	**SER1_VEC**	Asynchronous Serial Port Channel 1
0x12	**PIOA_VEC**	PIO Channel A (through INT0)
0x14	**PIOB_VEC**	PIO Channel B (through INT0)

To "vector" an interrupt to a user function in Dynamic C, a directive such as the following is used:

```
#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() {
   ...
}
```

Interrupt Priorities

Table G-5 lists the interrupt priorities from highest to lowest.

Table G-5. Interrupt Priorities

Interrupt Priorities	
(Highest Priority)	Trap (Illegal Instruction)
	NMI (Nonmaskable Interrupt)
	INT 0 (maskable interrupts, level 0. 3 modes, PIO interrupts)
	INT 1 (maskable interrupts, level 1. PLCBus attention line interrupt)
	INT 2 (maskable interrupts, level 2)
	PRT Timer Channel 0
	PRT Timer Channel 1
	DMA Channel 0
	DMA Channel 1
	Clocked Serial I/O
	Serial Port 0
(Lowest Priority)	Serial Port 1

APPENDIX H: ***SIMULATED EEPROM***

Appendix H provides information about how to use flash EPROM to simulate EEPROM for the BL1400.

The BL1400 does not have an EEPROM. However, boards that equipped with a flash EPROM simulate the presence of EEPROM.

The “EEPROM constants” in Table H-1 apply to a BL1400 or BL1410 with flash EPROM.

Table H-1. BL1400 “EEPROM” Constants

Address	Definition
0	Startup mode: 1—enter program mode 8— execute loaded program at startup.
1	Programming baud rate in multiples of 1200 bps. The factory value for this is 16, meaning 19,200 bps.

Symbols

#define F-9
#INT_VEC 4-8, 4-9, G-6
+5 V C-2, D-2, D-3
+5 V supply 3-14, 3-17
/ASTB 3-4
/AT E-3
/BSTB 3-4
/CTS 3-22, 3-23, 3-26
/CTS/PS 3-26
/CTS0 3-11
/DCD0 3-21, 3-22, 3-23, 3-24
/INT0 3-6
/RDX E-3
/RESET 3-14
/RTCRST 3-15, 4-3
/RTS0 3-25
/RTS1 3-21
/STBX E-3
/WRX E-3
= (assignment) A-4
2 × 2 LCD pinout F-4
4-bit bus operations E-4, E-6
4 × 6 keypad
  initializing F-9
5 × 3 addressing mode E-5
555 timer 1-3, 3-3, 3-15, 3-16, 4-6, 4-7, D-6
  Phillips NE555D 3-17
8-bit bus operations E-4, E-5, E-7
9th-bit
  binary communication 4-14, 4-15
  transmission 3-12

A

A0X E-3
A1X, A2X, A3X E-3, E-4
addresses
  encoding E-5
  modes E-5
  PLCBus E-4, E-5, F-5
analog input 1-3
ARDY 3-4
array of pads D-3
ASCI 3-23, 3-25, 3-26
  channel operation 3-25
  Control Register A 3-25
  Control Register B 3-27
  serial ports 3-21
  status registers 3-23
attention line E-3

B

background routine E-6
base plate 1-4, 3-14, B-6, B-7, C-2
battery
  external backup 3-15, 4-5
baud rate 1-5, 3-10, 3-20, 3-21, 3-26, 3-27, 4-7, 4-9, 4-14
  changing 3-2
  Development Board
    programming jumper settings B-9
  serial ports 3-21
bidirectional data lines E-3
BIOS
  Dynamic C 1-5, 2-2, 4-3, 4-7
bitwise mode
  PIO 3-6, 3-7
BL14_15.LIB 1-4, 4-3
BL1400 1-2
  block diagram 3-3
  default communication rate 2-8
  Development Board 2-2, 3-4
    dimensions B-4
    headers and jumper settings B-9

BL1400
dimensions B-2
features 1-3
interfaces 3-3
memory map 3-19
options 1-3
Prototyping Board 1-4, 2-9, 3-12, C-2, D-2, D-3, D-4, D-5
dimensions B-5
BL1410
features 1-3
options 1-3
block diagram
BL1400 3-3
BRDY 3-4
buffer
receive 3-10, 3-11, 4-9, 4-10, 4-14, 4-15
initialization 4-9
reading 4-10
transmit 3-10, 4-10, 4-15
initialization 4-9
writing 4-10
bus
control registers E-7
expansion 1-4, 3-10, E-2, E-3, E-4, E-5, E-6, E-7, F-5, F-6, F-7, F-8
4-bit drivers E-8
8-bit drivers E-10
addresses E-6
control registers E-7
devices E-6, E-7
functions E-8, E-9, E-10, E-11
rules for devices E-6
software drivers E-7
LCD E-3
operations
4-bit E-4, E-6
8-bit E-4, E-7
BUSADR0 E-4, E-5, F-5
BUSADR1 E-4, E-5
BUSADR2 E-4, E-5, F-5
BUSADR3 E-10, E-11
BUSRD0 E-7, E-8, E-9, E-11, F-5
BUSRD1 E-7, E-8, F-5
BUSRD2 F-5
BUSWR E-8, F-5
buzzer D-6
BZ1 (buzzer) D-6

C

changing baud rate 3-2
`Charger1302` 4-6
charging batteries 3-15
`check_opto_command` 4-14
checking for modem commands 4-12
`Checksum` 3-12
CKA1 3-25
CKA1 disable 3-25
CKA1/–TEND0 3-25
CKA1D 3-25
clock
external 3-21
real-time *See* real-time clock
time/date 1-3, 3-6, 3-14, 3-15, 4-3, 4-5, 4-6.
clock frequency 3-10
system 3-20, 3-21, 3-26, 3-27
CNTLA 3-24
CNTLB 3-26
CNTLB0 3-20
CNTLB1 3-20
COM port 1-5
COMMAND mode
modem communication 3-11
command protocol
master-slave 3-12
common problems
programming errors A-4
wrong COM port A-2
communication. *See* serial communication

compile-time interrupt directive 4-13
connections
RS-485 two-wire network 3-13
connectors B-7
26-pin
pin assignments E-2
Control Register A 3-25
Control Register B 3-26
CRC 3-12, 4-14, 4-15
computing 4-13
CSIO 3-24, 4-13
CTS 3-10, 3-11, 3-24, 4-9
CTS Enable 3-24
CTS/PS 3-26
CTS1 3-24
custom EPROM 2-4
cyclic redundancy check 3-12, 4-14, 4-15
computing 4-13

D

D0X-D7X E-3
D1 D-5
D2 D-5
DAC board
expansion bus F-7, F-8
PLCBus F-7, F-8
data format mode bits 3-25
DATA mode
modem communication 3-11
DB4 F-3
DB5 F-3
DB6 F-3
DB7 F-3
DC
input jack C-2, D-4
unregulated input voltage B-7
DCD 3-23
DCIN 2-9, 3-2, 3-14
Ddelay_100ms 4-13
Ddelay_1sec 4-13
DebounceCount F-9, F-10
debouncing 3-14, F-9
deciphering modem commands 4-12
delay
in slave response 4-15
modem communication 4-13
developer's kit 1-4
contents 1-4, 2-2
development
software 1-5
Development Board 2-2, 3-4, B-4, B-9
headers and jumper settings B-9
RAM 2-3
development methods 1-5, 2-2, 2-5
Dget_modem_command 4-12
digital input 1-3, 4-6
digital output 1-3
Dinit_z0 3-11, 4-9, 4-11
DIP relays E-2
disabling interrupts 4-13
DMA channels 4-13
Z180 serial channels 0 and 1 4-13
disabling RS-485 driver 4-16
display
liquid crystal E-3, F-3, F-8, F-9
Dkill_z0 4-10
DMA 3-25
DMA Channel 0
and 555 timer 4-7
DMA channels
disabling interrupts 4-13
downloading data 3-10, 3-11, 4-11
downloading programs 4-9, 4-13
DR 3-26
Dread_z0 4-10
Dread_z01ch 4-10
DREQ0 4-7
Dreset_z0rbuf 4-10
Dreset_z0tbuf 4-10

Drestart_z0modem 4-12
drivers
expansion bus E-7
4-bit E-8
8-bit E-10
high-level 4-3
low-level 4-3
relay E-7
DRIVERS.LIB E-7
function library 4-16
DS1232 supervisor IC 1-3, 3-14
DS1302 RTC IC 1-3, 3-15, 4-5, 4-6, G-4, G-5
Dwrite_z0 4-10
Dwrite_z01ch 4-10
Dxmodem_z0down 4-11
Dxmodem_z0up 4-11
Dynamic C 1-5
BIOS 1-5, 2-2, 4-3, 4-7
programming port 4-13
running a sample program 2-8
serial options 2-8
will not start A-2
Dz0_circ_int 4-13
Dz0modem_chk 4-12
Dz0send_prompt 4-10

E

E000 3-5
echo option 3-10, 3-11, 4-9
ee_rd 4-7
ee_wr 4-7
EEPROM 4-13
simulated 4-7, H-2
EFR 3-24
bit 3-24
eioPlcAdr12 E-8
eioReadD0 E-9
eioReadD1 E-9
eioReadD2 E-9
eioResetPlcBus E-8
eioWriteWR E-10
EN F-3
EN485 4-3
EPROM 1-3, 1-5, 2-2, 2-5, 2-7, 3-4, 3-19, 4-7, 4-8, 4-9
flash 1-3, 1-5, 2-2, 2-5, 4-7, H-2
how to write data 4-7
jumper settings B-8
escape sequences F-8
Exp-A/D12 E-2
expansion boards 1-4, 2-9, 3-9, 3-14, D-2, D-3, D-4, D-5
reset E-8
expansion bus 1-4, 3-10, E-2, E-3, E-4, E-5, E-6, E-7, F-5, F-6, F-7, F-8
4-bit drivers E-8
8-bit drivers E-10
addresses E-6
devices E-6, E-7
digital inputs E-7
functions E-8, E-9, E-10, E-11
registers E-4, E-6
rules for devices E-6
software drivers E-7
external battery 3-15
external clock 3-21
external reset switch 3-14
EXTRES 3-16, D-6
EZIOBL17.LIB E-7
EZIOLGPL.LIB E-7
EZIOMGPL.LIB E-7
EZIOPL2.LIB E-7
EZIOPLC.LIB E-7
EZIOTGPL.LIB E-7

F

F000 3-5
FE 3-24, 3-25
filter
low pass D-6
RC D-6
flash EPROM 1-3, 1-5, 2-2, 2-5, 4-7, H-2
writing to 4-7

frequency
system clock 3-10, 3-21, 3-26, 3-27
function libraries 4-16, E-4
DRIVERS.LIB 4-16
PLC_EXP.LIB 4-16
serial communication 4-16

G

getcrc 4-13
GND 2-9, 3-2, 4-6, D-2, D-3, F-3

H

H1 3-4, D-2, D-3
H2 3-3, 3-4, 3-10, 3-12, D-3
H3 2-9, 3-2, 3-3, 3-4, 3-9, 3-10, 3-12, 3-14, 3-15, 3-16, C-2, D-2, D-3, D-4
handshaking
RS-232 3-10
Hayes Smart Modem 4-12
headers B-8, B-9
heat dissipation
with base plate C-3
without base plate C-4
heat sink 1-4, 3-14, B-6, B-7, C-2
high-level drivers 4-3
hitwd 3-14

I

I/O devices G-2, G-4
I/O interface 4-3
I/O map G-2
I/O space G-2
impedance
PIO 3-4
initialization
keypad F-9
receive buffer 4-9
initialization
serial communication 4-8
transmit buffer 4-9
Z180 Port 1 3-12, 4-14
initiating serial transmission 4-10, 4-15
inport 3-20, E-8, E-9, E-11
input mode
PIO 3-6
strobed 3-6
inputs
analog 1-3, 4-6
digital 1-3, 4-6, E-7
RS-232 4-10
INT1 4-7, 4-13
INT2 4-13
interfaces 3-3
I/O 3-3
interrupt handling
Z180 Port 0 4-8
interrupt service functions 4-13
interrupt service routines 4-13, 4-16
interrupt vectors 3-21
interrupt-driven driver 3-22
interrupt-driven transmission 4-15
interrupts 3-22, 3-23, 3-24, E-3, E-6
disabling 4-13
keyword G-6
PIO 3-6, 3-7, 3-8
routines E-6
serial 4-8
and debugging 4-8
serial communication 3-10
vector table 4-8, 4-13
vectors G-6

J

J1 D-5
J2 D-5
JP4 B-9

jumper settings
BL1400 B-8
JP1 B-8
JP2 B-8
BL1400 flash EPROM 2-6
BL1400 non-flash EPROM 2-7
Development Board 2-3, B-9
Prototyping Board flash EPROM 2-6, D-3, D-5
run mode B-9

K

kernel
real-time 4-13, 4-15
KEY4x6 F-9
keypad 1-4
keypad driver F-8, F-9

L

lc_cgram F-8
lc_char F-8
lc_ctrl F-8
lc_init F-8
lc_keyscan F-9, F-10
lc_kxget F-9, F-10
lc_kxinit F-9
lc_printf F-8
LCD 1-4, E-3, F-3, F-8, F-9
printf F-8
special characters F-8
LCD bus E-3
LCDX E-3
LEDs D-5
libraries
function 4-16, E-4
communications 4-16
DRIVERS.LIB 4-16
PLC_EXP.LIB 4-16
liquid crystal display E-3, F-3, F-8, F-9
locations
mounting holes
base plate B-6
BL1400 B-2
Development Board B-4
Prototyping Board B-5
low-pass filter D-6
low-level drivers 4-3

M

master message format 3-12, 4-14
master-slave
command protocol 3-12
functional support 3-12
networking 3-12, 4-16
serial communication 3-12, 4-14, 4-15
software support 4-14
measuring resistance 4-6
memory
battery-backed 4-9
extended
and uploaded data 4-11
random access 3-11
read only 3-4, 3-19
read-only 1-3, 1-5, 2-2, 2-5, 2-7, 4-7, 4-8
memory map
BL1400 3-19
memory-mapped I/O register E-4
mginit_dac F-7
mglatch_dac1 F-7
mglatch_dac2 F-7, F-8
mgplc_dac_board F-7
mgplc_poll_node F-6
mgplc_set_relay F-6
mgplcrly_board F-6
mgplcuio_board F-6
mgread12data0 F-5
mgread12data1 F-5
mgread12data2 F-5
mgreset_pbus F-5
mgrestore_pbus F-5, F-6
mgsave_pbus F-5, F-6

mgset_dac1 F-7
mgset_dac2 F-8
mgset12adr F-5, F-7, F-8
mgset12data F-5
mgwrite_dac1 F-7
mgwrite_dac2 F-7, F-8
mgwrite4data F-5
misticware 4-15
MOD0 3-25
MOD1 3-24, 3-25
MOD2 3-25
modes
  addressing E-5
  PIO operation 3-5
    Mode 0 3-5, 3-6
    Mode 1 3-5, 3-6
    Mode 2 3-5, 3-6
    Mode 3 3-5, 3-6
modems 4-9, 4-11
  commands 3-11, 4-12
  communication delay 4-13
  control lines 3-21
  deciphering commands 4-12
  options 3-11
  restarting communication 4-12
mounting holes
  base plate B-6
  BL1400 B-2
  Development Board B-4
  Prototyping Board B-5, D-3
MP 3-26, 3-27
MPBR/EFR 3-25
MPBT 3-27
MPE 3-26
multiprocessor
  bit receive/error flag reset 3-25
  bit transmit 3-27
  enable 3-26
  mode 3-25, 3-27

N

NE555D
  Phillips 3-17
network
  RS-485 1-4, 3-12
network connections
  two-wire RS-485 3-13
NO_CARRIER message 3-11
nonmaskable interrupt A-3
nonvolatile memory 4-7, 4-8
NTC thermistors 3-18
NULL modem 4-12
number of bits 4-9

O

op_init_z1 4-14
op_kill_z1 4-16
op_rec_z1 4-15
op_send_z1 4-15
operating modes 4-7
  PIO 3-5
operating temperature B-7
Opto 22 binary protocol 3-12, 4-14, 4-15, 4-16
optodelay 4-15
outport 3-20, 3-21, E-8, E-9, E-11
outputs
  digital 1-3
  RS-232 3-10, 4-10
output mode
  PIO 3-6
  strobed 3-6
overrun 3-24
overrun error 3-24
OVRN 3-24, 3-25

P

PA0 F-3
PA0–PA7 3-4
PA1 F-3
PA2 F-3
PA3 F-3
PA4 F-3
PA5 F-3
PA6 F-3

PA7 F-3
parallel I/O 1-3, 3-3, 3-4, 3-5, 3-6, 3-7, F-3, F-5, F-8, F-9
parity 3-26, 4-9
PB0–PB7 3-4
PB2 3-6
  and RTCDAT 3-14
PB3 3-6
  and RTCCLK 3-14
PBUS_TG.LIB F-2
PC 2-2
PE 3-24, 3-25
PEO 3-26
Phillips NE555D 3-17
phy_adr 4-11
piggyback expansion boards 1-4, 3-9, 3-14, D-2, D-3, D-4, D-5
  power connection 2-9
PIO 1-3, 1-4, 3-3, 3-4, 3-5, 3-6, 3-7, 3-10, 4-3, F-3, F-5, F-8, F-9
  bitwise mode 3-6, 3-7
  I/O register control word 3-7
  impedance 3-4
  interrupt control word 3-7
  interrupt disable word 3-8
  interrupt vector word 3-7
  interrupts 3-6
  mask control word 3-8
  Mode 0 3-5, 3-6
  Mode 1 3-5, 3-6
  Mode 2 3-5, 3-6
  Mode 3 3-5, 3-6, 3-7
  mode control word 3-7
  operation modes 3-5
  ports 4-3
  Port A 3-4, 3-5, 3-7, 4-3, F-5, F-8, F-9
  Port B 3-4, 3-5, 3-6, 3-7, 4-3, F-9
  registers 3-5
  strobed input mode 3-6
  strobed output mode 3-6
PIOA_VEC 3-7
PIOB_VEC 3-7
PIOCA 4-4
PIOCAShadow 4-3, 4-4
PIOCB 4-4
PIOCBShadow 4-3, 4-4
PIODA 4-4
PIODB 4-4, 4-5
PLC_EXP.LIB
  function library 4-16
PLCBus 3-10, E-2, E-3, E-4, E-5, E-6, E-7, F-5
  26-pin connector
    pin assignments E-2
  4-bit operations E-4, E-5
  8-bit operations E-4, E-5
  addresses E-4, E-5, F-5, F-6, F-7
  DACs F-7, F-8
  installing
    BL1400 F-2
  reading data E-4
  relays F-6
    DIP E-2
    drivers E-7
  reset F-5
  state F-6
  writing data E-4
ports
  serial 3-10, 3-11, 3-20, 3-21
    asynchronous 3-21
    baud rate 3-21
    interrupt-driven 3-21
    multiprocessor communications feature 3-21
    polling 3-21
power 2-9, 3-2, 3-3, 3-14, 4-6, B-7, C-2, D-3, D-4
  consumption 3-14
  maximum dissipation 3-14, C-2
power failure 1-3
prescaler 3-27
printf
  for LCD F-8

processor
stack F-6
program development methods 1-5, 2-2, 2-5
via Development Board 2-2
via flash EPROM 2-5
via regular EPROM 2-7
programmable ROM 1-3, 1-5, 2-2, 2-5, 2-7, 3-4, 3-19, 4-7, 4-8, B-8
programming 1-5
protocol
command
master-slave 3-12
Prototyping Board 1-4, 3-12, B-5, C-2, D-2, D-3, D-4, D-5
pad array D-2
power rails D-2
sample circuits D-5
sample demonstration circuits
SW1 D-5
SW2 D-5
switches D-5
thermistor D-6, D-7
TP1 D-5
TP2 D-5
TP3 D-5
TP4 D-6
TP5 D-6
TP6 D-5
TP7 D-5
TP8 D-5
TP9 D-6
PRT 4-13
pulse-width modulation D-6
PWM. *See* pulse-width modulation

R

RAM 4-9
battery-backed 3-11
BL1400 1-3, 2-2, 3-19, 4-8
Development Board 2-2, 4-8
RAM
scratchpad 1-3, 3-15, 4-5, 4-6
static 3-11
rbuf_there 4-15
RC filter D-6
RDR 3-24
RDRF 3-22, 3-24, 3-26
RE 3-26
read-only memory 1-3, 1-5, 3-4, 3-20, 4-7, 4-8, 4-9, 4-13
read12data E-9
Read1302 4-6
read24data E-11
read4data E-10
Read555 4-7
read8data E-11
ReadBurst1302 4-6
reading data on the PLCBus E-4, E-9
ReadRam1302 4-5
ready line 3-4
real-time clock 1-3, 3-3, 3-6, 3-14, 3-15, 4-3, 4-5, 4-6, G-4
real-time kernel 4-13, 4-15
receive buffer 3-10, 3-11, 4-9, 4-10, 4-14, 4-15
initialization 4-9
reading 4-10
receiver interrupt 3-22, 3-23, 3-24
rechargeable batteries 3-15, 4-6
registers
DS1302 G-5
PIO 3-5
real-time clock G-5
Z180 I/O G-2
regulator 1-3, 3-14, C-2, D-3
relays
expansion bus F-6
PLCBus F-6
Reload_vec 4-8, 4-13
ReplyOpto22 4-15

reset
expansion boards E-8
reset switch
external 3-14
ResetZ180int 4-13
resistance measurement 3-16, 3-17, 4-6
conversion time 3-16
ResPIOCA 4-4
ResPIOCB 4-4
ResPIODA 4-4
ResPIODB 4-5
restarting modem communication 4-12
RIE 3-24
ROM 2-2, 2-5, 2-7
programmable 1-3, 1-5, 2-2, 2-5, 2-7, 3-4, 3-19, 3-20, 4-7, 4-8, 4-9, 4-13
RS F-3
RS-232 communication 1-3, 2-2, 3-3, 3-9, 3-10, 3-11, 4-10
handshaking 3-10
serial input 4-10
serial output 3-10, 4-10
software support 4-9
RS-485 communication 1-3, 1-4, 3-3, 3-10, 3-12, 3-13, 4-14, 4-15, 4-16
disabling driver 4-16
network connections 3-13
RSR 3-24
RTC. *See* real-time clock
RTCCLK 3-14, 3-15, 4-3
RTCDAT 3-15, 4-3
RTS 3-10, 3-11, 3-25, 4-9
RTS0 3-25
RUNKERNEL 4-13
RXS 3-24

S

sample programs 4-16
SAMPLES\BL14_15 2-8
scratchpad RAM 3-15, 4-5, 4-6
SE1100 E-2
select PLCBus address E-8
sendOp22 4-14
SER0_VEC 4-8
serial communication 1-3, 1-4, 1-5, 2-2, 3-3, 3-9, 3-10, 3-11, 3-12, 3-13, 3-20, 3-21, 3-22, 3-23, 3-25, 3-26, 3-27, 4-8, 4-9, 4-14, 4-15
establishing 2-8
function libraries 4-16
initialization routines 4-8
interrupts 3-10
master-slave 3-12, 4-14, 4-15
RS-232 1-3, 2-2, 3-3, 3-9, 3-10, 3-11, 4-9, 4-10
RS-485 1-3, 1-4, 3-3, 3-10, 3-12, 3-13, 4-14, 4-15, 4-16
RS-485 network 3-12
serial interrupts 4-8
and debugging 4-8
serial ports 3-10, 3-11, 3-20, 3-21
asynchronous 3-21
baud rate 3-21
interrupt-driven 3-21
low-level utility functions 3-20
multiprocessor communications feature 3-21
polling 3-21
serial transmission
initiating 4-10
terminating 4-10
set12adr E-8
set16adr E-8
set24adr E-10
set4adr E-9
Set555 4-7
set8adr E-11
setPIOCA 4-4
setPIOCB 4-4
setPIODA 4-4

setPIODB 4-4
shadow registers E-6
simulated EEPROM H-2
slave response delay 4-15
slave response format 3-12, 4-15
SmartModem™
Hayes 4-12
software development 1-5
software development methods 1-5, 2-2, 2-5
software libraries 4-16, E-4
source (C term) A-4
special characters
for LCD F-8
specifcations
general B-3
SS 3-26
SS0 3-26
SS1 3-26
SS2 3-26
stack
processor F-6
STAT0 3-23
static RAM 3-11
stop bits 4-9
storage temperature B-7
strobed input mode
PIO 3-6
strobed output mode
PIO 3-6
struct tm 4-5
super capacitors 3-15, 4-6
supervisor IC 1-3, 3-14
support libraries 4-16
suspend 4-13, 4-15
switch
external reset 3-14
sysclock 3-20
system clock frequency 3-10, 3-20, 3-21, 3-26, 3-27

T

TDR 3-25
TDRE 3-22, 3-23, 3-25
TE 3-25
temperature
operating B-7
storage B-7
temperatuve vs resistance in thermistors 3-18
termination resistors 3-13
test pads D-3
thermistors
NTC 3-18
temperatuve vs. resistance 3-18
TIE 3-22, 3-23
time and date
structure 4-5
time/date clock 1-3, 3-3, 3-6, 3-14, 3-15, 4-3, 4-5, 4-6
timer
555 1-3, 3-15, 3-16, 4-6, 4-7
PRT 4-13
watchdog 1-3, 3-14
tm 4-5
tm_rd 4-5
tm_wr 4-5
TMRD0L 4-7
transmission
initiating 4-10, 4-15
interrupt-driven 4-15
transmit buffer 3-10, 4-10, 4-15
initialization 4-9
writing 4-10
trickle charge timekeeper 4-5
trickle charger 4-6
troubleshooting
baud rate A-2
C will not start A-2
cables A-2
com port A-3
communication mode A-2
expansion boards A-2
input/output problems A-3

troubleshooting
memory size A-3
nonmaskable interrupts A-3
operating mode A-3
power supply A-3
repeated interrupts A-3
serial link A-3
watchdog timer A-3
two-wire connections
RS-485 network 3-13

U

U1 3-4
U10 3-10
U11 3-10
U2 3-14
U3 3-14
U6 3-4
U8 3-16
U9 3-15
UART 3-5
UIO board
expansion bus F-6
PLCBus F-6
unregulated input voltage B-7
uploading data 3-10, 3-11, 4-11

V

VBAT 3-15, 4-6
VCC 3-4, 3-14, F-3
VO F-3
VSS F-3

W

watchdog timer 1-3, 3-14, A-3
write12data E-10
Write1302 4-6
write24data E-11
write4data E-10
write8data E-11
WriteBurst1302 4-6
WriteFlash 4-7, 4-8
WriteRam1302 4-5
writing data on the PLCBus E-4, E-10

X

X7R 3-17
xdata 4-7, 4-11
XMODEM commands 4-11
XMODEM protocol 3-10, 3-11, 4-11
XP8100 E-2
XP8200 E-2, F-6
XP8300 E-2, F-6
XP8400 E-2, F-6
XP8500 E-2
XP8600 E-2
XP8700 E-2, E-4, E-7
XP8800 E-2, E-7
XP8900 E-2

Z

z0binaryreset 4-10
z0binaryset 4-11
z0modemset 4-11
z0modemstat 4-11
z1_op_int 4-16
Z180
I/O registers G-2
Microprocessor Family User's Manual 3-20
MPU User's Manual 3-20
Port 0 3-11, 4-9, 4-10, 4-11, 4-13
interrupt handling 4-8
Port 1 3-12, 4-16, G-6
initialization 3-12, 4-14
Serial Channel 0 3-21
Serial Channel 1 3-21
Serial Channels 0 and 1
disabling interrupts 4-13
Z180baud 3-20
Z80 Microprocessor Family Manual 3-7

Z-World
2900 Spafford Street
Davis, California 95616-6800 USA

Telephone: (530) 757-3737
Facsimile: (530) 753-5141
24-Hour FaxBack: (530) 753-0618
Web Site: http://www.zworld.com
E-Mail: zworld@zworld.com

Part No. 019-0017-02
Revision 2

Printed in U.S.A.