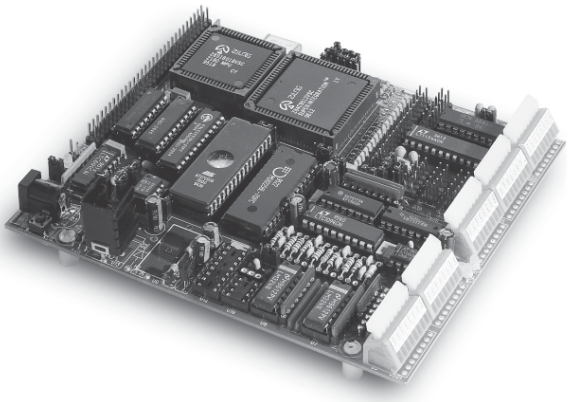


BL1100

C-Programmable Controller

User's Manual

019–0010 • 041015–E

BL1100 User's Manual

Part Number 019-0010 • 041015-E • Printed in U.S.A.

Notice to Users

Z-WORLD PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE-SUPPORT DEVICES OR SYSTEMS UNLESS A SPECIFIC WRITTEN AGREEMENT REGARDING SUCH INTENDED USE IS ENTERED INTO BETWEEN THE CUSTOMER AND Z-WORLD PRIOR TO USE. Life-support devices or systems are devices or systems intended for surgical implantation into the body or to sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling and user's manual, can be reasonably expected to result in significant injury.

No complex software or hardware system is perfect. Bugs are always present in a system of any size. In order to prevent danger to life or property, it is the responsibility of the system designer to incorporate redundant protective mechanisms appropriate to the risk involved.

Trademarks

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

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

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

TABLE OF CONTENTS

ABOUT THIS MANUAL

This manual provides instructions for installing, testing, configuring, and interconnecting any of the Dynamic C programmable controllers in the BL1100 series. (The BL1100 was previously called the Little Giant.)

The term “BL1100” will be used generically throughout this manual when referring to any controller in the BL1100 series. Where information applies to a specific controller, the model number will be specified. Models currently covered by this manual include the BL1100, BL1110, and BL1120.

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 a BL1100 will control.
- Understanding of the basics of operating a software program and editing files under Windows on a PC.
- Knowledge of the basics of C programming.

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

  The C Programming Language by Kernighan and Ritchie
  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 lists and defines the 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
LCD	Liquid Crystal Display
LED	Light-Emitting Diode
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

Icons

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

Table 2. Icons

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

Conventions

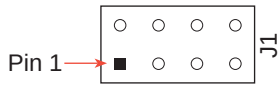
Table 3 lists and defines the typographical conventions that may be used in this manual.

Table 3. 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.

Pin Number 1

A black square indicates pin 1 of all headers.



Measurements

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

CHAPTER 1: OVERVIEW

Overview

The BL1100 is a versatile, general-purpose controller. Numerous digital, analog, and serial channels support complex data collection and control applications. The BL1100 is programmed using Dynamic C, Z-World's version of the C programming language designed for embedded control.

Figure 1-1 illustrates the BL1100 board layout.

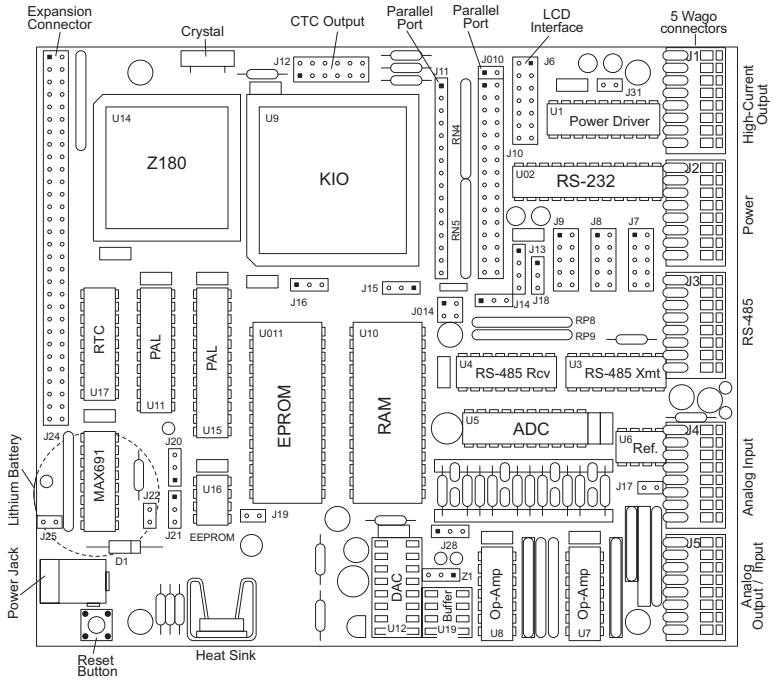


Figure 1-1. BL1100 Board Layout

Features

The BL1100 includes the following features.

- Compactness—dimensions of 5.6" × 4.8" (140 mm × 120 mm).
- Serial channels—two RS-232 and two RS-485/RS-422 serial ports.
- Configurable I/O—16 programmable TTL and CMOS compatible I/O lines based on the 16-bit PIO on the Zilog KIO chip.
- High-current digital outputs—8 high-current digital outputs based on the Sprague 5841 driver. Channels can be paralleled for more current. The drivers are protected against inductive kickback with integral diodes and are suitable for driving solenoids.
- Analog inputs—eight-channel A/D converter with configurable inputs. A built-in temperature sensor uses one of the inputs, leaving seven analog inputs for use by the application. The A/D converter can have 10-bit resolution (LTC1094) or 12-bit resolution (LTC1294).
- Optional factory-installed analog output—12-bit digital to analog converter (DAC) with output in the range of 0 V to 2.5 V. The Maxim AD7543 DAC is used with a MAX400 buffer amplifier for a maximum output of 2 mA.
- LCD interface—standard LCDs (such as Optrex part number DMC20481) can be plugged directly into the 14-pin connector. Many compatible displays are available in various formats (2 ×16 or 4 × 40).
- Z180 microprocessor running at 9.216 MHz with a partial wait state. Higher clock rates are available for better performance.
- Power failure detection and warning. A nonmaskable interrupt takes place when power drops below a certain level. The program has a few milliseconds to shut down.
- Watchdog timer. When enabled, the watchdog timer automatically resets the board if the watchdog timer system is not regularly “hit” by the program. This reliability feature helps a system to recover from software or hardware failures.
- 32K EPROM, supports up to 256K. Accepts either 28- or 32-pin ROM.
- 32K battery-backed SRAM, supports up to 512K. Either 28- or 32-pin RAM can be used. The lithium battery mounted on the board will last about 10 years.
- 512 bytes of EEPROM. One-half of the EEPROM memory (256 bytes) can be write-protected. EEPROM is used to hold the baud rate and other semipermanent calibration or setup constants. (Can be expanded to 2048 bytes by using different components.)

- Battery-backed Epson 72421 real-time clock runs up to 10 years on the lithium battery.
- Both linear and switching power regulators are present on the board.
- Enclosures are available from different suppliers in table-top plastic clamshell or wall-mounted industrial versions. For example, the Hoffman Engineering Company (Anoka, Minn.) enclosure D-864DLB and mounting plate A-8P6 provide a sturdy steel electrical box. This box can be wall mounted and connected to electrical conduit.
- The analog inputs, analog output, power, RS-485, and high-current outputs are on Wago connectors that accept up to approximately #18 copper wire. The Wago connectors have spring loaded clamps that make it easy to install and remove wires without special tools. The entire BL1100 series is also available without Wago connectors so that other connector types may be added instead.
- A 60-pin expansion connector allows a line of expansion boards designed exclusively for the BL1100 series to be connected to the microprocessor bus. Selected standard Z-World expansion boards can also be connected to the PIO header.

Standard Models

The Bl1100 series of controllers has three versions. Table 1-1 lists the specialized features for these versions. All three versions are available either with Wago connectors or without any connectors to enable a customer to use a customer-selected connector.

Table 1-1. BL1100 Series Features

Model	Features
BL1100	9.216 MHz clock, 16 configurable I/O, 8 high-current digital outputs, 8 ten-bit analog inputs, 2 RS-232 and 2 RS-485/RS-422 serial ports, expansion bus, switching regulator
BL1120	BL1100 with linear regulator
BL1120	BL1100 with 12.288 MHz clock and linear regulator

Options and Upgrades

Table 1-2 lists the options and upgrades available for the BL1100 series.

Table 1-2. BL1100 Options and Upgrades

Option	Description
12-bit digital-to-analog converter	Adds one analog output channel rated 2 mA at 0 V to 2.5 V
10-bit to 12-bit analog-to-digital converter upgrade	Upgrades analog-to-digital converter resolution to 12 bits
Op-amp upgrade	Replaces LT1094 op-amps with LT1014 for improved temperature drift and input offset characteristics
SRAM	128K or 512K SRAM factory installed
BL1120 switching power regulator	Switching power regulator factory installed on BL1120

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

Expansion Boards

The BL1100 has a 60-pin expansion connector that allows selected standard Z-World expansion boards and a line of expansion boards designed exclusively for the BL1100 series to be connected to the microprocessor bus. Table 1-3 lists these expansion boards.

Table 1-3. Expansion Boards Used With the BL1100

Expansion Board	Description
Standard Z-World Expansion Boards	
SE1100	Four SPDT relays
XP8300	Six SPDT relays
XP8400	Eight DIP SPST relays
XP8500	Four 12-bit ADC inputs with signal conditioning and seven 12-bit ADC inputs without signal conditioning
XP8600	Two DAC outputs
Exp-A/D12	Eight 12-bit ADC inputs
Expansion Boards Unique to BL1100	
ADC	One 8-bit PIO, two 4-bit optically isolated PIOs, eight TTL inputs, one PWM channel, one 20-bit ADC input, one instrumentation amplifier, and one 4-channel analog mux
DGL*	Two 8-bit PIOs, two 4-bit optically isolated PIOs, and eight TTL buffered inputs
DGL96**	96 individual I/O points
MUX	One 8-bit PIO, one 6-bit PIO, two 4-bit PIOs, and one 4-channel analog mux

* Available with or without Wago connectors.

** Available in stacking or nonstacking versions. Cannot be used with BL1120.

Appendix H provides more information about connecting expansion boards to the microprocessor bus. Appendix K provides detailed information on the expansion boards unique to the BL1100. Refer to the manuals on the individual tandard Z-World expansion boards for more information on the standard expansion boards.

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

Development and Evaluation Tools

The BL1100 is supported by a Developer's Kit that includes everything you need to start development with the BL1100.

Developer's Kit

The Developer's Kit includes these items.

- Manual with schematics.
- Programming cables and adapter.
- 9 V DC power supply.
- 128K SRAM.

Software

The BL1100 is programmed using Z-World's Dynamic C, an integrated development environment that includes an editor, a C compiler, and a debugger. Library functions provide an easy and robust interface to the BL1100.

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

CHAPTER 2: GETTING STARTED

Chapter 2 provides instructions for connecting the BL1100 to a host PC and running a sample program.

Developer's Kit Packing List

The BL1100 Developer's Kit includes the following items.

- Serial cable with DB-9 and 10-pin header connectors.
- DB-25 to DB-9 serial adapter.
- 9 V DC power transformer.
- 128K SRAM chip.
- ***BL1100 User's Manual*** (this document).

Connecting BL1100 to a Host PC

The BL1100 can be programmed using a PC using either an RS-232 serial port or an RS-485 serial port.

See Chapter 3, "System Development," for information on other programming configurations. Z-World recommends trying out the factory-default configuration first to make sure that your setup works.

Factory-Default RS-232 Programming

1. Make sure Dynamic C is installed on your PC as described in the ***Dynamic C Technical Reference*** manual.
2. The BL1100 is factory configured for RS-232 programming at 19,200 bps. Use the programming cable provided in the Developer's Kit (see Figure 2-1) to connect header J7 (shown in Figure 2-2) to the host PC COM1 serial port as shown in Figure 2-3.

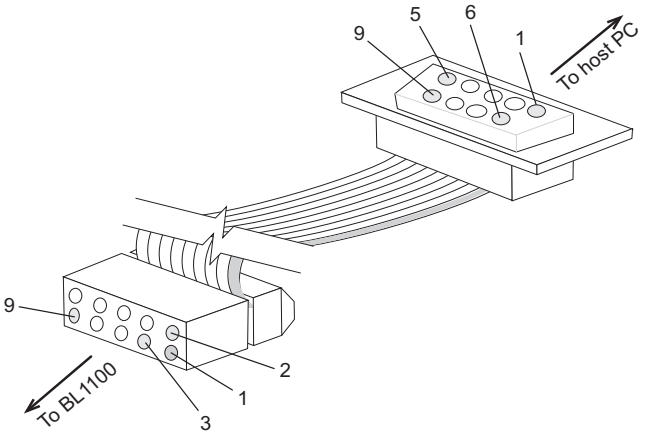


Figure 2-1. Programming Cable

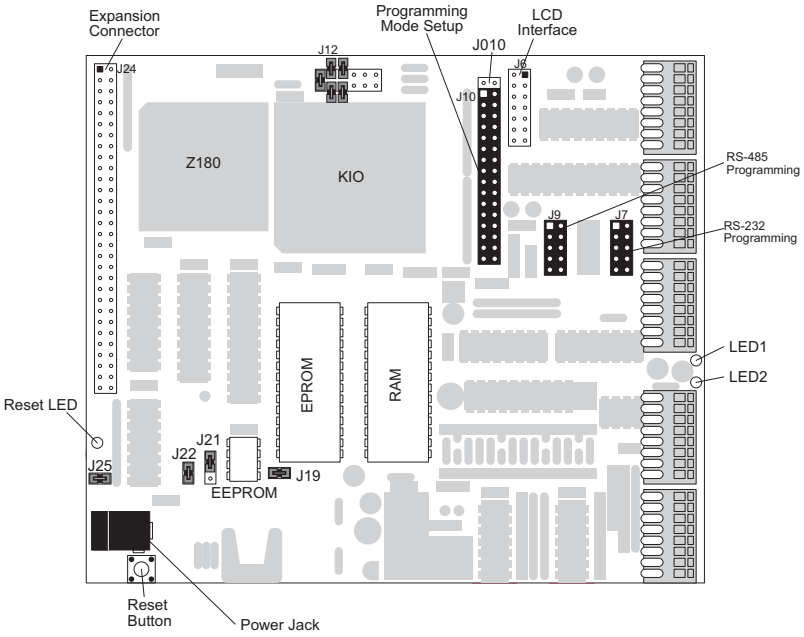


Figure 2-2. Programming Headers and Accessories

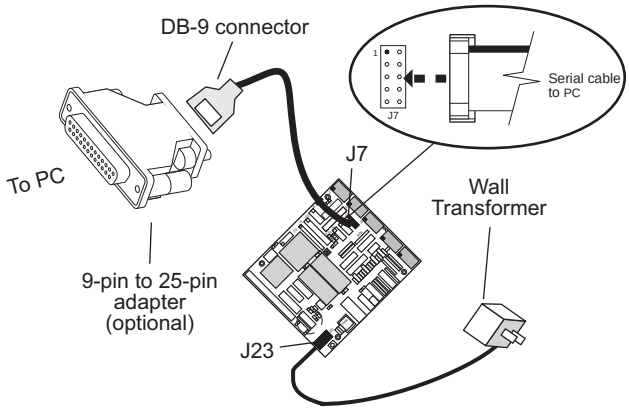


Figure 2-3. Programming Cable and Power Connections

3. Plug in the wall transformer and plug the free end into the BL1100 power jack. The reset LED will flash and LED2 will turn on.

Establishing Communication with the BL1100

1. Double-click the Dynamic C icon to start the software. Note that communication with the BL1100 is attempted each time you start Dynamic C.
2. If the communication attempt is successful, no error messages are displayed.
3. If the programming cable is connected to a port other than COM1, use the **Setup Target** option in the **SETUP** menu **<ALT S>** in Dynamic C to specify the correct port. You may need to reset the BL1100.

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 BL1100, use the Dynamic C shortcut **<Ctrl Y>** to reset the controller and initiate communication.

Running a Sample Program

1. Load the sample program **LGFLASH.C** located in the Dynamic C **SAMPLES** directory using **<ALT O>**. This program flashes LED2.
2. Compile the program by pressing **F3** or by choosing **Compile** from the **Compile** menu.

   During compilation, Dynamic C rapidly displays several messages in the compiling window. This condition is normal.

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

3. Run the program by pressing **F9** or by choosing **Run** from the **Run** Menu. You may also single step through the program using **F7** and **F8**, and experiment with changing the delay times.
4. To halt the program, press **<Ctrl Z>**. This action halts program execution.
5. To restart program execution, when required, press **F9**.

CHAPTER 3: SYSTEM DEVELOPMENT

Chapter 3 describes the more advanced aspects of setting up and using the BL1100. The following sections are included.

- Operating Modes
- Memory

Operating Modes

A hardware reset takes place when the BL1100 is powered up, when the reset button is pressed, or when the watchdog timer times out.

If a valid program (crated with Dynamic C) has been installed in EPROM, the program starts running. A valid program is recognized by a code that Dynamic C places in the file used to burn the EPROM.

The flowchart in Figure 3-1 shows the startup sequence of the BL1100 after a hardware reset.

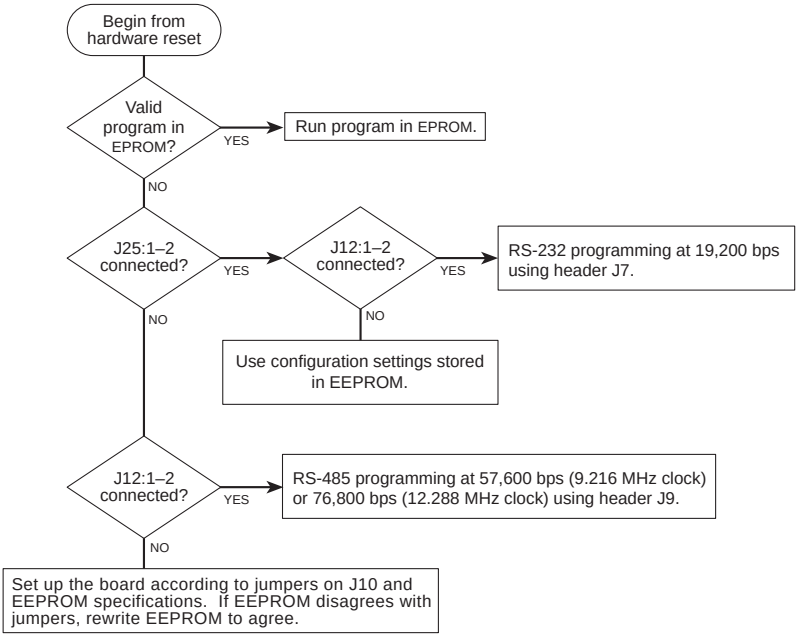


Figure 3-1. BL1100 Activity at Startup

Programming Options

In addition to the RS-232 factory-default programming at 19,200 bps described in Chapter 2, "Getting Started," the BL1100 may be programmed using RS-485/RS-422, and programming is also possible using either RS-232 or RS-485/RS-422 at different baud rates. Figure 3-2 shows the various programming modes and jumper configurations for headers J12 and J25.

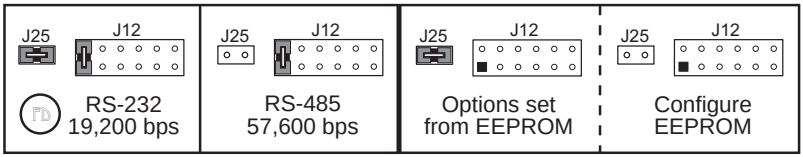


Figure 3-2. Jumper Configurations for Programming Options

Follow these steps for the alternative programming options.

1. Make sure Dynamic C is installed on your PC as described in the ***Dynamic C Technical Reference*** manual.
2. Plug the free end of the power supply into the BL1100 power jack as shown in Figure 3-3.

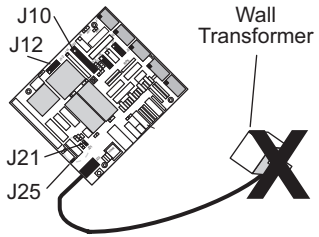


Figure 3-3. Power Supply Connection

> Do not plug the transformer into the wall until all the connections and jumpers have been set on the BL1100.

3. Set the jumpers on header J25 and on header J12 pins 1–2.

   Option 1 (RS-232 at 19,200 bps)—This factory-default option was described in Chapter 2, "Getting Started." No changes to the factory jumper settings are needed.

   Option 2 (RS-485 at 57,600 bps)—Remove the jumper from header J25 as shown in Figure 2-4. This option allows RS-485/RS-422 programming at 57,600 bps (BL1100, BL1110) or 76,800 bps (BL1120). Go to step 4.

   Option 3 (store and use settings in EEPROM)—Move the jumper on header J21 to pins 2–3 as shown in Figure 3-4 to write-enable the EEPROM.

   To be able to store new settings in the EEPROM, remove the jumpers from

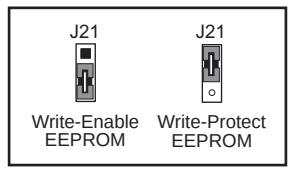


Figure 3-4. EEPROM Jumper Settings

headers J12 pins 1–2 and J25, as shown in Figure 3-2. Set the desired baud rate by connecting pins 23–24 and 25–26 on header J10 as shown in Figure 3-5. Connect either pins 27–28 on header J10 (RS-232 programming via header J7) or pins 28–29 on header J10 (RS-485 programming via header J9).

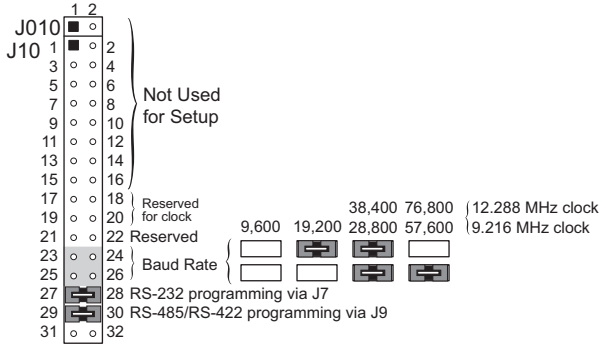
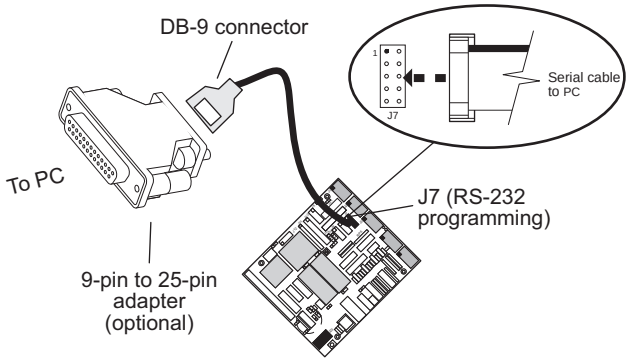


Figure 3-5. Jumper Connections on Header J10 to Set Programming Options in EEPROM

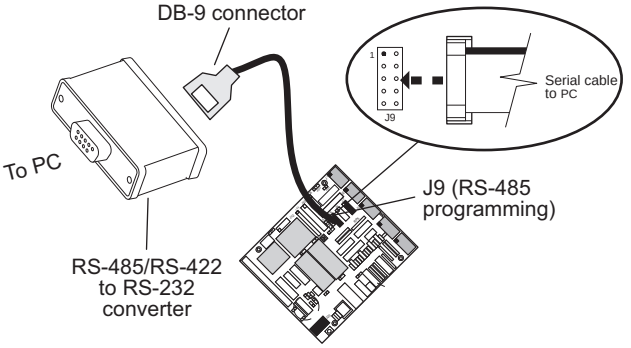
Do ***not*** connect both pins 27–28 ***and*** 28–29 on header J10.

Cycle the power by plugging in the wall transformer and then unplugging it. LED1 will flash to indicate the EEPROM has been rewritten. After the wall transformer is unplugged, remove the jumpers from header J10 and replace the jumper on header J25 to use the new programming mode stored in the EEPROM. Move the jumper on header J21 to pins 1–2 as shown in Figure 3-4 to write-protect the EEPROM.

4. Use the programming cable provided in the Developer's Kit to connect header J7 (RS-232 programming) or header J9 (RS-485 programming) to the host PC COM serial port as shown in Figure 3-6.
5. Plug in the wall transformer. The reset LED will flash and LED2 will turn on.



(a) RS-232



(b) RS-485

Figure 3-6. Programming Cable Connections

A commercially available RS-485/RS-422 to RS-232 converter is required as shown in Figure 3-6 if your PC COM serial port is RS-232 and you wish to program in the RS-485 mode.

Run Mode

If a valid program is already in EPROM, that program starts running following a hardware reset or a watchdog timeout as shown in Figure 3-1.

The BL1100 is factory-configured for RS-232 programming at 19,200 bps. In addition to running a program from EPROM, which requires burning a new EPROM to replace the EPROM supplied with the BL1100, a program may also be run from the onboard RAM. Follow these steps to configure the EEPROM to be able to run a program in RAM.

1. Plug the free end of the power supply into the BL1100 power jack as shown in Figure 3-7.

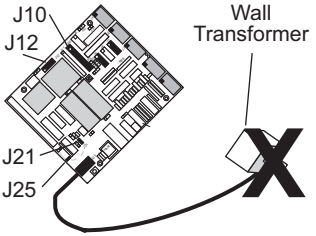


Figure 3-7. Power Supply Connection

2. Move the jumper on header J21 to pins 2–3 as shown in Figure 3-8 to write-enable the EEPROM.

   To be able to store new settings in the EEPROM, remove the jumpers from headers J12 pins 1–2 and J25. Connect pins 31–32 on header J10 as shown in Figure 3-9.

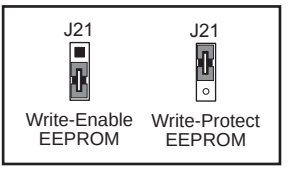


Figure 3-8. EEPROM Jumper Settings

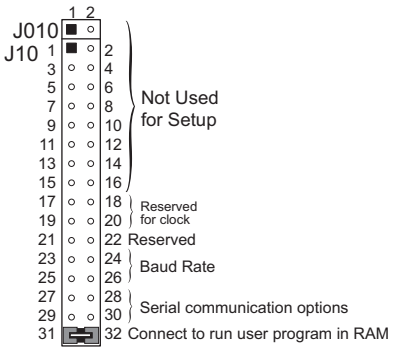


Figure 3-9. Jumper Connections on Header J10 to Set Run Program from RAM in EEPROM

3. Cycle the power by plugging in the wall transformer and then unplugging it. LED1 will flash to indicate the EEPROM has been rewritten. After the wall transformer is unplugged, remove the jumper from header J10 and replace the jumper on header J25 to use the new configuration stored in the EEPROM. Move the jumper on header J21 to pins 1–2 as shown in Figure 3-8 to write-protect the EEPROM.
4. Plug in the wall transformer. The reset LED will flash and the program in RAM will execute.

Switching Between Run and Program Mode

To return to Program Mode, follow the instructions in the ***Programming Options*** section in this chapter. The ***Run Mode*** steps will have to be performed if the EEPROM is rewrittten for programming.

Tip

To switch to factory-default RS-232 programming at 19,200 bps without rewriting the EEPROM, it is possible to "toggle" between run and program mode by placing a jumper across pins 1–2 of header J12 (program mode) or by removing the jumper across pins 1–2 of header J12 (run mode).

Tip

RS-485 programming at 57,600 bps is also possible without rewriting the EEPROM by removing the jumper on header J25 and placing a jumper across pins 1–2 of header J12 (program mode). To return to run mode, replace the jumper on header J25 and remove the jumper across pins 1–2 of header J12.

Memory

EPROM

The development EPROM supplied with the BL1100 in socket U011 must be used when programming the BL1100 with Dynamic C. The development EPROM has the following format for its part number.

680-20*nn* where *nn* indicates the revision number for the BIOS

The 32-pin socket (U011) for the development EPROM accepts 32K to 512K EPROM chips. The socket accepts either 28-pin or 32-pin EPROM chips, including the following.

27C256	32K	28 pins
27C512	64K	28 pins
27C010	128K	32 pins
27C020	256K	32 pins

When using a 28-pin EPROM, four pin positions at one end of the socket are left empty, as shown in Figure 3-10. The access time must be 100 ns or better at 9 MHz.

Header J16 reflects whether the amount of memory, and header J20 reflects the number of pins, as shown in Figure 3-10.

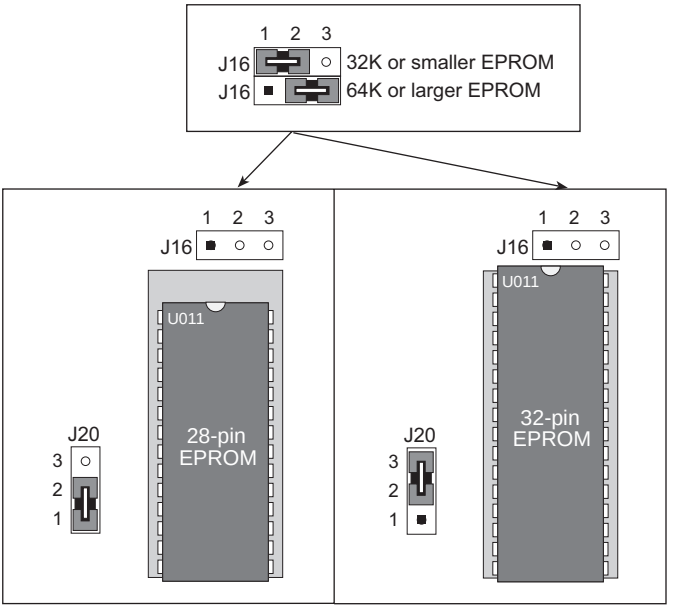


Figure 3-10. 28-pin and 32-pin EPROM Placement and Jumper Configurations

Programming EPROM

Dynamic C can be used to create a file for programming an EPROM by selecting the **Compile to File** option in the **COMPILE** menu with the development EPROM installed. The BL1500 must be connected to the PC running Dynamic C during this step because essential library routines must be uploaded from the development EPROM and linked to the resulting file. The output is a binary file (optionally an Intel hex format file) that can be used to build an application EPROM. The application EPROM is then programmed with an EPROM programmer that reads either a binary image or the Intel hex format file. The resulting application EPROM can then replace the development EPROM.

Copyrights

The Dynamic C library is copyrighted. Place a label containing the following copyright notice on the EPROM whenever an EPROM that contains portions of the Dynamic C library is created.

> ©1995 Z-World

Your own copyright notice may also be included on the label to protect your portion of the code.

Z-World grants purchasers of the Dynamic C software and the copyrighted BL1100 EPROM permission to copy portions of the EPROM library as described above, provided that:

1. The resulting EPROM is used only with the BL1100 manufactured by Z-World, and
2. Z-World's copyright notice is placed on all copies of the EPROM.

SRAM

When doing program development with Dynamic C, it is best to use a 128K SRAM, supplied with the Developer's Kit, or larger. Dynamic C will work with a 32K SRAM, but the total program space will be limited to 16K of root and 16K of extended memory. This is enough for many programs, but it is inconvenient to run out of memory during development. Once a program is burned into EPROM, there is no reason to use SRAM larger than 32K unless the data space is larger than 32K.

The BL1100 SRAM is socketed, and is found at U10 just to the right of the EPROM.

CHAPTER 4: SUBSYSTEMS

Chapter 4 describes the BL1100 hardware subsystems and digital inerfaces. The following sections are included.

- Interface Overview
- Power-Supervisor Integrated Circuit
- Digital Interfaces
- Serial Communication
- Direct Programming using Serial Ports
- Asynchronous Serial Communication Interface
- SIO Serial Ports

Interface Overview

Chapter 4 describes the various interfaces and subsystems of the BL1100. Figure 4-1 provides a block diagram reference.

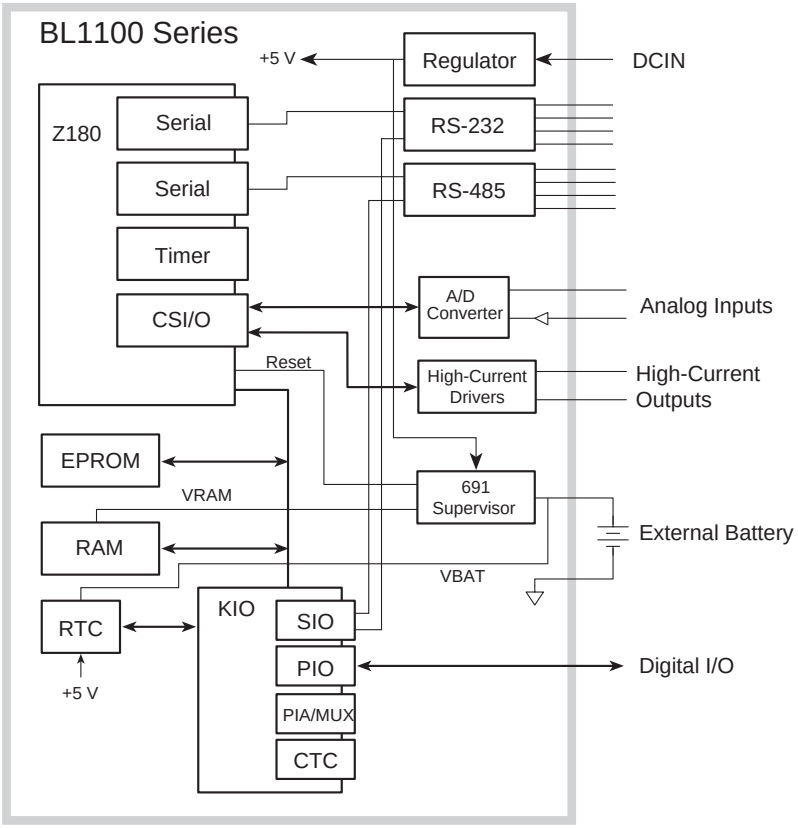


Figure 4-1. BL1100 Interfaces and Subsystems

Power-Supervisor Integrated Circuit

The 691 power-supervisor IC is a key component that helps a system to survive power fluctuations and power outages. Several vital services provided by the power supervisor are described below.

- **Power-on reset**

The supervisor IC generates the power-on reset for the BL1100 by holding /RESET low until the IC senses that VCC has risen above the reset threshold (~4.65 V) and the battery voltage (2.5 V to 4.25 V DC). When VCC falls below the threshold, the supervisor IC disables the RAM to prevent spurious writing of data.

- **RAM protection**

The power-supervisor IC gates the RAM's write-enable line (BATACT) whenever VCC is above the reset threshold and VBAT. When VCC falls below the threshold, the 691 de-asserts BATACT to prevent spurious writing to the RAM.

- **Watchdog timer**

The watchdog timer is enabled by placing a jumper across header 22. The watchdog timer guards against system or software faults. If an application does not "hit" the watchdog timer at least every 1.0 seconds, the watchdog timer resets the Z180. The supervisor's watchdog output (/WDO) connects to the Z180's /INT1 inetrrupt line. /WDO is at logic zero level after a watchdog reset and at logic 1 after a power-on reset.

To "hit" the watchdog timer, make a call to the library function **`hitwd`**. This call makes a dummy one-byte DMA transfer via DMA channel 1, which activates the DMA-end signal, /TEND1, "hitting" the watchdog timer.

- **Nonmaskable interrupt**

The 691 generates a nonmaskable interrupt (/HNMI) from its power-fail output (/PFO) for the microprocessor if the unregulated DC input (normally 9 V DC) falls below 7.6 V. This gives the BL1100 advanced warning of an impending power failuire, which allows it to execute shutdown routines. The voltage divider (R25 and R27) determines the power-fail voltage level.

/HNMI also connects to the KIO to allow your software to monitor the /HNMI line after the nonmaskable interrupt, and to recover from temporary low-input-voltage conditions or "brownouts."

- **Backup-battery switchover**

The 691 switches the RAM over to battery power if VCC falls below the battery voltage, VBAT (2.5 V to 4.25 V DC).

Digital Interfaces

The BL1100 has several digital interfaces.

- KIO interface
- 16-bit PIO (parallel I/O and keypad interface)
- Counter/timers circuit (CTC)
- High-current/high-voltage driver
- Liquid crystal display (LCD)

Zilog KIO Interface

The Zilog KIO interface chip (U9) is an 84-pin LSI chip that combines several functions into one package. Figure 4-2 is a block diagram of the functions performed by the KIO interface chip.

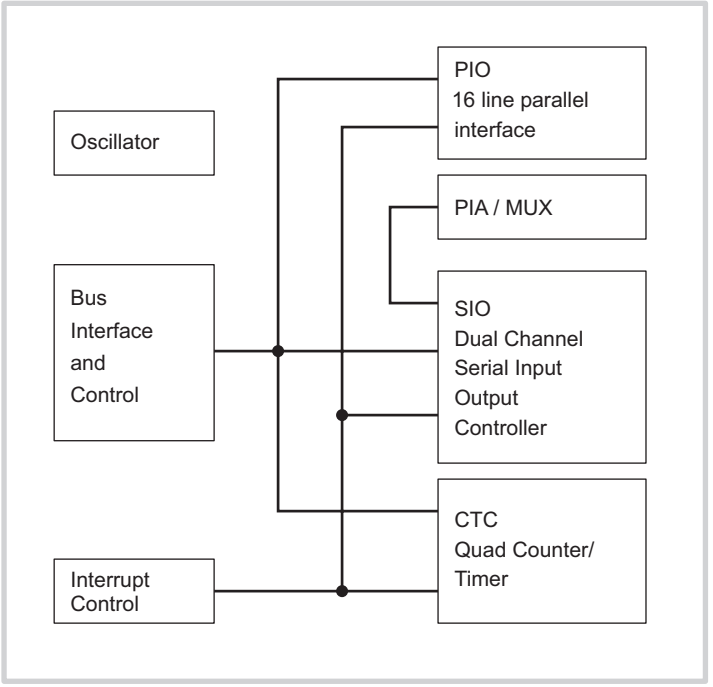


Figure 4-2. KIO Block Diagram

The KIO integrates the following three “peripheral” functions.

- PIO 16-line parallel I/O
- SIO dual-channel serial I/O controller
- CTC quad counter/timer

KIO Command Register

The KIO command register provides overall control of its functions. Figure 4-3 illustrates the layout of the command register.

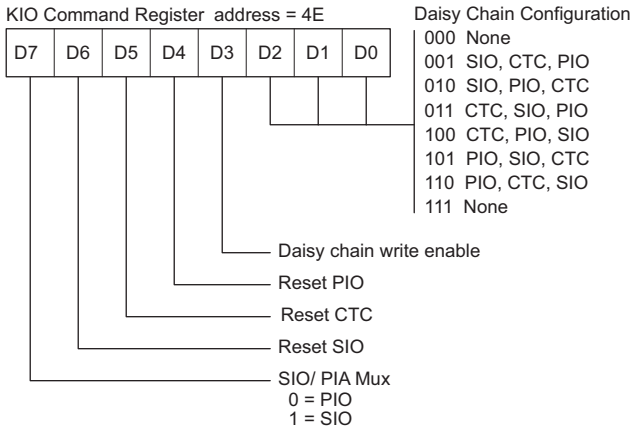


Figure 4-3. KIO Command Register Layout

The reset bits cause a momentary reset when a 1 is stored. The "daisy chain" establishes the relative interrupt priority of the various functions that make up the KIO. To modify the daisy chain, store a 1 in bit 3 at the same time as the new daisy chain configuration. Choice 011 in Figure 4-3 is the default set by Dynamic C. The following function sets the daisy chain where code is delineated in Figure 4-3.

```
int setdaisy( byte code );
```

The default setting gives the CTC the highest priority.

Bit 7 in the KIO command register is normally turned on by Dynamic C so that the SIO can be used.

The relative position on the daisy chain determines which interrupt will occur first when several are pending. All interrupts from the KIO are higher in priority than interrupts from the Z180's internal devices, except NMI and illegal instruction trap interrupts.

PIO Interface

The PIO unit in the KIO provides a 16-bit parallel interface. Figure 4-4 is a block diagram of the PIO parallel interface.

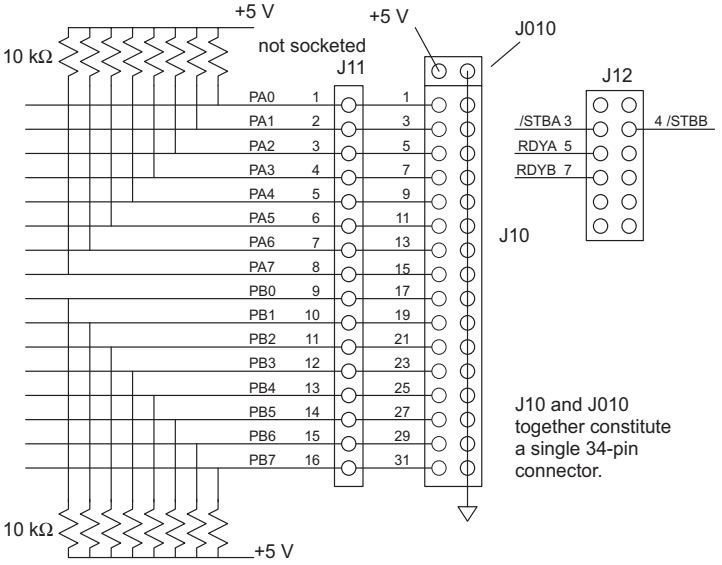


Figure 4-4. PIO Chip and Keypad Interface

Lines PA0–PA7 are considered as "Port A" and lines PB0–PB7 are considered as "Port B." Each line can serve as an input or output in different modes. The four lines on J12 are handshaking lines that consist of a ready line and a strobe line for each port.

J11 is designed to hold a special user-installed connector that clamps on to the flexible circuit ribbon of a membrane keypad. A membrane keypad is a matrix array like the one shown Figure 4-5.

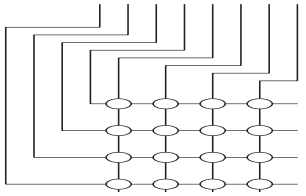


Figure 4-5. Keypad Matrix Array

The PIO can read this keypad by setting each row to zero volts and monitoring each column. Columns are held up by pull-up resistors. The microprocessor can use the PIO to detect closure of any key in the keypad. Debouncing must be done by software.

The impedance of the PIO is approximately 80 Ω for sinking current and 160 Ω for sourcing current. Do not apply voltages below ground or above VCC to the PIO.

The PIO is very flexible and has a number of modes of operation. The two ports are controlled by the following four registers.

40_H	(PIODA)	PIO Port A, data
41_H	(PIOCA)	PIO Port A, command
42_H	(PIODB)	PIO Port B, data
43_H	(PIOCB)	PIO Port B, command

Each register pair controls one of the 8-bit ports and the two handshaking lines associated with each port. The ports' four modes of operation are as follows.

Mode 0—strobed byte output
Mode 1—strobed byte input
Mode 2—bidirectional data transfer (port A only)
Mode 3—bitwise I/O, input/output selectable per bit

Mode 0 (Strobed Byte Output)

When the microprocessor stores a byte in a port's data register, the eight associated output lines change level according to how each bit is set: level high for a 1 or level low for a 0. The ready handshake line goes high. The ready line is reset when an external device pulses the strobe line (low) . 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 when it receives 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)

This mode uses Port A and all four handshake lines. Data can be transferred in both directions under control of the four handshake lines.

Mode 3 (Bitwise I/O)

Mode 3 is a general-purpose input-output mode. Each bit can be specified individually 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.

Using PIO Ports

To set up a port for I/O, first write a sequence of bytes to its command register. Then read, or write, its data register to transfer data. The sample program PIODEMO.C illustrates the use of the PIO registers.

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 as shown in Figure 4-6.

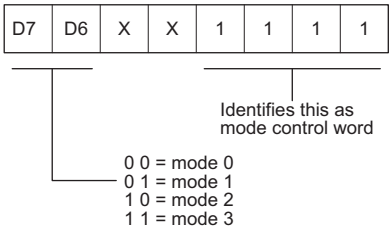


Figure 4-6. PIO Mode Control Word

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

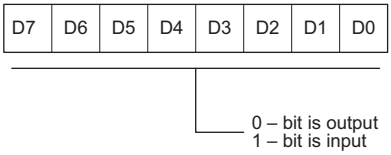


Figure 4-7. PIO Register Control Word

The ***interrupt vector word*** specifies the interrupt vector for the particular PIO channel. The vectors for the PIO ports areas follows.

0x12 (**`PIOA_VEC`**) PIO Port A

0x14 (**`PIOB_VEC`**) PIO Port B

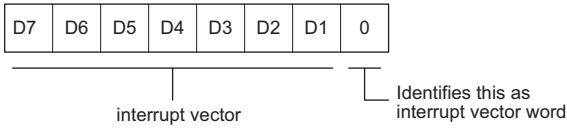


Figure 4-8. PIO Interrupt Vector Word

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

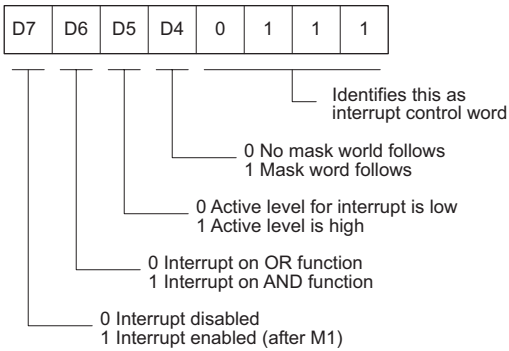


Figure 4-9. PIO Interrupt Control Word

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

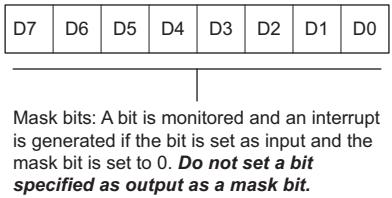


Figure 4-10. PIO Mask Control Word

The ***interrupt disable word*** allows an interrupt to be enabled or disabled 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.

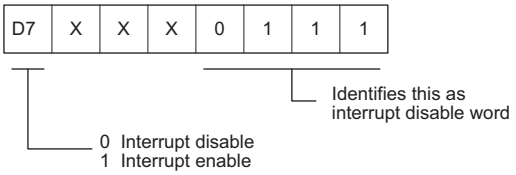


Figure 4-11. PIO Interrupt disable Word

PIA Parallel Port

The PIA parallel port is a part of the KIO chip. However, most of its lines are shared with lines that are essential to the operation of the SIO serial port. Two of the lines, PC1 and PC6, are connected to jumpers that are intended only to provide configuration information at bootup time. Normally, these jumpers will be read only by the Dynamic C kernel, which then disables the PIA and enables the SIO. The contents of these jumpers can be read from the external integer **JUMPERS**.

Counter/Timer Circuit (CTC)

The KIO chip's CTC has four counter/timer units as shown in Figure 4-12. There are also two programmable timers (PRTs) that are a part of the Z180.

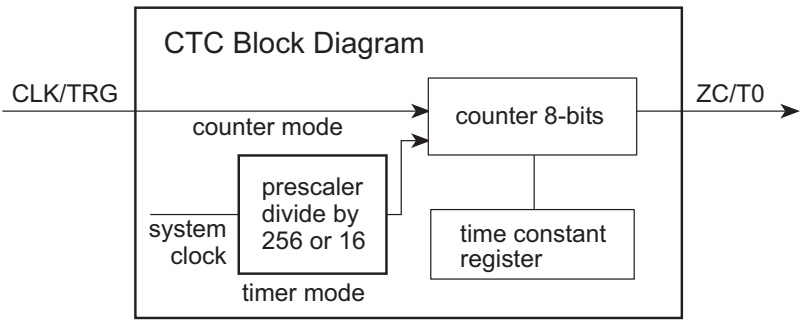


Figure 4-12. CTC Block Diagram

Refer to the Zilog Z80180/Z180 User's Manual for information on PRTs.

Each counter/timer has a counter and a time-constant register. Load the time constant and the counter, and the counter will count down until it reaches zero. This process is repeated, creating a periodic interrupt and/or output pulse. A time constant is a byte from 0 to 255 (0 is considered to be 256). The incoming clock can be derived from the system clock (9.216 MHz) passed through a prescaler unit, or it can be an external clock. Counting can also be started in response to an external trigger.

The BL1100's four counter/timers are connected as shown in Figure 4-13. One I/O port is used for each of the four units. Timer commands are sent by writing to the port. Reading the port returns the current value of the counter without disturbing the count in progress.

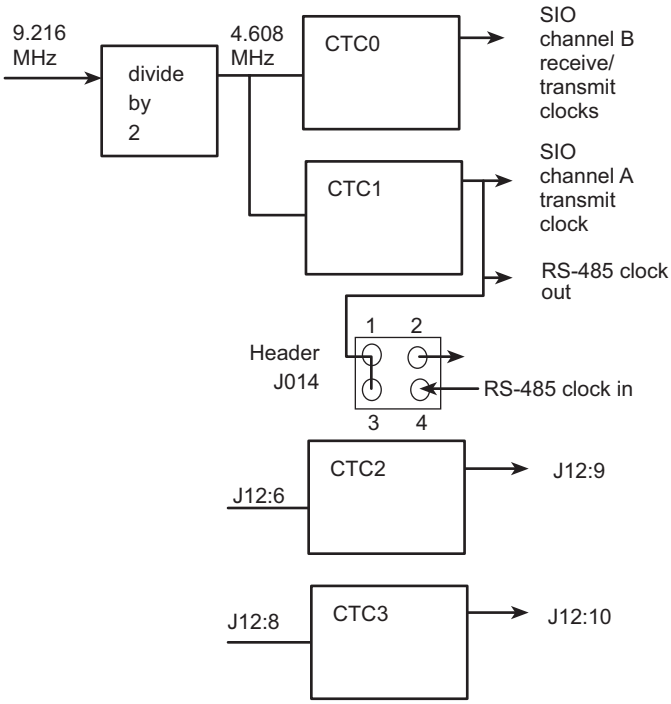


Figure 4-13. Counter/Timer Circuit

Table 4-1 Lists the addresses and functions for the four counter/timers.

Table 4-1. Addresses and Functions of BL1100 Counter/Timers

Address	Name	Function
44	CTC0	SIO clock B
45	CTC1	SIO clock A
46	CTC2	General use
47	CTC3	General use

Each port has an 8-bit counter that counts at the same frequency as the input clock (J12 pin 6 and J12 pin 8) or at the system clock frequency divided either by 16 or by 256. The external input can also be used as a trigger that causes counting to begin, using the system clock to drive the counter at either 1/16 or 1/256 of the clock frequency. Each time the counter passes the value zero, the external output is pulsed (J12 pin 9 or J12 pin 10). CTC units 0 and 1 are intended to provide a clock to the SIO. If not used for that purpose, they can be used as timers, although the external inputs and outputs are permanently connected as shown in Figure 4-13.

Timer Control Word

The ***timer control word*** is illustrated in Figure 4-14.

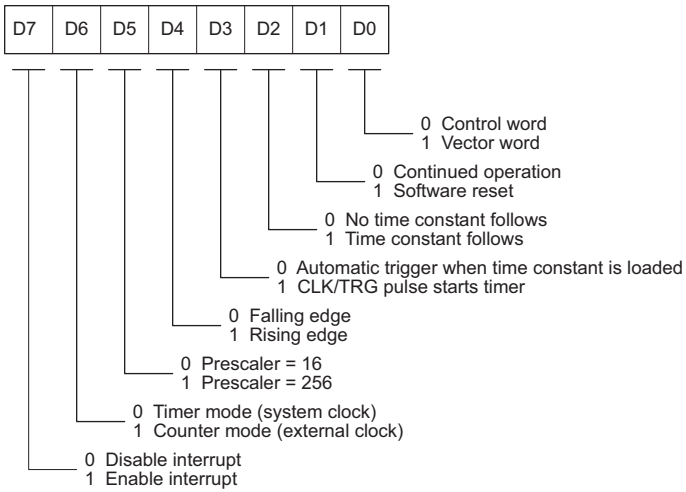


Figure 4-14. Timer Control Word

Note that the interrupt vector can be set by writing a command with bit 0 = 0. Set the time constant by writing a command with bit 2 set (1) immediately followed by a time constant.

Table 4-2 provides sample setups for the CTC.

Table 4-2. CTC Sample Setups

Mode	Setup
CTC1 Counter Mode	Input clock 4.608 MHz Divider: 5 Output clock: 0.9216 MHz SIO baud rate, divide by 16 mode: 57,600 bps
CTC1 Counter Mode	Input clock: 4.608 MHz Divider: 15 Output clock: 307.2 kHz SIO baud rate, divide by 16 mode: 19,200 bps
CTC2 Timer Mode	Input clock: 9.216 MHz Prescaler: 256 Divider: 180 Timer period: 5 ms

Note that it is not possible to obtain 38,400 bps unless the system clock is changed to 6.144 MHz, 12.288 MHz, or another equivalent frequency.

By using the output of one counter/timer as the input to another, much longer periods are obtained. By connecting a CTC or the serial clock outputs CKA0 or CKA1 of the Z180 serial port (available on pins 11 and 12 of header J12) to CLK/TRG2 or CLK/TRG3 (pins 6 or 8 on header J12), either another CTC or CKA0 or CKA1 can be used to drive the CTC at a slower rate. If two CTCs are cascaded, the maximum period is 1.8204 seconds. Keep in mind that the overhead for answering a CTC interrupt and extending the period with a memory counter is small.

Refer to Chapter 6, “Software Reference,” for a description of the setctc function used to initialize the CTC.

High Current / High Voltage Driver

Figure 4-15 shows a diagram of the high current/high voltage driver and outputs.

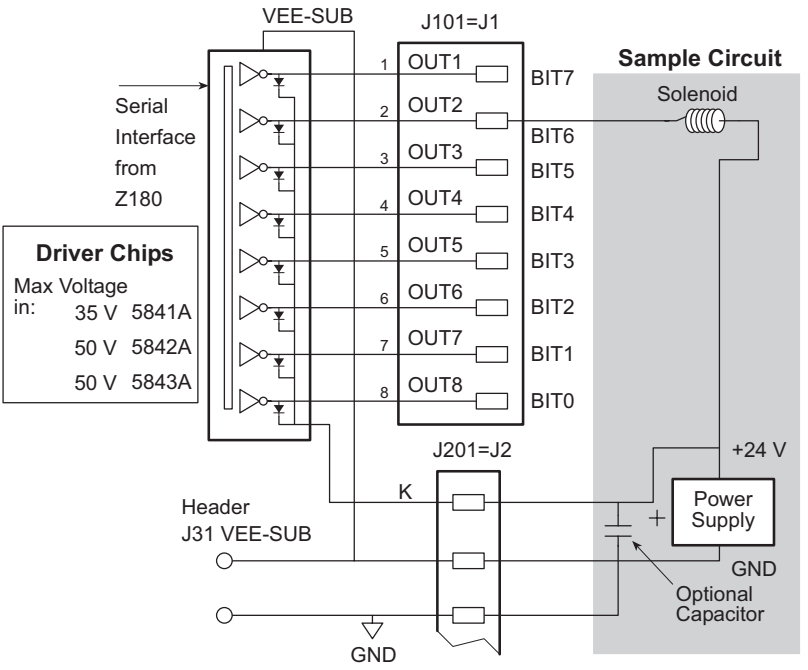


Figure 4-15. High-Current/High-Voltage Outputs

The output bits are in reverse order on headers J1 and J101.

The driver chip supplied by Z-World is the Allegro 5841. The following ICs can also be used for the high-current driver:

Driver	Breakdown	Sustained
UCN-5841A	50 V	35 V
UCN-5842A	80 V	48 V
CN-5843A	100 V	48 V

The maximum voltage allowed on a BL1100 is 48 V. Each channel is capable of sinking up to 400 mA. The maximum power dissipation allowed is 1.82 W at 25°C. Derate this by 18.2 mW for each degree above 25°C. For example, the allowed power dissipation at 70°C would be 1.00 W.

The collector to emitter saturation voltage must not exceed the following values:

Current	C-E Voltage	Power Dissipation
100 mA	1.1 V	0.11 W
200 mA	1.3 V	0.26 W
350 mA	1.6 V	0.56 W

The VEE pin can be up to 20 V negative with respect to ground, as when using split power supplies, but it is important that the ground be located between the plus and minus power supply voltage (no floating supply is allowed). This driver is designed to drive inductive loads such as solenoids or relays.

The K line drains inductive voltage excesses. If the wire connecting K to the power supply is long (inductive), place a local filter capacitor near the board to absorb the voltage surge when the device is turned off.

When driving incandescent lights, beware of the initial inrush current stressing the driver. It is not advisable to use the unregulated direct current into the board if there is any danger that the load will cause the power-fail circuitry of the BL1100 to be triggered. If unregulated input to the board is not used, be sure to take the current directly from the supply and not from the board connector. It is easy to blow out the driver chip by connecting and removing wires with the power enabled. If the protective diodes are not connected, inductive loads will ruin the chip immediately.

The high-voltage driver is not affected when a hardware reset occurs. Take special precautions if it is important to disable the high-voltage driver on system failure. For example, build an independent turn-off mechanism for equipment controlled by the high-voltage driver chip.

If the BL1100 power supply fails, then the high-voltage driver will be placed in the OFF state and remain off when power returns. If the 5841 chip fails due to stress, it can fail in the ON state, allowing current to flow. Be sure to consider the consequences of any such failure, and take appropriate protective precautions when necessary.

Liquid Crystal Display Interface

Header J6 is as an interface to standard LCDs (Figure 4-16) that use a 14-pin connector and are compatible with the Hitachi HD44780 LCD driver chip.

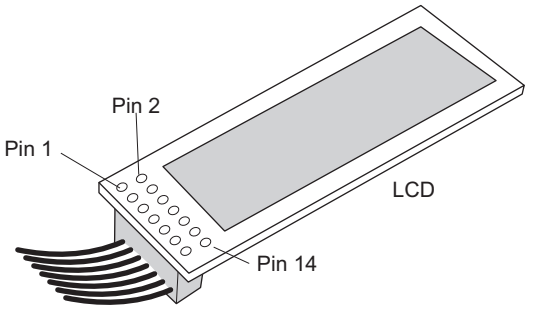


Figure 4-16. Standard LCD

The cable connects to the bottom side of LCD modules such as the Optrex DMC20481 (20 characters by 4 lines). Therefore, the pinout for header J6 is a mirror image compared to that for a standard 14-pin connector. Carefully analyze the connector pin assignments shown in Figure 4-17 before ordering an LCD unit. Units vary slightly.

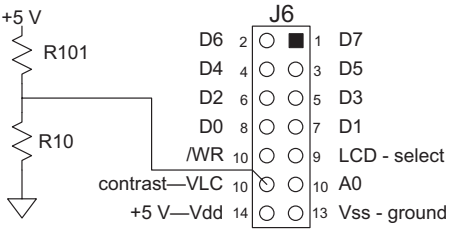


Figure 4-17. Header J6 Pinout

Tip

Z-World's OP6100, OP6200, and OP6300 operator interfaces may be connected directly to header J6. Refer to the ***OP6000 User's Manual*** for more information.

Resistors R10 and R101 create a divider to provide the contrast adjustment voltage for the LCD. Factory values (R10 open, R101 shorted) can be changed if the contrast is not satisfactory on a particular LCD. Suggested values are 1 kΩ for R10 and 22 Ω for R101. The value of R101 may then be adjusted further to produce the desired contrast.

The locations of R10 and R101 are shown in Figure 4-18.

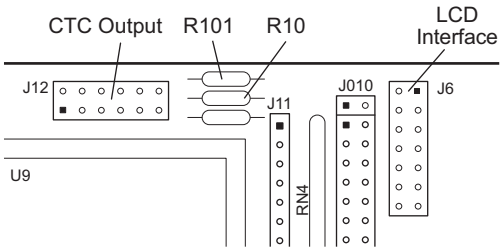


Figure 4-18. Locations of R10 and R101

The LCD driver has two registers accessed at the following I/O addresses:

62_H control register
63_H data register

Figure 4-19 shows the timing chart for the LCD.

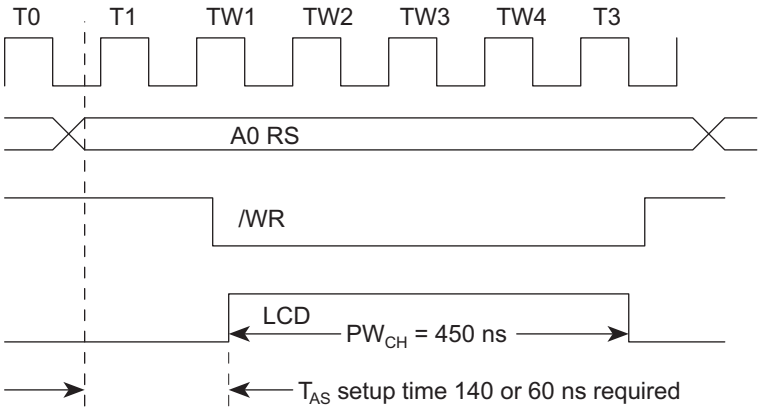


Figure 4-19. LCD Timing Chart

The BL1100 uses the Z180 “E” clock feature to generate the signal LCD. The BL1100 provides an interface to LCD controller chips that use a 6800 type of bus interface. Be sure to use sufficient wait cycles for the selected controller.

Serial Communication

The BL1100 has four built-in serial ports: two in the Z180 microprocessor and two in the SIO portion of the KIO. The Z180 ports are Port 0 and Port 1. The SIO ports are port A and Port B. Alternate names are “z0” and “z1,” and “s0” and “s1” respectively. Although similar, all four ports have different characteristics. For example, the Z180 contains UARTs, and the SIO contains USARTs. Thus, only the SIO ports support synchronous communications, and provide a greater degree of control.

ZIO Port 0 is a dedicated RS-232 port and SIO Port B is a dedicated RS-485 port. The other RS-232 and RS-485 connectors can be swapped using jumpers J13 and J14 as listed in Table 4-3 and shown in Figure 4-20.

Table 4-3. Serial Port Jumper Configurations

Routing	J13	J14
ZIO Port 0 (RS-232 on J7)	Always RS-232.	
ZIO Port 1 to RS-232* (J8)	1–4	—
ZIO Port 1 to RS-485 (J3)	3–4	2–3
SIO Port A to RS-232 (J8)	1–2	—
SIO Port A to RS-485 (J3)	2–3	1–2
SIO Port B (RS-485 on J9)	Always RS-485.	

* ZIO Port 1 becomes receive-only in this configuration.

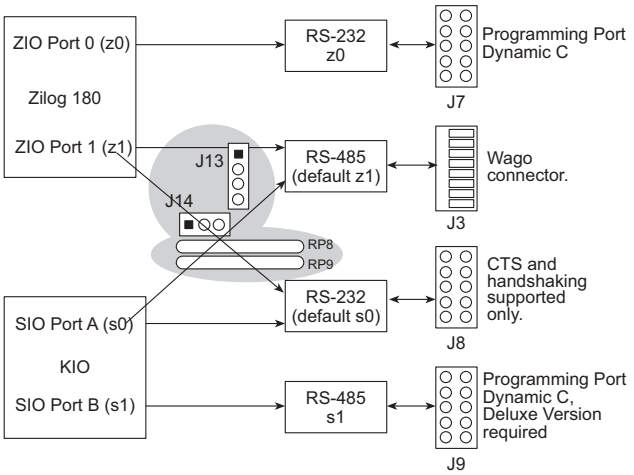


Figure 4-20. BL1100 Serial Port Configurations

Figure 4-21 shows the pinout for the headers with serial communication signals.

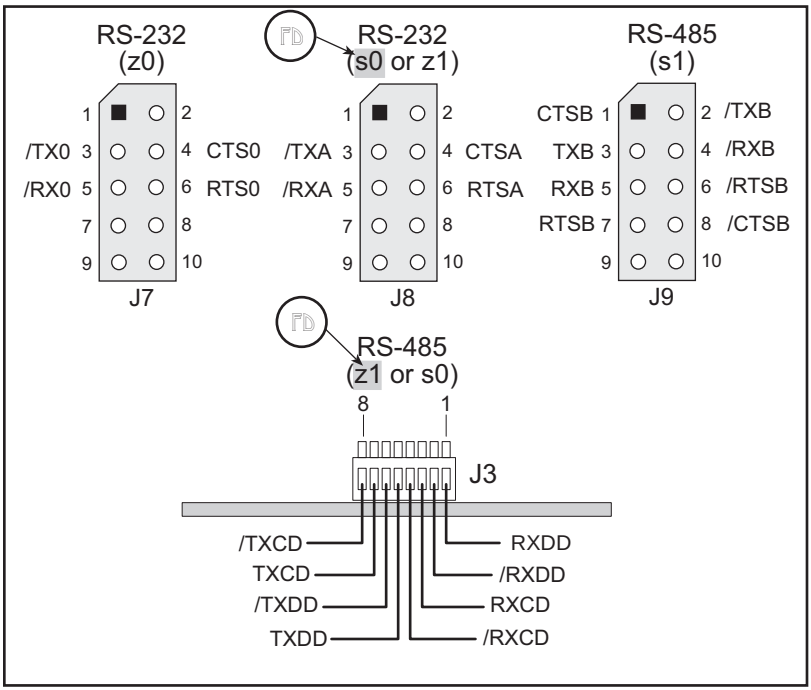


Figure 4-21. Serial Communication Pinouts

Header J7 is an RS-232 asynchronous serial port for ZIO Port 0. This port is the default programming port for the standard version of Dynamic C. When not used for programming, header J7 serves as a general-purpose serial port.

Header J3 is an RS-485 port for ZIO Port 1 or SIO Port A. Synchronous operation is possible when header J3 is connected to SIO Port A (J3 is connected, by default, to ZIO Port 1). When driven by the SIO, the synchronous clock line can also be engaged. When engaged, the clock is driven by either an onboard or an external clock. The clock line is jumpered through header J014.

Header J8 is an RS-232 port for SIO Port A or ZIO Port 1. Synchronous operation is possible when header J8 is connected to SIO Port A (J8 is connected, by default, to SIO Port A). Only CTS and RTS handshake lines are supported.

Header J9 is an RS-485 asynchronous or synchronous port for SIO Port B. This port is the high speed programming port for the deluxe version of Dynamic C. The handshaking lines are not used by Dynamic C, but can be used if the port is diverted to another purpose.

RS-232 Communication

Modem Communication

Modems and telephone lines allow RS-232 communication across great distances. A modem automatically scans character streams that are read from the receive buffer for modem commands. The RS-232 library supports communication with a Hayes Smart Modem or another compatible modem. If the modem is not compatible, you must tie the CTS, RTS, and DTR lines on the modem side together. Additionally, the CTS and RTS lines on the BL1100 side also have to be tied together. A NULL connection is also required for the TX and the RX lines. However, a commercial NULL modem already has its CTS and RTS lines tied together on both sides.

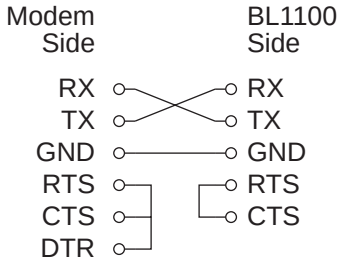


Figure 4-22. Connections Between Controller and Modem

Figure 4-22 illustrates the connections between a BL1100 and a modem.

See Z-World's Dynamic C reference manuals for details on the software functions used for modem communication.

XMODEM File Transfer

The BL1100 supports the XMODEM protocol for downloading and uploading data. Currently, the library supports downloading an array of data in multiples of 128 bytes.

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

Character echo is automatically suspended during XMODEM transfer.

RS-485 Communication

Header J3 on the BL1100 provides a full-duplex RS-485 interface, as shown in Figure 4-23. An RS-485 serial communication channel can be used to create a network of BL1100 (or other) controllers with links spanning several kilometers.

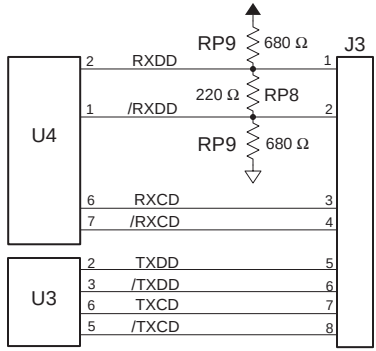


Figure 4-23. RS-485 Interface

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 Z180 Port 1, which can be configured for RS-485 communication on the BL1100 (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 the 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]
```

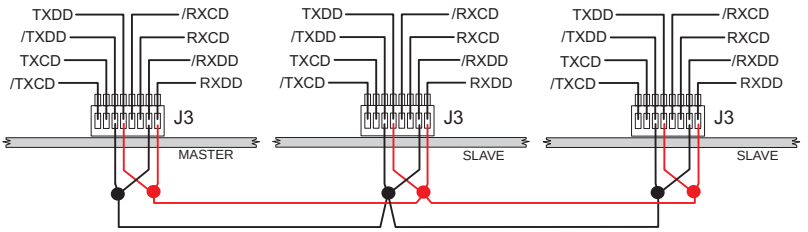
The term **len** is the length of the message that follows.

The BL1100 also supports four-wire full-duplex RS-485 networking. While there are no specific drivers for full-duplex RS-485 networks, the master may be enable with a call to **on_485** in the **DRIVERS.LIB** library. Slaves have to be turned on and off with **on_485** and **off_485** for the duration of the message they have to send.

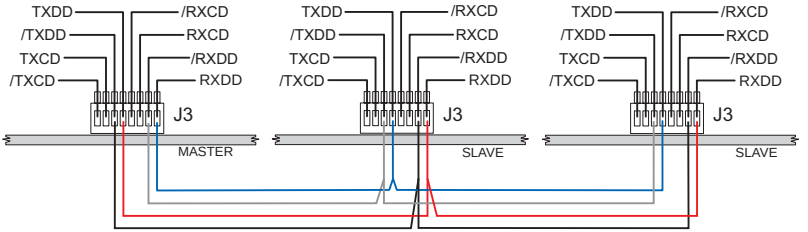
Refer to the Dynamic C manuals for more details on master/slave networking.

Hardware Connections

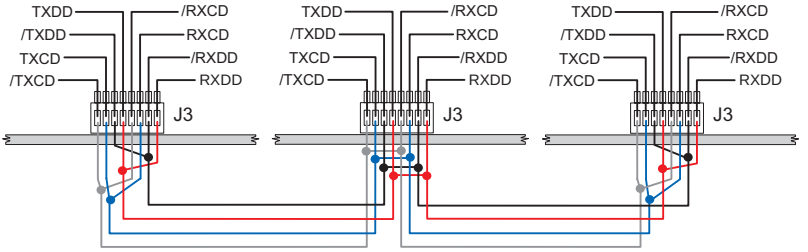
Figure 4-24(a) shows the connections for a two-wire RS-485 network, and Figures 4-24(b) and 4-24(c) show the connections for a four-wire RS-485 network. Use single twisted-pair wires (nonstranded, tinned).



(a) Two-Wire Half-Duplex RS-485 Wiring



(b) Four-Wire Full-Duplex RS-485 Wiring (Master/Slave)



(c) Four-Wire Full-Duplex RS-485 Wiring (Peer to Peer)

Figure 4-24. BL1100 RS-485 Networks

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 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. The BL1100 has 220 Ω termination resistors (RP8) and 680 Ω (RP9) bias resistors already installed. Remove the RP9 resistor packs from all the slave controllers, and remove the RP8 resistor packs from all the slave controllers except the last one.

Tip

Z-World recommends placing a 220 Ω termination resistor across the twisted-wire pair for the last slave unit instead of using RP8. This will ensure that a termination resistor remains in place if the slave is substituted.

Tip

Connect the grounds (pin 8 of Wago connector J2) of your RS-485 network to eliminate network faults resulting from the grounds being at different potentials.

Refer to National Semiconductor Application Note 847, ***FAILSAFE Biasing of Differential Buses***, for more details on the selection and use of termination and bias resistors in an RS-485 network.

Direct Programming Using Serial Ports

Z-World provides the following low-level utility functions.

```
int sysclock();
int siobaud ( int clock, int baud );
int z180baud( int clock, int baud );
```

The **sysclock** function returns the clock frequency in units of 1200 Hz as read from the EEPROM. The clock frequency is stored at location 108_H at the factory.

The baud functions return the byte to be stored in CNTLB0 / CNTLB1, considering only the bits needed to set the baud rate. Both the clock and baud rates are expressed in multiples of 1200 Hz. Thus a 9.216 MHz clock is expressed as 7680 and 19,200 bps is expressed as 16. The return value is –1 if the baud value cannot be derived from the given clock frequency.

siobaud assumes that the SIO operates at 19,200 bps (**baud** = 16)

Each of the four serial ports appear to the CPU as a set of registers. These registers can be directly accessed with the **inport** and **outport** library functions using the symbolic constants listed in Table 4-4.

Table 4-4. Serial Port Registers

Address	Name	Registers
Z180 UART		
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 0
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
SI0 USART		
48	SIODA	SIO Data Register, Channel A
49	SIOCA	SIO Control Register, Channel A
4A	SIODB	SIO Data Register, Channel B
4B	SIOCB	SIO Control Register, Channel B

The following example shows how to read and write from z0.

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

Ports may be polled or interrupt-driven. Table 4-5 lists the interrupt vectors.

Table 4-5. Serial Port Interrupt Vectors

Address	Name	Interrupt Vector
Z180 Interrupt Vectors		
0E	**SER0_VEC**	Z180 Serial Port 0 (higher priority)
10	**SER1_VEC**	Z180 Serial Port 1
SI0 Interrupt Vectors		
20	**SIOBT_VEC**	Channel B transmit buffer empty
22	**SIOBEX_VEC**	Channel B external/status change
24	**SIOBR_VEC**	Channel B receive character ready
26	**SIOBER_VEC**	Channel B special receive condition
28	**SIOAT_VEC**	Channel A transmit buffer empty
2A	**SIOAEX_VEC**	Channel A external/status change
2C	**SIOAR_VEC**	Channel A receive character ready
2E	**SIOAER_VEC**	Channel A special receive condition

Attainable Baud Rates

The serial ports built into the Z180 can generate standard baud rates when the crystal frequency is 9.216 MHz or a frequency related to this by a small multiple or small divisor, for example, 3.072 MHz, 4.608 MHz, 6.144 MHz, 9.216 MHz, or 12.288 MHz. The crystal will be stamped with twice this frequency.

The serial ports in the SIO have a clock supplied by the first two CTC (counter/timers circuit) units in the KIO peripheral chip. These counter/timers are driven at 1/2 the clock frequency. A division by at least 16 occurs in the SIO for asynchronous, self-clocked modes. The CTC can further divide the frequency by an integer from 1 to 256. Some crystals and the clock frequencies that are possible are given in Table 4-6.

Table 4-6. Crystal and Clock Frequencies

Baud Rate (bps)	Clock Frequency		Baud Rate (bps)	Clock Frequency	
	9.216 MHz	12.288 MHz		9.216 MHz	12.288 MHz
128,000		3	57,600	5	—
96,000	3	4	38,400	—	10
76,800	—	5	19,200	15	20
64,000	—	6	9600	30	40

The divisor should not be smaller than 3 to avoid violating the SIO rule that the baud clock must not be more than 1/4.5 times the system clock. A divisor of 3 gives a ratio of 6 to 1 between the baud clock and the system clock. Keep in mind that the crystal has double the system clock frequency: an 18.432 MHz crystal is installed to get a 9.216 MHz system clock.

Much higher baud rates, generally 16 times higher, can be obtained in synchronous modes or in asynchronous mode if a separate clock is used.

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.

Jumpers across pins 9–11 and 10–12 on header J12 can direct the output of counter/timer units to provide an external clock, when necessary, such as when operating on a nonstandard system clock.

Sampling techniques used to sense the external clock within the microprocessor limit the external clock speed so that it must not be faster then 1/40 the CPU clock speed.

One of the internal DMA controllers can be used in conjunction with the internal serial ports.

The serial ports have an optional multiprocessor communications feature. When enabled, the feature allows an extra bit to be included 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 bit enabled. This provides a 1-byte attention message that can be used to wake up a processor without the processor having to monitor all traffic on a shared communications link.

The block diagram in Figure 4-25 shows Serial Channel 0. Serial Channel 1 is similar, but modem control lines for /RTS1 and /DCD do not exist. The five unshaded registers shown in Figure 4-25 are accessible directly as internal I/O registers.

The serial ports can be polled or interrupt-driven.

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.

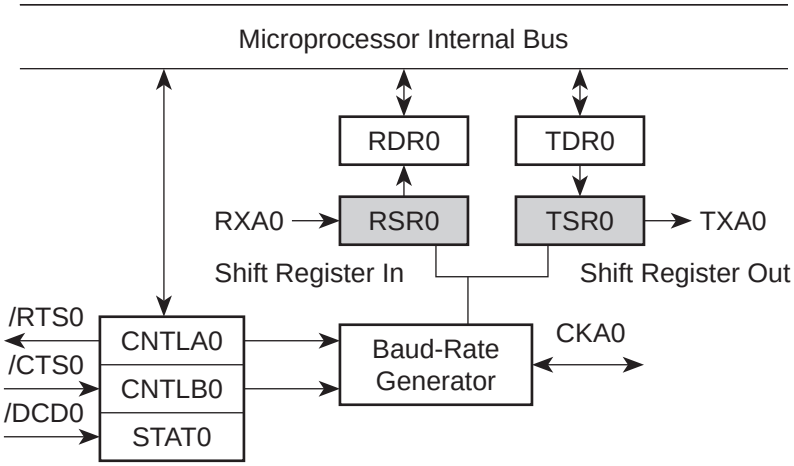


Figure 4-25. Microprocessor Internal Bus

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* method works as follows. The receiver interrupt is enabled as long as the program 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 /DCD0 is grounded on the BL1100.

Asynchronous Serial Communication Interface

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

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 / W	R	R / W

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.

/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 BL1100, 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 to 0 when you want 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 selects the pin to serve the CTS1 function. A 0 selects the RXS function. (The RXS pin is the CSIO data receive pin.) When set to the RXS function, the CTS line has no effect. It is not advisable to use the CTS1 function on the BL1100, 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 on any of the following conditions: /DCD0 (channel 0 only), RDRF (receiver data register full), OVRN (overrun), PE (parity error), FE (framing error). The condition causing the interrupt must be removed before 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 always set when one of the error flags is set, in which case defective data are 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 serial channel operation.

CNTLA0 (00H)

7	6	5	4	3	2	1	0
MPE	RE	TE	/RTSO	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/ –TEND0): 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 RDR, 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 4-7.

Table 4-7. Baud Rate Divide Ratios for Source/Speed Select Bits

SS2	SS1	SS0	Divide Ratio
0	0	0	÷ 1
0	0	1	÷ 2
0	1	0	÷ 4
0	1	1	÷ 8
1	0	0	÷ 16
1	0	1	÷ 32
1	1	0	÷ 64
1	1	1	external clock

The prescaler (PS), the divide ratio (DR), and the SS bits form a baud-rate generator.

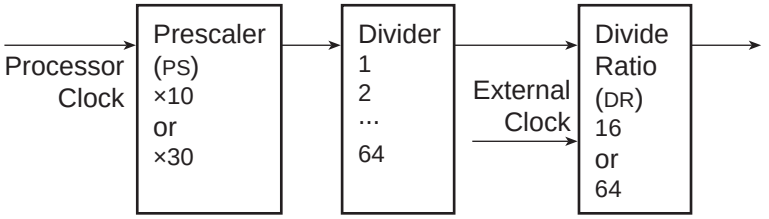


Figure 4-3. Baud-Rate Generator

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, the baud rate prescaler 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, and 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). The MPB is 1 when MPBT is 1, and 0 when MPBT is 0. When the MPB is 1, transmitted bytes will get the attention of other units listening only for bytes with MPB set.

SIO Serial Ports

SIO channel B has three read registers while channel A has only two. Channel B has eight write registers while channel A has only seven. In both cases, channel A lacks an interrupt vector register.

To read a register, place the register number in WR0. The next read cycle will place the contents of the read register on the data bus.

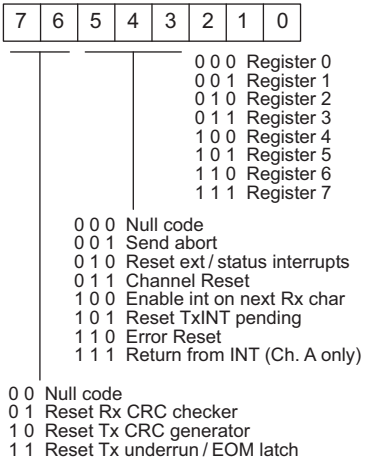


Figure 4-27. SIO Write Register 0

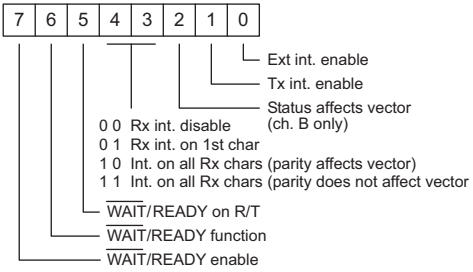


Figure 4-28. SIO Write Register 2

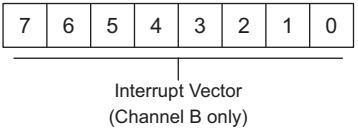


Figure 4-29. SIO Write Register 2

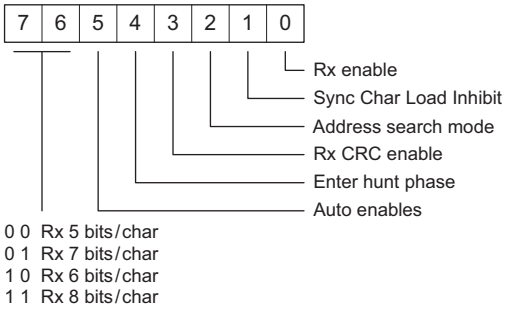


Figure 4-30. SIO Write Register 3

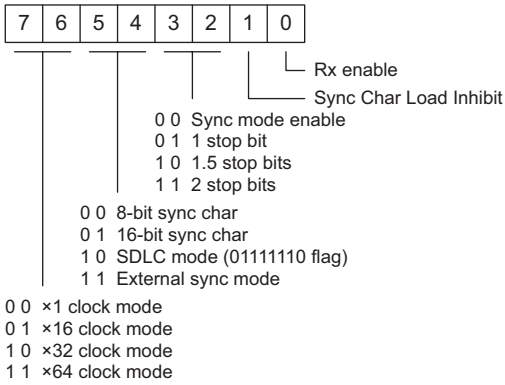


Figure 4-31. SIO Write Register 4

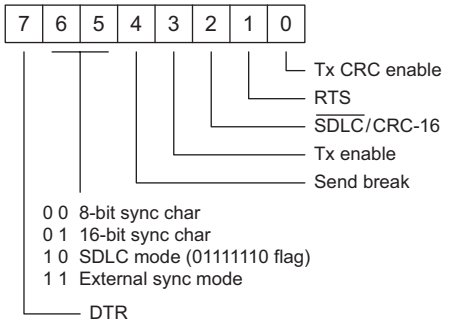


Figure 4-32. SIO Write Register 5

SIO write registers 6 and 7 hold SDLC sync bytes.

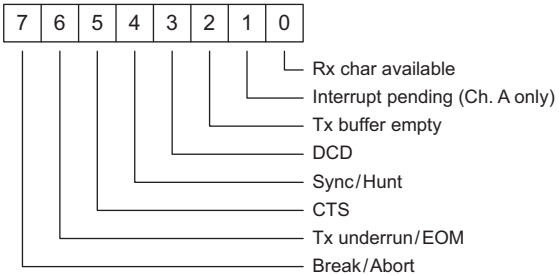


Figure 4-33. SIO Read Register 0

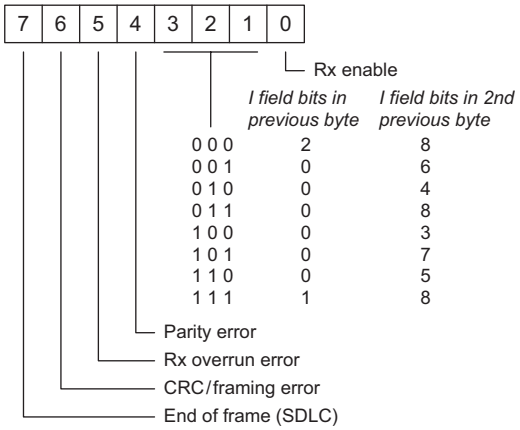


Figure 4-34. SIO Read Register 1

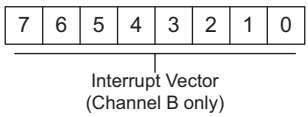


Figure 4-35. SIO Read Register 2

The interrupt vector registers are automatically set from the Dynamic C EPROM when the BL1100 is booted. Write registers 6 and 7 are used to set the sync words for synchronous communications. (Set these to 0x7E for SDLC.)

Certain SIO lines are shared with other functions on the BL1100. The SIO line /DCDB is used to input the voltage low signal (low when voltage is low). The SIO line /DTRB is used to hit the watchdog timer and is manipulated by the function **`hitwd`**. If the RS-485 communication mode

is selected for Dynamic C programming, then SIO channel B is used for communication and cannot be used in the program being developed. SIO Channel B is also connected so that the NMI can be used instead of the regular mode 2 interrupt. This type of interrupt is used by Dynamic C in the RS-485 communication mode, and allows the connection to be maintained even if the target program disables interrupts for long periods. In this case, the NMI is shared for communications and power-fail warning. When a NMI occurs, the interrupt routine determines whether the cause of the interrupt was a power failure by testing the /DCDB line to see if power fail is activated.

If the Dynamic C 485 mode is not enabled, then all nonmaskable interrupts are transferred to the programmer's NMI routine as defined by

```
#JUMP_VEC NMI_VEC name
```

Otherwise, only power-fail interrupts are dispatched to the programmer's NMI service routine.

Since the **hitwd** function shares channel B with any other driver, a global location **s0_wreg5** holds the contents of SIO Write Register 5 so that **hitwd** will not change any bits other than the DTR bit in this register.

If you decide to write a driver for SIO channel B using nonmaskable interrupts, be aware that the NMI line into the microprocessor, once set, is latched until the condition causing the NMI is removed. Then a memory-write cycle occurs. This prevents multiple NMIs when reading the SIO registers.

The /WAIT/READY line from Channel A is connected to the DREQ1 line of the Z180 microprocessor. This allows the DMA unit in the Z180 to perform high-speed input or output from SIO Channel A.

Programming the SIO in Asynchronous Mode

The three write registers (WR_3, WR_4 and WR_5) are set in the following configurations:

WR_3 *x x x* 0 0 0 0 *x*
WR_4 *x x* 0 0 *x x x x*
WR_5 *x x x x x* 0 *x* 0

The bits shown as *"x"* must be set. The bits shown as "0" are used only for the synchronous mode. Use the descriptions in the previous section to decide how to set individual bits. WR_2 is always set on initialization. WR_6 and WR_7 are used for the synchronous mode. In most cases, settings for write registers are as shown in the following configurations:

WR_3 1 1 0 0 0 0 0 *x*	*x* RX enable
WR_4 0 1 0 0 0 1 0 0	×16 clock, 1 stop bit, 8 data
WR_5 1 1 1 0 *x* 0 1 0	*x* TX enable

The CTC that corresponds to the SIO unit—CTC1 for SIO Channel A and CTC0 for SIO Channel B—must also be initialized. The CTC unit takes its input from the system clock divided by two. This clock is then further divided by the CTC and used to provide both receive and transmit clocks to the SIO. Header J014 can be used to configure Channel A to take its receive clock from an external source via pins 3–4 of Wago connector J3. Figure 4-36 shows the jumper configurations for header J014. When an external clock is used, ×1 mode can be used for the asynchronous mode. Synchronous modes can also be used.

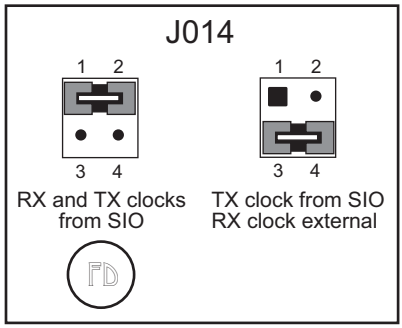


Figure 4-36. Header J014 Jumper Configurations

CHAPTER 5: ANALOG SECTION

Chapter 5 provides information on the BL1100 A/D converter and analog I/O.

Analog Inputs

The analog input system consists of an 8-channel 10-bit A/D converter and a signal-conditioning amplifier for each channel. An optional factory-installed 12-bit A/D converter is available. Seven of the input channels (0–6) are available for general use. The eighth channel (channel 7) is the onboard temperature sensor that is built into the voltage reference.

Signal Conditioning

Figure 5-1 shows a schematic of the signal-conditioning amplifier.

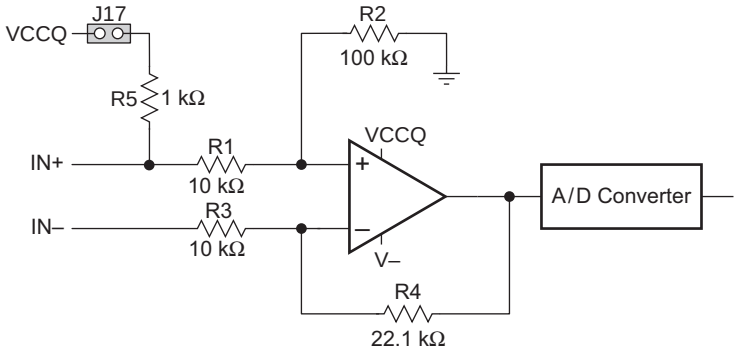


Figure 5-1. Signal-Conditioning Amplifier

The following five configurations can be achieved by installing or changing resistors R_1, R_2, R_3, R_4, and R_5. The gain is given by Equation (4-1).

$$g = \frac{\text{IN+}}{\text{IN-}} = \frac{R_2}{R_1 + R_2} \div \frac{R_3}{R_3 + R_4} = \frac{R_2(R_3 + R_4)}{R_3(R_1 + R_2)} \qquad (5\text{-}1)$$

Configuration 1

The channel is configured to be a differential amplifier when $R_1 = R_3$, $R_2 = R_4$, and R_5 is missing. The gain is given by

$$g = \frac{R_4}{R_3} = \frac{R_2}{R_1} \quad . \qquad (5\text{-}2)$$

A differential amplifier reports the difference in voltage between the input terminals and is insensitive to voltage swings in which both inputs participate equally.

Configuration 2

The channel is configured to be a noninverting single-ended amplifier when R_5 is missing. Input IN– is connected to ground. Input IN+ is sampled. R_2 is either missing or is a large value such as 100 kΩ. The gain is given by

$$g = \frac{R_3 + R_4}{R_3} = \frac{R_4}{R_3} + 1 \quad . \tag{5-3}$$

The input impedance is very high when R2 is missing. Otherwise, the input impedance is $R_1 + R_2$.

Configuration 3

The channel is an inverting single-ended amplifier when R_5 is missing. Input IN– is sampled and input IN+ is connected to ground. The gain is given by

$$g = \frac{R_4}{R_3} \quad . \tag{5-4}$$

R_2 can be missing without affecting the operation. If R_2 is present, input IN+ may be left unconnected.

Configuration 4

The channel measures resistance, where *R* is the unknown resistance across IN+ and IN–. Input IN– is connected to ground. *R* is given by

$$R = R_5\left(\frac{REF - V}{V}\right) \quad , \tag{5-5}$$

where V is the voltage at IN+. The gain from the input IN+ pin is the same as for Configuration 2 above. The reference (REF) can be the 2.5 V precision reference U6 (LT1019) or the +5 V power supply if less accuracy is needed. The ability of the reference to deliver current is limited to approximately 8 mA. Thus, the +5 V power supply or an external source must be used for low resistances. If +5 V is used, another channel can be used to track the voltage level, which is only regulated to a few percent.

Configuration 5

The channel measures current, where *R*, the resistance placed between IN+ and IN– is known, and the current passing through *R*, which is given by $i = V/R$, will be measured. The setup is similar to Configuration 4.

Installing Components

Configurations 1–6 can be achieved by installing or changing resistors and a capacitor on the BL1000 board. Figure 5-2 shows their locations.

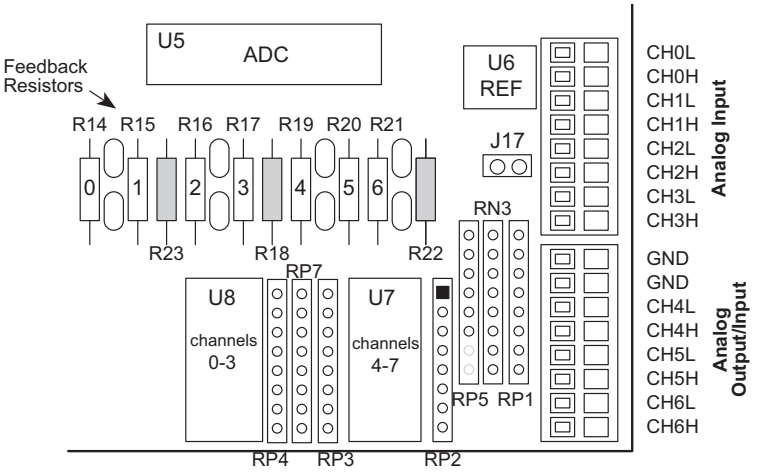


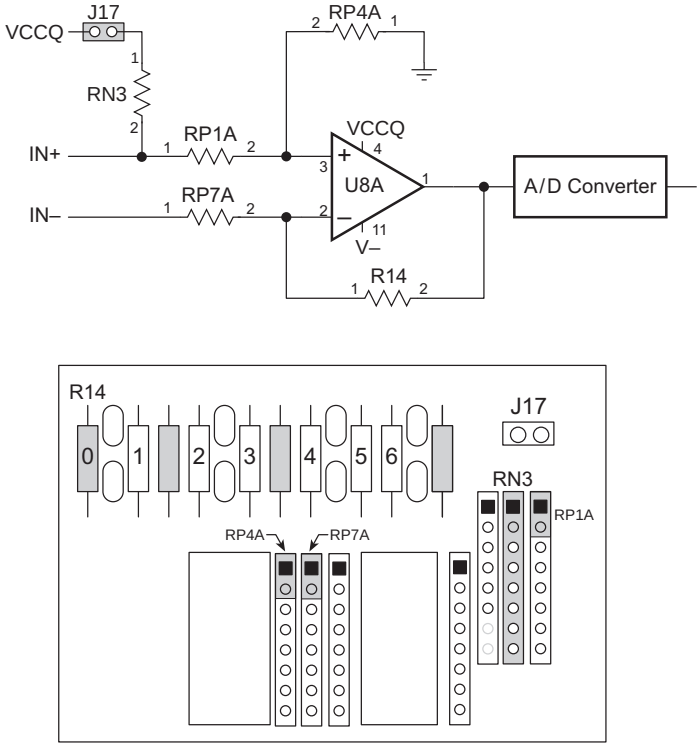
Figure 5-2. Locations of Components Used to Configure BL1100 A/D Converter

Resistor packs, networks and individual resistors can be installed in the sockets in the board.

Figure 5-3 shows the corresponding components for the seven analog input channels.

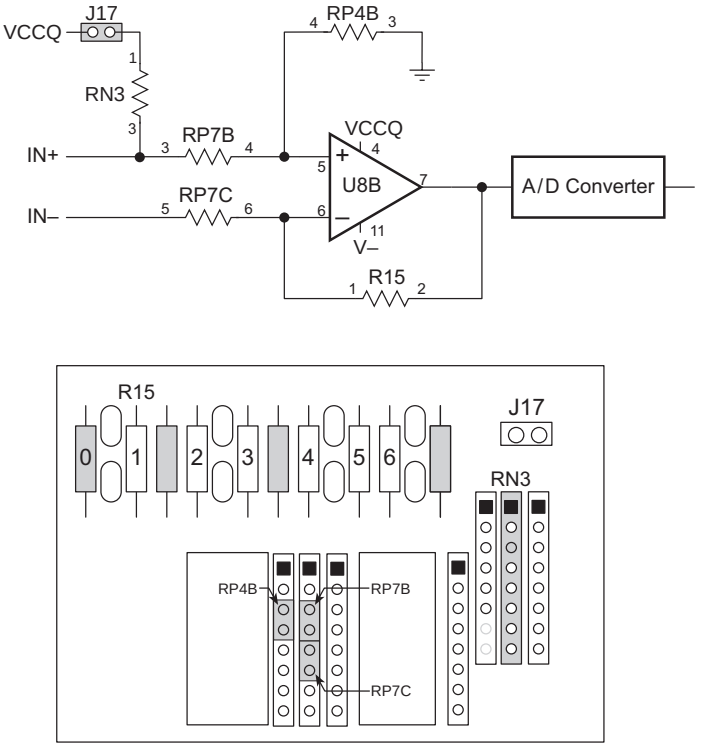
Resistor network RN3 is not factory installed. In a resistor network, note that one pin is common to the network. For practical reasons, the RN3 socket can then accommodate either a resistor network for all the channels, or an excitation resistor for one channel only connected between pin 1 and the pin corresponding to the channel as shown in Figure 5-3. The jumper must be installed on header J17 when RN3 is being used to connect the excitation resistors to VCC.

22.1 kΩ factory-default resistors are installed for R14–R17 and for R19–R21. The individual resistors in resistor packs RP1, RP2, RP3, and RP7 are 10 kΩ. The individual resistors in resistor packs RP4 and RP5 are 100 kΩ.



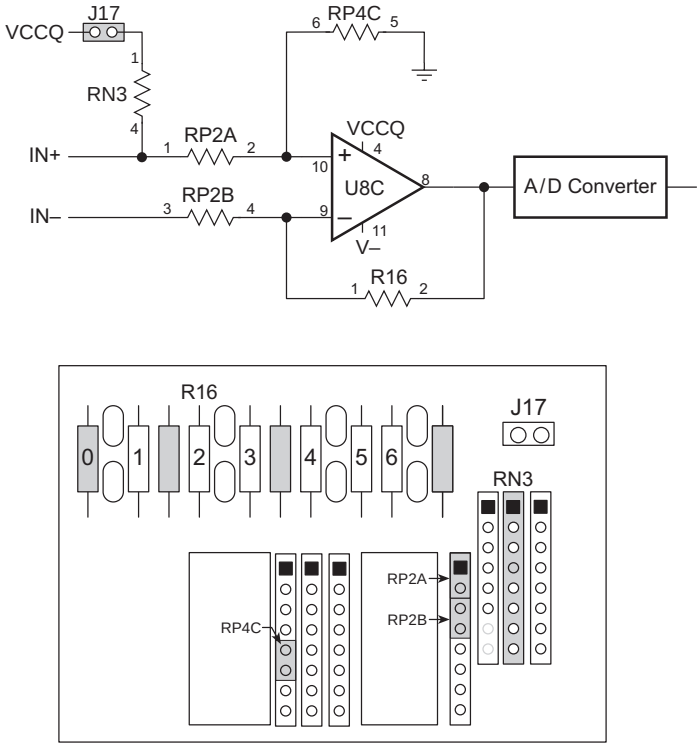
(a) Channel 0

Figure 5-3. Analog Input Channel Configuration



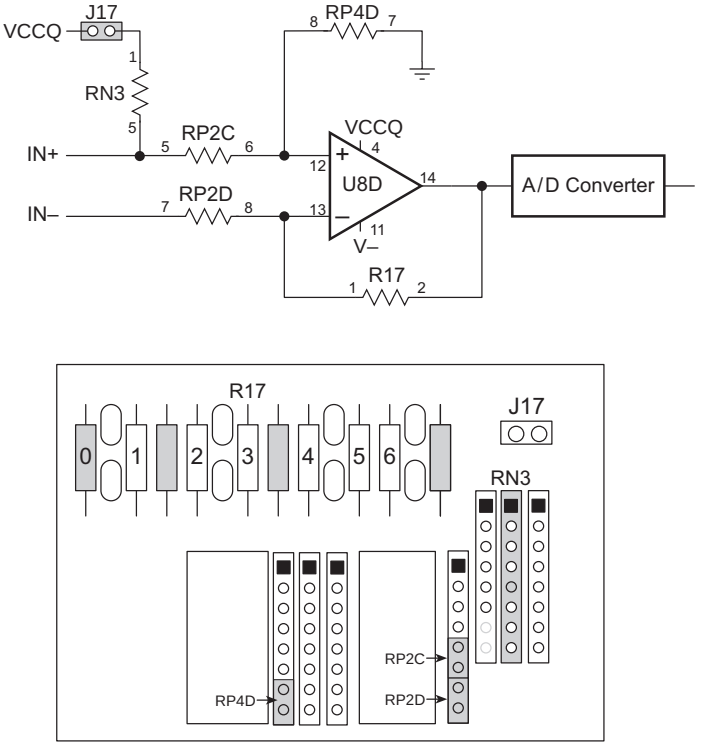
(b) Channel 1

Figure 5-3. Analog Input Channel Configuration



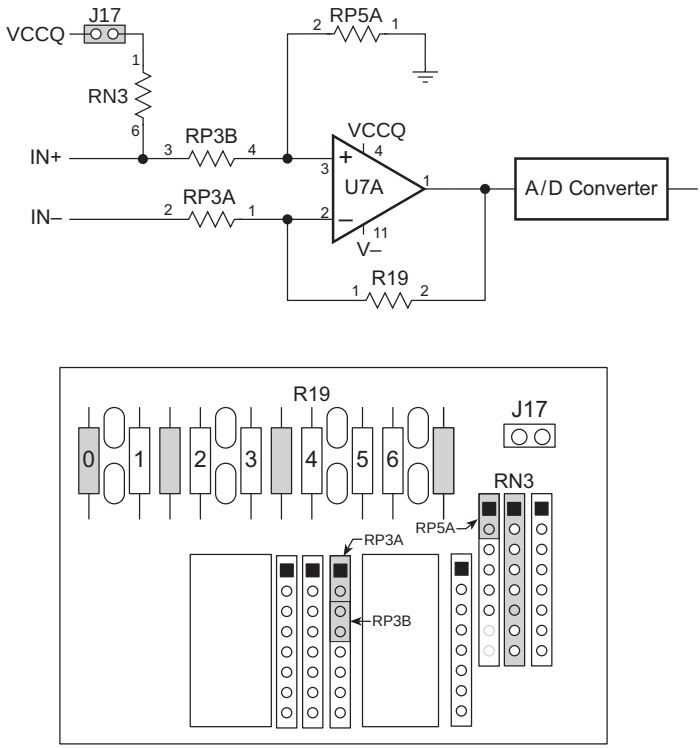
(c) Channel 2

Figure 5-3. Analog Input Channel Configuration



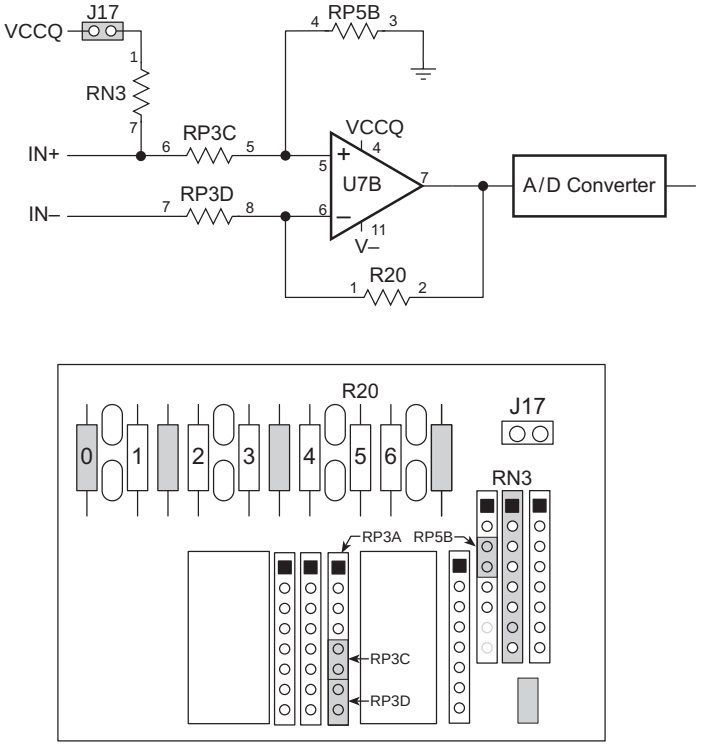
(d) Channel 3

Figure 5-3. Analog Input Channel Configuration



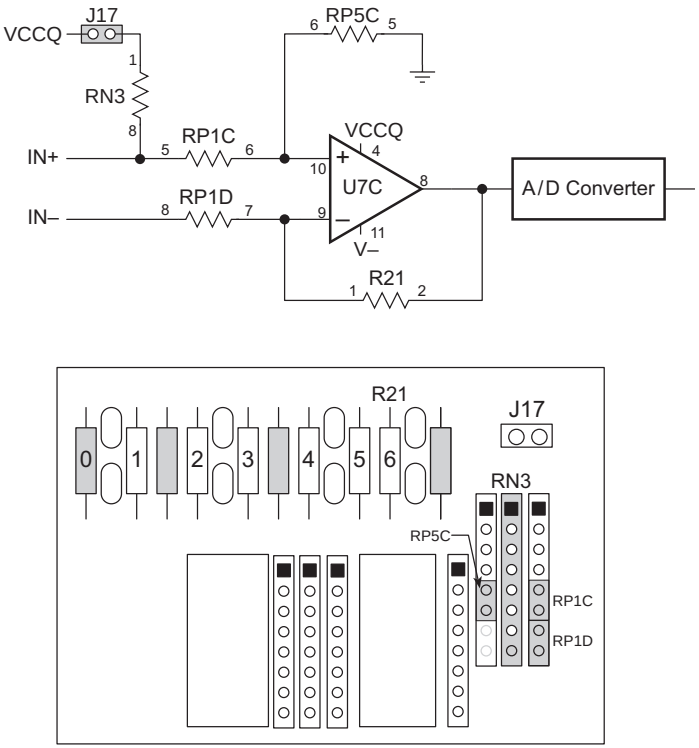
(e) Channel 4

Figure 5-3. Analog Input Channel Configuration



(f) Channel 5

Figure 5-3. Analog Input Channel Configuration



(g) Channel 6

Figure 5-3. Analog Input Channel Configuration

A/D Conversion

The eight amplifier channels feed into the A/D converter. Channel 7 is used to measure the temperature signal generated by the LTC1019 reference chip at U6. Channel 6 can also be used as a buffer to provide an external reference voltage.

To use Channel 6 as a buffer, the J5 Wago connector must be replaced with a header.

The A/D converter is a single-chip Linear Technology LTC1094 (10 bits at 120 µs) or LTC1294 (12 bits at 120 µs).

Sources of Error

The operational amplifiers (U7 and U8) used in the analog input section can be either a National Semiconductor LM324 or, for more precision, a Linear Technology LT1014. Table 5-1 compares the performance of these two op-amps.

Table 5-1. Op-Amp Specifications

Parameter	Amplifier	
	LM324	LT1014
Maximum Input Offset (µV)	7000	250
Maximum Input Offset Drift (µV/°C)	30	2

The LM324 amplifiers will serve many purposes. However, the better-quality LT1014 amplifiers are needed for measuring low voltages, such as those associated with thermocouples.

The input offset appears as a nonexistent voltage amplified by the gain of the amplifier. This offset is tested and recorded in the EEPROM for each amplifier, and can be used to correct the measurements. However, temperature drift gives an additional offset that is difficult to correct.

The LT1019, used by the A/D as an absolute voltage reference, has a maximum error of 0.2 percent (0.05 percent for the LT1019A). There are small errors in the reference voltage related to temperature and load.

The A/D circuitry has errors relating to linearity, offset, noise, drift, and gain amounting to a few least significant bits. The offset and gain can be compensated easily. Noise can be minimized by averaging many measurements. Averaging four times as many measurements cuts the noise in half. Linearity and drift errors are more difficult to correct, but should be no more than one-half of the least-significant bit.

Onboard Temperature Sensor

The LT1019 voltage reference at U6 on the BL1100 board generates an output voltage proportional to the temperature of its internal silicon die. This voltage is related to the temperature by the approximate relationship

$$V = 0.0021 \times T \ , \tag{5-6}$$

where V is the voltage in volts and T is the temperature in kelvins [T(K) = T(°C) + 273]. The temperature-dependent voltage is connected to analog input channel 7 as shown in Figure 5-4.

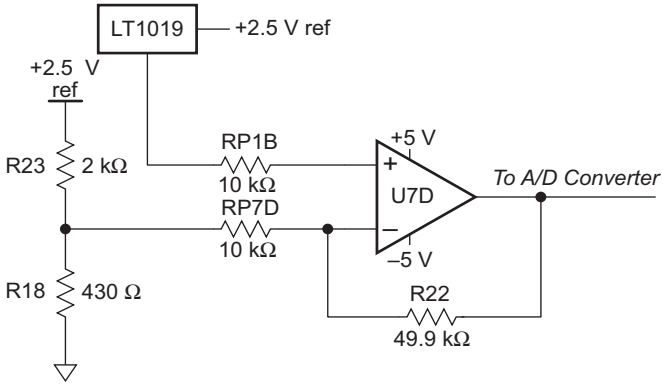


Figure 5-4. Signal Conditioning Circuit for Onboard Temperature Measurement

The equivalent voltage for the resistor divider is given by

$$V_E = V_{REF}\left(\frac{R18}{R18+R23}\right) \ . \tag{5-7}$$

Substituting the values shown in Figure 5-4,

$$V_E = (2.5\ V) \times [(430\ \Omega)/(430\ \Omega + 2000\ \Omega)] = 0.442\ V \ .$$

The equivalent voltage for the resistor divider is given by

$$R_E = \frac{1}{\left(\frac{1}{R18}+\frac{1}{R23}\right)} \ . \tag{5-8}$$

Substituting the values shown in Figure 5-4,

$$R_E = 1/[(1/430\ \Omega) + (1/2000\ \Omega)] = 354\ \Omega \ .$$

Both op-amp inputs will be at the voltage V_{TEMP}. The current through R_E and RP7D must equal the current through R22. This leads to the equation

$$\left(\frac{V_{TEMP} - 0.442\ V}{10354\ \Omega}\right) = \left(\frac{V_{OUT} - V_{TEMP}}{50000\ \Omega}\right)$$

or

$$V_{TEMP} = 0.172 \times V_{OUT} + 0.366\ V\ . \qquad (5\text{-}9)$$

Typically, the LT1019 heats itself about 2°C to 6°C above the ambient temperature, depending on the size of the load driven by the reference output. The entire enclosure containing the BL1100 may be additionally heated, so caution must be used in interpreting the temperature calculated. The actual temperature in degrees Celsius can then be calculated from

$$T = \left(\frac{V_{TEMP}}{T_{COEF}}\right) - \text{offset} - 273\ . \qquad (5\text{-}10)$$

The offset is typically 4°C to allow for self-heating and T_{COEF} is stored in the EEPROM as a calibration value for each particular LT1019 temperature reference chip. T_{COEF} is based on a voltage coefficient of 0.0021 V/°C.

An error in the temperature is an error in the coefficient and not a temperature offset, except for the offset due to self-heating.

The LT1019 temperature sensor can be used to record temperature over time, to measure ambient temperature, or to measure the cold-junction temperature in thermocouples. The temperature sensor is placed close to the field wiring connectors to minimize the thermal path between the chip and the cold junctions.

Appendix G provides sample applications for the analog inputs and A/D conversion.

Analog Output

An optional Maxim AD7543 D/A converter (DAC) chip may be installed on the BL1100 in socket U12. A Maxim OP07 op-amp buffer must also be installed in socket U19, and a National LM285 voltage reference must be installed in socket Z1. Figure 5-5 illustrates the locations of these components.

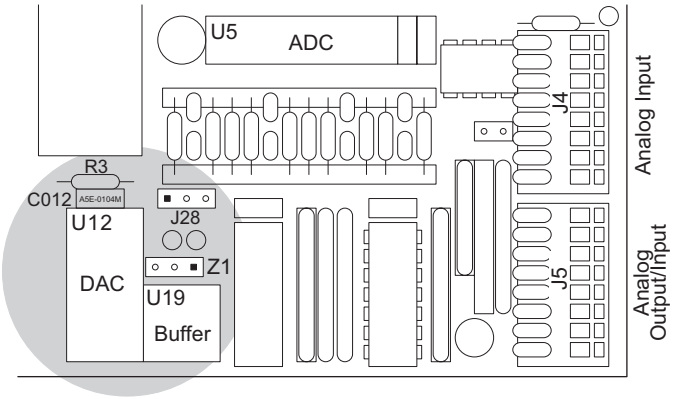


Figure 5-5. Locations of DAC Components

The DAC output is routed through header J28 to connector J5 pin 1. If J28 pins 1–2 are jumpered, then the board is set up for DAC output of up to 2 mA at 0 V to 2.5 V on Wago connector J5. If J28 pins 2–3 are jumpered, then J5 pin 1 will be a ground and the DAC output will not be sent to J5. Figure 5-6 summarizes these configurations.

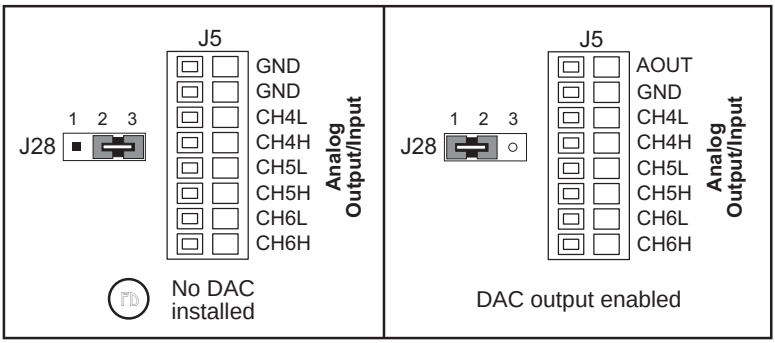


Figure 5-6. Header J28 Jumper Settings

CHAPTER 6: SOFTWARE REFERENCE

Chapter 6 describes the software functions available for the BL1100. This chapter covers the following topics.

- Supplied Software
- Digital Interfaces
- A/D Converter
- D/A converter
- Miscellaneous Drivers

Supplied Software

Software drivers for controlling the BL1100's inputs/outputs are provided with Dynamic C. The library **BL11XX.LIB** provides drivers specific to the BL1100. The **BL11XX.LIB** library is automatically included when a Dynamic C connection is initiated to a BL1100 in program mode. For the other libraries, it is necessary to include the appropriate Dynamic C libraries in your program. Table 6-1 lists the libraries for the BL1100.

Table 6-1. BL1100 Software Libraries

Library	Application
AASC.LIB	All BL1100 serial communication applications
AASCSIOA.LIB	AASC drivers for SIO Port A
AASCZ0.LIB	AASC drivers for Z180 ZIO Port 0
AASCZ1.LIB	AASC drivers for Z180 ZIO Port 1
BL11XX.LIB	All BL1100 applications
NETWORK.LIB	Utilities for half-duplex RS-485 communication
S0232.LIB	RS-232 drivers for SIO Port A
S1232.LIB	RS-232 drivers for SIO Port B
Z0232.LIB	RS-232 drivers for Z180 ZIO Port 0
Z1232.LIB	RS-232 drivers for Z180 ZIO Port 1

An AASC library does not exist at this time for SIO Port B. The **AASCSIOA.LIB** library may be used be replacing A with B.

Either the AASC libraries or the libraries listed after BL11XX.LIB in Table 6-1 need to be included in your program if the serial ports are bein used. To include these libraries, use the **#use** directive as shown below.

```
#use aasc.lib
```

See the ***Dynamic C Technical Reference*** and the ***Dynamic C Function Reference*** manuals for more information on **#use** and other directives, as well as the serial communication libraries.

Digital Interfaces

KIO Counter/Timer Circuit (CTC)

The following function initializes the CTC.

- **`void setctc( char ctc, char mode, char timer, char intr )`**

  PARAMETERS: **`ctc`** is the CTC unit (0–3)

  `mode` 0—timer mode **`sysclock`**/16, triggers immediately
  1—timer mode **`sysclock`**/256, triggers immediately
  2—counter runs off external clock
  4—timer mode **`sysclock`**/16 triggers on rising edge of /CLK
  5—timer mode **`sysclock`**/256 triggers on rising edge of /CLK
  6—timer mode **`sysclock`**/16 triggers on falling edge of /CLK
  7—timer mode **`sysclock`**/256 triggers on falling edge of /CLK

  `timer` is the timer reload value (0–255)

  `intr` 0—disables interrupt
  1—enables interrupt

  LIBRARY: **`BL11XX.LIB`**

The following code could be used to start the timer and enter an interrupt routine every 5 ms.

```
#INT_VEC CTC2_VEC myroutine
setctc( 2,1,180,1 );  // every 5 milliseconds
```

The sanple program **`CTCDEMO.C`** demonstrates the operation of the CTC and the **`setctc`** function.

The following interrupt service routine will divide the period by an additional factor of up to 256. The routine executes in about 10 µs.

```
char timer;

#INT_VEC CTC2_VEC ctc_interrupt
  #asm

ctc_interrupt::
      push af
      push hl
      ld   hl,timer
      dec  (hl)
      jr   z,ctc_2
      pop  hl
      pop  af
      ei
      reti

ctd_2:
      continue interrupt routine
   #endasm
```

High-Current/High-Voltage Driver

The following functions provide access to the 5841A high-voltage driver. Remember that the order of the output bits (on Wago connector J101) is reversed. OUT8 is bit 0 (its mask is 0x01) and OUT1 is bit 7 (0x80).

- **`void hv_wr( char value )`**

  Writes the byte **value** to the driver. The output enable remains the same. Note that all 8 bits are strobed to the output "register" in one clock, so all bits change simultaneously. A 1 enables the corresponding output (pulls low). A 0 disables the corresponding output.

  LIBRARY: **`DRIVERS.LIB`**

  This function uses the Z180 CSI/O serial interface to transmit one byte to the high-voltage driver chip. It can conflict with drivers for the A/D converter if either is used in interrupt routines at different priority level.

- **`void hv_enb()`**

  Enables the high-voltage driver chip.

  LIBRARY: **`DRIVERS.LIB`**

- **`void hv_dis()`**

  Disable the high-voltage driver chip.

LIBRARY: **`DRIVERS.LIB`**

Liquid Crystal Display Interface

The following functions from the **DRIVERS.LIB** library drive a 4 ×20 LCD such as the Optrex DMC20481.

- **void lcd_init (int mode)**

  Initializes the display. Under normal circumstances, **mode** should be 0x18.

- **int lputc(int ch)**

  Sends one character to the display. Special characters can be sent to control the display. These characters all have bit 7 set to 1.

  '\xF0' **clear line 1**

  '\xF1' **clear line 2**

  '\xF2' **cursor off, stop cursor blink**

  '\xF3' **cursor on**

  '\xF4' **cursor to blinking mode**

  '\xF5' **shift display left**

  '\xF6' **shift display right**

  '\x80' **codes 80, 81, etc. position cursor at column 0, 1, etc. on line 0**

  '\xC0' **codes C0, C1, etc. position cursor at column 0, 1, etc. on line 1**

  '\n' **position cursor to first column of second line**

- **char *lputs(char *string)**

  Sends a null-terminated string to the LCD.

- **int lprintf(char *,...)**

  lprintf() is the same as **printf** with output to the LCD driver.

These functions can be easily modified for displays with other formats such as 4 × 40 characters.

A/D Converter

The following functions work for either the 10-bit converter or the 12-bit converter as noted.

- **`int ad_rd( int channel )`**

  Reads the 10-bit LTC1094 by calling **`ad_rd10`**.

- **`int ad_rd10 (int channel)`**

  Reads the 10-bit LTC1094 A/D converter.

  PARAMETERS: The low three bits of **`channel`** specify the channel number (0–7); the fourth bit must be 0 for bipolar mode, or 1 for unipolar mode. For the unipolar mode, add 8 to the channel number.

  RETURN VALUE: The return value is shifted left by 2 bits, so it appears as a 12-bit number.

  - The function returns a number from –2048 to +2047 for the bipolar mode. This is the output from the A/D converter and corresponds to a range of –2.5 V to +2.499 V. The least two bits of the returned value are always zero.
  - In unipolar mode, the function returns a number in the range from 0 to 4095, corresponding to a range of 0 V to 2.499 V.
  - The function returns –32768 if an error occurs.

  LIBRARY: **`BL11XX.LIB`**

- **`int ad_rd12( int channel )`**

  Reads the 12-bit LTC1294 A/D converter. The function is basically the same as **`ad_rd10`**, but it applies to the 12-bit converter and the least two bits of the return result have value.

Approximately 120 µs are required to acquire a sample from the LTC1094 and LTC1294 chips. Interrupts are disabled during the sampling unless no interrupts are defined with **`#define NODISINT`**.

Temperature Measurements

The sample program **`BDTEMP.C`** in the Dynamic C **`SAMPLES`** subdirectory provides functions for testing the type of A/D converter, setting temperature coefficient, and reading the temrperature from the onboard temperature reference. The following functions are used and defined in **`BDTEMP.C`**.

- **`int bdtemp()`**

  Returns the temperature in degrees Celsius:

- **`float bdtemp_init()`**

  Must be called before **bdtemp** can be called.

- **`bdtemp`**

  Returns the temperature in units of 0.1°C. If **bdtemp** returns that the coefficient in EEPROM is bad, **bdtemp** takes an error exit.

- **`settemp()`**

  Sets the EEPROM coefficient.

- **`setad`**

  Tries to determine the type of A/D converter on board based on characteristic return values from reading the temperature sensor.

High-Speed Sampling

The following function collects samples using the 10-bit LTC1094 A/D converter at uniform intervals at sampling rates up to 12,800 samples per second.

- **`int ad_rd10s( int chan, int count, int *buf, unsigned int divider )`**

  Samples data from the Little Giant A/D converter at uniform intervals. This function uses Z180 programmable reload timer (PRT) number 1. Interrupts are disabled for the entire time the buffer of samples is being acquired, unless **`#define NODISINT`** is defined.

  PARAMETERS: **chan** is the channel number (0–7), plus 8 for the unipolar mode (otherwise the bipolar mode is assumed).

  `count` specifies the number of samples to collect.

  `buf` points to a buffer where data is stored.

  `divider` determines the sample rate based on sample rate = **`clock`**/(20 × **`divider`**). **`divider`** should not be smaller than 36, which yields 12,800 samples per second with a 9.216 MHz clock.

  RETURN VALUE: 1 for no error, or 0 indicating that a sample was missed. The function can miss a sample if the divider is too small or an interrupt occurs during sampling. Interrupts during sampling will disrupt the uniformity of the sampling.

  LIBRARY: **`BL11XX.LIB`**

D/A Converter

- **`int wdac( int value )`**

  Writes a 12-bit digital value to the DAC.

  PARAMETER: **`value`** is between 0 and 4095 for output voltages between 0 V and 2.499 V (nominal). For accurate work, apply a software gain adjustment since the voltage reference is subject to up to 80 counts of error and the DAC can have as many as 15 counts gain error. With correction, the DAC should operate with 1–2 bit accuracy over range and temperature.

Miscellaneous Drivers

Time/Date Clock

The battery-backed real-time clock is based on the Epson 72421 chip. The 72421A is accurate to approximately one second per day. The 73421B is accurate to approximately five seconds per day. Time values are resolved to one second and extend up to 80 years in the future. A Dynamic C sample program, **`SETCLOCK.C`**, is provided to read and write from/to the clock chip. The lithium battery should keep the clock going for about 10 years unless the board is stored at high temperatures for long periods with the power off.

Time/Date Functions

Date/time functions can be found in **`DRIVERS.LIB`**. Also, the sample program **`SETCLOCK.C`** provides a keyboard interface to display and set the date/time clock.

The following structure is defined to hold the date and time.

```
struct {
  byte tm_sec;        // seconds 0-59
  byte tm_min;        // minutes 0-59
  byte tm_hour;       // 24-hour time 0-23
  byte tm_mday;       // day of month 1-31
  byte tm_mon;        // month 1-12
  byte tm_year;       // 90=1990, 101=2001
  byte tm_wday;       // 0=Sun...6=Sat
} tm;
```

Time can also be expressed as “seconds since January 1, 1980” (i.e., midnight December 31, 1979).

The following functions in **`DRIVERS.LIB`** are provided to read the date/time clock. Note that it takes about 600 µs to read the clock chip.

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

  Reads the real-time clock. Returns 0 if successful, or –1 otherwise. The date/time value is passed back in **`*value`**.

- **`unsigned long clock()`**

  Reads the 72421 timer and returns the time as seconds since January 1, 1980.

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

  Writes the date and time, passed in ***value**, to the clock. Returns 0 if successful, and –1 otherwise.

- **`unsigned long mktime( struct tm *value )`**

  Converts time, passed as **`*value`**, into time expressed as seconds since January 1, 1980. Does not access the timer chip.

- **`int mktm( struct tm *value, long time )`**

  Converts **`time`**, expressed as seconds since January 1, 1980, into the structure **`*value`**. Does not access timer chip.

Watchdog Timer

The watchdog timer is a reliability feature. If the watchdog timer is enabled by connecting a jumper across header J22, a timer starts running and is reset by calling the library function **`hitwd()`**. When the timer runs for 1.6 seconds without being reset, the watchdog times out. The timeout forces the BL1100 into a hardware reset condition for 50 ms, then the board resumes operation as if the power had just been turned on. It is possible to distinguish between a power-on reset and a watchdog reset. While debugging under Dynamic C, the watchdog is automatically "hit" frequently. However, if a program starts to run without any watchdog "hits," and jumper J22 is connected, a reset will take place and Dynamic C will report a loss of communication.

- **`void hitwd()`**

  "Hits" the watchdog timer. A "hit" resets the timer counter, which postpones a hardware reset for another 1.6 seconds.

- **`int wderror()`**

  Returns non-zero if the watchdog timer caused the most recent reset. If the reset was caused by either turning the power on or by pushing the reset pushbutton, then the function returns zero.

The watchdog timer's purpose is to provide recovery from a fault condition, such as an endless loop or an invalid microprocessor state. Such a fault can be caused by an electrical transient or by a software bug. An electrical transient can generate a microprocessor state that would be impossible during normal operation. A transient effect strong enough to

upset the state of the microprocessor or erase part of the memory can be much weaker than that needed to cause permanent damage. The ability to recover from such faults improves system reliability under stressful environmental conditions.

Software bugs that only occur once a week or once a year and cause the program to enter an endless loop are not unusual, but are difficult to correct. The following three scenarios could result from a software malfunction or electrical interference.

1. The stack overflows only when a rare sequence of events takes place (such as an interrupt when a seldom executed, deeply nested piece of code is executing). If that code is executed for only 10 µs every 5 min, and the interrupts take place on the average only once every hour, then the program will probably crash about once a year.
2. A multi-byte variable is shared between a high-level function and an interrupt service routine, and proper precautions are not taken to prevent interrupts while the high-level function modifies the variable. If the store to the multi-byte variable is interrupted before all of its bytes have been stored, the interrupt routine will see a mixture of the old and new values, or garbage. Dynamic C provides a shared keyword to prevent this.
3. Software may not anticipate catastrophic conditions. For example, a function that processes an A/D conversion value may always expect a positive value. However, if an electrical transient occurs when a nearby motor starts (which may happen only once a day) and makes the value of the A/D conversion negative, the program might enter an endless loop. The programmer is unlikely to find the error through testing.

Take care to prevent a state that will include **`hitwd()`** in an endless loop. The watchdog will not time out and reset the system.

APPENDIX A: TROUBLESHOOTING

Appendix A provides procedures for troubleshooting system hardware and software.

Out of the Box

Check the items mentioned in this section before starting development.

- Verify that the BL1100 runs in standalone mode before connecting any expansion boards or I/O devices.
- Verify that the entire host system has good, low-impedance, separate grounds for analog and digital signals. Often the BL1100 is connected between the host PC and another device. Any differences in ground potential from unit to unit can cause serious problems that are hard to diagnose.
- Do not connect analog ground to digital ground anywhere.
- Double-check the connecting ribbon cables to ensure that all wires go to the correct screw terminals on the BL1100.
- Verify that the host PC's COM port works by connecting a good serial device to the COM port. Remember that COM1/COM3 and COM2/COM4 share interrupts on a PC. User shells and mouse drivers, 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 supplied Z-World power supply. If another power supply must be used, verify that it has enough capacity and filtering to support the BL1100.
- Use the supplied Z-World cables. The most common fault of user-made cables is failure to properly assert CTS at the RS-232 port of the BL1100. Without CTSs being asserted, the BL1100's RS-232 port will not transmit. Assert CTS by either connecting the RTS signal of the PC's COM port or looping back the BL1100's RTS.
- Experiment with each peripheral device connected to the BL1100 to determine how it appears to the BL1100 when powered up, powered down, and/or when its connecting wiring is open or shorted.

Dynamic C Will Not Start

In most situations, when Dynamic C will not start, an error message announcing a communication failure will be displayed. The following list describes situations causing an error message and possible resolutions.

- *Wrong Baud Rate* — Make sure the jumpers are set correctly for the desired baud rate. Headers J10, J12, and J25 all affect the baud rate..
- *Wrong Communication Mode* — Both sides must be talking RS-232 (header J7) or RS-485 (header J9), depending on which protocol is being used.
- *Wrong COM Port* — A PC generally has two serial ports, COM1 and COM2. Specify the one being used in the Dynamic C **Target Setup** menu. Use trial and error, if necessary.

- *Wrong Operating Mode* — Communication with Dynamic C will be lost if the BL1100 is set for the run mode. Reconfigure the board for programming mode.
- *Wrong Memory Size* — Jumpered pins on headers J16 and J20 specify EPROM size. Verify EPROM size configuration with Figure 3-10.
- *SRAM and EPROM installed* — Both the SRAM and EPROM chips must be installed in sockets U10 and U011.

If all else fails, connect the serial cable to the BL1100 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. Connect the RS-232 cable after power up to eliminate this problem.

Dynamic C Loses Serial Link

If the program disables interrupts for a period greater than 50 milliseconds, Dynamic C will lose its serial link with the application program. Make sure that interrupts are not disabled for a period greater than 50 milliseconds.

BL1100 Repeatedly Resets

The BL1100 resets every 1.0 seconds if the watchdog timer is not "hit." If a program does not "hit" the watchdog timer, then the program will have trouble running in standalone mode. To "hit" the watchdog, make a call to the Dynamic C library function **hitwd**.

PIO Problems

PIO Modes 0, 1, and 2 need a strobe to transfer the data. The strobe lines are connected to header J12. Use Mode 3 for static input. Mode 1 can appear to work for static input, but results will be erratic since the strobe line is floating.

Power-Supply Problems

The linear power supply regulator will overheat when digital input voltages above 15 V are present on the BL1110 and BL1120. Keep the digital input voltages below 28 V for the BL1100 with a switching power supply regulator

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.

Blown-Out 5841 Driver Chip

The 5841 driver chip may blow if SUB floats. Protect the chip by installing a filter capacitor in the line connecting K to the power supply if this wire is long.

If K is not connected, the 5841 chip will blow if an inductive load is connected. The circuit will then be enabled, so be sure to plan for this situation.

Common Programming Errors

- Values for constants or variables out of range. Table A-1 lists acceptable ranges for variables and constants.

Table A-1. Constant and Variable Ranges

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`	1.18×10^{-38} to 3.40×10^{38}
`char`	0 to 255

- Mismatched "types." For example, the literal constant `3293` is of type `int` (16-bit integer). However, the literal constant `3293.0` is of type `float`. Although Dynamic C can handle some type mismatches, avoiding type mismatches is the best practice.
- Counting up from, or down to, one instead of zero. In software, 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 ASCII space character between characters forming a different legal—but unwanted—operator.
- Confusing similar-looking operators such as `&&` with `&`, `==` with `=`, and `//` with `/`.
- Inadvertently inserting ASCII nonprinting characters into a source-code file.

APPENDIX B: SPECIFICATIONS

Appendix B provides comprehensive BL1100 physical, electronic and environmental specifications.

Electrical and Mechanical Specifications

Table B-1 lists electrical, mechanical, and environmental specifications for the BL1100.

Table B-1. BL1100 General Specifications

Parameter	Specification
Board Size	5.6" × 4.8" × 1.2" (142 mm × 122 mm × 30 mm)
Operating Temperature	-40°C to 70°C, may be stored at –55°C to 85°C
Humidity	5% to 95%, noncondensing
Power	9 V DC to 28 V DC (BL1100), 80 mA at 24 V, switching regulator; 9 V to 15 V for linear regulator
Configurable I/O	16, 5 V TTL and CMOS compatible (I/O lines are software-selectable as either inputs or outputs)
Digital I/O	See Configurable I/O
Digital Outputs	Eight high-current channels, one channel can sink up to 500 mA continuously at 25°C, 8 channels can sink up to 165 mA each at 50°C and 48 V DC
Analog Inputs	Seven 10-bit conditioned: default differential input is 0 V to 0.8 V for voltage range of –4.3 V to 4.3 V
Analog Outputs	One optional, 2 mA at 0 V to 2.5 V
Resistance Measurement Input	No
Processor	Z180
Clock	9.216 MHz (12.288 MHz optional)
SRAM	32K standard, supports up to 512K
EPROM	32K standard, supports up to 512K
Flash EPROM	No
EEPROM	512 bytes
Counter/Timers	Two in hardware, four in software
Serial Ports	Two RS-232 (with CTS/RTS) and two full-duplex RS-485
Serial Rate	Up to 57,600 bps
Watchdog	Yes
Time/Date Clock	Yes
Backup Battery	Panasonic BR2325-1HG 3 V DC lithium ion, rated life 165 mA·h

BL1100 Mechanical Dimensions

Figure B-1 shows the mechanical dimensions for the BL1100.

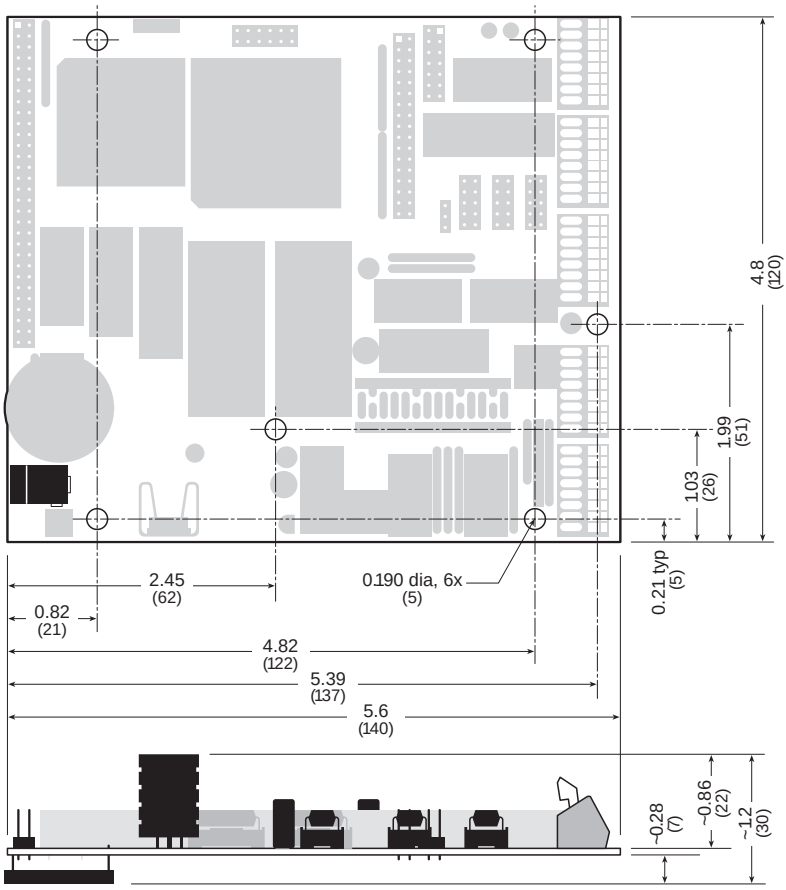


Figure B-1. BL1100 Dimensions

Jumper and Header Specifications

Figure B-2 shows the locations of the BL1100 headers.

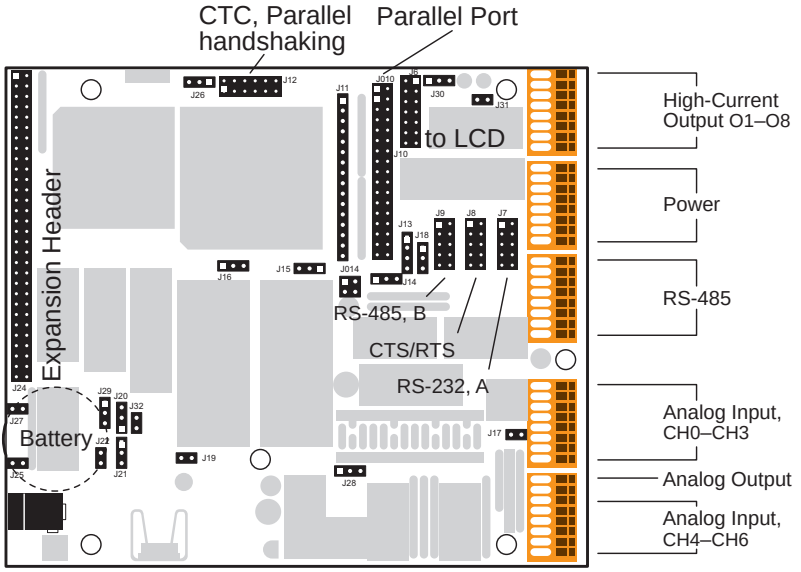


Figure B-2. BL1100 Headers

Table B-2 provides the absolute pin 1 locations for the input/output headers.

Table B-2. BL1100 Pin 1 Locations (in inches)

Header	Location	Header	Location
J1/J101	5.61, 4.73	J8	4.48, 3.30
J2/J201	5.61, 3.83	J9	4.18, 3.30
J3/J301	5.61, 2.93	J010	3.58, 4.58
J4/J401	5.61, 1.73	J10	3.58, 4.48
J5/J501	5.61, 0.83	J11	3.23, 4.48
J6	3.97, 4.67	J12	2.12, 4.58
J7	4.78, 3.30	J24	0.08, 4.73

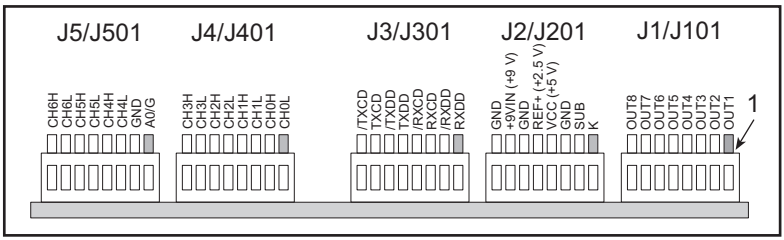
Table B-3 describes all the headers on the BL1100.

Table B-3. BL1100 Headers

Header	Description
J1/J101	High-current outputs
J2/J201	Power
J3/J301	RS-485 (Z180 Port 1 or SIO Port A)
J4/J401	Analog inputs
J5/J501	Analog inputs and optional analog output
J6	LCD interface
J7	Z180 Port 0 (RS-232)
J8	SIO Port A or Z180 Port 1 (CTS/RTS)
J9	SIO Port B (RS-485)
J10/J010	KIO parallel port
J11	Alternate parallel port for user-installed flexible keypad connector
J12	Counter/timer circuit interface
J24	BL1100 expansion bus

Wago Connector Signals

Figure B-3 shows the pinouts for Wago connectors J1/J101–J5/J501.



NOTES

1. J2/J201 SUB negative current supply for high-current driver, tied to ground unless split supply used. Must not float with respect to J2/J201–J3/J301.
2. J2/J201 VCC can be an input if the onboard supply is disabled (header J19).
3. J2/J201 +9VIN is alternative input to power plug connector (J23).
4. J5/J501 A0/G is analog ground or optional DAC output (selected on header J28).
5. Do not mix analog grounds (J5/J501) with digital grounds (J2/J201).

Figure B-3. Pinouts for Wago Connectors

LCD Interface

Figure B-4 shows the pinouts for the LCD interface on header J6.

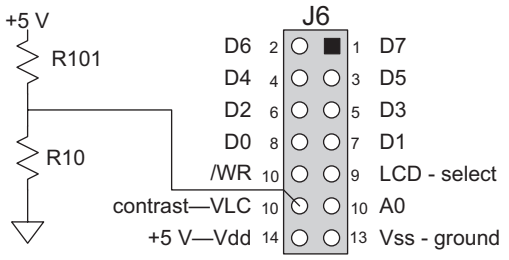


Figure B-4. LCD Interface Pinout

Serial Communication Signals

Figure B-5 shows the pinouts for the serial communication signals on headers J7–J9.

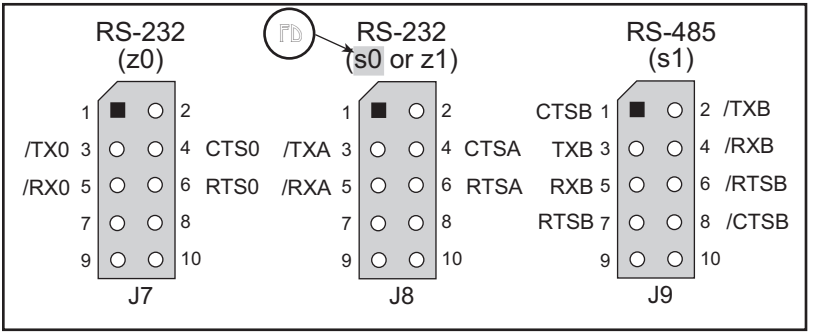


Figure B-5. Serial Communciation Pinouts

PIO Parallel Port and Other Lines

Figure B-6 shows the pinouts for the PIO parallel port and related clock lines on headers J10, J010, J11, and J12.

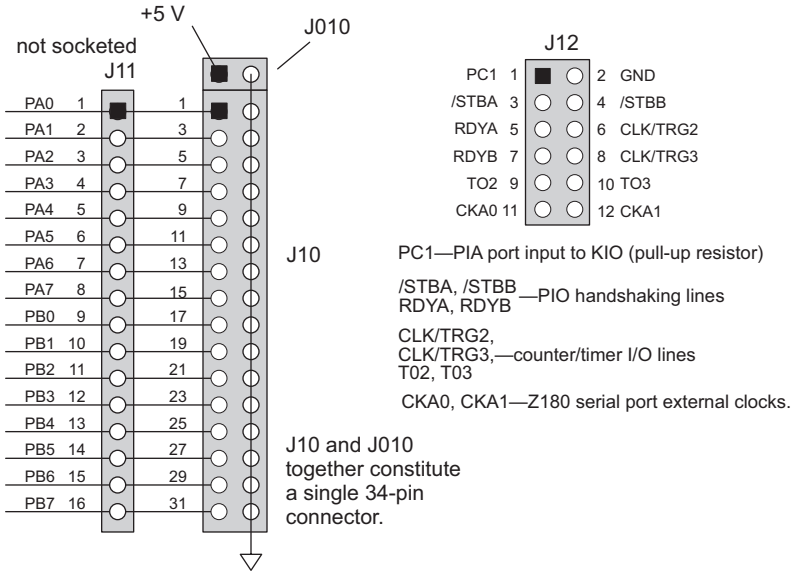


Figure B-6. PIO Parallel Port Pinouts

BL1100 Expansion Bus

Figure B-7 shows the pinout for the BL1100 expansion bus on header J24.

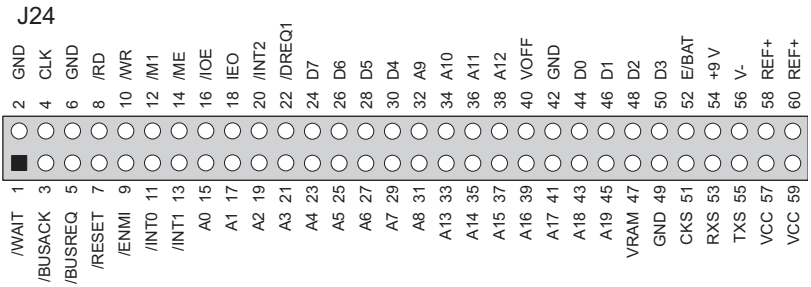


Figure B-7. Expansion Bus Pinout

Jumper Configurations

Table B-4 lists the jumper settings for the applicable BL1100 headers.

Table B-4. Standard BL1100 Jumper Settings

Header	Pins	Description	Factory Default
J12	1–2	Connect for factory-default RS-232 or RS-485 programming mode, remove for run mode or user-defined programming mode	Connected
	6–10	CTC3 output drives CTC2	
	8–9	CTC2 output drives CTC3	
	6–11	CTC2 operates at 16X baud rate of z0	
	6–12	CTC2 operates at 16X baud rate of z1	
	8–11	CTC3 operates at 16X baud rate of z0	
	8–12	CTC3 operates at 16X baud rate of z1	
	9–11	CTC2 drives external clock at 1/16 z0 baud rate	
	10–12	CTC3 drives external clock at 1/16 z1 baud rate	
J13	1–2	SIO Port A to RS-232 (J8)	Connected
	2–3	SIO Port A to RS-485 (J3)	
	3–4	ZIO Port 1 to RS-485 (J3)	Connected
	1–4	ZIO Port 1 to RS-232 (J8)	
J14	1–2	SIO Port A to RS-485 (J3)	
	2–3	ZIO Port 1 to RS-485 (J3)	Connected
J014	1–2	RX and TX clocks come from SIO	Connected
	2–4	TX clock from SIO, RX clock from external source via pins 3–4 of header J3/J301	
J15	1–2	SRAM smaller than 256K	Connected
	2–3	SRAM 256K or larger	
J16	1–2	EPROM 32 K	Connected
	2–4	EPROM larger than 32K	
J17	1–2	Connect to enable excitation resistors for analog input signal conditioning op-amps	Not connected

continued…

Table B-4. Standard BL1100 Jumper Settings (concluded)

Header	Pins	Description	Factory Default
J18	1–2	RS-485 three-state always enabled	Connected
	2–3	RS-485 three-state controlled by PB0	
	none	RS-485 three-state always disabled	
J19	1–2	Connect to enable onboard +5 V power supply, disconnect for external power supply	Connected
J20	1–2	EPROM smaller than 256K	Connected
	2–4	EPROM 256K	
J21	1–2	Write-protect upper part of EEPROM	Connected
	2–3	Write-enable upper part of EEPROM	
J22	1–2	Connect to enable watchdog timer	Not connected
J25	1–2	Startup mode factory-default RS-232 or read from EEPROM	Connected
J26	1–2	Remote reset using RXC	Not installed
	2–3	Remote reset using CTSB	
	none	No remote reset	
J27	1–2	Connect to enable switching power supply, disconnect for time/date clock to control power supply	Connected
J28	1–2	Output DAC to pin 1 on J5/J501	
	2–3	Pin 1 of J5/J501 is analog ground	Connected
J29	1–2	Connects E to expansion bus J24 pin 52	Not installed
	2–3	Connects BATACT to expansion bus J24 pin 52	
J31	1–2	Ties high-voltage driver SUB signal to ground to keep it from floating	Connected
J32	1–2	Connect to control use of ENMI signal on expansion bus J24 pin 9	Connected

APPENDIX C: MEMORY, I/O MAP, AND INTERRUPT VECTORS

Appendix C provides detailed information on memory, provides an I/O map, and lists the interrupt vectors.

BL1100 Memory

There are two 32-pin memory sockets, one for ROM and one for RAM. Sockets U011 and U10 will accept either 32-pin or 28-pin memory chips.

Physical Memory

Depending on PAL coding and wiring, the BL1100 can address 256K or 512K of ROM, and 512K of RAM. Figure C-1 illustrates the physical memory mapping.

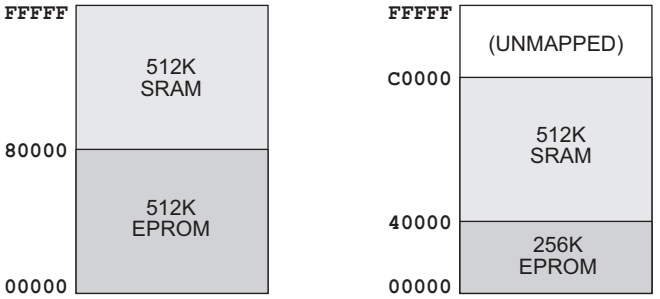


Figure C-1. BL1100 Physical Memory Map

The memory chips usually installed on the BL1100 have a capacity less than 512K. Typical SRAM chips have 32K or 128K.

If there is less than 512K of SRAM, addresses outside the memory range will map to addresses within the range. For example, addresses on a 32K chip evaluate modulo 32K. Therefore, data may seem to be replicated in memory. Or worse, you data may be overwritten.

Memory Management

Z180 instructions can specify 16-bit addresses, giving a *logical* address space of 64K (65,536 bytes). Dynamic C supports a 1M *physical* address space (20-bit addresses).

An on-chip memory management unit (MMU) translates 16-bit Z180 addresses to 20-bit memory addresses. Three MMU registers (CBAR, CBR, and BBR) divide the logical space into three sections and map each section onto physical memory, as shown in Figure C-2. (Potentially, the three sections may overlap or be of zero size.)

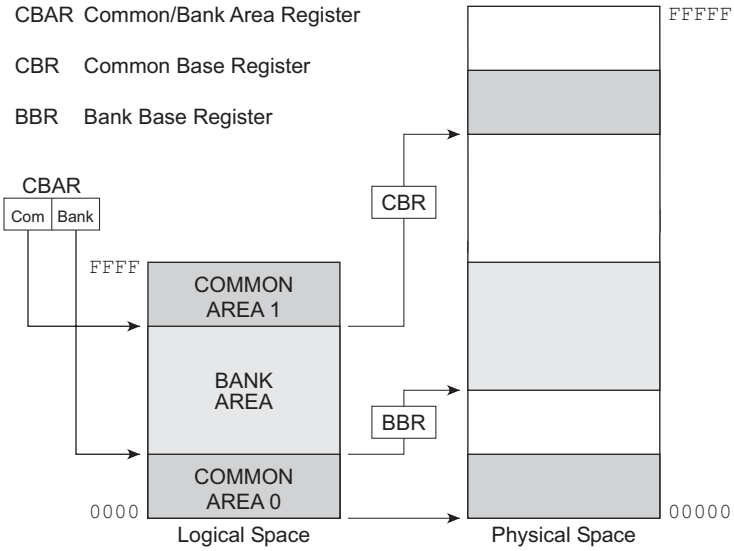


Figure C-2. Mapping Logical Memory to Physical Memory

The logical address space is partitioned on 4K boundaries. Given a 16-bit address, the Z180 uses CBAR to determine whether the address is in common area 1, common area 0, or the bank area. If the address is in common area 1, the Z180 uses the CBR as the base to calculate the physical address. If the address is in the bank area, the Z180 uses the BBR. If the address is in common area 0, the Z180 uses a base of 00H.

The physical address is, essentially,

(base << 12) + logical address,

as shown in Figure C-3.

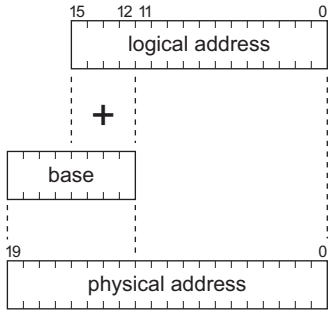


Figure C-3. Determining Physical Address from Logical Address

How Dynamic C Uses the MMU

In order to use a 1M physical address space, Dynamic C partitions logical space into three areas, which correspond to the Z180 common and bank areas listed in Table C-1.

Table C-1. Dynamic C Logical Space Partitions

Name	Size	Description
BIOS	8K	Basic Input/Output System. Loosely named, the BIOS is always present and is always mapped to address 0 of ROM. The BIOS contains the power-up code, the communication kernel, and important system features. The BIOS corresponds to common area 0.
ROOT	48K	The area between the BIOS and XMEM (the bank area). The root—normal memory—resides in a fixed portion of physical memory. Root code grows upward in logical space from address 2000 (hex) and root data (static variables, stack and heap) grow down from E000. (Initialized static variables are placed with code, in ROM or RAM.)
XMEM	8K	Extended Memory. XMEM is essentially an 8 kbyte "window" into physical memory. XMEM can map to any part of physical memory (ROM or RAM) simply by changing the CBR. XMEM corresponds to common area 1.

The XMEM area has many mappings to physical memory. The root and BIOS have 1:1 mappings.

Dynamic C can either "compile to RAM" (for development) or "compile to ROM" (for a standalone programs), as shown in Figure C-4.

See the ***Dynamic C Technical Reference*** manual for details.

Dynamic C uses physical memory in slightly different ways, depending on the options selected by the programmer. When compiling to ROM, the compiler places root code (including constants and preset variables) in ROM, directly above the BIOS. When compiling to RAM, the entire root (code and data) is placed in RAM.

Z180 memory management is not automatic. The Dynamic C compiler emits code that will set the mapping registers. Assembly language programmers must do it themselves. However, Dynamic C does generate bouncers for assembly language calls to XMEM functions. Dynamic C

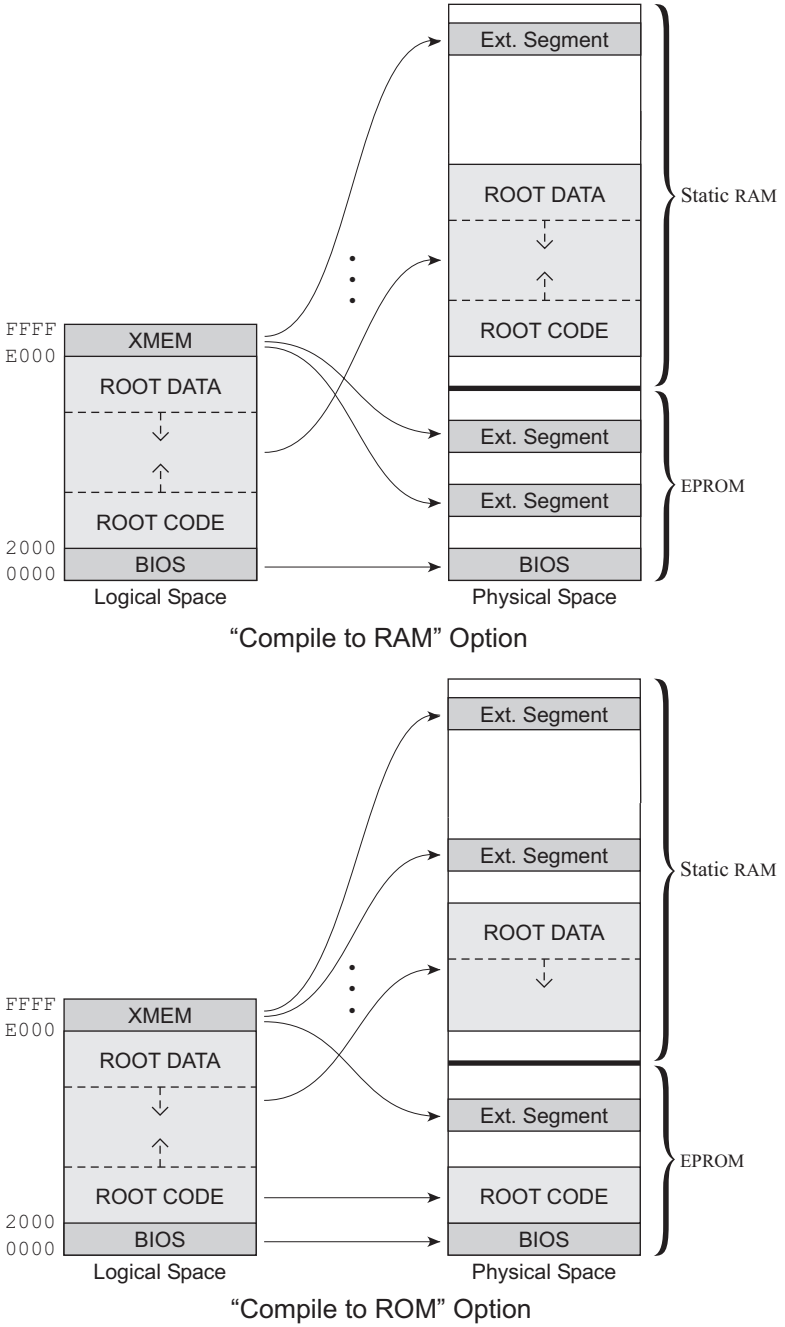


Figure C-4. Dynamic C Memory Mapping

supports the use of extended memory with several extensions to the C language: function classes, additional compiler directives, and extended memory data declarations.

Control over Memory Mapping

The programmer has complete control over how Dynamic C allocates and maps memory with the options on the **SETUP** menu listed in Table C-2.

Table C-2. Dynamic C Memory Mapping Options

Option	Description
Reserve Control	Specifies the size of the root memory reserve or the extended memory reserve.
RAM Map Control	Specifies how RAM is to be mapped when "compiling to RAM."
ROM Map Control	Specifies how ROM is to be mapped when "compiling to ROM."

See the ***Dynamic C Technical Reference*** manual for details.

Extended Memory Code

Physical memory is divided into 4K "pages." Two of these pages are visible in the extended memory window (XMEM) at any one time. Additional code is required to handling calls to other functions or jumps to locations not currently mapped in the extended memory window.

A program can use many 4K pages of extended memory. Normally, the page being executed is mapped to the address region E000H to F000H. As the execution approaches F000, the pages are shifted so that the code in the region F000 to FFFF is moved down to the E000 to F000 region. A "bouncer" in low memory is called to accomplish this task. The bouncer modifies the memory management unit (MMU) to slide the code down one page and then jumps to the new location. This transfer of control is made is at the end of the first statement that crosses F000. No single C expression can be more than 4K long.

However, statements such as **`switch`** or **`while`** that cause program jumps can be as long as desired. A bouncer is used to execute the jump if a jump crosses page boundaries.

Any C function can call any other C function, no matter where in memory it is located. However, it is less efficient to call a function located in extended memory than to call a function located in root memory, because a bouncer must be used to modify the memory management registers before and after the call. A separate bouncer is compiled for each call and is specific to that particular call. The compiler places all bouncers in root memory.

Programs run faster when functions that are short and frequently used are placed in the root memory.

Extended Memory Data

Accessing data in extended memory is not as "transparent" as calling functions in extended memory.

Dynamic C has two keywords, **xdata** and **xstring**, and some library functions that allow data to be stored in extended memory.

The deluxe version of Dynamic C is required to be able to declare extended memory data.

Library functions (**xmem2root**, **root2xmem**, **xstrlen**, **xgetlong**) exist for manipulating extended memory data.

See the Dynamic C manuals for details.

Execution Timing

Table C-3 provides the execution times for a BL1100 with a 9.216 MHz clock and a 12.288 MHz clock. These times reflect the use of Dynamic C libraries. The time required to read from memory is included, but the time to store a result is not.

The Z180 memory cycle requires faster memory during certain op-code fetch cycles (LIR cycles). The term "1/2 wait state" means that these cycles are stretched by adding a wait state. The term "0 wait state" means that none of the cycles have wait states added. Generally, 1/2 wait state cycles are used for higher clock speeds. Zero wait states are fine for slower clock speeds.

Table C-3. BL1100 Execution Times
(μs)

Operation	9.216 MHz		12.288 MHz
	½ Wait	0 Wait	0 Wait
DMA copy per byte	0.73	0.73	0.55
Integer assignment	3.7	3.4	2.6
Integer add	4.9	4.4	3.3
Integer multiply	21	18	13.5
Integer divide	104	90	68
Floating add (typical)	102	85	64
Floating multiply	135	113	85
Floating divide	386	320	240
Long add	34	28	21
Long multiply	114	97	73
Long divide	503	415	312
Floating square root	1016	849	637
Floating exponent	2920	2503	1877
Floating cosine	3597	3049	2287

Memory-Access Timing

Two types of memory cycles must be considered: standard memory cycles and Load Instruction Register (LIR) cycles. LIR cycles, which fetch the op code, have the most critical timing requirement. The memory access time, *t*, in nanoseconds, for these cycles can be calculated using

$$t = 2T - 95 \quad \text{(9.216 MHz clock)}$$

or (C-1)

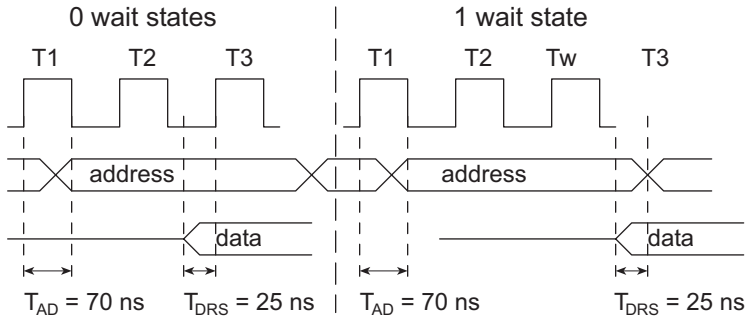
$$t = 2T - 70 \quad \text{(12.288 MHz clock)},$$

where T is the period of a clock cycle. These cycles are shown in Figure C-5 with and without a wait state. The corresponding memory access times are listed for several clock frequencies.

There are two versions of the PAL (located in socket U11) available. The standard version generates a wait state only during the LIR cycles. Thus, it is called a "½ wait state" PAL. The other version generates no wait states.

Zero wait states can also be generated with the standard U11 PAL by removing pin 18 from the socket or by clearing bit 7 in the OMCR register to disable the /LIR signal.

Refer to Technical Note 109, ***Interrupt Daisy Chain, Wait States, and the Use of PIO***, for more information on the consequencers of disabling the /LIR signak.



12.288 MHz	
0 wait access time = 2T - 70 ns	1 wait access time = 3T - 70 ns
= 93 ns for 12.288 MHz clock	= 162 ns for 12.288 MHz clock
9.216 MHz	
0 wait access time = 2T - 95 ns	1 wait access time = 3T - 95 ns
= 105 ns for 10.00 MHz clock	= 205 ns for 10.00 MHz clock
= 122 ns for 9.216 MHz clock	= 230 ns for 9.216 MHz clock
= 229 MHz for 6.144 MHz clock	= 391 ns for 6.144 MHz clock

Figure C-5. Memory Cycles for 9.216 MHz Processor With and Without a Wait State

Table C-4 lists the memory access times required for various clock frequencies and wait states.

Table C-4. Memory Access Times
(ns)

Clock Frequency	EPROM	SRAM
9.216 MHz ½ wait state	176	176
9.216 MHz 0 wait states	122	176
12.288 MHz ½ wait state	133	133
12.288 MHz 0 wait states	93	133

The times for the 12.288 MHz clock are based on a faster version of the Z180, which is why the access time can be slower. However, the SRAM's access time must equal that of the EPROM during program development or when executing code in SRAM, because code executes from SRAM during these periods.

Memory Map

The various registers in the I/O space can be accessed in Dynamic C by the symbolic names listed in Table C-5.

Table C-5. Z180 Internal I/O Registers Addresses 0x00–0x3F

Address	Name	Description
0x00	CNTLA0	Serial Channel 0, Control Register A
0x01	CNTLA1	Serial Channel 1, Control Register A
0x02	CNTLB0	Serial Channel 0, Control Register B
0x03	CNTLB1	Serial Channel 1, Control Register B
0x04	STAT0	Serial Channel 0, Status Register
0x05	STAT1	Serial Channel 1, Status Register
0x06	TDR0	Serial Channel 0, Transmit Data Register
0x07	TDR1	Serial Channel 1, Transmit Data Register
0x08	RDR0	Serial Channel 0, Receive Data Register
0x09	RDR1	Serial Channel 1, Receive Data Register
0x0A	CNTR	Clocked Serial Control Register
0x0B	TRDR	Clocked Serial Data Register
0x0C	TMDR0L	Timer Data Register Channel 0, low
0x0D	TMDR0H	Timer Data Register Channel 0, high
0x0E	RLDR0L	Timer Reload Register Channel 0, low
0x0F	RLDR0H	Timer Reload Register Channel 0, high
0x10	TCR	Timer Control Register
0x11–0x13	—	Reserved
0x14	TMDR1L	Timer Data Register Channel 1, low
0x15	TMDR1H	Timer Data Register Channel 1, high
0x16	RLDR1L	Timer Reload Register Channel 1, low
0x17	RLDR1H	Timer Reload Register Channel 1, high
0x18	FRC	Free-running counter
0x19–0x1F	—	Reserved
0x20	SAR0L	DMA source address Channel 0, low
0x21	SAR0H	DMA source address Channel 0, high

continued…

Table C-5. Z180 Internal I/O Registers Addresses 0x00–0x3F (concluded)

Address	Name	Description
0x22	SAR0B	DMA source address Channel 0, extra bits
0x23	DAR0L	DMA destination address Channel 0, low
0x24	DAR0H	DMA destination address Channel 0, high
0x25	DAR0B	DMA destination address Channel 0, extra bits
0x26	BCR0L	DMA Byte Count Register Channel 0, low
0x27	BCR0H	DMA Byte Count Register Channel 0, high
0x28	MAR1L	DMA Memory Address Register Channel 1, low
0x29	MAR1H	DMA Memory Address Register Channel 1, high
0x2A	MAR1B	DMA Memory Address Register Channel 1, extra bits
0x2B	IAR1L	DMA I/O Address Register Channel 1, low
0x2C	IAR1H	DMA I/O Address Register Channel 1, high
0x2D	—	Reserved
0x2E	BCR1L	DMA Byte Count Register Channel 1, low
0x2F	BCR1H	DMA Byte Count Register Channel 1, high
0x30	DSTAT	DMA Status Register
0x31	DMODE	DMA Mode Register
0x32	DCNTL	DMA/WAIT Control Register
0x33	IL	Interrupt Vector Low Register
0x34	ITC	Interrupt/Trap Control Register
0x35	—	Reserved
0x36	RCR	Refresh Control Register
0x37	—	Reserved
0x38	CBR	MMU Common Base Register
0x39	BBR	MMU Bank Base Register
0x3A	CBAR	MMU Common/ Bank Area Register
0x3B–0x3D	—	Reserved
0x3E	OMCR	Operation Mode Control Register
0x3F	ICR	I/O Control Register

These names are treated as unsigned integer constants. Use the library functions **inport** and **outport** to access the I/O registers.

```
data_value = inport( CNTLA0 );
outport( CNTLA0, data_value );
```

The library functions **IBIT**, **ISET**, and **IRES** can be used to set and clear bits in I/O 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.

The addresses in Table C-6 cover the KIO registers.

Table C-6. KIO Registers Addresses 40-4F

Address	Name	Description
40	PIODA	PIO Port A Data
41	PIOCA	PIO Port A Control
42	PIODB	PIO Port B Data
43	PIOCB	PIO Port B Control
44	CTC0	CTC Channel 0
45	CTC1	CTC Channel 1
46	CTC2	CTC Channel 2
47	CTC3	CTC Channel 3
48	SIODA	SIO Channel A Data
49	SIOCA	SIO Channel A Control
4A	SIODB	SIO Channel B Data
4B	SIOCB	SIO Channel B Control
4C	PIAD	PIA Port C Data
4D	PIAC	PIA port C Control
4E	KIOC	KIO Control
4F	—	Reserved

Table C-7 lists the addresses of other BL1000 registers.

Table C-7. Other Register Addresses (60–FF)

Address	Name	Description
60	SDA_R	Read only, 1 bit, EEPROM SDA register
62	LCD	LCD interface R/W
64	BMS	Strobe serial register to output register 5841
66	ENB485	Enable 485 driver, write only, 1 bit
68	BMO	Write only enable 5841 output 1 bit
6A	AD_ADE	Write only, 1 bit register, enable A/D
6C	SC	Write only, 1 bit register, EEPROM clock
6E	SDA_W	Write only, 1 bit register, EEPROM data
70–7F	—	Reserved for duplicate decodes
80–FF	—	Unused I/O space

Time/Date Clock

The battery-backed clock appears as 16 I/O registers with addresses of 50–5F. The 16 registers are 4 bits wide. The upper 4 bits of the register are undefined. The register arrangement is shown in Table C-8.

Table C-8. Epson 72421 Real-Time Clock Registers

Address	Bit 3	Bit 2	Bit1	Bit 0	Description
50	S8	S4	S2	S1	seconds, units
51		S40	S20	S10	seconds, tens
52	M8	M4	M2	M1	minutes, units
53		M40	M20	M10	minutes, tens
54	H8	H4	H2	H1	hours, units
55		AM/PM	H20	H10	hours, tens
56	D8	D4	D2	D1	days, units
57			D20	D10	days, tens
58	M8	M4	M2	M1	months, units
59				M10	months, tens
5A	Y8	Y4	Y2	Y1	years, units
5B	Y80	Y40	Y20	Y10	years, tens
5C		W4	W2	W1	weeks
5D	30 ADJ	IRQ FLG	BUSY	HOLD	Register D
5E	T1	T0	INTR/ STND	MASK	Register E
5F	TEST	12/24	STOP	RSET	Register F

The 4-bit registers are mostly binary-coded decimal numbers that make up the date and time.

The following notes apply to the registers listed in Table C-8.

- Set the 12/24 bit to 1 for 24-hour mode and 0 for 12-hour mode. The a.m./p.m. bit will be 1 for p.m. Mask out the a.m./p.m. bit in the 24-hour mode.
- The days of the week are represented by 0 for Sunday and 6 for Saturday.
- Leap year is automatically taken into account.
- Set the year to 90 for 1990, to 91 for 1991, and so on.
- Constant reading of the clock (as in a tight loop) will create a loss of accuracy.

Initialized Memory Locations

Table C-9 lists the constants that are initialized at startup.

Table C-9. Constants Initialized at Setup

CLOCKSPEED	Integer containing the clock speed read in multiples of 1200 Hz from the EEPROM at startup
BAUDCODE	Integer containing the baud rate in multiples of 1200 bps as read from the EEPROM at startup
JUMPERS	Byte read from the P1B port of the KIO at startup

To access these variables, declare them as follows.

```
extern unsigned int CLOCKSPEED
extern unsigned int BAUDCODE
extern unsigned int JUMPERS
```

Interrupt Vectors

Tables C-10, C-11, and C-12 present a suggested interrupt vector map. Most of these interrupt vectors can be altered under software control. The addresses are given 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 C-10. 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

Table C-11. Interrupt Vectors for KIO Devices

Address	Name	Description
0x12	**PIOA_VEC**	PIO parallel port Channel A
0x14	**PIOB_VEC**	PIO parallel port Channel B
0x18	**CTC0_VEC**	CTC Timer Channel 0
0x1A	**CTC1_VEC**	CTC Timer Channel 1
0x1C	**CTC2_VEC**	CTC Timer Channel 2
0x1E	**CTC3_VEC**	CTC Timer Channel 3

Table C-12. SIO Interrupt Vectors

Address	Name	Description
0x20	**SIOBT_VEC**	Channel B Transmit Buffer Empty
0x22	**SIOBEX_VEC**	Channel B External/Status Change
0x24	**SIOBR_VEC**	Channel B Receive Character Ready
0x26	**SIOBER_VEC**	Channel B Special Receive Condition
0x28	**SIOAT_VEC**	Channel A Transmit Buffer Empty
0x2A	**SIOAEX_VEC**	Channel A External/Status Change
0x2C	**SIOAR_VEC**	Channel A Receive Character Ready
0x2E	**SIOAER_VEC**	Channel A Special Receive Condition

A directive such as the following is used to "vector" an interrupt to a user function in Dynamic C.

```
#INT_VEC 0x10 myfunction
```

This statement 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 as follows.

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

Nonmaskable Interrupts

Power-Fail Interrupts

The following sequence of events takes place when power fails.

1. The nonmaskable interrupt (NMI) signaling power failure is triggered whenever the unregulated DC input voltage falls below approximately 8 V (subject to voltage divider R3/R4).
2. The system reset is triggered when the regulated +5 V falls below 4.5 V; the reset remains enabled as the voltage falls further. At this point, the chip select for the SRAM is forced high (standby mode). Power for the time/date clock and the SRAM is switched to the lithium backup battery as the regulated voltage falls below the battery voltage (approximately 3 V).

The following sample routine demonstrates how to handle a power-fail interrupt.

```
#JUMP_VEC NMI_INT myint

interrupt retn myint(){
  body of interrupt routine...
  while( 1 ) if( !powerlo() ) return;
}
```

Normally, a power-fail interrupt routine will not return. Instead, it will execute shutdown code and then enter a loop until the +5 V falls low enough to trigger a reset. However, the voltage might fall low enough in a brownout situation to trigger the power-fail interrupt but not the reset, resulting in an endless hangup. The library function **`powerlo`** returns 1 if the voltage level is below the NMI threshold; otherwise, it returns 0. Call **`powerlo`** only from an NMI routine. If powerlo detects that the low-voltage condition has reversed itself, then the power-failure routine can restart execution. If a low-voltage condition that is not fatally low persists, then the user must decide decide what action to take, if any.

A situation similar to brownout occurs if the power supply is overloaded. Say the load temporarily increases, perhaps when an LED is turned on, causing the power supply to appear to have failed. The interrupt routine sheds some of the load by doing a shutdown procedure, causing the power-fail condition to go away. If no action is taken to correct the overload, the system will oscillate around the power fail. To correct this, use a larger power supply.

Do not forget the interaction that can occur between the watchdog timer and the power-fail interrupt. If the watchdog timer is enabled and a brownout causes an extended stay in the power-fail interrupt routine, then the watchdog can time out, causing a system restart.

Even if the power is cut off from the board abruptly, a certain amount of computing time will remain before the +5 V supply falls below 4.5 V. The amount of time depends on the size of the capacitors in the power supply. The standard wall transformer provides about 10 ms. If the power cable is abruptly removed from the BL1100, then only the capacitors on the board are available and the time is reduced to a few hundred microseconds. These times can vary considerably, depending on the board configuration and the other loads, if any, drawing from the board's power supply.

The time interval between detection of a power failure and entry to the user's power-fail interrupt routine is approximately 100 µs, or less if Dynamic C's nonmaskable interrupt communications are not in use.

It is hard to test power-fail interrupt routines presents because the interrupts are normally disabled. Probably the best test method involves leaving messages in battery-backed memory to track the execution of the power-fail routines. If a variable transformer is available to drive the wall transformer, then brownouts and other types of power-fail conditions can be easily simulated.

The power-failure interrupt must be disabled if an external +5 V power supply is used.

Jump Vectors

Jump vectors are special interrupts that are different from INT0 interrupts. Instead of loading the address of the interrupt routine from the interrupt vector, jump vectors cause a jump directly to the address of the vector, for example,

0x66 nonmaskable power-failure interrupt,

that contains a jump instruction to the interrupt routine.

Since nonmaskable interrupts can be used for Dynamic C communications, the interrupt vector for power failure is normally stored just in front of the Dynamic C program. A vector may be stored there by the following command.

```
#JUMP_VEC NMI_VEC name
```

The Dynamic C communication routines relay to this vector when the nonmaskable interrupt is caused by a power failure rather than by a serial interrupt.

Interrupt Priorities

Table C-13 lists the interrupt priorities.

Table C-13. Interrupt Priorities

Interrupt Priorities	
(Highest Priority)	Trap (Illegal Instruction)
	NMI (Nonmaskable Interrupt)
	SIO Channel A*
	SIO Channel B*
	CTC Channel 0*
	CTC Channel 1*
	CTC Channel 2*
	CTC Channel 3*
	PIO Channel A*
	PIO Channel B*
	INT 1 (expansion bus attention line interrupt)
	INT 2 (expansion bus attention line interrupt)
	PRT Timer Channel 0
	PRT Timer Channel 1
	DMA Channel 0
	DMA Channel 1
	Clocked Serial I/O
	Asynchronous Serial Port 0
(Lowest Priority)	Asynchronous Serial Port 1

* The priority of the SIO, CTC, and PIO interrupts can be altered through the KIO control register.

APPENDIX D: EEPROM

Parameters

The onboard EEPROM (electrically erasable, programmable, read-only memory) is used to store the constants and parameters listed in Table D-1.

Table D-1. BL1100 EEPROM Assignments

Address	Bytes	Function
00	1	Operating Mode: 1—RS-232 programming via header J7 2—RS-485 programming with NMI via header J9 4—RS-485 programming with regular interrupts via header J9 8—Run user program in SRAM on startup
01	1	Baud rate code (in multiples of 1200 bps)
100	6	Unit serial number—binary-coded decimal time and date in the format: seconds, minutes, hour, day, month, year
108	2	Clock speed (in multiples of 1200 Hz)
10A	1	Wait states, value to be inserted in DCNTL for I/O and memory wait states. Default = 0x20 for 3 I/O wait states and 0 memory wait states.
10C	2	Network node address
110	4	Reference temperature coefficient: T (K) = coefficient × ADC value

The EEPROM has 512 bytes. Bytes 0–255 can be written to at any time, but the upper 256 bytes can be written to only when header J21 is enabled (pins 2 and 3 are connected). Connect pins 1 and 2 on J21 to write-protect the EEPROM.

Figure D-1 shows the EEPROM memory and J21 jumper settings.

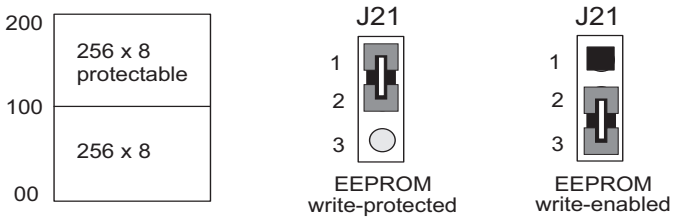


Figure D-1. BL1100 EEPROM and Jumper Settings

Library Routines

The following library routines can be used to read and write the EEPROM:

```
int ee_rd( int address );
int ee_wr( int address, byte data );
```

The function **ee_rd** returns a data value or, if a hardware failure occurred, –1. The function **ee_wr** returns –1 if a hardware failure occurred, –2 if an attempt was made to write to the upper 256 bytes with the protection jumper (J16) installed, or 0 to indicate a successful write. A write-protection violation does not wear out the EEPROM. These routines each require about 2 ms to execute. They are not re-entrant, that is, only one routine at a time will run.

The EEPROM has a rated lifetime of only 10,000 writes (unlimited reads). Do not write the EEPROM from within a loop. The EEPROM should be written to only in response to a human request for each write.

APPENDIX E: POWER MANAGEMENT

Appendix E provides information about power consumption and intermittent operation.

Power Consumption

Table E-1 provides the power consumption for various BL1100 components. The figures are approximate. Remember to add a safety margin.

Table E-1. Current Draw of Major BL1100 Components

Component	Current Draw (mA)	
	4.608 MHz	9.216 MHz
Z180	10	20
KIO	10	20
PALs std	100	110
PALs ¼ power	25	28
SRAM 32K	10	20
EPROM 64K	20	30
LTC1134 RS-232	24	24
Analog section	5	5
RS-485 drivers	120	120
5841	5	5
24C04 EEPROM (Standby)	1	1
24C04 EEPROM (Program)	7	7

A minimum power configuration would consist of the following items.

Z180, zero-power PALs, SRAM, EPROM, 4.608 MHz clock	50 mA–60 mA
LTC 1134 RS-232	24 mA
KIO	10 mA
Analog section	5 mA
Total	89 mA–99 mA

The standard configuration at 9.216 MHz requires about 270 mA.

The +5 V switching regulator, LM2575 at U06, will generate at least 500 mA if the input voltage is 10 V. A higher output current can be generated with a higher input voltage (and a larger inductor at L3).

The LM340 linear regulator (U13) is rated for 1 A for an input voltage of 10 V, and a maximum ambient temperature of 45°C. The LM340 can operate at up to 85°C when producing up to 500 mA, which is a higher temperature rating than the rating for the rest of the components.

Intermittent Operation

Power can be turned on and off under software control on BL1100s equipped with a switching power regulator. This is done under the control of the time/date clock or by an external switch.

The switching power regulator turns off when the signal VOFF is raised high and turns on when VOFF is pulled low. When the regulator turns on, there is a power-on reset lasting for approximately 50 ms. After the power-on reset, the program's main routine begins processing after approximately 10 ms.

VOFF can be driven by an external circuit, permanently enabled (low) with a jumper by installing a jumper across J27, or controlled by the open drain output of the 72421 clock chip (jumper across header J27 removed). The power can be controlled in one of the following ways.

1. An operator pushbutton grounds VOFF, enabling power. The software then calls the library function **`powerup`** to keep the power enabled after the operator releases the pushbutton. When power is no longer needed, the program calls the function **`powerdown`** to turn the power off until another external event re-enables power. This logic can be used to create a battery-powered instrument that turns off automatically after a certain period of inactivity to conserve the battery.
2. Power is turned on periodically for a short period of time. The available periods are

   1 second,
   1 minute, and
   1 hour.

   Power is turned on at the start of the next period. If, for example, the wakeup time is set to one minute, the board will wake up when the minute *changes*, not after one minute has elapsed. If the program runs for 10 s and then goes to sleep, 50 s will elapse before the board wakes up again.

   The minimum time for power to be on is approximately 60 ms. Power consumption will be decreased by a factor of approximately 15 to 1 if power is on for only 60 ms in every second. If power is on only once a minute, the ratio will be 900 to 1. Once every hour reduces the ratio to54,000 to 1. If a 9 V, 500 mA·h battery is used, the battery life with

power on continuously is only 1.5 h. The battery life would be extended to approximately one day with power enabled every second. Enabling power only once a minute extends the battery life to approximately two months. Enabling power once every hour extends battery life to approximately 10 years. This type of power usage is convenient for data collection applications, for example, recording the temperature at one-minute intervals under battery power.

The following library functions support intermittent operation.

- **`setperiodic( int code )`**

  This function specifies the interval between VOFF pulses from the date/time clock: **`code`** = 4 ⇒ 1 second, 8 ⇒ 1 minute, 12 ⇒ 1 hour.

- **`sleep()`**

  Turns power off until next periodic time.

  The periodic interrupts depend on the modes set into the battery-backed memory of the date/time clock (the 72421 chip). If the 72421 is upset by a voltage transient, or the lithium battery goes dead, the board could fail to wake up at the specified time. Z-World recommends adding an external wakeup circuit to replace or supplement the 72421 for critical applications that must run unattended.

APPENDIX F: OPTO 22 SUPPORT

Appendix F summarizes the BL1100's Opto 22 support and compares the performance of the BL1100 with Opto 22 controllers.

The Opto 22 Company (Huntington Beach, CA, 714-891-5861) provides a family of components that can be connected to an RS-485 bus. The system that uses these components must include a master controller, which may be any controller capable of sending and receiving RS-485 messages at up to 19,200 bps. Opto 22 supplies several types of master controllers, including PC software drivers. The slave units, called *brain boards*, are provided in analog and digital two varieties. The brain boards are connected to digital or analog I/O mounting racks. The mounting racks accept plug-in modules that are available for many types of I/O, as shown in Table F-1.

Table F-1. Opto 22 Plug-In Modules

Digital Modules	• Solid-state relays to control AC or DC • Digital inputs for a variety of voltages (AC or DC) • Relays
Analog Inputs	• Voltage (various) • Current (4–20 mA) • Thermocouple (various) • RTD • Frequency
Analog Outputs	• Voltage (various) • Current (4–20 mA)

The modules are optically isolated, and are intended for industrial use.

The following publications give additional details on the workings of the Opto 22 system. The publications are available from Opto 22, and are usually supplied at no charge.

Computer-Based I/O Catalog (form 132.6)

Optomux B1 and B2 Digital and Analog Brain Boards Operations Manual (Part #1927) (Form 203.1)

LC4 Operations Manual

Figure F-1 shows a block diagram of the Opto 22 system.

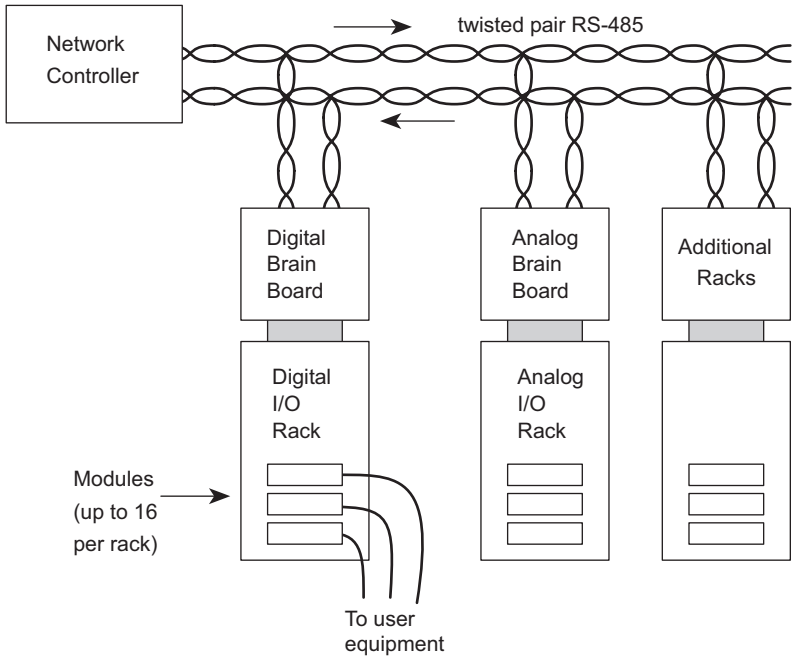


Figure F-1. Opto 22 System

Up to 255 racks, each with as many as 16 modules, can be interfaced on a single Opto 22 RS-485 bus. An ASCII communications protocol based on code letters and hex numbers is used. A typical message and reply appear in Figure F-2.

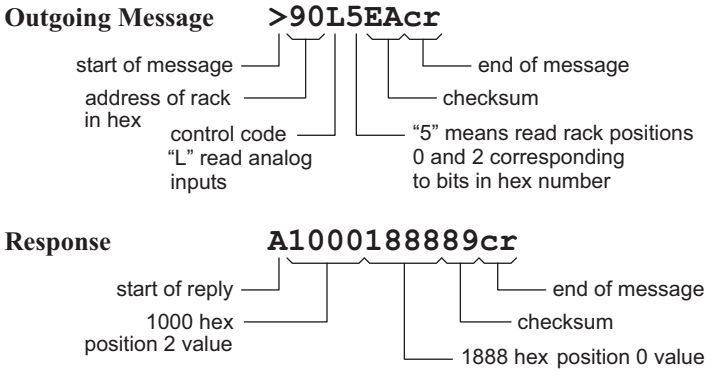


Figure F-2. Opto 22 Message and Response

The BL1100 can fit into an Opto 22 network in these ways.

1. The BL1100 can serve as the network controller. Compared to the equivalent Opto 22 controller, the LC4, the BL1100 has several advantages.
    - C programmability allows larger programs to execute much faster. The LC4 is programmable only in Basic or Forth.
    - Direct digital I/O in the controller.
    - Lower price.
    - Smaller physical size.
2. The BL1100 can serve as a combination of brain board and digital rack. The following advantages exist over the equivalent Opto 22 equipment.
    - Lower price and smaller size.
    - The ability to program the BL1100 to provide local intelligence, which is not possible with the Opto 22 brain board.
    - The ability to download software to the BL1100 over the network.

If optical isolation is required, then the BL1100 can directly control a digital or an analog rack using the parallel port, replacing the brain board entirely.

An Opto 22 network usually uses separate transmit and reply twisted pairs but, if desired, these can be combined in a single twisted pair for communication in both directions. (Separate twisted pairs make it easier to construct repeaters.)

RS-485 data communication use a differential voltage to signal a 1 or a 0. This is shown schematically in Figure F-3.

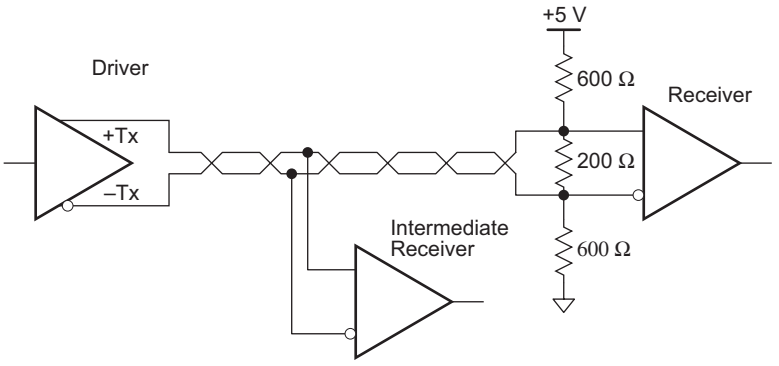


Figure F-3. RS-485 Multiple Receivers

The diagram shows only the master controller driver and the various receivers. Answers from the slaves (brain boards) are returned on another line with many drivers and only one receiver at the master controller (Figure F-4).

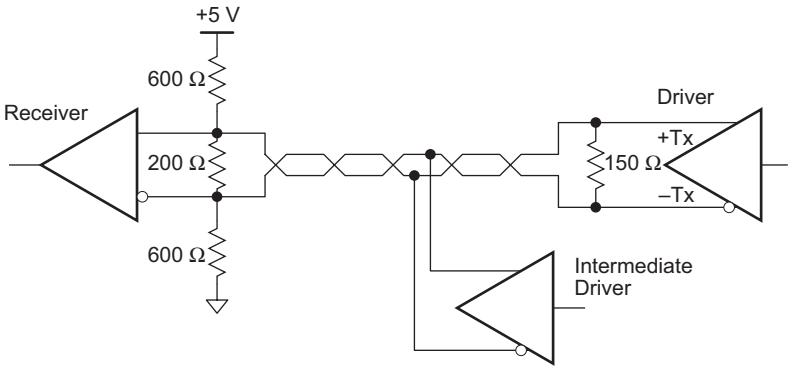


Figure F-4. RS-485 Multiple Drivers

The driver drives one line high and the other low for a 1. The polarity is reversed for a 0. The receiver is sensitive only to the relative polarity of the signals, and has a substantial tolerance (about 12 V) for a shift in the ground potential between the driver and receiver. For example, if the ground potential changes by 5 V, as might happen between two different locations separated by 1000 ft in a factory, the voltages at the driver and receiver might appear as shown in Table F-2.

Table F-2. Signal Strengths at Various Locations

Signal	At Driver	At Receiver
+TX	+2.8 V	+7.8 V
–TX	+0.4 V	+5.4 V
Reversed Signal	**At Driver**	**At Receiver**
+TX	+0.4 V	+5.4 V
–TX	+2.8 V	+7.8 V

The receiver will still detect the correct signal since it only sees that TX is greater than –TX or vice versa.

The resistors shown provide termination and bias for the signal pair. Bias is necessary to prevent noise on the line from toggling the receivers when no driver is driving the line. The receivers have about 50 mV of hysteresis. Termination is needed to prevent reflections when a pulse arrives at the end of the line. If a line is terminated with a resistor equal to the

characteristic impedance of the line, then no reflection will take place. The indicated resistor networks are intended for a twisted pair with a characteristic impedance of 100 Ω (Belden 8261 wire). The effective termination resistance is 150 Ω, which is less than the characteristic impedance, and will cause a reflection equal to 20% of the incoming voltage. The small reflection is further attenuated, since the wires have a series resistance amounting to 25 Ω per 1000 ft for #24 wire. The series resistance attenuates reflections that might disturb the signal.

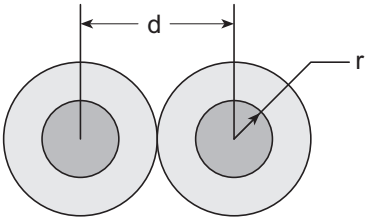


Figure F-5. Twisted Wire Pair for RS-485 Communication

Use Equation (F-1) to compute the transmission line impedance of a twisted pair. Figure F-5 illustrates the variables used in Equation (F-1).

$$R_C = \sqrt{\frac{1}{e}} \times 120 \times \ln\left(\frac{d}{r}\right) \qquad \text{(F-1)}$$

where e is the effective dielectric constant (air = 1). The effective dielectric constant is somewhere between that of air and the insulator material. If the capacitance per meter is known, use Equation (F-2).

$$R_C = \sqrt{\frac{C}{400{,}000 \times \ln(d/r)}} \qquad \text{(F-2)}$$

where C is the capacitance parameter (in picofarads).

The reflection coefficient giving the size of the reflected pulse compared to the incoming pulse is given by

$$\frac{R_1 - R_C}{R_1 + R_C}$$

where R_1 is the terminating resistor and R_C is the characteristic impedance of the line.

Most twisted pairs will have R_C between 50 and 150 Ω. Larger wires with thin insulation and a high dielectric constant will have a lower R_C. The propagation velocity will usually be between 50% and 80% of the speed of light (3.3 ns/m or about a foot per nanosecond). The propagation velocity is proportional to the inverse square root of the dielectric constant.

The higher the R_C of the cable and the larger the size of the wire, the further the signal can be sent. For #24 wire and R_C =100 Ω, the limit is about 1000 m or 3300 ft. With #20 wire, this can be extended to about 2500 m or 8000 ft without using a repeater.

Figure F-6 shows how to build a cable between a BL1100 and an Opto 22 device.

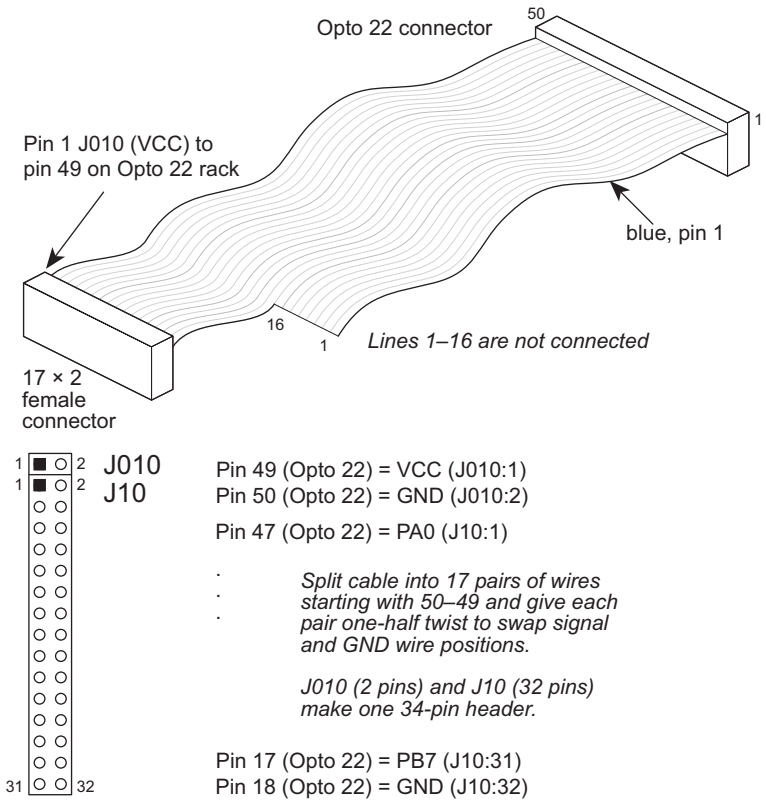


Figure F-6. Cable Between BL1100 and Opto 22 Device

Up to 255 racks, each with as many as 16 modules, can be interfaced on a single Opto 22 RS-485 bus.

APPENDIX G:
SAMPLE ANALOG APPLICATIONS

The demonstrations, or “projects,” in Appendix G explore some sample applications for the BL1100’s analog inputs.

Semiconductor Temperature Sensor

The example in Figure G-1 shows how temperatures can be measured using a semiconductor temperature sensor. The resistor network RN3 (1 kΩ resistors) can provide excitation to each of the positive inputs of the op-amps. A jumper must be installed across header J17 to enable excitation from the +5 V supply. If the channels are to be excited selectively, then individual pins can be cut on the resistor network.

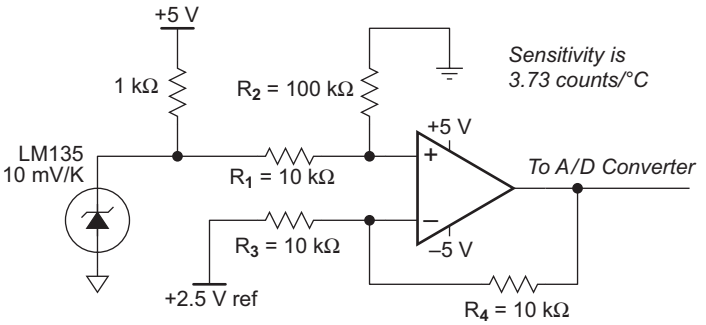


Figure G-1. Semiconductor Temperature Sensor (–100°C to +100°C)

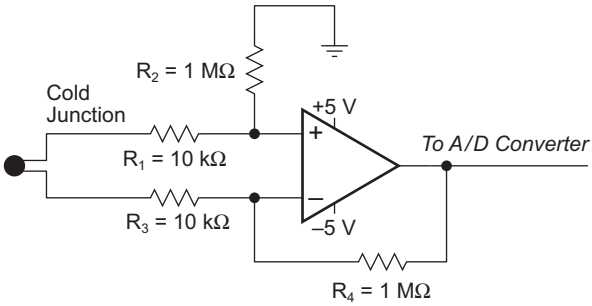
The LM135 temperature sensor measures absolute temperature, starting at 0 K or –273°C. Thus, the output is 2.7315 V at a temperature of 0°C. By using the 2.5 V reference voltage to offset the op-amp in a differential configuration, the zero point is shifted to 250 K or –23.15°C. The 10 kΩ resistor at R_4 provides a gain of 1.8181. The A/D converter accepts inputs from –2.5 V to +2.5 V. The A/D output is a 10-bit number from –512 to +511. (A value of +512, which cannot occur, corresponds to +2.5 V.) The net result is that the output of the A/D converter is 86 counts at 0°C, and is 459 counts at 100°C. The temperature in degrees Celsius can be easily calculated with Equation (G-1).

$$\begin{aligned} T_C &= -23.15 + \left(\frac{275}{1024}\right) \times AD \quad \text{output} \\ &= -23.15 + 0.26855 \times AD \quad \text{output} \end{aligned} \tag{G-1}$$

These coefficients can be adjusted by a calibration procedure for each individual temperature sensor and amplifier.

Thermocouple

A thermocouple generates small voltages, about 40 µV/°C for a copper-nickel thermocouple. The circuit in Figure G-2 uses a gain of 100 to amplify the small voltages to about 4 mV/°C.



FigureG-2. Thermocouple

The cold junction can be at the point where the thermocouple wires connect to the BL1100. Since the thermocouple measures the difference in temperature between the cold junction and the other junction, an independent temperature measurement device is needed at the cold junction. The onboard temperature sensor can be used if it has sufficient accuracy for the application. The offset in the amplifier and the nonlinearity of the thermocouple can both be compensated in software. A capacitor may be added across R_4 as a part of an input filter.

4–20 mA Loop

Many industrial-style sensors use 4–20 mA loops. The sensor is powered by the current loop and reports the value sensed by modulating the amount of current flowing in the loop. A circuit such as the one shown in Figure G-3 can be used to obtain the loop current. External power and an external 68 Ω resistor are used. The 68 Ω resistor can often be mounted on the field-wiring terminal block. This circuit gives 2.47 V for a full-scale reading.

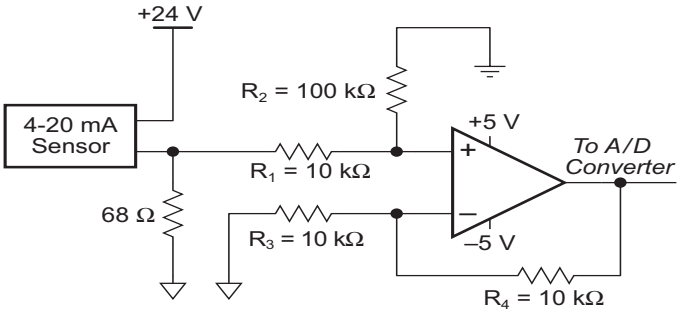


Figure G-3. 4–20 mA Loop

APPENDIX H: PLCBUS

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

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 BL1100 supports the XP8300, XP8400, XP8600, XP8900, and Exp-A/D12 expansion boards using the controller's parallel input/output port.

The BL1100 also has a series of expansion boards exclusive to the BL1100. These boards and their use are described in Appendix I.

Table H-1 lists all Z-World expansion devices that are supported on the PLCBus.

Table H-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 H-1 shows the pin layout for the PLCBus connector.

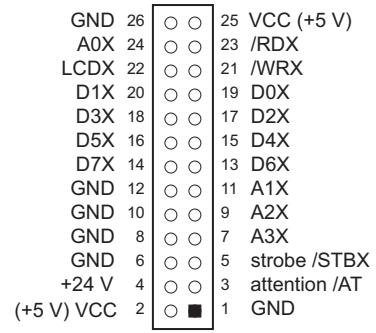


Figure H-1. PLCBus Pin Diagram

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

The LCD bus consists of the following lines.

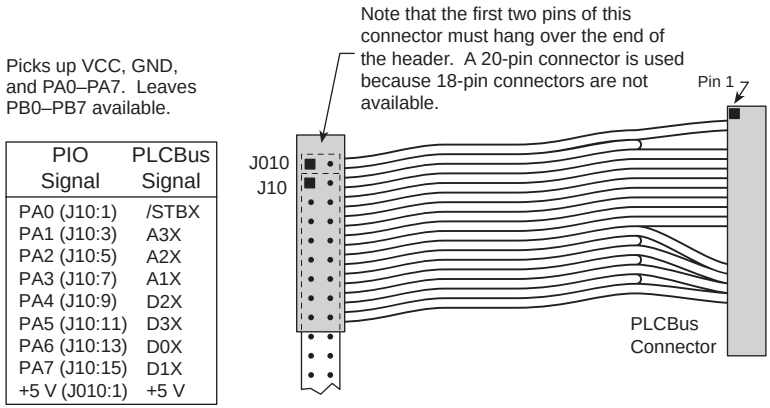
- LCDX—positive-going strobe.
- /RDX—negative-going strobe for read.
- /WRX—negative-going strobe for write.
- A0X—address line for LCD register selection.
- D0X-D7X—bidirectional data lines (shared with expansion bus).

The 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.

Connecting a Z-World expansion board to a BL1100 requires a special cable. Fasten the cable's 20-pin connector to headers J010 and J10 as shown in Figure H-2. Pins 1 and 2 of the connector must hang over the end of the headers. Fasten the cable's PLCBus connector to header P1 or P2 of the expansion board, observing the orientation of pin 1, as shown.



PIO Signal	PLCBus Signal
PA0 (J10:1)	/STBX
PA1 (J10:3)	A3X
PA2 (J10:5)	A2X
PA3 (J10:7)	A1X
PA4 (J10:9)	D2X
PA5 (J10:11)	D3X
PA6 (J10:13)	D0X
PA7 (J10:15)	D1X
+5 V (J010:1)	+5 V

Figure H-2. Connecting BL1100 to Expansion Boards

Use an external power supply with expansion boards connected to the BL1100. There is no provision in the special cable to supply +24 V from the controller to header P1 or P2 on the expansion boards.

Software for interfacing the BL1100's PIO port to a PLCBus port may be found in the Dynamic C **PBUS_LG.LIB** library.

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

Table H-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 H-3. An "x" indicates that the address bit may be a "1" or a "0."

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

Table H-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 H-5 lists the libraries.

Table H-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
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`**.

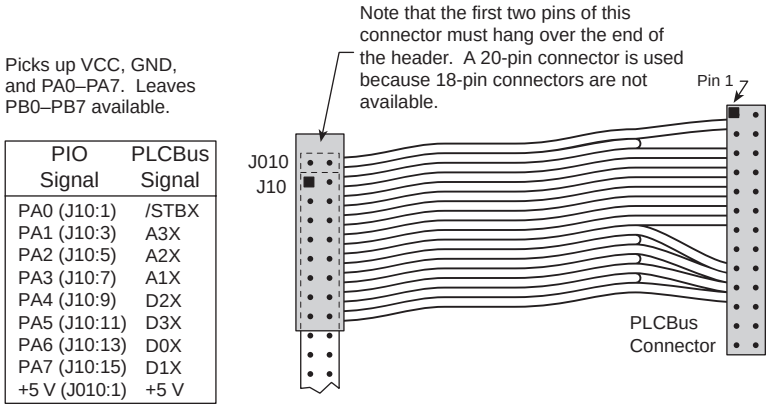
APPENDIX I:
SIMULATED PLCBUS CONNECTION

Certain standard Z-World expansion boards may be used with the BL1100, as explained in Appendix H. Appendix I explains how to make a simulated PLCBus connected using the PIO header on the BL1100. Selected software functions are also described.

PIO Port Connections

Standard Z-World Expansion Boards

Expansion boards may be connected to header J010/J10 on the BL1100. To add expansion boards, the user must make a custom cable (Z-World part number 540-0015). To assist with making the connection via a custom-made ribbon cable, the table in Figure I-1 maps the signals from the controller's PIO to the expansion board PLCBus header. Fasten the cable's 20-pin connector to the combined headers J010 and J10 as shown in Figure I-1. Pins 1 and 2 of the custom cable connector must hang over the end of the combined headers. Fasten the cable's PLCBus connector to expansion board header P1 or P2. Note the orientation of pin 1.



PIO Signal	PLCBus Signal
PA0 (J10:1)	/STBX
PA1 (J10:3)	A3X
PA2 (J10:5)	A2X
PA3 (J10:7)	A1X
PA4 (J10:9)	D2X
PA5 (J10:11)	D3X
PA6 (J10:13)	D0X
PA7 (J10:15)	D1X
+5 V (J010:1)	+5 V

Figure I-1. BL1100 Expander Cable Connection

Software for interfacing the BL1100's PIO port to a PLCBus port may be found in the Dynamic C **`EZIOMGPL.LIB`** library (for XP8900 Series expansion boards) or the **`PBUS_LG.LIB`** library.

Use an external power supply when connecting expansion boards to the BL1100. There is no provision in the custom cable to supply +24 V from the controller to header P1 or P2 on the expansion board.

Software Drivers

Using Expansion Boards with PIO 1 Port A

A PLCBus driver is implemented using the 8-bit PIO 1 Port A (PIOA). With the BL1100, the developer is limited to 4-bit PLCBus peripherals. An attention line (/AT) is not available. The wiring connections shown in Figure I-1 are used.

General-Purpose Drivers

- **`void PBus12_Addr( int addr )`**

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

  PARAMETER: **`addr`** is the 12-bit physical address, with the first and third nibbles swapped (the most significant nibble is in the low four bits).

  RETURN VALUE: None.

- **`void PBus4_Write( char data )`**

  Writes 4-bit data on PLCBus. The address must be set by a call to **`PBus12_Addr`** before calling this function.

  PARAMETER: **`data`** should contain the value to write in the lower four bits.

  RETURN VALUE: None.

- **`int PBus4_Read0( )`**

  Reads 4 bits of data from the PLCBus using a BUSRD0 cycle. The address must be set by a call to **`PBus12_Addr`** before calling this function.

  RETURN VALUE: PLCBus data in the lower 4 bits (the upper bits are undefined).

- **`int PBus4_Read1( )`**

  Reads 4 bits of data from the PLCBus using a BUSRD1 cycle. The address must be set by a call to **`PBus12_Addr`** before calling this function.

  RETURN VALUE: PLCBus data in the lower 4 bits (the upper bits are undefined).

- **`int PBus4_ReadSp( )`**

  Reads 4 bits of data from the PLCBus using a BUSSPARE cycle. The address must be set by a call to **`PBus12_Addr`** before calling this function.

  RETURN VALUE: PLCBus data in the lower 4 bits (the upper bits are undefined).

- **`void Reset_PBus( )`**

  Resets the PLCBus.

  RETURN VALUE: None.

- **`int Poll_PBus_Node( int addr )`**

  Polls a PLCBus device by performing a BUSRD0 cycle and checking the low bit of the returned value.

  PARAMETER: **`addr`** is the 12-bit physical address of the device, with the first and third nibbles swapped.

  RETURN VALUE: Returns 1 if node answers poll, 0 if not.

- **`void Reset_PBus_Wait( )`**

  Provides the minimum delay necessary for PLCBus expansion boards after a bus reset, using a 9 MHz CPU. This delay will be insufficient for a faster CPU, and must be increased.

  RETURN VALUE: None.

Relay Expansion Board Drivers

- **`int Relay_Board_Addr( int board )`**

  Converts a logical relay board address to a physical PLCBus address.

  PARAMETER: **`board`** must be a number between 0 and 63 representing the relay board to access. This number has the binary form pppzyx, where ppp is determined by the board PAL number, and x, y, and z are determined by jumpers on header J1 on the expansion board.

  ppp values of 000, 001, 010,…, correspond to PAL numbers of FPO4500, FPO4510, FPO4520,…

  x, y, and z correspond to a jumper across J1 pins 1–2, 3–4, and 5–6 respectively (0 = closed, 1 = open). The resulting address is in the form pppx000y000z.

  RETURN VALUE: The PLCBus address of the board specified, with the first and third nibbles swapped. This address may be passed directly to **`PBus12_Addr`**.

- **`void Set_PBus_Relay( int board, int relay, int state )`**

  Sets a relay on an expansion bus relay board.

  PARAMETER: **`board`** must be a number between 0 and 63 representing the relay board to access. This number has the binary form pppzyx, where ppp is determined by the board PAL number, and x, y, and z are determined by jumpers on header J1 on the expansion board.

  ppp values of 000, 001, 010,…, correspond to PAL numbers of FPO4500, FPO4510, FPO4520,…

  x, y, and z correspond to a jumper across J1 pins 1–2, 3–4, and 5–6 respectively (0 = closed, 1 = open).

  `relay` is the relay number on the board (0–5 for XP8300, 0–7 for XP8400).

  `state` must be 1 to turn the relay on and 0 to turn the relay off.

  RETURN VALUE: None.

D/A Converter Expansion Board Drivers

- **`int DAC_Board_Addr( int bd )`**

  Converts a logical DAC board address to a physical PLCBus address.

  PARAMETER: **`bd`** must be a number between 0 and 63 representing the DAC board to access. This number has the binary form pppzyx where ppp is determined by the board PAL number, and x, y, and z are determined by jumpers on header J3 on the expansion board.

  ppp values of 000, 001, 010,…, correspond to PAL numbers of FPO4800, FPO4810, FPO4820,…

  x, y, and z correspond to a jumper across J3 pins 1–2, 3–4, and 5–6 respectively (0 = closed, 1 = open). The resulting address is in the form pppx001y000z.

  RETURN VALUE: The PLCBus address of the board specified, with the first and third nibbles swapped. This address may be passed directly to **`PBus12_Addr`**.

- **`void Write_DAC1( int val )`**

  Loads Register A of DAC #1 with the given 12-bit value. The board address must have been set previously with a call to **`PBus12_Addr`**. The value in **`val`** will not actually be output until **`Latch_DAC1`** is called.

  RETURN VALUE: None.

- **`void Write_DAC2( int val )`**

  Loads Register A of DAC #2 with the given 12-bit value. The board address must have been set previously with a call to **`PBus12_Addr`**. The value in **`val`** will not actually be output until **`Latch_DAC2`** is called.

  RETURN VALUE: None.

- **`void Latch_DAC1( )`**

  Moves the value from Register A of DAC #1 to Register B. The value in Register B represents the actual DAC output. The board address must have been set previously with a call to **`PBus12_Addr`**, and the value should have been loaded into Register A with a call to **`Write_DAC1`**.

  RETURN VALUE: None.

- **`void Latch_DAC2( )`**

  Moves the value from Register Aof DAC #2 to Register B. The value in Register B represents the actual DAC output. The board address must have been set previously with a call to **`PBus12_Addr`**, and the value should have been loaded into Register A with a call to **`Write_DAC2`**.

  RETURN VALUE: None.

- **`void Init_DAC( )`**

  Initializes DAC board and sets all output values to 0. Call this function before writing data to the DAC. The board address must have been set previously with a call to **`PBus12_Addr`**.

  RETURN VALUE: None.

- **`void Set_DAC1( int val )`**

  Sets DAC #1 to the value specified in the lower 12 bits of **`val`**. In the voltage output mode (J1 pins 2–3 jumpered), Vout = (**`val`**/4096) * 10.22 V with the Z-World default settings. In the current output mode (J1 pins 1–2 jumpered), Iout = (**`val`**/4096) * 22 mA with the Z-World default settings. The board address must have been set previously with a call to **`PBus12_Addr`**.

  RETURN VALUE: None.

- **`void Set_DAC2( int val )`**

  Sets DAC #2 to the value specified in the lower 12 bits of **`val`**. In the voltage output mode (J1 pins 2–3 jumpered), Vout = (**`val`**/4096) * 10.22 V with the Z-World default settings. In the current output mode (J1 pins 1–2 jumpered), Iout = (**`val`**/4096) * 22 mA with the Z-World default settings. The board address must have been set previously with a call to **`PBus12_Addr`**.

  RETURN VALUE: None.

APPENDIX J: STANDALONE OPERATION

Appendix J provides information about running the BL1100 after it is programmed, burning EPROMs, and downloading software remotely.

There are two ways to run an application with the BL1100 disconnected from the PC. The application may either be burned into a new EPROM to take the place of the existing Dynamic C EPROM, or it can run from (battery-backed) static RAM.

There are a few considerations when running an application as a standalone program.

1. To run correctly, the code should not have any **printf**, **putchar**, **getchar**, or **kbhit** function calls. These attempt to talk to the PC and will lock up the program in a standalone mode.
2. Defined constants (for example, `float pi = 3.141593;`) will now be truly constant for EPROM-based applications, since they are burned in. In a RAM-based application, constants are be initially downloaded with their declared value, but can be modified by software. That is, the statement `pi = 1.0;` will affect a RAM-based application, but not an EPROM-based one.
3. The timing of the application will change. Since the application is no longer connected to Dynamic C, it will run a little faster. Debugging code will not be loaded into EPROM (or RAM if the **nodebug** option is used). Be especially careful when any hardware interfacing depends on timing.
4. It is possible, although rare, for a program not to run out of ROM when it could run out of RAM. This is because of the complex paging scheme used by the compiler to manage code and data spaces in the interactive environment. This should not be a problem in small programs.

Option 1: Burn an EPROM

A program may be "burned" into a new EPROM chip that then replaces the Dynamic C EPROM on the controller. This requires an EPROM burner and a blank EPROM.

1. Compile the program to an **.ROM** file by selecting the **Compile to File** option in the **COMPILE** menu. The BL1000 must be connected to the PC running Dynamic C during this step because essential library routines must be uploaded from the Dynamic C EPROM and linked to the resulting file. The output is a binary file or an Intel hex format file with the program name and an **.ROM** extension.
2. Exit Dynamic C.
3. Use the binary or the hex format file to burn an EPROM with an EPROM burner. Refer to the burner instructions for specific information on this.

4. Replace the Dynamic C EPROM on the board with the new one. If the new EPROM has s different size than the Dynamic C EPROM, a jumper on the BL1000 may need to be moved.
5. When power is applied to the board, the new program will have complete control. As far as the controller is concerned, Dynamic C no longer exists.

Option 2: Use Battery-Backed Static RAM

When a program is compiled from a PC, it is actually stored in the static RAM and executed from there. Since the RAM has a battery to keep the code intact, the code will remain in the board even when power is removed.

1. Compile the program to RAM in the usual manner. If the code compiles with no errors, the watch window pop up. Dynamic C still assumes that the program will run with the PC attached.
2. Set the appropriate jumpers on the board to indicate to Dynamic C that the BL1100 will now run in standalone mode from static RAM.
3. Press the RESET button on the controller or cycle power off-on.
4. The program will start running and break off communication with the PC. A message on the PC will indicate that communication was lost. (The PC has no way of knowing that the above events took place. It just sees that communication with the controller has stopped.)
5. The controller may now be disconnected from the PC.

Reliability

Since an EPROM is a read-only device, the program cannot be accidentally written over because of a software bug. Code and data in RAM, however, does incur the risk of being overwritten. A misused pointer or a stack overflow can cause a program to overwrite itself.

Program Life

An EPROM-based program is, for practical purposes, permanent. Programs in battery-backed RAM have an indefinitely long life unless the battery dies. A battery's life is around 3 "power-off" years. The battery does not discharge significantly with power applied. If the controller is never powered down, then the program essentially lasts forever. Rarely, the RAM or EPROM chips can fail. This possibility could be taken into consideration for long-term use.

Speed

Debugger information a RAM-based program is not present in an EPROM-based program since the compiler knows that a program in EPROM will not be using the Dynamic C interface. This results in a slight increase in performance for EPROM-based programs.

The **nodebug** option may be used in a program to eliminate this difference. Debugging information will not be stored in RAM. But it will not be possible to debug the program interactively from a PC. Refer to the Dynamic C manuals for details.

Data Space

Using an EPROM to store code (and constants) leaves more space in the RAM to hold variable data.

Cost

A typical PC-based EPROM burner costs several hundred dollars. An EPROM eraser can be found for around $50. On the other hand, the static RAM is already paid for. No other accessories are necessary to execute code out of RAM.

Ease

For developing a single board, it is easier and quicker to run from RAM. For volume applications, it will become very tedious to connect each controller to the PC and compile code into it.

Remote Downloading

Code based in RAM can be modified by an outside source. An example of this is a remote connection through a modem where new code could be downloaded to the controller and executed. In this case, a *monitor* program, burned into ROM, is needed to serve as a master controller to load and start execution of the programs and to receive control when the executed program finishes.

The monitor program also gains control if the board is reset through hardware, either by power-on, watchdog timeout, or reset. An external hardware reset line can also be installed so that the computer that is downloading the program can force a hardware reset of the as a sure way of gaining control. Install an external hardware reset by using one of the RS-485 handshaking lines (CTS or RXC). The output of the receiver for the line chosen can be connected to the same reset line as the reset push-button. Then the external input will be a direct reset.

Review the memory map and program format before writing a monitor program for downloading software. Figure H-1 shows the EPROM memory layout. All changeable data areas are in upper memory and must always be in RAM. The user program can be in EPROM or in RAM, beginning with the interrupt vectors. The BIOS is always in EPROM.

The 64K space shown in Figure J-1 is the code space visible to the microprocessor. The memory management unit of the Z180, which is between the microprocessor and physical memory, can address 1 M of memory.

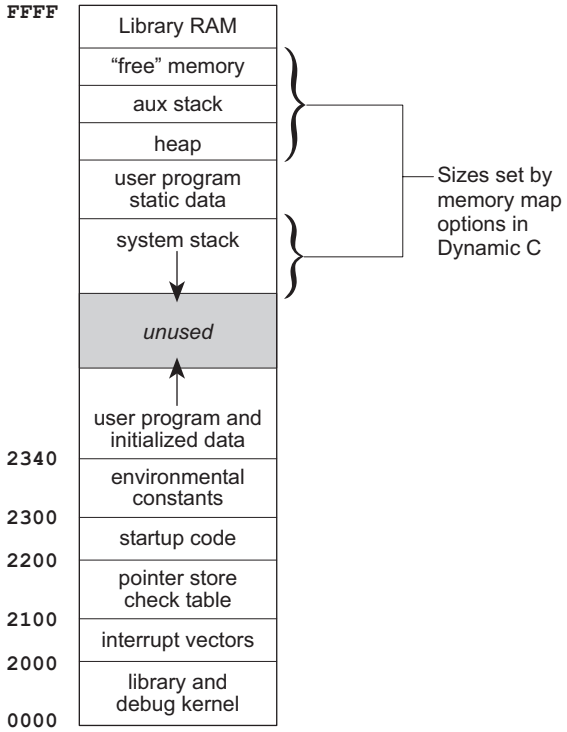


Figure J-1. EPROM Memory Layout

See Appendix C, "Memory, I/O Map, and Interrupt Vectors," for details on memory management.

The ROM chip occupies the space from 0 to 256K in the physical memory. If a ROM is smaller than 256K, data will seem to appear at multiple addresses. If, for example, a ROM has the standard 32K, ROM addresses are evaluated modulo 32K. Each datum “appears” at eight addresses.

The RAM chip occupies the space from 256K to 768K (40000 to BFFFF hex). Once again, the contents of the RAM will be appear to be repeated in the allocated space unless a 512K RAM chip is installed. The top of the RAM space—the last “image” of the RAM contents—is always mapped to the top of the logical 64K microprocessor code space.

To download and run a program, burn the monitor program into EPROM. When the BL1100 powers up, it automatically executes the monitor program, which can then download the program, storing the program in RAM. Part of the BL1100 RAM is mapped out of the microprocessor space, but it can be accessed by using the DMA block copy feature. Once the program is in memory, the monitor program will change the memory mapping registers and jump to the startup code for the downloaded program. This can be made to happen by copying a small routine to high memory and then jumping to that routine. This is code that actually changes the memory mapping registers and jumps to the startup code.

It is possible for the program running in RAM to return to the monitor program by a similar trick in reverse. By manipulating the size of free memory (in the **OPTIONS** menu), the data spaces of the programs can be assigned separate memory areas. It is also possible to have shared data areas by careful attention to data declarations. Data declarations consume memory in reverse order of their declaration. The data area grows down, although each array or structure is defined in a forward direction in memory.

A major application for downloading programs is distributed processing systems that are subject to frequent software updates or which need to be configured differently depending on the site or the application. In some cases, it may simpler or more convenient to download the software.

If the downloaded program will not return control to the monitor program, then the BL1100 has to be reset externally. If a separate reset line is not available and the power cannot be interrupted to reset the BL1100, then it is necessary to take elaborate precautions against the possibility of the BL1100 losing communication with the base computer. This could happen if a software fault or a transient electrical disturbance causes the BL1100 to enter an endless loop and stop communicating. If the BL11To write a monitor program for downloading software, you must understand the memory map and program format. The diagram following shows the layout of the memory for EPROM. All changeable data areas are in upper memory and must always be in RAM. The user program can be in EPROM

or in RAM, beginning with the interrupt vectors. The “BIOS” is always in EPROM. When you create a hex program file with <**Ctrl F3**> and the program **DCROM.EXE**, there are two options (set with the **Memory Options** command of the **SETUP** menu):

1. EPROM model. The hex file includes everything from 0000 to the top of the user program.
2. Hex for download model. Hex file same as (1), but the ID flag identifying the file as residing in ROM is removed.

The 64K space shown in the diagram below is the code space visible to the microprocessor. The memory management unit of the Z180, which is between the microprocessor and physical memory, can address one megabyte of memory. 00 is in a remote location or is otherwise inaccessible, then such a failure can be expensive. The watchdog timer provides a suitable means of recovery from such a situation.

APPENDIX K:
BL1100 EXPANSION BOARDS

Appendix K provides complete information about using the expansion boards unique to the BL1100. The following sections are included.

- Introduction
- Installation
- Subsystems
- Software
- Board Layouts
- I/O Map
- Jumper and Header Specifications

Introduction

Four I/O expansion boards are available to provide additional I/O channels for the BL1100. These boards and their features are listed in Table K-1.

Table K-1. BL1100 Expansion Board Features

Expansion Board	Features
DGL*	Two 8-bit PIOs Two 4-bit optically isolated PIOs Eight TTL-level buffered inputs
MUX	One 8-bit PIO One 6-bit PIO One 4-bit PIO One 4-channel analog mux
ADC	One 8-bit PIO Two 4-bit optically isolated PIOs Eight TTL-level inputs One PWM channel One 20-bit A/D converter input One instrumentation amplifier One 4-channel analog mux
DGL96**	96 digital I/O points Optional op-amp and power booster for BL1100 D/A converter output

* Available with or without Wago connectors

** Works only on BL1100/BL1110, nonstacking version available

The board layouts for these expansion boards are shown in Figures K-20 to K-23 at the end of this appendix.

Installation

Expansion boards can be stacked—up to four deep—on the BL1100. They snap together. The connection to the BL1100 is through the 60-pin BL1100 expansion bus (header J24 on the BL1100 and header J2 on the expansion boards) and the BL1100's 12-pin header J12. Pins 5–12 of header J12 match the 8 pins of J3 on the expansion boards.

These steps describe the installation of expansion boards on the BL1100. Refer to Figures K-1 and K-2 for the locations of headers J24, J12, and J28 on the BL1100.

1. Align header J2 of the expansion board with header J24 of the BL1100. The eight pins of header J3 on the expansion board connect to pins 5–12 of header J12 on the BL1100. If you are installing a DGL96 expansion board with the optional power amplifier, connect header H28 on the DGL96 to header J28 of the BL1100. This DGL96 expansion board must be installed first.

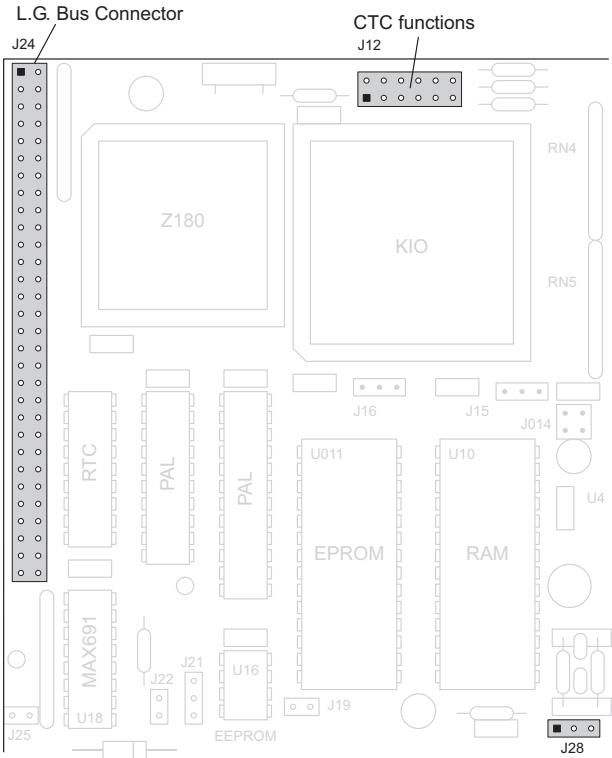


Figure K-1. Header Locations Used for Expansion Board Connections

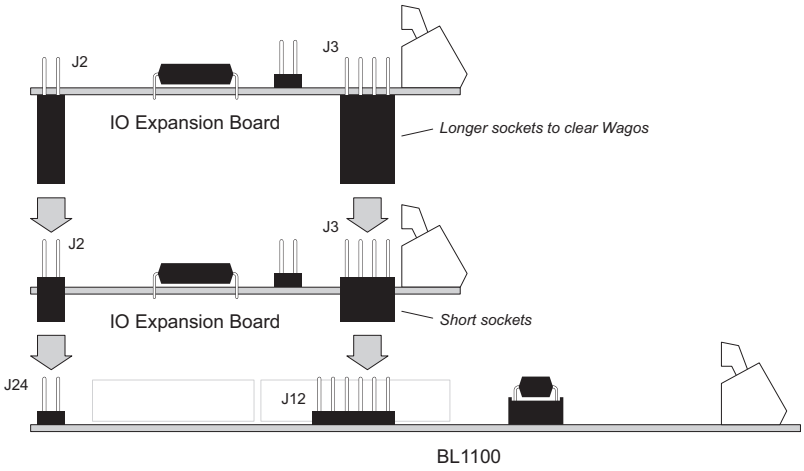


Figure K-2. Stacking Installation

If any jumpers were installed on pins 5–12 on header J12 to change the default counter/timer circuit configurations, be sure to move these jumpers to the top expansion board in the stack.

Only one DGL96 with the optional power amplifier may be used.

2. Apply firm pressure to the expansion board, directly over J2, J3, and, if applicable, H28, until the sockets underneath the expansion board completely engage the pins on the BL1100.

   When you are installing more than one expansion board, the sockets of the one board engage the pins of J2 and J3 of the expansion board below it, as shown in Figure K-2.

Address Mapping for Multiple Cards

Since up to four expansion cards can be installed, each expansion card requires a unique address.

The ADC, MUX and DGL expansion boards each have seven possible addresses. For these boards, the upper hex digit of the address is selected with pins 1–4 of header J10 as shown in Figure K-3.

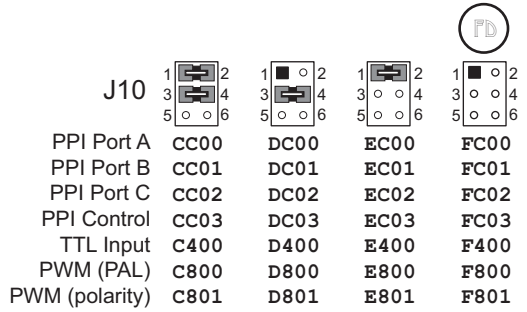


Figure K-3. Setting ADC, MUX, and DGL Adfdresses

The DGL96 board has 24 addressable I/O registers grouped into these six sets of four.

Data Register A	Control Register A
Data Register B	Control Register B

These six groups of four registers correspond to the six Zilog PIO chips on the board. Pins 1–4 of header H10 on the DGL96 expansion board set the "board address," which is represented in the upper hex digit of the register address as shown in Figure K-4.

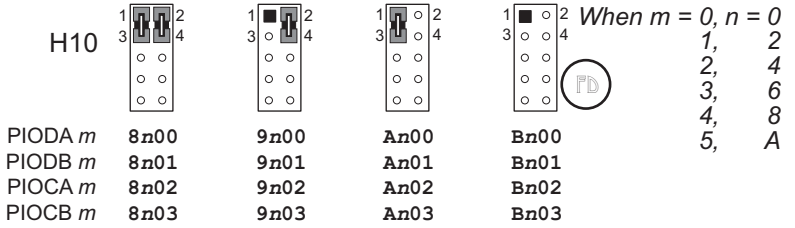


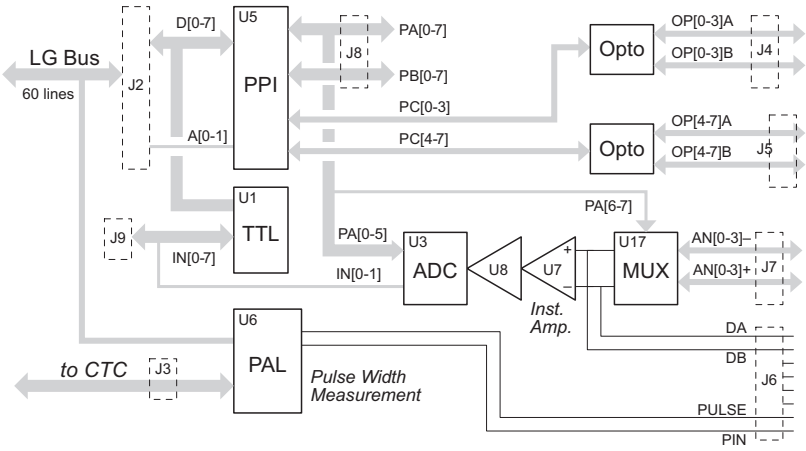
Figure K-4. Setting DGL96 Adfdresses

Thus, the addresses of the four expansion board registers range from 8xxx to Fxxx: the DGL96 expansion board registers have addresses ranging from 8xxx to Bxxx, and the ADC, MUX, and DGL expansion boards have addresses ranging from Cxxx to Fxxx. All the registers are individually addressable. The registers from Cxxx to Fxxx are named with symbolic constants in the **`IOEXPAND.LIB`** library. The DGL96 registers do not need names because of the way the software in the **`96IO.LIB`** library is written.

These libraries are discussed in the "Software" section later in this appendix.

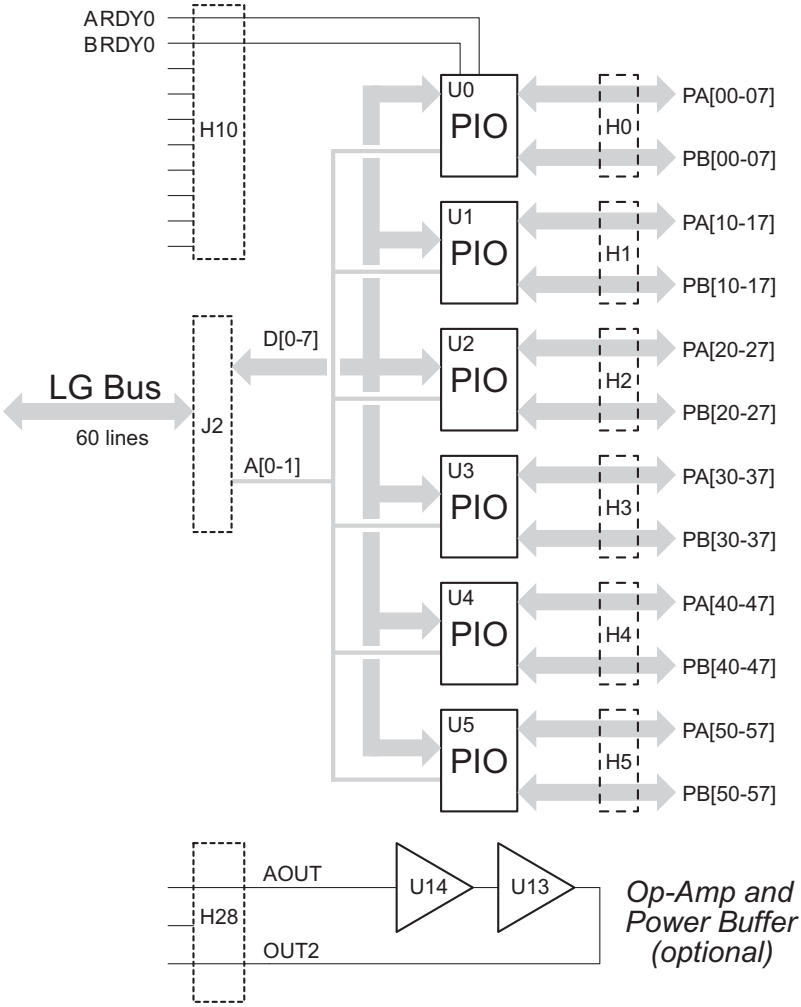
Subsystems

Figure K-5(a) illustrates the subsystems on the ADC expansion board. The DGL and MUX boards are similar, but do not have all the features. The DGL96 subsystems are shown in Figure K-5(b).



(a) ADC, MUX, and DGL

Figure K-5. Expansion Board Subsystems



(b) DGL96

Figure K-5. Expansion Board Subsystems

Digital I/O

The DGL, MUX, and ADC boards have two types of digital I/O—the 82C55 programmable peripheral interface (the PPI) and a 74HC244 chip. The DGL96 board has 6 Zilog PIOs, which provide a total of 96 individual inputs and outputs. Each of the 6 PIOs has two 8-bit ports, A and B, and four registers, as shown below.

Data Register A	Control Register A
Data Register B	Control Register B

For each PIO (0–5) there is a corresponding 26-pin header (H0–H5). For each PIO chip, there are also two pull-up resistor networks: RN0 and RN1, which correspond to PIO 0; RN2 and RN3, which correspond to PIO 1; and so on, up to RN10 and RN11.

For more information on how to read and write a PIO, refer to the Zilog ***Z80 PIO Technical Manual***.

Programmable Peripheral Interface (PPI)

The 82C55 is a general-purpose programmable I/O device, and is shown in Figure K-6. It has 24 I/O pins divided into two 8-bit ports (A and B) and two 4-bit ports (C1 and C2). Each can be programmed as either an input or output port.

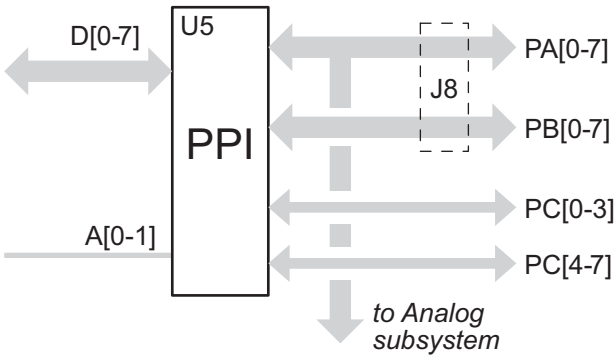


Figure K-6. Programmable Peripheral Interface

The Port A lines control the analog MUX and the ADC. Ports A and B connect to header J8. The PPI ports are accessed via the BL1100 bus (D0–D7).

When used as an output, each line exhibits an ON resistance between 40 Ω and 100 Ω for sinking current, and between 80 Ω and 200 Ω for sourcing current. The ON resistance is low enough for the outputs to sink 20 mA and source 10 mA easily, but use caution not to exceed the power dissipation of the package by sinking or sourcing too much total current. The

optical isolators on ports C1 and C2 can sink much larger currents than the 82C55, and are intended for driving larger loads.

Figure K-7 shows the Port A and Port B signals on the expansion board header J8.

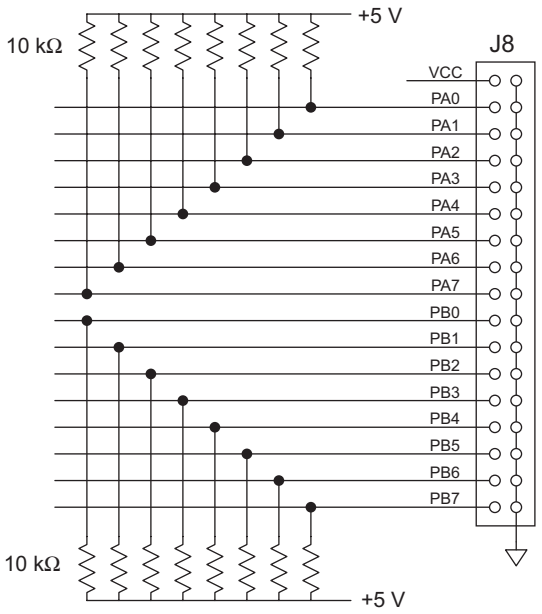


Figure K-7. Expansion Board Header J8 Pinout

Port A

Port A is used to control the multiplexer and the A/D converter on the MUX and ADC boards, and must be configured as an output on these boards. Only lines 6 and 7 are used on the MUX, leaving lines 0–5 for the user's application. Lines 0–5 of Port A control the A/D converter on the ADC board and are unavailable for other applications.

Port B

Port B is fully available on the DGL, MUX, and ADC boards.

Port C

Port C has two groups of 4 lines each: C1 and C2.

Port C of the PPI on the DGL and ADC boards is connected to optical isolators (4N26). The optoisolators can be used for either DC input or "solid state relay" output. The line can be changed from an input to an output by reversing the optical isolator in its socket and changing the jumper settings of J14 or J15 on the expansion board, as shown in Figure K-8.

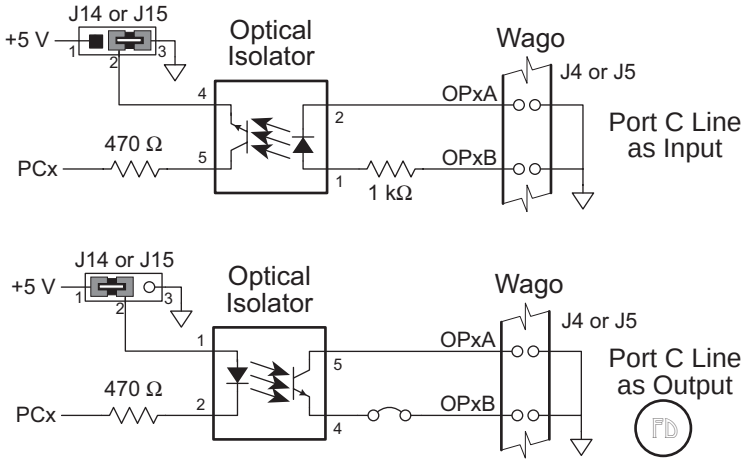


Figure K-8. Port C Lines Used as Inputs or Outputs on DGL and ADC

As input, the light source is driven by an external device and the photoconductor output is read by Port C of the PPI. The eight optical isolators, which are associated with port C input bits, should be installed with the light source pins (pins 1–3) connected to the Wago connector. Place pin 1 so that it is towards the Wago connectors (the notch on the chip opposite the notch on the socket.) Pins 2 and 3 of J14 or J15 must be connected.

As output, the light source is driven by Port C. The optical isolator output can be used via Wago connectors J4 and J5 as a relay to control external devices. The optical isolator chips, which are associated with Port C output bits, should be installed with the light source pins (pins 1–3) connected to the PPI. Pin 1 of the chip should be inserted into pin 1 of the socket (the notch on the chip matching the notch on the socket). Pins 1 and 2 of J14 or J15 must be connected.

A variety of optical isolators can be used for different applications in terms of light source characteristics, speed of response, output current and power dissipation, as long as they have a pinout matching the socket on the expansion board. For example, the 4N26 optical isolator has a transistor output with a maximum 150 mA collector current at a maximum of 30 V. This is the device that is installed on the DGL and ADC expansion boards. A 4N29 has a photo-Darlington output designed for high sensitivity at low input current.

The 4N40 is an SCR output optical isolator designed for applications requiring isolation between a TTL signal and AC lines up to 400 V. The output driver of the 4N40 has a maximum current of 300 mA at 400 V AC. Figure K-9 shows a circuit with the 4N40 SCR output optical isolator.

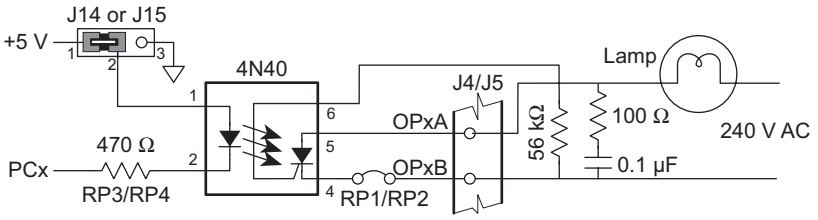


Figure K-9. Schematic of 4N40 SCR Optical Isolator Use

The H11AA1 optical isolator has an AC input and a transistor output, and is designed for applications requiring the detection of AC signals. The detection of AC signals requires repeated sampling of the input port. This is because the output of the optical isolator is driven by a pair of LEDs. Each LED only emits during one half of the AC cycle. So there is a brief period, when the cycle is at zero, when neither LED is emitting and there is no output. The Dynamic C sample program **`IOEACCNT.C`** helps determine the minimum number of samples needed to test for AC.

The H11AA1 and the 4N40 are available in a kit from Z-World. For ordering information or assistance, call a Z-World Sales Representative at (530) 757-3737.

The MUX board has no optical isolators. Port C is available through a 1×16 header where RP3 and RP4 would be as shown in Figure K-10.

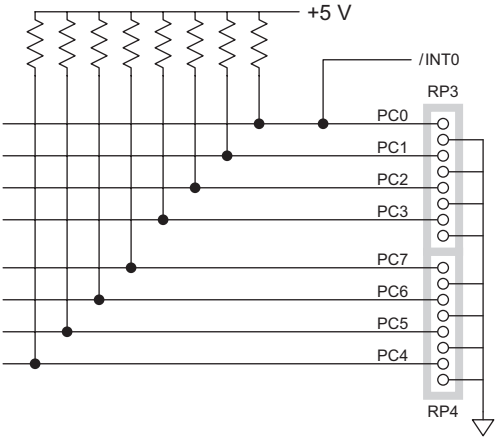


Figure K-10. Port C Lines on MUX Expansion Board

Port C bit 0 is connected to /INT1 through PAL U2. Unexpected interrupts are possible if INT1 is enabled on the BL1100 for use with other peripherals. However, this also means that the BL1100 can be interrupted using an optically isolated input. Mode 1 (described below) supports an interrupt on Port B caused by the strobe signal.

Operating Modes

Three operating modes are supported by the programmable peripheral interface (PPI).

Mode 0

Mode 0 is the "input/output mode," and supports two 8-bit ports and two 4-bit ports. Each port can be either input or output, but as a port, not as individual pins. Outputs are latched and inputs are not latched. Port A and Port B are connected to a 34-pin header (J8). Port C's two 4-bit ports can be configured individually as either inputs or outputs.

Mode 0 is the recommended mode of operation when Port C is connected to optical isolators, which makes it difficult to use Port C for the handshaking needed by the other two modes.

Mode 1

Mode 1 is the "strobed I/O mode," with Ports A and B using the lines on Port C for handshaking. Port C bit 0 can be used as an interrupt request signal to INT1 for I/O through Port B.

Mode 2

Mode 2 is the "strobed bidirectional bus I/O mode," with Port A acting as a bidirectional port with latched inputs and outputs. Five bits of Port C are used for control and status information. Mode 2 cannot be used with the MUX and the ADC.

Upon reset, all ports are in Mode 0 with all 24 port lines held high. No additional initialization is required for the 82C55 if all lines are for input.

Whenever the operational mode is changed, all of the output registers are reset and the output pulled low. However, low output can cause problems. When the output port is driving an optical isolator for output, a low signal will turn the optical isolator on. The PPI initialization function **`exp_init`** automatically sets these lines to a specified level.

TTL Input Buffer

There are eight TTL-level buffered inputs on the DGL and ADC boards. The **`inport`** function is used to read signals from the 74HC244 TTL buffer by reading register C400, D400, E400, or F400, depending on which board is being referenced.

The names of these registers are listed at the end of this appendix.

Note, however, that the first two TTL input lines on the ADC board are used to read the 20-bit A/D converter, and are not available for use as regular inputs. Place a jumper across header J9 to connect signals to the TTL buffer (U1). Figure K-11 shows the TTL buffer.

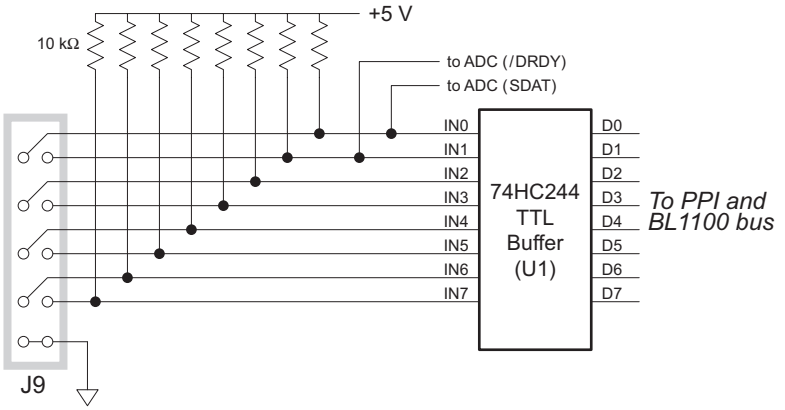


Figure K-11. TTL Buffer

Pulse Width Measurement

The ADC expansion board is equipped with a PAL for measuring external pulses with wave forms of any duty cycle with a resolution of about 1 µs. This PAL occupies socket U6, which has a number of connections to the BL1100 CTCs and clock outputs. This socket can be used for other purposes involving timer-based measurement. The FP04100 PAL supplied is for pulse width measurement. Figure K-12 shows the signals on U6.

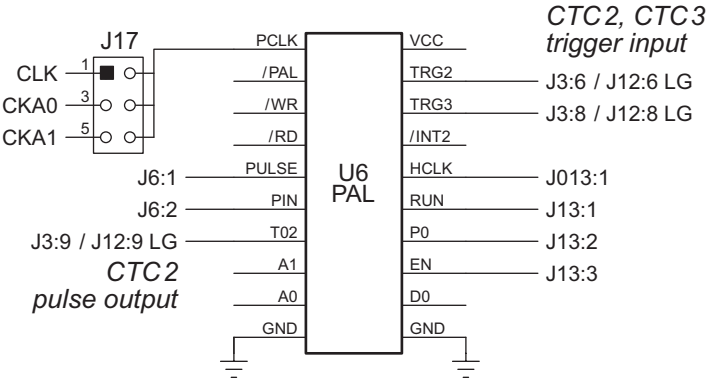


Figure K-12. Socket U6 Pinout

Two registers on an ADC board are used for pulse width measurement. Table K-2 provides the names of these registers for the up to four boards that can be stacked on the BL1100.

Table K-2. ADC Expansion Board PWM Registers

Register	Address	Description
PAL_C	C800	Write 1 to enable PAL U6. Write 0 to disable it.
POL_C	C801	Write 1 for positive pulses, 0 for negative pulses.
PAL_D	D800	Write 1 to enable PAL U6. Write 0 to disable it.
POL_D	D801	Write 1 for positive pulses, 0 for negative pulses.
PAL_E	E800	Write 1 to enable PAL U6. Write 0 to disable it.
POL_E	E801	Write 1 for positive pulses, 0 for negative pulses.
PAL	F800	Write 1 to enable PAL U6. Write 0 to disable it.
POL	F801	Write 1 for positive pulses, 0 for negative pulses.

The PAL accepts inputs and provides outputs to interrupt the BL1100 or to drive the BL1100 CTC counters. Before starting a pulse-width measurement, initialize the BL1100 KIO CTC3 in counter mode for 255 counts, with interrupts enabled. Direct the CTC3 interrupt vector to a function that increments a count variable. Enable the INT2 interrupt to detect the end of the CTC operation.

Refer to the sample programs in this appendix for details on how to do this. The “Digital Interfaces” section in Chapter 4, “Subsystems,” explains how to program a CTC.

Write 1 to the P0L register to measure positive pulses and 0 to measure negative pulses. After initializing CTC3, INT2 and POL, enable the PAL (write 1 to the PAL register). During the active period of the pulse being measured, the PAL will output counting clocks to TRG3 of CTC3. CTC3 will generate an INT2 interrupt at the end of the active period. The INT2 service routine should disable the PAL and turn off INT2. These events are summarized in Figure K-13.

Pulses can be measured on four separate input channels when the PAL is coupled with the expansion board’s analog multiplexer. The PAL can also be connected to serial pulse train output devices such as Opto 22 analog modules.

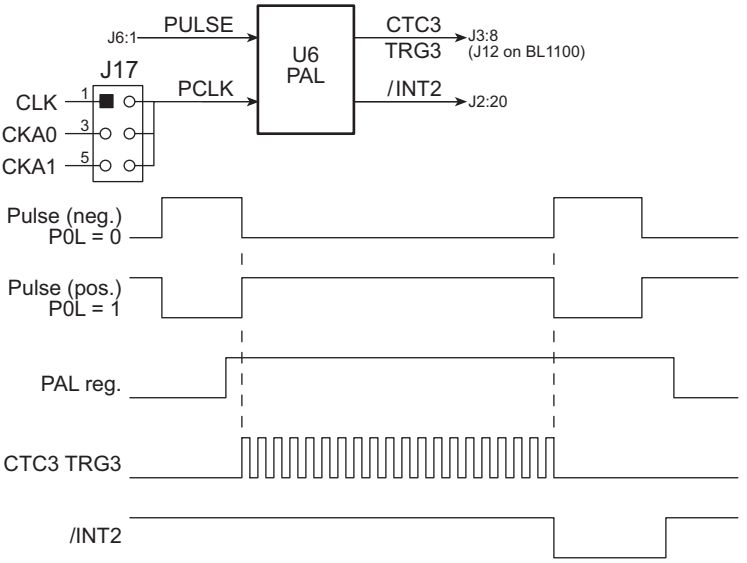


Figure K-13. Pulse Width Measurement

The actual pulse width is calculated from the output of CTC3. When set up as specified above, the counter incremented by the interrupt routine will contain the number of output pulses divided by 255. The output of the counting clock from the PAL to CTC3 is derived from the input clock divided by four. The input clock is selected using header J17 as shown in Figure K-14.

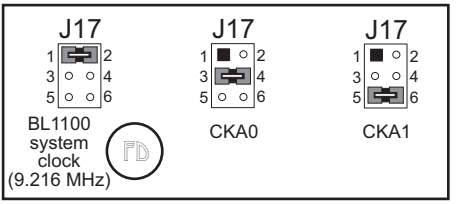


Figure K-14. Input Clock Selection

If the system clock is used as the input clock, the pulse width would be calculated as follows.

$$pw = [count \times 255 + (255 - CTC)] \times 434 \text{ ns} \qquad \text{(K-1)}$$

where *count* is the value obtained from the CTC3 interrupt routine, *CTC* is the contents of the CTC3 register, and 434 ns is four times the duration of one system clock.

Clocks

The clocks from the asynchronous port of the Z180 are based on the baud rate set for the port in its CNTLB register.

CNTLB0 (02_H) and CNTLB1 (03_H)

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

Bit 5 specifies the prescaler value. If this bit is zero, the prescaler equals 10. If bit 5 is set, the prescaler equals 30. The SS bits (0, 1 and 2) set the divider value according to Table 4-7.

The clock output to CTC3 is calculated as follows.

clock output = system clock/(prescaler × divider × 4)

where the system clock is 9.216 MHz.

If the clock rate is changed, remember to change the baud rate of the associated port. Do not change the baud rate for CKA0 if you are using header J7 of the BL1100 as your Dynamic C programming port. This will cause the BL1100 to lose contact with Dynamic C.

Another consideration when using these ports is that CKA0 and CKA1 are multiplexed pins. The CKA0 pin is shared with DREQ0, and CKA1 is shared with TEND0. When the system is reset, these pins are switched to CKA0 and CKA1. CKA0 changes when DMA Channel 0 is programmed for memory to/from I/O (and memory to/from memory mapped I/O). CKA1 will change only if CKA1D of CNTLA0 is set to 1. Memory-to-memory DMA transfers using the function **dmacopy** will not affect CKA0.

An application for the pulse width measurement system is a device that generates pulse trains. One such device is the Opto22 A/D Module (AD11), which converts an analog signal value to a serial pulse train. The frequency range of the output is from 1920 Hz to 9600 Hz.

Other Information

An additional digital signal is available on pin 2 (PIN) of Wago connector J6. This signal can be read by reading the PAL register (x800) for that board and examining the low-order bit:

```
value = inport( PAL ) && 0x01;
```

Only ADC boards support a PAL at U6.

Analog Multiplexer

The MUX and ADC boards have a DG509A 4-channel analog multiplexer. The DG509A is a differential four-channel multiplexer powered at –5 V to +5 V. The DG509A chip can be used as either an input or an output, and provides 4 differential channels or 8 single-ended channels. The address lines are driven by PPI lines PA06 and PA07. (PPI Port A must be configured as an output when using the multiplexer.)

The multiplexer provides multiple I/O channels for many different devices. On the ADC board, the multiplexer supplies bipolar inputs to the 20-bit A/D converter. However, it can also be used to multiplex signals for external devices. The multiplexer input is available on Wago connector J7. The multiplexed signals DA and DB are available for external devices on Wago connector J6. Both DA and DB can be inputs for one (bipolar) or two (unipolar) BL1100 A/D channels. DA can also be connected to PULSE (J6 pin 1) to multiplex pulse width measurements. The DA and DB pins on J6 can be used as direct analog signal inputs to the instrumentation amplifier on the MUX and the ADC boards if the DG509A chip is not installed. The output from the instrumentation amplifier on the ADC board, which provides a fixed gain of 10X (or 100X), can also be multiplexed. The output from the instrumentation amplifier is also available on pin 6 (V/A) of Wago connector J6 when a jumper connects pins 2–3 on header J18. Figure K-15 shows the pinout for the multiplexer input signals on Wago connector J7.

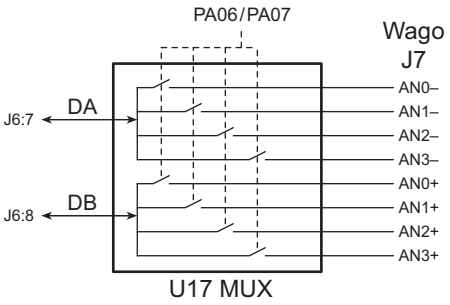
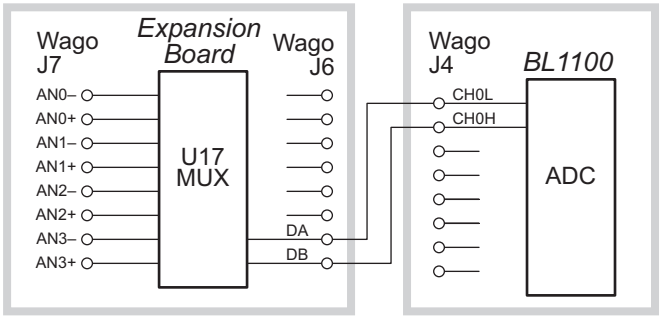


Figure K-15. Multiplexer Input Signals

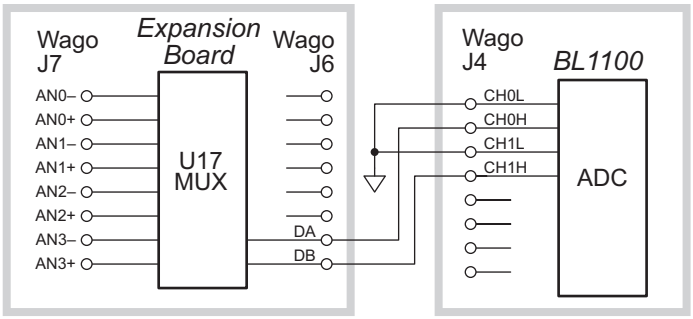
The multiplexer can provide I/O for 8 separate devices. However, the channels on the multiplexer are in pairs. When you switch channels, both lines of the pair are switched.

Figure K-16 shows the connection for a multiplexed bipolar input to the BL1100 ADC Channel 0, and the connections to two BL1100 ADC channels for a unipolar input.

Refer to the “Software” section later in this appendix for the library functions and sample programs showing how to use the multiplexer.



Bipolar Input to BL1100



Unipolar Input to BL1100

Figure K-16. Connecting Multiplexed Bipolar and Unipolar Inputs to BL1100

Instrumentation Amplifier

The MUX and ADC boards include an instrumentation amplifier (LT1101 at U7) that has a high input resistance, high linearity (8 ppm), and low offset voltage drift (0.4 µV/°C). The LT1101 has a pair of differential inputs from the analog multiplexer, and is used with the multiplexer for signal conditioning for the A/D converter or for other off-board devices. A secondary op amp, U8, conditions the amplifier's output.

The LT1101 has a fixed gain of 10 when its pins 1 and 2 and pins 7 and 8 are connected. A fixed gain of 100 is attainable by cutting the printed circuit board traces between pins 1 and 2 and between pins 7 and 8.

Figure K-17 shows the instrumentation amplifier, and where to cut the circuit board trace to realize a gain of 100.

The output from the instrumentation amplifier can be multiplexed. The output is also available on pin 6 (V/A) of header J6 when the jumper on header J18 connects pins 2–3.

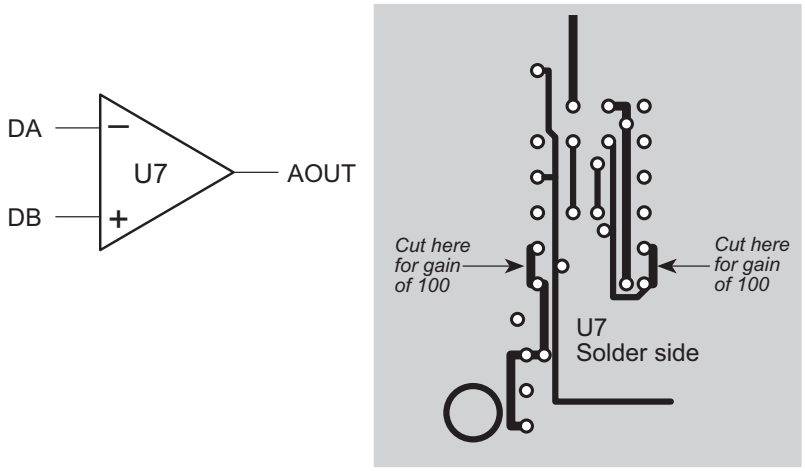


Figure K-17. Instrumentation Amplifier I/O and Gain

A/D Converter

The ADC board includes a high-resolution 20-bit A/D converter (AD7701 at U3) for measuring low-frequency analog signals. The AD7701 has a very high resolution (1 bit = 2.384 µV for a voltage input range of 0 V to 2.5 V) with 0.0003% accuracy, and allows either bipolar or unipolar measurements. There are three internal calibration modes. An independent voltage measurement can be taken every 125 ms. Two-stage signal conditioning uses the instrumentation amplifier as the first stage of amplification, followed by a second op-amp whose gain is adjustable.

Port A of the programmable peripheral interface (PPI) and lines 0 and 1 of the buffered TTL input must be used in conjunction with the A/D converter. These I/O lines are ***not*** available for normal use when the A/D convertre is using them. The other PPI ports and lines 2–7 of the TTL input remain available for your application. Figure K-18 shows these interfaces to the A/D converter.

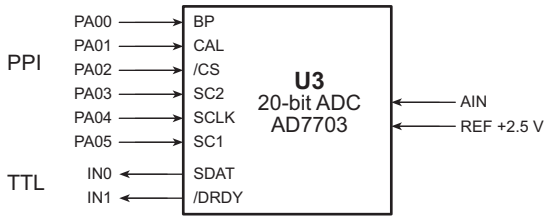


Figure K-18. A/D Convertre Interface with PPI and TTL Buffer

The AD7701 contains a sigma delta conversion A/D converter, a calibration microcontroller with on-chip static RAM, an on-chip digital filter, a clock oscillator, and a serial communications port. An external crystal oscillator of 4.096 MHz is used as the master clock. The sampling rate, filter corner frequency and output word rate are determined by the master clock whose output is read from SDAT (TTL line 0) randomly or periodically at any rate up to 4 kHz. The analog input signal is continuously sampled by an analog modulator whose mean output is proportional to the input signal. The modulator output is processed by an on-chip digital filter, which updates the output data register at a rate up to 4 kHz.

The cutoff frequency for the on-chip digital filter is 10 Hz with the 4.096 MHz master clock. This low-pass filtering limits the valid input signal frequency. A reading is invalid after a step input change until a settling time has elapsed. The worst-case settling time is 125 ms.

The DG509A chip at U17 is a differential four-channel multiplexer that provides an input signal to the instrumentation amplifier (U7) secondary op-amp and to the A/D converter. The long settling time does not allow high-speed multiplexing. After switching to a new channel, wait 125 ms before reading data from the A/D converter.

Figure K-19 shows a schematic of the A/D converter. The gain on the U8 secondary op-amp can be changed with resisters R1, R2, R10 and R11. The factory-configured gain is unity.

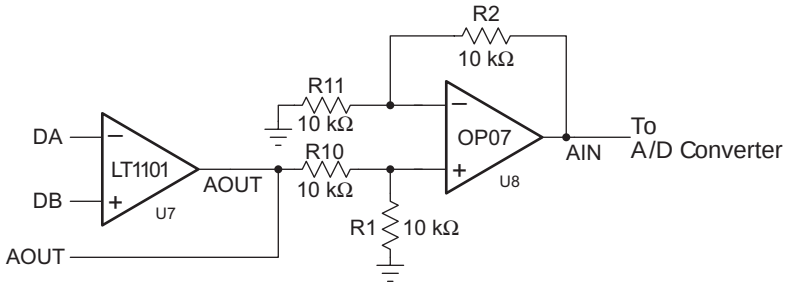


Figure K-19. A/D Converter and Secondary Op-Amp

The reference voltage is +2.5 V from the BL1100's LTC1019. The analog input voltage range is selected with the BP/UP pin connected to PPI PA00. With PA00 low, the input range is unipolar and spans 0 V to +2.5 V. With PA00 high, the input range is bipolar and spans –2.5 V to +2.5 V. Each count represents 2.384 µV in the unipolar mode, and each count represents 4.768 µV in the bipolar mode.

The output coding format looks like this.

Unipolar	Bipolar	Output Data
+2.5 V	+2.5 V	1111 1111 1111 1111 1111 1111
+1.25 V	0.0 V	1000 0000 0000 0000 0000 0000
0.0 V	–2.5 V	0000 0000 0000 0000 0000 0000

The AD7701 offers both self- and system calibration using an on-chip calibration controller and SRAM. The calibration mode pin, CAL, is connected to PA01. When CAL is taken high for more than four cycles of the master clock, the A/D converter is reset and performs a calibration cycle when CAL is brought low again. The calibration control inputs SC1 and SC2 are connected to PA05 and PA03. Table K-3 lists the calibration options.

Table K-3. AD7703 Calibration Options

Calibration Type	SC1	SC2	Zero	Full Scale	Steps
Self calibration	0	0	AGND	VREF	1
System offset	1	1	AIN		1 of 2
System gain	0	1		AIN	2 of 2
System offset	1	0	AIN	VREF	1

In the self-calibration mode, the zero scale is calibrated against the AGND pin and the full scale is calibrated against the VREF pin to remove internal errors. System calibration uses voltage values presented by the system to the AIN pin for the zero and full-scale points to compensate for system gain and offset errors.

There are two system-calibration modes. The two-step system calibration mode calibrates both the system offset and the system gain. The one-step calibration mode uses an input voltage as the system zero-scale calibration point, but uses the VREF value as the full-scale point.

Any drift in the temperature input offset is minimized by chopper stabilization techniques. The drift arising from temperature changes is relatively flat at temperatures from 0 °C to 70°C, with less than 10 LSB. Above 75°C, the offset drift doubles approximately every 10°C.

Gain drift is significantly less than offset drift. The typical gain drift is less than 40 LSB over the specified temperature range.

Using system calibration can minimize offset and gain errors in the signal conditioning circuit, multiplexer, and sensor. Measurement errors attributed to offset drift or gain drift can be eliminated at any time by recalibrating the system.

The digital interface uses the Synchronous External Clock mode (SEC). A falling edge on CS (PA02) enables the serial data output with MSB initially valid at SDAT (IN0) pin. Subsequent data bits change on the falling edge of SCLK (PA04). After the LSB has been transmitted, DRDY (IN1) and SDAT go three-state. If CS is high, DRDY will return high for four clock cycles, then fall as a new word is loaded into the output register.

DGL96

The DGL96 expansion board provides 96 I/O channels, each of which can be used individually as an input or an output. The DGL96 board optionally comes with an op-amp and power buffer that boosts the BL1100's D/A converter output from 2 mA to 150 mA. Since up to four boards can be stacked on a BL1100, there can be up to 384 I/O lines in a system based on the BL1100.

The DGL96 board has six Zilog PIOs, which provide the 96 inputs and outputs. Each PIO has two 8-bit ports, A and B. Each port has a data register and a control register. Thus, each DGL96 board has 24 addressable I/O registers, grouped into six sets of four as shown below.

PIODA 0–5	Data Register A
PIODB 0–5	Data Register B
PIOCA 0–5	Control Register A
PIOCB 0–5	Control Register B

Chapter 4, "Subsystems," provides further details on Zilog PIOs. The "Software" section later in this appendix explains how to program the DGL96 board using the Dynamic C **`96IO.LIB`** library.

Software

Software for programming the BL1100 expansion boards is available in the Dynamic C **IOEXPAND.LIB** and **96IO.LIB** libraries. The **IOEXPAND.LIB** library is for the DGL, MUX, and ADC boards, and the **96IO.LIB** library is for the DGL96 board.

A program running on the BL1100 communicates with an expansion board via the board's registers. Each register on each board is individually addressable.

IOEXPAND.LIB

The **IOEXPAND.LIB** library has two parts. One part is for a BL1100 system with only one expansion board (DGL, MUX, or ADC). If the system has only one such expansion board, its register addresses should be Fxxx (all jumpers removed from header J10). Instead of specifying a node number (0–3), specify –1. This will load the correct default addresses.

The other section of this library is for systems with more than one of these three boards. These systems run a bit slower and refer to a structure that holds the "current" or "default" register set. Multiple boards are indexed 0–3, corresponding to addresses Cxxx, Dxxx, Exxx, and Fxxx. The current register set is referenced by the current board index (or node).

Table K-4 explains the functions.

Table K-4. IOEXPAND.LIB Functions

Function		Description
One board	Multiple Boards	
exp_init	**exp_init_n**	Initializes an expansion board
mux_ch	**mux_ch_n**	Selects a multiplexer channel.
ad20_mux	**ad20_mux_n**	Selects MUX channel and A/D conversion mode
ad20_rdy	**ad20_rdy_n**	Test-ready state of A/D converter
ad20_cal	**ad20_cal_n**	Performs A/D converter calibration
ad20_rd	**ad20_rd_n**	Reads the A/D converter
—	**get_na**	Gets node addresses, given index
—	**get_def_na**	Gets current addresses and index
—	**set_def_na**	Sets current addresses and index

To use an analog multiplexer or an ADC, first initialize PPI Port A of the board to be an output. Use the **exp_init** or **exp_init_n** function.

- **int exp_init(int a, int b, int c1, int c2)**

  Initializes the programmable peripheral interface (PPI) for Mode 0. This function affects only I/O expansion boards with the default base address of Fxxx. The function sets up the correct control word for the configuration of the ports and immediately sets all output lines to the value of the given port parameter.

  PARAMETERS: **a**, **b**, **c1**, and **c2** are output values for the PPI output register. Pass **a** = –1 to configure Port A as an input. Pass **b** = –1 to configure Port B as an input. Pass **c1** = –1 to configures Port C upper nibble as inputs. Pass **c2** = –1 to configures Port C lower nibble as inputs. All PPI output ports are reset to low when the mode is changed.

  Output a correct value to the output port right after the mode is changed.

  Use any other value to specify a port as output. The parameter's value is written to the port after the mode is changed.

  After initializing the ports, they can be read or written with the **inport** and **outport** functions. Each bit position represents one line of the port, with a 1 indicating that the level is high and a 0 indicating that the level is low.

  The sample program **IOE_DGL.C** shows how the initialize and read PPI ports.

- **int exp_init_n(int node, int a, int b, int c1, int c2, int def)**

  Initializes the expansion board specified by **node**. The function sets up the correct control word for the configuration of the ports and immediately sets all output lines to the values of the given parameters.

  PARAMETERS: **node** is the base address for the expansion board to initialize. Use the values 0–3 for specific base addresses. Use –1 for the default node address (see **def**).

  a, **b**, **c1**, **c2** are port values that specify whether the port is an input or an output. Use –1 to specify a port as input and any other value to specify a port as output as in **exp_init**. The parameter's value (lower eight bits) is written to the port after the port's mode is changed.

  def specifies that the specified **node** is to be the new default when **def** is non-zero..

- **`int mux_ch( int chan )`**

  Sets the DG509A multiplexer channel with a default address of 0xFxxx.

  PARAMETER: **`chan`** is a value from 0 to 3.

  RETURN VALUE: 0 if successful, and –1 if an invalid channel number is specified.

- **`int mux_ch_n( int node, int chan, int def )`**

  Sets the DG509A multiplexer channel on a specified node. This function is for systems with multiple expansion boards.

  PARAMETERS: **`node`** is 0 to 3 for address Cxxx to Fxxx respectively, or –1 to use the default address saved in **`def_na`**.

  `chan` is a value from 0 to 3.

  `def` specifies whether to make the specified node the default: 1 saves **`node`** as the default node, 0 does not.

  RETURN VALUE: 0 if successful, and –1 if an invalid channel number or invalid node number is specified.

- **`void ad20_mux( int chan )`**

  Sets the DG509A multiplexer for the 20-bit AD7703 of the expansion board with a default address of Fxxx.

  PARAMETER: **`chan`** is a value from 0 to 7 specifying the channel and unipolar or bipolar operation as follows.

chan	Operation	MUX Channel	Input Voltage (V)
0	unipolar	0	0 to +2.5
1	unipolar	1	0 to +2.5
2	unipolar	2	0 to +2.5
3	unipolar	3	0 to +2.5
4	bipolar	0	–2.5 to +2.5
5	bipolar	1	–2.5 to +2.5
6	bipolar	2	–2.5 to +2.5
7	bipolar	3	–2.5 to +2.5

- **`int ad20_mux_n( int node, int chan, int def )`**

Sets the DG509A multiplexer for the 20-bit AD7703 of the expansion board in a system with multiple expansion boards.

PARAMETERS: **node** is 0 to 3 for address Cxxx to Fxxx respectively, or –1 to use the default address saved in **def_na**.

chan is a value from 0 to 7 specifying the channel and unipolar or bipolar operation as described for **ad20_mux**.

def specifies whether to make the specified node the default: 1 saves **node** as the default node, 0 does not.

RETURN VALUE: 0 if successful and –1 if an invalid parameter is passed.

- **int ad20_rdy()**

  Tests AD7703 /DRDY status from RDTTL bit 1.

  RETURN VALUE: 1 if the A/D convertre is ready, and 0 if not.

- **int ad20_rdy_n(int node)**

  Tests AD7703 /DRDY status from RDTTL bit 1 of a specified node.

  PARAMETER: **node** is 0 to 3 for address Cxxx to Fxxx respectively, or –1 to use the default address saved in **def_na**.

  RETURN VALUE: 0 if the A/D converter on the specified board is ready, and –1 if the A/D convertre is busy or if an invalid node is passed.

- **int ad20_cal(int mode)**

  Calibrates the A/D converter of the expansion board with a default address of Fxxx. The function waits for the A/D converter to become ready before returning.

  PARAMETER: **mode** can be 0, 1, or 2, as explained below.

  Mode 0, self-calibration as described in Table K-3, does not use the multiplexer.

  Mode 1 calibration uses the multiplexer to get zero-scale and full-scale readings on AIN. MUX ch0 is the A/D signal to be measured. MUX ch1 is AIN for the first step, which calibrates the system offset. MUX ch2 is AIN for the second step, which calibrates the system gain.

  Mode 2 calibration uses the current channel to get AIN as the zero-scale reading to calibrate the system offset.

  RETURN VALUE: 0 if successful, and –1 if an error occured.

- **int ad20_cal_n(int node, int mode, int def)**

Calibrates the AD7703 A/D converter on the specified board.

PARAMETER: **node** is 0 to 3 for address Cxxx to Fxxx respectively, or –1 to use the default address saved in **def_na**.

mode can be 0, 1, or 2, as explained below.

Mode 0, self-calibration as described in Table K-3, does not use the multiplexer.

Mode 1 calibration uses the multiplexer to get zero-scale and full-scale readings on AIN. MUX ch0 is the A/D signal to be measured. MUX ch1 is AIN for the first step, which calibrates the system offset. MUX ch2 is AIN for the second step, which calibrates the system gain.

Mode 2 calibration uses the current channel to get AIN as the zero-scale reading to calibrate the system offset.

def specifies whether to make the specified node the default: 1 saves **node** as the default node, 0 does not.

RETURN VALUE: 0 if successful, and –1 if an error occured.

- **long ad20_rd()**

Reads 20-bit data from the AD7703 serial data port. The function waits for the A/D converter to become ready before reading. It does this by sampling TTL line 0 (which represents ADC SDAT) 20 times.

RETURN VALUE: 20-bit data. Unipolar mode: 00000 = AGND, 7FFFF = 1.25 V, and FFFFF = 2.5 V; bipolar mode: 00000 = –2.5 V, 7FFFF = AGND, and FFFFF = 2.5 V.

Because of the 125 ms step response time of the AD7703, a time delay should be guaranteed after a multiplexer switching.

- **long ad20_rd_n(int node, int def)**

Reads 20-bit data from the AD7703 serial data port. The function waits for the A/D converter to become ready before reading.

PARAMETER: **node** is 0 to 3 for address Cxxx to Fxxx respectively, or –1 to use the default address saved in **def_na**.

def specifies whether to make the specified node the default: 1 saves **node** as the default node, 0 does not.

RETURN VALUE: 20-bit data. Unipolar mode: 00000 = AGND, 7FFFF = 1.25 V, and FFFFF = 2.5 V; bipolar mode: 00000 = –2.5 V, 7FFFF = AGND, and FFFFF = 2.5 V. Returns –1 for an invalid node.

Because of the 125 ms step response time of the AD7703, a time delay should be guaranteed after a multiplexer switching.

- **`int get_na( int node, struct node_addr *na )`**

  Gets the corresponding node address from the specified node. Loads the structure to which **`na`** points with the correct addresses for the node specified.

  PARAMETER: **`node`** is 0 to 3 for address Cxxx to Fxxx respectively.

  RETURN VALUE: 0 if the node is proper, –1 if the node is out of range.

- **`int set_def_na( int node )`**

  Sets the default node address.

  RETURN VALUE: Data from **`get_na`**, –1 if an invalid node number is specified.

- **`int get_def_na( struct node_addr *na )`**

  Loads the default node address into the structure to which **`na`** points.

  RETURN VALUE: default node number.

96IO.LIB

This library includes several parameters, a number of functions to control, read and write from/to the PIO ports, and an error-trapping facility.

- **`void Init_Board_0()`**

  Sets all bits on board address 0 to inputs.

- **`void Set_Port_Dir( int board, int pio_num, int port, char mask )`**

  Sets the bit directions for the specified port of the specified PIO on the specified board.

  PARAMETERS: **`board`** specifies which board to address: 0—8x00, 1—9x00, 2—Ax00, 3—Bx00.

  `pio_num` specifies PIO 0–5.

  `port` specifies the PIO port: 0—Port A, 1—Port B.

  `mask` is the bit mask for the port: Bit Set = input bit (1), Bit Clear = output bit (0s).

- **`void Set_Bit_Dir( int board, int pio_num, int port, int bit_num, int dir )`**

Sets the direction of an individual bit of the specified PIO port on the specified board.

PARAMETERS: **board** specifies which board to address: 0—8x00, 1—9x00, 2—Ax00, 3—Bx00.

pio_num specifies PIO 0–5.

port specifies the PIO port of a PIO: 0—Port A, 1—Port B.

bit_num specifies the bit in the port (0–7).

dir may be input (1) or output (0). Any other value for **dir** produces an error.

- **int Port_Addr(int board, int pio_num, int register)**

  Returns the address of a PIO register on the specified board. An error results if any of the parameters is out of range.

  PARAMETERS: **board** specifies which board to address: 0—8x00, 1—9x00, 2—Ax00, 3—Bx00.

  pio_num specifies PIO 0–5.

  register must be **DA** (0), **DB** (1), **CA** (2), or **CB** (3).

  RETURN VALUE: Address of PIO register

- **void Set_PIO_Board(int board, int dir)**

  Sets all PIOs on the specified board to "Mode 3": all input or all output.

  PARAMETERS: **board** specifies which board to address: 0—8x00, 1—9x00, 2—Ax00, 3—Bx00.

  dir may be input (1) or output (0). Any other value for **dir** produces an error.

- **void Write_Port(int board, int pio_num, int port, byte value)**

  Writes **value** to the specified port of the specified PIO on the specified board.

  PARAMETERS: **board** specifies which board to address: 0—8x00, 1—9x00, 2—Ax00, 3—Bx00.

  pio_num specifies PIO 0–5.

  port specifies the PIO port of a PIO: 0—Port A, 1—Port B.

- **char Read_Port(int board, int pio_num, int port, int mode)**

- **`char Read_Port( int board, int pio_num, int port, int mode )`**

  Reads the value of the specified port of the specified PIO on the specified board. The function reads only the input bits of the port.

  PARAMETERS: **`board`** specifies which board to address: 0—8x00, 1—9x00, 2—Ax00, 3—Bx00.

  `pio_num` specifies PIO 0–5.

  `port` specifies the PIO port of a PIO: 0—Port A, 1—Port B.

  `mode` determines the output bits: 1 if **`mode`** = **`FILL_MODE`**, 0 if **`mode`** = **`STRIP_MODE`**.

  RETURN VALUE: The value read. The output bits of the returned value are set to 1 if **`mode`** is **`FILL_MODE`**, and are set to 0 of **`mode`** is **`STRIP_MODE`**. An error results if **`port`** or **`mode`** is not as specified.

Sample Programs

Pulse Width Measurement

The following sample program program, **`PWM.C`** from the Dynamic C **`SAMPLES\IOE`** directory, illustrates how to set up and perform a pulse width measurement.

PWM.C

```
/*
Use PAL FP04100.PDS for pulse width measurement.
Pulse input is at Wago J6, line 1.
Counting clock (from J17) is CKA0. PAL FP04100 divides
counting clock by 4. For system clock of 9.216MHz, each
count is 0.434 µs. Count clock is sent to CTC3 TRG3.
POL = 0: measure Negative Pulse. PAL = 0: disabled
POL = 1: measure Positive Pulse. PAL = 1: enabled
Writing 0xC5 (1100 0101) to CTC3 has these effects:
              ^^    ^ ^
              ||    | |- this is a control word
              ||    |- time constant follows (it's 255)
              ||- counter mode, not timer mode
              |- enable interrupt
*/

int  i, l, m;             // temporaries
unsigned int n;           // pulse width
char flag;                // set when pulse ends
char count;               // count full CTC cycles
```

continued…

```
main(){
   outport(CTC0,CTC3_VEC);  // set interrupt vector
   outport(PAL,0);          // disable PAL
   outport(POL,0);          // negative pulses
   while(1){
      count = 0;
      flag = 0;
      outport(CTC3,0x03);   // software reset
      outport(CTC3,0xC5);   // see note above
      outport(CTC3,0xFF);   // 255
      ISET(ITC,2);          // enable INT2
      outport(PAL,1);       // enable PAL
      while(!flag){}
      m = inport(CTC3);
      width = count * 255 + (255 - m);
      ++l;
      printf("Count = %d. Width = %u %d\n", count, width, l);
      for(i=0; i<10000; ++i){}      // wait
   }
}
#INT_VEC CTC3_VEC ctc3int
#asm nodebug
ctc3int::                // increments count
   push af
   ld   a,(count)
   inc  a
   ld   (count),a
   pop  af
   ei
   reti
#endasm
#INT_VEC INT2_VEC int2int
interrupt ret int2int(){
   outport(PAL,0);       // disable PAL
   IRES(ITC,2);          // turn off INT2
   flag = 1;
}
```

Table K-5 lists other sample programs in the Dynamic C **SAMPLES\IOE** directory that illustrate the use of the BL1100 expansion boards.

Table K-5. Sample Programs for BL1100 Expansion Boards

Sample Program	Description
IOE_AD.C **IOE_AD_N.C**	Measures voltage difference from set value with ADC expansion board
IOE_DGL.C **IOEDGL_N.C**	Illustrates use of DGL I/O expansion board
IOE_DGL96.C	Illustrates use of DGL96 I/O expansion board
IOE_MUX.C **IOEMUX_N.C**	Illustrates use of 4-channel multiplexer on ADC and MUX expansion boards
IOE_OPTO.C	Determines number of samples required to detect AC signals when using optional H11AA1 optical isolation chip

PWM PAL Equations

The following PAL equations are for the pulse width measurement system.

```
Title LGEXP1 IAB202 expansion card for BL1100 SBC200E
DESIGN
Pattern FP04100.pds
Revision A
Author Shaoqi Tang
Company Z-WORLD, DAVIS, CA
CHIP IAB202_U6 PALCE16V8

;PINS 1    2    3   4   5   6  7   8  9  10
      PCLK /PAL /WR /RD PP /NP TO2 A1 A0 GND

;     11  12 13 14 15  16   17    18   19   20
      /OE D0 EN P0 RUN HCLK /INT2 TRG3 TRG2 VCC

SIGNATURE = 4100

EQUATIONS

D0 = PAL*/A1*/A0*RD*/WR*/NP
   + PAL*/A1*A0*RD*/WR*PP
D0.TRST = PAL*/A1*RD*/WR     ;D0 OUTPUT
                             ;READ NP AT 08000H
                             ;READ PP AT 08001H
P0 := EN*PP*/RUN*/TRG2
    + EN*/PP*/RUN*TRG2
    + EN*P0

EN = PAL*WR*/RD*/A1*/A0*D0
   + /PAL*EN
   + /WR*EN
   + RD*EN
   + A1*EN
   + A0*EN                   ;CPU wr EN

TRG2 = PAL*WR*/RD*/A1*A0*D0
     + /PAL*TRG2
     + /WR*TRG2
     + RD*TRG2
     + A1*TRG2
     + /A0*TRG2              ;CPU wr TRG2=POLARITY

RUN := EN*P0*/PP*/TRG2
     + EN*P0*PP*TRG2
     + EN*RUN

INT2 := EN*RUN*PP*/TRG2
      + EN*RUN*/PP*TRG2
      + EN*INT2

TRG3 := TRG3*HCLK*RUN*/INT2*EN
      + /TRG3*/HCLK*RUN*/INT2*EN   ;1/4 PCLK

HCLK := /HCLK
```

Board Layouts

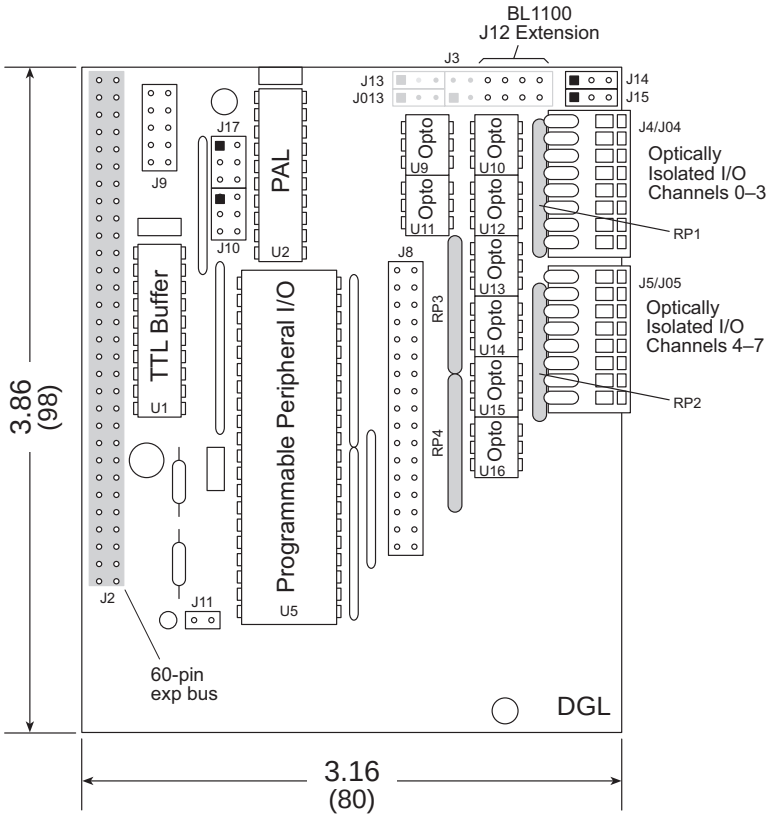


Figure K-20. DGL Expansion Board Layout (Digital I/O)

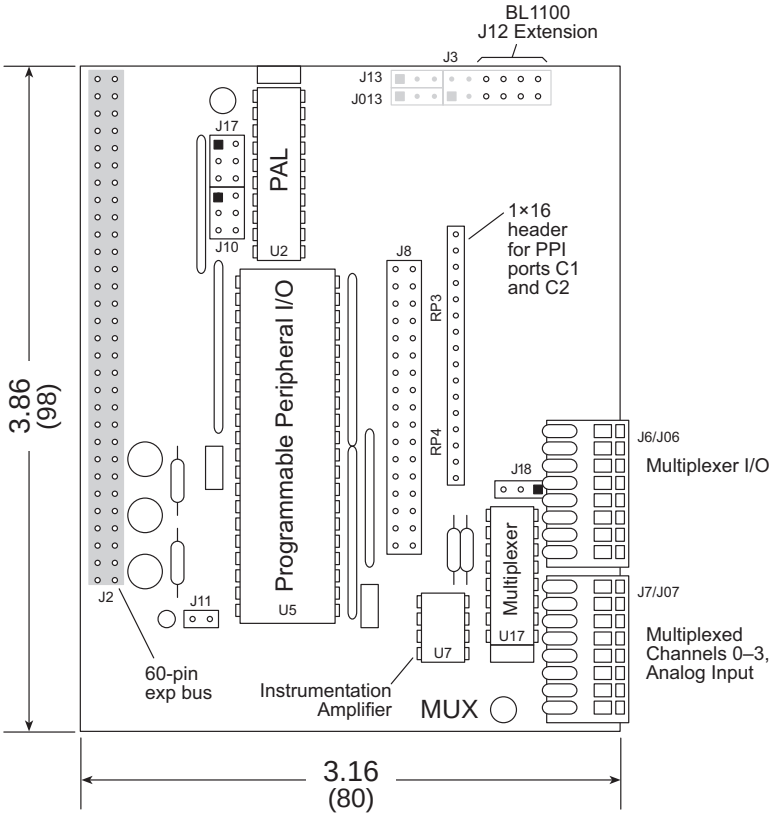


Figure K-21. MUX Expansion Board Layout (Multiplexer)

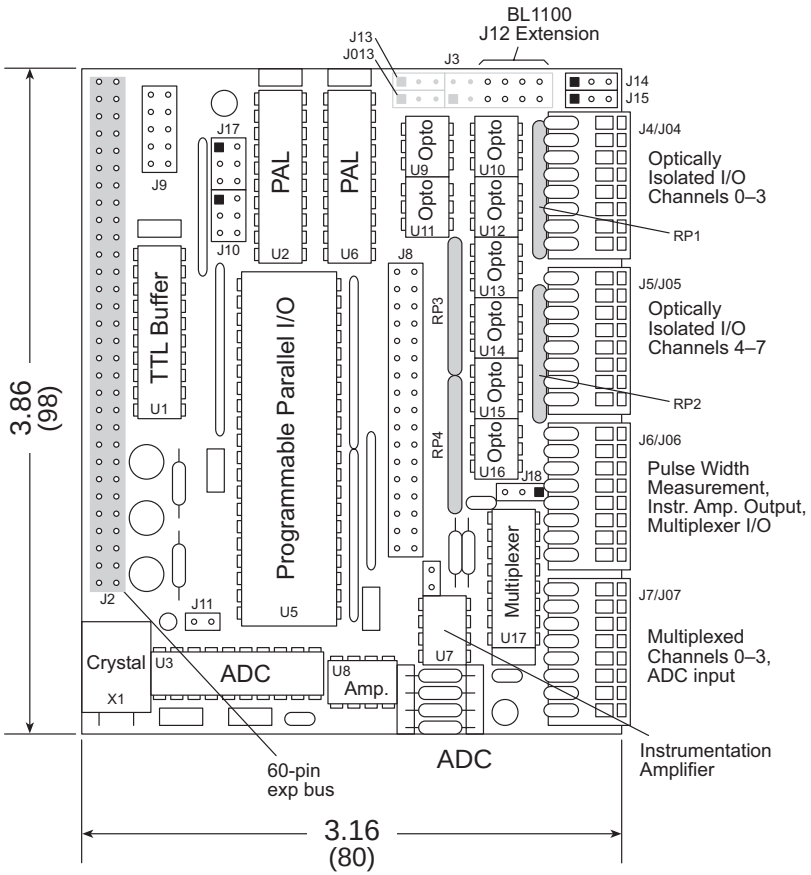
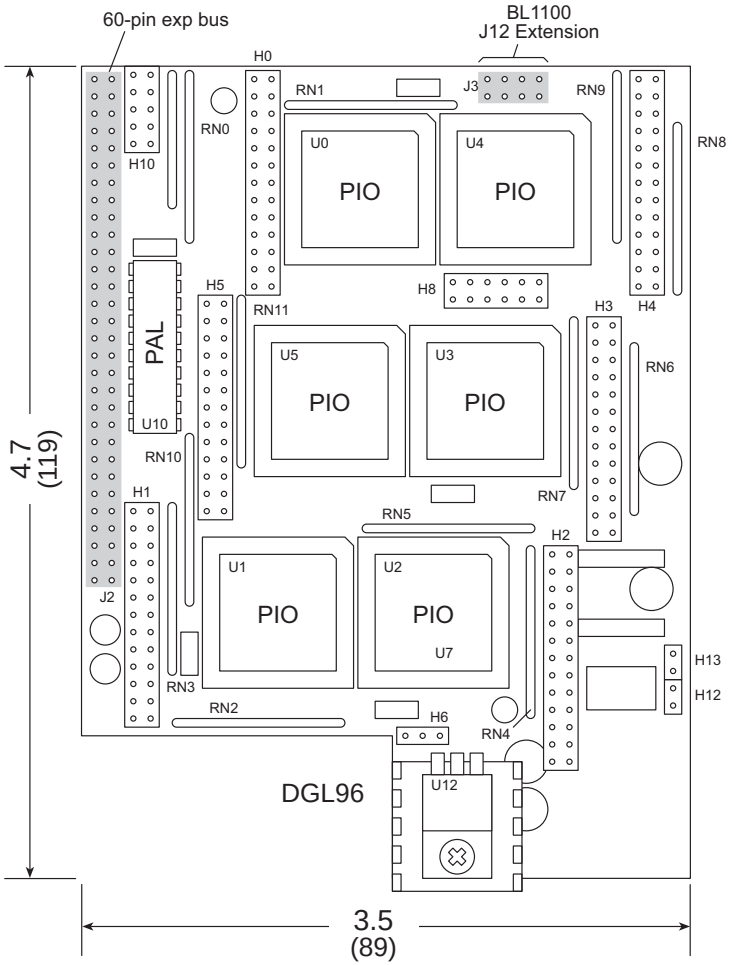
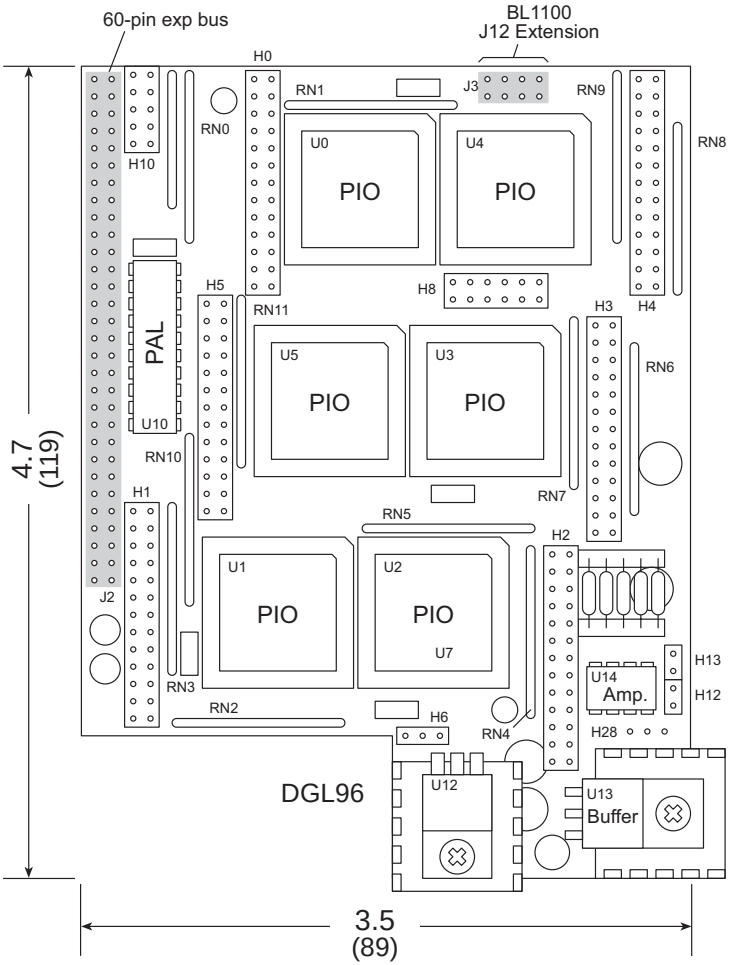


Figure K-22. ADC Expansion Board Layout (A/D Conversion)



(a) Standard Board

Figure K-23. DGL96 Expansion Board Layout (Digital I/O)



(b) Board With Optional D/A Converter Power Amp

Figure K-23. DGL96 Expansion Board Layout (Digital I/O)

I/O Map

The different registers in the I/O space can be accessed in Dynamic C using the register addresses in Tables K-6 and K-7. The library functions **inport** and **outport** access the I/O registers directly.

```
data_value = inport (PAL_C);
outport (PAL_C, data_value);
```

The library functions **IBIT**, **ISET**, and **IRES** allow bits in the I/O registers to be tested, set, and cleared respectively..

Both 16-bit and 8-bit I/O addresses can be used. Up to four I/O expansion boards can be stacked.

Table K-6 lists the register addresses for the DGL, MUX, and ADC expansion boards. The default base address is Fxxx, so the symbolic names do not include the base address specifier.

Table K-7 lists the register addresses for the DGL96 expansion board. Each PIO on each board has four registers.

Table K-6. DGL, MUX, and ADC Expansion Board Registers

Address	Name	Description
C400	RDTTL_C	Buffered TTL-level input
C800	PAL_C	Pulse width measurement PAL
C801	POL_C	Polarity for PWM
CC00	PPIA_C	Port A, PPI
CC01	PPIB_C	Port B, PPI
CC02	PPIC_C	Port C, PPI
CC03	PPICN_C	PPI control register
D400	RDTTL_D	Buffered TTL-level input
D800	PAL_D	Pulse width measurement PAL
D801	POL_D	Polarity for PWM
DC00	PPIA_D	Port A, PPI
DC01	PPIB_D	Port B, PPI
DC02	PPIC_D	Port C, PPI
DC03	PPICN_D	PPI control register
E400	RDTTL_E	Buffered TTL-level input
E800	PAL_E	Pulse width measurement PAL
E801	POL_E	Polarity for PWM
EC00	PPIA_E	Port A, PPI
EC01	PPIB_E	Port B, PPI
EC02	PPIC_E	Port C, PPI
EC03	PPICN_E	PPI control register
F400	RDTTL	Buffered TTL-level input
F800	PAL	Pulse width measurement PAL
F801	POL	Polarity for PWM
FC00	PPIA	Port A, PPI
FC01	PPIB	Port B, PPI
FC02	PPIC	Port C, PPI
FC03	PPICN	PPI control register

Table K-7. DGL96 Expansion Board Registers

Address	Description	Address	Description
8000	PIO 0, data, Port A	8002	PIO 0, control, Port A
8001	PIO 0, data, Port B	8003	PIO 0, control, Port B
8200	PIO 1, data, Port A	8202	PIO 1, control, Port A
8201	PIO 1, data, Port B	8203	PIO 1, control, Port B
8400	PIO 2, data, Port A	8402	PIO 2, control, Port A
8401	PIO 2, data, Port B	8403	PIO 2, control, Port B
8600	PIO 3, data, Port A	8602	PIO 3, control, Port A
8601	PIO 3, data, Port B	8603	PIO 3, control, Port B
8800	PIO 4, data, Port A	8802	PIO 4, control, Port A
8801	PIO 4, data, Port B	8803	PIO 4, control, Port B
8A00	PIO 5, data, Port A	8A02	PIO 5, control, Port A
8A01	PIO 5, data, Port B	8A03	PIO 5, control, Port B
9000	PIO 0, data, Port A	9002	PIO 0, control, Port A
9001	PIO 0, data, Port B	9003	PIO 0, control, Port B
9200	PIO 1, data, Port A	9202	PIO 1, control, Port A
9201	PIO 1, data, Port B	9203	PIO 1, control, Port B
9400	PIO 2, data, Port A	9402	PIO 2, control, Port A
9401	PIO 2, data, Port B	9403	PIO 2, control, Port B
9600	PIO 3, data, Port A	9602	PIO 3, control, Port A
9601	PIO 3, data, Port B	9603	PIO 3, control, Port B
9800	PIO 4, data, Port A	9802	PIO 4, control, Port A
9801	PIO 4, data, Port B	9803	PIO 4, control, Port B
9A00	PIO 5, data, Port A	9A02	PIO 5, control, Port A
9A01	PIO 5, data, Port B	9A03	PIO 5, control, Port B
A000	PIO 0, data, Port A	A002	PIO 0, control, Port A
A001	PIO 0, data, Port B	A003	PIO 0, control, Port B
A200	PIO 1, data, Port A	A202	PIO 1, control, Port A
A201	PIO 1, data, Port B	A203	PIO 1, control, Port B
A400	PIO 2, data, Port A	A402	PIO 2, control, Port A
A401	PIO 2, data, Port B	A403	PIO 2, control, Port B

continued…

Table K-7. DGL96 Expansion Board Registers (concluded)

Address	Description	Address	Description
A600	PIO 3, data, Port A	A602	PIO 3, control, Port A
A601	PIO 3, data, Port B	A603	PIO 3, control, Port B
A800	PIO 4, data, Port A	A802	PIO 4, control, Port A
A801	PIO 4, data, Port B	A803	PIO 4, control, Port B
AA00	PIO 5, data, Port A	AA02	PIO 5, control, Port A
AA01	PIO 5, data, Port B	AA03	PIO 5, control, Port B
B000	PIO 0, data, Port A	B002	PIO 0, control, Port A
B001	PIO 0, data, Port B	B003	PIO 0, control, Port B
B200	PIO 1, data, Port A	B202	PIO 1, control, Port A
B201	PIO 1, data, Port B	B203	PIO 1, control, Port B
B400	PIO 2, data, Port A	B402	PIO 2, control, Port A
B401	PIO 2, data, Port B	B403	PIO 2, control, Port B
B600	PIO 3, data, Port A	B602	PIO 3, control, Port A
B601	PIO 3, data, Port B	B603	PIO 3, control, Port B
B800	PIO 4, data, Port A	B802	PIO 4, control, Port A
B801	PIO 4, data, Port B	B803	PIO 4, control, Port B
BA00	PIO 5, data, Port A	BA02	PIO 5, control, Port A
BA01	PIO 5, data, Port B	BA03	PIO 5, control, Port B

Jumper and Header Specifications

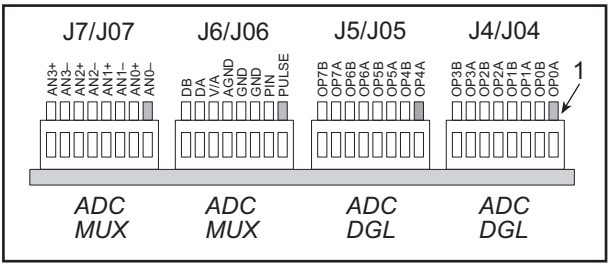
Table K-8 describes all the BL1100 expansion board headers that carry signals.

Table K-8. BL1100 Expansion Board Headers

Header	Description	Expansion Board			
		DGL96	ADC	DGL	MUX
H0–H5	PIO Signals	✓			
H28	Power amp connection to BL1100 DAC at BL1100 header J28	✓			
J2	60-pin connector to BL1100 header J24	✓	✓	✓	✓
J3	8-pin connector to BL1100 header J12	✓	✓	✓	✓
J4/J04	Optically isolated I/O		✓	✓	
J5/J05	Optically isolated I/O		✓	✓	
J6/J06	PWM and multiplexer signals		✓		✓
J7/J07	Multiplexed analog channels 0–3		✓		✓
J8	PPI Ports A and B		✓		✓
J9	TTL Input		✓	✓	

ADC, DGL, and MUX Expansion Board Signals

Figure K-24 shows the pinouts for Wago connectors J4/J04–J7/J07.



NOTE

Do not mix analog grounds (J6/J06:5) with digital grounds (J6/J06:3-4).

Figure K-24. Pinouts for Wago Connectors

Figure K-25 shows the pinouts for headers J8 and J9.

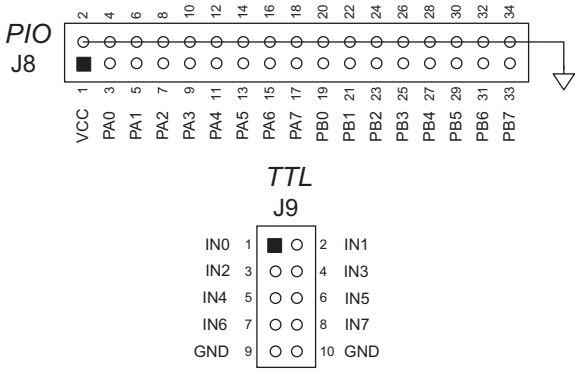


Figure K-25. PIO Parallel Port and TTL Input Pinouts

DGL96 Expansion Board Signals

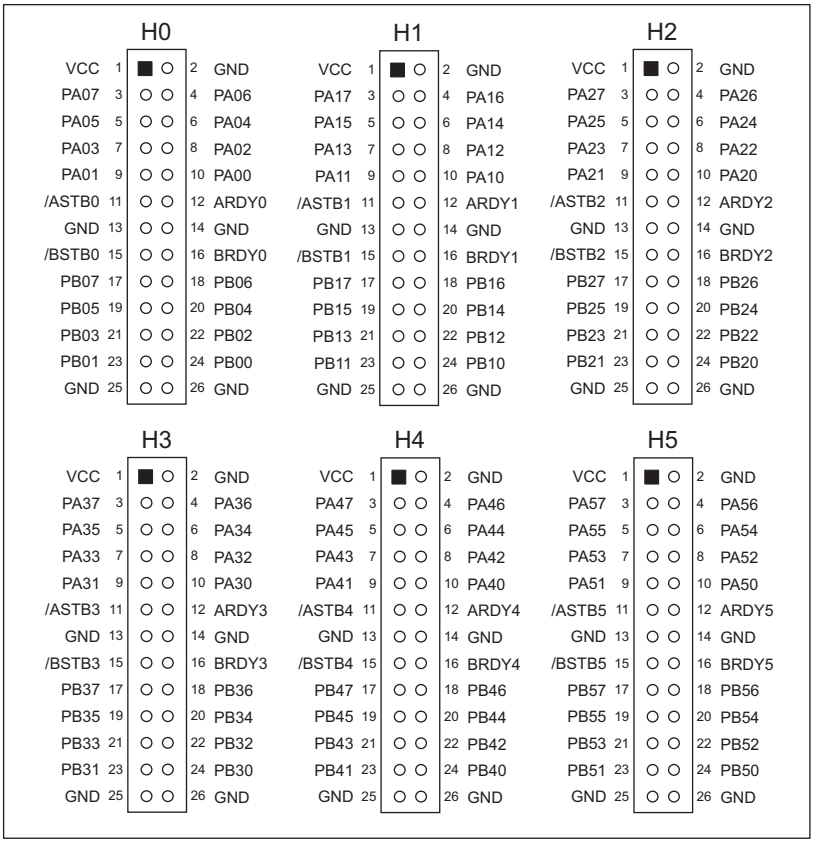


Figure K-26. DGL96 PIO Parallel Port Pinouts

Jumper Configurations

Table K-9 lists the jumper settings for the applicable BL1100 expansion board headers.

Table K-9. Standard BL1100 Expansion Board Jumper Settings

Header	Pins	Description	Factory Default
ADC, DGL, MUX Expansion Boards			
J10	1–2 3–4	Board address Cxxx	
	3–4	Board address Dxxx	
	1–2	Board address Exxx	
	—	Board address Fxxx	FD
J11	1–2	Connect to connect analog ground (AGND) to digital ground (GND)	Connected
J14	1–2	Optoisolator channels 0–3 are outputs	
	2–3	Optoisolator channels 0–3 are inputs	Connected
J15	1–2	Optoisolator channels 4–7 are outputs	Connected
	2–3	Optoisolator channels 4–7 are inputs	
J17	1–2	PWM based on system clock	Connected
	3–4	PWM based on CKA0 from Z180 ASCI	
	5–6	PWM based on CKA1 from Z180 ASCI	
J18	1–2	VCCQ to pin 6 of Wago connector J6/J06	Connected
	2–3	AOUT to pin 6 of Wago connector J6/J06	
DGL96 Expansion Boards			
H10	1–2 3–4	Board address 8xxx	
	3–4	Board address 9xxx	
	1–2	Board address Axxx	
	—	Board address Bxxx	FD

Header H10 on the DGL96 board also carries interrupt and ready signals from the PIOs.

APPENDIX L: BACKUP BATTERY

Battery Life and Storage Conditions

The battery on the BL1100 will provide at least 9,000 hours of backup time for the onboard real-time clock and SRAM. However, backup time longevity is affected by many factors, including the amount of time the controller is unpowered and the SRAM size. Most systems are operated on a continuous basis, with the battery supplying power to the real-time clock and the SRAM during power outages and/or during routine maintenance. The time estimate reflects the shelf life of a lithium ion battery with occasional use rather than the ability of the battery to power the circuitry full time.

The battery has a capacity of 165 mA·h. At 25°C, the real-time clock draws 3 µA when idle, and the 32K SRAM draws 2 µA. If the BL1100 were unpowered 100 percent of the time, the battery would last 33,000 hours (3.8 years).

To maximize the battery life, the BL1100 should be stored at room temperature in the factory packaging until field installation. Take care that the BL1600 is not exposed to extreme temperature, humidity, and/or contaminants such as dust and chemicals.

To ensure maximum battery shelf life, follow proper storage procedures. Replacement batteries should be kept sealed in the factory packaging at room temperature until installation. Protection against environmental extremes will help maximize battery life.

Replacing Soldered Lithium Battery

Use the following steps to replace the battery.

1. Locate the three pins on the bottom side of the printed circuit board that secure the battery to the board.
2. Carefully de-solder the pins and remove the battery. Use a solder sucker to clean up the holes.
3. Install the new battery and solder it to the board. Use only a Panasonic BR2325-1HG or its equivalent.

Battery Cautions

- Caution **(English)**

  There is a danger of explosion if battery is incorrectly replaced. Replace only with the same or equivalent type recommended by the manufacturer. Dispose of used batteries according to the manufacturer's instructions.

- Warnung **(German)**

  Explosionsgefahr durch falsches Einsetzen oder Behandein der Batterie. Nur durch gleichen Typ oder vom Hersteller empfohlenen Ersatztyp ersetzen. Entsorgung der gebrauchten Batterien gemäb den Anweisungen des Herstellers.

- Attention **(French)**

  Il y a danger d'explosion si la remplacement de la batterie est incorrect. Remplacez uniquement avec une batterie du même type ou d'un type équivalent recommandé par le fabricant. Mettez au rebut les batteries usagées conformément aux instructions du fabricant.

- Cuidado **(Spanish)**

  Peligro de explosión si la pila es instalada incorrectamente. Reemplace solamente con una similar o de tipo equivalente a la que el fabricante recomienda. Deshagase de las pilas usadas de acuerdo con las instrucciones del fabricante.

- Waarschuwing **(Dutch)**

  Explosiegevaar indien de batterij niet goed wordt vervagen. Vervanging alleen door een zelfde of equivalent type als aanbevolen door de fabrikant. Gebruikte batterijen afvoeren als door de fabrikant wordt aangegeven.

- Varning **(Swedish)**

  Explosionsfära vid felaktigt batteribyte. Använd samma batterityp eller en likvärdigt typ som rekommenderas av fabrikanten. Kassera använt batteri enligt fabrikantens instruktion.

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Z

SCHEMATIC