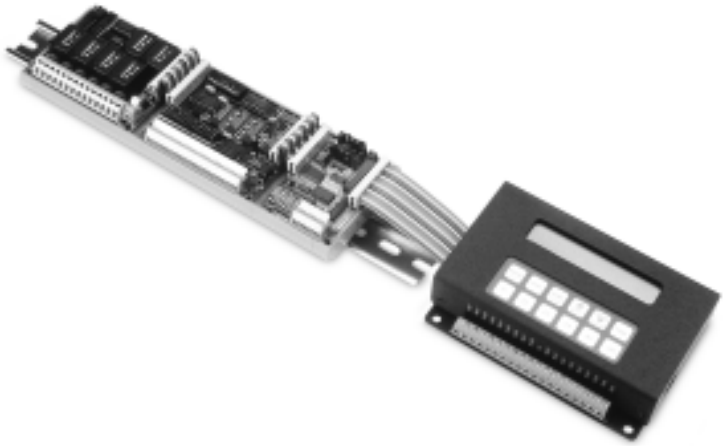


PLCBus Expansion Boards

User's Manual

019–0047 • 010620–A

PLCBus Expansion Boards User's Manual

Part Number 019-0047 • 010620-A • 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

XP8500

XP8800

XP8900

APPENDICES

ABOUT THIS MANUAL

This manual provides instructions for installing, testing, configuring, and interconnecting the Z-World PLCBus expansion boards. Instructions are also provided for using Dynamic C® functions.

Assumptions

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

- Ability to design and engineer the target system that is controlled by a controller with expansion boards attached to the PLCBus.
- 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 controllers with a Z180 microprocessor.

  For documentation from Zilog, refer to the following texts.

  Z180 MPU User's Manual
  Z180 Serial Communication Controllers
  Z80 Microprocessor Family User's Manual

- Knowledge of basic Intel assembly language and architecture for controllers with an Intel™386 EX processor.

  For documentation from Intel, refer to the following texts.

  Intel™386 EX Embedded Microprocessor User's Manual
  Intel™386 SX Microprocessor Programmer's Reference 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

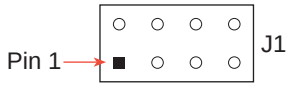
Table 3 lists and defines the typographical conventions that may be used in this manual.

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.

XP8100

CHAPTER 1: OVERVIEW

Chapter 1 provides an overview description and board layout for the XP8100 Series input/output expansion boards.

XP8100 Series Overview

The XP8100 Series consists of compact input/output (I/O) expansion boards that connect to any Z-World controller supporting a Z-World PLCBus expansion port. The XP8100 Series expansion boards can more than double the digital I/O channels of a Z-World controller.

The XP8100's 32 I/O channels are configured as 16 inputs and 16 outputs. Other versions of the board are available, as indicated in Table 1-1, for added flexibility. Up to eight XP8100 boards may be linked together to provide 256 additional I/O lines.

Table 1-1. XP8100 Series Features

Model	Features
XP8100	16 protected digital inputs and 16 output drivers
XP8110	32 protected digital inputs
XP8120	32 output drivers

Because of the similarities, this manual refers to the functionality of all three XP8100 Series boards. References to all three boards will be made by referring to them as the XP8100 Series. Individual reference will be made where needed.

Connector Terminals

Three field wiring terminals (FWT) make it easy to plug and unplug wiring connections. Table 1-2 lists the FWT available for the XP8100 Series. Any of the boards in the XP8100 Series can support two FWT of any type.

Table 1-2. XP8100 Series Options

Option	Description
FWT50	Field wiring terminal with twenty 5 mm screw terminal connectors in two banks of 10 terminals each
FWT38	Field wiring terminal with 0.15 inch (3.81 mm) quick-release connectors in two banks of 10 terminals each
FWT-Opto	Field wiring terminal for inputs only, has optical isolation, uses 0.15 inch (3.81 mm) quick-release connectors in two banks of 10 terminals each

Refer to Appendix E, "Field Wiring Terminals," for more information on how to use the FWT.

Outputs

The high-current outputs are capable of providing up to 500 mA, which is sufficient to drive inductive loads, relays, and other circuit-driven devices. The output drivers are socketed to allow a sourcing driver or TTL/CMOS parts to be added.

Inputs

The TTL/CMOS-compatible inputs can handle input signals between -19 and +20 volts. Input bias resistors may be user-configured to be pull-up or pull-down. Each input line is protected against transient voltages of -48 to +48 volts. A low-pass filter also blocks incoming voltage spikes.

Additional protection is possible by adding a field wiring terminal with optical isolation. See Table 1-2.

Factory Configurations

The XP8100 Series is available from the factory in three standard configurations, as listed in Table 1-1. Depending on the version, the board will have 32 channels of inputs, outputs or a combination of the two. It is not possible to change inputs to be outputs, or vice versa.

For ordering information, call your Z-World Sales Representative at (530) 757-3737.

XP8100 Series Default Board Layouts

The default layouts for the XP8100, XP8110 and XP8120 expansion boards are shown in Figures 1-1, 1-2, and 1-3 for the boards as they are shipped from the factory. An outline around a particular component indicates the presence of the part in the default configuration of the board.

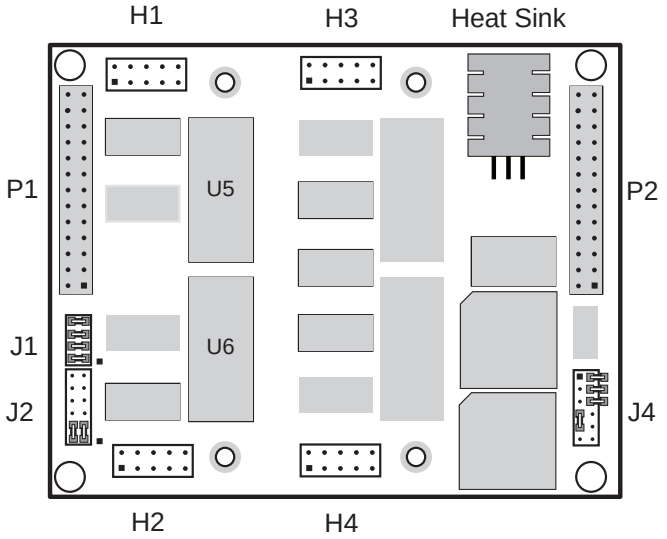


Figure 1-1. XP8100 Default Board Layout

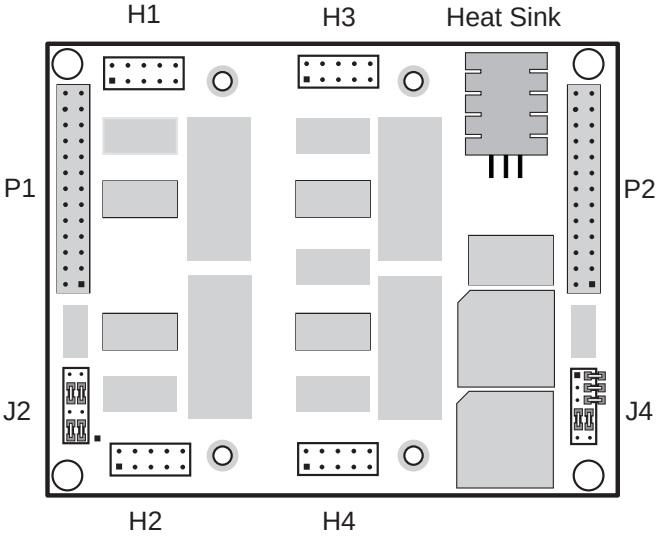


Figure 1-2. XP8110 Default Board Layout

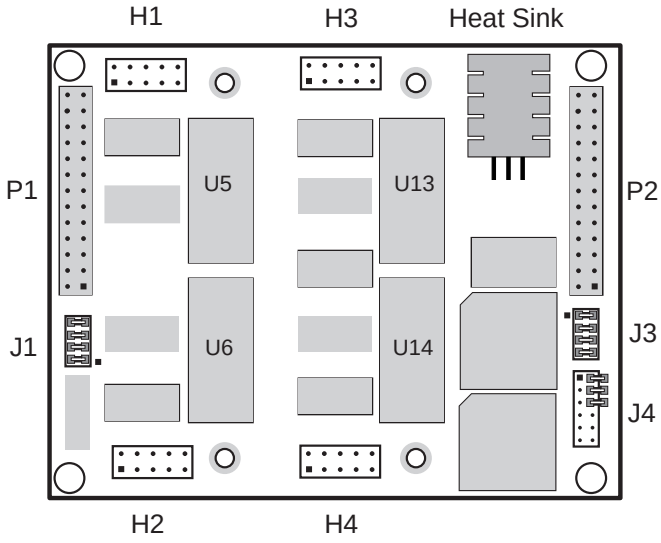


Figure 1-3. XP8120 Default Board Layout

Figure 1-4 shows the locations of the various components.

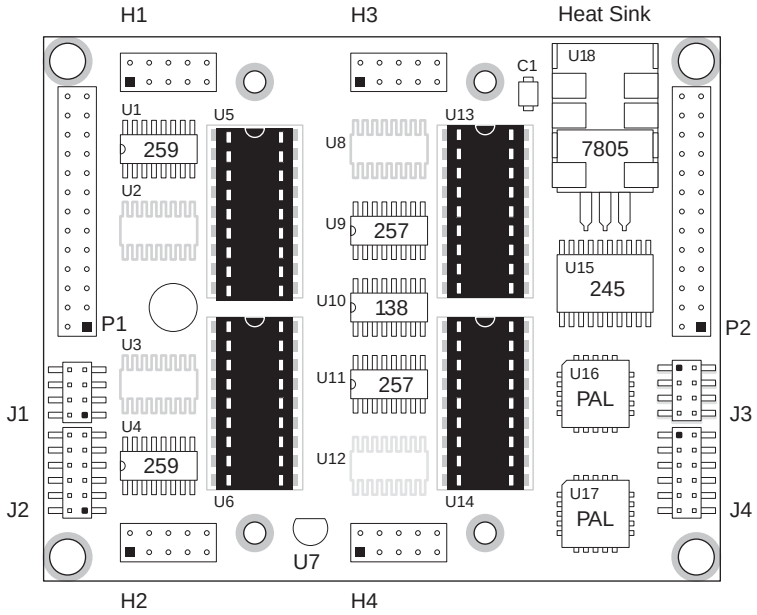


Figure 1-4. XP8100 Series Component Layout

XP8100 Hardware Specifications

Inputs

Table 1-3 summarizes the input specifications for the XP8100 Series expansion boards.

Table 1-3. Input Specifications

Input Specifications	Standard Input
Input Voltage	-20 V to +24 V
Logic Threshold	2.5 V
Bias Resistors	User-settable "pull up" or "pull down"
Transient Voltage	–48 V to +48 V max
Input Protection	22 kΩ current-limiting series resistor, input-protection diode
Noise-Spike Filter	t_{RC} = 220 µs low-pass filter
I/O Connectors	Four 10-pin headers
Input Leakage Current	5 µA

The inputs will accept a voltage level between -20 and +24 volts with a logic threshold of 2.5 volts. A 22 kΩ current-limiting resistor paired with a CMOS input diode provides input protection. The resistor/capacitor connection to ground acts as a low-pass filter, where T_{RC} = 220 µs. Jumpers pull inputs to either +5 volts or ground through a bias resistor in groups of four or eight.

Figure 1-5 shows a typical XP8100 Series expansion board input.

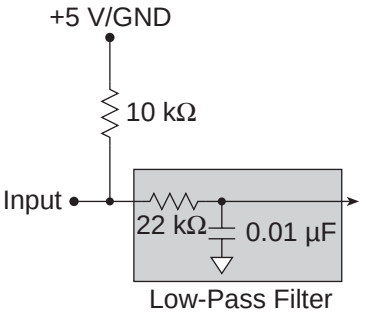


Figure 1-5. XP8100 Series Input

Outputs

Table 1-4 summarizes the output specifications for the XP8100 Series expansion boards.

Table 1-4. Output Specifications

Output Specifications	Default Sinking Driver
Maximum Current	500 mA, single channel ON
Connections	(4) 10-pin headers
Noninductive voltage	+5 V to +48 V
Inductive Voltage	+5 V to +30 V
Switching Response Time	1 µs
Output Leakage Current	100 µA max

The maximum current is subject to the maximum power dissipation for the package and the ambient temperature. Make sure that the maximum current is properly derated for temperature and package power dissipation.

See Chapter 3, "I/O Configurations," for more information on derating.

All outputs are arranged in groups of eight and are driven by a ULN2803 sinking driver. If installed, the chip would be located at U5, U6, U13, or U14, shown in Figure 2-1 and in Chapter 1.

The sinking driver is rated up to a maximum voltage of 48 V and a maximum current of 500 mA per individual output. When all the outputs are on simultaneously, thermal limits restrict the current to 100 mA per output. Similarly, if multiple outputs are activated at the same time, the driver current should not exceed 350 mA per output.

A UDN2985 sourcing driver is optional. The UDN2985 is rated at 30 V and 250 mA for an individual output at 25°C. The sourcing drivers would be installed at U5, U6, U13, or U14 instead of the sinking drivers, and jumpers on headers J1 and J3 would be reconfigured, as discussed in Appendix D.

Refer to "Sourcing and TTL/CMOS Outputs" in Chapter 3 for information on installing and configuring your board for sourcing outputs and for TTL/CMOS outputs.

Mechanical Dimensions

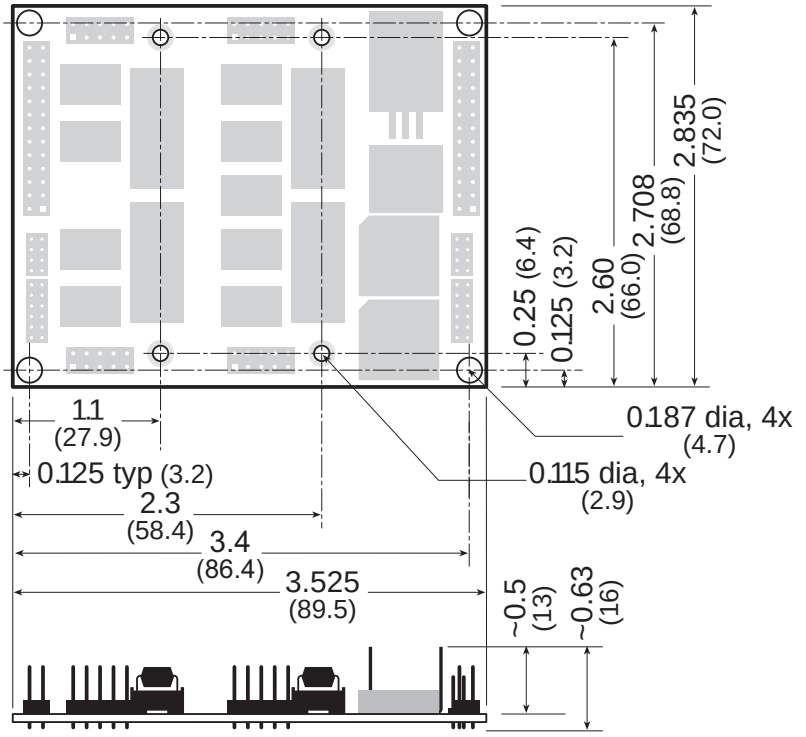


Figure 1-6. XP8100 Series Board Dimensions

CHAPTER 2: GETTING STARTED

Chapter 2 provides instructions for connecting XP8100 Series expansion boards to a Z-World controller. The following sections are included.

- Expansion Board Components
- Connecting Expansion Boards to a Z-World Controller
- Confirming Communications

XP8100 Series Components

The XP8100 Series boards offer protected digital inputs and high-current driver outputs. Figure 2-1 illustrates the basic layout and orientation of the expansion boards, including headers and other components. Some headers and other devices may not be present, depending on the specific board (XP8100, XP8110, or XP8120).

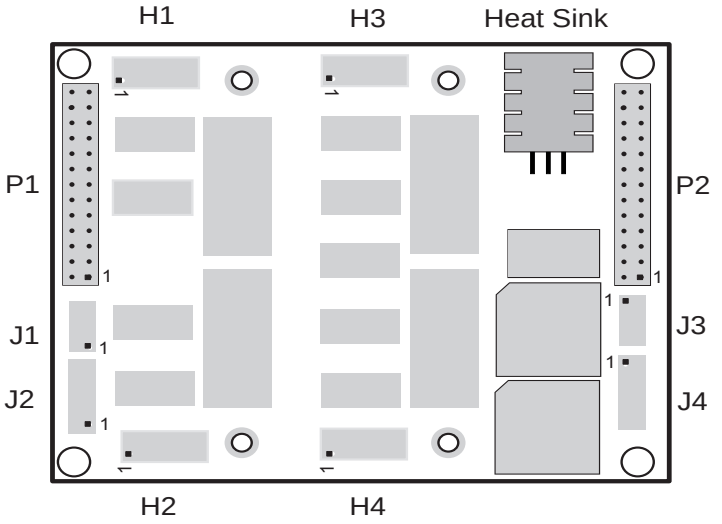


Figure 2-1. XP8100 Series Board Layout

Pay particular attention to the location of pin 1 of headers J1–J4, as indicated by a small squares in Figure 2-1. The layout orientation of J1 and J2 is opposite that of J3 and J4, so the pin 1 locations are rotated 180 degrees. Figure 2-1 is referenced throughout the manual.

See Chapter 1, "Overview," for the exact layouts of the XP8100, XP8110 and XP8120 expansion boards.

Be careful to orient H1, H3, and the heat sink to the top, as shown in Figure 2-1, when referring to jumper and header locations.

Connecting Expansion Boards to a Z-World Controller

Use the 26-conductor ribbon cable supplied with an expansion board to connect the expansion board to the PLCBus on a Z-World controller. See Figure 2-2. The expansion board's two 26-pin PLCBus connectors, P1 and P2, are used with the ribbon cable. Z-World recommends using the cable supplied to avoid any connection problems.

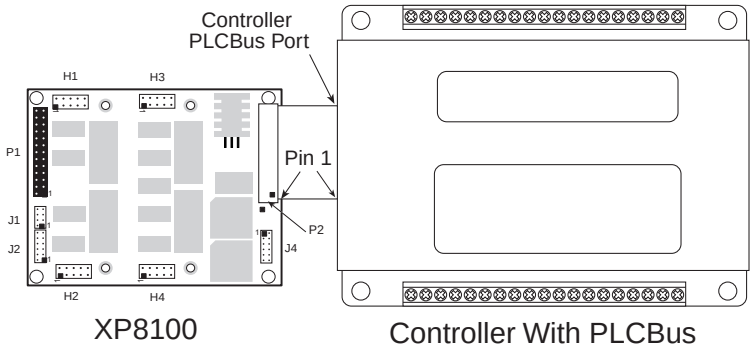


Figure 2-2. Connecting XP8100 Expansion Board to Controller PLCBus

Be sure power to the controller is disconnected before adding any expansion board to the PLCBus.

Follow these steps to connect an expansion board to a Z-World controller.

1. Attach the 26-pin ribbon cable to the expansion board's **P2** PLCBus header.
2. Connect the other end of the ribbon cable to the PLCBus port of the controller.

Be sure pin 1 of the connector cable matches up with pin 1 of both the controller and the expansion board(s).

3. If additional expansion boards are to be added, connect header **P2** on the new board to header **P1** of the board that is already connected. Lay the expansion boards side by side with headers P1 and P2 on adjacent boards close together, and make sure that all expansion boards are facing right side up.

See Appendix B, "Connecting and Mounting Multiple Boards," for more information on connecting multiple expansion boards.

Setting Board Addresses

Z-World has established an addressing scheme for the PLCBus on its controllers to allow multiple expansion boards to be connected to the controller.

Every XP8100 Series board is shipped from the factory with a default address of 7. An XP8100 Series board may be assigned any address between 0 and 7. Jumpers are placed on the pins of header J4 to configure the board address. Figure 2-3 shows the jumper settings to set addresses 0–7.

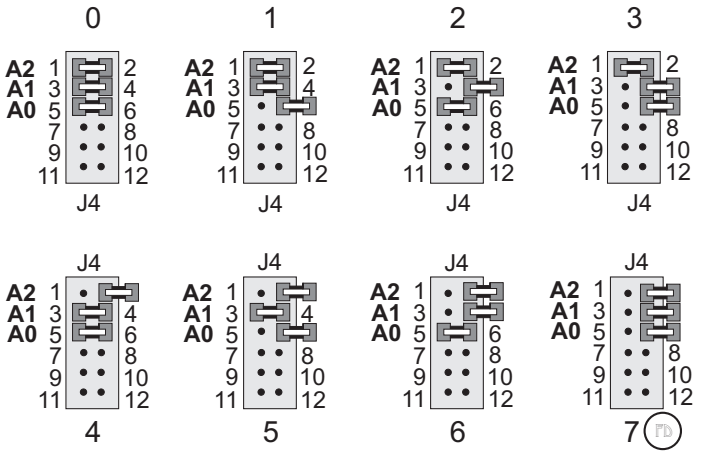


Figure 2-3. J4 Jumper Settings for XP8100 Series PLCBus Addresses

Only the first six pins of the 12-pin header J4 on the XP8100 Series are used to set the board address.

Remember that each expansion board must have a unique PLCBus address if multiple boards are to be connected. If two boards have the same address, communication problems will occur that may go undetected by the controller. A maximum of eight XP8100 boards may be addressed by a controller at one time.

Power

Z-World's expansion boards receive power from the controller over the +24 V line of the PLCBus. An onboard regulator converts this to the +5 V used by the expansion boards. The expansion boards draw about 110 mA, which means a power requirement of 1.3 W for a 12 V controller and 2.6 W for a 24 V controller.

Power may be applied to the controller once the controller and the expansion boards are properly connected using the PLCBus ribbon cable.

Confirming Communications

Run the following test program once the XP8100 Series expansion board is connected to a controller and power is applied. The sample program will confirm whether the controller and expansion board are communicating properly.

See the ***Dynamic C Technical Reference*** manual for more detailed instructions.

XP81ID.C

```
#use vdriver.lib
#use eziocmmn.lib
#use eziopbdv.lib
// uncomment #use ezioplc.lib line for PK2100(Rugged
// Giant), PK2200(Little Star), BL1200(Little PLC),
// BL1600(Little G)
//#use ezioplc.lib
// uncomment #use eziomgpl.lib line for BL1400(Micro-G)
// or BL1500(Micro-G2)
//#use eziomgpl.lib
char TITLE[] = {"XP81xx Board Detection"};
main(){
   int i;
   VdInit();
   printf("%s\n\n", TITLE);
   eioResetPlcBus();     // reset the PLCBus
   eioPlcRstWait();      // delay ensures the PLCBus
                         // boards reset
   // locate all possible jumper-set addresses
   // from 0 to 7 and display status
   for (i = 0; i <= 7; ++i) {
      // read to locate the board
      if (plcXP81In(i*32)==-1)
         printf("Board %d is not located\n",i);
      else
         printf("Board %d is located\n",i);
   }
}
```

Use the following steps to run the sample program.

1. Open the sample program **XP81ID.C** located in the Dynamic C **samples\plcbus** subdirectory. This program is designed to locate and display the address numbers of XP8100 Series boards connected on the PLCBus.
2. Be sure to "uncomment" the appropriate library at the top of the sample program for the particular controller being used. Do this by removing the forward slashes (**//**) in front of the appropriate **#use** library.

3. Compile the program by pressing **F3** or by choosing **Compile** from the **COMPILE** menu. Dynamic C compiles and downloads the program into the controller's memory. During compilation, Dynamic C rapidly displays several messages in the compiling window, which is normal.
4. Run the program by pressing **F9** or by choosing **Run** from the **RUN** menu.
5. The **STDIO** window will display a message once the program is running. If communication between the XP8100 Series expansion board and the controller is ok, the message will be **Board (#) is located.** If a problem exists with communications, the message will be **Board (#) is not located.** Remember that the default address is 7 for XP8100 Series expansion boards.
6. To halt the program, press **<CTRL Z>**.
7. To restart the program, press **F9**.

Check the board jumpers, PLCBus connections, and the PC/controller communications if an error message appears.

CHAPTER 3: I/O CONFIGURATIONS

Chapter 3 describes the built-in flexibility of the XP8100 Series expansion boards, and describes how to configure the available inputs/outputs. The following sections are included.

- Input/Output Pin Assignments
- Inputs
- Outputs
- Making Input/Output Connections

XP8100 Series Input/Output Pin Assignments

There are two “banks” of inputs/outputs that total up to 32 inputs/outputs for the XP8100 Series expansion boards. Bank A consists of headers H1 and H2, Bank B consists of headers H3 and H4. Figure 3-1 shows an outline of input/output Banks A and B.

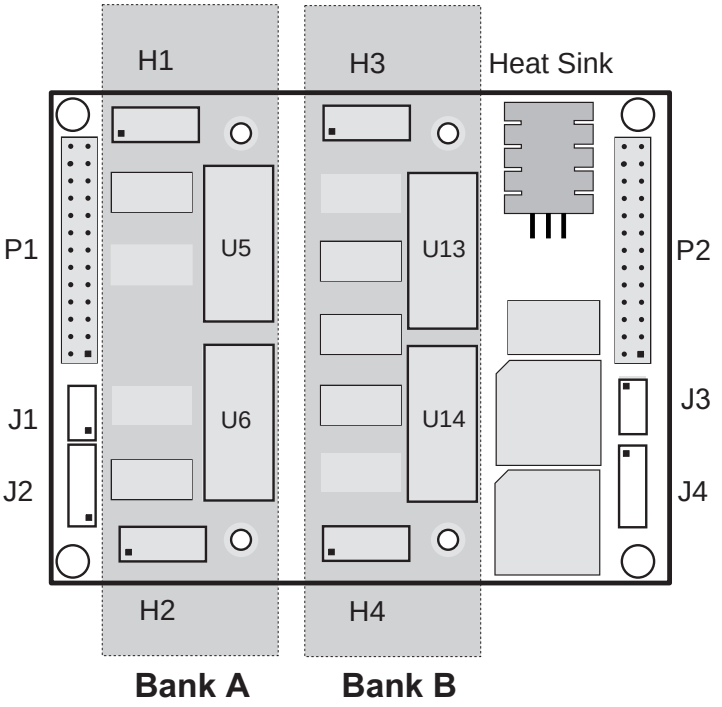


Figure 3-1. Outline of Input/Output Banks A and B

Banks A and B each have 16 input/output channels. The pins on headers H1 through H4 will function either as inputs or as outputs, depending on the specific XP8100 Series model.

Each header (H1–H4) contains a group of 10 pins. The 10 pins on each individual header function similarly to one another.

Some headers and other devices may or may not be present, depending on the specific XP8100 Series expansion board. See Chapter 1, “Overview,” for the exact board layouts.

Table 3-1 lists the functionality of the header pins for the XP8100 Series expansion boards.

Table 3-1. Header I/O Designations

	I/O Bank A		I/O Bank B	
	H1	H2	H3	H4
XP8100	8 outputs	8 outputs	8 inputs	8 inputs
XP8110	8 inputs	8 inputs	8 inputs	8 inputs
XP8120	8 outputs	8 outputs	8 outputs	8 outputs

The pin locations are different for the optional field wiring terminal (FWT) blocks described in Table 1-2. Refer to Appendix F, “Using FWT Boards,” for the correct header and pin locations in these circumstances.

XP8100 Series Inputs

Protected Digital Inputs

The XP8100 and XP8110 boards are equipped with protected digital inputs designed as logical data inputs, returning a 1 or 0. The inputs accept voltages between -20 V and +24 V DC, with a logic threshold of 2.5 V DC. This means that an input returns a 0 if the input voltage is below 2.5 V, and a 1 if the input voltage is above 2.5 V DC.

A low-pass filter on each input channel has a time constant of:

T_{RC} = 220 µs at 4.5 kHz.

If the XP8100 Series board has inputs, they may be configured as "pull-up" or "pull-down" in groups of fours and eights. The configuration of each input should be determined by normal operating conditions, powerdown mode, and possible failure modes, including open or shorted conditions. These factors will influence decisions about whether to configure the inputs as "pull-up" or "pull-down."

FD The factory default is for inputs to be pulled up to +5 V.

Inputs may be pulled up to +5 V or pulled down to ground by configuring jumpers on headers J2 and J4.

See Figure 3-1 to help locate headers J2 and J4.

The jumpers on headers J2 and J4 configure the inputs on Bank A (H1 and H2) and Bank B (H3 and H4) as pull-up or pull-down To pull down an input from the factory default (pull-up), place a jumper across the appropriate two pins of J2 and/or J4, as shown in Figures 3-2 and 3-3 for the XP8100 and XP8110 expansion boards.

> **!** Input lines connected to opto-isolator devices must be configured as "pull-up." Otherwise, the expansion board may be damaged.

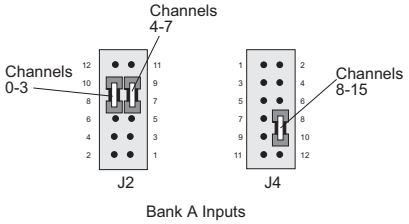
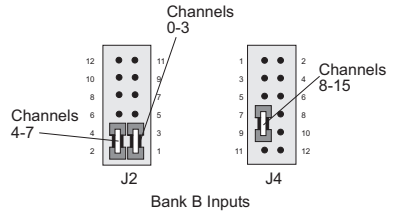
Pull-Up Configurations	Function
	Note: Other jumpers may be present on J2 and J4. The J2 and J4 jumper configurations relate to Bank A inputs 0-15.
	Note: Other jumpers may be present on J2 and J4. The J2 and J4 jumper configurations relate to Bank B inputs 0-15.

Figure 3-2. XP8100 Series Jumper Pull-Up Configurations

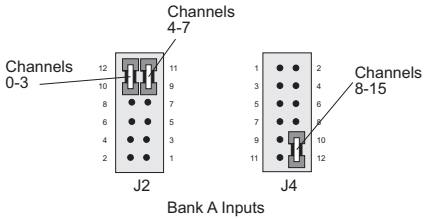
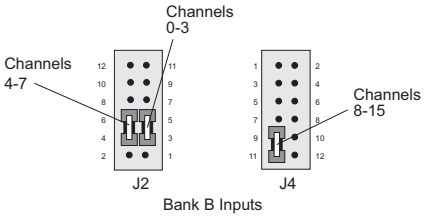
Pull-Down Configurations	Function
	Note: Other jumpers may be present on J2 and J4. The J2 and J4 jumper configurations relate to Bank A inputs 0-15.
	Note: Other jumpers may be present on J2 and J4. The J2 and J4 jumper configurations relate to Bank B inputs 0-15.

Figure 3-3. XP8100 Series Jumper Pull-Down Configurations

Note that board address jumpers occupy the top three rows of header J4 (pins 1–6) as seen relative to the heat sink being at the top of the board.

XP8100 Series Outputs

The XP8100 Series expansion boards are shipped from the factory with the outputs configured with “sinking” drivers. The sinking drivers are rated up to a maximum output voltage of 48 V and a maximum current of 500 mA per individual output when only one output in a particular bank is active at once.

When all outputs are on simultaneously, thermal limits restrict the current to less than 100 mA per output. Similarly, if multiple outputs are turned on at the same time, the driver current should not exceed 350 mA per output. If the temperature exceeds 50°C, derate power dissipation by 55°C/W.

Jumpers across headers J1 and J3 define the sinking or sourcing configuration of the outputs. For the default sinking setting, the jumpers are placed horizontally across headers J1 and J3, as shown in Figure 3-4. The XP8100 uses only header J1 and the XP8120 uses both headers J1 and J3.

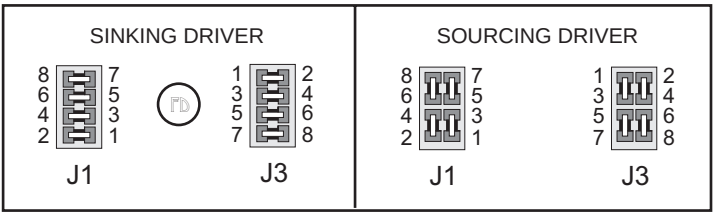


Figure 3-4. Jumper Configurations for Sinking and Sourcing Outputs

The factory default is for outputs to be configured with sinking drivers (ULN2803).

Sinking and Sourcing Outputs

Figure 3-5 shows a typical sinking driver output configuration.

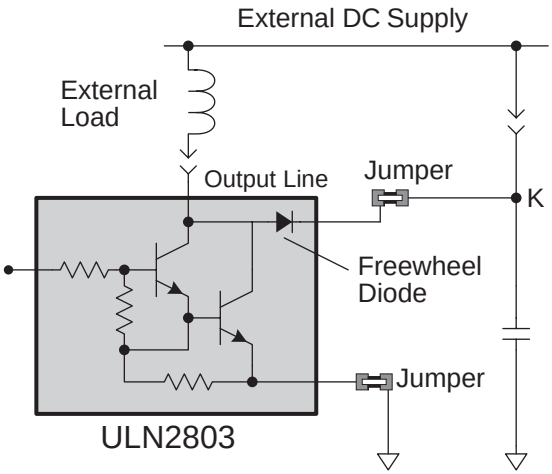


Figure 3-5. XP8100 Series Sinking Driver Output

Sourcing outputs are possible by replacing the factory-installed sinking driver chips with sourcing output drivers (UDN2985). The UDN2985 sourcing driver chip is capable of sourcing a maximum of 250 mA per output.

Figure 3-6 shows a typical sourcing driver output.

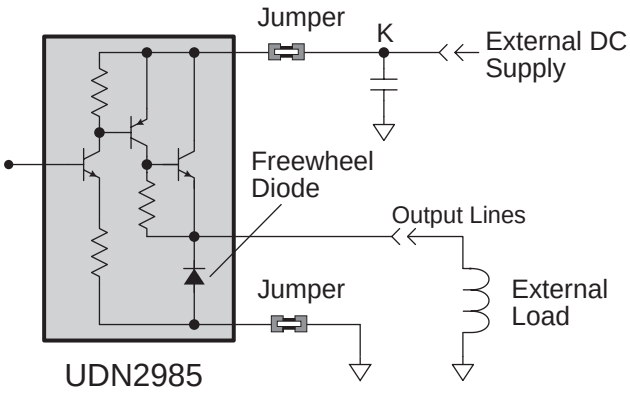


Figure 3-6. XP8100 Series Sourcing Driver Output

Installing Sourcing Drivers

Figure 3-7 shows the location of the drivers and headers with jumpers to be changed.

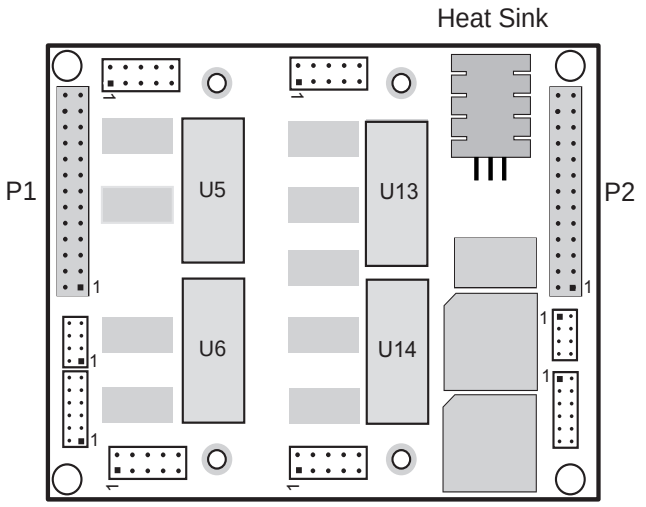


Figure 3-7. U5, U6, U13 and U14 Locations of Sinking Drivers

Pay particular attention to the orientation of the jumpers when changing the driver output from sinking to sourcing. Exercise caution when installing sourcing drivers in the field.

1. Be sure power is removed from the controller, then disconnect the expansion card from the controller..
2. Remove the ULN2803 sinking drivers from the IC sockets. Note that the XP8100 has two ULN2803 chips (at U5 and U6) and the XP8120 has four (U5, U6, U13 and U14).
3. Install the jumpers on header J1 in the sourcing configuration, as shown in Figure C-4, for "Bank A" output channels 0–15. This step applies to both the XP8100 and the XP8120 expansion boards. Note the location of pin number 1.
4. For the XP8120 expansion board, install the J3 jumpers in the sourcing configuration, as shown in Figure D-4, for "Bank B" output channels 0–15. Note the location of pin number 1.

> ⚠ Be sure the jumper settings conform to what is specified. Failure to install jumpers correctly will cause the expansion board to fail.

5. Install UDN2985 sourcing driver chips into the IC sockets.

Z-World also offers all XP8100 Series expansion boards with factory-installed sourcing drivers. For ordering information, call your Z-World Sales Representative at (530) 757-3737.

Tables 3-2 and 3-3 indicate which I/O channels are modified by the jumpers on the J1 and J3 headers. The tables also list the specific location of each output chip.

Table 3-2. Header J1 I/O Channels

J1 Pins	"Bank A" I/O Channels Modified	IC Location
1–4	8–15	U6
5–8	0–7	U5

Table 3-3. Header J3 I/O Channels

J3 Pins	"Bank B" I/O Channels	IC Location
1-4	8-15	U14
5-8	0-7	U13

TTL/CMOS Outputs

Z-World also offers TTL- or CMOS-compatible outputs for the XP8100 Series expansion boards. Input and output channels may be configured independently in any combination. However, the functionality of each input is not independent; the inputs are still characterized in groups of eight.

Z-World offers all XP8100 Series expansion boards with factory-installed TTL- or CMOS-compatible outputs. For ordering information, call your Z-World Sales Representative at (530) 757-3737.

Using Output Drivers

The common supply for the digital output channels supplied by a driver chip is called "K," and is available on pin 10 of headers H1, H2, H3, and H4 when they are configured to operate as digital outputs. "K" must be used with digital outputs to allow proper operation.

The "K" connection performs two vital functions to the high-voltage driver circuitry on the XP8100.

1. "K" supplies power to driver circuitry inside the driver chip.
2. "K" also allows a diode internal to the driver chip to "snub" voltage transients produced during the inductive kick associated with switching inductive loads such as relays, solenoids, and speakers.

Long leads may present enough induction to also produce large potentially damaging voltage transients. The anodes of the protection diodes for each channel are common, and so only one voltage supply can be used for all high-voltage driver loads.

The following points summarize the functions of "K."

- K provides power to the driver chip circuitry.
- K provides "clamping" for all high-voltage driver loads.
- It is mandatory to connect K regardless of whether sourcing or sinking.
- The load's supply must have a common ground with all other supplies in your system.
- All loads must use same supply voltage.

K must be connected to the power supply used for the high-voltage load as shown in Figure 3-8.

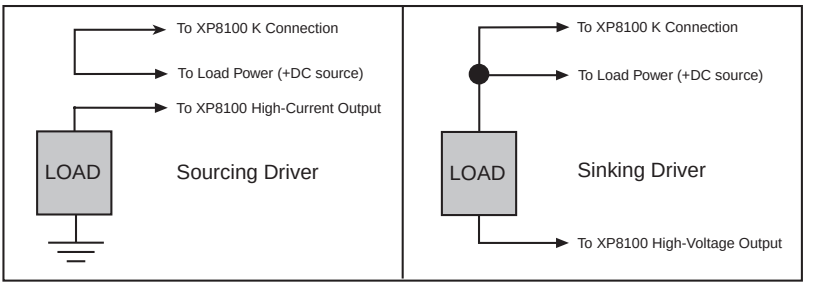


Figure 3-8. XP8100 K Connections

Making XP8100 Series I/O Connections

The four 10-pin headers (H1–H4) accept either ribbon-cable connectors or up to two XP8100 Series FWT blocks for input/output connections. Input and output lines are wired to the 10-pin headers directly using a custom-built cable and connector, or by using the FWT connectors available from Z-World.

The hardware pin assignments for each header are referenced in Figure 3-9. Note that the first pin, indicated by the square, is labeled zero.

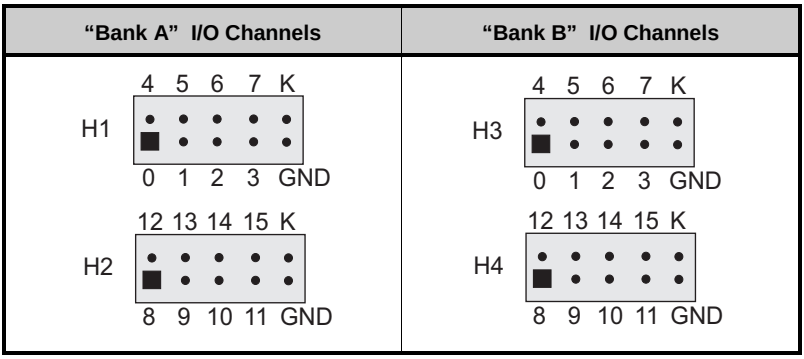


Figure 3-9. XP8100 Series Header H1–H4 Pin Assignments

Note that the hardware pin assignments for Bank B (H3 and H4) do not match up with the Bank B software input/output assignments. Both hardware and software assignments are cross-referenced in Table 4-2 in Chapter 4, "Software Reference."

Inputs/outputs may be connected with discrete wires instead of a ribbon cable. Refer to Appendix E, "Field Wiring Terminals," for information on the optional FWT connectors.

Pay close attention to the locations of pins on the header when connecting inputs/outputs.

I/O Jumper Configurations

There are four headers for jumper blocks. Depending on the specific XP8100 Series expansion board, not all the four headers may be installed on a particular board. Headers J1 and J3 are used to configure outputs, while headers J2 and J4 are used to configure inputs. Header J4 is present on all XP8100 Series expansion boards, and is used to configure inputs and address settings.

Table 3-4 lists the headers that are installed specifically for each XP8100 Series expansion board and provides a reference to the jumper configurations.

Table 3-4. XP8100 Series I/O Jumper Configurations

Header	Pins Connected	Configures
XP8100—16 inputs and 16 outputs		
J1	Sinking or sourcing drivers: see Figure 3-4	"Bank A" Output Channels 0–15
J2	Pull-up or pull-down inputs: see Figures 3-2 and 3-3	"Bank B" Input Channels 0–7
J4		"Bank B" Input Channels 8–15 and board address
XP8110—32 inputs		
J2	Pull-up or pull-down inputs: see Figures 3-2 and 3-3	"Bank A" Input Channels 0–7 and "Bank B" Input Channels 0–7
J4		"Bank A" Input Channels 8–15, "Bank B" Input Channels 8–15, and board address
XP8120—32 outputs		
J1	Sinking or sourcing drivers: see Figure 3-4	"Bank A" Output Channels 0–15
J3		"Bank B" Output Channels 0–15
J4	—	Board address only

See Figure 2-3 in Chapter 2, "Getting Started," for the jumper configurations to set board addresses.

CHAPTER 4: FIELD WIRING TERMINALS

Discrete input/output lines may be connected to any of the XP8100 Series expansion boards with field wiring terminal (FWT) modules. This eliminates the need for ribbon cables. The optional quick-disconnect modules provide screw terminals for simple wiring.

Each module mates to two of the XP8100 Series board headers (H1–H2 and H3–H4). This is equivalent to 16 connections per module. One XP8100 Series expansion board can accept up to two FWT modules in any combination. The FWT50, FWT38, and the FWT-Opto modules are available.

Figures 4-1 and 4-2 show the mounting configuration for the FWT modules.

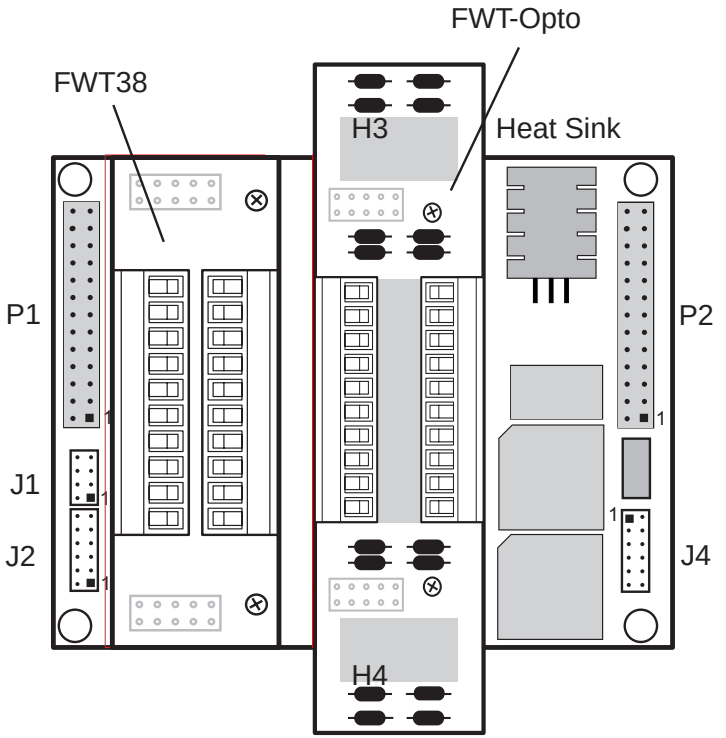


Figure 4-1. Top View of XP8100 with FWT Modules

The four FWT styles described in this appendix are available from Z-World. Your application may use a different arrangement than that shown in Figure 4-1.

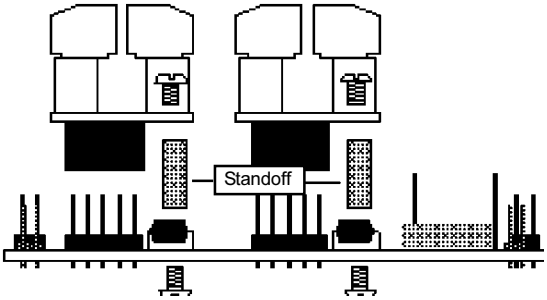


Figure 4-2. FWT Installation

FWT38

The FWT38 has 20 terminals in two groups with 10 terminals each. Each group of terminals may be removed independently.

Table 4-1 summarizes the specifications for the FWT38.

Table 4-1. FWT38 Specifications

Parameter	Specification
Total I/O Channels	16
Screw Terminal Pitch	3.81 mm
Maximum Wire Gauge	28-16 AWG
Quick-Disconnect Capability	Wiring banks can be unplugged from the board separately (Phoenix Combicon type connection)
Wire Orientation	Top-exiting wires

Figure 4-3 provides the dimensions for the FWT38.

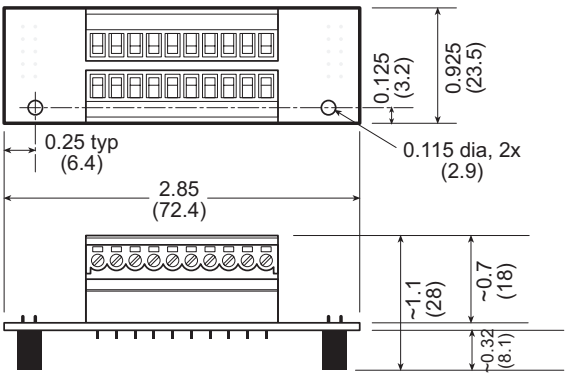


Figure 4-3. FWT38 Dimensions

Figure 4-4 shows the I/O channel assignments and pinouts for the FWT38.

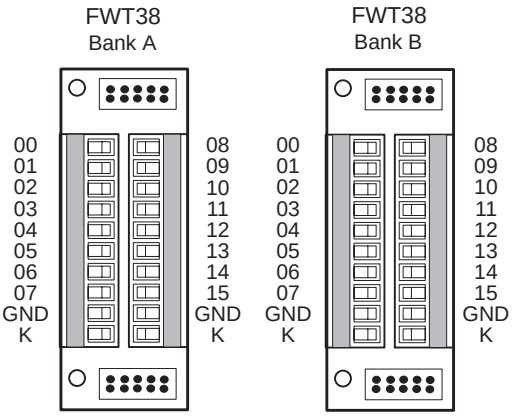


Figure 4-4. FWT38 Pinouts

FWT50

The FWT50 provides 20 screw terminals. The terminal connectors are fixed to the FWT module and cannot be removed.

Table 4-2 summarizes the specifications for the FWT50.

Table 4-2. FWT50 Specifications

Parameter	Specification
Total I/O Channels	16
Screw Terminal Pitch	5.00 mm
Maximum Wire Gauge	24-12 AWG
Quick-Disconnect Capability	Unplugs from the XP8100 board as a single unit
Wire Orientation	Side-exiting wires

Figure 4-5 provides the dimensions for the FWT50.

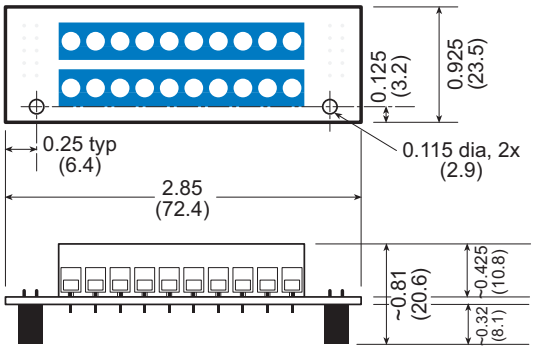


Figure 4-5. FWT50 Dimensions

Figure 4-6 shows the I/O channel assignments and pinouts for the FWT50.

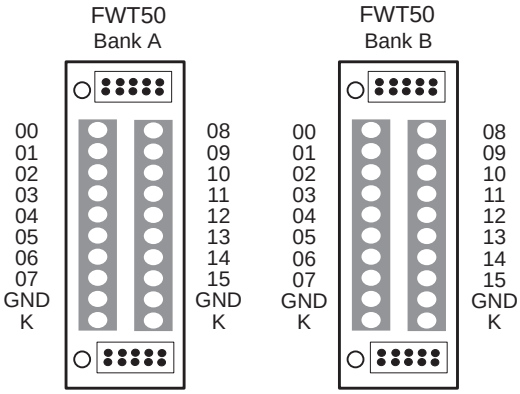


Figure 4-6. FWT50 Pinouts

FWT-Opto

The FWT-Opto provides optical isolation to the input channels. The FWT-Opto is used only for inputs, and is not used with the XP8120 expansion board. All 16 channels must be committed to inputs when an FWT-Opto module is used.

Every four FWT-Opto inputs share a common return. The excitation resistors need to be pulled up to +5 V when the FWT-Opto module is used.

Table 4-3 lists the specifications for the FWT-Opto module.

Table 4-3. FWT-Opto Specifications

Parameter	Specification
Total Input Channels	16 optically isolated input channels only
Screw Terminal Pitch	3.81 mm
Maximum Wire Gauge	28-16 AWG
Quick-Disconnect Capability	Wiring banks can be unplugged from the board separately (Phoenix Combicon type connection)
Wire Orientation	Top-exiting wires
Input Protection Range	5 kV rms between input and output
Maximum Input Voltage	±40 V
Guaranteed Input Switching Threshold	±9.5 V

The FWT-Opto module uses 4.7 kΩ input resistors to accommodate the large range of input voltages. This limits the input switching threshold to ±9.5 V. These 4.7 kΩ input resistors need to be replaced with 1.2 kΩ input resistors to handle smaller input voltages such as 5 V logic. If 0.125 W resistors are used, this will limit the maximum input voltage to ±12.2 V.

Figure 4-7 provides the dimensions for the FWT-Opto module.

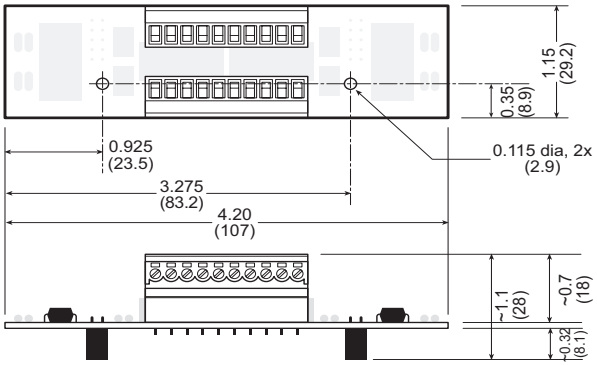


Figure 4-7. FWT-Opto Dimensions

Figure 4-8 shows the input channel assignments and pinouts for the FWT-Opto module.

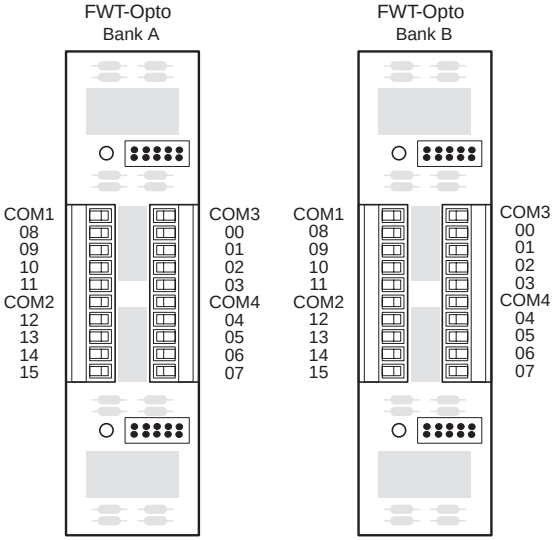


Figure 4-8. FWT-Opto Pinouts

Figure 4-9 shows an FWT-Opto optical isolation circuit.

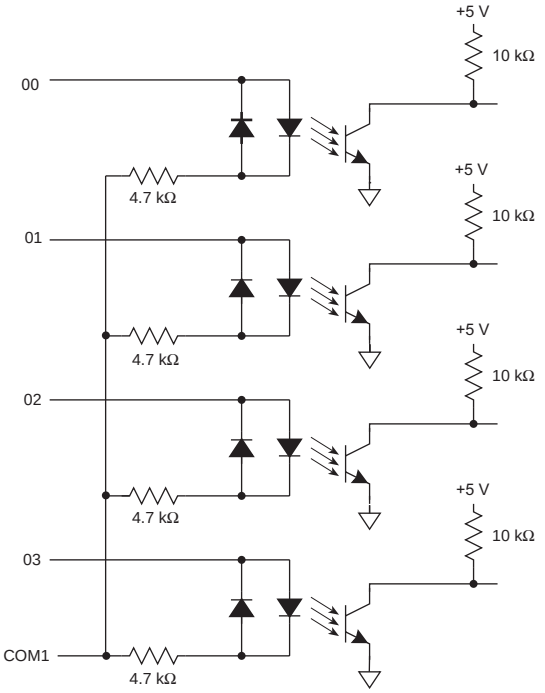


Figure 4-9. FWT-Opto Optical Isolation Circuit

The opto-isolated inputs share a common return in groups of four. The software channel assignments remain the same for Banks A and B.

CHAPTER 5: SOFTWARE REFERENCE

Chapter 5 describes the Dynamic C functions that initialize the XP8100 Series expansion boards, and perform input/output operations. The following major sections are included.

- Software Input/Output Channel Assignments
- Software Overview
- Digital Inputs/Outputs
- Advanced Input/Output Programming

XP8100 Series Software Input/Output Channel Assignments

Together, the four headers of Banks A and B provide a total of 32 inputs/outputs. In hardware, the input/output channels are numbered 0–15 for Bank A and are also numbered 0–15 for Bank B. However, the channels must have unique software numbers, and so the inputs/outputs for Bank A retain their numbering of 0–15, but the inputs/outputs for Bank B are numbered 16–31.

Therefore, header H1 consists of software I/O channels 0–7, header H2 consists of software I/O channels 8-15, header H3 consists of software I/O channels 16–23, and header H4 consists of software I/O channels 24–31.

See Chapter 1, "Overview," for the board layouts showing the exact locations of the headers.

Table 5-1 summarizes the software I/O assignments for each header.

Table 5-1. I/O Channel Assignments for XP8100 Series Headers

Header	Software I/O Channels
H1	0–7
H2	8–15
H3	16–23
H4	24–31

Table 5-2 lists the software I/O channel assignments for each header pin. The table details the software function number assigned to the actual hardware pin for headers H1–H4. Refer to this table when planning which channel to activate or read during program development.

Table 4-2. I/O Channel Assignments forXP8100 Header Pins

Hardware Headers H1–H4 Pin Channel Assignment	Bank A		Bank B	
	H1 Software Channel Assignment	H2 Software Channel Assignment	H3 Software Channel Assignment	H4 Software Channel Assignment
0	0	–	16	–
1	1	–	17	–
2	2	–	18	–
3	3	–	19	–
4	4	–	20	–
5	5	–	21	–
6	6	–	22	–
7	7	–	23	–
8	–	8	–	24
9	–	9	–	25
10	–	10	–	26
11	–	11	–	27
12	–	12	–	28
13	–	13	–	29
14	–	14	–	30
15	–	15	–	31

Software Overview

This section describes a set of simple software functions to use when controlling the XP8100 Series expansion board inputs/outputs.

See the section "Advanced Programming" later in this chapter to get more information on developing applications to meet tight timing requirements.

Dynamic C Libraries

Several Dynamic C function libraries need to be used with the routines defined in this chapter. There are three common libraries used by all Z-World controllers and specific libraries designed for certain controllers. The chart in Table 5-3 identifies which libraries must be used with particular Z-World controllers.

Table 4-3. Dynamic C Libraries Required by Z-World Controllers

Library Needed	Controller
VDRIVER.LIB	All controllers
EZIOCMMN.LIB	All controllers
EZIOPBDV.LIB	All controllers
EZIOTGPL.LIB	BL1000
EZIOLGPL.LIB	BL1100
EZIOMGPL.LIB	BL1400, BL1500
EZIOPLC.LIB	BL1200, BL1600, PK2100, PK2200
EZIOPLC2.LIB	BL1700

Before using one of these libraries in an application, first include the library name in a **#use** command. For example, to use functions in the library **EZIOPLC.LIB**, be sure there is a line at the beginning of the program in the following format.

```
#use ezioplc.lib
```

Supplied Software

These Dynamic C functions are used to initialize the PLCBus. Call these functions in a program before any code to read inputs or set outputs.

- **`VdInit()`**

  Initializes the timer mechanism.

  LIBRARY: **`VDRIVER.LIB`**

- **`void eioResetPlcBus()`**

  Resets all expansion boards connected to the PLCBus.

  When using this function, initialize timers with **`VdInit()`** before resetting the PLCBus. All PLCBus devices must reset before performing any subsequent operations.

  LIBRARY: **`EZIOPLC.LIB`**

- **`void eioPlcRstWait()`**

  Provides a delay long enough for the PLCBus to reset.

  This function provides a delay of 1–2 seconds to ensure devices on the PLCBus reset. This function should be called after resetting the PLCBus.

  LIBRARY: **`EZIOPBDV.LIB`**

- **`long int eioErrorCode`**

  Represents a global bit-mapped variable whose flags reflect error occurrences.

  This register for this variable is initially set to 0. If the application tries to access an invalid channel, the flag **`EIO_NODEV`** (the first bit flag) is set in this register. Note that the other bits in **`EIO_NODEV`** deal with networked controllers.

Digital Inputs/Outputs

The following functions provide an easy way to read inputs and activate outputs. The digital input and output functions are located in the Dynamic C **EZIOPBDV.LIB** library.

Setting Inputs

- **int plcXP81In(unsigned eioAddr)**

  Reads the state of an XP8100 Series input channel.

  PARAMETER: **eioAddr** specifies the board address and the input pin to be read. Use the following formula in the function's argument to determine **eioAddr**.

  ```
  32*brdNum+pin
  ```

  The variable **brdNum** is the board address (the default address is 7, as explained in Chapter 2) and the variable **pin** is the input being read (software pin assignment **0**–**31**).

  RETURN VALUE:

  - 0 if the board is found and the input channel reads low.
  - 1 if the board is found and the input channel reads high.
  - Sets the flag **EIO_NODEV** in **eioErrorCode** and returns –1 if the channel does not exist (that is, if **eioAddr** is greater than **31**).

For the XP8100 board, **eioAddr** is a number ranging from 16 through 31. For the XP8110 board, **eioAddr** is a number ranging from 0 through 31.

Program 5-1 demonstrates how to read the status of a digital input.

Program 5-1. Input Demonstration Program

```
#use vdriver.lib
#use eziocmmn.lib
#use eziopbdv.lib
// uncomment #use ezioplc.lib line below for
// PK2100(Rugged Giant), PK2200(Little Star),
// BL1200(Little PLC) and BL1600(Little G)
// #use ezioplc.lib
// uncomment #use eziomgpl.lib line below for
// BL1400(Micro-G) or BL1500(Micro-G2)
// #use eziomgpl.lib
char TITLE[] = {"XP81xx Digital Input"};
main() {
   int channum;
   int i, j;
   VdInit();
   printf("%s\n\n", TITLE);
   eioResetPlcBus();      // reset the PLCBus
   eioPlcRstWait();       // delay ensures the
                          // PLCBus boards reset
                          // locate all possible
                          // jumper-set addresses
                          // from 0 to 7 and
                          // display status
   for (i = 0; i <= 7; ++i){
      if (plcXP81In(i*32)==-1) {    // do a read to
                                    // locate the board
         printf("Board %d is not located\n\n",i);
      }
      else {
         printf("Board %d is located\n",i);
         //read each channel from 0 to 31 and display status
         printf("Reading all 32 positions\n");
         for (channum = 0; channum <= 31; ++channum) {
            j = plcXP81In(i*32+channum);
            // read the input of the channel
            printf("HV%d reads %d\n", channum, j);
         }
      printf("\nPress a key to continue...\n");
      while (!kbhit());
      getchar();
      }
   }
}
```

Setting Outputs

- **`int plcXP81Out(unsigned eioAddr, int state);`**

  Writes to an output channel.

  PARAMETERS: **`eioAddr`** specifies both the board address and the output to turn on or off. Use the following formula in the function's argument to determine **`eioAddr`**.

  ```
  32*brdNum+pin
  ```

  The variable **`brdNum`** is the board address (the default address is 7, as explained in Chapter 2) and the variable **`pin`** is the output being set (software pin assignment **`0`**–**`31`**).

  `state` is 0 if the corresponding output is to be disabled or turned "OFF," **`state`** to 1 if the corresponding output is to be enabled or turned "ON."

  RETURN VALUE:

  - 0 if the output is within range.
  - Sets the flag **`EIO_NODEV`** in **`eioErrorCode`** and returns a -1 if and only if the channel does not exist (that is, if **`eioAddr`** is greater than **`31`**).

Program 5-2 demonstrates how to set the status of a digital output.

Program 5-2. Output Demonstration Program

```
#use vdriver.lib
#use vdriver.lib
#use eziocmmn.lib
#use eziopbdv.lib
#use ezioplc.lib
// uncomment #use ezioplc.lib line below for
// PK2100(Rugged Giant), PK2200(Little Star),
// BL1200(Little PLC) and BL1600(Little G)
// #use ezioplc.lib
// uncomment #use eziomgpl.lib line below for
// BL1400(Micro-G) or BL1500(Micro-G2)
// #use eziomgpl.lib
char TITLE[] = {"XP81xx Digital Output"};
main() {
   int channum;
   int i;
   VdInit();
   printf("%s\n\n", TITLE);
   eioResetPlcBus();      // reset the PLCBus
   eioPlcRstWait();       // delay ensures the
                          // PLCBus boards reset
                          // locate all possible
                          // jumper-set addresses
                          // from 0-7 and display
                          // status
   for (i = 0; i <= 7; ++i) {
      if (plcXP81In(i*32)==-1) { //read to locate board
         printf("Board %d is not located\n\n",i);
      }
      else {
         printf("Board %d is located\n",i);
                          // enable each chan from 0-31
         printf("Enabling all 32 positions\n");
         or (channum = 0; channum <= 31; ++channum)
         plcXP81Out(i*32+channum,1); //wr state to out chan
         printf("Press a key to continue...\n");
         while (!kbhit());
         getchar();       // disable each chan from 0-31
         printf("Disabling all 32 positions\n");
         for (channum = 0; channum <= 31; ++channum)
         plcXP81Out(i*32+channum,0); //wr state to out chan
         printf("Press a key to continue...\n");
         while (!kbhit());
         getchar();
      }
      printf("\n");
   }
}
```

Advanced Programming

While the functions described in the last four pages are easy to use to read and set input/output channels, they may not be able to meet the requirements of critical, real-time applications. This section discusses how to access the inputs/outputs on the XP8100 Series expansion boards more efficiently. To this, the reader must be familiar with binary arithmetic, C programming, and low-level PLCBus operations.

Functions for PLCBus Cycles, Reading and Writing

The PLCBus functions described in this section for the XP8100 Series expansion boards will make a program more abstract and portable. Dynamic C's **inport** and **outport** statements or **in** and **out** assembly instructions may still be used for controllers that support the PLCBus directly. However, the expansion boards still have to be reset and a delay has to be provided to ensure that all resets have occurred.

The following functions are located in **EZIOPBDV.LIB**.

- **unsigned _eioPlcXP81Addr(char BrdAddr)**

  Converts the logical address into a 12-bit physical address.

  PARAMETER: **BrdAddr** is the jumper-configured board address, which ranges from 0 to 7. The logical address of the XP8100 Series expansion boards is **0000 0pqr**, where **pqr** is the binary representation for a board address of 0 to 7.

  The function converts the logical address into a 12-bit physical address, **r000 01pq 0001**.

  RETURN VALUE: The bit-mingled XP8100 Series physical address.

The following functions are located in **EZIOPLC.LIB** and can be used to simplify the multiple writes and reads on the PLCBus.

- **void eioPlcAdr12(unsigned addr)**

  Specifies an address on the PLCBus using the BUSADR0, BUSADR1, and BUSADR2 cycles. **addr** is broken into three nibbles, and one nibble is written during each BUSADRx cycle, with BUSADR0 the first bus cycle.

  addr contains the PLCBus cycle addresses. BUSADR0 contains the least significant four bits as shown below.

addr:	**0000**	**rxyz**	**01pq**	**0001**
BUSxxxx:.		ADR2	ADR1	ADR0

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

  Writes to PLCBus register BUSADR2.

  `addr` is the most significant four bits, **`rxyz`**. Here **`xyz`** is represented as a group number. This function writes **`rxyz`** only in register BUSADR2. Table 5-4 on the next page lists the **`rxyz`** addresses.

- **`char _eioReadD0()`**

  This function reads the BUSRD0 register and returns the four data bits D3–D0 read off the PLCBus.

- **`char _eioReadD1()`**

  This function reads the BUSRD1 register and returns the four data bits D3–D0 read off the PLCBus.

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

  This function writes to the BUSWR register.

  `ch` is the four data bits, D3–D0, written in the BUSWR register.

Address Calculation

Addressing an XP8100 Series expansion board first involves explicitly determining each bit of the board's address and then arranging those bits in a particular order. This form of addressing is more complex than the simple formula presented in the preceding section.

Let **`p`**, **`q`**, and **`r`** represent the most significant to least significant bits of the jumper-set address of an XP8100 Series expansion board. The "logical" address of each board in binary notation is then **`0000 0000 0pqr`**. The default address of any XP8100 Series expansion board is 7, as explained in Chapter 2.

The actual address that is passed to advanced PLCBus functions, however, must be rearranged to a physical address, **`rxyz 01pq 0001`**, where **`xyz`** corresponds to either the board identification address or a group number for the input/output data. The physical address is passed during a PLCBus cycle by presenting the least-significant nibble, **`0001`**, to the BUSADR0 register, the middle nibble **`01pq`** to the BUSADR1 register, and the most-significant nibble **rxyz** to the BUSADR2 register. Table 5-4 on the next page lists the **`rxyz`** addresses.

For convenience, the function **`_eioPlcXP81Addr`** described in the previous section is available to transform the logical address into the physical address **`r000 01pq 0001`** required by the PLCBus.

Table 5-4 lists the software input/output group numbers and the corresponding register BUSADR2 values to use when accessing the XP8100's input channels via PLCBus registers BUSRD0 and BUSRD1. The bit positions of all 32 channels are also included. The input/output channels are shown as channels 00 through 31.

Table 4-4. Software Input Registers

			n (I/O Channels)			
Group Number	**BUSADR2 rxyz**	**PLCBus Register**	**D3**	**D2**	**D1**	**D0**
0	r000	BUSRD0	X	X	X	0
0	r100	BUSRD0	03	02	01	00
		BUSRD1	07	06	05	04
1	r101	BUSRD0	11	10	09	08
		BUSRD1	15	14	13	12
2	r110	BUSRD0	19	18	17	16
		BUSRD1	23	22	21	20
3	r111	BUSRD0	27	26	25	24
		BUSRD1	31	30	29	28

Checking for Presence of XP8100 Using Dynamic C Functions

It is possible to verify whether an XP8100 Series expansion board with a given bus address is actually responding. If the program addresses an XP8100 with the lowest three bits of the highest nibble cleared, then the XP8100 at that address will enter an "ID mode." A correctly identified board in the ID mode responds with a nibble that has the least significant bit cleared.

Use the following procedure and sample program with the Dynamic C functions to check whether a board actually exists on the PLCBus.

1. Calculate the physical PLCBus address of the board using the function

   `_eioPlcXP81Addr`.

   The address will automatically be in "ID mode" in the form **`r000 01pq 0001`**. Remember that **`pqr`** is the jumper-configured board address, as explained in Chapter 2.

2. Send the physical address to the PLCBus using the function

   eioPlcAdr12.

3. Read back the nibble D3-D0 using the function

   eioReadD0.

4. Determine whether a board exists on the PLCBus by checking if the least significant bit D0 is cleared or contains a zero. Refer to Table 5-4 to help determine D0.

Program 5-3, **XP81IDX.C**, shows how to detect XP8100 expansion boards connected on the PLCBus using Dynamic C functions. Compile and run this program from the Dynamic C **SAMPLES\PLCBUS** subdirectory.

Program 5-3. Board Detection Program

XP81IDX.C

```
#use vdriver.lib
#use eziocmmn.lib
#use eziopbdv.lib
#use ezioplc.lib
   // for PK2100(Rugged Giant), PK2200(Little Star),
   // BL1200(Little PLC), BL1600(Little G)
   //#use eziomgpl.lib
   // for BL1400(Micro-G) or BL1500(Micro-G2)
main(){
   int i;
   int brdAdr;
   VdInit();              // auto hit watch dog
   eioResetPlcBus();      // reset PLCBus
   eioPlcRstWait(); // delay ensures PLCBus boards reset
// locate all possible jumper-set board addresses from 0 to 7
   for (i = 0; i <= 7; ++i) {
      brdAdr = _eioPlcXP81Addr(i);
                          // convert to PLCBus format
   eioPlcAdr12(brdAdr);  // send board address
   if (_eioReadD0()&1)   // read nibble, mask bit 0
      printf("Board %d is not located\n",i);
   else
      printf("Board %d is located\n",i);
   }
}
```

Checking for Presence of XP8100 Without Using Dynamic C Functions

The following steps may be used to check whether a board is connected to the PLCBus without using Dynamic C functions. The procedure requires accessing the BUSADR0 and BUSADR1 registers during a PLCBus cycle. The procedure essentially checks if a board with a specific address exists.

1. The physical address is always in the form **`rxyz 01pq 0001.`**

   The letters **`pqr`** stand for the board address, which is 0 to 7, in binary notation. The letters **`xyz`** will always be **`000`** for "ID mode." Thus, the string becomes **`0000 0101 0001`** for a board address of 2 since the binary notation for 2 is **`010`**.

2. First write the nibble **`0001`** in the BUSADR0 register during a PLCBus cycle.
3. Write **`01pq`** in the BUSADR1 register.
4. Write **`r000`** in the BUSADR2 register (remember **`xyz`** is always **`000`** for ID mode).
5. Read back the data bits D3–D0 from the BUSRD0 register as **`n`**.
6. Determine if the least significant bit 0 (D0) of **`n`** is cleared. One method of checking bit 0 is to mask **`n`** by performing a "logical and 1" of **`n`**. If the result is zero, the XP8100 board is present.
7. At this point, repeat Steps 3-6 to check for another board only if the BUSADR0 register has not been accessed, and use an address number that is different from the one just checked. Then, change **`pqr`** to identify the next board address.

See Appendix D, "PLCBus States," for detailed states and transitions for the PLCBus. These will be useful for advanced programming.

Reading an Input State Using Dynamic C Functions

The specific XP8100 Series expansion board will determine how many inputs are available, if any.

See the board layouts in Chapter 1, "Overview," to determine which XP8100 Series expansion board is actually being used.

The XP8100 Series of expansion boards is a 5-bit PLCBus device. Each read register can return up to four bits during a cycle. There are two read registers, BUSRD0 and BUSRD1.

The XP8100 Series input channels are organized into four groups, and each group has eight individual channels. Group 0 corresponds to I/O channels 0–7, Group 1 corresponds to channels 8–15, Group 2 corresponds to channels 16–23, and Group 3 corresponds to channels 24–31.

Use the following procedure when reading an input state to first select the proper group of inputs and then read the state of that group's inputs.

1. Use the function **`eioPlcXP81Addr`** to calculate the physical PLCBus board address (**`r000 01pq 0001`**).
2. Use the function **`eioPlcAdr12`** to send the physical address to the PLCBus.
3. Use the function **`eioPlcAdr4`** to send the selected group number **`rxyz`**. Table 5-4 provides the group numbers for the I/O channels.
4. Use either function **`_eioReadD0`** or **`eioReadD1`**, depending on the I/O channel number, to read the nibble D3–D0.
5. Determine if a board exists on the PLCBus by checking if the I/O channel number and corresponding bit position contains a one. Refer to Table 5-4 for corresponding bit positions D3–D0.
6. At this point, the program may do one of the following.
    - Go to Step 1 to select another board
    - Go to Step 3 to select another group on the same board
    - Go to Step 4 to read from the same channel group

The sample program **`XP81INX.C`** demonstrates how to read inputs using the Dynamic C functions supplied. Compile and run this program from the Dynamic C **`SAMPLES\PLCBUS`** subdirectory.

Reading an Input State Without Using Dynamic C Functions

The following steps demonstrate how to operate the PLCBus to read an input without using the supplied Dynamic C functions.

See Appendix D, "PLCBus States," for detailed states and transitions for the PLCBus. These will be useful for advanced programming.

1. Refer to Table 5-4 for the register and channel assignments.
2. The physical address must be in the format

   `rxyz 01pq 0001`.

   The board's address is represented in binary notation as **`pqr`**. The group number is **`xyz`**.
3. First write the nibble **`0001`** in register BUSADR0 during a PLCBus read cycle.
4. Write **`01pq`** in register BUSADR1.
5. Write **`rxyz`** in register BUSADR2.
6. Read back the data bits from the proper register (BUSRD0 and BUSRD1) as **`n`**.
7. Determine if a board exists on the PLCBus by checking if the channel number and corresponding bit position contain a one.
8. The program may now do one of the following if the BUSADR0 read cycle has not been accessed using a **`0001`**.
   - Go to Step 3 to select another board
   - Go to Step 4 to select another group on same board
   - Go to Step 5 to read from the same group.

Controlling Outputs Using Dynamic C Functions

Controlling outputs using Dynamic C functions is similar to the procedure for reading an input's state. The procedure for writing an output also considers the XP8100 Series expansion board to have four groups of input/output channels, with each group having eight channels.

However, the output write procedure deals with only one channel for each PLCBus cycle, unlike the input procedure which handles four input channels during each PLCBus cycle.

Table 5-5 lists which PLCBus address to use when accessing a group of eight channels via the PLCBus BUSWR register.

Table 4-5. Software Output Registers

					BUSWR			
BUSADR2					n			
Group Number	0	1	2	3	Channel Data			state 0=off, 1=on
rxyz	r100	r101	r110	r111	D3	D2	D1	D0
Output Channel	00	08	16	24	0	0	0	0
					0	0	0	1
	01	09	17	25	0	0	1	0
					0	0	1	1
	02	10	18	26	0	1	0	0
					0	1	0	1
	03	11	19	27	0	1	1	0
					0	1	1	1
	04	12	20	28	1	0	0	0
					1	0	0	1
	05	13	21	29	1	0	1	0
					1	0	1	1
	06	14	22	30	1	1	0	0
					1	1	0	1
	07	15	23	31	1	1	1	0
					1	1	1	1

The following procedure first selects the proper group of outputs and then writes the state to the group's output channel.

1. Use the function **_eioPlcXP81Addr** to calculate the physical address **r000 01pq 0001**.
2. Use the function **eioPlcAdr12** to send the physical address to the PLCBus.
3. Use the function **eioPlcAdr4** to send the selected group number **rxyz**. Table 5-5 lists the output registers for the I/O channel group numbers.
4. Use **_eioWriteWr** to send the output state D3–D0. Table 5-5 lists the output registers for the corresponding bit positions D3–D0 and channel numbers.
5. At this point, the program may do one of the following.
   - Go to Step 1 to select another board
   - Go to Step 3 to select another group on same board
   - Go to Step 4 to write to the same group.

Controlling Outputs Without Using Dynamic C Functions

The following steps demonstrate how to perform the PLCBus operation of setting an output without using the supplied Dynamic C functions. Refer to Table 5-5 for the register, channel and group number assignments.

See Appendix D, "PLCBus States," for detailed states and transitions for the PLCBus. These will be useful for advanced programming.

1. The physical address must be in the format **rxyz 01pq 0001**. The board's address is **pqr** and the group number is **xyz**.
2. First write the nibble **0001** in register BUSADR0 during a PLCBus cycle.
3. Write **01pq** in register BUSADR1.
4. Write **rxyz** in register BUSADR2.
5. To turn the output channel on, write the data bits D3–D0 to the BUSWR register. Refer to Table 5-5 to find the corresponding bit positions D3–D0.
6. The program may do one of the following if the BUSADR0 cycle has not been accessed using a **0001**.
   - Go to Step 3 to select another board
   - Go to Step 4 to select another group on the same board
   - Go to Step 5 to write to an output of the same group.

XP8300

CHAPTER 6: OVERVIEW

Chapter 6 gives an overview of the XP8300 relay board and its specific features.

Z-World's XP8300 expansion boards provide a simple way to add relays to a control system built around a Z-World controller. These relay output boards can be connected on the PLCBus in conjunction with other expansion boards. The actuation voltage for the board's relays comes from the controller via the PLCBus port. The XP8300's six relays are high-power relays.

Figure 6-1 illustrates a system of expansion boards mounted on a DIN rail and connected to a controller. Chapter 7, "Getting Started," provides instructions and illustrations for connecting a relay board to a controller's PLCBus port. Appendix D, "Simulated PLCBus Connection," provides instructions and illustrations for connecting relay boards to a specific controller that does not have a PLCBus port.

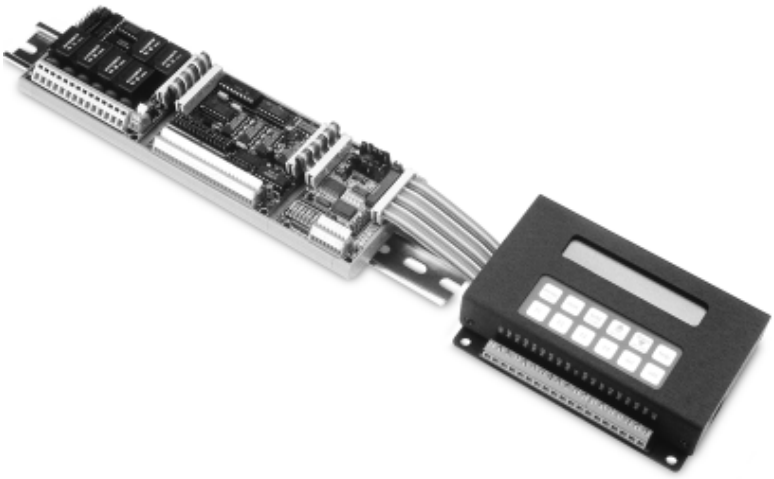


Figure 6-1. Expansion Board System

Features

The XP8300 board has six 24 V high-power relays installed as standard equipment: two are configured as SPDT and four are configured as SPST. All the relays are accessed through screw terminals on headers H1, H2, and H4 to allow easy connections to external devices. Each relay is protected with a 10 A fuse. To help eliminate noise transients, a metal oxide varistor (MOV) and an RC snubber are attached between pin 1 and pin 3 of each relay.

The inputs (pin 1) and normally open output contacts (pin 3) for all relays on an XP8300 board are accessible on headers H1 and H2. The normally closed outputs (pin 4) for relays 4 and 5 are available at header H4.

The XP8300 also has six LEDs that correspond to the six relays. An LED turns on when the corresponding relay's coil is energized. However, an illuminated LED does not verify that the contacts within the relay actually switch.

The XP8310 is a 12 V version of the XP8300.

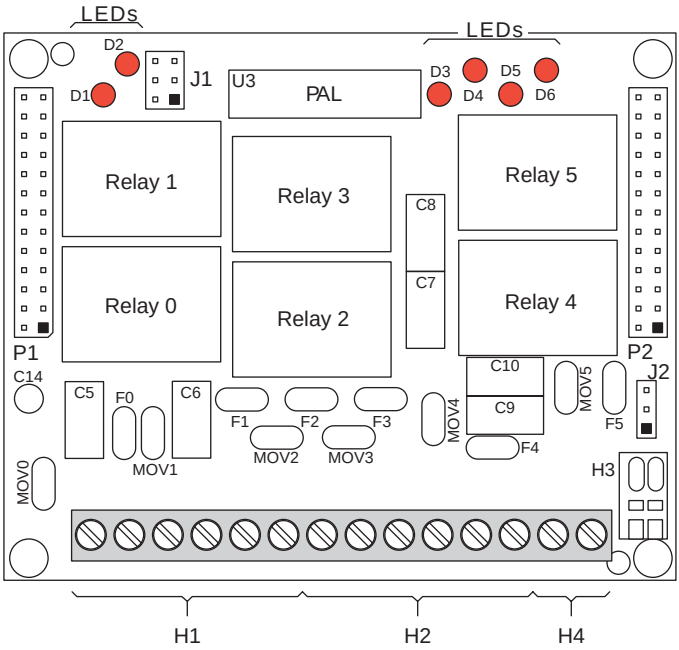


Figure 6-2. XP8300 Relay Expansion Board Layout

Specifications

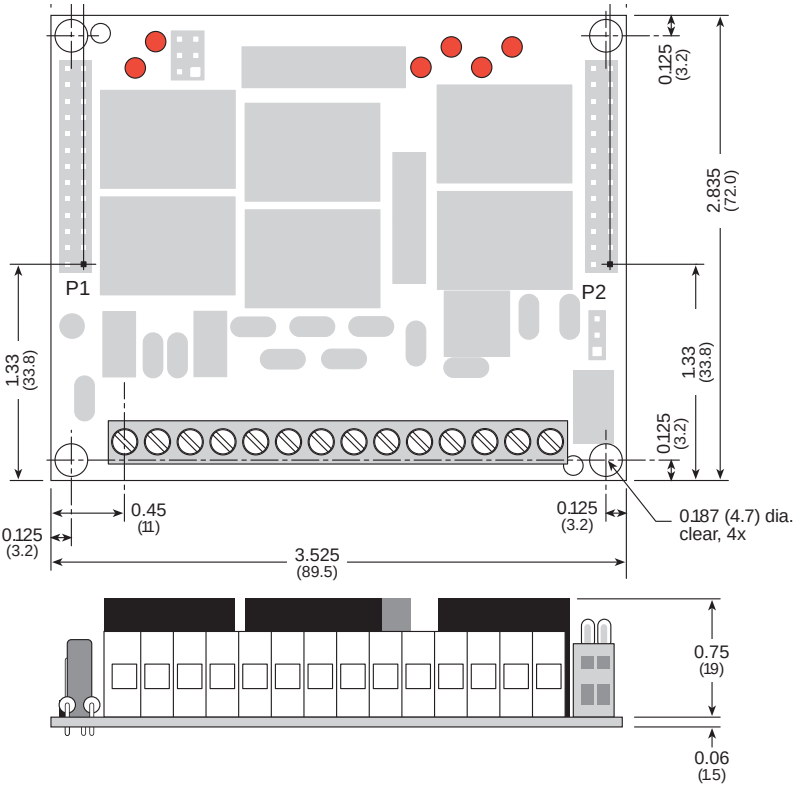


Figure 6-3. XP8300 Dimensions

Table 6-1. XP8300 Specifications

Feature	Specification
Board Size	2.835" × 3.525" × 0.78" (72.0 mm × 89.5 mm × ~20 mm)
Operating Temperature	–40°C to +70°C
Humidity	5% to 95%, noncondensing
Input Voltage and Current	24 V DC, 100 mA
Relays	6 SPDT relays—2 used as SPDT relays and 4 used as SPST relays 6 A at 250 V AC *or* 6 A at 24 V DC

CHAPTER 7: GETTING STARTED

Connecting Expansion Boards to a Z-World Controller

Use the 26-conductor ribbon cable supplied with the expansion board to connect the expansion board to the PLCBus on a Z-World controller. See Figure 7-1. The expansion board's two 26-pin PLCBus connectors, P1 and P2, are used with the ribbon cable. Z-World recommends using the cable supplied to avoid any connection problems.

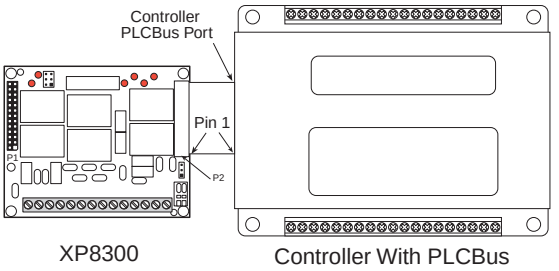


Figure 7-1. Connecting XP8300 Expansion Board to Controller PLCBus

Be sure power to the controller is disconnected before adding any expansion board to the PLCBus.

Follow these steps to connect an expansion board to a Z-World controller.

1. Attach the 26-pin ribbon cable to the expansion board's **P2** PLCBus header.
2. Connect the other end of the ribbon cable to the PLCBus port of the controller.

Be sure pin 1 of the connector cable matches up with pin 1 of both the controller and the expansion board(s).

3. If additional expansion boards are to be added, connect header **P2** on the new board to header **P1** of the board that is already connected. Lay the expansion boards side by side with headers P1 and P2 on adjacent boards close together, and make sure that all expansion boards are facing right side up.

See Appendix C, "Connecting and Mounting Multiple Boards," for more information on connecting multiple expansion boards.

Controllers with simulated PLCBus ports require special expander cables, but are as easily connected. Appendix D, "Simulated PLCBus Connection," gives detailed illustrated instructions for connecting relay boards to controllers without PLCBus ports.

XP8300 Configuration

The XP8300 board holds six high-power relays. Each XP8300 relay has the following specifications:

- Standard coil voltage 24 V DC.
- Contact ratings:
  10 A at 24 V DC
  10 A at 120 V AC
  7 A at 250 V AC resistive
  maximum.

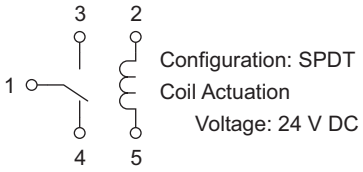


Figure 7-2. Relay Circuit

Pin 1 is the common. Pin 5 goes to a high-voltage/high-current driver on the relay board. Pin 2 is for the actuation voltage. Turning on the driver allows current to flow through the coil, switching on the relay. Pin 3 is the normally open contact. Pin 4 is the normally closed contact.

Each relay is protected by a 10 A fuse on pin 1. To help eliminate transients, a metal oxide varistor (MOV) is attached between pin 1 and pin 3 on each relay. An LED is connected in line with the coil on each relay, and lights up when current passes through the coil.

> Althought the relays are rated at up to 10 A, and are protected with 10 A fuses, the size of the traces on the printed circuit boards limits the current through each relay to 6 A.

Headers H1, H2, and H4 are used to connect external devices to the relays. Pin 1 and pin 3 connections for all relays are provided on headers H1 and H2. In addition, header H4 provides pin 4 connections for relays 4 and 5, allowing relays 4 and 5 to be used as SPDT relays. Relays 0 to 3 do not have their pin 4 available for external connection, and therefore can be used only as SPST relays.

Figure 7-3 illustrates the pinouts for the relay connection pins on headers H1, H2, and H4.

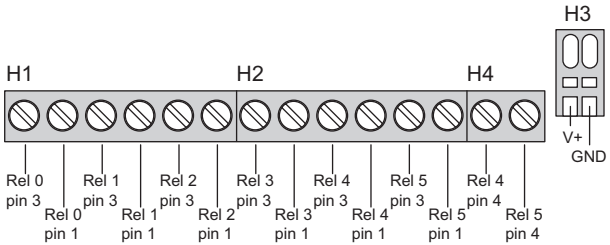


Figure 7-3. Relay Connection Pins

Jumper settings on header J2 determine the actuation voltage for the board's relays. When pins 1–2 are connected, the actuation voltage is supplied by the +24 V line on the PLCBus. When pins 2–3 are connected, the actuation voltage is supplied by the VCC line on the PLCBus.

When no pins on header J2 are connected, an actuation voltage must be supplied by connecting a 24 V power supply at sockets V+ and GND on header H3.

The XP8300 relays require an actuation voltage of 24 V, and the XP8310 relays require an actuation voltage of 12 V. These relays will not work with J2 pins 2–3 connected.

Apply a voltage on header H3 only when header J2 is not jumpered. Applying power to the board when J2 pins 1–2 or 2–3 are connected can damage the relay board and other boards on the bus.

Setting Board Addresses

Jumpers on header J1 (along with PAL encoding) determine the board's bus address. Figure 7-4 shows the jumper settings to set addresses 0–7.

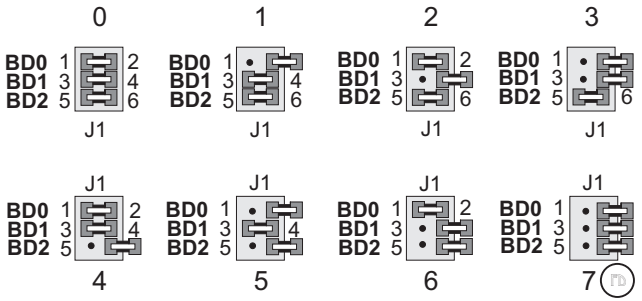


Figure 7-4. J1 Jumper Settings for XP8300 Board PLCBus Addresses

CHAPTER 8: SOFTWARE REFERENCE

Relay Board Addresses

Physical Addresses

Up to 64 addresses are possible on a single PLCBus. The 12-bit address of a particular relay board is determined by two factors: (1) the encoding of the PAL chip installed on the board, and (2) jumper settings on header J1. Since eight different PALs are available and J1 can be set eight different ways, 64 unique addresses are possible.

A 12-bit address can be conveniently placed on the bus using 4-bit addressing. A 12-bit physical address has the following format:

000z 000y pqrx

Jumper bits are defined by the following pin settings:

z = 1 when J1 pins 5–6 are not connected
y = 1 when J1 pins 3–4 are not connected
x = 1 when J1 pins 1–2 are not connected

and

pqr is determined by the PAL.

The physical addresses correspond to the following PLCBus addresses.

000z—BUSADR0
000y—BUSADR1
pqrx—BUSADR2

Logical Addresses

PLCBus expansion boards have "logical addresses." Relay-specific software defines 64 integer board addresses, 0–63. The formula mapping physical address to logical address is defined by the following equation:

logical address = pqr × 8 + zyx

The PAL encoding (pqr) and jumper bits (z, y, x) are defined above.

For example, a relay board that has PAL FPO4550 (pqr = 101) and J1 pins 5 and 6 connected (zyx = 011) would have the following addresses.

physical address: 000z 000y pqrx = 0000 0001 1011 = 0x01B.

logical address: $101_B \times 8 + 011_B = 43 = 0x2B$.

Certain library functions expect a logical relay address.

Software

Dynamic C Libraries

Several Dynamic C function libraries are used with the routines defined in this section. Table 8-1 identifies which libraries are used with specific Z-World controllers.

Table 8-1. Dynamic C Libraries for Controllers

Library	Controller
EZIOCMMN.LIB	All controllers
EZIOPBDV.LIB	All controllers
EZIOTGPL.LIB	BL1000
EZIOLGPL.LIB	BL1100
EZIOMGPL.LIB	BL1400, BL1500
EZIOPLC.LIB	BL1200, BL1600, PK2100, PK2200
EZIOPLC2.LIB	BL1700
EZIOBL17.LIB	BL1700

Before using a library in an application, first include the library name in a **#use** command. For example, to use functions in the library **EZIOPLC.LIB**, insert the following line at the beginning of the program:

```
#use ezioplc.lib
```

How to Use the Relay Boards

1. Send a reset command to all boards on the PLCBus.
2. Place the address of the target board on the PLCBus.
3. Operate the relays.

Reset Boards on PLCBus

These Dynamic C functions are used to initialize the PLCBus. Use these functions in a program before introducing any code to operate the relays.

- **`VdInit()`**

  Initializes the timer mechanism.

  LIBRARY: **`VDRIVER.LIB`**

- **`void plcBusReset()`**

  Resets all expansion boards connected to the PLCBus.

  When using this function, initialize timers with **`VdInit()`** before resetting the PLCBus. All PLCBus devices must reset before performing any subsequent operations.

  LIBRARY: **`EZIOPBDV.LIB`**

- **`void eioPlcRstWait()`**

  Provides a delay long enough for the PLCBus to reset.

  This function provides a delay of 1–2 seconds to ensure devices on the PLCBus reset. Call this function after resetting the PLCBus.

  LIBRARY: **`EZIOPBDV.LIB`**

- **`long int eioErrorCode`**

  Represents a global bit-mapped variable whose flags reflect error occurrences.

  This register for this variable is initially set to 0. If the application tries to access an invalid channel, the flag **`EIO_NODEV`** (the first bit flag) is set in this register. Note that the other bits in **`EIO_NODEV`** deal with networked controllers.

Address Target Board

- **`unsigned _eioPlcRelayAddr( unsigned BrdAddr );`**

  Converts bit pattern **`00000000 00pqrabc`** to **`pqrc 000b 000a`** where **`pqr`** is the PAL number and **`abc`** is the address of the selected board.

  PARAMETERS: The low byte of **`BrdAddr`** should contain the logical address (*8*PAL# + Board#*). The board number is 0–63 (0–7 if only the factory default PAL is used).

  RETURN VALUE: The bit-mingled BUSADR address **`pqrc 000b 000a`** for the XP8300 board.

  LIBRARY: **`EZIOPBDV.LIB`**

Operate Relays

- **`int plcXP83Out( unsigned address, int state );`**

  Energizes a relay on an XP8300 expansion board.

  PARAMETERS: **`address`** is *8*Board# + Relay#*. The board number is 0–63 (0–7 if only the factory default PAL is used). The relay number range is 0–5.

  `state` indicates whether the relay should be energized—the specified relay is energized when **`state`** is non-zero, but is ***not*** energized when **`state`** is zero.

  RETURN VALUE: 0 if the specified XP8300 and relay exist, otherwise –1. If the specified relay/board do not exist, the global variable **`eioErrorCode`** is bit-ored with the constant **`EIO_NODEV`**.

  LIBRARY: **`EZIOPBDV.LIB`**

The **`plcXP83Out`** driver implements other function calls such as **`eioPlcAdr12`**, **`eioPlcAdr4`**, **`eioReadD0`**, **`eioReadD1`**, and **`eioWriteWR`**.

Refer to Appendix A, “PLCBus,” for a description of these other functions.

Advanced Programming

Controlling a Relay

Once a relay's address is placed on the bus (the most recent address on the bus remains in effect), relays can be switched indefinitely. Use the BUSWR bus cycle to place four bits of data on the bus. Table 8-2 shows the relay physical addresses and states.

Table 8-2. Relay Addresses and States

Relay	Data Bits			
	D3	D2	D1	D0
0	0	0	0	0 = relay off
1	0	0	1	1 = relay on
2	0	1	0	
3	0	1	1	
4	1	0	0	
5	1	0	1	
6	1	1	0	
7	1	1	1	

A formula for turning on a relay is

```
relay# << 1 | 1
```

The following code fragments illustrate how to turn on a relay using this formula for a BL1200, PK2200, or PK2100.

```
#define ON  1
#define OFF 0
#define BOARD 0x0301          // board address is
                              // 0x0103
#define REL3 6                // 3 << 1 = 6
set12adr( BOARD );            // select the board
outport( BUSWR, REL3|ON );    // turn relay 3 on
```

(or)

```
write12data( BOARD, REL3|ON );
```

Use the following code for a BL1100 or BL1000.

```
#define ON  1
#define OFF 0
#define BOARD 0x0301          // board address is
                              // 0x0103
#define REL3 3
PBus_Addr( BOARD );           // select the board
PBus4_Write( REL3|ON );       // turn on relay 3
```


PLC_EXP.LIB

The **PLC_EXP.LIB** library supports PLCBus controllers when operating PLCBus expansion boards. This library provides general bus functions and specific functions for the XP8300expansion boards.

There are four groups of functions in this library. Table 8-3 lists the two groups used by relay boards. Analogous functions exist in other libraries.

Table 8-3. PLC_EXP.LIB Groups

Group	Functions
General	**plc_poll_node, Reset_PBus, Reset_PBus_Wait**
Relay	**plc_set_relay**

- **int plc_poll_node(int board)**

  Returns 1 if the board identified by physical address **board** can be found on the PLCBus and 0 if not.

- **void plc_set_relay(int board, int relay, byte state)**

  Switches a relay on an XP8300 board.

  PARAMETERS: **relay** must be from 0–7 (0–5 on an XP8300 board).

  state must be 1 (on) or 0 (off).

  board must be a logical board address (0–63).

- **void Reset_PBus()**
  void rset_pbus_wait()

  The function **Reset_PBus** resets the PLCBus. The function **Reset_PBus_Wait** provides the necessary delay (~450 ms) for the bus to reset.

- **int plcrel_addr(int board)**

  Returns the (nibble-interchanged) bus address for a relay board identified by a logical address (0–63).

PBUS_TG.LIB

The **PBUS_TG.LIB** library allows the BL1000 to operate Z-World's relay expansion boards. The **PBUS_TG.LIB** library does not support any other expansion boards.

The functions in this library are identical (except for internal details) to those in the **PBUS_LG.LIB** library.

PBUS_LG.LIB

The **PBUS_LG.LIB** library allows the BL1100 to operate Z-World's relay expansion boards. This library does not support any other expansion boards.

There are three groups of functions in this library. The two groups used by relay boards are listed in Table 8-4. Analogous functions exist in other libraries. For example, **reset_pbus** in **PLC_EXP.LIB** is used with controllers with a PLCBus and performs the same function as **Reset_PBus** in this library, which is used with the BL1100 and the BL1300.

Table 8-4. PBUS_LG.LIB Groups

Group	Functions
General	**PBus12_Addr, PBus4_Write, PBus4_Read0, PBus4_Read1, PBus4_ReadSp, Reset_PBus, Reset_PBus_Wait, Poll_PBus_Node**
Relay	**Relay_Board_Addr, Set_PBus_Relay**

- **void PBus12_Addr(int addr)**

  Places a 12-bit address on the PLCBus, in 4-bit mode. That is, it places three 4-bit nibbles on the bus. The first and third nibbles of **addr** must be interchanged: if the bus address is 0x125, **addr** must be 0x521.

- **int PBus4_Read0()**
  int PBus4_Read1()
  int PBus4_ReadSp()

  Carries out a bus read cycle. These functions correspond to bus cycles BUSRD0, BUSRD1 and BUSSPARE, respectively.

- **void PBus4_Write(byte value)**

  Carries out a BUSWR cycle.

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

  Returns 1 if there is a board at addr on the PLCBus, and 0 if not. The first and third nibbles of **`addr`** must be interchanged: if the bus address is 0x125, **`addr`** must be 0x521.

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

  Returns the (nibble-interchanged) bus address for a relay board identified by a logical address (0-63).

- **`void Reset_Pbus()`**
  `void Reset_Pbus_Wait()`

  The function **`Reset_PBus`** resets the PLCBus. The function **`Reset_PBus_Wait`** provides the necessary delay (~450 ms) for the bus to reset.

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

  Switches a relay on an XP8300 board. **`relay`** must be from 0–7. **`state`** must be 1 (on) or 0 (off). **`board`** must be specified by a logical board address (0–63).

DRIVERS.LIB

The functions **`set12adr`**, **`read12data`**, and **`write12data`** in **`DRIVERS.LIB`** use 12-bit bus addresses. When using the functions in the drivers library, swap the first and third nibbles of the address before passing the address to the function. For example, if the address is 0x125, pass 0x521.

Sample Projects

The following two sample programs activate the relays on one or more XP8300 boards attached to a controller. Two versions of the program are shown: one for PLCBus controllers, and one for the BL1100 and BL1300.

The following instructions tell how to set up a system, write and compile a program, and run a sample program to operate relay boards on a bus.

PLCBus Controllers

Instructions

1. Power up the controller and make sure it is working properly. If you encounter problems, consult the controller's reference manual.
2. Disconnect power from the controller.
3. Using a PLCBus ribbon cable, connect header P2 of the relay board to the PLCBus on the controller. Make sure both boards are right-side up, with their input and output headers facing toward you. If you have additional relay boards, chain them to the first board with PLCBus ribbon cables.
4. Check the jumpers on headers J1 and J2 on the relay boards. With only one board, leave J1 unjumpered. With more than one board, leave J1 unjumpered on the first board and set J1 with a different and unique address on each additional board. On every relay board, connect pins 1–2 on J2. This connection causes each board to draw its relay-actuation voltage from the +24 V provided over the PLCBus by the controller.

When using the standard XP8300 with 24 V relays, the controller must be powered by a 24 V supply or 24 V must be brought in externally in order to actuate the relays reliably.

5. Power up the controller and bring up Dynamic C on your PC. If you encounter problems reestablishing communications between your PC and the controller, consult the controller's reference manual.
6. Open and run the sample program. Refer to the ***Dynamic C Technical Reference*** manual for detailed instructions on running a program.
7. The LEDs on the relay board(s) will begin flashing to indicate the relays are actuating.

Sample Program

The relay board demonstration program can be used to locate all XP8300 expansion boards. The program then loops, activating the relays on each board. For each board, the program concludes with an all-on/all-off sequence. To locate each board, the program polls all 64 possible addresses, then displays the logical address in Dynamic C's **STDIO** window for each board that responds.

```
/*****************************************************
   Relay Board Demo for XP8300 and XP8400
*****************************************************/
#define ON  1
#define OFF 0
main(){
   int board,relay,found,list[64];
   Reset_PBus();    // always do this, first thing
   delay(1000);     // pause 1000ms for reset
                    // Locate relay boards. Build list
                    // and print board IDs
   found=0;
   printf("\nLogical relay addresses found: ");
   for( board=0; board<64; board++ ){
      if( plc_poll_node(plcrel_addr(board)) ){
         list[found++] = board;
         printf(" %d ",board);
         if( found%10 == 0 ) printf("\n");
      }
   }
                    // Activate relays on each board
                    // found
   while( 1 ){      // loop forever
      for( board=0; board<found; board++ ){
         for( relay=0; relay<8; relay++ ){
            plc_set_relay(list[board],relay,ON);
            delay(333);
            plc_set_relay(list[board],relay,OFF);
         }
         for( relay=0; relay<8; relay++ ){
            plc_set_relay(list[board],relay,ON); // all
         }
         delay(750);
         for( relay=0; relay<8; relay++ ){
            plc_set_relay(list[board],relay,OFF);// all
         }
      }
   }
}
delay( int ms ){    // Max delay time = 2375 ms
   unsigned int ival, i, j;
   ival = (int)(ms * 27.30667) + 1;
   for( i=0; i<ival; i++ ) j = j;
}
```

Controllers with Simulated PLCBus

Instructions for BL1000 and BL1100

1. Power up the BL1000 or BL1100 and make sure it is working properly. If you encounter problems, consult the controller's technical reference manual.
2. Disconnect power from the controller.
3. Using the appropriate cable, connect the XP8300 to the PIO port on the controller. See Appendix D, "Simulated PLCBus Connection," for detailed information regarding this cable. With more than one relay board, chain the additional boards to the first one with PLCBus ribbon cables. Make sure all relay boards are positioned with headers facing the same direction.
4. Check header J1 on the relay board(s) for correct jumper setting(s). With only one board, leave J1 unjumpered. With more than one board, leave J1 unjumpered on the first board and set J1 with a different and unique address on each additional board.
5. Make sure that header J2 has no pins connected. Connect a wall transformer or equivalent 24 V direct current power supply to the V+and GND terminals on header H3 (when using XP8300).
6. Power up the controller and bring up Dynamic C on the host PC. If a problem reestablishing communication occurs, consult ***Dynamic C Technical Reference*** manual.
7. Open and run the program. See the ***Dynamic C Technical Reference*** manual for details on opening and running programs.
8. The LEDs on the relay board(s) will begin flashing to indicate that the relays are actuating.

Sample Program for BL1000 and BL1300

The program locates all XP8300 boards attached to the PLCBus. The program then loops, activating the relays on each board. For each board, the program concludes with an all-on/all-off sequence. To locate boards, the program polls all 64 possible addresses. The integer (logical) address of each board that responds is displayed in Dynamic C's **STDIO** window.


```
/*****************************************************
            Relay Board Demo - for BL1100
*****************************************************/
#define ON  1
#define OFF 0
main(){
   int board,relay,found,list[64];
   Reset_PBus();    // always do this, first thing
   Stall(3000);     // pause ~1sec for reset
                    // Locate relay boards. Build list
                    // and print board IDs
   found=0;
   printf("\nLogical relay addresses found: ");
   for( board=0; board<64; board++ ){
     if( Poll_PBus_Node(Relay_Board_Addr(board)) ){
       list[found++] = board;
       printf(" %d ",board);
       if( found%10 == 0 ) printf("\n");
     }
   }                // Activate relays on each board
                    // found
   while( 1 ){      // loop forever
     for( board=0; board<found; board++ ){
       for( relay=0; relay<8; relay++ ){
         Set_PBus_Relay(list[board],relay,ON);
         Stall(1000);
         Set_PBus_Relay(list[board],relay,OFF);
       }
       for( relay=0; relay<8; relay++ ){
         Set_PBus_Relay(list[board],relay,ON); // all
       }
       Stall(2000);
       for( relay=0; relay<8; relay++ ){
         Set_PBus_Relay(list[board],relay,OFF);// all
       }
     }
   }
}
```

XP8500

XP8500

CHAPTER 9: OVERVIEW

Chapter 9 provides an overview and description of the XP8500 analog-to-digital conversion expansion boards.

The XP8500 provides 11 channels of 12-bit analog-to-digital (A/D) conversion, with onboard signal conditioning for four of these channels to match the input voltage range between 0 V and 10 V. Gain and bias resistors may be selected and installed by the user to determine the voltage ranges of the conditioned input signals.

The XP8500 may be operated either in a ratiometric mode (a mode that reduces errors arising from power-supply variations) or in an absolute mode (where an onboard precision voltage reference assures accurate measurements). The printed circuit board has space for optional sensor-excitation resistors.

Each XP8500 has its zero offset and gain for the four conditioned channels stored in an onboard, serial EEPROM. An application can use library functions to access the EEPROM's calibration constants to correct measurements for offset and gain error.

The XP8500 receives its power from the PLCBus +24 V and +5 V. An onboard voltage regulator develops a clean +5 V supply for the board's analog circuitry from the +24 V PLCBus. The same version of the XP8500 works with both +12 V and +24 V controllers.

Like other Z-World expansion boards, the XP8500 boards can be installed in modular plastic circuit-board holders attached to a DIN rail. The XP8500 boards can also be mounted, with plastic standoffs, on any surface that will accept screws. Up to 16 XP8500 boards addresses are possible on a single PLCBus.

For ordering information, call your Z-World Sales Representative at (530) 757-3737.

Specifications

Table 9-1 summarizes the specifications for the XP8500 expansion board.

Table 9-1. XP8500 Specifications

Board Size	2.835" × 2.125" × 0.75" (72 mm × 54 mm × 19 mm)
Operating Temperature Range	-40°C to +70°C
Humidity	5% to 95%, noncondensing
Power (quiescent, no output)	24 V DC, 32 mA
Inputs	Eleven 12-bit analog inputs • 4 channels with signal conditioning • 7 unconditioned channels

Figure 9-1 shows the dimensions of the XP8500 expansion board.

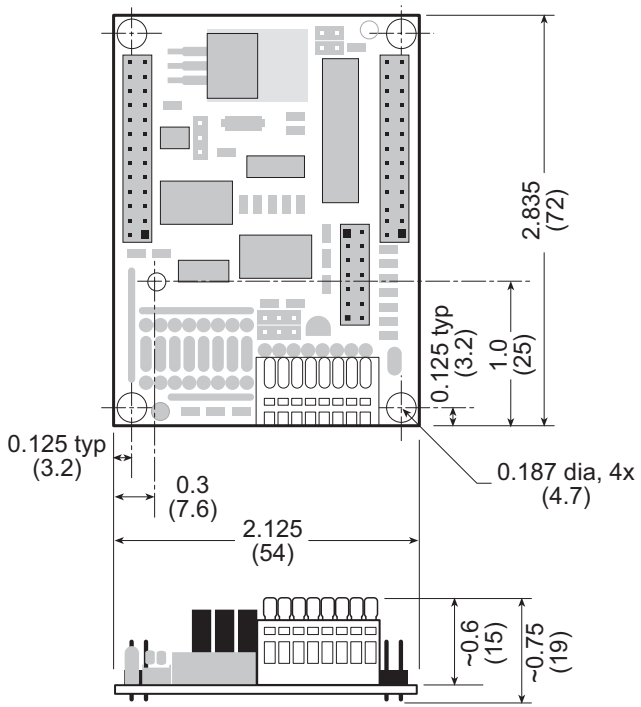


Figure B-1. XP8500 Board Dimensions

CHAPTER 10: GETTING STARTED

Chapter 10 provides instructions for connecting XP8500 expansion boards to a Z-World controller. The following sections are included.

- XP8500 Components
- Connecting Expansion Boards to a Z-World Controller
- Setting Expansion Board Addresses
- Power

XP8500 Components

The XP8500 boards offer eleven channels of 12-bit analog-to-digital conversion. Figure 2-1 illustrates the basic layout and orientation of components, headers, and connectors.

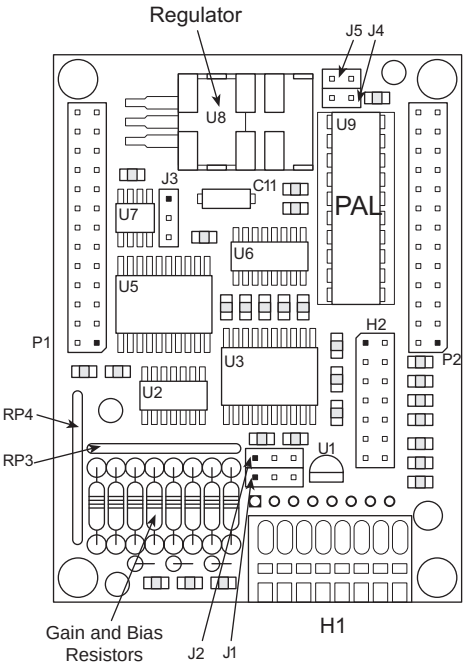


Figure 10-1. XP8500 Board Layout

Connecting Expansion Boards to a Z-World Controller

Use the 26-conductor ribbon cable supplied with an expansion board to connect the expansion board to the PLCBus on a Z-World controller. See Figure 10-2. The expansion board's two 26-pin PLCBus connectors, P1 and P2, are used with the ribbon cable. Z-World recommends using the cable supplied to avoid any connection problems.

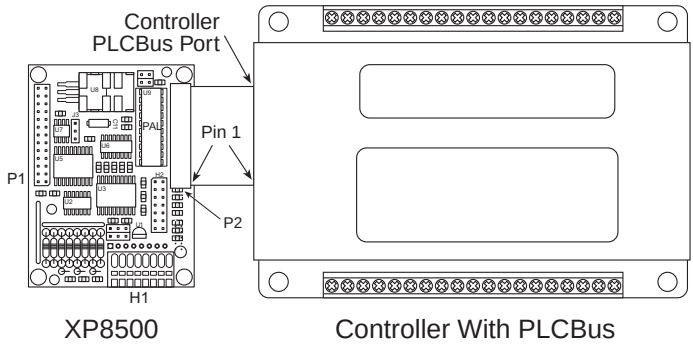


Figure 10-2. Connecting XP8500 Expansion Board to Controller PLCBus

Be sure power to the controller is disconnected before adding any expansion board to the PLCBus.

Follow these steps to connect an expansion board to a Z-World controller.

1. Attach the 26-pin ribbon cable to the expansion board's **P2** PLCBus header.
2. Connect the other end of the ribbon cable to the PLCBus port of the controller.

Be sure pin 1 of the connector cable matches up with pin 1 of both the controller and the expansion board(s).

3. If additional expansion boards are to be added, connect header **P2** on the new board to header **P1** of the board that is already connected. Lay the expansion boards side by side with headers P1/H1 and P2/H2 on adjacent boards close together, and make sure that all expansion boards are facing right side up.

See Appendix C, "Connecting and Mounting Multiple Boards," for more information on connecting multiple expansion boards.

4. Each expansion board comes with a factory-default board address. If more than one expansion board of each type is to be used, be sure to set a unique address for each board.

The following section on "Setting Expansion Board Addresses," and Chapter 4, "Software Reference," provide details on how to set and use expansion board addresses.

5. Power may be applied to the controller once the controller and the expansion boards are properly connected using the PLCBus ribbon cable.

See Appendix D, "Simulated PLCBus Connection," for details on the special connections that enable XP8500 expansion boards to be used with BL1400 and BL1500 controllers.

Setting Expansion Board Addresses

Z-World has established an addressing scheme for the PLCBus on its controllers to allow multiple expansion boards to be connected to a controller.

Remember that each expansion board must have a unique PLCBus address if multiple boards are to be connected. If two boards have the same address, communication problems will occur that may go undetected by the controller.

XP8500 Addresses

XP8600 expansion boards are shipped from the factory with no pins on header J4 or J5 connected. Four different PALs are available. There are four different ways to configure the pair of pins on header J4 and J5, and so up to 16 XP8500s may be addressed individually over a single PLCBus.

See Chapter 4, "Software Reference," for further details on how to determine the physical address for XP8500 expansion boards based on whether the pins on header J4 or header J5 are connected.

Power

Z-World's expansion boards receive power from the controller over the +24 V line of the PLCBus. An onboard regulator converts this to the +5 V reference used by the XP8500. The XP8500 draws 32 mA at +24 V.

The XP8500 may be used with +12 V controllers without having any modifications.

CHAPTER 11: I/O CONFIGURATIONS

Chapter 11 describes the built-in flexibility of the XP8500 expansion boards, and describes how to configure the available inputs/outputs. The following sections are included.

- XP8500 Pin Assignments
- Operating Modes
- Using A/D Converter Boards
- How to Set Up an XP8500
- Selecting Gain and Bias Resistors

XP8500 Pin Assignments

The XP8500's eleven 12-bit analog-to-digital converter channels are accessed through Wago connector H1 (conditioned channels CH0–CH3) and header H2 (unconditioned channels AIN4–AIN10), as shown in Figure 11-1. The bias voltage set by J1, VREF+, is available on header H2, and +5 V (analog) is available on both Wago connector H1 and header H2.

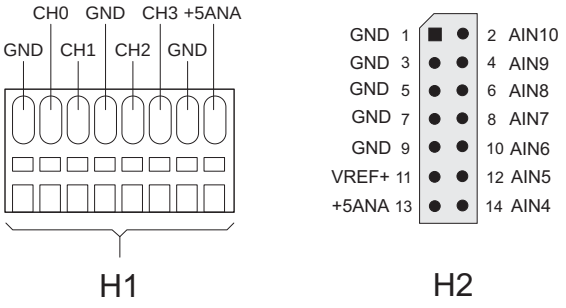


Figure 11-1. XP8500 Pin Assignments

Operating Modes

The XP8500 operates in an absolute mode (as configured in the factory), or in a ratiometric mode. Jumpers J1 and J2 configure the XP8500 to operate in either the absolute or ratiometric mode. J1 selects the reference voltage supplied to the op-amps bias networks, and J2 selects the reference voltage supplied to the A/D converter chip. Figure 11-2 summarizes the jumper connections.

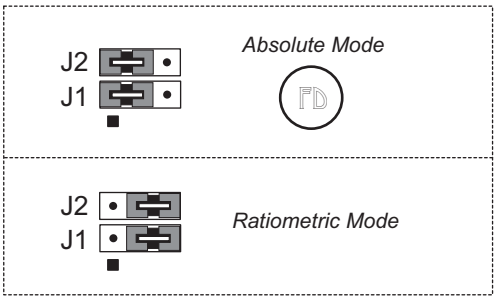


Figure 11-2. XP8500 Jumper Settings for Absolute or Ratiometric Modes

Jumpers pins 1–2 of both headers J1 and J2 are used to select the absolute-conversion mode where the input signal is compared against an accurate fixed voltage reference. With this setting, 2.5 V from the precision voltage reference goes to both the A/D converter chip and to the op-amp bias networks.

Jumper pins 2–3 on both headers J1 and J2 are used to select the ratiometric conversion mode where both the voltage reference and the input will fluctuate with fluctuations in power because they both use the same power source. With this setting, a voltage divider derives 2.5 V from the analog +5 V supply for the A/D converter chip, and the analog +5 V now goes to the op-amp bias networks. (The 2.5 V from the voltage divider cannot power the op-amp bias networks directly because it is not a low-impedance source and the op-amp bias networks would put too large a load on the divider.)

Using Analog-to-Digital Converter Boards

These steps summarize how to use the A/D converter boards.

1. Send a reset command to all boards on the PLCBus.
2. Place the address of the A/D converter board on the PLCBus.
3. Read an input channel, allowing time for the multiplexer to settle and for the digital output to be determined.
4. Allow the controller to use the digital information to calculate a meaningful value for the quantity measured.
5. Use the data to control relays, switches, or other devices with the controller.

These steps rely on software drivers in Dynamic C function libraries. Use **DRIVERS.LIB** and **PLC_EXP.LIB** for controllers with a PLCBus port. The XP8500 will also work with BL1400 and BL1500 controllers.

How to Set Up An XP8500

Conditioned Inputs (CH0–CH3)

Signals from devices connected to a conditioned input channel on H1 go to an inverting input on one of the four op-amps in U2, as shown in Figure 3-3. User-selectable precision resistors R1 through R8 (R_g and R_{bias}) set the gain and bias voltages of the op-amps to match the voltage range of the input to the fixed 2.5 V range of the A/D converter chip.

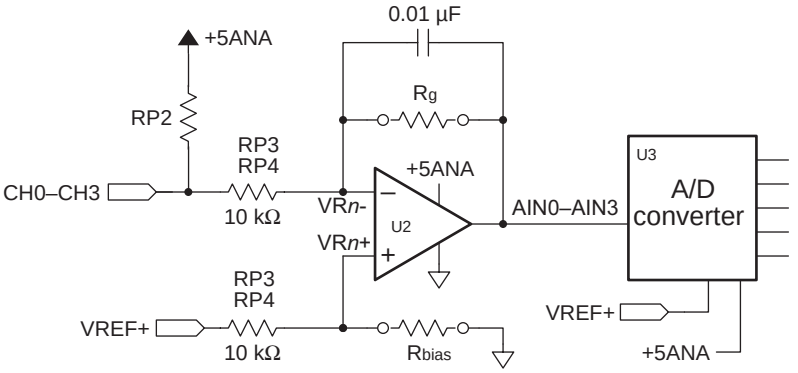


Figure 11-3. Schematic of XP8500 Signal Conditioning

The 10 kΩ input resistors, RP3 or RP4, are fixed; 0.01 µF feedback capacitors roll off the high-frequency response of the op-amps to attenuate noise. Equation (11-1) gives the 3 dB corner frequency.

$$f_{3db} = \frac{1}{2\pi \times R_g \times 0.01\,\mu F} \qquad (11\text{-}1)$$

For the factory default, where the gain is 0.25 using R_g = 2370 Ω, the 3 dB corner frequency is 6715 Hz.

Strip sockets accommodate resistors R1–R8, as shown in Figure 11-4. The factory-installed gain resitors and bias resistors are 2370 Ω and 39.2 kΩ, respectively, and provide a range of 0 V to 10 V for the inputs to be conditioned.

Z-World offers the XP8500 with customer-specified surface-mounted gain and bias resistors installed at R1–R8. For ordering information, call your Z-World Sales Representative at (530) 757-3737.

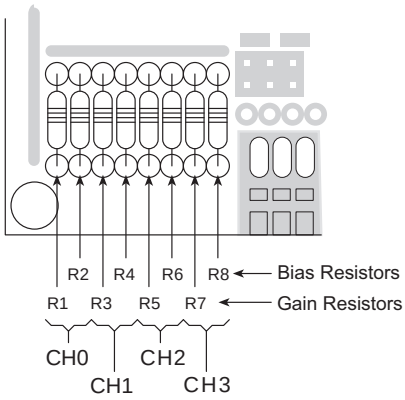


Figure 11-4. Location of XP8500 Gain and Bias Resistors

Table 11-1 provides values for the gain and bias resistors for a range of input voltages. The section on "Selecting Gain and Bias Resistors" at the end of this chapter provides a detailed explanation on how to calculate these values for a particular range of input voltages.

Table 11-1. Gain and Bias Resistors for a Selected Range of Input Voltages

Input Range (V)	Gain	R_g (kΩ)	R_{bias} (kΩ)	
			Absolute Mode	Ratiometric Mode
-10.0 to +10.0	0.125	1.18	8.06	2.87
-5.0 to +5.0	0.250	2.37	6.65	2.49
-2.5 to +2.5	0.500	4.75	4.99	2.00
-2.0 to +2.0	0.625	5.90	4.53	1.82
-1.0 to +1.0	1.250	11.8	2.87	1.27
-0.5 to +0.5	2.500	23.7	1.69	0.787
-0.25 to +0.25	5.000	47.5	0.931	0.442
-0.10 to +0.10	12.500	118	0.392	0.196
0 to + 10.0*	**0.250***	**2.37***	**39.2***	6.49
0 to +5.0	0.500	4.75	20.0	4.99
0 to +2.5	1.000	9.53	10.0	3.32
0 to +1.0	2.500	23.2	4.02	1.69

* These are the factory defaults.

Excitation Resistors

Some transducers, such as thermistors, require an excitation voltage, particularly in some ratiometric applications. These excitation voltages are set using excitation resistors in the RP2 sockets, as shown in Figure 11-5. Either a resistor pack or individual resistors may be used.

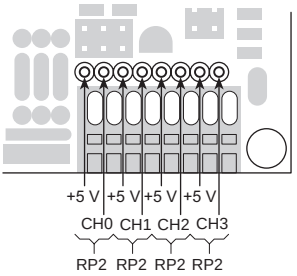


Figure 11-5. Optional XP8500 Excitation Resistors

EEPROM

The jumpers on header J3 write-protect the calibration contents stored in the upper half of the EEPROM—the lower half cannot be write-protected. Figure 11-6 shows the jumper settings for the EEPROM.

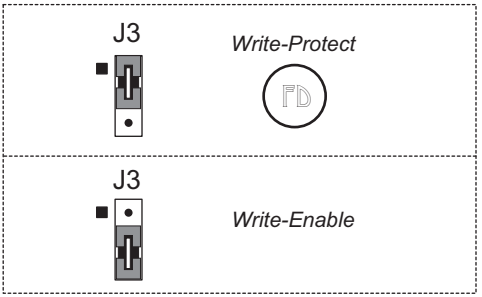


Figure 11-6. XP8500 EEPROM Jumper Settings for Header J3

See Chapter 12, “Software Reference,” for details on how to read and write the EEPROM contents.

Unconditioned Inputs (AIN4–AIN10)

The seven unconditioned input channels, AIN4–AIN10, use 10 kΩ pulldown resistors at R9–R15 as shown in Figure 11-7 to keep the inputs from floating when they are not being used.

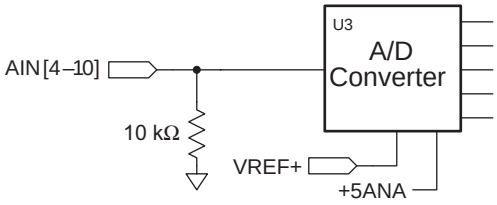


Figure 11-7. Schematic of XP8500 Unconditioned Inputs

These channels are accessed with software by inserting the desired channel number in the library functions that control the XP8500. These channels are located on header H2. For optimum results, drive these channels with low-impedance (< 50 Ω) voltage sources such as LM660 op-amps. High-impedance signal sources are susceptible to coupled noise and will become distorted when loaded by the 10 kΩ pulldown resistors. In addition, only a low-impedance source can charge the sampling capacitors accurately within the A/D converter. When designing the signal sources to drive the extra channels, be sure to consider whether the op-amps can handle the capacitance of the cable used to connect them to header H2.

Internal Test Voltages

In addition to the 11 external input channels of the A/D converter chip, three additional internal channels exist to measure reference points within the chip. The A/D converter compares its internal nodes to REF+ and REF- so the conversions yield either all 1s or all 0s. These channels are accessed using ordinary library routines by specifying the appropriate channel address when calling the functions.

See Chapter 12, “Software Reference,” for further details on Channels 11, 12, 13, and 14.

Power-Down Mode

If Channel 14 on the A/D converter chip is called by the software, the chip enters a power-down mode in which all circuits in the chip go into a low-current, standby mode. The chip also goes into the power-down mode when it is first powered on and before the first A/D conversion. The chip remains in the power-down mode until a channel other than Channel 14 is

selected. The normal operating current of the A/D converter chip is 1 mA to 2.5 mA. This consumption drops to 4 µA to 25 µA when the chip is in a power-down mode. The reduction represents only about 10–20 percent of the XP8500 board's analog supply current and none of its digital supply current

Drift

The AD680JT voltage reference experiences a voltage drift of 10 ppm/°C (typ) to 30 ppm/°C (max). This drift corresponds to 25 mV/°C to 75 mV/°C, or 1.75 mV to 5.25 mV over the temperature range of 0°C to 70°C.

The LMC660C op-amp has an offset-voltage drift of 1.3 µV/°C (typ), or 91 µV over the temperature range of 0°C to 70°C.

A greater contribution to overall drift arises from differences in the temperature coefficients of the gain and bias resistors, and the fixed 10 kΩ resistors in resistor packs RP3 and RP4. These resistor networks and the one used for the ratiometric voltage divider have a temperature coefficient of 200 ppm/°C. Because the packages are small, the resistors within each package are always at essentially the same temperature and their deviations track closely.

Selecting Gain and Bias Resistors

The section "How to Set Up An XP8500" provided representative values of gain and bias resistors for the XP8500's conditioned channels. This section provides a detailed explanation on how to calculate these values for a particular range of input voltages. Figure 11-8 shows a schematic representation of the signal conditioning for channels CH0–CH3.

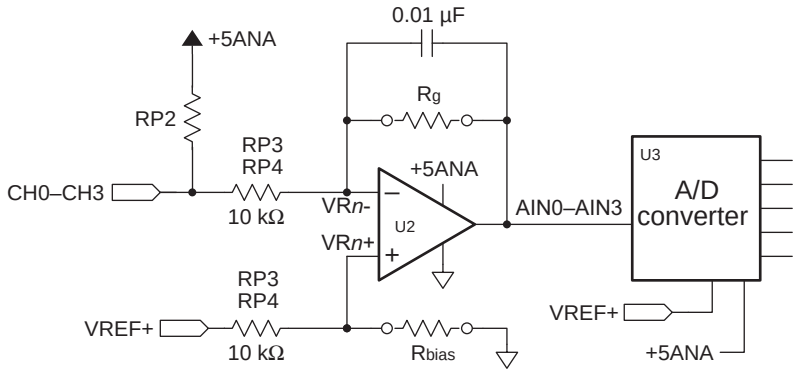


Figure 11-8. Schematic of XP8500 Signal Conditioning

Step 1. Select Gain Resistor

The gain and bias resistors, R1–R8 (R_g and R_{bias} in Figure 11-8), determine the input signal's voltage relative to ground, as well as its range. For example, assume the XP8500 must handle an input signal spanning -5 V to +5 V. First select gain resistor R1 to suit a voltage range of 10 V.

The gain of the amplifier is the ratio of its maximum output-voltage swing to the swing in the software application's maximum input voltage. The 2.5 V input range of the TLC2543 A/D converter chip (U3) limits the LMC660 (U2) op-amps' output swings to 2.5 V. Therefore. Equation (11-2) expresses an amplifier's gain in terms of the range of its input voltage.

$$g = \frac{2.5\ V}{V_{IN_{max}} - V_{IN_{min}}} \qquad (11\text{-}2)$$

where g is the gain, V_{INmax} is the maximum input voltage, and V_{INmin} is the minimum input voltage.

The ratio of the user-specified gain resistor R_g (R_g = R1, R3, R5, or R7) to its associated fixed input resistor (RP4A, RP4C, RP3A, or RP3C) determines an amplifier's gain. Equation (11-3) provides the gain for the configuration shown in Figure 11-8 with the input resistor fixed at 10 kΩ.

$$g = \frac{R_g}{10{,}000\ \Omega} \qquad (11\text{-}3)$$

Given a range of 10 V for the input voltage, Equation (E-1) fixes the amplifier's gain at 0.25. This gain correctly scales the input signal's range to the op-amp's 2.5 V maximum output range. Therefore, R_g must be 2500 Ω.

Step 2. Calculate Bias Resistance

Next, if the op-amp is to servo its output properly around the desired center voltage, the appropriate bias voltage needs to be established at the op-amp's noninverting input. Select the bias resistor, R_{bias}, to position the input-voltage range correctly with respect to ground—in this example, -5 V to +5 V.

The value for R_g has already been selected, and so the maximum input voltage, V_{INmax}, determines the maximum voltage seen at the amplifier's summing junction (inverting input)—circuit nodes VR0– to VR4–. Compute VR0– to VR4– using Equation (11-4).

$$VRn- = V_{IN_{max}} \times \left(\frac{g}{1+g} \right) \tag{11-4}$$

The bias voltage, V_{bias}, must equal its corresponding VRn– for each op-amp. A voltage divider, which consists of a bias resistor, R_{bias} (R_{bias} = R2, R4, R6, or R8), and a fixed 10 kΩ resistor (RP4B, RP4D, RP3B or RP3D), derive this bias voltage, V_{bias} (V_{bias} = VR0+, VR1+, VR2+, or VR3+), from VREF+. Note that VREF+ is *not* necessarily the same as REF+. (REF+ is the positive reference voltage the A/D converter chip uses.)

The XP8500's conversion mode determines which reference voltage the op-amps uses. When the XP8500 operates in the ***absolute mode***, VREF+ is 2.5 V and R_{bias} is

$$R_{bias} = \frac{V_{bias} \times 10{,}000\ \Omega}{2.5\ V - V_{bias}} \quad . \tag{11-5a}$$

When the XP8500 operates in the ***ratiometric mode***, VREF+ is +5 V, and

$$R_{bias} = \frac{V_{bias} \times 10{,}000\ \Omega}{5.0\ V - V_{bias}} \quad . \tag{11-5b}$$

Continuing the example, the gain is 0.25 and V_{INmax} = +5 V; V_{bias} is then 1.0 V using Equation (11-4) R_{bias}, therefore, is 6667 Ω in the absolute mode and 2500 Ω in the ratiometric mode.

Step 3. Choose Best Standard Resistor Values

The calculated resistor values, of course, will not always be available. In these cases, use the nearest standard resistor value. For example, use 6650 Ω (1% resistors) instead of 6667 Ω, or use 6800 Ω (5% resistors).

Step 4. Bracket Input Range

To be sure of measuring signals accurately at the extremes of the range of input voltages, be aware of the interaction between the 10 kΩ fixed resistors, RP3–RP4, and the gain and bias resistors, R1–R8. Ideally, a signal at the minimum input level would be output to the A/D converter's input at the maximum expected value of 2.5 V (remember that U2 is an inverting op-amp).

But real-world resistor values vary within their rated tolerances. Thus, if the fixed input resistor has a resistance lower than its nominal value, and the installed resistors have a resistance slightly higher than their nominal value, the actual input to the A/D converter chip would be greater than 2.5 V. A loss of accuracy then results because the A/D converter input would reach its maximum input value before the true signal input reaches the minimum expected input level.

Similarly, a deviation from nominal values in the bias network could skew the A/D converter's input voltage away from the theoretically computed value. For example, a small positive or negative deviation of the bias voltage arising from variances in the resistor divider would offset the A/D converter's input voltage. This offset would be positive or negative, tracking the deviation's sign, and would be equal to the bias deviation multiplied by the amplifier's gain plus one. Both of these effects could occur in the same circuit.

Step 5. Pick Proper Tolerance

Use care when compensating for discrepancies. For example, if standard 5% resistors are used for R1–R8, the values are spaced approximately 10% apart. If the gain is too high by just a small amount, then going to the next smallest standard 5% value could decrease the gain, and there would be an A/D converter excursion approaching 10%. The same caveat applies to the bias network. Use 1% resistors to have a more precise choice of values.

Figure E-2 shows the result of adjusting the resistor values such that the input to the A/D converter stays within its specified 2.5 V range.

Step 6. Confirm Performance

For critical measurements, always check the setup after installing resistors by measuring test signals at and near the input-voltage limits. See if the U2 op-amp output voltages fall within the A/D converter's input range or if accuracy is lost because of over-excursions at the A/D converter input.

The resistance of the 10 kΩ fixed input resistors can be measured after installing the gain and bias resistors by measuring the voltages at the op-amps' inputs and outputs. Using Channel 0 as an example, ground the CH0 input at pin 2 of Wago connector H1.

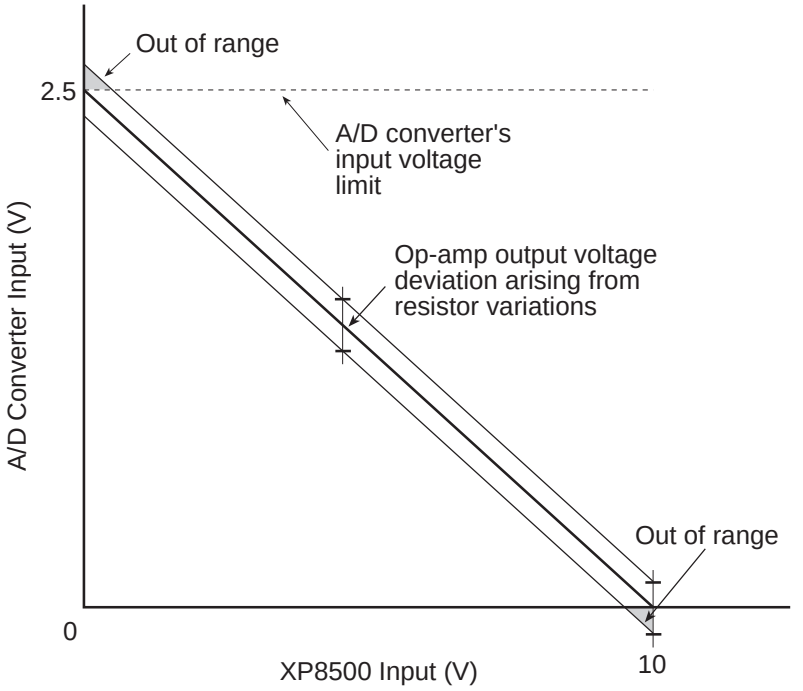


Figure 11-9. Effects on A/D Converter Input from Adjusting Resistor Values

Then measure the voltages at VR0- and at the U2 op-amp output. Because the currents through the input resistor and R_g are essentially identical, the ratio of the voltages across the resistors is equivalent to the ratio of the resistances. Therefore, the gain is

$$g = \frac{V(U2)_{OUT} - VR0-}{VR0-} = \frac{R_g}{R_{IN}} \quad .$$

Again using Channel 0 as an example, measure the voltage VREF and the voltage at VR0+ (see Figure 11-8). Because the current into the op-amp input is negligible, the resistance ratio of the two resistors in the voltage divider alone determines VR0+. The value of the fixed resistor in the divider can then be calculated based on R_{bias} and the value of VR0+.

Step 7. Calibrate the A/D Converter

Regardless of whether the mathematically derived resistance values or the scaled resistance values are found, the inherent component-to-component variations of 5% or 1% resistors can completely swamp the 0.25% resolution of the A/D converter. To achieve the highest accuracy possible, the A/D converter itself must be calibrated.

The software drivers for the A/D converter provide routines to compute calibration coefficients, given two reference points, and then to store the calibration coefficients in a defined location in nonvolatile memory. Each reference point consists of a pair of values: the actual applied test voltage and the raw converted A/D value (a 12-bit integer). Z-World's software will automatically use these calibration coefficients to correct all subsequent A/D readings.

Op-Amp Test Points

The factory-installed gain and bias resistors (R1, R3, R5, R7 = 2370 Ω and R2, R4, R6, R8 = 39.2 kΩ) have a 2% tolerance. These resistors yield a gain of 0.25 for a unipolar input-signal range of 0 V to 10 V.

Figure 11-10 shows some convenient points at which to make voltage measurements of the op-amp.

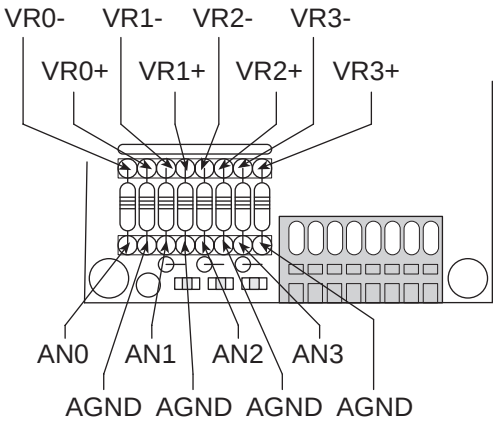


Figure 11-10. XP8500 LMC660 Op-Amp Test Points

Step 8. Recalibrate the XP8500

To recalibrate an XP8500, apply two known test voltages to each channel, **chan,** to be used. Get the converted reading for each test voltage and pass the readings and the test voltages, to the function **adc4_compute** to calculate the conversion coefficients, **zero_offset** and **invgain**, for that channel. **adc4_compute** will automatically store the coefficients in an **adc4coeff** structure (be sure to declare an **adc4coeff** structure for each channel to be calibrated). Lastly, pass the new conversion coefficients to the function **adc4_writecoeff** to store them in the appropriate locations in the XP8500's EEPROM.

The sample program **ADC4SMP3.C** in the Dynamic C **SAMPLES\PLCBUS** subdirectory shows how to calibrate the first four channels of an XP8500 board manually, assuming test voltages of 1.00 V and 9.00 V.

CHAPTER 12: SOFTWARE REFERENCE

Chapter 12 describes the Dynamic C functions used to initialize the XP8500 expansion boards and to control the resulting analog-to-digital conversions. The following major sections are included.

- Expansion Board Addresses
- XP8500 Software
- Advanced XP8500 Programming

Expansion Board Addresses

Up to 16 XP8500s may be addressed individually over a single PLCBus. Each XP8500 has a 12-bit address. The address is determined by the encoding of PAL chip U9 on the board and by the jumper connections on headers J4 and J5.

Four different PALs are available and the jumpers can be set four different ways, giving 16 unique addresses in the form

0000 1100 pqxy

where the PAL determines pq while and the jumper connections on headers J5 and J4 determine x and y, respectively. x and y are zero when their corresponding jumpers are installed on the headers, and are one when the jumpers are removed.

The address can be placed on the bus using 4-bit addressing. The functions **`set12adr`**, **`read12data`**, and **`write12data`** (in **`DRIVERS.LIB`**) use 12-bit bus addresses.

When using these, and certain other functions, swap the first and third nibbles of the address before passing the address to the function. For example, if the address is **`0x125`**, pass **`0x521`**. The function **`eioPlcADC4Addr`** in **`EZIOPBDV.LIB`** is available to do this swap.

- **`unsigned _eioPlcADC4Addr(char BrdAddr)`**

  Swaps bit pattern from 0000 0000 pqxy to pqxy 1100 0000.

  PARAMETERS: **`BrdAddr`** is the logical address (**`4*PAL# + Jumper_number`**) with a bit pattern of 0000 pqxy, where pq is determined by the PAL, and xy is determined by the jumper setting.

XP8500 Software

This section describes a set of simple software functions to use when controlling the XP8100 Series expansion board inputs/outputs.

Dynamic C Libraries

Several Dynamic C function libraries need to be used with the routines defined in this chapter. There are specific libraries designed for certain controllers and there are three common libraries used by all Z-World controllers. Table 12-1 identifies which libraries must be used with particular Z-World controllers.

Table 12-1. Dynamic C Libraries Required by Z-World Controllers

Library	Controller
VDRIVER.LIB	All controllers
EZIOCMMN.LIB	All controllers
EZIOPBDV.LIB	All controllers
EZIOTGPL.LIB	BL1000
EZIOLGPL.LIB	BL1100
EZIOMGPL.LIB	BL1400, BL1500
EZIOPLC.LIB	BL1200, BL1600, PK2100, PK2200, ZB4100
EZIOPLC2.LIB	BL1700
PLC_EXP.LIB	BL1200, BL1600, PK2100, PK2200

The Dynamic C library **EZIOPLC.LIB** replaces **PLC_EXP.LIB**, and is planned to support most Z-World controllers introduced in the future.

Before using one of these libraries in an application, first include the library name in a **#use** command. For example, to use functions in the library **EZIOPLC.LIB**, be sure there is a line at the beginning of the program in the following format.

```
#use ezioplc.lib
```

Initialization Software

These Dynamic C functions are used to initialize the PLCBus. Call these functions before using other expansion board functions.

- **`VdInit()`**

  Initializes the timer mechanism.

  LIBRARY: **`VDRIVER.LIB`**

- **`void eioResetPlcBus()`**

  Resets all expansion boards connected to the PLCBus.

  When using this function, initialize timers with **`VdInit()`** before resetting the PLCBus. All PLCBus devices must reset before performing any subsequent operations.

  LIBRARY: **`EZIOPLC.LIB`**

- **`void eioPlcRstWait()`**

  Provides a delay long enough for the PLCBus to reset.

  This function provides a delay of 1–2 seconds to ensure devices on the PLCBus reset. This function should be called after resetting the PLCBus.

  LIBRARY: **`EZIOPBDV.LIB`**

- **`long int eioErrorCode`**

  The global variable lso needs to be defined. **`eioErrorCode`** represents a global bit-mapped variable whose flags reflect error occurrences.

  This register for this variable is initially set to 0. If the application tries to access an invalid channel, the flag **`EIO_NODEV`** (the first bit flag) is set in this register. The other bits in **`EIO_NODEV`** deal with networked controllers.

XP8500 Drivers

Use the software drivers in this section to interface with the XP8500.

- **`int plcXP85Init( unsigned Addr )`**

  Resets the selected XP8500 and reads back the associated calibration coefficients into an internal array.

  PARAMETER: **`Addr`** is the jumper-selected address of the board (0–7).

  RETURN VALUE: 0 if the reset is successful, -1 if the board cannot be found.

  LIBRARY: **`EZIOPBDV.LIB`**

- **`int plcXP85In( unsigned int address )`**

  Reads an XP8500 A/D converter channel. Note that this function reads back only the raw value—use **`plcXP85InC`** to read back a calibrated value.

  PARAMETER: **`address`** is **`16*board_address + channel_number`**. **`board_address`** ranges from 0–3, depending on the address jumpers, and **`channel_number`** ranges from 0–10, depending on the A/D channel number.

  RETURN VALUE: whole number from 0 to 4095, –1 if the XP8500 board is not found. The global variable **`eioErrorCode`** is bit-ored with **`EIO_NODEV`** if the board is not found.

  LIBRARY: **`EZIOPBDV.LIB`**

- **`float plcXP85InC( unsigned int address )`**

  Reads an XP8500 A/D converter channel and converts the value to a calibrated value using the constants read by **`eioAdc4Init`**. Note that **`eioAdc4Init`** must be called before **`plcXP85InC`**. Use **`plcXP85In`** to read back a raw value.

  PARAMETER: **`address`** is **`16*board_address + channel_number`**. **`board_address`** ranges from 0–3, depending on the address jumpers, and **`channel_number`** ranges from 0–10, depending on the A/D channel number.

  RETURN VALUE: floating-point number that represents the calibrated value read by the A/D channel. The global variable **`eioErrorCode`** is bit-ored with **`EIO_NODEV`** if the board is not found.

  LIBRARY: **`EZIOPBDV.LIB`**

- **`int eioAdcMakeCoeff( struct _eioAdcCalib *cnvrsn, unsigned d1, unsigned d2, float f1, float f2 )`**

  Takes raw values and actual values of two data points, and computes the calibration coefficients. The function assumes the data points are linear.

  PARAMETERS: **`cnvrsn`** is a pointer to a calibration structure that stores the coefficients.

  `d1` is the raw value for the first data point. **`d1`** should be a whole number from 0 to 4095.

  `d2` is the raw value for the second data point. **`d2`** should be a whole number from 0 to 4095.

  `f1` is the actual value for the first data point. **`f1`** is a floating-point number.

  `f2` is the actual value for the second data point. **`f2`** is a floating-point number.

  RETURN VALUE: 0 if the operation is successful, –1 if the calibration coefficients cannot be computed.

  LIBRARY: **`EZIOCMMN.LIB`**

- **`int plcXP85RdCalib( int Addr, struct _eioAdcCalib *pCalib )`**

  Reads the calibration structure of an A/D converter channel on an XP8500.

  PARAMETERS: **`Addr`** is **`16*board_address + channel_number`**. **`board_address`** ranges from 0–3, depending on the address jumpers, and **`channel_number`** ranges from 0–10, depending on the A/D channel number.

  `pCalib` points to a calibration structure used to compute the actual output for a given value.

  RETURN VALUE: 0 if the operation is successful, otherwise a negative number.

  LIBRARY: **`EZIOPBDV.LIB`**

- **int plcXP85WrCalib(int Addr, struct _eioAdcCalib *pCalib)**

  Writes a calibration structure to the EEPROM storage corresponding to a channel on the XP8500.

  PARAMETERS: **Addr** is **16*board_address + channel_number**. **board_address** ranges from 0–3, depending on the address jumpers, and **channel_number** ranges from 0–10, depending on the A/D channel number.

  pCalib points to a calibration structure, which should be initialized by calling **eioAdcMakeCoeff**.

  LIBRARY: **EZIOPBDV.LIB**

Other XP8500 Drivers

The following software drivers from the **PLC_EXP.LIB** library are still supported by Z-World. Z-World recommends using the newer drivers from the **EZIOCMMN.LIB**, **EZIOPBDV.LIB**, and **EZIOPLC.LIB** libraries.

- **int adc4_init(unsigned int board_adr)**

  Determines if an XP8500 board is on the PLCBus. If the function call finds the board, the A/D chip TLC2543 is initialized by enabling its chip-select line. The chip-select line remains enabled until the board powers down.

  PARAMETER: **board_adr** is the physical address of the XP8500 board, defined as 0000 1100 ppxy, where pp is the portion of the board's address set by a particular PAL and xy is the portion of the board's address set with jumpers.

  RETURN VALUE: 1 if the specified XP8500 board is on the PLCBus; 0 if it cannot be found.

- **int adc4_read(unsigned int board_adr, int chan)**

  Enables an analog-input channel, **chan**, and reads the A/D data conversion for the specified channel.

  PARAMETERS: **board_adr** is the physical address of the XP8500 board, defined as 0000 1100 ppxy.

  chan ranges from 0 to 10, corresponding to the board's 11 A/D channels. In addition, passing channel numbers above 10 will access the A/D chip's internal nodes: passing **chan** = 11 will return (VREF+ – VREF–)/2, passing **chan** = 12 will return VREF–, and passing **chan** = 13 will return VREF+. All data defaults to 12 bits unipolar mode, with the most significant bit first. The nominal zero point is 4095 for unipolar input and 2047 for bipolar input.

  RETURN VALUE: whole number from 0 to 4095, –1 if the specified XP8500 board cannot be found.

- **`int adc4_set( unsigned int board_adr, int chan )`**

  Sets the A/D converter chip to the specified channel (**`chan`**).

  PARAMETERS: **`board_adr`** is the physical address of the XP8500 board, defined as 0000 1100 ppxy.

  `chan` ranges from 0 to 10, corresponding to the board's 11 A/D channels. Passing **`chan`** = 11 will return (VREF+ – VREF–)/2, passing **`chan`** = 12 will return VREF–, passing **`chan`** = 13 will return VREF+, and passing **`chan`** = 14 will put the board's A/D chip, a TLC2543, into software power-down mode. All data defaults to 12 bits unipolar mode with MSB first. The returned data's nominal zero point is 4095 for unipolar conversion and 2047 for bipolar conversion.

  RETURN VALUE: whole number from 0 to 4095 from the last A/D conversion (caller should be aware of which A/D channel was set previously); –1 if the specified XP8500 board cannot be found.

Because the A/D converter chip is hardwired to return a converted value while accepting new settings, **`adc4_set`** returns a value converted with the chip's *previous* settings. Therefore, subsequent calls to **`adc4_set`** using the same arguments will return conversions using the *new* settings.

The key to understanding the difference between **`adc4_read`** and **`adc4_set`** is the "pipelined" nature of the A/D converter. By design, shifting a command *into* the A/D converter simultaneously shifts a reading *out.* However, the A/D converter made this shifted-out reading according to the *previous* command's setup.

So to return a correct reading for a single function call, the **`adc4_read`** command shifts a command into the A/D converter, discards the resulting reading, and makes a second read from the now properly set up A/D converter. The faster **`adc4_set`** function simply returns the first reading. Succeeding **`adc4_set`** calls will return proper readings with the same arguments.

- **`int adc4_sample( unsigned int board_adr, int chan, int count, int *buf, unsigned int divider )`**

  Samples data from an A/D **`chan`** at uniform intervals of time.

  PARAMETERS: **`board_adr`** is the physical address of the XP8500 board, defined as 0000 1100 ppxy.

  `chan` ranges from 0 to 10, corresponding to the board's 11 A/D channels. In addition, passing channel numbers above 10 will access the A/D chip's internal nodes: passing **`chan`** = 11 will return (VREF+ – VREF–)/2, passing **`chan`** = 12 will return VREF–, and passing **`chan`** = 13 will return VREF+.

  `count` specifies the number of samples to collect.

  `buf` points to a buffer where the samples will be stored.

  `divider` specifies the sample rate based on the formula

  sample rate = **`sysclock`**/(20 * **`divider`**) , or

  `divider` = **`sysclock`** * (sample period/20) .

  All data default to 12 bits unipolar mode with MSB first. The minimum value for **`divider`** depends on the clock speed, the number of I/O wait states, and the number of memory wait states. The number of states is approximately

  12 * (131 + 4 * IOWait + 38 * Mwait) .

  For example, a 9 MHz clock with 4 I/O wait states and 0 memory wait states has a sample period of approximately 192 µs; for 1 memory wait state, the sample period is approximately 240 µs. For a 6 MHz clock with 4 I/O wait states and 0 memory wait states, the sample period is approximately 290 µs; with 1 memory wait state the sample period becomes approximately 357 µs.

  RETURN VALUE: 0 if the data collection is successful, –1 if the XP8500 board cannot be found, –2 if the sampling rate is too fast. The function will not collect data if the sampling rate is set too fast.

  This function turns off the interrupts for the duration of each sampling period.

- **`float adc4_convert( int data, struct adc4coeff *cnvrsn )`**

  Converts A/D **`data`** read by **`adc4_read( )`** or **`adc4_set ( )`** into voltage equivalent. An **`adc4coeff`** structure pointed to by **`cnvrsn`** stores the conversion constants for this function. The voltage is

  ```
  voltage = cnvrsn->invgain *
    ( cnvrsn->zero_offset – data).
  ```

  RETURN VALUE: voltage value of the A/D data.

- **`int adc4_readcoeff( unsigned int board_adr, int chan, struct adc4coeff *cnvrsn )`**

  Reads the constants for converting A/D data to voltages.

  PARAMETERS: **`board_adr`** is the physical address of the XP8500 board, defined as 0000 1100 ppxy.

  `chan` ranges from 0 to 10, corresponding to the board's 11 A/D channels.

  `cnvrsn` is a pointer to the **`adc4coeff`** structure that stores the constant **`zero_offset`** and the data-to-voltage conversion constant **`invgain`**. The structure stores the constants as 6 continuous bytes in reserved spaces of the XP8500's EEPROM.

  RETURN VALUE: 0 if the constants are read successfully from the EEPROM, –1 if the XP8500 board cannot be found, –2 if a problem occurs while accessing the EEPROM.

- **`int adc4_writecoeff( unsigned int board_adr, int chan, struct adc4coeff *cnvrsn )`**

  Stores the constants for converting A/D data to voltages.

  PARAMETERS: **`board_adr`** is the physical address of the XP8500 board, defined as 0000 1100 ppxy.

  `chan` ranges from 0 to 10, corresponding to the board's 11 A/D channels.

  `cnvrsn` is a pointer to an **`adc4coeff`** structure that stores the constant **`zero_offset`** and the data-to-voltage conversion constant **`invgain`**. The structure stores the constants as 6 continuous bytes in reserved spaces of the XP8500's EEPROM.

  RETURN VALUE: 0 if the constants are stored successfully in the EEPROM, –1 if the XP8500 board cannot be found, -2 if a problem occurs while accessing the EEPROM, –3 if the upper 256 bytes of the EEPROM are write-protected.

- **`int adc4_compute( struct adc4coeff cnvrsn, int data1, float volt1, int data2, float volt2 )`**

  Computes the **`zero_offset`** and **`invgain`** for the **`adc4coeff`** structure pointed to by **`cnvrsn`**. The function computes the constants **`zero_offset`** and **`invgain`** based on A/D readings of two known input voltages to allow input data to be corrected later using the formula

  voltage = **`invgain`** * (**`zero_offset`** – data) .

  `data1` is the raw A/D reading for the known input voltage **`volt1`**.

  `data2` is the raw A/D reading for the known input voltage **`volt2`**.

  RETURN VALUE: 0 if the constants are computed successfully, –1 if the data used resulted in divide by zero while computing the constants.

- **`int adc4_eerd( unsigned int board_adr, int address )`**

  Reads byte data from EEPROM data **`address`**.

  `board_adr` is the physical address of the XP8500 board, defined as 0000 1100 ppxy. Addresses range from 0 to 511 for the 512 bytes of EEPROM memory storage.

  RETURN VALUE: non-negative value for data, –1 if the XP8500 board cannot be found, –2 if a problem occurs while accessing the EEPROM.

- **`int adc4_eewr( unsigned int board_adr, int address, char data )`**

  Writes byte **`data`** to EEPROM data **`address`**.

  `board_adr` is the physical address of the XP8500 board, defined as 0000 1100 ppxy.

  `address` is 0 to 511 for the 512 bytes of EEPROM memory storage. The top 256 bytes can be write-protected using with jumpers on header J3.

  RETURN VALUE: 0 if the data is successfully written to the EEPROM, –1 if the XP8500 board cannot be found, –2 if a problem occurs while accessing the EEPROM, –3 if a write to the top 256 bytes of EEPROM was attempted while the write-protect jumper is connected.

Correcting Readings

The structure **adc4coeff** that holds the constants for correcting readings is defined as follows.

```
struct adc4coeff {
   int zero_offset;
   float invgain;
}
```

This structure must be declared in an application.

The following equation, which the function **adc4_convert** uses, adjusts A/D data from any channel voltage to correct for gain and offset errors.

voltage = **invgain** * (**zero_offset** – A/D data).

The top sixty six bytes (addresses 446 to 511) of the XP8500 board's EEPROM are reserved to store the calibration constants for the board's eleven A/D channels (six bytes per channel).

The factory will calibrate Channels 0 to 3 based on the installed resistors and store the constants in their appropriate locations in the EEPROM.

Channels 4 to 10 will come with nominal calibration constants of:

zero_offset = 0 and **invgain** = -0.0006105

stored in their respective locations in the EEPROM.

Sample Program

The sample program **ADC4SMP1.C** in **PLC_EXP.LIB** reads data from an XP8500 board over the PLCBus. The program reads the first four (conditioned) channels of the XP8500 board, then displays the data showing both the raw A/D readings and their equivalent voltages.

The program converts the raw readings to voltages based on the calibration constants stored in the EEPROM. The XP8500 board stores these calibration constants in the last 66 bytes (6 bytes/channel) of its EEPROM.

Use the following steps to run the sample program.

1. Compile the program by pressing **F3** or by choosing **Compile** from the **COMPILE** menu. Dynamic C compiles and downloads the program into the controller's memory. During compilation, Dynamic C rapidly displays several messages in the compiling window, which is normal.
2. Run the program by pressing **F9** or by choosing **Run** from the **RUN** menu. It is also possible to single-step through the program with **F7** or **F8**.
3. To halt the program, press **<CTRL-Z>**.
4. To restart the program, press **F9**.

ADC4SMP1.C

```
#if (BOARD_TYPE == CPLC_BOARD) ||
   (BOARD_TYPE==L_STAR)
#use cplc.lib              // Program runs on PK2200
                           // and PK2100 controllers
                           // only.
#endif

main(){
   struct adc4coeff adc4conv0;      // Structs needed
   struct adc4coeff adc4conv1;      // only if you
   struct adc4coeff adc4conv2;      // use calibration
   struct adc4coeff adc4conv3;      // constants to
                                    // convert raw A/D
                                    // data to voltages.
   int data0, data1, data2, data3;// Raw data.
   unsigned int adc4_board;         // Brd address.
   int i;

#if (BOARD_TYPE == CPLC_BOARD) ||
   (BOARD_TYPE==L_STAR)
   uplc_init();
#endif
   reset_pbus();           // reset the PLCBus
   reset_pbus_wait();

   if(sysclock() > 0x1e00)
   reset_pbus_wait();      // wait double if the
                           // clock is faster than
                           // 9 MHz

                           // find the first available
                           // XP8500 board on the PLCBus
   for(i=0;i<4;i++) {
      if(adc4_init(0x0c0+i)) {
         adc4_board = 0x0c0 + i;
         break;
      }
   }
   if( i >= 4) {
      printf("No XP8500 Board detected.\n");
      while(1) runwatch();
   }
   printf("XP8500 board %x has been
      detected.\n", adc4_board);
   printf("Reading XP8500 board calibration
      constants...\n");
   adc4_readcoeff(adc4_board, 0, &ADC4conv0);
                              // read cal for chan0
   adc4_readcoeff(adc4_board, 1, &ADC4conv1);
                              // read cal for chan1
```

continued…

```
   adc4_readcoeff(adc4_board, 2, &ADC4conv2);
                              // read cal for chan2
   adc4_readcoeff(adc4_board, 3, &ADC4conv3);
                              // read cal for chan3
   printf("Chan0 Calibration, zero_offset =
      %d, invgain = %f\n", ADC4conv0.zero_offset,
      ADC4conv0.invgain);
   printf("Chan1 Calibration, zero_offset =
      %d, invgain = %f\n", ADC4conv1.zero_offset,
      ADC4conv1.invgain);
   printf("Chan2 Calibration, zero_offset =
      %d, invgain = %f\n", ADC4conv2.zero_offset,
      ADC4conv2.invgain);
   printf("Chan3 Calibration, zero_offset =
      %d,invgain = %f\n", ADC4conv3.zero_offset,
      ADC4conv3.invgain);
   printf("Toggle F4 (DOS only) to make
      keyboard input active as STDIO.\n");
   printf("Hit any key to read A/D data from
      XP8500 board.\n");
   for(;;) {
      while(!kbhit()) runwatch();   // wait for
                                    // key from the PC
   getchar();                       // get the key
      data0 = adc4_read(adc4_board, 0);
                              // read A/D Channel 0
      data1 = adc4_read(adc4_board, 1);
                              // read A/D channel 1
      data2 = adc4_read(adc4_board, 2);
                              // read A/D channel 2
      data3 = adc4_read(adc4_board, 3);
                              // read A/D channel 3
   printf("\nData for ADC4 channels 0-3 !!!\n");
   printf("chan 0 >> %6d, %8.3f volts\n",data0,
      adc4_convert(data0, &ADC4conv0));
   printf("chan 1 >> %6d, %8.3f volts\n", data1,
      adc4_convert(data1, &ADC4conv1));
   printf("chan 2 >> %6d, %8.3f volts\n", data2,
      adc4_convert(data2, &ADC4conv2));
   printf("chan 3 >> %6d, %8.3f volts\n", data3,
      adc4_convert(data3, &ADC4conv3));
   }
}
```

Sample program **ADC4SMP3.C** also shows how to recalibrate an XP8500 channel and how to store the new calibration constants in the EEPROM.

Check the board jumpers, PLCBus connections, and the PC/controller communications if an error message appears.

See the ***Dynamic C Technical Reference*** manual for more detailed instructions.

PLCBus-Level Communication

Dynamic C functions perform the bus-level operations described here. A program controls and communicates with an XP8500 though the PLCBus interface register, a reserved memory location. This global register occupies a single address on the PLCBus. After the program has selected a particular XP8500 (by calling **`set12addr`** with the XP8500's address as an argument), the program may write data to the XP8500's EEPROM or A/D converter chip via BUSWR cycles to the bus interface register. Similarly, the program can fetch EEPROM data or retrieve converted A/D results via BUSRD0 cycles via the bus interface register. The bus interface register allows the control program and a selected XP8500 to exchange only one 4-bit nibble per cycle.

The EEPROM and the A/D converter are both serial I/O devices. Consequently, the IC control lines can be set or cleared only one bit at a time, and only one bit at a time may be read or written from/to the data lines.

During a BUSWR cycle, each 4-bit nibble transmitted via the bus interface register through the PLCBus to an XP8500 board sets or resets a control line according to Table 12-2.

Table 12-2. Effects of Nibbles Passed Over PLCBus to XP8500

Nibble	Function
0000	A/D Clock = 0
0001	A/D Clock = 1
0010	A/D Write Data = 0
0011	A/D Write Data = 1
0100	A/D Chip Select = 0
0101	A/D Chip Select = 1
0110	EEPROM SDA = 0
0111	EEPROM SDA = 1
1000	EEPROM SCL = 0
1001	EEPROM SCL = 1
1010	Not Used
1011	Not Used
1100	Not Used
1101	Not Used
1110	Not Used
1111	Not Used

The control program must input a series of 4-bit nibbles to read a converted value from a selected XP8500's serial A/D converter chip or serial EEPROM. Again, each nibble can carry only one bit of data or control information. Each 4-bit nibble read back from an XP8500 during a BUSRD0 cycle has the following format.

Bit 3: ENDC—A/D end of conversion
1 = conversion cycle is not in process; OK to send/receive data
0 = conversion cycle is in process; do not send or receive data

Bit 2: SDA—EEPROM serial data out

Bit 1: DOUT—A/D serial data out

Bit 0: Board present
0 = selected board actually present
1 = selected board not found.

The standard Dynamic C library functions for the XP8500 will probably suffice for all applications. Refer to the manufacturer's data sheets for the 24C04 EEPROM and the TLC2543 A/D converter if there is a need to write other routines using the BUSWR and BUSRD cycles.

XP8800

XP8800

CHAPTER 13: OVERVIEW

Chapter 13 provides an overview and description of the XP8800 motion control expansion boards.

XP8800 Overview

Z-World's XP8800 expansion board may be attached to a Z-World controller with a PLCBus port. The XP8800 does not have the software drivers to enable it to be used with other Z-World controllers.

The XP8800 controls a single axis of motion. Multiple XP8800s may be connected to provide up to four axes of control. The benefit of the XP8800 is that it can handle motor control operations, leaving the master controller free to perform other tasks.

The onboard motor driver IC (UCN5804) is capable of driving 1 A per phase and motor voltages up to 35 V. The driver automatically generates the sequencing for 1-phase, 2-phase, and half-step operations. The XP8800 includes a 16-bit quadrature decoder/counter (HCTL-2016) that can count at speeds up to 3 MHz.

An XP8810 version of the XP8800 expansion board is available. The XP8810 offers optical isolation for the quadrature and sense inputs.

Note that there is a common ground for the board and the inputs. Therefore the optical isolation is not absolute.

Like other Z-World expansion boards, the XP8800 can be installed in modular plastic circuit board holders attached to a DIN rail. The XP8800 can also be mounted, with plastic standoffs, on any surface that will accept screws.

Features

- Continuous (manual), preset (counted), or origin-seeking modes of operation.
- Switching between high-speed and low-speed operation, with or without acceleration and deceleration.
- "Bidirectional" pulse output modes.
- Sensing of origin, end-limit, and slowdown signals.
- Interrupt generation.
- 13-bit (8,191) step rate resolution, 18-bit (256K) counter.
- User-definable output speed range up to 16 kHz.
- Single-phase, dual-phase, and half-step modes.
- 16-bit quadrature decoder/counter.
- Watchdog reset.
- Optional optical isolation for quadrature and sense inputs.

Specifications

Table 13-1 summarizes the specifications for the XP8800 expansion board.

Table 13-1. XP8800 Series Specifications

Parameter	Specification
Board Size	2.835" × 4.0" × 0.58" (72 mm × 102 mm × 15 mm)
Operating Temperature Range	-40°C to +70°C
Humidity	5% to 95%, noncondensing
Power (quiescent, no output)	40 mA @ 5 V DC
Output	One-axis stepper motor control rated at 35 V • 1.25 A per phase in full-step mode • 1.0 A per phase in half-step mode

Figure 13-1 shows the dimensions of the XP8800 Series expansion boards.

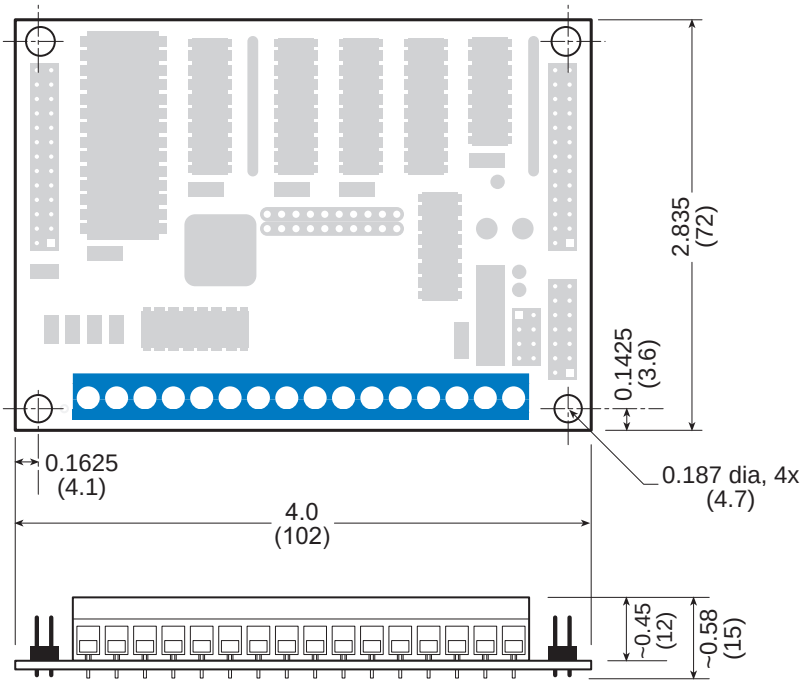


Figure 13-1. XP8800 Board Dimensions

CHAPTER 14: GETTING STARTED

Chapter 14 provides instructions for connecting XP8800 expansion boards to a Z-World controller. The following sections are included.

- XP8800 Series Components
- Connecting Expansion Boards to a Z-World Controller
- Setting Expansion Board Addresses

XP8800 Components

The XP8800 stepper motor control expansion board controls a single axis of motion. Figure 14-1 shows the basic layout and orientation of components, headers, and connectors.

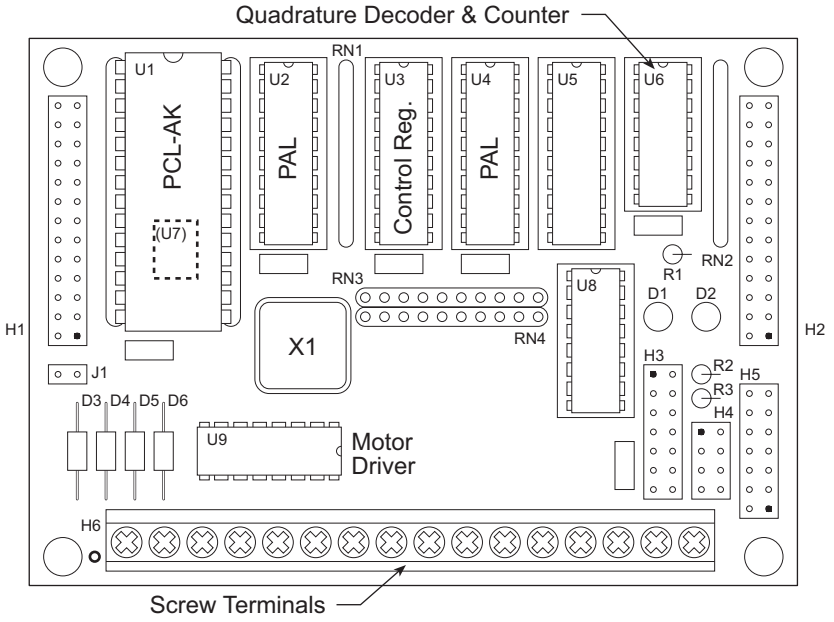


Figure 14-1. XP8800 Board Layout

Connecting Expansion Boards to a Z-World Controller

Use the 26-conductor ribbon cable supplied with an expansion board to connect the expansion board to the PLCBus on a Z-World controller. See Figure 14-2. The expansion board's two 26-pin PLCBus connectors, P1 and P2, are used with the ribbon cable. Z-World recommends using the cable supplied to avoid any connection problems.

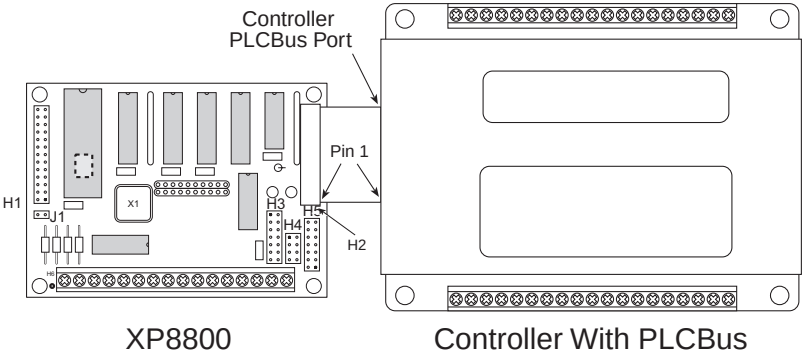


Figure 14-2. Connecting XP8800 Expansion Board to Controller PLCBus

> Be sure power to the controller is disconnected before adding any expansion board to the PLCBus.

Follow these steps to connect an expansion board to a Z-World controller.

1. Attach the 26-pin ribbon cable to the expansion board's **P2** or **H2** PLCBus header.
2. Connect the other end of the ribbon cable to the PLCBus port of the controller.

> Be sure pin 1 of the connector cable matches up with pin 1 of both the controller and the expansion board(s).

3. If additional expansion boards are to be added, connect header **P2/H2** on the new board to header **P1/H1** of the board that is already connected. Lay the expansion boards side by side with headers P1/H1 and P2/H2 on adjacent boards close together, and make sure that all expansion boards are facing right side up.

> See Appendix C, "Connecting and Mounting Multiple Boards," for more information on connecting multiple expansion boards.

4. Each expansion board comes with a factory-default board address. If more than one expansion board of each type is to be used, be sure to set a unique address for each board.

The following section on "Setting Expansion Board Addresses," and Chapter 8, "Software Reference," provide details on how to set and use expansion board addresses.

5. Power may be applied to the controller once the controller and the expansion boards are properly connected using the PLCBus ribbon cable.

Setting Expansion Board Addresses

Z-World has established an addressing scheme for the PLCBus on its controllers to allow multiple expansion boards to be connected to a controller.

Remember that each expansion board must have a unique PLCBus address if multiple boards are to be connected. If two boards have the same address, communication problems will occur that may go undetected by the controller.

XP8800 Addresses

XP8800 expansion boards are shipped from the factory with no pins on header H4 connected. An XP8800 expansion board may be assigned any one of 16 addresses using jumpers on the pins of header H4. The LED at D2 lights up whenever the XP8800 is addressed on the PLCBus.

See Chapter 16, "Software Reference," for further details on how to determine the physical address for XP8800 expansion boards.

Power

Z-World's expansion boards receive power from the controller over the +24 V and VCC lines of the PLCBus. The XP8800 expansion boards use VCC, which is +5 V. The XP8800 draws from 40 mA (quiescent) to a maximum of 105 mA.

CHAPTER 15: I/O CONFIGURATIONS

Chapter 15 describes the built-in flexibility of the XP8800 expansion boards, and describes how to configure the available inputs/outputs. The following sections are included.

- XP8800 Series Pin Assignments
- Using Expansion Boards

XP8800 Pin Assignments

External connections are made to the XP8800 expansion board using H5, a 14-pin header, and H6, a 16-screw terminal block. Figure 15-1 shows the pin assignments.

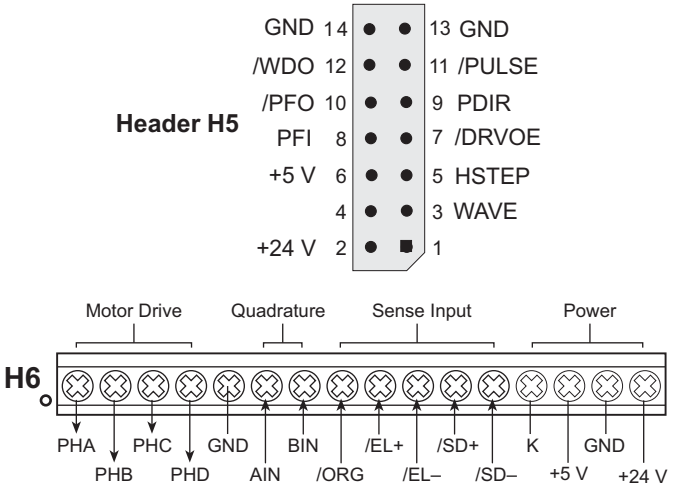


Figure 15-1. XP8800 Output Header H5 and Terminal Block H6

Header H5 Signals

H5 provides connection points for motor control signals, power and ground, power failure, and watchdog signals. The motor control signals are usually used with an amplifier to drive the motor.

/DRVOE—A low signal enables output from the TTL motor driver IC.

GND—is the PLCBus ground, common to the entire system.

Connect the motor power supply ground only to **GND** on the screw terminal block (H6).

HSTEP—Together with the **WAVE** signal, **HSTEP** determines the operation of the TTL motor driver IC: single-phase, two-phase, or half-step.

PDIR—This signal indicates in which direction the TTL motor driver IC is to move. A high level means movement in the + direction. A low level means movement in the – direction.

PFI—is an analog signal input to the power-fail comparator. The **/PFO** line becomes active when **PFI** drops below 1.25 V (±0.05 V).

/PFO—is the open-collector power-failure indicator. **/PFO** goes low when **PFI** goes below 1.25 V (±0.05 V). **/PFO** can be connected to the NMI or interrupt line on the master controller.

/PULSE —A low pulse on this line signals a one-step move to the TTL motor driver IC.

WAVE—Together with the **HSTEP** signal, **WAVE** determines the operation of the TTL motor driver IC: single-phase, two-phase, or half-step.

/WDO—This is the active low, open-collector watchdog output line. When the watchdog is enabled, this line will go low—upon a watchdog timeout—to generate a hard reset at the PCL-AK pulse generator.

+5 V—is the regulated PLCBus +5 V digital power supply. This supply should not be used for motor power, but can be used to power external logic.

+24 V—is the unregulated PLCBus +24 V supply. Though nominally 24 V, this can be anywhere from 9 V to 30 V DC. This supply may be used to drive the motor if the controller's power supply can handle the current requirements.

Screw Terminal Block H6 Signals

PHA, **PHB**, **PHC**, **PHD**—are the open-collector motor control outputs. They connect to the motor phase lines, and can sink up to 1 A, depending on the ambient temperature.

AIN, **BIN**—are the TTL-compatible quadrature-encoded input signals.

/ORG—is the active-low origin pulse input. **/ORG** goes directly to the PCL-AK pulse generator, thereby allowing the PCL-AK to generate pulses until it receives an origin signal. **/ORG** is readable in the PCL-AK (address 0) status bits.

/EL+, **/EL–** —are the active-low end-limit inputs, one for the + direction, the other for the – direction. These signals go directly to PCL-AK pulse generator, where they are typically used to indicate end-of-travel, usually to stop pulse generation. These signals are readable in the PCL-AK (address 0) status bits.

/SD+, **/SD–** —are the active-low "slowdown" inputs, one for the + direction, the other for the – direction. These signals go directly to PCL-AK pulse generator, where they are typically used to force the PCL-AK to decelerate to its slower speed. These signals are readable in the PCL-AK (address 3) status bits.

K—is protection for the driver chip. **K** is connected to the motor control voltage source through protective diodes.

> Be sure to connect **K** to the motor's voltage source. Damage can occur or performance can degrade if this connection is not made.

+5 V—is the regulated PLCBus +5 V digital power supply. This supply should not be used for motor power, but can be used to power external logic.

+24 V—is the unregulated PLCBus +24 V supply. Though nominally 24 V, this can be anywhere from 9 V to 30 V DC. This supply may be used to drive the motor if the controller's power supply can handle the current requirements.

GND—is the PLCBus ground, common to the entire system. The motor's power supply ground should be connected here only. There are two **GND** connections on H6.

Sample XP8800 Connections

Figure 15-2 shows an example of a stepper motor connected to an XP8800 expansion board.

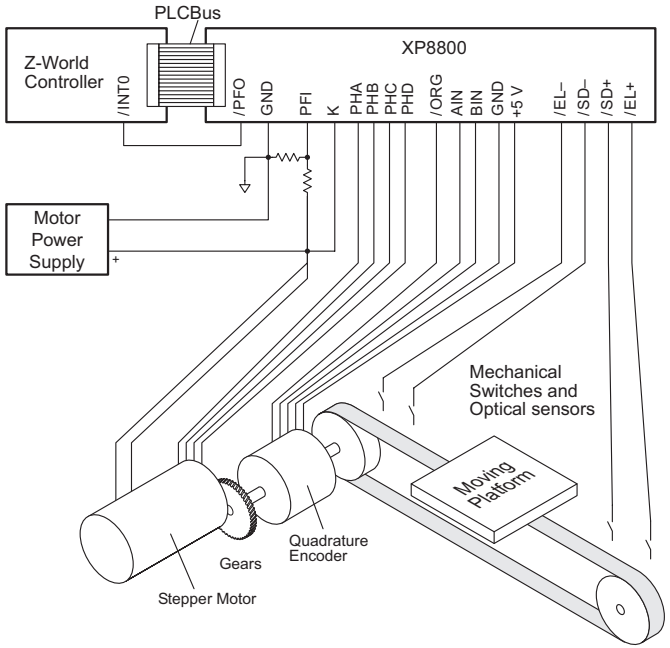


Figure 15-2. Sample Stepper Motor Connection to XP8800

Optional Optical Isolation

The quadrature and sense inputs (AIN, BIN, /ORG, /EL+, /EL-, /SD+, and /SD-) may be optically isolated, as shown in Figure 15-3. The XP8810 version of the XP8800 expansion board features this optical isolation.

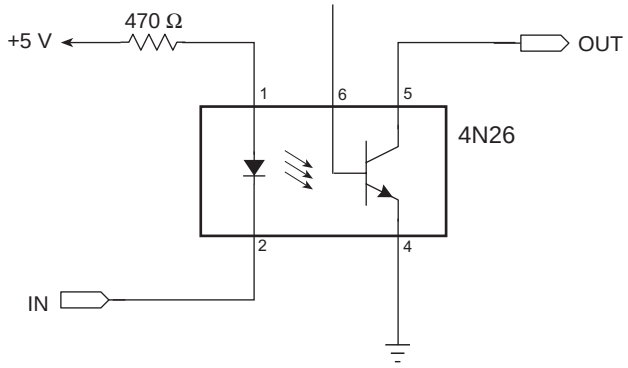


Figure 15-3. XP8810 Optical Isolation Circuit

Note that there is a common ground for the board and the inputs so that the optical isolation is not absolute.

Using Expansion Boards

The following steps summarize how to use the XP8800 boards.

1. Send a reset command to the PLCBus.
2. Place the address of the XP8800 registers on the PLCBus. The address will actually be the address of one of the components, the PCL-AK pulse generator, or the quadrature decoder/counter.
3. Operate the XP8800. The following operations are the ones done most frequently.
   - Set XP8800 control register.
   - Issue command to PCL-AK pulse generator.
   - Set PCL-AK parameters or read PCL-AK registers or status.
   - Reset the quadrature counter or read its value.
   - Wait for interrupt requests.
4. Once the XP8800 operation is done, issue a soft reset to the PCL-AK pulse generator.

The Dynamic C **`STEP.LIB`** library handles the details of operating the XP8800.

Resetting XP8800 Expansion Boards

There are many ways to reset the XP8800 and its components.

1. Power-Up Reset

On power-up, both the PCL-AK pulse generator chip and the quadrature decoder/counter undergo a hardware reset.

The control register powers up to an unknown state, making it necessary for the application to initialize the control register before using anything else on the board. (Use the function **`sm_find_boards`** to do this.)

2. PLCBus Reset

A PLCBus reset command strobes both the PCL-AK and quadrature decoder/counter reset lines, forcing hardware resets for both. The control register and motor driver IC are not affected by a PLCBus reset.

3. Watchdog Reset

The watchdog timer is a safety feature that halts the PCL-AK (and therefore motion) in the event of a system crash. When the watchdog is turned on, the application must "hit" the watchdog at least every 1.5 seconds. The watchdog is "hit" every time the application reads the quadrature counter (the actual chip need not be present), writes the control register, or calls the function **`sm_hitwd`**. The quadrature counter is ***not*** reset in the event of a watchdog timeout.

Once reset this way, the PCL-AK pulse generator chip will stay reset until the application hits the watchdog again. Connecting the jumper on header J1 enables the watchdog. The watchdog is disabled if this jumper is not connected.

4. PCL-AK Reset

In addition to the watchdog reset and the power-up reset, there are two other ways to reset the PCL-AK pulse generator:

To achieve a hardware reset, drive the PCL-AK reset line low. This line is connected to the control register (bit 1). A hardware reset halts all activity of the controller and resets all internal counters and registers. The function **`smc_hardreset`** will pulse this line and generate the reset.

To achieve a software reset, write a reset command to the controller. A software reset immediately stops pulse generation and deactivates the PCL-AK's interrupt request line if it is active. The contents of PCL-AK registers are not affected. A software reset is typically used at the end of a programmed operation that generates an interrupt when it finishes. The function **`smc_softreset`** is used to generate a software reset.

5. Quadrature Counter Reset

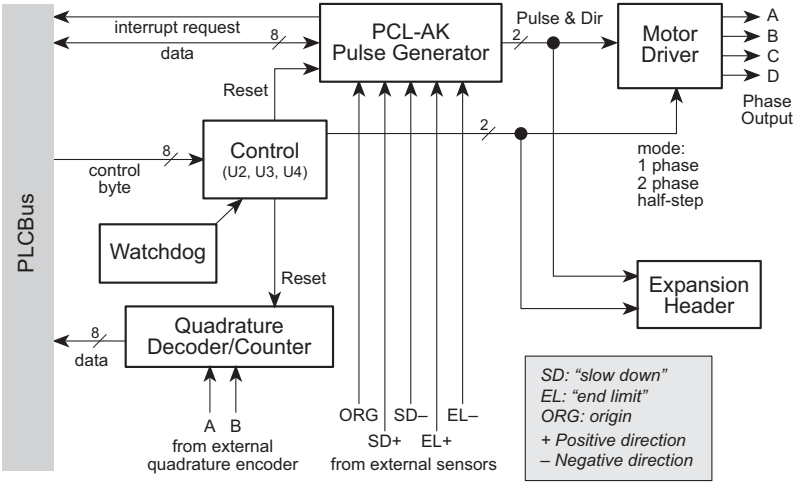
The quadrature counter is reset to zero on power-up. Use the function **`smq_hardreset`** at any time to reset the quadrature counter.

XP8800 Operation

The XP8800 has these three major components.

1. PCL-AK pulse generator.
2. UCN5804 motor driver.
3. HCTL-2016 quadrature decoder/counter.

These components are coupled with a control register (U3) and control logic (U2, U4), as shown in Figure 15-4. One or more of these components may be left unused. For example, the XP8800 can be used solely as a quadrature counter by ignoring the PCL-AK and the motor driver ICs. The XP8800 can even be used as a timer by ignoring or disabling its



outputs.

Figure 15-4. XP8800 Block Diagram

PCL-AK Pulse Generator

The PCL-AK pulse generator at the heart of the XP8800 controls the motor driver IC. The bidirectional /PULSE output signal steps a motor. If PDIR is 1, the move is in the + direction, 0 means the move is in the – direction. The PCL-AK can generate thousands of different pulse rates.

The PCL-AK can sense external signals such as "slow down," "end limit," and "origin," and can accelerate and decelerate the motor driver IC between high-speed and low-speed settings. The PCL-AK is able to generate interrupt requests in response to certain conditions such as the end of the operation. The PCL-AK chip can signal the stepper motor to stop immediately or decelerate to a stop.

The PCL-AK has the following three basic modes of operation.

1. Continuous Mode—The PCL-AK continues to generate pulses until instructed to stop or an external signal arrives.
2. Preset Mode—The PCL-AK generates pulses until its preset counter decrements to 0 or an external signal arrives.
3. Origin Mode—The PCL-AK generates pulses until an “origin” pulse arrives.
4. Stop Mode—The PCL-AK either generates pulses for the stepper motor chip to bring the stepper motor to an immediate stop or it generates pulses leading to a deceleration to a stop.

Figure 15-5 shows a block diagram of the PCL-AK.

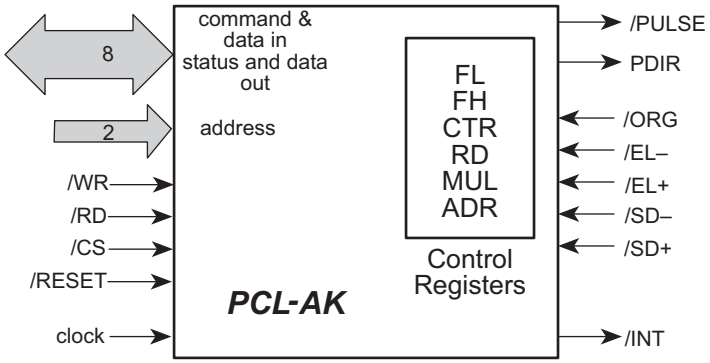


Figure 15-5. Block Diagram of PCL-AK Pulse Generator Chip

Communicating with the PCL-AK

The PCL-AK is controlled by writing to its command buffer and by writing values to its control registers. The chip can be monitored to find out what it is doing by reading the status register or a control register. Only the counter and ramp-down point registers are readable.

The internal registers of the PCL-AK can be reset by pulsing the /RESET line. A software reset does not reset the internal registers.

Table 15-1 provides the meanings for commands used with the PCL-AK.

Table 15-1. PCL-AK Commands

PCL-AK Signals					Meaning
/CS	A1	A0	/RD	/WR	
0	0	0	1	0	Write command buffer.
0	0	1	1	0	Write register bits 0–7.
0	1	0	1	0	Write bits 8–15.
0	1	1	1	0	Write register bits 16–17 (counter).
0	0	0	0	1	Read status.
0	0	1	0	1	Read register bits 0–7.
0	1	0	0	1	Read register bits 8–15.
0	1	1	0	1	Read register bits 16–17 (counter) with assorted status bits.
1	×	×	×	×	D0–D7 at high impedance.
0	×	×	0	0	Inhibit.

Registers

Table 15-2 lists the PCL-AK registers.

Table 15-2. PCL-AK Registers

Register	Bits	Description
CTR	18	Down counter, gives the number of pulses to generate for Preset Mode. This register is readable. When read, it gives the number of remaining pulses.
FL	13	Low frequency from which to accelerate or decelerate.
FH	13	High frequency from which to decelerate or accelerate.
ADR	10	Acceleration/deceleration rate.
RD	16	Ramp-down point. When the PCL-AK is generating pulses in the Preset Mode, the ramp-down point is the point (number of pulses before end-of-count) at which the PCL-AK will start ramping down (decelerating) from high speed to low speed. This register is readable.
MUL	10	Multiplier register, interacts with FL and FH to give various pulse rates.

Acceleration/Deceleration Rate (ADR) Register

The ADR register—with settings from 2 to 1023—governs the ramping-up (acceleration) and ramping-down (deceleration) characteristics. When started in a high-speed mode, the PCL-AK pulse generator starts with the speed set in the FL register and accelerates to reach the speed set in the FH register.

The Z-World reference clock frequency is 6 MHz. Thus, a clock period is 1/6 µs. The time it takes to accelerate or decelerate is

$$T_{RAMP} = (FH - FL) \times (\text{rate in ADR})/6 \ \mu s.$$

The relationship between acceleration and the rate in ADR is

$$\text{acceleration} = \frac{\text{CLOCK}}{\text{rate in ADR}} \text{pulses/s}^2.$$

The stepper motor might not operate if the ADR rate is too small because the acceleration will then be too fast.

The relationship between the value of a speed register (FL or FH varies from 1 to 8191) and the actual output frequency of PCK-AL is

$$\nu_{HIGH} = \frac{FH}{8192} \times \frac{\text{CLOCK}}{\text{MUL}} \text{pulses/s.}$$

$$\nu_{LOW} = \frac{FL}{8192} \times \frac{\text{CLOCK}}{\text{MUL}} \text{pulses/s.}$$

The term MUL is the value of the multiplier register, and can be from 2 to 1023. With Z-World's 6 MHz reference clock, MUL = 732 (732.421875 rounded off).

Referring to Figure 15-6, the number of pulses output during T_{DEC} is represented by the area of the shaded trapezoid.

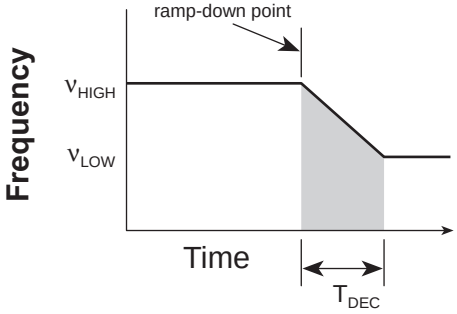


Figure 15-6. Calculating the Number of Pulses

$$\text{Number of pulses} = \frac{(\nu_{HIGH} + \nu_{LOW}) \times T_{DEC}}{2} \text{ pulses}$$

$$= \frac{(FH^2 - FL^2) \times ADR}{16{,}384 \times MUL} \text{ pulses.}$$

Status Bits

Status bits are available at PCL-AK address 0 and 3. The status bits for address 0 are explained below.

D7	D6	D5	D4	D3	D2	D1	D0

D0 1—**/EL–** (end limit) signal

D1 1—**/EL+** signal

D2 1—**/ORG** (origin) signal

D3 1—counter output = 0

D4 1—ramp-down point register (**RD**) selected
0—other register selected

D5 1—frequency stabilized after ramp down or ramp up

D6 1—operation in progress

D7 0—**/INT** (interrupt request) active

Bits 0 and 1 of the address 3 status depend on whether the RD (ramp-down point) register was selected prior to reading the status. The status bits for address 3 are explained below.

D0 If **RD** register is selected

0—stop interrupt signal is being output
else—bit 16 of counter is output

D1 If **RD** register is selected

0—ramp-down point interrupt signal is being output
else—bit 17 of counter is output

D2 1—**/SD–** (slow down) signal

D3 1—**/SD+** signal

D4 1—Ramp up in progress

D5 1—Ramp down in progress

D6 1—counter < ramp-down point

D7 0—**/PULSE** signal is not active
1—**/PULSE** signal is active

See Z-World Technical Note 101, ***Operating the PLC-AK High-Speed Pulse Generator***, for more information on the PCL-AK chip.

UCN5804 Motor Driver IC

The motor driver chip (UCN5804) receives two pulse signals from the PCL-AK pulse generator. One signal, /PULSE, steps the motor. The other signal, PDIR, specifies the motor rotation (high = forward, low = reverse).

The driver receives two mode signals from the control register. Their meanings are summarized in Table 15-3. The 0s in the table indicate that the driver line is ON, that is, it is sinking current.

Table 15-3. Motor Driver Chip Modes

Bit 7	Bit 6	Mode
0	0	Two phase
0	1	Half-step
1	0	Single phase
1	1	Undefined–Do not use

The motor driver chip generates phase signals A, B, C, and D to produce these modes according to the chart in Figure 15-7. The top line of each sequence indicates the state of the driver at power-up.

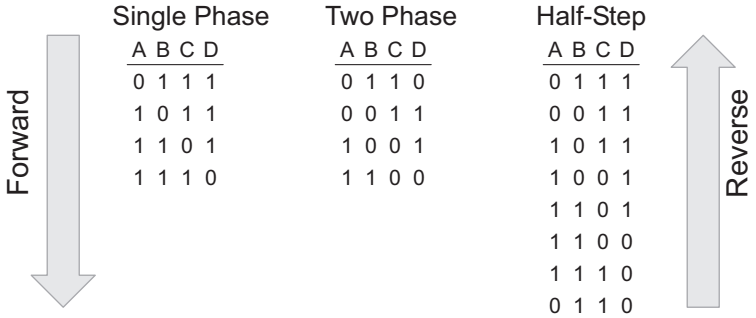


Figure 15-7. Illustration of Phase Signals A, B, C, and D Produced by Motor Driver Chip

Figure 15-8 shows how the phase lines are connected to the motor's windings.

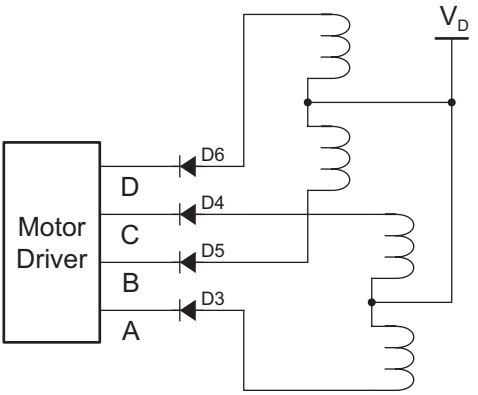


Figure 15-8. Connection of Phase Lines to Motor

Driver Power

To select a voltage for the motor driver chip, be sure to consider the various losses in the drive circuit, including the collector/emitter voltage and the voltage of the blocking diode. Figure 15-9 illustrates these voltages.

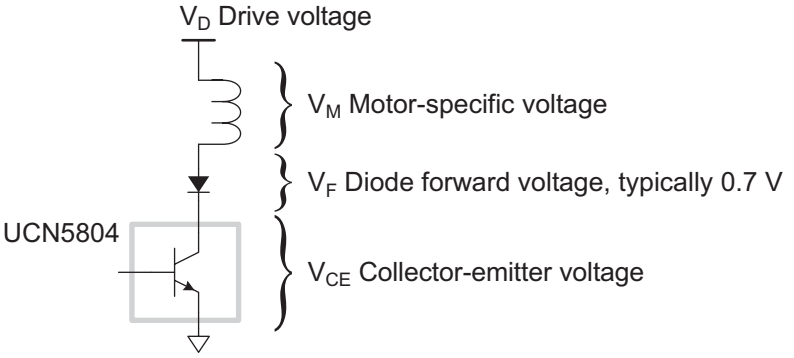


Figure 15-9. Voltage Drops Associated with UCN5804 Motor Driver Chip

Table 15-4 lists typical ratings for the UCN5804 motor driver chip.

Table 15-4. Typical Ratings UCN5804 Motor Driver IC

I_D	V_{CE}
0.7 A	1.0 V
1.0 A	1.1 V
1.25 A	1.2 V

For example, consider a 5 V, 1 A motor.

$$\begin{aligned} V_D &= V_M + V_{CE} + V_F \\ &= 5\ V + 1.1\ V + 0.7\ V \\ &= 6.8\ V \end{aligned}$$

You would need a 6.8 V, 2 A power supply (for 2-phase drive) in addition to the power required by the logic.

Remember to connect the K line on the screw connector block (H6) to the high side of the drive voltage.

Quadrature Decoder/Counter

The HCTL-2016 is a 16-bit quadrature decoder and counter. Its two lines, A and B, accept two quadrature encoded signals, that is, two square waves 90° out of phase. The order in which these signals make transitions determines the direction that is counted. Figure 15-10 illustrates this counting operation.

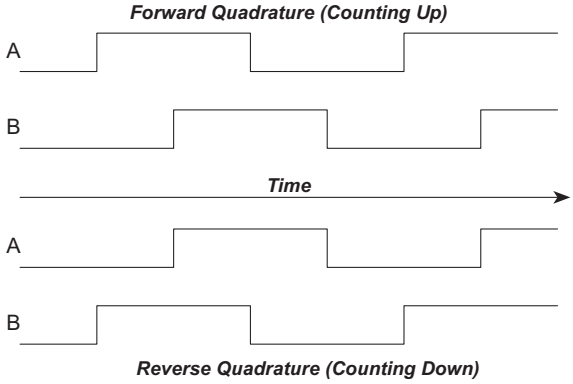
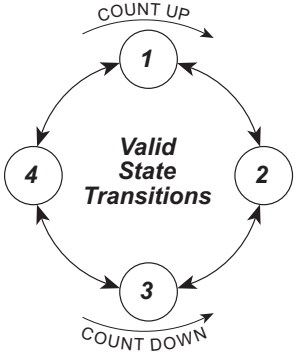


Figure 15-10. HCTL-2016 Quadrature Counting Operation

There are four states of lines A and B, as shown in Figure 15-11. The counter counts up or down, depending on the state transition.



CH A	CH B	STATE
1	0	1
1	1	2
0	1	3
0	0	4

Figure 15-11. HCTL-2016 Quadrature Counting Operation

The speed at which the counter can operate is limited by the reference clock (12 MHz). The counter can operate at up to one quarter of this frequency. Thus, the maximum reliable counting frequency is 3 MHz.

The counter can be read as two successive bytes.

Control Register

The control register is an 8-bit write-only latch that controls the operation of the XP8800. Table 15-5 explains the meaning of each bit in the register (bit 0 is the least significant bit).

Table 15-5. Control Register Bits

Bit	Name	Meaning
0	**RESCNT**	Reset quadrature decoder/counter. Low means reset.
1	**RESCTL**	Reset the PCL-AK. Low means reset.
2	**LED**	LED. Low means ON.
3	**SEL0**	Local address line.
4	**SEL1**	Local address line.
5	**DRVOE**	Enable motor driver IC output. Low means ON.
6	**HSTEP**	Half-step mode for motor driver IC when this bit is 1 and bit 7 is 0.
7	**WAVE**	• Single-phase mode for motor driver IC when this bit is 1 and bit 6 is 0. • Two-phase mode when this bit is 0 and bit 6 is 0.

The select lines SEL0 and SEL1 have a specific meaning. They are connected to the two address lines of the PCL-AK pulse generator. SEL0 is also connected to the quadrature decoder/counter. Coupled with PAL logic, these select lines allow you to read and write to the PCL-AK and to read the 16-bit counter value. (The function library **STEP.LIB** takes care of the details.)

PLCBus Interrupts

Be careful when processing interrupts from the PLCBus. Interrupts can come from any source, including other expansion boards. A PLCBus interrupt service routine must determine where the interrupt originated and what to do.

CHAPTER 16: SOFTWARE REFERENCE

Chapter 16 describes the Dynamic C functions used to initialize the XP8800 Series expansion boards and to control the resulting outputs. The following major sections are included.

- XP8800 Board Addresses
- Dynamic C Libraries
- XP8800 Software

XP8800 Board Addresses

Up to 16 XP8800 addresses are possible on the PLCBus. Power constraints usually limit the number of XP8800 expansion boards to four, allowing four axes of control.

Each XP8800 has three addressable components: the PCL-AK pulse generator, the quadrature decoder/counter, and the control register. The address of a particular XP8800 is determined by jumpers on header H4 as shown here.

abcd1100 x0000Rxx

where

a = 0 if H4 pins 1–2 are connected, and 1 if not
b = 0 if H4 pins 3–4 are connected, and 1 if not
c = 0 if H4 pins 5–6 are connected, and 1 if not
d = 0 if H4 pins 7–8 are connected, and 1 if not
x = does not matter
R = 0 to read or write PCL-AK pulse generator
= 1 to read the quadrature counter
= 1 to write the control register

The address is placed on the PLCBus as 2 bytes using two bus cycles, BUSADR0 and BUSADR1. The lower four bits of the first byte (1100) identify the address as being 8×2 format.

The address is placed on the bus using the functions **`set82adr`** and **`set81adr`**.

The LED (D2) will light up on the XP8800 that matches the address the software placed on the PLCBus.

Examples

1. Write the control register on the XP8800 whose address jumpers are 3 (abcd = 0011).

```
out0 (BUSADR0), 3Ch ; 00111100 1st addr byte
out0 (BUSADR1), 04h ; 00000100 2nd addr byte
Set shadow variable = control register value, then...
out0 (BUSWR),  ‹shadow› ; control bits
```

2. Write a command to the PCL-AK on the XP8800 whose address jumpers are 8 (abcd = 1000).

```
;first, make select lines 00
  out0 (BUSADR0), 8Ch ; 10001100 1st addr byte
  out0 (BUSADR1), 04h ; 00000100 2nd addr byte
  Set shadow variable = AND( shadow variable,
    0xE7 ), then ...
  out0 (BUSWR),  <shadow> ; control bits
;now address the PCL-AK and send command
  out0 (BUSADR1), 00h ; 00000000 2nd addr byte
  out0 (BUSWR), <command> ; command
```

3. Read the 16-bit quadrature counter on the XP8800 whose address jumpers are 13 (abcd = 1101).

```
;first, make select lines 00 to get high byte
  out0 (BUSADR0), DCh ; 11011100 1st addr byte
  out0 (BUSADR1), 04h ; 00000100 2nd addr byte
  Set shadow variable = AND( shadow variable,
    0xE7 ), then ...
  out0 (BUSWR),  <shadow> ; control bits
  in0  <high>, (BUSRD0) ; get high byte
;next, make select lines 01 to get low byte
  Set shadow variable = OR( shadow variable,
    0x08 ), then ...
  out0 (BUSWR),  <shadow> ; control bits
  in0  <low>, (BUSRD0) ; get low byte
Return counter value = high << 8 + low
```

In general there is no need to program the XP8800 at these low levels. Software in the Dynamic C **STEP.LIB** library takes care of these details.

Logical Addresses

Software in the Dynamic C **STEP.LIB** library keeps information for all XP8800s on the PLCBus in a table, sorted by XP8800 address. Thus, XP8800s have "logical addresses" that are simply indexes in the table.

For example, suppose there are three XP8800s on the PLCBus with addresses of 3 (0011), 8 (1000), and 13 (1101). Table 8-1 shows the table used by the software.

The logical addresses for these 3 boards would be 0, 1, and 2. The physical addresses are stored in the table. The function **sm_find_boards** sets up this table.

Table 16-1. Example of STEP.LIB Table for XP8800 Logical Addresses

Index	Address
0	0011
1	1000
2	1101
marker	—

Dynamic C Libraries

Several Dynamic C function libraries contain the software functions described in this chapter. The chart in Table 8-2 identifies which libraries must be used with particular Z-World controllers.

Table 16-2. Dynamic C Libraries Required by Z-World Controllers for XP8800 Expansion Boards

Library Needed	Controller
DRIVERS.LIB	BL1200, BL1600, PK2100, PK2200
EZIOCMMN.LIB	BL1200, BL1600, PK2100, PK2200
EZIOPBDV.LIB	BL1200, BL1600, PK2100, PK2200
EZIOPLC.LIB	BL1200, BL1600, PK2100, PK2200
EZIOPLC2.LIB	BL1700
EZIOBL17.LIB	BL1700
PLC_EXP.LIB	BL1200, BL1600, PK2100, PK2200

Before using one of these libraries in an application, first include the library name in a **#use** command. For example, to use functions in the library **PLC_EXP.LIB**, be sure there is a line at the beginning of the program in the following format.

```
#use PLC_EXP.LIB
```

XP8800 Software

The sample programs **SM_DEMO1.C**, **SM_DEMO2.C**, and **SM_DEMO3.C** in the Dynamic C **SAMPLES\PLCBUS** subdirectory illustrate the use of these functions.

The software is designed to simplify the task of using the XP8800 on the PLCBus. Z-World recommends using the software or at least following the guidelines for the software structure.

1. Only access the control register using driver functions. These functions keep track of the shadow variables that prevent inadvertently changing other control lines.
2. Initialize and use the arrays designated for handling multiple board addresses and status. These are described in detail.
3. If using interrupts, make the declaration

   ```
   #define USE_STEPPER
   ```

   early in the main program. This tells the PLCBus interrupt service to call the function **sm_int**.
4. Also, if using interrupts, add the call

   ```
   relocate_int1();
   ```

   This connects the proper PLCBus interrupt service routine to the interrupt vector.

Data Structures

The XP8800 driver software uses a table to represent all the XP8800s in a system. There can be up to four XP8800s, and other PLCBus expansion boards may also be used, subject to power constraints.

Table 4-2 shows how the Dynamic C **STEP.LIB** library assigns and sorts the XP8800 logical addresses. These XP8800 "logical addresses" are simply indexes in the table.

For example, the logical addresses for the three boards in Table 8-1 are 0, 1, and 2. The physical addresses are stored in the table. Call **sm_find_boards** before doing anything else. This function searches the PLCBus and initializes the table to represent the state of the XP8800s.

These four arrays define the table.

```
int  sm_addr  [17];
char sm_stat  [16];
char sm_flag  [16];
char sm_shadow[16];
```

The array **sm_addr** holds the PLCBus address of each XP8800 existing on the PLCBus. This array has one extra element, because the software places a marker (address = –1 or 0xFFFF) following the last real board address in the array.

The array **sm_stat** contains copies of the address 0 status bits of the PCL-AK pulse generator for each XP8800 on the PLCBus. The array is updated by the motor control interrupt service routine (ISR) every time a PLCBus interrupt is generated.

The array **sm_flag** is updated at the same time as **sm_stat** and represents whether a board is awaiting service (its interrupt line asserted).

The array **sm_shadow** holds shadow variables for the XP8800 control registers. Control registers are write-only. If software fails to remember how control lines are set, chances are good that control lines will become set incorrectly. The shadow variables provide the memory.

Interrupts

Since the PLCBus has a single shared interrupt line, special care must be taken when servicing interrupts across it. During PLCBus interrupt service, all possible interrupt sources must be checked to see if they are currently awaiting service. These include other PLCBus expansion boards.

The interrupt function **sm_int** polls all XP8800s on the PLCBus and updates the arrays **sm_stat** and **sm_flag** for each. It also sends a software reset to each XP8800 that is asserting an interrupt request. The software reset clears the interrupt request. If this reset is not issued, the system would lock up since the interrupt line would never go inactive.

By including the statement

```
#define USE_STEPPER
```

early in the main program, the PLCBus interrupt service routine will call **sm_int**. Your application should periodically check the status of the interrupt request flags in the **sm_flag** array to determine when to service the XP8800.

Although the function **sm_int** does what it is supposed to do, it probably does not do what you would want it to do. Z-World has provided **sm_int** to demonstrate how to use the XP8800 in an interrupt-driven system. Since **sm_int** requires polling flags to provide service, it is not as efficient as a true interrupt-controlled driver would be. What this function does is guarantee that interrupts generated by a motor controller are serviced so that the PLCBus interrupt is not held active by the controller, locking up the system.

If you wish to do all motor processing in the background, replace the code in the function **sm_int (**between the labels **mirq** and **fin**) with your own code.

XP8800 Driver Functions

Tables 8-3, 8-4, 8-5, 8-6, and 8-7 list the various XP8800 software drivers in the Dynamic C **STEP.LIB** library.

Table 16-3. XP8800 General and Initialization Functions

Type	Function	Description
int	sm_bdaddr	Generates address from jumper value
void	sm_board_reset	Issues full board reset
int	sm_find_boards	Finds and initialize all XP8800s
void	sm_hitwd	Hits watchdog timer
void	sm_int	General ISR for XP8800s
int	sm_poll	Polls specified XP8800

Table 16-4. XP8800 Control Register Functions

Type	Function	Description
void	sm_ctlreg	Writes control register and updates shadow variable
void	sm_drvoe	Turns motor driver IC output on or off
void	sm_led	Turns LED (D1) on or off
void	sm_sel00	Sets select lines to 00
void	sm_sel01	Sets select lines to 01
void	sm_sel10	Sets select lines to 10
void	sm_sel11	Sets select lines to 11

Table 16-5. XP8800 Motor Controller Functions

Type	Function	Description
`void`	`smc_cmd`	Writes to PCL-AK command register
`void`	`smc_hardreset`	Pulses PCL-AK reset line, registers are reset
`void`	`smc_manual_move`	Starts continuous movement, movement continues until told to stop
`void`	`smc_seek_origin`	Starts continuous movement, movement continues until origin pulse (/ORG)
`void`	`smc_setmove`	Sets PCL-AK registers for a move operation
`void`	`smc_setspeed`	Sets PCL-AK's two speed registers
`void`	`smc_softreset`	Sends reset command to PCL-AK, registers are not reset
`char`	`smc_stat0`	Reads PCL-AK status register (at address 0)
`char`	`smc_stat3`	Reads PCL-AK status register (at address 3)

Table 16-6. XP8800 Quadrature Counter Functions

Type	Function	Description
`void`	`smq_hardreset`	Pulses quadrature counter reset line
`unsigned int`	`smq_read16`	Reads entire 16-bit counter value
`char`	`smq_read8`	Reads counter's lower 8 bits

Table 16-7. Miscellaneous XP8800 Functions

Type	Function	Description
`void`	`set81adr`	Places XP8800 address on bus (shortcut)
`void`	`set82adr`	Places XP8800 address on bus
`unsigned int`	`smcq_moveto`	Uses the motor's quadrature decoder to move to location

Miscellaneous XP8800 Function Descriptions

In all the following function descriptions, the parameter **index** is a number from 0 to 15 that represents the sequence of boards found by **sm_find_boards**. The board with the lowest jumper setting is at position 0, and so on.

- **void set82adr(int addr)**

  Places the specified address on the PLCBus in 8×2 addressing mode. The term **addr** is a physical board address. Its upper byte must be xxxx1100 (binary), and the lower byte should be 0 to read or write the PCL-AK pulse generator, or 1 to read the quadrature counter or to write the control register. The upper 4 bits of the address correspond to the jumpers on the intended XP8800.

  The execution time for this function is 87 cycles, assuming 0 wait states, that is

  14.16 µs at 6.144 MHz (71 kHz)

  9.44 µs at 9.216 MHz (109 kHz)

- **void set81adr(int addr)**

  Places the specified address on the PLCBus in 8×2 addressing mode. The term **addr** is the lower byte a physical board address. This function assumes that the upper byte has already been placed on the bus. The lower byte should be 0 to read or write the PCL-AK pulse generator, or 1 to read the quadrature counter or to write the control register. The main purpose of this function is to save PLCBus cycles.

  The execution time for this function is 60 cycles, assuming 0 wait states, that is

  9.77 µs at 6.144 MHz (102 kHz)

  6.50 µs at 9.216 MHz (154 kHz)

- **int sm_bdaddr(int jumpers)**

  Returns the physical PLCBus address for an XP8800 that has the specified jumper settings on header H4. The term **jumpers** must be an integer from 0 to 15.

  The function returns the physical PLCBus address in a form directly passable to **set82adr**.

- **void sm_board_reset(int index)**

  Performs a hardware reset XP8800 identified by **index**. This resets the PCL-AK pulse generator and the quadrature decoder/counter, and disables the motor driver IC and sets it to two-phase mode. The function also sets the control register's two select lines to 00.

- **`void sm_ctlreg( int index, int value )`**

  Writes **`value`** to the control register on the XP8800 specified by **`index`**. The function updates the shadow variable for the control register.

- **`void sm_drvoe( int index, int onoff )`**

  Turns the motor driver IC of the XP8800 specified by **`index`** on or off. The term **`onoff`** is Boolean: when zero, the motor driver IC gets turned off. Otherwise, it gets turned on.

- **`int sm_find_boards()`**

  Searches for all possible XP8800s and fills in the XP8800 table, which is sorted according to physical board address. The table holds physical addresses in the array **`sm_addr`**. The table also holds status bytes and interrupt service flags, which this function initializes.

  The function return is the number of boards found. The function places a marker (–1 or 0xFFFF) following the last entry in the table.

  The function sends a control register value of 0xA7 (1010 0111) to all XP8800s found. This puts the motor driver IC in two-phase mode and turns it off, makes the select lines 00, turns the LED (D2) on, and resets both the PCL-AK pulse generator and the quadrature counter.

  The function return is the number of XP8800 boards on the PLCBus that respond to the search.

  The XP8800 table consists of these four arrays.

  `sm_addr` a board's physical PLCBus address.

  `sm_stat` holds the last status (address 0) read from the board's PCL-AK.

  `sm_flag`, when non-zero, indicates the XP8800 has requested an interrupt and is awaiting service.

  `sm_shadow` holds the last value written to the board's control register.

  This function is among the first to call when operating XP8800 expansion boards. After the table is initialized, function calls will generally refer to XP8800s by their ***table index***.

- **`void sm_hitwd( int index )`**

  Resets the watchdog timer on the XP8800 specified by **`index`**. It does this by reading the quadrature counter. (The quadrature chip does not have to be present.)

- **`void sm_int()`**

  This is a general-purpose XP8800 function that can be called by the PLCBus interrupt service routine (ISR). This function checks the status (at PCL-AK address 0) of all boards, updating the **`sm_stat`** array. When an interrupt request is detected, the appropriate **`sm_flag`** value is set and the function issues a software reset to the PCL-AK to deactivate the interrupt request.

  The application must then monitor the interrupt service flags to determine when an operation has been completed.

  To use this function, do the following.

  1. Call **`sm_find_boards`** at the beginning of the application to initialize the XP8800 table.
  2. Add the following statement early in the application to link **`sm_int`** to the PLCBus ISR.

     ```
     #define USE_STEPPER // activate sm_int
     ```

  3. Add the following statement early in the application to ensure that the PLCBus interrupt line is activated.

     ```
     outport( ITC, (inport(ITC)&0xFD) );
                                    // enable INT1
     ```

  If all motor processing is to be done in the background (that is, as part of the interrupt service), open and edit **`STEP.LIB`**. Find **`sm_int`** and replace the code between the labels **`mirq`** and **`fin`** with your own code.

- **`void sm_led( int index, int onoff )`**

  Turns the LED (D1) on the XP8800 specified by **`index`** on or off. The value **`onoff`** is Boolean: when zero, the function turns the LED off. Otherwise, it turns the LED on.

- **`int sm_poll( unsigned int address )`**

  Returns 0 if the XP8800 specified by **`address`** is present (and responding) on the PLCBus. The parameter **`address`** must be a physical board address, such as that returned by **`sm_bdaddr`** (**`jumpers`**).

  All PLCBus expansion boards respond to a BUSRD1 cycle by sinking data line 0 (normally high). The board is not present if a 1 is returned.

- **`void sm_sel00( int index )`**

  Sets the select lines to 00 on the XP8800 specified by **`index`**.

- **`void sm_sel01( int index )`**

  Sets the select lines to 01 on the XP8800 specified by **`index`**.

- **`void sm_sel10( int index )`**

  Sets the select lines to 10 on the XP8800 specified by **`index`**.

- **`void sm_sel11( int index )`**

  Sets the select lines to 11 on the XP8800 specified by **`index`**.

- **`void smc_cmd( int index, int data )`**

  Writes **`data`** to the command register of the PCL-AK pulse generator on the XP8800 specified by **`index`**.

- **`void smc_hardreset( int index )`**

  Causes a hardware reset of the PCL-AK on the XP8800 specified by **`index`**. This stops any pulse output (that is, motor movement) and clears the internal registers of the PCL-AK. It does this by giving a negative pulse on bit 1 of the control register.

- **`void smc_manual_move( int index, int dir, int speed )`**

  Starts a manual (or continuous) move operation on the XP8800 specified by **`index`**. The motor will move until the application issues a decelerating stop command, a software or hardware reset, or until the application detects an end-limit or origin signal (if these are enabled).

  The terms **`dir`** and **`speed`** are Boolean. If **`dir`** is non-zero, movement is in the "+ "direction. Otherwise, movement is in the "–" direction. If **`speed`** is zero, the PCL-AK pulse generator operates at low speed. (Pulses are generated at the rate in the FL register.) Otherwise, the PCL-AK pulse generator operates at high speed. (Pulses are generated at the rate in the FH register.)

  It is important to note that this function starts the movement and ***does not wait*** for the movement to complete. The application may then perform other tasks while the movement takes place.

- **`void smc_seek_origin( int index, int dir, int speed )`**

  Starts an "origin mode" operation on the XP8800 specified by **`index`**. The PCL-AK will generate pulses, expecting an origin pulse to occur. The motor will move until the application issues a decelerating stop command, a software or hardware reset, or until the application detects an end-limit or origin signal (if these are enabled).

  The terms **`dir`** and **`speed`** are Boolean. If **`dir`** is non-zero, movement is in the "+" direction. Otherwise, movement is in the "–" direction. If **`speed`** is zero, the PCL-AK pulse generator operates at low speed.

(Pulses are generated at the rate in the FL register.) Otherwise, the PCL-AK pulse generator operates at high speed. (Pulses are generated at the rate in the FH register.)

It is important to note that this function starts the movement and ***does not wait*** for the movement to complete. The application may then perform other tasks while the movement takes place.

The function issues a software reset to the board before proceeding.

- **`void smc_setmove( int index, long CTR, int FL, int FH,int ADR, int RD, int MUL )`**

  Sets up the registers of the PCL-AK pulse generator on the XP8800 specified by **`index`**. The meaning of the registers (listed in Table 7-2) and their interaction is complex.

  See Z-World Technical Note 101, ***Operating the PLC-AK High-Speed Pulse Generator***, for more information on the PCL-AK chip.

  When the value of the MUL register is 732, the values of the FL and FH registers approximate "pulses per second," that is, when MUL = 732, the actual pulse frequency is

  $freq_H$ = FH × 1.000576331967 pulses per second

  $freq_L$ = FL × 1.000576331967 pulses per second

- **`void smc_setspeed( int index, int fast, int slow )`**

  Sets the high (FH) and low (FL) speed registers of PCL-AK pulse generator on the XP8800 specified by **`index`**. The parameter **`fast`** is for the FH register and the parameter **`slow`** is for the FL register. Both must be in the range 1–8191.

- **`void smc_softreset( int index )`**

  Sends a software reset command to the PCL-AK pulse generator on the XP8800 specified by **`index`**. This stops pulse output (and therefore, motion) without clearing the internal registers.

- **`char smc_stat0( int index )`**

  Reads the 8-bit status register at address 0 (A1 = A0 = 0) on the PCL-AK pulse generator on the XP8800 specified by **`index`**. The function returns the status bits D0–D7 explained in Chapter 7, "Status Bits."

- **`char smc_stat3( int index )`**

  Reads the 8-bit status register at address 3 (A1 = A0 = 1) of the PCL-AK pulse generator on the XP8800 specified by **`index`**. If the RD register (ramp-down point) is selected before reading the status with address = 3, bits 0 and 1 are status bits. If any other register is selected, bits 0 and 1 represent bits 16 and 17, respectively, of the counter register.

  The function returns the status bits D0–D7 explained in Chapter 7, "Status Bits."

- **`unsigned int smcq_moveto( int index, unsigned dest, int dir, unsigned accuracy )`**

  Steps the motor on the XP8800 specified by **`index`** until the quadrature decoder/counter reaches the specified **`dest`** ± **`accuracy`**. The movement is done at the slow rate (specified in the FL register) of the PCL-AK pulse generator. The movement continues until the quadrature counter reaches the "zone of acceptance" and then stops.

  The parameter **`dir`** is Boolean: if non-zero, motion is in the "+" direction. Otherwise, motion is in the "–" direction.

  The function returns the reading of the quadrature counter when the function finally stops motion. Inertia and step locations may make this value different from the final resting place of the motor's encoder.

  The function issues a software reset to the PCL-AK following the operation.

  Example

```
main(){
   ...
   uplc_init();              // init master
   sm_find_boards();         // init all XP8800s

   smc_setspeed(0,100,200);  // move at 200 pps
   smcq_moveto(0,5000,1,25); // to location 5000±25

   delay to allow time for motor to stop fully

   loc=smq_read16(0);        // check final pos
   if(loc>5025){             // overshot?
     smc_setspeed(0,100,20);// move back at 20 pps
     smcq_moveto(0,5000,0,25);// to location 5000±25
   }
   ...
```

The function **smcq_moveto** is not a PID loop. It is the application's responsibility to manage the final position of the motor. The move speed, encoder resolution, and motor degrees/phase will affect how precise you can get. It is possible to miss a stop point if you specify too much precision. Read the quadrature counter after the operation (allowing time for the motor to come to a stop) to obtain its correct location.

- **void smq_hardreset(int index)**

  Sends a hardware reset command to the quadrature counter on the XP8800 specified by **index**. The function resets the counter to zero.

- **unsigned int smq_read16(int index)**

  Returns the entire 16-bit value of the quadrature counter on the XP8800 specified by **index**.

- **char smq_read8(int index);**

  Returns the lower 8 bits of the quadrature counter on the XP8800 specified by **index**, a number from 0 to 15 as in **smq_read16**.

Sample Program

The sample program simulates a single-axis system with end-limit and slowdown sensors in both directions.

After initialization, the XP8800 first seeks the origin. Then the motor goes back and forth a few times, moving in one direction until an "end-limit" signal occurs, then switches direction. As the motor moves, it responds to any "slowdown" signal it receives.

The following items are needed to run this program.

- A stepper motor connected to an XP8800 connected, via the PLCBus, to a Z-World PK2200 or PK2100 controller.
- A length of wire or a test probe to connect various signals to ground. This simulates the occurrence of end-limit, slowdown or origin conditions.

The sample program prompts you to make the appropriate connections.

```
/*****************************************************
Simulate origin signal.
*****************************************************/
void wait_origin( int id ){
#define ORG 0x04           // bit 2
   printf("Connect /ORG to GND ");
   printf("to simulate origin signal... ");
   while(!( smc_stat0(id) & ORG )) runwatch();
   printf( "ORG detected.\n" );
}
/*****************************************************
Simulates end-limit signal. Dir = CW or CCW.
*****************************************************/
void wait_EL( int id, int dir ){
   int mask;               // bit 0 for EL-, bit 1 for EL+
   char sign;              // "+" or "-"
   if( dir ){
      mask = 2; sign = '+'; // + direction (CW)
   }else{
      mask = 1; sign = '-'; // - direction (CCW)
   }
   printf( "Connect /EL%c to GND ", sign );
   printf( "to simulate end-limit... ");
   while(!( smc_stat0(id) & mask ))runwatch();
   printf( "end-limit detected.\n" );
}
/*********************************************/
#define CCW 0    // counterclockwise direction (-)
#define CW  1    // clockwise direction (+)
/*********************************************/
main(){
   int FL  =  10;          // low speed 10 pps
   int FH  = 100;          // high speed 100 pps
   int ADR = 500;          // accel/decel "rate" = 500
   int MUL = 732;          // makes FL and FH units "pps"
   int ID = 0;             // board index
   int i;
   uplc_init();            // assume PK2200 or PK2100
   Reset_PBus();           // reset PLCBus with delay
   Reset_PBus_Wait();
// Search PLCBus. Build table
   if( sm_find_boards() == 0 ){
      printf( "No XP8800s." ); exit(0);
   }
// Use first board. Set up operation
   sm_board_reset( ID );
   sm_drvoe( ID, 1 );      // motor driver on
   sm_led  ( ID, 1 );      // LED on
   smc_setmove( ID,0L,FL,FH,ADR,0,MUL );
                                  // registers
```

```
// find origin
   smc_seek_origin( ID, CCW, 1 );  // high speed
   wait_origin( ID );
   smc_softreset( ID );
// back & forth
   for( i=0; i<3; i++ ){
   // move till EL+ slowing down upon SD+
   // 0x42 = 01xx 0010.
   // Op-mode: pos. dir. not preset. SD yes. ORG no.
      smc_cmd( ID, 0x42 );
   // 0x15 = 00x 10 101.
   // Start. High-speed. FH register
      smc_cmd( ID, 0x15 );
      wait_EL( ID, CW );   // wait for EL signal
      smc_softreset( ID );
   // move till EL- slowing down upon SD-
   // 0x4A = 01xx 1010.
   // Op-mode: neg. dir. not preset. SD yes. ORG no.
      smc_cmd( ID, 0x4A );
   // 0x15 = 00x 10 101.
   // Start. High-speed. FH register
      smc_cmd( ID, 0x15 );
      wait_EL( ID, CCW );  // wait for EL signal
      smc_softreset( ID );
   }
   sm_board_reset( ID );   // cleanup
}/*end*/
```

XP8900

XP8900

CHAPTER 17: OVERVIEW

Chapter 17 provides an overview and description of the XP8900 digital-to-analog conversion expansion boards.

The XP8900 Series is a 12-bit digital-to-analog (D/A) converter expansion board that can be used in conjunction with any Z-World PLCBus-compatible controller.

Like other Z-World expansion boards, the XP8900 Series boards can be installed in modular plastic circuit-board holders attached to a DIN rail. The XP8900 Series boards can also be mounted, with plastic standoffs, on any surface that will accept screws. Up to eight different XP8900 board addresses may be used on one PLCBus.

The XP8900 Series consists of two boards, the XP8900 with eight channels of D/A converter outputs, and the XP8910 with four channels of D/A converter outputs. Each channel produces a bipolar output of up to ±10 V DC.

The XP8900 Series features an onboard voltage regulator for PLCBus-powered operation. The XP8900 Series has connectors for user-supplied analog voltage rails, and is able to sink or source up to 7 mA with the user rails, or up to 2 mA on its own. The D/A outputs are monotonic.

An XP8900 Series board can be factory-built with one to eight D/A channels, with 8-bit or 10-bit D/A outputs, or with user-defined output voltage ranges. For more information, call your Z-World Sales Representative at (530) 757-3737.

For ordering information, call your Z-World Sales Representative at (530) 757-3737.

Specifications

Table 17-1 summarizes the specifications for the XP8900 Series expansion boards.

Table 17-1. XP8900 Series Specifications

Board Size	2.835" × 4.00" × 0.75" (72 mm × 102 mm × 19 mm)
Operating Temperature Range	-40°C to +70°C
Humidity	5% to 95%, noncondensing
Power	24 V DC, 100 mA min., 30 mA standby Accepts optional external ±12 V DC for analog power
Outputs	8 12-bit D/A channels (4 channels for XP8910), bipolar voltage output 0 V to ±10 V, can source/sink up to 2 mA per channel on internal power (up to 7 mA per channel with user-supplied rails) Slew rate: 1 V/µs in D/A converter, 0.5 V/µs in op-amp Settling time: 10 µs max. Relative accuracy: ±16 LSB (prior to op-amps) Gain temperature coefficient: -5 ppm of full-scale range per °C

The XP8900 Series expansion boards derive +5 V digital power from the PLCBus supply via LM7805 at U6. When operating without user-supplied external voltage rails, the XP8900 Series expansion boards get their +12 V analog power from the PLCBus +24 V supply via LM7812 at U7. Charge pump NJU7662 at U17 inverts this for onboard -12 V analog power. Precision +5 V and + 2 V reference voltages are derived from the +12 V supply via the REF195 at U2 and the voltage divider formed from R33, R34, and R39. The n-channel FET FDV301N at Q1 is used to switch the 2 V reference to 0 V during a power-on reset.

Figure 17-1 shows the dimensions of the XP8900 Series expansion boards.

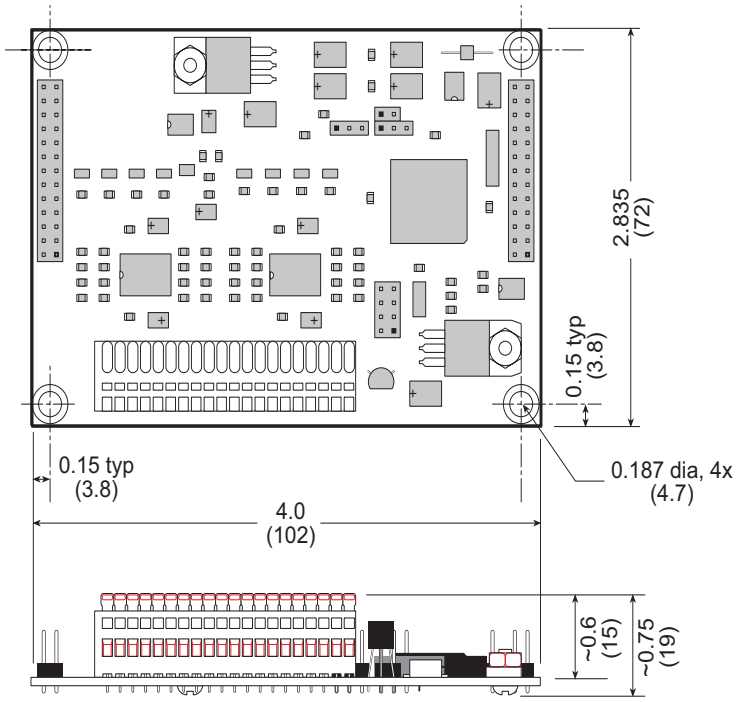


Figure 17-1. XP8900 Board Dimensions

CHAPTER 18: GETTING STARTED

Chapter 18 provides instructions for connecting XP8900 Series expansion boards to a Z-World controller. The following sections are included.

- XP8900 Series Components
- Connecting Expansion Boards to a Z-World Controller
- Setting Expansion Board Addresses
- Power

XP8900 Series Components

The XP8900 Series of expansion boards offers up to eight channels of digital-to-analog conversion outputs. Figure 18-1 shows the basic layout and orientation of components, headers, and connectors.

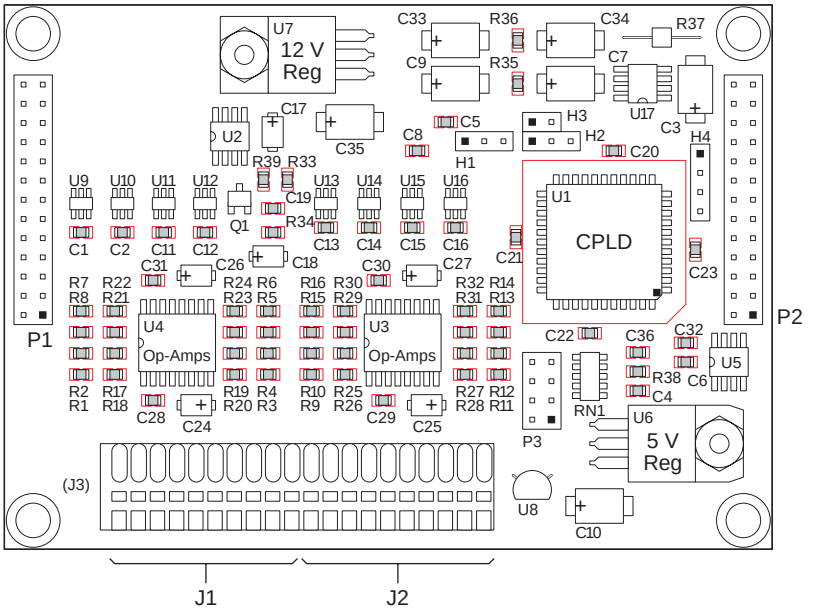


Figure 18-1. XP8900 Series Board Layout

Connecting Expansion Boards to a Z-World Controller

Use the 26-conductor ribbon cable supplied with an expansion board to connect the expansion board to the PLCBus on a Z-World controller. See Figure 18-2. The expansion board's two 26-pin PLCBus connectors, P1 and P2, are used with the ribbon cable. Z-World recommends using the cable supplied to avoid any connection problems.

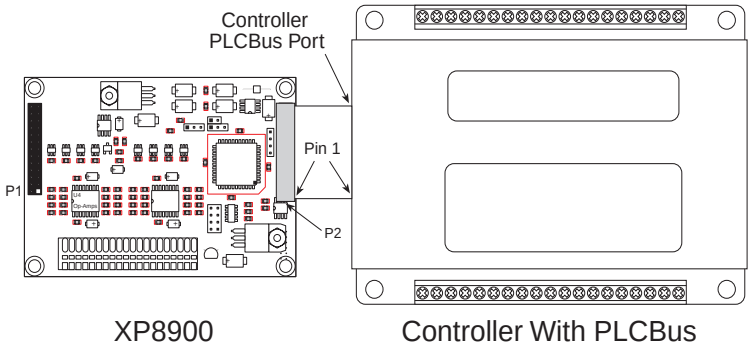


Figure 18-2. Connecting XP8600 Expansion Board to Controller PLCBus

Be sure power to the controller is disconnected before adding any expansion board to the PLCBus.

Follow these steps to connect an expansion board to a Z-World controller.

1. Attach the 26-pin ribbon cable to the expansion board's **P2** PLCBus header.
2. Connect the other end of the ribbon cable to the PLCBus port of the controller.

Be sure pin 1 of the connector cable matches up with pin 1 of both the controller and the expansion board(s).

3. If additional expansion boards are to be added, connect header **P2** on the new board to header **P1** of the board that is already connected. Lay the expansion boards side by side with headers P1 and P2 on adjacent boards close together, and make sure that all expansion boards are facing right side up.

See Appendix C, "Connecting and Mounting Multiple Boards," for more information on connecting multiple expansion boards.

4. Each expansion board comes with a factory-default board address. If more than one expansion board of each type is to be used, be sure to set a unique address for each board.

The following section on "Setting Expansion Board Addresses," and Chapter 8, "Software Reference," provide details on how to set and use expansion board addresses.

5. Power may be applied to the controller once the controller and the expansion boards are properly connected using the PLCBus ribbon cable.

See Appendix D, "Simulated PLCBus Connections," for details on the special connections that enable these expansion boards to be used with the BL1000, BL1100, BL1400, and BL1500 controllers.

Setting Expansion Board Addresses

Z-World has established an addressing scheme for the PLCBus on its controllers to allow multiple expansion boards to be connected to a controller.

Remember that each expansion board must have a unique PLCBus address if multiple boards are to be connected. If two boards have the same address, communication problems will occur that may go undetected by the controller.

Every XP8900 Series board is shipped from the factory with a default address of 7. An XP8900 Series board may be assigned any address between 0 and 7 using jumpers on the pins of header P3 to configure the board address. Figure 18-3 shows the jumper settings to set addresses 0–7. A maximum of eight XP8900 Series boards may be addressed by a controller at one time.

Pins 1–2 on header P3 are on the lower end of P3 when the XP8900 board is oriented in line with a controller and other expansion boards as shown in Figure 18-2.

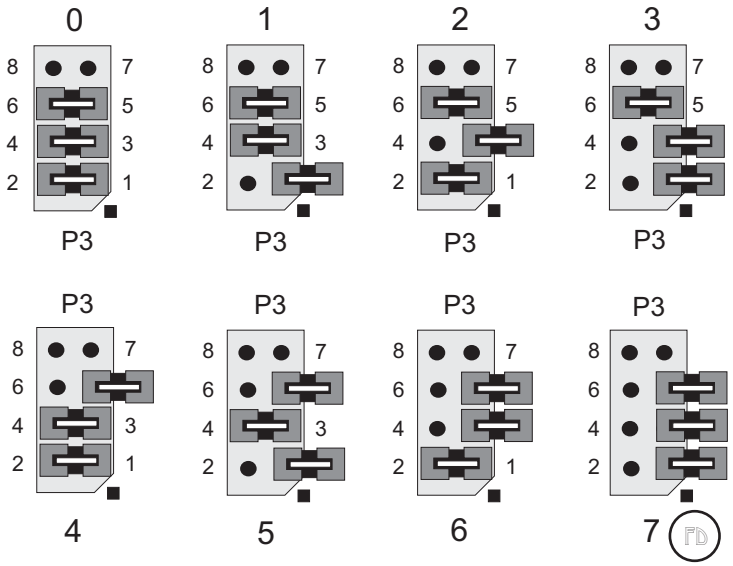


Figure 18-3. P3 Jumper Settings for XP8900 Series PLCBus Addresses

Power

Z-World's expansion boards receive power from the controller over the +24 V line of the PLCBus. An onboard regulator converts this to the +5 V and the ±12 V reference used by the expansion boards. With no output, the XP8900 Series expansion boards draw about 30 mA; with all their output channels operating at maximum current (2 mA per channel on internal power, 7 mA per channel with external voltage), the XP8900 draws 45 mA (75 mA if the external power rails are connected).

Using Digital-to-Analog Converter Boards

The follow steps summarize how to use the D/A converter boards.

1. Send a reset command to the PLCBus.
2. Place the address of the D/A converter on the PLCBus.
3. Send data serially to one of the D/A converters (Register A). When Register A is filled, load the data to D/A converter Register B where it is converted and output.
4. Use the board's analog output to control motors, attenuators or other analog devices.

These steps are done using software drivers in Dynamic C function libraries.

These steps are done using software drivers in Dynamic C function libraries. Use **DRIVERS.LIB** and **PLC_EXP.LIB**.for controllers with a PLCBus port. Use **PBUS_TG.LIB** for a BL1000, and use **PBUS_LG.LIB** for a BL1100 or a BL1300.

CHAPTER 19: I/O CONFIGURATIONS

Chapter 19 describes the built-in flexibility of the XP8900 Series expansion boards, and describes how to configure the available inputs/outputs. The following sections are included.

- XP8900 Series Pin Assignments
- XP8900 Series Circuitry

XP8900 Series Pin Assignments

The XP8900 has eight channels of bipolar voltage outputs, each with its own individual ground, and terminals for user-supplied positive and negative voltage rails, also with their own individual grounds. These are all located on Wago connectors J1 and J2, as shown in Figure 19-1.

The pin assignments for the XP8910 are similar, except there are only four output channels. There are no outputs on pin 9 of J1, and there are no outputs on pins 1, 3, and 5 of J2.

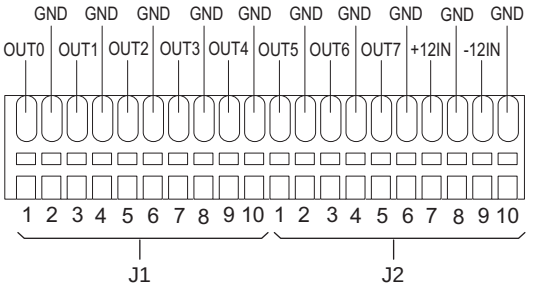


Figure 19-1. XP8900 Wago Connectors J1 and J2

No special configurations are needed for the D/A converter outputs, which are controlled by the software drivers.

An external ±12 V DC may be connected to the XP8900 Series boards to reduce analog noise or to increase the current drive. Figure 19-2 provides the jumper settings for headers H1, H2, and H3 to accommodate the external power. The external ±12 V supply is connected to the XP8900 Series board via pins 7 and 9 on Wago connector J2.

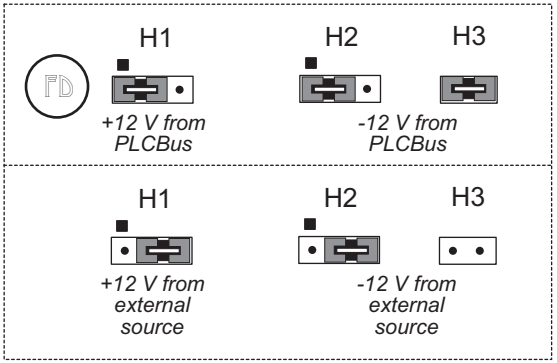


Figure 19-2. XP8900 Series ±12 V Supply Jumper Settings

XP8900 Series Circuitry

The XP8900's D/A circuitry consists of eight 12-bit AD5320 D/A converters, U9 to U16, and two OP497G quad op-amp chips, U3 and U4. The outputs of the D/A converters are amplified, and the analog outputs appear on Wago connectors J1 and J2. The input comes on the PLCBus from the program running on the controller.

Figure 19-3 illustrates the operation of the D/A conversion.

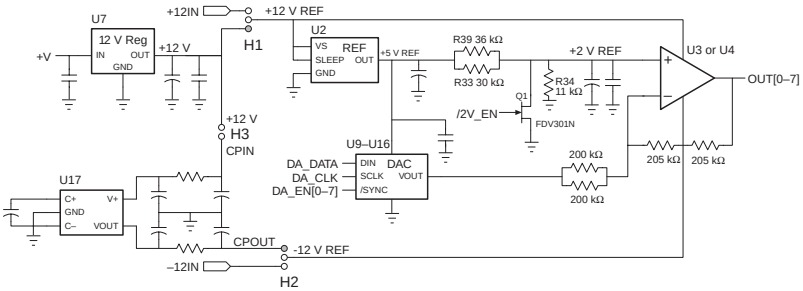


Figure 19-3. Schematic Illustration of D/A Conversion in XP8900 Series

The analog outputs do not need any special configuration. The desired analog output voltage is set using the software drivers.

The XP8900 Series expansion boards derive +5 V digital power from the +24 V PLCBus supply via LM7805 at U6. When operating without user-supplied external voltage rails, the XP8900 Series D/A converters get their +12 V analog power from the PLCBus +24 V supply via LM7812 at U7. Charge pump NJU7662 at U17 inverts this for onboard -12 V analog power. Precision +5 V and + 2 V reference voltages are derived from the +12 V supply via REF195 at U2 and the voltage divider formed from R33, R34, and R39. The n-channel FET FDV301N at Q1 is used to switch the 2 V reference to 0 V during a power-on reset.

The XP8900 Series D/A converters have the capability of receiving their +12 V or -12 V supply from an external source. This provides for greater control of electrical noise in the analog output signals.

The XP8900 Series D/A converters may be used with 12 V controllers only when ±12 V is supplied externally to pins 7 and 9 of Wago connector J2. Remember to set the jumpers on headers H1, H2, and H3 as shown in Figure 19-2.

CHAPTER 20: SOFTWARE REFERENCE

Chapter 4 describes the Dynamic C functions used to initialize the XP8600 and XP8900 Series expansion boards and to control the resulting analog outputs. The following major sections are included.

- Expansion Board Addresses
- XP8900 Series Software

Expansion Board Addresses

XP8900 Series

Up to eight XP8900 Series expansion boards may be addressed over a single PLCBus using a logical address of 0 to 7.

The 12-bit address of a particular XP8900 is determined by the jumper setting on header P3. P3 may be set eight different ways. The unique physical address is in the form

0010 000x yzRR

where

x = 1 when P3 pins 1–2 are not connected
y = 1 when P3 pins 3–4 are not connected
z = 1 when P3 pins 5–6 are not connected

and RR is reserved for the registers. There are no PAL codes.

The 12-bit address can be placed on the bus using 4-bit addressing. The functions **`set12adr`**, **`read12data`**, and **`write12data`** (in **`DRIVERS.LIB`**) use 12-bit bus addresses.

When the address is passed to **`set12adr`**, it should be in the format

yzRR 000x 0010

where the least significant nibble in the physical address, yzRR, has swapped places with the most significant nibble in the physical address, 0010.

XP8900 Series Software

This section describes a set of simple software functions to use when controlling the XP8900 Series expansion boards.

Dynamic C Libraries

Several Dynamic C function libraries need to be used with the routines defined in this section. The chart in Table 20-1 identifies which libraries must be used with particular Z-World controllers.

Table 20-1. Dynamic C Libraries Required by Z-World Controllers for XP8900 Series Expansion Boards

Library Needed	Controller
EZIOCMMN.LIB	All controllers
EZIOPBDV.LIB	All controllers
EZIOTGPL.LIB	BL1000
EZIOLGPL.LIB	BL1100
EZIOMGPL.LIB	BL1400, BL1500
EZIOPLC.LIB	BL1200, BL1600, PK2100, PK2200
EZIOPLC2.LIB	BL1700
EZIOBL17.LIB	BL1700

Before using one of these libraries in an application, first include the library name in a **#use** command. For example, to use functions in the library **EZIOPLC.LIB**, be sure there is a line at the beginning of the program in the following format.

```
#use ezioplc.lib
```

The **#use eziopbdv.lib** already included in other library calls for the XP8900 Series expansion boards, and does not have to be repeated.

Using Digital-to-Analog Converter Boards

The follow steps summarize how to use the D/A converter boards.

1. Send a reset command to the PLCBus.
2. Place the address of the D/A converter on the PLCBus.
3. Send data serially to one of the D/A converters (Register A). When Register A is filled, load the data to D/A converter Register B where it is converted and output.
4. Use the board's analog output to control motors, attenuators or other analog devices.

These steps are done using software drivers in Dynamic C function libraries.

Reset Boards on PLCBus

These Dynamic C functions are used to initialize the PLCBus. Use these functions in a program before introducing any code to operate the relays.

- **`VdInit()`**

  Initializes the timer mechanism.

  LIBRARY: **`VDRIVER.LIB`**

- **`void plcBusReset()`**

  Resets all expansion boards connected to the PLCBus.

  When using this function, initialize timers with **`VdInit()`** before resetting the PLCBus. All PLCBus devices must reset before performing any subsequent operations.

  LIBRARY: **`EZIOPBDV.LIB`**

The XP8900 output voltages cannot be reset by resetting the PLCBus. The rest of this chapter provides information on setting or resetting the XP8900 output voltages.

- **`void eioPlcRstWait()`**

  Provides a delay long enough for the PLCBus to reset.

  This function provides a delay of 1–2 seconds to ensure devices on the PLCBus reset. Call this function after resetting the PLCBus.

  LIBRARY: **`EZIOPBDV.LIB`**

- **`long int eioErrorCode`**

  Represents a global bit-mapped variable whose flags reflect error occurrences.

  This register for this variable is initially set to 0. If the application tries to access an invalid channel, the flag **`EIO_NODEV`** (the first bit flag) is set in this register. Note that the other bits in **`EIO_NODEV`** deal with networked controllers.

Address Target Board

- **`int plcXP89Init( int Addr )`**

  Initializes XP8900 Series board. Call this function before using the other **`plcXP89`**... functions. This function also initializes the XP8900 Series D/A converters to tristate their outputs. Call **`plcXP89Sw`** to turn the voltage reference on. The first **`plcXP89Out`** call enables the output of the corresponding D/A converter channel. Both the voltage reference and the D/A converter channel must be set up correctly to get the proper output.

  PARAMETER: **`Addr`** is the logical address, 0–7, of the board set by jumpers.

  RETURN VALUE: –1 if the board cannot be found, 0 if the initialization is completed.

  LIBRARY: **`EZIOPBDV.LIB`**

```
void main(void){
   plcBusReset();    // reset the PLCBus
   if(plcXP89Init(4)){
   ...
   } else {
   ...
   }
}
```

Operate Target Board

- **`int plcXP89Sw( int Addr, int state )`**

  Turns the D/A converters and references to op-amps on or off. Note that all channels on a particular board are switched at the same time.

  PARAMETERS: **`Addr`** is the logical address, 0–7, of the board set by jumpers. Both the reference (switched on by this call) and the D/A converter output (switched off by this call, switched on by **`plcX89Out`**) must be set correctly to get the proper output.

  `state` indicates whether the D/A converter and reference voltage should be turned on or off. The reference is turned on when **`state`** is nonzero. Otherwise the D/A converters will tristate and the reference will output 0. The output voltage of all channels should be approximately 0 at the op-amp when the D/A converter is off.

  RETURN VALUE: –1 if the board cannot be found, 0 if the operation is completed.

  LIBRARY: **`EZIOPBDV.LIB`**

  The XP8900 output voltages may fluctuate to 2 V for each channel while **`plcX89Sw`** is executing to turn on the op-amp reference and to switch off the D/A converter.

- **`int plcXP89Out( int Addr, unsigned int oValue )`**

  Sends the 12-bit **`oValue`** to the proper D/A converter channel. Call **`plcXP89Init`** and **`plcXP89Sw`** before calling **`plcXP89Out`**. Note that **`plcXP89Out`** does not switch the voltage reference on or off. Both the D/A converter and the voltage reference must be set up correctly to get the proper voltage output. **`plcXP89Sw`** enables the voltage reference.

  PARAMETERS: **`Addr`** is *8*board_number + channel_number*. Note that *board_number* and *channel_number* start from zero. *board_number* ranges from 0 to 7 as set by the address jumpers. *channel_number* ranges from 0 to 7 (XP8900), or from 0 to 3 (XP8910).

  `oValue` is the 12-bit value to send to the D/A converter.

  RETURN VALUE: –1 if the D/A converter cannot be found, 0 if the operation is successful. If the D/A converter does not exist, this function also bit-ors the constant **`EIO_NODEV`** to **`eioErrorCode`**.

  LIBRARY: **`EZIOPBDV.LIB`**

```
plcXP89Out(42,2048)
      // make channel 2 on board 5 output about 0 V
```

Table 8-2 summarizes these three functions. The order in which they appear in Table 8-2 is the sequence in which they should be used to start an XP8900 Series board.

Table 20-2. Summary of Basic XP8900 Series Function Calls

Function	Description
plcXP89Init	Disables everything, leaves output of 0 V for all channels
plcXP89Sw	Enables voltage reference so the output will be at the voltage level specified by **plcXP89Out**
plcXP89Out	Sets all channels to midpoint or other acceptable value (the output experiences a slight jump as channels are being set; remember to set all 4 or 8 channels since one call sets only one channel)

- **int plcXP89WrCalib(int chan, struct _eioAdcCalib *pCalib)**

  Writes a calibration structure to the EEPROM storage corresponding to a channel on the XP8900 Series board.

  PARAMETERS: **chan** is the channel number, 0–63, of the XP8900 Series D/A channel. **chan** = *8*board_number + channel_number*.

  _eioAdcCalib *pCalib is a pointer to a calibration structure initialized by calling **eioAdcMakeCoeff**.

  RETURN VALUE: 0 if the calibration is successful, otherwise returns a negative number.

  LIBRARY: **EZIOPBDV.LIB**

```
plcXP89WrCalib(15,&cstruct)
     // write calib info in cstruct to channel 7 of
     // XP8900 Series board 1
```

- **`int plcXP89RdCalib( int chan, struct _eioAdcCalib *pCalib )`**

  Reads the calibration structure of a D/A channel from an XP8900 Series board.

  PARAMETERS: **`chan`** is the channel number, 0–63, of the XP8900 Series D/A channel. **`chan`** = *8*board_number + channel_number*.

  `_eioAdcCalib *pCalib` is a pointer to a calibration structure. Use **`eioAdcDigitize`** to compute the actual D/A output of a given analog value.

  RETURN VALUE: 0 if the operation is successful, otherwise returns a negative number.

  LIBRARY: **`EZIOPBDV.LIB`**

```
plcXP89RdCalib(32,&cinfo)
    // read calib info of channel 0 of XP8900
    // XP8900 Series board 4 into cinfo
```

- **`int eioAdcMakeCoeff( struct _eioAdcCalib *cnvrsn, unsigned d1, unsigned d2, float f1, float f2 )`**

  Takes the raw values and actual values of two data points, then computes the calibration coefficients (assumes linearity).

  PARAMETERS: **`struct _eioAdcCalib *cnvrsn`** is a pointer to a calibration structure that stores the coefficients.

  `d1` is the raw (quantized) value of the first data point.

  `d2` is the raw (quantized) value of the second data point.

  `f1` is the actual (real) value (in volts) of the first data point.

  `f2` is the actual (real) value (in volts) of the second data point.

  RETURN VALUE: –1 if it is not possible to compute the calibration coefficients, otherwise 0.

  LIBRARY: **`EZIOPBDV.LIB`**

```
eioAdcMakeCoeff(&cinfo,96,4000,9.97,-10.33)
   // the actual value at quantized value 96 is 9.97 V
   // the actual value at quantized value 4000 is -10.03 V
   // compute the coefficients and put into cinfo
```

- **long eioAdcDigitize(float f,**
  struct _eioAdcCalib *pCalib)

  Converts analog value to digital number according to calibration coefficients. This function is used to convert an analog value such as voltage to the actual digital number for a D/A converter device.

  PARAMETERS: **f** is the analog value to output.

  _eioAdcCalib *pCalib is a pointer to a structure that stores the calibration coefficients.

  RETURN VALUE: Long integer that corresponds to the number to send to a D/A converter device.

  LIBRARY: **EZIOPBDV.LIB**

```
L=eioAdcDigitize(2.54,&cinfo);
   // L will contain the digitized value to output
   // to D/A converter device given the
   // calibration coefficients in cinfo so that
   // the output is about 2.54 of some real units
```

Sample Program

The sample program **XP89_1.C** in the Dynamic C **SAMPLES\PLCBUS** subdirectory demonstrates how to calibrate the D/A converter channels.

The basic sample program is designed for the BL1200, BL1600, PK2100, and PK2200 controllers. Remember to uncomment the lines that apply to the controller being used with the XP8900 Series expansion board.

To use this program properly, it may be necessary to edit the statements that initialize the channel, margin, **f1**, and **f2**. The program may also be compiled as is, with watch expressions added to override the assignment statements (be sure to execute the watch expression AFTER the assignment statement is executed).

Use the following steps to run the sample program.

1. Compile the program by pressing **F3** or by choosing **Compile** from the **COMPILE** menu. Dynamic C compiles and downloads the program into the controller's memory. During compilation, Dynamic C rapidly displays several messages in the compiling window, which is normal.
2. Run the program by pressing **F9** or by choosing **Run** from the **RUN** menu. It is also possible to single-step through the program with **F7** or **F8**.
3. To halt the program, press **<CTRL-Z>**.
4. To restart the program, press **F9**.

Check the board jumpers, PLCBus connections, and the PC/controller communications if an error message appears.

See the Dynamic C ***Technical Reference*** manual for more detailed instructions.

XP89_1.C

```
#use eziocmmn.lib
/* #use ezioplc.lib  // for BL1200, BL1600, PK2100, PK2200 */
/* #use eziotgpl.lib // for BL1000 */
/* #use eziolqpl.lib // for BL1100 */
/* #use eziomgpl.lib // for Bl1400 & BL1500 */
/* #use eziobl17.lib // for BL1700 */
/* #use ezioplc2.lib // for BL1700 */

main() {
   auto int i;
   auto struct _eioAdcCalib c;
   auto int channel;
   auto float f1, f2, fout;
   auto long l;
   auto int margin;
   channel = 0;      // execute watch expression
        // to override
   margin = 0x40;    // execute watch expression
        // to override
   plcBusReset();
   if (plcXP89Init(channel / 8)) {
      printf("DAC8 board not found\n");
   } else {
      plcXP89Sw(channel / 8,1);
             // enable voltage reference
      plcXP89Out(channel,margin);
             // use meter to record level
      f1 = 10;       // use watch expr to override
      plcXP89Out(channel,0xfff-margin);
             // use meter to record level
      f2 = -10;      // use watch expr to override
      eioAdcMakeCoeff(&c,margin,0xfff-margin,f1,f2);
      if (plcXP89WrCalib(channel,&c)) {
        printf("Can't write calibration constant\n");
      }
      memset(&c,0,sizeof(struct _eioAdcCalib));
      if (plcXP89RdCalib(channel,&c)) {
        printf("Can't read calibration constant\n");
      }
      fout = 2.345;  // use watch expr to override
      l = eioAdcDigitize(fout, &c);
      plcXP89Out(channel,(unsigned)l);
                     // use meter to check voltage now
   }
}
```

APPENDICES

Appendices

APPENDIX A: PLCBUS

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

PLCBus Overview

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

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

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

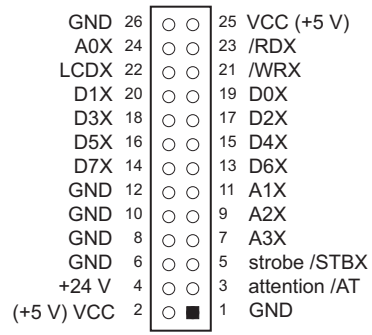


Figure A-1. PLCBus Pin Diagram

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

The LCD bus consists of the following lines.

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

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

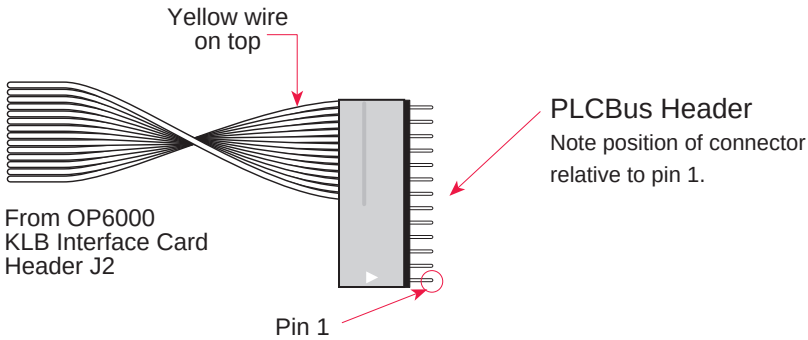


Figure A-2. OP6000 Connection to PLCBus Header

The PLCBus consists of the following lines.

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

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

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

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

Table A-3. First-Level PLCBus Address Coding

First Byte	Mode	Addresses	Full Address Encoding
– – – – 0 0 0 0 – – – – 0 0 0 1 – – – – 0 0 1 0 – – – – 0 0 1 1	4 bits × 3	256 256 256 256	0000 xxxx xxxx 0001 xxxx xxxx 0010 xxxx xxxx 0011 xxxx xxxx
– – – x 0 1 0 0 – – – x 0 1 0 1 – – – x 0 1 1 0 – – – x 0 1 1 1	5 bits × 3	2,048 2,048 2,048 2,048	x0100 xxxxx xxxxx x0101 xxxxx xxxxx x0110 xxxxx xxxxx x0111 xxxxx xxxxx
– – x x 1 0 0 0 – – x x 1 0 0 1	6 bits × 3	16,384 16,384	xx1000 xxxxxx xxxxxx 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 xxxxx xxx
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 A-4 provides the address allocations for the registers of 4-bit devices.

Table A-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 A-5 summarizes the libraries.

Table A-5. Dynamic C PLCBus Libraries Used by Z-World Controllers

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 B: CONNECTING AND MOUNTING MULTIPLE BOARDS

Connecting Multiple Boards

Eight or more expansion boards can be connected ("daisy chained") at one time. The actual number of expansion boards may be limited by capacitative loading on the PLCBus.

Be sure that each expansion board has a unique address to prevent communication problems between the controller and the expansion board.

Follow these steps to install several expansion boards on a single PLCBus.

1. Place all expansion boards right side up.
2. Use the ribbon cable supplied with the boards.
3. Connect one board to the main controller.
4. Connect another expansion board to the first expansion board, connecting each board's header P1 to the adjacent board's header P2.

Figure B-1 illustrates a controller with expansion boards attached.

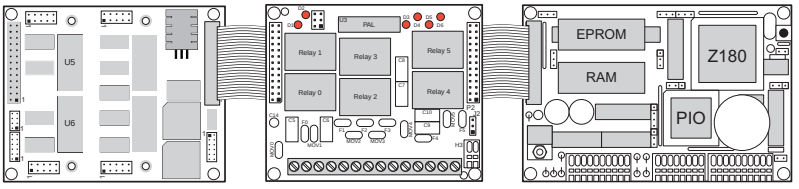


Figure B-1. Connecting Multiple Expansion Boards

> *Do not twist the ribbon cable or mount the expansion boards upside down!* Damage may occur. Be sure Pin 1 of P1 and P2 of each board matches up with Pin 1 of the previous board. Pin 1 should be at the lower right when the expansion board is right side up, that is, the board markings are right side up.

When several expansion boards are connected, there may be a voltage drop along the network of expansion boards. No action is necessary as long as the digital voltage, VCC, is greater than 4.9 V on the last board.

VCC can be measured at pin 2 on header P1, and GND is pin 1 on header P1.

There are two ways to compensate for the voltage dropoff. The easiest way is to connect +5 V DC and ground from the host controller to pins 2 and 1 of header P1 on the last expansion board. Another solution, which can approximately double the number of boards that could otherwise be connected to a single controller, is a Y cable available from Z-World. Figure B-2 illustrates the use of the Y cable.

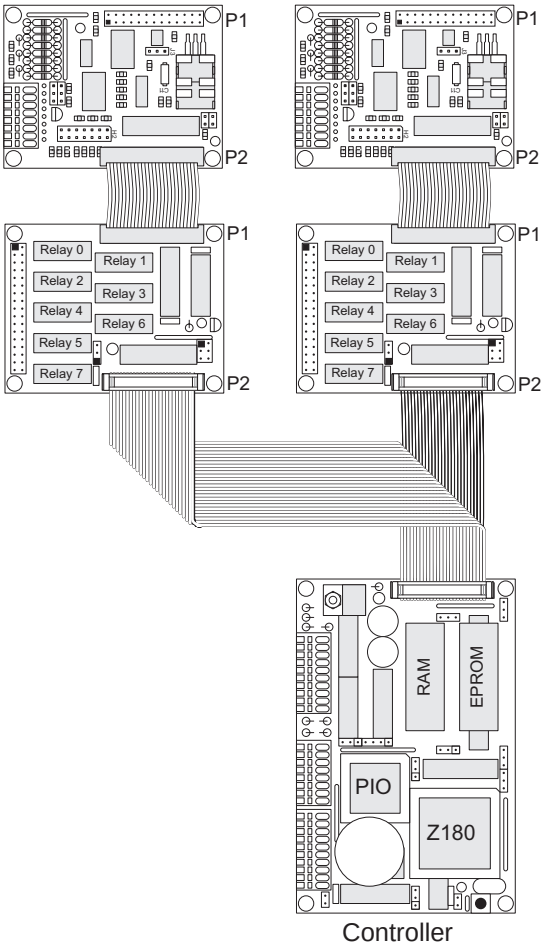


Figure B-2. Use of Y Cable to Connect Multiple Expansion Boards

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

Mounting

Expansion boards can be installed in modular plastic circuit-board holders attached to a DIN rail, a widely used mounting system, as shown in Figure B-3.

The circuit-board holders are 77 mm wide and come in lengths of 11.25 mm, 22.5 mm , and 45 mm. The holders, available from Z-World and from other suppliers, snap together to form a tray of almost any length. Z-World's expansion boards are 72 mm wide and fit directly in these circuit-board holders.

Z-World's expansion boards can also be mounted with plastic standoffs to any flat surface that accepts screws. The mounting holes are 0.125 inches (1/8 inch) in from the edge of a board, and have a diameter of 0.190 inches.

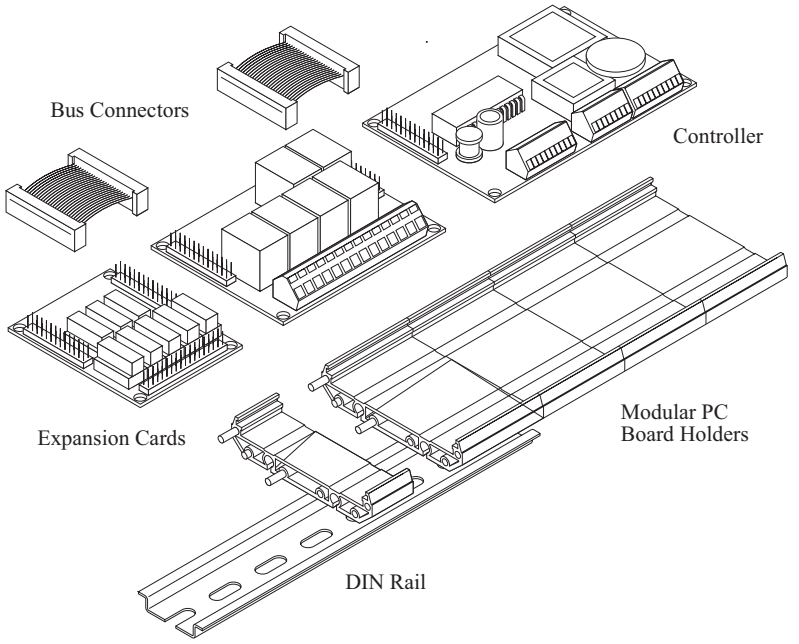


Figure C-3. Mounting Expansion Boards on DIN Rail

For information on ordering DIN rail mounts, call your Z-World Sales Representative at (530) 757-3737.

APPENDIX C:
SIMULATED PLCBUS CONNECTIONS

Some Z-World controllers do not have a PLCBus, but signals on their configurable PIO ports or KIO ports are those that would be available on a PLCBus. Appendix C provides the hookup information to allow expansion boards to be used with these controllers.

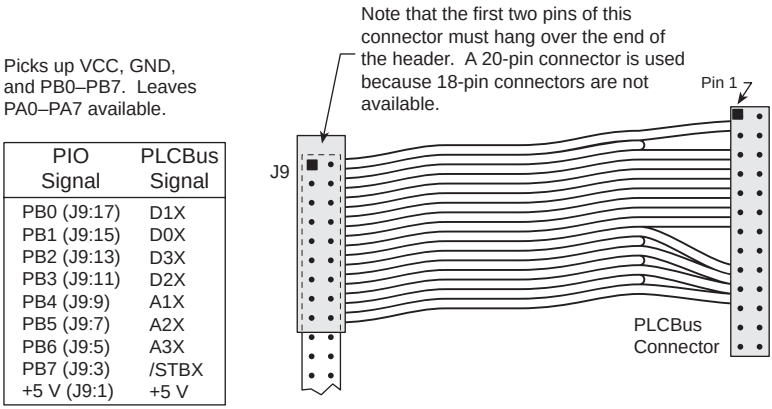
Table C-1 provides a list of which expansion boards may be used with which non-PLCBus controllers.

Table C-1. Expansion Board Compatibility with non-PLCBus Controllers

Expansion Board	Z-World Controller				
	BL1000	BL1100	BL1300	BL1400	BL1500
XP8100					
XP8300	X	X	X	X	X
XP8500				X	X
XP8800					
XP8900				X	X

BL1000

The XP8300 expansion board may be connected to a BL1000 using an expander cable (Z-World part number 540-0015). Fasten the cable's 20-pin connector to header J9 as shown in Figure C-1. Pins 1 and 2 of the connector must hang over the end of the header. 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
PB0 (J9:17)	D1X
PB1 (J9:15)	D0X
PB2 (J9:13)	D3X
PB3 (J9:11)	D2X
PB4 (J9:9)	A1X
PB5 (J9:7)	A2X
PB6 (J9:5)	A3X
PB7 (J9:3)	/STBX
+5 V (J9:1)	+5 V

Figure C-1. BL1000 Expander Cable Connection

Software for interfacing the BL1000's PIO port to a PLCBus port may be found in the Dynamic C **PBUS_TG.LIB** library.

Use an external power supply with expansion boards connected to the BL1000. There is no provision in the special cable to supply +24 V from the controller to header P1 or P2 on the expansion boards.

BL1100

The XP8300 expansion boards may be connected to a BL1100 using an expander cable (Z-World part number 540-0015). Fasten the cable's 20-pin connector to the combined headers J010 and J10 as shown in Figure C-2. Pins 1 and 2 of the expander cable connector must hang over the end of the combined headers. Fasten the cable's PLCBus connector to XP8200 header P1 or P2. Note the orientation of pin 1.

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

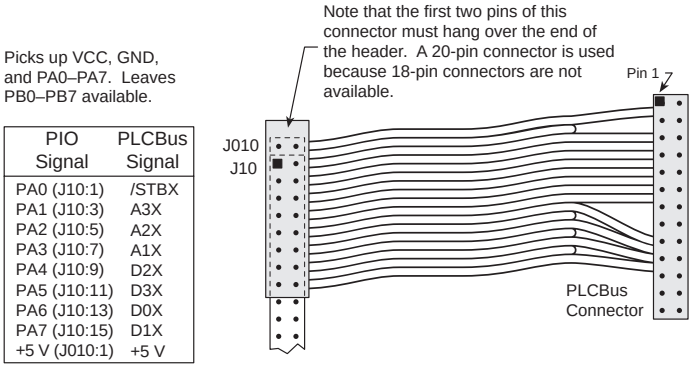


Figure C-2. BL1100 Expander Cable Connection

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

BL1300

The XP8300 expansion board may be connected to header P5 on the BL1300 using the same special cable used to connect them to the BL1000 or to the BL1100, as shown in Figure D-2. The first two pins of the special cable hang over the end of header P5 as before. However, the wire leading to pin 1 on the BL1300's header P5 must be cut, and may then be used to supply +5 V from an external source to the expansion board. Software from the Dynamic C **PBUS_LG.LIB** library may be used.

Use an external power supply with expansion boards connected to the BL1300. There is no provision in the special cable to supply +24 V from the controller to header P1 or P2 on the expansion boards.

BL1400 or BL1500

XP8300, XP8500, and XP8900 expansion boards may be connected to header H3 on either the BL1400 or the BL1500. To add these expansion boards, the user must either make a custom cable or use an adapter board (Z-World part number 101-0050). To assist with making the connection via a ribbon cable, Table C-2 maps the signals from the controller's PIO to the expansion board. Dynamic C's **EZIOMGPL.LIB** library may be used for programming.

Table C-2. PIO to PLCBus Signal Map

BL1400/Bl1500		Expansion Board	
H3 Pin No.	**PIO Port Signal**	**Pin No.**	**PLCBus Signal**
1	VCC (+5 V)	2	VCC (+5 V)
2	PA0	5	/STBX
3	PA1	19	D0X
4	PA2	20	D1X
5	PA3	17	D2X
6	PA4	18	D3X
7	PA5	11	A1X
8	PA6	9	A2X
9	PA7	7	A3X
10	GND	10	GND

The adapter board provides an easy way to add an expansion board to either BL1400 or BL1500 controllers. Power is supplied to the controller via the power jack and to the expansion board via a screw terminal. For specifics on how to install an adapter board with a specific controller, see that controller's user's manual.

Use an external power supply with expansion boards connected to the BL1400 or BL1500 because there is no provision to supply power from the controller to header P1 or P2 on the expansion boards. The adapter board has a jack and a screw terminal for the external +12 V/+24 V.

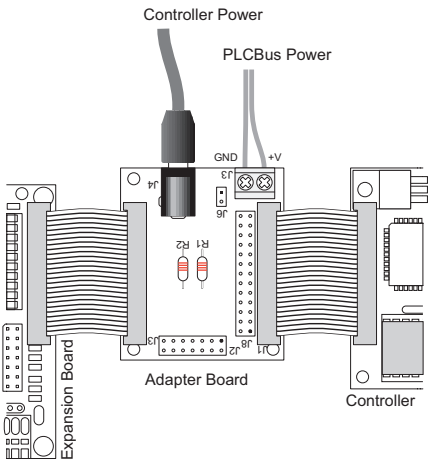


Figure C-3. Adapter Board Connections

APPENDIX D: PLCBUS STATES

PLCBus State Tables

This appendix is provided for advanced programmers who wish to write their own application drivers for the XP8100 expansion boards, and require detailed information about PLCBus cycles.

Two state tables are provided. Table D-1 presents the PLCBus states, and Table D-2 describes what state the PLCBus transitions to from a given input.

Table D-1. State Definitions

State	State Number (see Table G-2)
0	Board not selected
1	BUSADR0 recognized
2	BUSADR1 recognized
3	BUSADR2 recognized for ID mode
4	BUSADR2 recognized for Group 0
5	BUSADR2 recognized for Group 1
6	BUSADR2 recognized for Group 2
7	BUSADR2 recognized for Group 3

Reading State Table D-2

The letters **pqr** represent the binary version of the jumper-set address of the XP8100 Series board (**r** is the least-significant bit). The letters **xyz** represent the group number for data from an I/O channel, and **efgh** represent the data bits D3–D0.

To use Table D-2, read across the "state" row to the current state of the PLCBus. Then read down the "action" column to the particular PLCBus cycle to be performed in that state. The number corresponding to the next state that the PLCBus will transition to is at the intersection of the row and column. Some of the cases also have a superscripted reference to a note that explains how to interpret the value returned.

Table D-2. PLCBus State Table

Action	State							
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
BUSADR0 ←**0001**	1	1	1	1	1	1	1	1
BUSADR1 ←**00pq**	0	2	2	2	2	2	2	2
BUSADR2 ←**r000**	0	1	3	3	3	3	3	3
BUSADR2 ←**r100**	0	1	4	4	4	4	4	4
BUSADR2 ←**r101**	0	1	5	5	5	5	5	5
BUSADR2 ←**r110**	0	1	6	6	6	6	6	6
BUSADR2 ←**r111**	0	1	7	7	7	7	7	7
BUSWR ←**efgh**	0	1	2	3	4[b]	5[b]	6[b]	7[b]
BUSDR0 →**efgh**	0	1	2	3[a]	4[c]	5[c]	6[c]	7[c]
BUSDR1 →**efgh**	0	1	2	3	4[d]	5[d]	6[d]	7[d]
BUSADR0 ←! **0001**	0	0	0	0	0	0	0	0
BUSADR1 ←! **01pq**	0	1	1	1	1	1	1	1
BUSADR2 ←! **rxyz**	0	1	2	2	2	2	2	2

Notes

(a) **h**=**0** indicates an XP8100 exists; **h**=**1** indicates there is no XP8100 Series board at this address.

(b) **h**=**0** indicate off, **1** indicates on, **efg** specifies which of the 8 output channels in the group is selected.

(c) **h** indicates the state of the zeroth (lowest number) input channel in the group of 8, **e** indicates the state of the third input channel in the group of 8.

(d) **h** indicates the state of the fourth input channel in the group of 8, **e** indicates the state of the seventh input channel in the group of 8.

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

O

P

Q

R

S

T

U

V

W

X

Y

SCHEMATICS