# Table of Contents

**About This Manual** vii

**Chapter 1: Overview** 11
  - Introduction ................................................................. 12
  - Standard Features ......................................................... 14
  - Flexibility and Customization Options .......................... 15
  - Development Kit ......................................................... 15
  - CE Compliance ............................................................. 16

**Chapter 2: Getting Started** 17
  - Connecting the PK2200 to a PC ......................................... 18
  - Establishing Communication with the PK2200 .................. 19
  - Running a Sample Program .......................................... 20

**Chapter 3: Subsystems** 21
  - Subsystem Overview ...................................................... 22
  - Processor Core ............................................................. 23
    - CPU ............................................................................. 23
    - Microprocessor Supervisor/Watchdog Timer ................. 23
    - Static RAM .............................................................. 23
    - EPROM/Flash EPROM ............................................ 23
    - EEPROM ............................................................... 24
    - Real Time Clock (RTC) ............................................ 24
    - Digital Inputs .......................................................... 24
    - Digital Outputs ....................................................... 27
    - Serial Communication ............................................. 29
      - Serial Channel Configuration ................................. 29
    - Keypad and Display ................................................. 31

**Chapter 4: System Development** 33
  - Changing Modes .......................................................... 34
  - Setting the Mode ........................................................ 34
  - Development Options ................................................ 35
    - Memory Options ...................................................... 35
    - Battery-Backed RAM ............................................. 35
    - EPROM ................................................................. 36
    - Flash EPROM ........................................................ 36
| Digital Inputs ................................................................. | 36  |
| Using the Digital Inputs ............................................... | 36  |
| Interrupt Inputs ............................................................. | 36  |
| High Speed DMA Counter .................................................. | 36  |
| Digital Outputs ............................................................... | 37  |
| Using the Digital Outputs .................................................. | 37  |
| Serial Communication ....................................................... | 38  |
| Receive and Transmit Buffers ............................................. | 38  |
| Echo Option ....................................................................... | 39  |
| CTS/RTS Control ................................................................... | 39  |
| XMODEM File Transfer ....................................................... | 39  |
| Modem Communication ....................................................... | 39  |
| Interrupt Handling for Z180 Port 0 .................................... | 41  |
| Remote Downloading ............................................................ | 41  |
| Developing an RS-485 Network ............................................. | 42  |
| Keypad and LCD .................................................................. | 43  |
| Using the Keypad and Display ............................................. | 43  |
| PK2200 Keypads ................................................................. | 43  |
| Keypad Insert Templates .................................................... | 44  |
| Keypad Codes ..................................................................... | 45  |
| PK2200 LCDs ....................................................................... | 46  |
| Graphic LCD Status ............................................................ | 46  |
| Bitmapped Graphics ............................................................ | 47  |

**Chapter 5: Software Reference**

- Software Drivers ................................................................. 50
- Real Time Clock (RTC) ......................................................... 50
- EEPROM ................................................................................. 51
- Digital Inputs and Outputs .................................................... 52
- Digital Input Drivers ............................................................. 52
- Digital Output Drivers .......................................................... 54
- LCD and Keypad ................................................................. 55
- Sample Programs ................................................................. 62
- Communication Sample Programs ........................................ 63
- PK2240 Sample Programs ..................................................... 64

**Appendix A: Troubleshooting**

- Out of the Box ................................................................. 66
- Dynamic C Will Not Start ..................................................... 67
- Dynamic C Loses Serial Link ............................................... 67
- PK2200 Repeatedly Resets .................................................. 67
- Common Programming Errors ............................................ 68
<table>
<thead>
<tr>
<th>Appendix B: Specifications</th>
<th>69</th>
</tr>
</thead>
<tbody>
<tr>
<td>General Specifications</td>
<td>70</td>
</tr>
<tr>
<td>Hardware Mechanical Dimensions</td>
<td>71</td>
</tr>
<tr>
<td>High Voltage Driver Specifications</td>
<td>75</td>
</tr>
<tr>
<td>Environmental Temperature Constraints</td>
<td>75</td>
</tr>
<tr>
<td>Connectors</td>
<td>76</td>
</tr>
<tr>
<td>Header Locations and Jumper Settings</td>
<td>76</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Appendix C: Power Management</th>
<th>79</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power Failure Detection Circuitry</td>
<td>80</td>
</tr>
<tr>
<td>Power Failure Sequence of Events</td>
<td>80</td>
</tr>
<tr>
<td>Recommended Power Fail Routine</td>
<td>82</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Appendix D: Interrupt Vectors and I/O Addresses</th>
<th>85</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt Vectors</td>
<td>86</td>
</tr>
<tr>
<td>Jump Vectors</td>
<td>87</td>
</tr>
<tr>
<td>EEPROM Addresses</td>
<td>88</td>
</tr>
<tr>
<td>Processor Register Addresses</td>
<td>89</td>
</tr>
<tr>
<td>PK2200 Peripheral Addresses</td>
<td>91</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Appendix E: PLCBus</th>
<th>95</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLCBus Overview</td>
<td>96</td>
</tr>
<tr>
<td>Allocation of Devices on the Bus</td>
<td>100</td>
</tr>
<tr>
<td>4-Bit Devices</td>
<td>100</td>
</tr>
<tr>
<td>8-Bit Devices</td>
<td>101</td>
</tr>
<tr>
<td>Expansion Bus Software</td>
<td>101</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Appendix F: Backup Battery</th>
<th>107</th>
</tr>
</thead>
<tbody>
<tr>
<td>Battery Life and Storage Conditions</td>
<td>108</td>
</tr>
<tr>
<td>Replacing Soldered Lithium Battery</td>
<td>108</td>
</tr>
<tr>
<td>Battery Cautions</td>
<td>109</td>
</tr>
</tbody>
</table>

Index 111
Blank
This manual provides instructions for installing, testing, configuring, and interconnecting the Z-World PK2200 controller.

All product references in this manual are made to the PK2200 series. The term “PK2200” is used as a generic term referring to any of the PK2200 series. Where necessary, specific model numbers are used.

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 a PK2200 will control.

• Understanding of the basics of operating a software program and editing files under Windows on a PC.

• Knowledge of the basics of C programming.

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

*The C Programming Language* by Kernighan and Ritchie

and/or

*C: A Reference Manual* by Harbison and Steel

• Knowledge of basic Z80 assembly language and architecture.

For documentation from Zilog, refer to the following texts.

*Z180 MPU User's Manual*

*Z180 Serial Communication Controllers*

*Z80 Microprocessor Family User's Manual*
Acronyms

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

Table 1. Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPROM</td>
<td>Erasable Programmable Read-Only Memory</td>
</tr>
<tr>
<td>EEPROM</td>
<td>Electronically Erasable Programmable Read-Only Memory</td>
</tr>
<tr>
<td>LCD</td>
<td>Liquid Crystal Display</td>
</tr>
<tr>
<td>LED</td>
<td>Light-Emitting Diode</td>
</tr>
<tr>
<td>NMI</td>
<td>Nonmaskable Interrupt</td>
</tr>
<tr>
<td>PIO</td>
<td>Parallel Input/Output Circuit</td>
</tr>
<tr>
<td></td>
<td>(Individually Programmable Input/Output)</td>
</tr>
<tr>
<td>PRT</td>
<td>Programmable Reload Timer</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
<tr>
<td>RTC</td>
<td>Real-Time Clock</td>
</tr>
<tr>
<td>SIB</td>
<td>Serial Interface Board</td>
</tr>
<tr>
<td>SRAM</td>
<td>Static Random Access Memory</td>
</tr>
<tr>
<td>UART</td>
<td>Universal Asynchronous Receiver Transmitter</td>
</tr>
</tbody>
</table>

Icons

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

Table 2. Icons

<table>
<thead>
<tr>
<th>Icon</th>
<th>Meaning</th>
<th>Icon</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>Refer to or see</td>
<td>!</td>
<td>Note</td>
</tr>
<tr>
<td>$</td>
<td>Please contact</td>
<td>$</td>
<td>Tip</td>
</tr>
<tr>
<td></td>
<td>Caution</td>
<td>$</td>
<td>Tip</td>
</tr>
<tr>
<td></td>
<td>Factory Default</td>
<td>$</td>
<td>High Voltage</td>
</tr>
</tbody>
</table>
Conventions

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

Table 3. Typographic Conventions

<table>
<thead>
<tr>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>while</code></td>
<td>Courier font (bold) indicates a program, a fragment of a program, or a Dynamic C keyword or phrase.</td>
</tr>
<tr>
<td><code>// IN-01...</code></td>
<td>Program comments are written in Courier font, plain face.</td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Indicates that something should be typed instead of the italicized words (e.g., in place of <code>filename</code>, type a file’s name).</td>
</tr>
<tr>
<td><em>Edit</em></td>
<td>Sans serif font (bold) signifies a menu or menu selection.</td>
</tr>
<tr>
<td>...</td>
<td>An ellipsis indicates that (1) irrelevant program text is omitted for brevity or that (2) preceding program text may be repeated indefinitely.</td>
</tr>
<tr>
<td><code>[ ]</code></td>
<td>Brackets in a C function’s definition or program segment indicate that the enclosed directive is optional.</td>
</tr>
<tr>
<td><code>&lt; &gt;</code></td>
<td>Angle brackets occasionally enclose classes of terms.</td>
</tr>
<tr>
<td>`a</td>
<td>b</td>
</tr>
</tbody>
</table>

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.
Blank
Chapter 1 provides a comprehensive overview and description of the PK2200.
**Introduction**

The PK2200 is an inexpensive control computer well suited for a variety of applications in areas such as packaging, materials handling, and process control.

Figure 1-1 illustrates the PK2200 with the $2 \times 20$ character LCD and a $2 \times 6$ keypad.

Figure 1-1. PK2200 with Character LCD and Keypad

Figure 1-2 illustrates the PK2240 with the $128 \times 64$ graphic LCD and a $4 \times 3$ keypad.

Figure 1-2. PK2240 with Graphic LCD and Keypad
Figure 1-3 illustrates the PK2200 without an enclosure.
Standard Features

The PK2200 series includes the following standard features:

- Compact size: 4" × 5.5" × 1.34"
- 16 protected digital inputs for detecting contact closures, counting pulses, or detecting voltage input signals.
- 14 high-current digital outputs, suitable for driving relays, solenoids, or lamps.
- RS-485 and RS-232 serial ports for external communication and controller networking using links up to several kilometers
- 9.216 MHz clock with 18.432 MHz optional
- Switching power supply for reduced power consumption. The PK2200 consumes less than 2 W at 18.432 MHz.
- A PLCBus port allows system expansion including relays, A/D converters, D/A converters, UARTs and more.
- EPROM (up to 512K) or flash EPROM (up to 256K) for program and nonvolatile data storage.
- Battery-backed RAM (up to 512K).
- Battery-backed real-time clock with time and date functions.
- Programmable timers.
- EEPROM (512 byte standard) for storing system information.
- Watchdog timer and power-fail detection circuitry for improved system reliability.

Table 1-1 lists PK2200 series models and each model’s standard features.

<table>
<thead>
<tr>
<th>Model</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>PK2200</td>
<td>18.432 MHz clock, 2 × 20 character LCD, 2 × 6 keypad, rugged metal enclosure.</td>
</tr>
<tr>
<td>PK2210</td>
<td>9.216 MHz clock, 2 × 20 character LCD, 2 × 6 keypad, rugged metal enclosure.</td>
</tr>
<tr>
<td>PK2220</td>
<td>18.432 MHz clock, 2 × 20 character LCD, 2 × 6 keypad</td>
</tr>
<tr>
<td>PK2230</td>
<td>9.216 MHz clock.</td>
</tr>
<tr>
<td>PK2240</td>
<td>18.432 MHz clock, 128K flash EPROM, 128 × 64 backlit graphic LCD, 4 × 3 keypad, rugged metal enclosure.</td>
</tr>
</tbody>
</table>
Flexibility and Customization Options

The PK2200 is available with either quick-release pluggable terminals or fixed screw terminals.

For added flexibility, special order the PK2200 Series controller with the following options installed.

- Backlit character LCD (for PK2200 and PK2210 only).
- 128K or 512K battery-backed RAM.
- 128K flash EPROM for program and nonvolatile data storage.
- High-voltage sourcing drivers.

For quantity orders, customization of the PK2200 modified is available to better suit your application. A wide variety of options are available for I/O, memory, and packaging.

For details on PK2200 customization, contact your Z-World Sales Representative at (530) 757-3737.

Development Kit

The PK2200 Development Kit contains all the tools required for fast development. The kit includes the following items:

- Programming cable
- Power supply
- 128K flash EPROM
- High-current sourcing drivers
- Demonstration board that simulates I/O
- User’s manual with schematics
CE Compliance

The PK2200 has been tested by an approved competent body, and was found to be in conformity with applicable EN and equivalent standards. Note the following requirements for incorporating the PK2200 in your application to comply with CE requirements.

- The power supply provided with the Development Kit is for development purposes only. It is the customer’s responsibility to provide a clean DC supply to the controller for all applications in end-products.

- Fast transients/burst tests were not performed on this controller. Signal and process control lines longer than 3 m should be routed in a separate shielded conduit.

- The PK2200, PK2210, PK2220, and PK2230 were tested to Industrial Immunity Standards. The PK2240 has been tested to Light Industrial Immunity standards. Additional shielding or filtering may be required for the PK2240 for an industrial environment.

- The PK2200 has been tested to EN55022 Class A emission standards. Additional shielding or filtering will be required to meet Class B emission standards.

Chapter 2 provides instructions for connecting the PK2200 to a PC and running a sample program.
Connecting the PK2200 to a PC

The PK2200 is programmed with a PC through an RS-232 port using the programming cable provided in the Development Kit.

To connect the PK2200 to a PC use the following steps:

1. Install Dynamic C as described in your Dynamic C manuals.

2. Using the supplied adapter, connect the programming cable from the PK2200’s RJ-12 (J2) socket to the appropriate COM port of your computer.

3. Connect the supplied 24V DC power supply as follows.
   - Connect the lead with the red sleeving to the +DC terminal of the PK2200 (J1 terminal 1).
   - Connect the other lead to the GND terminal (J1 terminal 3).

![Programming Connections Diagram]

**Figure 2-1. Programming Connections**

⚠️ Only use the supplied adapter and programming cable.

The supplied 24 V wall power supply is sufficient for all power requirements. The PK2200 accepts from 9 V to 36 V DC.
Figure 2-2 illustrates the power supply connections.

4. Plug the power supply into a wall socket.

The PK2200 is now ready to run.

**Establishing Communication with the PK2200**

To establish communication with the PK2200 use the following steps.

1. Double-click the Dynamic C icon to start the software. Note that each time you start Dynamic C, communication with the attached PK2200 is attempted.

2. If the communication attempt is successful, no error messages are displayed.

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

After making necessary changes to establish communication between a PC and the PK2200, use the Dynamic C shortcut `<Ctrl-Y>` to reset the controller and initialize communication.
Running a Sample Program

To run a sample program on the PK2200 use the following steps.

1. Open the sample program `CDEMO_RT.C` located in the `SAMPLES\CPLC` Dynamic C subdirectory.

2. Compile the program by pressing **F3** or by choosing **Compile** from the compile menu. Dynamic C compiles and downloads the program into the PK2200’s memory.

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

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

3. Run the program by pressing **F9** or by choosing **Run** from the **Run** menu.

4. To halt the program, press **<Ctrl-Z>**.

5. To restart program execution, press **F9**.

Chapter 3 describes the various PK2200 subsystems and interfaces, software drivers and sample programs.
Subsystem Overview

The PK2200 is composed of several subsystems. The following list of subsystem elements is illustrated in Figure 3-1.

- Processor core
- Protected digital inputs
- High-voltage driver outputs
- Serial communication channels
- Keypad and LCD

![Figure 3-1. PK2200 I/O Systems Block Diagram](image-url)
**Processor Core**

The PK2200’s processor core is composed of the CPU, microprocessor supervisor/watchdog timer, battery-backed static RAM, EPROM/flash EPROM, EEPROM, and RTC.

**CPU**

The PK2200 is available with either 9.216 MHz or 18.432 MHz CPU clock speeds. The 18.432 MHz clock improves system performance and allows baud rates up to 11,500 bps. PK2200s with the 9.216 MHz option are limited to 57,600 bps. The system clock speed is a 16-bit value stored at location 0x108 in the EEPROM. The clock speed is expressed in multiples of 1200 Hz. The value read for 9.216 MHz clocks is 7,680 and for 18.432 MHz clocks the value read is 15,360.

**Microprocessor Supervisor/Watchdog Timer**

The microprocessor supervisor/watchdog timer provides the following functions for the PK2200.

- Power monitoring for the processor. Protects the system during brownouts and fluctuating power conditions. The supervisor provides a power-fail output that can be monitored by the processor, allowing the processor to save important information before a complete power-fail and then halt operation until power is fully restored.
- Battery backup for the static RAM. Allows data to remain intact even when power is removed from the PK2200.
- Watchdog timer function. Resets the system in the event of a software or hardware error that causes the processor to enter an infinite loop.

**Static RAM**

Static RAM is normally used to store program data. RAM can also be used to store program code. This is especially useful during software development because it allows quick program changes without having to change EPROMs.

**EPROM/Flash EPROM**

EPROMs offer a low-cost, permanent medium for storing program code and constant data. Once the application program is fully functioning and debugged, an EPROM can be programmed and installed. EPROMs can be quickly and easily duplicated, and are easy to install.
Even though slightly more expensive than standard EPROM, flash EPROM offers the following benefits.

- In-system programmability.
- Remote downloading of program code and data.
- Easier to reprogram.
- Erases quicker without a special eraser.

**EEPROM**

EEPROM offers a separate area for storing permanent or semi-permanent information such as clock speed, network address, calibration coefficients, and installation data. The EEPROM can be write-protected using a jumper, which prevents data from being accidentally overwritten.

**Real Time Clock (RTC)**

The RTC provides the application program with the current date and time of day. The PK2200’s battery keeps the RTC running even when the power is off. The RTC is accurate to about one second a day and compensates for leap years and variances in the number of days in each month.

**Digital Inputs**

The PK2200’s 16 digital inputs (PN01 through PN16) are flexible and robust. Configurable pull-up or pull-down resistors and high voltage protection circuits allow the inputs to detect switch contacts, relay contacts, outputs from open-collector transistor devices, logic level outputs, and high voltage outputs. In addition, two inputs may be used for generating interrupts and another two may be used for high-speed counting. The protected digital inputs have the following features:

- Nominal input voltage range of $-20 \text{ V}$ to $+24 \text{ V}$.
- Protection against overloads over the range of $-48 \text{ V}$ to $+48 \text{ V}$.
- Logic level detection.
- Configurable pull-ups and pull-downs. Jumper the digital inputs in groups of eight to pull up to $+5 \text{ V}$ or down to GND through $4.7 \text{ k}\Omega$ resistors.

The nominal voltage range for the protected digital inputs is $-20 \text{ V}$ to $+24 \text{ V}$. The inputs are protected against overvoltages in the range $-48 \text{ V}$ to $48 \text{ V}$; however, inputs should not be regularly subjected to voltages outside the nominal voltage range.

Logic-level signals can also be detected using the digital inputs. The logic threshold is nominally $2.5 \text{ V}$. The maximum guaranteed low voltage is $1.25 \text{ V}$. The minimum guaranteed high voltage is $3.75 \text{ V}$.
The digital inputs can be pulled up to +5 V or down to GND by installing jumpers on JP2. When jumpered, the digital input line impedance is 4.7 kΩ in the range 0–5 V for inputs 1–10 and 15–16. The impedance on inputs 11–14 is approximately 1.5 kΩ. Outside this range, the input impedance is greater than 3.9 kΩ for inputs 1–10 and 15–16. Jumper JP2 connects the inputs to pull-up or pull-down resistors. Table 3-1 lists the JP2 jumper settings and Figure 3-2 illustrates JP2 jumper settings.

### Table 3-1. JP2 Digital Input Jumper Settings

<table>
<thead>
<tr>
<th>Pins Jumpered</th>
<th>Inputs</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>7–9</td>
<td>1–4 and 9–12</td>
<td>Pulled up</td>
</tr>
<tr>
<td>8–10</td>
<td>5–8 and 13–16</td>
<td>Pulled up</td>
</tr>
<tr>
<td>9–11</td>
<td>1–4 and 9–12</td>
<td>Pulled down</td>
</tr>
<tr>
<td>10–12</td>
<td>5–8 and 13–16</td>
<td>Pulled down</td>
</tr>
</tbody>
</table>

The Figure 3-3 illustrates a typical digital input line.
Inputs 11–14, in addition to the protected digital input function, have the capabilities listed in Table 3-2.

**Table 3-2. Digital Input 11-14 Alternate Functions**

<table>
<thead>
<tr>
<th>Input</th>
<th>Z180 Signal</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>/INT0</td>
<td>Interrupt for user programs</td>
</tr>
<tr>
<td>12</td>
<td>/INT2</td>
<td>Interrupt for user programs</td>
</tr>
<tr>
<td>13</td>
<td>CKA0/DREQ0</td>
<td>DMA channel 0, used for counting</td>
</tr>
<tr>
<td>14</td>
<td>/DREQ1</td>
<td>DMA channel 1, used for counting</td>
</tr>
</tbody>
</table>

Inputs 11 and 12 can be used to generate hardware interrupts on the PK2200 CPU. Input 11 is connected to /INT0 and input 12 is connected to /INT2. With Dynamic C, you can easily implement service routines for these interrupts. Table 3-2 lists the alternate functions for digital inputs 11 through 14.

Refer to the *Dynamic C Technical Reference User’s Manual* for more information on writing interrupt service routines.

Inputs 13 and 14 are connected to the CPU’s DMA channels. These inputs may be used for counting high-speed digital signals. For high-speed counting (above 5 kHz), remove capacitor network CN2.

![Figure 3-4. CN2 Capacitor Networks](image)

Removing CN2 from the PK2240 disables the filtering on channels IN5, IN6, IN7, IN8, IN13, IN14, IN15, and IN16.

The high-speed counters and can perform a variety of functions including time stamping, pulse width measurement and duty cycle measurement.
**Digital Outputs**

The PK2200’s 14 digital outputs (HV01 through HV14) provide high-voltage, high-current digital outputs for your application. Sinking and optional sourcing drivers will drive a variety of loads including inductive loads such as relays, small solenoids, or stepping motors.

Note the following points regarding the digital outputs:

- Each output is individually addressable.
- Each output includes a protective diode that returns inductive spikes to the power supply.
- Sinking drivers are standard. Sourcing drivers are optional. Both drivers must be of the same type, either sinking or sourcing.

The total number of outputs that can be on simultaneously is subject to chip power limits and ambient temperature. There are power limitations on each channel as well as the entire driver IC. Eight channels, HV1–HV8, are driven by one driver IC. The other six, HV9–HV14, are driven by the other driver IC. Since fewer outputs are being driven by the HV9–HV14 driver IC, the current limit on these channels is higher than on the HV1–HV8 channels.

Figure 3-5 illustrates the configuration for the ULN2803 sinking driver.

![Figure 3-5. Sinking Driver Configuration](image)

Note the following points regarding the ULN2803 sinking driver chip.

- Outputs pull low (sink current) when turned on.
- The chip’s rating is 48 V and 500 mA maximum per channel, subject to the chip’s thermal limits and ambient temperature.
- With all channels on, each channel can sink up to 170 mA continuously (100% duty cycle) as long as the chip temperature is less or equal to 50°C. At 70°C the current must be reduced to 140 mA or less.
Figure 3-6 illustrates the connection for the UDN2985A sourcing driver. (Note the connections on header JP1.)

![Figure 3-6. Sourcing Driver Configuration](image)

Note the following points regarding the UDN2985A sourcing driver.

- Outputs pull high (source current) when turned on.
- The chip’s rating is 30 V and 250 mA maximum per channel, subject to the chip’s thermal limits and ambient temperature.
- With all channels on, each channel can source up to 170 mA continuously (100% duty cycle) as long as the chip temperature is less or equal to 50°C. At 70°C the current must be reduced to 140 mA or less.

Header JP1 configures the outputs for either sourcing or sinking drivers. Table 3-3 lists the JP1 jumper configurations shown in Figure 3-7.

**Table 3-3. JP1 High-Current Output Jumper Settings**

<table>
<thead>
<tr>
<th>JP1 Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1–3, 2–4</td>
<td>Sinking Outputs</td>
</tr>
<tr>
<td>1–2, 3–4</td>
<td>Sourcing Outputs</td>
</tr>
</tbody>
</table>

**Tip**

If incandescent lights are driven, use a series resistor to limit the incoming current.

See Appendix B: “Specifications” for more detailed information on the sinking and sourcing drivers.
Serial Communication

Two serial ports support asynchronous communication at baud rates from 300 bps to 57,600 bps on 9.216 MHz versions up to 115,200 bps with the 18.432 MHz versions. The serial ports can be configured as follows:

- Two 3-wire RS-232 ports.
- One 5-wire RS-232 port (with RTS and CTS) and one half-duplex RS-485 port.

The RJ-12 phone jack connector J2 supports full-duplex RS-232 communication with handshake lines. The RS-485 lines (J1 terminals 18 and 19) provide half-duplex asynchronous communication over twisted pair wires, up to 3 kilometers. The RS-232 ports on the PK2200 support a subset of the RS-232 standard that is in common use.

Serial Channel Configuration

Figure 3-8 illustrates the configuration of two 3-wire RS-232 channels.

![Figure 3-8. Two RS-232 Channels](image)

Figure 3-9 illustrates the configuration of one 5-wire RS-232 channel and one half-duplex RS-485 channel.

![Figure 3-9. RS-232 and RS-485](image)
Table 3-4 lists JP3 jumper settings and Figure 3-10 illustrates jumper setting configurations for the two serial channels. If only one RS-232 channel is desired, use one of the first two configurations. With these configurations, the RS-485 port is also active on the second Z180 serial channel (Z1). Unless the application software explicitly enables Z1, the RS-485 channel has no effect on the Z180. The RS-485 is connected to Z1 in the first two configurations in order to keep the Z180 CMOS input (RXA1) from floating.

### Table 3-4. JP3 Serial Communication Jumper Settings

<table>
<thead>
<tr>
<th>JP3 Jumpered Pins</th>
<th>Serial Communication Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>5–6, 7–8, 9–11</td>
<td>One 5-wire RS-232 channel (Z180 Port 0) with RTS/CTS</td>
</tr>
<tr>
<td></td>
<td>One RS-485 channel (port 1)</td>
</tr>
<tr>
<td>7–8</td>
<td>One 3-wire RS-232</td>
</tr>
<tr>
<td></td>
<td>One RS-485</td>
</tr>
<tr>
<td>5–7, 11–13</td>
<td>Two 3-wire RS-232</td>
</tr>
</tbody>
</table>

**Figure 3-10. JP3 Jumper Settings**
**Keypad and Display**

The PK2200 Series supports operator I/O through both keypad and LCD. The following two standard operator I/O configurations are available on PK2200 controller models with enclosures:

- 2-row by 20-column character LCD module plus a 2-row by 6-column keypad.
- 128-column by 64-row backlit graphic LCD module plus a 4-row by 3-column keypad.

The character LCD module is also available with an LED backlighting option and the graphic LCD has a software controllable electroluminescent backlighting installed as a standard feature. Table 3-5 lists and describes header connections and functions.

### Table 3-5. Header Connections and Function

<table>
<thead>
<tr>
<th>Header</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>H1</td>
<td>The LCD connector. Connect a 14-wire ribbon cable from the LCD to this header. Not used on the PK2240.</td>
</tr>
<tr>
<td>H2</td>
<td>The PLCBus expansion connector. This connector supports the “LCD bus” as well. Use a 26-pin ribbon cable to attach PLCBus devices to the PK2200.</td>
</tr>
<tr>
<td>H3</td>
<td>The keypad connector. Connect a 10-wire flat flexible cable from the keypad to this header. Not used on the PK2240.</td>
</tr>
</tbody>
</table>

The PK2200 series also interfaces easily with the Z-World line of operator interface products. Operator interfaces are available with a variety of keypad sizes and LCD configurations.

For more information on Z-World operator interfaces, contact your Z-World Sales Representative at (530) 757-3737.
Blank
Chapter 4 describes system development using the PK2200 interfaces and presents some sample programs to illustrate their use.
Changing Modes

The operating mode of the PK2200 is determined during power-up initialization. If a valid program is found in EPROM, then it is executed. Otherwise, the operating mode is determined by the jumper settings on JP4 or by keypress combinations. Following are the possible modes of operation:

- Run a program stored in RAM or flash EPROM.
- Prepare for Dynamic C programming using the RS-232 port.

The mode can be changed by either of the following two methods:

1. Set jumper JP4 to the desired position. Remove power from the PK2200. Apply power to the PK2200.
2. With power off, hold down the appropriate keys on the keypad and apply power. Refer to Figure 4-1 for the appropriate keypress combinations.

You will hear a series of beeps indicating that the mode has been set.

The PK2240 has a sample program loaded at the factory that will run automatically when the PK2240 is powered. You can set the PK2240 to program mode using the procedures above. All other models are preconfigured for program mode.

Setting the Mode

Figures 4-1, 4-2, and 4-3 illustrate keypad and jumper settings for run and program modes for different PK2200 configurations. The keypress combination for the 3x4 keypad will work only with the PK2240 model. The keypress combination for the 2x6 keypad will work with any model with a 2x6 keypad. If programming at normal 19,200 bps, then press the “menu setup” and “up pgm” keys. If programming at 28,800 bps, then press the “menu setup” and the “down pgm” keys. You may instead set the programming baud rate with jumpers on JP4.

At startup, a PK2200 can also be put into run mode by placing a jumper across pin 6 and pin 7 of JP4.

![Figure 4-1. 2x6 Keypad Mode Settings](image)
Development Options

Memory Options

Programs for the PK2200 are written and compiled on a PC and then downloaded to the PK2200 memory and executed. There are three memory options for program storage on the PK2200:

1. Battery-backed RAM,
2. EPROM,
3. Flash EPROM.

Battery-Backed RAM

Battery-backed RAM is a standard feature on every PK2200. RAM is available in 32K, 128K, and 512K. During development you can use RAM to download and execute programs. This speeds development because you don’t need to program and erase EPROMs. Once a program is fully debugged and running, you can create a binary file and use an EPROM burner to store the program in EPROM. Since the RAM is used to store both data and program, Z-World recommends using a larger RAM during development. If the PK2200 has flash EPROM installed, the program will be compiled to flash EPROM instead of RAM.
**EPROM**

EPROMs offer a permanent storage option for programs and data. The PK2200 BIOS is factory installed in the EPROM. After an application is fully debugged and running, it can be compiled and stored in EPROM with an EPROM burner. Each time the PK2200 powers up, it will run the stored application.

**Flash EPROM**

Flash EPROM offers the benefits of both battery-backed RAM and standard EPROM. You can quickly change and download a program as if you were using RAM. Using flash EPROM frees up RAM for data storage rather than program storage. Flash EPROM does not depend upon the onboard battery to retain data, so a program is safe in the event that the battery is drained.

For more information on memory options or to place an order, contact your Z-World Sales Representative at (530) 757-3737.

**Digital Inputs**

The digital inputs can be used for a variety of applications such as detecting high-voltage and logic level digital signals, providing interrupts for time critical events, and high-speed counting.

**Using the Digital Inputs**

The digital inputs are supported in software by Dynamic C functions and virtual driver variables. There are several methods for reading the digital inputs. Some of the digital inputs have additional features listed below.

**Interrupt Inputs**

Inputs 11 and 12 can be used to generate level sensitive hardware interrupts on the PK2200 CPU. Interrupts can be used to signal events that need to be serviced in real-time.

**High Speed DMA Counter**

Two counters connected to digital inputs 13 and 14 are actually the CPU’s DMA channel counters.

- The maximum counting speed is ≈1.5 MHz for 9.216 MHz PK2200 series controllers.
- The maximum counting speed is 3.0 MHz for 18.432 MHz PK2200 series controllers.
The following points summarize the counter’s capabilities:

- The counter can measure the time at which a negative edge occurs with a precision of a few microseconds. A minimum time must occur between successive events to allow for interrupt processing.
- The counter can measure the width of a pulse by counting (up to 65,536) at a rate that varies from 300 Hz to 600 kHz, providing 16-bit accuracy.
- Count negative-going edges for up to two channels. The maximum count for high-speed counting (5 kHz and up) is 65,536. For low speeds, the maximum count is unlimited.

Function calls load the count-down value for the DMA channel and enable the DMA interrupt. Once a counter reaches zero, flags for the DMA channel are set to 1. DMA flags can be monitored by an application program.

**Digital Outputs**

**Using the Digital Outputs**

The digital outputs are supported in software by Dynamic C functions and virtual driver variables. There are several methods for writing to the digital outputs.

The digital outputs can be used for a wide variety of applications including the following:

- Driving solenoids, relays, motors and other inductive loads directly.
- Driving incandescent lamps, LEDs and resistive loads directly.
- Driving FETs, transistors, thyristors or solid state relays to increase the current or voltage output capability as well as providing a.c. drive capability.
Serial Communication

Dynamic C has serial communication support libraries. For the Z180 port z0 and Z180 port z1, use AASC.LIB, Z0232.LIB, and Z1232.LIB. For RS-232 expansion cards that interfaced through the PLCBus on the PK2200, use EZIOPLC.LIB.

Functional support for serial communication includes the following:

- Initialization of the serial ports
- Monitoring, and reading, a circular receive buffer
- Monitoring, and writing to, a circular transmit buffer
- An echo option
- CTS (clear to send) and RTS (request to send) control for RS-232.
- XMODEM protocol for downloading and uploading data
- A modem option

The PK2200 can be configured for either two RS-232 channels or one RS-232 and one RS-485. Z0 is RS-232 only and Z1 may be configured for RS-232 or RS-485. See Chapter 3 for information on configuring the serial communication channels.

Z180 Port Z0 is configured at the factory for RS-232 and Port Z1 is configured for RS-485.

Receive and Transmit Buffers

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

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

If the echo option is turned on during initialization of the serial port (with `Dinit_z0`, `Dinit_z1`, or `Dinit_uart`) any character received is automatically echoed back (transmitted out). This feature is ideal for use with a dumb terminal and also for checking the characters received.

**CTS/RTS Control**

Z180 port 0 is constrained by hardware to have the CTS (clear to send) pulled low by the RS-232 device with which it is communicating. An RS-232 expansion card, however, can enable or disable the effect of the CTS line. Z180 port 1 does not support the CTS / RTS lines.

If you choose the CTS/RTS option, the support software pulls the RTS (request to send) line high when the receive buffer has reached 80 percent of capacity. Thus, the transmitting device (if its CTS is enabled) stops transmitting. The RTS line is pulled low again when the received buffer has gone below 20 percent of capacity.

If the device with which the PK2200 is communicating does not support CTS and RTS, the CTS and RTS lines on the PK2200’s side can be tied together to make communication possible.

**XMODEM File Transfer**

The PK2200 supports the XMODEM protocol for downloading and uploading data. Currently, the library supports downloading an array of data whose size is a multiple of 128 bytes.

Uploaded data is written to a specified area in RAM. The targeted area for writing should not conflict with the current resident program or data. During XMODEM transfers, character echo is automatically suspended.

**Modem Communication**

Modems and telephone lines allow serial communication across a great distance. If you choose the modem option, character streams that are read from the receive buffer are automatically scanned for modem commands. When a modem command is found, the software takes appropriate action. Normally, the communication package functions in COMMAND mode while waiting for valid modem commands or messages. Once a link is established, communication functions in DATA mode. However, the software continues to monitor the modem for a `NO_CARRIER` message.

The software assumes that modem commands are terminated with `CR`, which is carriage return (0x0D). The modem option is easiest to use when the user protocol also has `CR` as the terminating character. Otherwise, the software has to check for two different terminating characters. The user’s terminating character cannot be any of the ASCII characters used in modem commands, nor can it be a line-feed character.
Library functions for the RS-232 port support communication with a Hayes Smart Modem or compatible. Note the following points:

- The CTS, RTS, and DTR lines of the modem are not used.
- If the modem used is not truly Hayes Smart Modem compatible, the user has to tie the CTS, RTS, and DTR lines on the modem side together. The CTS and RTS lines on the PK2200 side also have to be tied together.
- A NULL connection is required for the TX and RX lines.
- A commercial NULL modem will have its CTS and RTS lines tied together on both sides.

Figure 4-4 shows the correct modem to PK2200 wiring.

Following are descriptions for Z180 port 0 functions. Similar functions are available for the RS-232 (UART) expansion card. Please note the following substitutions:

For the RS-232 expansion card, substitute `uart` for `z0` in the function name.

For Z180 port 1, substitute `z1` for `z0` in the function name.

For example, the initialization routine for the Z180 port 0 is called `Dinit_z0()`. The equivalent function for the RS-232 expansion card is `Dinit_uart()` The equivalent function for Z180 port 1 is `Dinit_z1()`.

Refer to Appendix F, “PLCBus,” for details on software support for the RS-232 expansion card.
**Interrupt Handling for Z180 Port 0**

Normally, a serial interrupt service routine would be declared with the compiler directive:

```
#define INT_VEC SER0_VEC routine
```

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

The following function loads the address of the service function into the specified location in the interrupt vector table.

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

The `INT_VEC` directive should not be used with this function. Once the service routine has taken over, you can’t debug your program in Dynamic C.

If you communicate with a serial device other than the PC’s Dynamic C programming port, your program has to make sure that the hardware is properly configured before sending any messages. For example, when using Z180 port 0 for serial communication with a modem, use the PK2200’s keypad to trigger serial port initialization. Without this trigger, the modem may not properly communicate with the support software because the initialization routine also sends initialization commands to the modem.

When executable programs are generated either for EPROM or for downloading to RAM, there is no need for communication with Dynamic C. The compile-time directive `INT_VEC` can then be used freely.

**Remote Downloading**

The PK2200 has the capability of remote downloading program code. This allows units to be reprogrammed in the field, eliminating the need to recall units for reprogramming or sending field service personnel to install new software. In order to use the remote download feature, the PK2200 must have a serial link to the remote PC, either a direct RS-232 link or a modem. The RS-232 connection is limited to several hundred feet. Modems allow communication over virtually unlimited distances.

If you plan to use the remote download feature, make sure that the PK2200 has enough memory to store future program revisions and data. Refer to Dynamic C Technical Reference Manual for a detailed description of the remote downloading procedure.
**Developing an RS-485 Network**

The two-wire RS-485 serial-communication port and Dynamic-C network software allow network development. Screw terminal strip J1 provides a half-duplex RS-485 interface. The RS-485 signals are on screw terminals 18 and 19.

The PK2200 and/or other controllers can be linked together over several kilometers. When configuring a multi-drop network, use single twisted pair wires on all controllers to connect RS-485+ to RS-485+ and RS-485- to RS-485. A diagram of a two-wire RS-485 network is shown in Figure 4-5.

Any Z-World controller can be a master or a slave. A network can have up to 255 slave controllers, but only one controller can be the master.

In a multidrop network, termination and bias resistors are required to minimize reflections (echoing) and to keep the network line active during an idle state. Only the first and last board on a multidrop RS-485 cable should have termination resistors. Therefore, when networking multiple boards via RS-485, remove termination resistors from all boards in the network, except for the first and last board of the network.

Only a single, solid conductor should be placed in a screw clamp terminal. Bare copper, particularly if exposed to the air for a long period before installation, can become oxidized. The oxide can cause a high resistance (~20 ohm) connection, especially if the clamping pressure is not sufficient. To avoid oxidation, use tinned wires or clean, shiny copper wire. If you are using multiple conductors or stranded wire, consider soldering the wire bundle or using a crimp connector to avoid a later loss of contact pressure to a spontaneous rearrangement of the wire bundle. Soldering may make the wire subject to fatigue failure at the junction with the solder if there is flexing or vibration.
Keypad and LCD

The PK2200 Series supports operator I/O with a keypad and LCD. Two standard operator I/O configurations are available on PK2200 series controllers with enclosures:

- 2-row by 20-column character LCD module with a 2-row by 6-column keypad.
- 128-column by 64-row backlit graphic LCD module with a 4-row by 3-column keypad.

The character LCD module is also available with an LED backlighting option. The graphic LCD has electroluminescent backlighting installed as a standard feature.

Using the Keypad and Display

The PK2200 keypad and display are supported by a large number of software drivers. The keypad and display can be used for a variety of user interface applications including the following:

- User code or password entry
- System status display
- Multiple language/character-set displays
- Parameter monitoring and adjustment

PK2200 Keypads

Table 4-1 shows standard keypad configurations.

<table>
<thead>
<tr>
<th>Model</th>
<th>Keypad</th>
</tr>
</thead>
<tbody>
<tr>
<td>PK2200</td>
<td>2 x 6</td>
</tr>
<tr>
<td>PK2210</td>
<td>2 x 6</td>
</tr>
<tr>
<td>PK2220</td>
<td>None</td>
</tr>
<tr>
<td>PK2230</td>
<td>None</td>
</tr>
<tr>
<td>PK2240</td>
<td>4 x 3</td>
</tr>
</tbody>
</table>
**Keypad Insert Templates**

The keypads are designed to accept paper inserts. Inserts can be produced on regular paper using a laser printer, thus allowing quick and easy customization of keypad legends.

You can use the templates below for creating inserts. All dimensions are in inches. Inserts can be secured by taping the portion of the insert that extends beyond the keypad to the supporting bracket.

*Figure 4-6. 2x6 Keypad Insert Template*

*Figure 4-7. 3x4 Keypad Insert Template*
Keypad Codes
The PK2200 keypads are supported by Dynamic C functions that return codes corresponding to the key pressed. The figures below show the codes for the 2x6 and 3x4 keypads used on the PK2200 Series controllers.

Figure 4-8. 2x6 Keypad Codes

Figure 4-9. 3x4 Keypad Codes
**PK2200 LCDs**

The PK2200 Series LCDs are easy to use with Dynamic C software libraries. Shown below are the layouts for both the 2x20 character display and the 64x128 graphic display.

![2x20 Character LCD](image)

*Figure 4-10. 2X20 Character LCD*

![64x128 Graphic LCD](image)

*Figure 4-11. 64X128 Graphic LCD*

**Graphic LCD Status**

Several Dynamic C library functions return the operating status of the LCD. The LCD status bits are shown in the following bitmap.

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUSY</td>
<td>0</td>
<td>ON/OFF</td>
<td>RESET</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Most significant bit  Least significant bit

**BUSY** - Reading a “1” indicates LCD is performing an operation. Reading a “0” indicates the LCD is ready to accept more data.
**ON/OFF** - When the ON/OFF bit is set (1) the display is on, any image on the screen will be visible. When the bit is reset (0) any images on the display will not be visible. The image is still in the display memory.

**RESET** - Resets the LCD module when low (0).

**Bitmapped Graphics**

Many of the Dynamic C functions that operate on the graphic LCD use bitmaps. These bitmaps represent the images on a section of the display. An individual dot, or pixel, is represented by one bit in the bitmap. If the pixel is on, the corresponding bit is set. If the pixel is off, the corresponding pixel is reset.

The image on the display is two-dimensional (width and height). The bitmap used to store that display information is a one-dimensional array. Two-dimensional images are stored in column major, byte aligned bitmap format.

Column major means that bits are stored in the bitmap column by column. The first pixel of the first column (row 0, column 0) of the image is stored in the first bit position in the bitmap. The second pixel of the first column is stored in the second bit position in the bitmap and so on. When the entire first column is stored in the bitmap, the process begins again with the second column and repeats until all columns of the image are stored.

Byte aligned means that a column data will end on a byte boundary. If a column has a number of bits that is not evenly divisible by eight, then the remaining bits of the last byte representing a column will be left unused. Image data from the next column will be stored starting in the next byte.
Chapter 5 covers the software drivers used with the PK2200 series controllers.
Software Drivers

Drivers are functions that simplify accessing PK2200 hardware and I/O. For the following reasons, Z-World drivers make writing software easier:

- Provide commonly needed functionality.
- Eliminate the need to know all of the details of operation.
- Previously tested.
- Simplify source code by replacing multiple lines of code with one function call.


Real Time Clock (RTC)

The RTC keeps the date and the time of day with a resolution of one second. The worst case error is 50ppm (4.3 seconds) per day. Leap years and variances in the number of days in a month are automatically tracked.

The following structure holds the time and date:

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

The following routines read and write to the real-time clock:

- `int tm_wr( struct tm *x )`
  
  Sets the system time to the values in the structure pointed to by *x.

  PARAMETER1: Pointer to the structure holding the system time information to be written.

  RETURN VALUE: 0 if successful; -1 if clock failing or not installed.

- `int tm_rd( struct tm *t )`

  Reads the current system time into the structure t.

  PARAMETER1: Pointer to the structure used to store the system time.

  RETURN VALUE: 0 if successful; -1 if clock failing or not installed.
**EEPROM**

The following functions provide access to the EEPROM. The EEPROM is generally used for storing system information, calibration information, or any data that does not need to change often.

- **int ee_rd (int address)**
  
  Reads value from EEPROM at specified address.
  
  PARAMETER1: The address to read from.
  
  RETURN VALUE: EEPROM data (0-255) if successful; negative value if unable to read EEPROM.

- **int ee_wr (int address, char data)**
  
  Writes value to EEPROM at specified address.
  
  PARAMETER1: The address to write to.
  
  RETURN VALUE: Returns 0 if write is successful, negative value if unsuccessful.
Digital Inputs and Outputs

- **DIGIN1, DIGIN2, ..., DIGING16**
  The virtual driver variables DIGIN1, DIGIN2, ..., DIGING16 represent the state of the digital inputs. These variables take the value 1 if the input is high and 0 if the input is low. The value is changed only if the new value remains the same for 2 ticks (25 to 50 milliseconds) of the virtual driver.

- **void VIOInit();**
  VIOInit is a dummy function used as a host for the `GLOBAL_INIT` of the virtual I/O variables. Virtual inputs are read and virtual outputs are written out whenever the function `VIODrvr()` is called. Inputs are DIGIN1 to DIGIN16. Outputs are OUT1 to OUT16. The DIGIN values have to be the same for two successive reads to be valid.
  RETURN VALUE: None.

- **void VIODrvr();**
  Updates the virtual inputs DIGIN1 to DIGIN16. The virtual outputs OUT1 to OUT14 are send out to corresponding output ports.
  RETURN VALUE: None.

**Digital Input Drivers**

There are several methods for reading digital inputs and setting digital outputs on the PK2200. Below is a listing of drivers for the digital inputs and outputs, including the high-speed DMA counters.

- **int up_digin( int channel )**
  Read the value at the specified digital input channel (1–16).
  RETURN VALUE: The function returns 1 when the channel is high and 0 when the channel is low.

- **unsigned inport( unsigned port )**
  Reads value from I/O port.
  PARAMETER: port is the I/O port to be read.
  RETURN VALUE: Value from I/O port.
  You can read multiple PK2200 digital inputs simultaneously using the `inport()` function.

DIGBANK1 is the address (0x180) of the first eight digital inputs DIN1 through DIN8. Bit zero represents the state of DIN1, bit one is DIN2, etc. DIGBANK2 is the address (0x181) of the second eight digital inputs DIN9 through DIN16. Bit zero represents the state of DIN9, bit one is DIN10, etc.
The lower eight bits returned by the inport() function represent the status of the digital inputs. Bits which are set (1) represent inputs which are high. Bits which are reset (0) correspond to inputs which are low.

Example:

```c
lowDIBank = inport( DIGBANK1 );
highDIBank = inport( DIGBANK2 );
```

- **void DMA0Count (unsigned int count)**
  Loads the DMA channel 0 with the `count` value and enables the DMA channel 0 interrupt.
  The function sets the flag `_DMAFLAG0` to zero. When `count` negative edges have been detected, the channel causes an interrupt, and the interrupt service routine sets the flag `_DMAFLAG0` to 1. A program can monitor `_DMAFLAG0` to determine if the number of counts has occurred.
  PARAMETER: `count` is the number of pulses to count.
  RETURN VALUE: None

- **void DMA1Count (unsigned int count)**
  Loads DMA channel 1 with the `count` value and enables the DMA channel 1 interrupt.
  The function sets the flag `_DMAFLAG1` to zero. When `count` negative edges have been detected, the channel causes an interrupt, and the interrupt service routine sets the flag `_DMAFLAG1` to 1. A program can monitor `_DMAFLAG1` to determine if the number of counts has occurred.
  PARAMETER: `count` is the number of pulses to count
  RETURN VALUE: None

- **unsigned int DMASnapShot (byte channel, unsigned int *count)**
  Reads the number of pulses that a DMA channel has counted. A DMA counter is initialized with one of the two preceding functions.
  PARAMETERS: `channel` is the DMA channel (0 or 1)
  `*count` is a pointer to variable holding the pulse count.
  RETURN VALUE: 0, if pulse train is too fast to have a snapshot taken; 1, if a snapshot is obtained and valid data is in `*count`.

Even if a program is unable to read the counts, DMA interrupts still occur when the DMA channel counts down from its loaded value.
**Digital Output Drivers**

Following are the digital output drivers for the PK2200:

- **int up_setout (int channel, int value )**  
  Sets the state of digital output.  
  PARAMETER1: The digital output channel to set.  
  PARAMETER2: The state to set. 1 (active) or 0 (inactive).  
  RETURN VALUE: None.  
  Pass `channel` (1–14) and `value`: 0 for OFF, 1 for ON.

- **OUT1, OUT2, ..., OUT14**  
  Set the virtual driver variables OUT1, OUT2, OUT3, ... OUT13,  
  OUT14 to a value of 0 to turn off the output, or 1 to turn on the output.

- **void outport(unsigned port, unsigned value);**  
  Writes value to I/O port.  
  PARAMETER1: The output port.  
  PARAMETER2: The value to be written.  
  RETURN VALUE: None  
  The addresses DRV1 - DRV14 are the port addresses for the digital outputs. Writing 0 to any of these ports will turn the output OFF. To turn ON digital outputs 1 through 8 write 0x20 to the corresponding port. For digital outputs 9 through 14 write 0x40 to turn the output ON.

- The digital outputs are individually addressed and must be set one at a time.
**LCD and Keypad**

The following functions provide routines for writing to the LCD and reading the keypad.

Include the following directives in your program if using the PK2240.

```c
#include <wintek.lib>
#include <kp.lib>
```

The following directives provide information to the compiler about the graphic LCD and keypad on the PK2240.

- **void lc_init_keypad()**
  
  Initializes timer1, keypad driver, and variables, and, if used, the real-time kernel.

  RETURN VALUE: None.

- **int lc_kxget (byte mode)**
  
  Fetches the key value from the FIFO keypad buffers. If mode = 0, value is removed from the buffer; otherwise, value remains in the buffer.

  RETURN VALUE: Key value or -1, if no key is available.

The “Keypad and LCD” section in Chapter 4 describes key values.

- **void lc_kxinit()**
  
  Initialize the keypad driver and accessory variables. If virtual watchdogs are defined the virtual watchdogs are initialized.

  RETURN VALUE: None.

- **void lc_setbeep (int count)**
  
  Sounds the beeper for the number of 1280 Hz cycles specified by count.

  RETURN VALUE: None.

- **up_beep (int milliseconds)**
  
  Sets beeper on for specified number of milliseconds. Scaling of the count passed is dependent on the periodic routines that calls lc_beepscan. If BeepScale is undefined, it is defaulted to 0.04.

  RETURN VALUE: None.

- **void lc_char (char ch)**
  
  Sends a character to the LCD. The function waits for the LCD to become free before sending the character.

  RETURN VALUE: None.
• **int lc_cmd (int cmd)**  
Waits for LCD busy flag to clear, then sends cmd to LCD command register.
RETURN VALUE: 0, if successful in writing to the LCD; else -1, if timeout

• **void lc_ctrl (byte cmd)**  
Write a control cmd to the LCD.
RETURN VALUE: None.

• **void lc_init()**  
Initializes the LCD. The display is turned on, cleared, and the cursor (now in the top left character position) blinks.
RETURN VALUE: None

• **void lc_nl()**  
Performs a new line function on the LCD.
RETURN VALUE: None

• **void lc_pos (int line, int column)**  
Positions the cursor at the line designated by line and column designated by column on the LCD.
RETURN VALUE: None

• **void lc_printf (char* fmt, ...)**  
Performs a printf to the LCD. The function arguments are specified as they are for the standard printf.
RETURN VALUE: None

• **int lc_wait()**  
Waits for LCD busy flag to clear. Caution, doesn’t time out.
RETURN VALUE: 0, when LCD busy flag has cleared; else -1, if timeout after ten tries.

• **void glSetBrushType(int type)**  
Sets the brush type for all following graphics operations in this library. It controls how pixels are drawn on the screen with respect to existing pixels.
PARAMETER1: This is the type of the brush. Possible values are GL_Set for forcing pixels on, GL_Clear for forcing pixels off, GL_Xor for toggling the existing pixels and GL_Block to overwrite the entire memory location corresponding to the pixel.
RETURN VALUE: None.
• **int glInit()**

Initializes the LCD module (software and hardware).

**RETURN VALUE:** returns the status of the LCD. If the initialization was successful, this function returns 0. Otherwise, the returned value indicates the LCD status.

• **int glBlankScreen()**

Blanks the screen of the LCD.

**RETURN VALUE:** The returned value indicates the status of the LCD after the operation.

• **int glPlotDot(int x, int y)**

Plots one pixel on the screen at coordinate (x,y).

**PARAMETER1:** the x coordinate of the pixel to be drawn.

**PARAMETER2:** the y coordinate of the pixel to be drawn.

**RETURN VALUE:** Status of the LCD after the operation.

• **void glPlotLine(int x1, int y1, int x2, int y2)**

Plots a line on the LCD.

**PARAMETER1:** x coordinate of first endpoint.

**PARAMETER2:** y coordinate of first endpoint.

**PARAMETER3:** x coordinate of second endpoint.

**PARAMETER4:** y coordinate of second endpoint.

**RETURN VALUE:** None.

• **void glPutBitmap(int x, int y, int bmWidth, int bmHeight, char *bm)**

Displays a bitmap stored in root memory on the LCD. For bitmaps defined in xmem memory, use **glXPutBitmap**.

**PARAMETER1:** x coordinate of the bitmap (left edge).

**PARAMETER2:** y coordinate of the bitmap (top edge).

**PARAMETER3:** width of the bitmap.

**PARAMETER4:** height of the bitmap.

**PARAMETER5:** pointer to the bitmap.

**RETURN VALUE:** None.
• **void glXPutBitmap(int x, int y, int bmWidth, int bmHeight, unsigned long bmPtr)**

Displays a bitmap stored in xmem on the LCD. For bitmaps stored in root memory, use `glPutBitmap`.

PARAMETER1: x coordinate of the bitmap (left edge).
PARAMETER2: y coordinate of the bitmap (top edge).
PARAMETER3: width of the bitmap.
PARAMETER4: height of the bitmap.
PARAMETER5: pointer to the bitmap.
RETURN VALUE: None.

• **void glGetBitmap(int x, int y, int bmWidth, int bmHeight, char *bm)**

Gets a bitmap from the LCD.

PARAMETER1: x coordinate of the bitmap (left edge).
PARAMETER2: y coordinate of the bitmap (top edge).
PARAMETER3: width of the bitmap.
PARAMETER4: height of the bitmap.
PARAMETER5: pointer to the bitmap.
RETURN VALUE: None.

• **void glFontInit(struct _fontInfo *pInfo, char pixWidth, char pixHeight, unsigned startChar, unsigned endChar, char bitmapBuffer)**

Initializes a font descriptor with the bitmap defined in the root memory. For fonts with bitmaps defined in xmem, use `glXFontInit`.

PARAMETER1: pointer to the font descriptor to be initialized.
PARAMETER2: width of each font item (must be uniform for all items).
PARAMETER3: height of each font item (must be uniform for all items).
PARAMETER4: offset to the first useable item (useful for fonts for ASCII or other fonts with an offset).
PARAMETER5: index of the last useable font item.
PARAMETER6: pointer to a linear array of font bitmap.
RETURN VALUE: None.
• **void glXFontInit(struct _fontInfo *pInfo, char pixWidth, char pixHeight, unsigned startChar, unsigned endChar, unsigned long xmemBuffer)**

Initializes a font descriptor that has the bitmap defined in xmem. For bitmaps defined in root memory, use glFontInit.

**PARAMETER1:** pointer to the font descriptor to be initialized.

**PARAMETER2:** width of each font item (must be uniform for all items).

**PARAMETER3:** height of each font item (must be uniform for all items).

**PARAMETER4:** offset to the first useable item (useful for fonts for ASCII or other fonts with an offset).

**PARAMETER5:** index of the last useable font item.

**PARAMETER6:** pointer to a linear array of font bitmap.

**RETURN VALUE:** None.

• **void glPutFont(int x, int y, struct fontInfo *pInfo, unsigned code)**

Puts an entry from the font table to the LCD.

**PARAMETER1:** x-coordinate of the entry (left edge).

**PARAMETER2:** y-coordinate of the entry (top edge).

**PARAMETER3:** pointer to the font descriptor that describes the font table to be indexed.

**PARAMETER4:** code (offset) in the font table that indexes the bitmap to display.

**RETURN VALUE:** None.

• **void glVPrintf(int x, int y, struct fontInfo *pInfo, char *fmt, void *firstArg)**

Prints a formatted string on the LCD screen, similar to vprintf.

**PARAMETER1:** x coordinate of the text (left edge).

**PARAMETER2:** y coordinate of the text (top edge).

**PARAMETER3:** pointer to font descriptor that describes the font used for printing the text.

**PARAMETER4:** pointer to the string that describes the format.

**PARAMETER5:** pointer to the first argument to instigate the format string.

**RETURN VALUE:** None.
• **void glPrintf(int x, int y, struct _fontInfo *pInfo, char *fmt,...)**

  Prints a formatted string (much like printf) on the LCD screen.
  
  **PARAMETER1:** x coordinate of the text (left edge).
  
  **PARAMETER2:** y coordinate of the text (top edge).
  
  **PARAMETER3:** pointer to the font descriptor used for printing on the
  LCD screen.
  
  **PARAMETER4:** pointer to the format string
  
  **RETURN VALUE:** None.

• **void glPlotCircle(int xc, int yc, int rad)**

  Draws a circle on the LCD.
  
  **PARAMETER1:** x coordinate of the center.
  
  **PARAMETER2:** y coordinate of the center.
  
  **PARAMETER3:** radius of the circle.
  
  **RETURN VALUE:** None.

• **int wtDisplaySw(int onOff)**

  Switches the display on and off.
  
  **PARAMETER1:** If this parameter is 1, the display is turned on. If this
  parameter is 0, the display is turned off.
  
  **RETURN VALUE:** Status of the LCD after the operation.

• **void kdiELSw(int value)**

  Switchs the EL backlight of the LCD.
  
  **PARAMETER1:** 1 to turn the backlight on, 0 to turn the backlight off.
  
  **RETURN VALUE:** None.

• **void kdiSetContrast(unsigned content)**

  Sets the contrast control to `content`.
  
  **PARAMETER1:** Specifies the contrast (the higher the value, the higher
  the contrast).
  
  **RETURN VALUE:** None
• **void kpInit(int (*changeFn)())**

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

PARAMETER1: This is a pointer to a function that will be called when the driver detects a change (when **kpScanState** is called). Two arguments are passed to the call-back function. The first argument is a pointer to an array that indicates the current state of the keypad. The second pointer is a pointer to an array that indicates what keypad positions are changed and detected by **kpScanState**. The byte offset in the array represents the line pulled high (row number), and the bits in a byte represents the positions (column number) read back.

RETURN VALUE: None

• **int kpScanState()**

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

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

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

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

PARAMETER1: Points to an array that reflects the current state of the keypad (bitmapped, 1 indicates key is not currently pressed).

PARAMETER2: Points to an array that reflects the CHANGE of keypad state from the previous scan. (bitmapped, 1 indicates there was a change).

RETURN VALUE: -1 if no key is pressed. Otherwise it returns the normalized key number. The normalized key number is 8*row+col+edge*256. Edge is 1 if the key is released, and 0 if the key is pressed.
• **int kpDefGetKey()**

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

   **RETURN VALUE:** -1 if no key is pressed. Otherwise it returns the normalized key number. The normalized key number is 8*row+col+edge*256. Edge is 1 if the key is released, and 0 if the key is pressed.

• **void kpDefInit()**

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

   **RETURN VALUE:** NA.

**Sample Programs**

The sample programs listed in Table 5-1 are specific to the PK2200. They can be found in the `SAMPLES\CPLC` directory.

<table>
<thead>
<tr>
<th>Program</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5KEYCODE.C</td>
<td>Code-driven sample program for the five-key system.</td>
</tr>
<tr>
<td>5KEYDEMO.C</td>
<td>Uses a code-driven five-key system and the RT-kbyte for I/O monitor and control.</td>
</tr>
<tr>
<td>5KEYLAD.C</td>
<td>Combines 5KEYCODE.C and LADDERC.C.</td>
</tr>
<tr>
<td>5KEYLINK.C</td>
<td>Linked-list sample program for the five-key system.</td>
</tr>
<tr>
<td>5KEYSCAN.C</td>
<td>Combines 5KEYCODE.C and SCANBLK.C.</td>
</tr>
<tr>
<td>CDEMO_RT.C</td>
<td>Demonstrate the use of the real-time kernel.</td>
</tr>
<tr>
<td>DIGDEMO.C</td>
<td>Use the keypad to select which digital input channel to monitor.</td>
</tr>
<tr>
<td>DIGVDVR.C</td>
<td>Similar to DIGDEMO.C but uses the virtual driver to monitor the state of the input.</td>
</tr>
<tr>
<td>DMACOUNT.C</td>
<td>Demonstrates the use of the high speed counters.</td>
</tr>
</tbody>
</table>

*continued*
### Table 5-1. PK2200 Sample Programs (concluded)

<table>
<thead>
<tr>
<th>Program</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LADDERC.C</td>
<td>Use ladder C for I/O control.</td>
</tr>
<tr>
<td>LCGRAM.C</td>
<td>Illustrates use of the LCD character generator.</td>
</tr>
<tr>
<td>OUTDEMO.C</td>
<td>Use keypad to toggle the state of the digital outputs.</td>
</tr>
<tr>
<td>OUTVDVR.C</td>
<td>Similar to OUTDEMO.C, but uses the virtual driver to change the state of the output.</td>
</tr>
<tr>
<td>PRT0DEMO.C</td>
<td>Use TIMER0 for timer interrupt.</td>
</tr>
<tr>
<td>READIO.C</td>
<td>Read and toggle the I/Os through STDIN. The I/Os are driven by function calls.</td>
</tr>
<tr>
<td>READKEY.C</td>
<td>Read the keypad and write to the LCD and to the STDIO window.</td>
</tr>
<tr>
<td>SCANBLK.C</td>
<td>Use function blocks for I/O control.</td>
</tr>
<tr>
<td>UREADIO.C</td>
<td>Read and toggle the I/Os through STDIN. The I/Os are driven by the virtual driver.</td>
</tr>
<tr>
<td>VWDog.C</td>
<td>Illustrates the use of the virtual watchdogs and of KEYREQUEST</td>
</tr>
</tbody>
</table>

### Communication Sample Programs

The sample communication programs listed in Table 5-2 are located in the SAMPLES\NETWORK directory.

### Table 5-2. Sample Communication Programs

<table>
<thead>
<tr>
<th>Program</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSREMOTE.C</td>
<td>Slave version of CZ0REM.C, that includes most capabilities of CZ0REM.C. Master-to-slave communication is via opto22 9th-bit binary protocol.</td>
</tr>
<tr>
<td>CUARTREM.C</td>
<td>Same as CZ0REM.C but uses XP8700 expansion card.</td>
</tr>
</tbody>
</table>

continued…
### Table 5-2. Sample Communication Programs (concluded)

<table>
<thead>
<tr>
<th>Program</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CZ0REM.C</td>
<td>More elaborate sample of serial communication between board and PC dumb terminal. Includes modem communication, data monitoring, time and date setup, memory read and write, data logging, XMODEM download of the data log, XMODEM upload of binary file for remote downloading. Also supports master-to-slave communication. (Slave has to be running the program <strong>CSREMOTE.C</strong>.)</td>
</tr>
<tr>
<td>RS232.C</td>
<td>RS-232 communication with a PC dumb terminal, with or without modem. Also, master-to-slave communication with another board running RS-485.C.</td>
</tr>
<tr>
<td>RS485.C</td>
<td>Slave program to communicate with the master running RS-232.C.</td>
</tr>
<tr>
<td>UART232.C</td>
<td>RS-232 communication through an RS-232 expansion card with the PK2200.</td>
</tr>
</tbody>
</table>

**PK2240 Sample Programs**

The sample programs listed in Table 5-3 are specific to the PK2240 and are located in the `SAMPLES\PK224X` directory. These programs illustrate the use of the graphic LCD and keypad.

### Table 5-3. PK2240 Sample Programs

<table>
<thead>
<tr>
<th>Program</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLPRINTF.C</td>
<td>Demonstrates the <code>glprintf</code> function and shows how to print text on the graphics display.</td>
</tr>
<tr>
<td>KPDEFLT.C</td>
<td>Demonstrates key scanning techniques using functions in the KP library.</td>
</tr>
</tbody>
</table>
APPENDIX A: TROUBLESHOOTING

Appendix A provides procedures for troubleshooting system hardware and software.
Out of the Box

Check the items mentioned in this section before starting development.

- Verify that the PK2200 runs in standalone mode before connecting any expansion boards or I/O devices.
- Verify that the entire host system has good, low-impedance, separate grounds for analog and digital signals. Often the controller is connected between the host PC and another device. Any differences in ground potential from unit to unit can cause serious problems that are hard to diagnose.
- Do not connect analog ground to digital ground anywhere.
- Double-check the connecting ribbon cables to ensure that all wires go to the correct headers.
- Verify that the host PC’s COM port works by connecting a good serial device to the COM port. Remember that COM1/COM3 and COM2/COM4 share interrupts on a PC. User shells and mouse drivers, in particular, often interfere with proper COM port operation. For example, a mouse running on COM1 can preclude running Dynamic C on COM3.
- Use the supplied Z-World power supply. If another power supply must be used, verify that it has enough capacity and filtering to support the PK2200.
- Use the supplied Z-World cables. The most common fault of user-made cables is failure to properly assert CTS at the RS-232 port of the controller. Without CTSs being asserted, the controller’s RS-232 port will not transmit. Assert CTS by either connecting the RTS signal of the PC’s COM port or looping back the PK2200’s RTS.
- Experiment with each peripheral device connected to the controller to determine how it appears to the controller when powered up, powered down, and/or when its connecting wiring is open or shorted.
- If a DB9 connector or an RJ-12 connector is wired up to a 10-pin connector, carefully check the connections. These wires do not run pin-for-pin.

Note: Telephone company wiring does not follow a standardized color code.
Dynamic C Will Not Start

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

- **Wrong Baud Rate** — In rare cases, the baud rate has to be changed when using the Serial Interface Board for development.

- **Wrong Communication Mode** — Both sides must be talking RS-232.

- **Wrong COM Port** — A PC generally has two serial ports, COM1 and COM2. Specify the one being used in the Dynamic C “Target Setup” menu. Use trial and error, if necessary.

- **Wrong Operating Mode** — Communication with Dynamic C will be lost if the controller’s jumper is set for standalone operation. Reconfigure the board for programming mode.

- **Wrong Memory Size** — Jumpered pins on JP2 specify the EPROM size.

If all else fails, connect the serial cable to the controller after power up. If the PC’s RS-232 port supplies a large current (most commonly on portable and industrial PCs), some RS-232 level converter ICs go into a nondestructive latch-up. Connect the RS-232 cable after power up to eliminate this problem.

Dynamic C Loses Serial Link

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

PK2200 Repeatedly Resets

The PK2200 resets every 1.0 seconds if the watchdog timer is not “hit.” If a program does not “hit” the watchdog timer, then the program will have trouble running in standalone mode. To “hit” the watchdog, make a call to the Dynamic C library function `hitwd`. 
Common Programming Errors

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>(-32,768 \text{ (}2^{15}\text{)} ) to (+32,767 \text{ (}2^{15} - 1\text{)})</td>
</tr>
<tr>
<td>long int</td>
<td>(-2,147,483,648 \text{ (}2^{31}\text{)} ) to (+2,147,483,647 \text{ (}2^{31} - 1\text{)})</td>
</tr>
<tr>
<td>float</td>
<td>(1.18 \times 10^{-38}) to (3.40 \times 10^{38})</td>
</tr>
<tr>
<td>char</td>
<td>0 to 255</td>
</tr>
</tbody>
</table>

- Mismatched “types.” For example, the literal constant 3293 is of type int (16-bit integer). However, the literal constant 3293.0 is of type float. Although Dynamic C can handle some type mismatches, avoiding type mismatches is the best practice.

- Counting up from, or down to, one instead of zero. In software, ordinal series often begin or terminate with zero, not one.

- Confusing a function’s definition with an instance of its use in a listing.

- Not ending statements with semicolons.

- Not inserting commas as required in functions’ parameter lists.

- Leaving out ASCII space character between characters forming a different legal, but unwanted operator.

- Confusing similar-looking operators such as & & with &, == with =, and // with /.

- Inadvertently inserting ASCII nonprinting characters into a source-code file.
Appendix B provides comprehensive PK2200 physical, electronic, and environmental specifications.
General Specifications

Table B-1 lists the electrical, mechanical, and environmental specifications for the PK2200.

**Table B-1. PK2200 General Specifications**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating Temp</td>
<td>−40°C to 70°C</td>
</tr>
<tr>
<td>Humidity</td>
<td>5% to 95%, noncondensing</td>
</tr>
<tr>
<td>Input Voltage</td>
<td>9 V to 24 V DC</td>
</tr>
<tr>
<td>Digital Inputs</td>
<td>16 protected, −20 V to +24 V DC</td>
</tr>
<tr>
<td>Digital Outputs</td>
<td>14 high-current sinking (500 mA max.) or sourcing (250 mA max.).</td>
</tr>
<tr>
<td>Processor</td>
<td>Z80180</td>
</tr>
<tr>
<td>Clock</td>
<td>9.216 MHz or 18.432 MHz.</td>
</tr>
<tr>
<td>SRAM</td>
<td>32K standard, 512K maximum</td>
</tr>
<tr>
<td>EEPROM</td>
<td>512 bytes</td>
</tr>
<tr>
<td>Flash EPROM</td>
<td>Up to 256K</td>
</tr>
<tr>
<td>Serial ports</td>
<td>2 RS-232 or 1 RS-232 with RTS/CTS and 1 RS-485</td>
</tr>
<tr>
<td>Serial rate</td>
<td>Up to 115,200 bps</td>
</tr>
<tr>
<td>Watchdog/supervisor</td>
<td>Yes</td>
</tr>
<tr>
<td>Time/date clock</td>
<td>Yes</td>
</tr>
<tr>
<td>Backup battery</td>
<td>Yes, internal 3 V DC lithium ion</td>
</tr>
</tbody>
</table>
Hardware Mechanical Dimensions

Top view for models PK2200 and PK2210.

Figure B-1. Top View PK2200 and PK2210
Top view for model PK2240.
End view for models PK2200, PK2210 and PK2240.

The board dimensions are 4.0"×5.32" overall. The centers of the mounting holes are inset (0.220", 0.770") from the corners of the board. They are 2.46" and 4.88" on center. Mounting holes are 0.160" in diameter.
Top view of models PK2220 and PK2230.

*Figure B-4. Top View PK2220 and PK2230*
High Voltage Driver Specifications

Table B-2. Sinking Driver Specifications

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Absolute Maximum Rating at 25°C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output Voltage</td>
<td>50 V DC</td>
</tr>
<tr>
<td>Output Current</td>
<td>500 mA</td>
</tr>
<tr>
<td>Power Dissipation (Change)</td>
<td>1.0 W</td>
</tr>
<tr>
<td>Power Dissipation (Package)</td>
<td>2.25 W</td>
</tr>
<tr>
<td>C-E Saturation Voltage (max.)</td>
<td>1.3 V</td>
</tr>
<tr>
<td>Derating Factor</td>
<td>18.18 mW/°C above 25°C</td>
</tr>
</tbody>
</table>

Table B-3. Sourcing Driver Specifications

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Absolute Maximum Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output Voltage</td>
<td>30V DC</td>
</tr>
<tr>
<td>Output Current</td>
<td>250 mA</td>
</tr>
<tr>
<td>Power Dissipation (Chan)</td>
<td>1.0 W</td>
</tr>
<tr>
<td>Power Dissipation (Package)</td>
<td>2.2 W</td>
</tr>
<tr>
<td>C-E Saturation Voltage (maximum)</td>
<td>1.2 V</td>
</tr>
<tr>
<td>Derating Factor</td>
<td>18 mW/°C above 25°C</td>
</tr>
</tbody>
</table>

Environmental Temperature Constraints

No special precautions are necessary over the range of 0°C to 50°C (32°F to 122°F). For operation at temperatures below 0°C, the PK2200 should be equipped with a low temperature LCD that is specified for operation down to –20°C. The heating effect of the power dissipated by the unit may be sufficient to keep the temperature above 0°C, depending on the enclosure’s insulating capability. The LCD storage temperature is 20°C lower than its operating temperature, which may protect the LCD in case the power should fail, thus removing the heat source. The LCD unit is specified for a maximum operating temperature of 50°C. Except for the LCD, which fades at higher temperatures, the PK2200 operates at 60°C or more without problem.

External loads and expansion cards can increase power consumption.
Connectors

Only a single, solid conductor should be placed in a screw clamp terminal. Bare copper, particularly if exposed to the air for a long period before installation, can become oxidized. The oxide can cause a high resistance (~20 \( \Omega \)) connection, especially if the clamping pressure is not sufficient. To avoid this, use tinned wires or clean, shiny copper wire. If you are using multiple conductors or stranded wire, consider soldering the wire bundle or using a crimp connector to avoid a loss of contact pressure to a spontaneous rearrangement of the wire bundle at a latter time. Soldering may make the wire subject to fatigue failure at the junction with the solder if there is flexing or vibration.

Header Locations and Jumper Settings

Figure B-5 illustrates the location of the headers on the PK2200. Table B-4 lists each header and explains possible pin connections.

---

*Figure B-5 PK2200 Jumpers and Headers*
### Table B-4. Headers and Jumper Settings

<table>
<thead>
<tr>
<th>Header</th>
<th>Pins</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JP1</td>
<td>1–3, 2–4</td>
<td>Sink/source control. The drivers will be damaged if the jumpers are set incorrectly.</td>
</tr>
<tr>
<td></td>
<td>1–2, 3–4</td>
<td>Connect for the ULN2803 sinking drivers (default).</td>
</tr>
<tr>
<td></td>
<td>1–2, 3–4</td>
<td>Connect for the UDN2985A sourcing drivers.</td>
</tr>
<tr>
<td>JP2</td>
<td>1–3, 2–4</td>
<td><strong>EPROM</strong> 32K</td>
</tr>
<tr>
<td></td>
<td>3–5, 2–4</td>
<td>flash EPROM 64K</td>
</tr>
<tr>
<td></td>
<td>3–5, 4–6</td>
<td>128K 128K</td>
</tr>
<tr>
<td></td>
<td>3–5, 4–6</td>
<td>256K 256K</td>
</tr>
<tr>
<td></td>
<td>3–5, 4–6</td>
<td>512K</td>
</tr>
<tr>
<td></td>
<td>7–9</td>
<td><strong>Input pullup/pulldown resistors</strong></td>
</tr>
<tr>
<td></td>
<td>9–11</td>
<td>Inputs 1–4 and 9–12 are pulled up.</td>
</tr>
<tr>
<td></td>
<td>8–10</td>
<td>Inputs 5–8 and 13–16 are pulled up.</td>
</tr>
<tr>
<td></td>
<td>10–12</td>
<td>Inputs 5–8 and 13–16 are pulled down.</td>
</tr>
<tr>
<td>JP3</td>
<td>1–2</td>
<td><strong>Miscellaneous</strong> Enables the watchdog timer (default).</td>
</tr>
<tr>
<td></td>
<td>3–4</td>
<td>Allows the CTS line to reset the board.</td>
</tr>
<tr>
<td></td>
<td>5–6, 9–11</td>
<td><strong>Serial Communication</strong> One 5-wire RS-232 channel (Z180 Port 0) with RTS/CTS</td>
</tr>
<tr>
<td></td>
<td>7–8</td>
<td>One 3-wire RS-232</td>
</tr>
<tr>
<td></td>
<td>5–6, 7–8, 9–11</td>
<td>One RS-485 channel (Port 1)</td>
</tr>
<tr>
<td></td>
<td>5–7, 11–13</td>
<td>Two 3-wire RS-232 channels</td>
</tr>
<tr>
<td>JP4</td>
<td>10–12</td>
<td><strong>SRAM sizing</strong> 32K or 128K SRAM (default)</td>
</tr>
<tr>
<td></td>
<td>12–14</td>
<td>512K SRAM</td>
</tr>
<tr>
<td></td>
<td>7–8</td>
<td>Readable jumper equivalent to mode-setting keypad keys. JP4 overrides the keypad if jumper installed.</td>
</tr>
<tr>
<td></td>
<td>6–7</td>
<td>Places unit in program mode at 19,200 bps.</td>
</tr>
<tr>
<td></td>
<td>2–3</td>
<td>Runs the program.</td>
</tr>
<tr>
<td></td>
<td>4–5</td>
<td>Places unit in program mode at 28,800 bps.</td>
</tr>
<tr>
<td></td>
<td>2–3</td>
<td>indicates to Dynamic C that watchdog timer is enabled. Connect when JP3:1-2 is installed.</td>
</tr>
<tr>
<td>JP5</td>
<td>1–2</td>
<td>Write protect the EEPROM.</td>
</tr>
<tr>
<td></td>
<td>2–3</td>
<td>Write-enable the EEPROM.</td>
</tr>
<tr>
<td>JP6</td>
<td>1–2</td>
<td>EPROM</td>
</tr>
<tr>
<td></td>
<td>2–3</td>
<td>flash EPROM</td>
</tr>
</tbody>
</table>
Appendix C provides information about power management and hardware and software specific to power management on the PK2200.
Power Failure Detection Circuitry

Figure C-1 shows the power failure detection circuitry of the PK2200.

![Diagram of PK2200 Power-Fail Circuit]

**Figure C-1. PK2200 Power-Fail Circuit**

**Power Failure Sequence of Events**

The following events occur as the input power fails:

1. The 691 power-management IC first triggers a power-failure /NMI (non-maskable interrupt) when the unregulated DC input voltage falls below approximately 7.9 V (as determined by the voltage divider R1/R2).

2. At some point, the raw input voltage level will not exceed the required regulated voltage level by the regulator’s dropout voltage whereupon the regulated output begins to droop.

3. The 691 next triggers a system reset when the regulated +5 V supply falls below ∼4.75 V, allowing your power-failure routine the “holdup” interval, t_H, to store your important state data.

4. The 691 forces the chip enable of the SRAM high (standby mode).

5. The time/date clock and SRAM switches to the lithium backup battery when the regulated voltage falls below the battery voltage of approximately 3 V.

6. The 691 keeps the reset asserted until the regulated voltage drops below 1 V.

7. At this point the 691 ceases operating. By this time, the portion of the circuitry not battery-backed has long since ceased functioning.
The ratio of your power supply’s output capacitor’s value to your circuit’s current draw determines the actual duration of the holdup-time interval, $t_H$.

Figure C-2. Power Fail Sequence of Events

This setup can fail when multiple power fluctuations happen rapidly — a common occurrence in the real world. If the PK2200’s Z180 processor receives multiple /NMI s, it overwrites an internal register, making a correct return from the first /NMI impossible. Also, depending on the number of fluctuations of the raw DC input (and hence, the number of stacked /NMI s), the processor’s stack could overflow, corrupting your program’s code or data.

When the Z180 senses an NMI, it saves the program counter (PC) on its processor stack. It copies the maskable interrupt flag, IEF1, to IEF2 and zeroes IEF1. The Z180 restores IEF2’s saved state information when it executes a RETN (Return from Nonmaskable Interrupt) instruction.
Recommended Power Fail Routine

Z-World recommends the following routines to handle an NMI. The routines monitor the state of the /PFO line, via U18 and the data bus, to determine if the brownout condition is continuing or if the power has returned to normal levels. If you use one of these routines, you need not worry about multiple power-failure /NMIs because these routines never return from the first /NMI unless the power returns.

Program C-1. Suggested Power Fail Routine

```c
main(){
   ...
}
...
char dummy[24]; // reserve dummy stack
   // for /NMI processing
...
#define NMI_BIT 3  // routine will test data
#define NMI PIODB2 // bit 3 to determine
                // state of /NMI line
#JUMP_VEC NMI_VEC myint
#asm
myint::
   ld   sp,dummy+24 ; force stack pointer
                  ; to top of “dummy”
                  ; array to prevent
                  ; overwriting of code
                  ; or data
   ; do whatever service, within allowable
   ; execution time
loop:
   call hitwd     ; make sure no
                  ; watchdog reset
                  ; during brownout
   ld   bc,NMI    ; load the read-NMI
                  ; register to bc
   in   a,(c)     ; read the read-NMI
                  ; register to /PFO
   bit  NMI_BIT, a ; check /PFO status
   jr   z,loop    ; wait until brownout
                  ; condition clears
timeout:
   ; then... a tight loop
   ; to force a watchdog
   ; timeout
   jp   timeout   ; which will reset the
                  ; Z180
#endasm
```
The watchdog timer should be enabled. However, if the watchdog is not enabled, you can force the processor to restart execution at 0x0000. Substitute this section for the one labeled “timeout” above.

**Program C-2. Alternate Power Fail Code**

```assembly
restart:
    ld   a,0xe2      ; make sure 0x0000
    ; points to start of
    ; EPROM BIOS
    out0 (CBAR),a   ; set the CBAR
    jp   0000h       ; jump to logical
    ; (physical) address
    ; 0x0000
#endasm
```

If the DC input voltage continues to decrease, the controller powers down. The routine calls hitwd to make sure that watchdog does not timeout and thereby reset the processor. The controller can continue to run, after a fashion, at low voltage and might not be able to detect the low voltage condition, because the Z180’s /NMI input needs to see a high-to-low transition edge.

A situation similar to a brownout occurs if the power supply is overloaded. For example, when an LED turns on, the raw voltage supplied to the PK2200 may dip below 7.9 V. The interrupt routine does a shutdown, which turns off the LED, clearing the problem. However, if the cause of the overload persists, the system oscillates, alternately experiencing an overload and then resetting. To correct this situation, use a power supply which can provide the needed current and voltage.

A few milliseconds of computing time remain when the regulated +5 V supply falls below $\approx 4.75$ V, even if power cuts off abruptly. The amount of time depends on the size of the capacitors in the power supply. The standard wall power supply provides about 10 ms. If you remove the power cable abruptly from the PK2200 side, only the capacitors on the board are available, reducing computing time to a few hundred microseconds. These times can vary considerably depending on system’s configuration and loads on the 5 V or 9 V supplies.

The interval between the power failure detection and entry to the power-failure interrupt routine is approximately 100 μs or less if Dynamic C NMI communication is not in use.
Blank
Appendix D provides a suggested interrupt vector map and information on EEPROM address, processor I/O addresses, and peripheral addresses.
Most of the following interrupt vectors can be altered under program control. The addresses are given in hex, relative to the start of the interrupt vector page, as determined by the contents of the I-register. These are the default interrupt vectors set by the boot code in the Dynamic C EPROM.

**Interrupt Vectors**

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

```c
#INT_VEC 0x10 myfunction
```

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

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

<table>
<thead>
<tr>
<th>Address</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>INT1_VEC</td>
<td>Expansion bus attention /INT1 vector</td>
</tr>
<tr>
<td>0x02</td>
<td>INT2_VEC</td>
<td>/INT2 vector</td>
</tr>
<tr>
<td>0x04</td>
<td>PRT0_VEC</td>
<td>PRT timer channel 0</td>
</tr>
<tr>
<td>0x06</td>
<td>PRT1_VEC</td>
<td>PRT timer channel 1</td>
</tr>
<tr>
<td>0x08</td>
<td>DMA0_VEC</td>
<td>DMA channel 0</td>
</tr>
<tr>
<td>0x0A</td>
<td>DMA1_VEC</td>
<td>DMA channel 1</td>
</tr>
<tr>
<td>0x0C</td>
<td>CSIO_VEC</td>
<td>Clocked Serial I/O</td>
</tr>
<tr>
<td>0x0E</td>
<td>SER0_VEC</td>
<td>Asynchronous Serial Port Channel 0</td>
</tr>
<tr>
<td>0x10</td>
<td>SER1_VEC</td>
<td>Asynchronous Serial Port Channel 1</td>
</tr>
</tbody>
</table>

**Table D-1. Z180 Internal Device Interrupt Vectors**

Digital input 11 connects to /INT0 and digital input 12 connects to /INT2, allowing external events to generate interrupts.
Jump Vectors

These special interrupts occur in a different manner: instead of loading the address of the interrupt routine from the interrupt vector, these interrupts cause a jump directly to the address of the vector, which contains a jump instruction to the interrupt routine. For example,

\[ \text{0x66 non-maskable power-failure interrupt} \]

Because nonmaskable interrupts can be used for Dynamic C communication, your interrupt vector for power failure is normally stored just in front of the Dynamic C program. You can store a vector there by using the following command:

\[ \text{#JUMP_VEC NMI_VEC name} \]

The Dynamic C communication routines relay to this vector when a power failure causes the NMI rather than a serial interrupt. Table D-2 lists interrupt priorities from the highest to lowest priority.

<table>
<thead>
<tr>
<th>Interrupt Priorities</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Highest Priority)</td>
</tr>
<tr>
<td>Trap (Illegal Instruction)</td>
</tr>
<tr>
<td>NMI (Nonmaskable Interrupt)</td>
</tr>
<tr>
<td>INT 0 (Maskable Interrupt, Level 0, 3 modes, PIO interrupts)</td>
</tr>
<tr>
<td>INT 1 (Maskable Interrupt, Level 1, PLCBus attention line interrupt)</td>
</tr>
<tr>
<td>INT 2 (Maskable Interrupt, Level 2)</td>
</tr>
<tr>
<td>PRT Timer Channel 0</td>
</tr>
<tr>
<td>PRT Timer Channel 1</td>
</tr>
<tr>
<td>DMA Channel 0</td>
</tr>
<tr>
<td>DMA Channel 1</td>
</tr>
<tr>
<td>Clocked Serial I/O</td>
</tr>
<tr>
<td>Serial Port 0</td>
</tr>
<tr>
<td>(Lowest Priority)</td>
</tr>
<tr>
<td>Serial Port 1</td>
</tr>
</tbody>
</table>
EEPROM Addresses

These EEPROM constants apply to the standard PK2200.

**Table D-3. Z180 I/O Device Register Addresses**

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x000</td>
<td>Startup Mode. If 1, enter program mode. If 8, execute loaded program at startup.</td>
</tr>
<tr>
<td>0x001</td>
<td>Baud rate in units of 1200 baud.</td>
</tr>
<tr>
<td>0x100</td>
<td>Unit “serial number.” BCD time and date with the following format: second, minutes, hours, day, month, year.</td>
</tr>
<tr>
<td>0x108</td>
<td>Microprocessor clock speed in units of 1200 Hz (16-bits). For 9.216 MHz clock speed, this value is 7680. For 18.432 MHz, this value is 15,360.</td>
</tr>
<tr>
<td>0x16C</td>
<td>Long coefficient relating speed of microprocessor clock relative to speed of real-time clock. Nominal value is 107,374,182 which is 1/40 of a second microprocessor clock time on the scale where $2^{32}$ is 1 second. This value requires 4 bytes of EEPROM, stored least byte first.</td>
</tr>
</tbody>
</table>
### Processor Register Addresses

The Z180’s I/O-device registers occupy the first 40 addresses.

#### Table D-4. Z180 Internal I/O Device Registers

<table>
<thead>
<tr>
<th>Address</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>CNTLA0</td>
<td>Serial Channel 0, Control Register A</td>
</tr>
<tr>
<td>0x01</td>
<td>CNTLA1</td>
<td>Serial Channel 1, Control Register A</td>
</tr>
<tr>
<td>0x02</td>
<td>CNTLB0</td>
<td>Serial Channel 0, Control Register B</td>
</tr>
<tr>
<td>0x03</td>
<td>CNTLB1</td>
<td>Serial Channel 1, Control Register B</td>
</tr>
<tr>
<td>0x04</td>
<td>STAT0</td>
<td>Status Register, Serial Channel 0</td>
</tr>
<tr>
<td>0x05</td>
<td>STAT1</td>
<td>Status Register, Serial Channel 1</td>
</tr>
<tr>
<td>0x06</td>
<td>TDR0</td>
<td>Transmit Data Register, Serial Channel 0</td>
</tr>
<tr>
<td>0x07</td>
<td>TDR1</td>
<td>Transmit Data Register, Serial Channel 1</td>
</tr>
<tr>
<td>0x08</td>
<td>RDR0</td>
<td>Receive Data Register, Serial Channel 0</td>
</tr>
<tr>
<td>0x09</td>
<td>RDR1</td>
<td>Receive Data Register, Serial Channel 1</td>
</tr>
<tr>
<td>0x0A</td>
<td>CNTR</td>
<td>Clocked Serial Control Register</td>
</tr>
<tr>
<td>0x0B</td>
<td>TRDR</td>
<td>Clocked Serial Data Register</td>
</tr>
<tr>
<td>0x0C</td>
<td>TMDR0L</td>
<td>Timer Data Register, Channel 0, low</td>
</tr>
<tr>
<td>0x0D</td>
<td>TMDR0H</td>
<td>Timer Data Register, Channel 0, high</td>
</tr>
<tr>
<td>0x0E</td>
<td>RLDR0L</td>
<td>Timer Reload Register, Channel 0, low</td>
</tr>
<tr>
<td>0x0F</td>
<td>RLDR0H</td>
<td>Timer Reload Register, Channel 0, high</td>
</tr>
<tr>
<td>0x10</td>
<td>TCR</td>
<td>Timer Control Register</td>
</tr>
<tr>
<td>0x11–13</td>
<td>—</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x14</td>
<td>TMDR1L</td>
<td>Timer Data Register, Channel 1, low</td>
</tr>
<tr>
<td>0x15</td>
<td>TMDR1H</td>
<td>Timer Data Register, Channel 1, high</td>
</tr>
<tr>
<td>0x16</td>
<td>RLDR1L</td>
<td>Timer Reload Register, Channel 1, low</td>
</tr>
<tr>
<td>0x17</td>
<td>RLDR1H</td>
<td>Timer Reload Register, Channel 1, high</td>
</tr>
<tr>
<td>0x18</td>
<td>FRC</td>
<td>Free-Running Counter</td>
</tr>
<tr>
<td>0x19–1E</td>
<td>—</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x1F</td>
<td>CCR</td>
<td>CPU control register for the 18 MHz chip. Write 0x80 to get 18.432 MHz. Write 0 to get 9.216 MHz.</td>
</tr>
<tr>
<td>0x20</td>
<td>SAR0L</td>
<td>DMA Source Address, Channel 0, low</td>
</tr>
</tbody>
</table>

continued…
### Table D-4. Z180 Internal I/O Device Registers (concluded)

<table>
<thead>
<tr>
<th>Address</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x21</td>
<td>SAR0H</td>
<td>DMA Source Address, Channel 0, high</td>
</tr>
<tr>
<td>0x22</td>
<td>SAR0B</td>
<td>DMA Source Address, Channel 0, extra bits</td>
</tr>
<tr>
<td>0x23</td>
<td>DAR0L</td>
<td>DMA Destination Address, Channel 0, low</td>
</tr>
<tr>
<td>0x24</td>
<td>DAR0H</td>
<td>DMA Destination Address, Channel 0, most</td>
</tr>
<tr>
<td>0x25</td>
<td>DAR0B</td>
<td>Destination Address, Channel 0, extra bits</td>
</tr>
<tr>
<td>0x26</td>
<td>BCR0L</td>
<td>DMA Byte Count Register, Channel 0, low</td>
</tr>
<tr>
<td>0x27</td>
<td>BCR0H</td>
<td>DMA Byte Count Register, Channel 0, high</td>
</tr>
<tr>
<td>0x28</td>
<td>MAR1L</td>
<td>DMA Memory Address Register, Channel 1, low</td>
</tr>
<tr>
<td>0x29</td>
<td>MAR1H</td>
<td>DMA Memory Address Register, Channel 1, high</td>
</tr>
<tr>
<td>0x2A</td>
<td>MAR1B</td>
<td>DMA Memory Address Register, Channel 1, extra bits</td>
</tr>
<tr>
<td>0x2B</td>
<td>IAR1L</td>
<td>DMA I/O Address Register, Channel 1, low</td>
</tr>
<tr>
<td>0x2C</td>
<td>IAR1H</td>
<td>DMA I/O Address Register, Channel 1, high</td>
</tr>
<tr>
<td>0x2D</td>
<td>—</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x2E</td>
<td>BCR1L</td>
<td>DMA Byte Count Register, Channel 1, low</td>
</tr>
<tr>
<td>0x2F</td>
<td>BCR1H</td>
<td>DMA Byte Count Register, Channel 1, high</td>
</tr>
<tr>
<td>0x30</td>
<td>DSTAT</td>
<td>DMA Status Register</td>
</tr>
<tr>
<td>0x31</td>
<td>DMODE</td>
<td>DMA Mode Register</td>
</tr>
<tr>
<td>0x32</td>
<td>DCNTL</td>
<td>DMA/WAIT Control Register</td>
</tr>
<tr>
<td>0x33</td>
<td>IL</td>
<td>Interrupt Vector Low Register</td>
</tr>
<tr>
<td>0x34</td>
<td>ITC</td>
<td>Interrupt/Trap Control Register</td>
</tr>
<tr>
<td>0x35</td>
<td>—</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x36</td>
<td>RCR</td>
<td>Refresh Control Register</td>
</tr>
<tr>
<td>0x37</td>
<td>—</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x38</td>
<td>CBR</td>
<td>MMU Common Base Register</td>
</tr>
<tr>
<td>0x39</td>
<td>BBR</td>
<td>MMU Bank Base Register</td>
</tr>
<tr>
<td>0x3A</td>
<td>CBAR</td>
<td>MMU Common/Bank Area Register</td>
</tr>
<tr>
<td>0x3B–3D</td>
<td>—</td>
<td>Reserved</td>
</tr>
<tr>
<td>0x3E</td>
<td>OMCR</td>
<td>Operation Mode Control Register</td>
</tr>
<tr>
<td>0x3F</td>
<td>ICR</td>
<td>I/O Control Register</td>
</tr>
</tbody>
</table>
PK2200 Peripheral Addresses

The following addresses control the I/O devices that are external to the Z180 processor.

**Table D-5. PK2200 External I/O Device Registers**

<table>
<thead>
<tr>
<th>Address</th>
<th>Bit(s)</th>
<th>Symbol</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x40</td>
<td>7</td>
<td>WDOG</td>
<td>Watchdog is “hit” (when JP3:1-2) by setting bit 7 of this address.</td>
</tr>
<tr>
<td>0x60</td>
<td>7</td>
<td>LED</td>
<td>Turns on LED by setting bit 7 of this address. Turn off by clearing bit 7.</td>
</tr>
<tr>
<td>0x80</td>
<td>7</td>
<td>SCL</td>
<td>EEPROM clock bit. Set the clock high by setting bit 7 of this address, and low by clearing bit 7.</td>
</tr>
<tr>
<td>0xA0</td>
<td>7</td>
<td>SDA_W</td>
<td>EEPROM serial data, write. Send data in bit 7.</td>
</tr>
<tr>
<td>0xC0</td>
<td>0–7</td>
<td>BUSRD0</td>
<td>First read, PLC expansion bus</td>
</tr>
<tr>
<td>0xC2</td>
<td>0–7</td>
<td>BUSRD1</td>
<td>Second read, PLC expansion bus</td>
</tr>
<tr>
<td>0xC4</td>
<td>0–7</td>
<td>BUSSPARSE</td>
<td>Spare read, PLC expansion bus</td>
</tr>
<tr>
<td>0xC6</td>
<td>—</td>
<td>BUSRESET</td>
<td>Read this address to reset all devices on expansion bus</td>
</tr>
<tr>
<td>0xC8</td>
<td>0–7</td>
<td>BUSADR0</td>
<td>PLC expansion bus, first address byte</td>
</tr>
<tr>
<td>0xCA</td>
<td>0–7</td>
<td>BUSADR1</td>
<td>PLC expansion bus, second address byte</td>
</tr>
<tr>
<td>0xCC</td>
<td>0–7</td>
<td>BUSADR2</td>
<td>PLC expansion bus, third address byte</td>
</tr>
<tr>
<td>0xCE</td>
<td>0–7</td>
<td>BUSWR</td>
<td>Expansion bus write to port</td>
</tr>
<tr>
<td>0xE0</td>
<td>0–7</td>
<td>LCDRD</td>
<td>LCD read/write register, control</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LCDWR</td>
<td></td>
</tr>
<tr>
<td>0xE1</td>
<td>0–7</td>
<td>LCDRD+1</td>
<td>LCD read/write register, data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LCDWR+1</td>
<td></td>
</tr>
</tbody>
</table>

continued…
Table D-5. PK2200 External I/O Device Registers (continued)

<table>
<thead>
<tr>
<th>Address</th>
<th>Bit(s)</th>
<th>Symbol</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x100</td>
<td>0–3</td>
<td>RTALE</td>
<td>Real-time clock, address register</td>
</tr>
<tr>
<td>0x120</td>
<td>0–3</td>
<td>RTRW</td>
<td>Real-time clock, read/write data register</td>
</tr>
<tr>
<td>0x140</td>
<td>7</td>
<td>BUZZER</td>
<td>Self-resonating buzzer. Set bit 7 to turn on. Clear bit 7 to turn off.</td>
</tr>
<tr>
<td>0x160</td>
<td>7</td>
<td>ENB485</td>
<td>Set bit 7 to enable RS-485 channel. Clear bit 7 to disable.</td>
</tr>
<tr>
<td>0x180</td>
<td>0–7</td>
<td>DIGBANK1</td>
<td>Digital Input, Bank 1. Bit 0 corresponds to input 1; bit 7 corresponds to input 8.</td>
</tr>
<tr>
<td>0x181</td>
<td>0–7</td>
<td>DIGBANK2</td>
<td>Digital Input, Bank 2. Bit 0 corresponds to input 9; bit 7 corresponds to input 16.</td>
</tr>
<tr>
<td>0x1A1</td>
<td>4–7</td>
<td>KROW1L</td>
<td>Keypad drive row 1, rightmost 4 keys. Bit 4 is rightmost key. Bit 5 is key next to that, etc. Row 1 is the bottom-most row.</td>
</tr>
<tr>
<td>0x1A2</td>
<td>4–7</td>
<td>KROW2L</td>
<td>Keypad drive row 2, rightmost 4 keys. Bit 4 is rightmost key. Bit 5 is key next to that, etc.</td>
</tr>
<tr>
<td>0x1A4</td>
<td>4–7</td>
<td>KROW3L</td>
<td>If there were support for a 4x6 keypad, this would be drive row 3. As of now, it reads jumper JP4.</td>
</tr>
<tr>
<td>0x1A8</td>
<td>4–7</td>
<td>KROW4L</td>
<td>As of now, this address is reserved.</td>
</tr>
<tr>
<td>0x1AF</td>
<td>4–7</td>
<td>KROWAL</td>
<td>Keypad, all rows, rightmost 4 keys. Bit 4 is rightmost key. Bit 5 is key next to that, etc. Row 1 is the bottom-most row.</td>
</tr>
<tr>
<td>0x1B0</td>
<td>6-7</td>
<td>SDA_R NMI</td>
<td>Bit 7 represents the EEPROM SDA line. Bit 6 presents the power-failure (NMI) state.</td>
</tr>
</tbody>
</table>

continued…
Table D-5. PK2200 External I/O Device Registers (continued)

<table>
<thead>
<tr>
<th>Address</th>
<th>Bit(s)</th>
<th>Symbol</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xB1</td>
<td>4–7</td>
<td>KROW1H</td>
<td>Keypad drive row 1, leftmost 2 keys. Bit 5 is leftmost key. Bit 4 is key next to that. Bit 7 represents EEPROM SDA line. Bit 6 presents power-failure (NMI) state. Row 1 is bottom-most row.</td>
</tr>
<tr>
<td>0xB 2</td>
<td>4–7</td>
<td>KROW2H</td>
<td>Keypad drive row 2, leftmost 2 keys. Bit 5 is leftmost key. Bit 4 is next. Bit 7 represents EEPROM SDA line. Bit 6 presents power-failure (NMI) state.</td>
</tr>
<tr>
<td>0xB4</td>
<td>4–7</td>
<td>KROW3H</td>
<td>If there were support for a 4×6 keypad, this would be drive row 3. As of now, bits 4 and 5 are reserved. Bit 7 is EEPROM SDA line. Bit 6 presents power-failure (NMI) state.</td>
</tr>
<tr>
<td>0xB8</td>
<td>4–7</td>
<td>KROW4H</td>
<td>If there were support for a 4×6 keypad, this would be drive row 4. As of now, bits 4 and 5 are reserved. Bit 7 represents EEPROM SDA line. Bit 6 is power-failure (NMI) state.</td>
</tr>
<tr>
<td>0xBF</td>
<td>4–7</td>
<td>KROWAH</td>
<td>Keypad, all rows, leftmost 2 keys. Bit 5 is the leftmost key. Bit 4 is the key next to that. Bit 7 represents the EEPROM SDA line. Bit 6 presents the power-failure (NMI) state. Row 1 is the bottom-most row.</td>
</tr>
<tr>
<td>0xC0</td>
<td>5</td>
<td>DRV1</td>
<td>Digital output 1. Writing 0x20 turns on output. Writing 0 turns off output.</td>
</tr>
<tr>
<td>0xC1</td>
<td>5</td>
<td>DRV2</td>
<td>Digital output 2. Writing 0x20 turns on output. Writing 0 turns off output.</td>
</tr>
<tr>
<td>0xC2</td>
<td>5</td>
<td>DRV3</td>
<td>Digital output 3. Writing 0x20 turns on output. Writing 0 turns off output.</td>
</tr>
</tbody>
</table>

continued...
<table>
<thead>
<tr>
<th>Address</th>
<th>Bit(s)</th>
<th>Symbol</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1C3</td>
<td>5</td>
<td>DRV4</td>
<td>Digital output 4. Writing 0x20 turns on output. Writing 0 turns off output.</td>
</tr>
<tr>
<td>0x1C4</td>
<td>5</td>
<td>DRV5</td>
<td>Digital output 5. Writing 0x20 turns on output. Writing 0 turns off output.</td>
</tr>
<tr>
<td>0x1C5</td>
<td>5</td>
<td>DRV6</td>
<td>Digital output 6. Writing 0x20 turns on output. Writing 0 turns off output.</td>
</tr>
<tr>
<td>0x1C6</td>
<td>5</td>
<td>DRV7</td>
<td>Digital output 7. Writing 0x20 turns on output. Writing 0 turns off output.</td>
</tr>
<tr>
<td>0x1C7</td>
<td>5</td>
<td>DRV8</td>
<td>Digital output 8. Writing 0x20 turns on output. Writing 0 turns off output.</td>
</tr>
<tr>
<td>0x1E0</td>
<td>6</td>
<td>DRV9</td>
<td>Digital output 9. Writing 0x40 turns on Outputs. Writing 0 turns off Outputs.</td>
</tr>
<tr>
<td>0x1E1</td>
<td>6</td>
<td>DRV10</td>
<td>Digital Outputs 10. Writing 0x40 turns on output. Writing 0 turns off output.</td>
</tr>
<tr>
<td>0x1E2</td>
<td>6</td>
<td>DRV11</td>
<td>Digital output 11. Writing 0x40 turns on output. Writing 0 turns off output.</td>
</tr>
<tr>
<td>0x1E3</td>
<td>6</td>
<td>DRV12</td>
<td>Digital output 12. Writing 0x40 turns on output. Writing 0 turns off output.</td>
</tr>
<tr>
<td>0x1E4</td>
<td>6</td>
<td>DRV13</td>
<td>Digital output 13. Writing 0x40 turns on output. Writing 0 turns off output.</td>
</tr>
<tr>
<td>0x1E5</td>
<td>6</td>
<td>DRV14</td>
<td>Digital output 14. Writing 0x40 turns on output. Writing 0 turns off output.</td>
</tr>
</tbody>
</table>
Appendix E provides the pin assignments for the PLCBus, describes the registers, and lists the software drivers.
PLCBus Overview

The PLCBus is a general-purpose expansion bus for Z-World controllers. The PLCBus is available on the BL1200, BL1600, BL1700, PK2100, PK220, and PK2600 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 relay expansion capability to all controllers through their digital outputs.

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

### Table E-1. Z-World PLCBus Expansion Devices

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

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

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

Figure E-1. PLCBus Pin Diagram
Two independent buses, the LCD bus and the PLCBus, exist on the single connector.

The LCD bus consists of the following lines.

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

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

![Figure E-2. OP6000 Connection to PLCBus Port](image)

The PLCBus consists of the following lines.

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

The PLCBus may be used as a 4-bit bus (D0X–D3X) or as an 8-bit bus (D0X–D7X). Whether it is used as a 4-bit bus or an 8-bit bus depends on the encoding of the address placed on the bus. Some PLCBus expansion cards require 4-bit addressing and others (such as the XP8700) require 8-bit addressing. These devices may be mixed on a single bus.
There are eight registers corresponding to the modes determined by bus lines A1X, A2X, and A3X. The registers are listed in Table E-2.

### Table E-2. PLCBus Registers

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

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

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

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

If eight data lines are used, then the addressing possibilities of the bus become much greater—more than 256 million addresses according to the conventions established for the bus.
Place an address on the bus by writing (bytes) to BUSADR0, BUSADR1 and BUSADR2 in succession. Since 4-bit and 8-bit addressing modes must coexist, the lower four bits of the first address byte (written to BUSADR0) identify addressing categories, and distinguish 4-bit and 8-bit modes from each other.

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

### Table E-3. First-Level PLCBus Address Coding

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

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.

<table>
<thead>
<tr>
<th>SHBUS0</th>
<th>SHBUS0+1</th>
<th>SHBUS1</th>
<th>SHBUS1+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bus expansion</td>
<td>BUSADR0</td>
<td>BUSADR1</td>
<td>BUSADR2</td>
</tr>
</tbody>
</table>

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

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

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

**Allocation of Devices on the Bus**

**4-Bit Devices**

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

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

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:

<table>
<thead>
<tr>
<th>bit 3</th>
<th>bit 2</th>
<th>bit 1</th>
<th>bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>A2</td>
<td>A1</td>
<td>A0</td>
<td>D</td>
</tr>
</tbody>
</table>

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

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

**8-Bit Devices**

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

**Expansion Bus Software**

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

**Table E-5. Dynamic C PLCBus Libraries**

<table>
<thead>
<tr>
<th>Library Needed</th>
<th>Controller</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRIVERS.LIB</td>
<td>All controllers</td>
</tr>
<tr>
<td>EZIOTGPL.LIB</td>
<td>BL1000</td>
</tr>
<tr>
<td>EZIOLGPL.LIB</td>
<td>BL1100</td>
</tr>
<tr>
<td>EZIOMGPL.LIB</td>
<td>BL1400, BL1500</td>
</tr>
<tr>
<td>EZIOPLC.LIB</td>
<td>BL1200, BL1600, PK2100, PK2200, ZB4100</td>
</tr>
<tr>
<td>EZIOPLC2.LIB</td>
<td>BL1700, PK2600</td>
</tr>
<tr>
<td>PBUS_TG.LIB</td>
<td>BL1000</td>
</tr>
<tr>
<td>PBUS_LG.LIB</td>
<td>BL1100, BL1300</td>
</tr>
<tr>
<td>PLC_EXP.LIB</td>
<td>BL1200, BL1600, PK2100, PK2200</td>
</tr>
</tbody>
</table>
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 BUSADRx 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`.
Blank
**Battery Life and Storage Conditions**

The ten-year estimated life of a battery on the PK2200 is based on typical use. Most systems are operated on a continuous basis with the battery only powering the SRAM and real time clock during power outages and/or routine maintenance. A ten-year life expectancy is an estimate that reflects the shelf-life of a lithium battery with occasional usage rather than the ability of the battery to power the circuitry full time.

The battery on the PK2200 has a 165 mA·h capacity. Older versions of the PK2200 have a Toshiba clock that consumes 8 µA in idle mode. Newer boards have an Epson clock that consumes 3 µA in idle mode. In standby mode, SRAM consumes from a low of 1 µA (32K SRAM) to a high of 8 µA (512K SRAM). If a system were unpowered 100 percent of the time, the battery life with a Toshiba clock will be approximately 18,300 hours (2.1 years), and with an Epson clock will be approximately 41,250 hours (4.7 years). All life-expectancy ranges are based on normal operating temperatures of 25°C.

Backup time longevity is affected by many factors, including the amount of time the controller is not powered, and the SRAM size. To help achieve a full ten years of backup, a larger capacity cell can replace the BR2325. Alkaline batteries (mounted external to the board, like in many PCs) can easily and cheaply give over ten years of backup.

The controller should be stored at room temperature in the factory packaging until field installation. Take care that the controller is not exposed to extreme temperature, humidity, and/or contaminants such as dust and chemicals.

To ensure maximum battery shelf life, follow proper storage procedures. Replacement batteries should be kept sealed in the factory packaging at room temperature until installation. Protection against environmental extremes will help maximize battery life.

**Replacing Soldered Lithium Battery**

Use the following steps to replace the battery.

1. Locate the three pins on the bottom side of the printed circuit board that secure the battery to the board.

2. Carefully de-solder the pins and remove the battery. Use a solder sucker to clean up the holes.

3. Install the new battery and solder it to the board. Use only a Panasonic BR2325-1GM or equivalent.
Battery Cautions

- Caution (English)
  There is a danger of explosion if battery is incorrectly replaced. Replace only with the same or equivalent type recommended by the manufacturer. Dispose of used batteries according to the manufacturer’s instructions.

- Warnung (German)

- Attention (French)
  Il y a danger d’explosion si la remplacement de la batterie est incorrect. Remplacez uniquement avec une batterie du même type ou d’un type équivalent recommandé par le fabricant. Mettez au rebut les batteries usagées conformément aux instructions du fabricant.

- Cuidado (Spanish)
  Peligro de explosión si la pila es instalada incorrectamente. Reemplace solamente con una similar o de tipo equivalente a la que el fabricante recomienda. Deshágase de las pilas usadas de acuerdo con las instrucciones del fabricante.

- Waarschuwing (Dutch)
  Explosiegevaar indien de batterij niet goed wordt vervangen. Vervanging alleen door een zelfde of equivalent type als aanbevolen door de fabrikant. Gebruikte batterijen afvoeren als door de fabrikant wordt aangegeven.

- Varning (Swedish)
Symbols

#INT_VEC ............................................ 41, 86
#JUMP_VEC ........................................ 87
/AT .................................................. 97
/DREQ1 ........................................... 26
/INT0 ............................................. 26
/INT2 ............................................. 26
/RDX ............................................... 97
/STBX ............................................ 97
/WRX ............................................... 97
<_Ctrl F9> ......................................... 20
=(assignment) use ................................ 68
_DMAFLAG0 ........................................ 53
_DMAFLAG1 ........................................ 53
4-bit bus operations .......... 97, 98, 100
5 × 3 addressing mode ........ 99
5KEYCODE . C ................................. 62
5KEYDEMO . C .................................. 62
5KEYLAD . C ..................................... 62
8-bit bus operations .......... 97, 99, 101
9th-bit binary protocol ........ 63

A

A0X .................................................. 97
A1X, A2X, A3X ................................. 97, 98
addresses
  digital input .......................... 52
  EEPROM ...................................... 88
  encoding ................................... 99
  I/O register ............................. 89
  modes ....................................... 99
  peripheral I/O ......................... 91
  PLCBus .................................... 98, 99
ASCII characters
  and modem commands ........ 39
asynchronous serial port
  Channel 0 ................................. 86
  Channel 1 ................................. 86

attention line .................................. 97
attention line interrupt ............. 87
B

background routine .................. 100
battery
  backup ..................................... 80
  replacing .................................. 108
baud rate ................................. 29, 88
BBR ............................................... 89
BCR0H ........................................... 89
BCR0L ........................................... 89
BCR1H ........................................... 89
BCR1L ........................................... 89
beeper
  sounding .................................. 55
bidirectional data lines ............ 97
bitmapped graphics ................ 47
bitmaps ....................................... 47
board dimensions .................... 73
board jumpers ........................... 34
brownout ................................. 83
buffer
  receive .................................... 38, 39
  transmit ................................... 38
bus
  control registers .................. 101
digital inputs ......................... 101
expansion
  96, 97, 98, 99, 100, 101
  4-bit drivers ............................ 102
  8-bit drivers ............................ 104
  addresses ............................... 100
devices ................................. 100, 101
functions 102, 103, 104, 105
rules for devices .................. 100
software drivers ..................... 101
LCD ............................................ 97
bus (continued)
operations
   4-bit ................. 97, 98, 100
   8-bit ................. 97, 101
BUSADR0 ............ 91, 98, 99
BUSADR1 ............ 91, 98, 99
BUSADR2 ............ 91, 98, 99
BUSADR3 ............ 104, 105
BUSRD0 91, 101, 102, 103, 105
BUSRD1 ............ 91, 101, 102
BUSRESET .......... 91
BUSSPARE .......... 91
BUSWR ............ 91, 102
BUZZER ............ 91
byte aligned .......... 47

C

calibration constants .......... 88
carriage return (CR)
as modem command terminator
........................................ 39
CBAR ................. 89
CBR ................. 89
CCR ................. 89
CE compliance .......... 16
CKA0/DREQ0 .......... 26
clock
   real time ............ 86
   real-time .......... 88, 91
   system .......... 29, 88
time/date ............ 50
clock frequency
system ............ 88
clocked serial control register ... 89
clocked serial data register ... 89
clocked serial I/O .... 86, 87
CN2 ............ 26
CNTLA0 .......... 89
CNTLA1 .......... 89
CNTLB0 .......... 89
CNTLB1 .......... 89
CNTR .......... 89
column major .......... 47
COMMAND mode
   modem communication .......... 39
common problems
   programming errors .......... 68
communication
   Dynamic C ............ 87
error ................ 19
   initialization routines .... 40
   RS-232 ............ 29, 63
   RS-485 ............ 29, 63
   serial ............ 29, 38, 63
   interrupts ........ 38
Compile
   icon ............ 20
   program ............ 20
connect PK2200 to PC ........ 18
connectors ........ 42, 76
   26-pin PLCBus
   pin assignments .......... 96
constants
   calibration .......... 88
   core ............ 23
   counters .......... 26, 36
   DMA ............ 53
   inputs ........ 36, 53
   CPU ............ 23
   CSIO_VEC ........ 86
   CSREMOTE.C .......... 63
   CTS ........ 38, 39, 40
   CUARTREM.C .......... 63
customization .......... 15
   CZ0REM.C .......... 63

D

D0X–D7X .......... 97
DAR0B ............ 89
DAR0L ............ 89
DATA mode
   modem communication .......... 39
date and time .......... 50, 63, 88
DCNTL ........ 89
Development Kit .......... 15
DIGBANK1 .......... 91
DIGBANK2 ......................... 91
digital inputs ........ 24, 26, 36, 62
drivers ...................... 52
reading ...................... 52
multiple inputs .......... 52
virtual driver variables .. 52
digital outputs .... 27, 37, 54, 91
drivers ...................... 52
virtual driver variables .. 54
writing ..................... 54
Dinit_uart ................. 40
Dinit_z0 .................... 40
Dinit_z1 .................... 40
DIP relays ................ 96
display .................... 31, 43
liquid crystal. See LCD
dissipation
heat ......................... 75
DMA ......................... 89
DMA channels .......... 26, 36, 53
DMA Channel 0 .......... 86, 87
DMA Channel 1 .......... 86, 87
DMA counter .............. 53
DMA interrupts .......... 53
DMA/WAIT .................. 89
DMA0_VEC .................. 86
DMA1_VEC .................. 86
DMA0_Count ............... 53
DMA1_Count ............... 53
DMASnapShot ............. 53
DMODE ..................... 89
downloading data ...... 38, 39
downloading programs .. 41
drivers
digital input ........ 52
digital output ........ 52
expansion bus .......... 101
4-bit ..................... 102
8-bit ..................... 104
high voltage .......... 27
relay ...................... 101
sinking ................... 27
software ................. 50
sourcing ................ 28

DRIVERS.LIB .................. 101
dropout voltage .......... 80
DRV10 ...................... 91
DRV11 ...................... 91
DRV4 ...................... 91
DRV5 ...................... 91
DRV6 ...................... 91
DRV7 ...................... 91
DRV8 ...................... 91
DRV9 ...................... 91
DSTAT ...................... 89
DTR ...................... 40
Dynamic C
communication .......... 87
error messages .......... 19
programming ........... 34

E
echo option ............. 38, 39
dropout voltage .......... 80
drivers
digital input ........ 52
digital output ........ 52
expansion bus .......... 101
4-bit ..................... 102
8-bit ..................... 104
high voltage .......... 27
relay ...................... 101
sinking ................... 27
software ................. 50
sourcing ................ 28

PK2200 Index • 113
expansion boards
reset .......................................... 102
expansion bus ... 86, 87, 91, 96–101
4-bit drivers ................. 102
8-bit drivers ................. 104
addresses ................... 100
devices ................... 100, 101
digital inputs ................. 101
functions ................... 102–105
rules for devices ............ 100
software drivers .......... 101
expansion register ............ 100
EZIOLGPL.LIB ............ 101
EZIOMGPL.LIB ............ 101
EZIOPL2.LIB ............ 101
EZIOPLC.LIB ............ 101
EZIOTGPL.LIB ............ 101
EZIOTGPL.LIB ............ 101

F

F3 .............................. 20
F9 .............................. 20
features ....................... 12
five-key system
   sample program .............. 62
flash EPROM .................. 36
float
   use .............................. 68
fonts ......................... 58, 59
format
   bitmapped image .......... 47
FRC .............................. 89
frequency
   system clock .............. 88
function libraries .......... 98

G

glBlankscreen ..................... 57
glFontInit ..................... 58
glInit ......................... 57
glPlotCircle .................... 60
glPlotDot .................... 57
glPlotLine .................... 57
glPrintf ...................... 60
glPutBitmap .................... 57
glPutFont ..................... 59
glSetBrushType ............... 56
glVPrintf ..................... 59
glxFontInit ..................... 59
glxPutBitmap .................... 58

H

halt program ...................... 20
handshaking
   RS-232 ................................. 38
Hayes Smart Modem ............ 40
headers ........................... 76
heat dissipation .............. 75
high-current output drivers ... 22
high-voltage drivers .......... 27
holdup time ......................... 81
hooking up the PK2200 .......... 19

I

I/O addresses ..................... 89, 91
I/O control register ............ 89
I/O devices ................... 91
I/O map ......................... 89, 91
I/O register addresses ........ 89
IAR1H ......................... 89
IAR1L ......................... 89
ICR ............................ 89
IL ............................ 89
illegal instruction interrupt ... 87
initial PK2200 setup .......... 34
initialization routines
   communication ................ 40
inport ................... 52, 102, 103, 105
inputs
   counter ..................... 53
digital .................. 24, 26, 36, 62
inserts
   keypad ..................... 44
int
   type specifier, use .......... 68
INT1 .......................................... 87
INT1_VEC .................................. 86
INT2 .......................................... 87
INT2_VEC .................................. 86
interface
   operator .................................. 31
interrupt handling for Z180 Port 0 .. 41
interrupt routines ................. 83, 87
interrupt vector
   CSIO_VEC ................................ 86
   DMA0_VEC ................................ 86
   DMA1_VEC ................................ 86
   INT1_VEC ................................ 86
   INT2_VEC ................................ 86
   PRT0_VEC ................................ 86
   PRT1_VEC ................................ 86
   SER0_VEC ................................ 86
   SER1_VEC ................................ 86
interrupt vector low register ....... 89
interrupt vectors ...................... 87
   default .................................. 86
interrupt/trap control register .... 89
interrupts .... 26, 36, 86, 87, 97, 100
   attention line ......................... 87
   DMA ...................................... 53
   illegal instruction ................... 87
   nonmaskable ............................ 83, 87
   power-fail .............................. 80, 83, 87
   routines ................................ 100
   serial ................................... 41, 87
   serial communication ............... 38
   T0 output .............................. 87
   ITC ...................................... 89

J
jumper settings .................. 34, 77, 86
   JP1 (digital outputs) .......... 28, 77
   JP2 (digital inputs) .......... 25, 77
   JP3 (run/program mode) ...... 34, 35
   JP3 (serial communication) .... 30, 77
   JP4 (run/program mode) ...... 77
   JP5 (EEPROM write-protect) 77
   JP6 (EPROM/flash EPROM) 77

K
kdiELSw .................................... 60
kdiSetContrast ............................ 60
keypad ................................. 31, 34, 55, 91
   2 x 6 .................................. 34
   4 x 3 .................................. 35
   initialization ...................... 55, 61, 62
   mode settings ...................... 34
   reading .............................. 61, 62
   states ................................. 61
   keypad inserts ....................... 44
   KP.LIB ................................ 55
   kpDefGetKey ........................... 62
   kpDefInit .............................. 62
   kpDefStChgPn ......................... 61
   kpInit ................................. 61
   kpScanState ........................... 61
   KROW1L ................................ 91
   KROW2L ................................ 91
   KROW3L ................................ 91
   KROW4L ................................ 91
   KROWAL ................................ 91

L
lc_char .................................. 55
lc_cmd .................................... 56
lc_ctrl .................................... 56
lc_init .................................... 56
lc_init_keypad ....................... 55
lc_kxget ................................ 55
lc_kxinit ............................... 55
lc_nl ..................................... 56
lc_pos .................................... 56
lc_printf .............................. 56
lc_setbeep ....................... 55
lc_wait .................................... 56
LCD ..................................... 46, 55, 91, 97
   backlighting ....................... 43
   busy ................................... 56
   character ............................ 43
   graphic ............................... 43
   backlighting ....................... 60
   clear display ....................... 57
LCD
graphic (continued)
  contrast ......................... 60
  drawing ........... 56, 57, 58, 60
  font initialization ...... 58, 59
  initialization .............. 57
  turning display ON/OFF ... 60
  writing ...................... 59, 60
  initialization .............. 56
  positioning text .......... 56
  writing ...................... 55, 56
  commands .................... 56
LCD bus ............................. 97
LCDRD ..................................... 91
LCDWR ..................................... 91
LCDX ........................................ 97
leap year .............................. 50
LED .................................... 83, 91
libraries
  function .......................... 98
liquid crystal display. See LCD
literal (C term)
  use .................................. 68
lithium battery .................. 108

M
MAR1B .................................. 89
MAR1H .................................. 89
MAR1L .................................. 89
master-slave communication .... 63
mechanical dimensions ...... 71–74
MMU bank base register ...... 89
MMU common base register ... 89
MMU common/bank area register ........................................... 89
mode
  startup ............................ 88
MODEM .................................. 38, 39
modem commands .............. 39
  termination ...................... 39
modem communication ...... 41, 63
  serial link wiring .............. 40

modes
  addressing ...................... 99
  operating ........................ 34

N
NMI .............................. 80, 83, 87, 91
NMI_VEC .............................. 87
NO_CARRIER message .......... 39
nonmaskable interrupt. See NMI
NULL modem ..................... 40

O
OMCR .............................. 89
operating modes ................. 34
operation mode control register ... 89
operator interface .................. 31
Opto22 9th-bit binary protocol ... 63
outport .............................. 54, 102, 103, 105
outputs
  digital ...................... 27, 37, 54, 91
overload protection ............ 24

P
peripheral I/O addresses ........ 91
pinouts
  PLCBus .............................. 96
PK2200
  connect to PC ................. 18
  establish communication ...... 19
  features .......................... 12
  hook-up .......................... 19
  subsystems ...................... 21
PLCBus .................. 86, 87, 91, 96–98,
  100, 101
  26-pin connector
    pin assignments ............... 96
  4-bit operations ............. 97, 99
  8-bit operations ............. 97, 99
  addresses ...................... 98, 99
  memory-mapped I/O register . 98
  reading data .................. 98
PLCBus (continued)
  relays
  DIP ..................................... 96
  drivers .................................. 101
  writing data ............................ 98
ports
  serial ...................................... 38
power failure
  detection ................................ 80
  interrupts .................. 80, 83, 87
  recommended routine ............ 82
power management .............. 79
power supply ........................ 18, 83
  overload .................................. 83
program
  run ........................................ 20
program development ............ 35
programming
  remote .................................... 41
programming cable .................... 18
protected digital inputs .............. 22
protocol
  9th-bit binary ......................... 63
  Opto22 .................................. 63
  XMODEM ...................... 38, 39
PRT timer
  Channel 0 ........................ 86, 87
  Channel 1 ........................ 86, 87
PRT0_VEC .................................. 86
PRT1_VEC .................................. 86
pulse measurement .................. 37

R
R
RAM ........................................ 23, 35
RCR .......................................... 89
RDR0 ......................................... 89
RDR1 ......................................... 89
read12data ............................. 103
read24data .................................. 105
read4data .................................. 104
read8data .................................. 105
reading data on the PLCBus ........
 ........................................... 98, 103
real-time clock ... 24, 50, 86, 88, 91
  reading ................................... 50
  writing ................................... 50
real-time kernel ....................... 62
receive buffer ...................... 38, 39
receive data register ............. 89
refresh control register .......... 89
regulated input voltage .......... 80
RELOAD_VEC .................................. 41
remote downloading ............ 41, 63
reset
  controller .............................. 19
  expansion boards ................. 102
  system .................................... 80
RLDR0H .................................... 89
RLDR0L .................................... 89
RLDR1H .................................... 89
RLDR1L .................................... 89
RS-232 .................................... 29, 38
  communication .......................... 63
  expansion card ....... 38, 39, 40
  handshaking ............................ 38
  jumper settings ........................ 30
RS-485 .................................... 29
  channel .................................. 91
  communication .......................... 63
  jumper settings ........................ 30
  network .................................. 42
RTALE ....................................... 91
RTK .......................................... 62
RTRW ........................................ 91
RTS ........................................ 38, 39, 40
RX line ...................................... 40
S
sample program .................. 20, 62
SAR0B .................................... 89
SAR0H .................................... 89
SAR0L .................................... 89
SCL ......................................... 91
screw terminals ................... 42, 76
SDA_R ...................................... 91
SDA_W ...................................... 91
SE1100 ........................................... 96
select PLCBus address .................. 102
SER0_VEC .................................. 41, 86
SER1_VEC .................................. 86
Serial Channel 0
 control register B .................... 89
 receive data register ............... 89
 status register ......................... 89
 transmit data register .............. 89
Serial Channel 1
 control register A .................... 89
 control register B .................... 89
 receive data register ............... 89
 status register ......................... 89
 transmit data register .............. 89
serial channels
 configuring ................................ 30
serial communication ... 29, 38, 63
 channels ................................ 22
serial interrupts ...................... 41, 87
serial link wiring ..................... 40
Serial Port 0 ............................ 87
Serial Port 1 ............................ 87
serial ports ................................ 38
set12adr .................................. 102
set16adr .................................. 102
set24adr .................................. 104
set4adr .................................. 103
set8adr .................................. 105
shadow registers ..................... 100
sinking drivers ....................... 27
software
 drivers ................................ 50
 libraries ................................ 62, 98
source (C term)
 use ........................................ 68
sourcing drivers ...................... 28
specifications .......................... 69
electrical and environmental .. 70
stack overflow ......................... 81
standby mode .......................... 80
startup mode ........................... 88
STAT0 ...................................... 89
STAT1 ...................................... 89
structure tm .......................... 50
supervisor ................................ 23
system clock ......................... 88
 frequency .............................. 29, 88
T
T0 output interrupt .................... 87
target not responding ................ 19
TCR ......................................... 89
TDR0 ....................................... 89
TDR1 ....................................... 89
templates
 keypad insert .......................... 44
text
displaying on graphic LCD ...........
 ........................................ 59, 60
time and date .......................... 50, 63, 88
time/date clock ....................... 50
timer control register ............... 89
timer data clock register
 Channel 0 ................................ 89
 Channel 1 ................................ 89
timer reload register
 Channel 0 ................................ 89
 Channel 1 ................................ 89
tm ......................................... 50
tm_rd .................................... 50
tm_wr .................................... 50
TMDR0H .................................. 89
TMDR0L .................................. 89
TMDR1H .................................. 89
TMDR1L .................................. 89
transmit buffer ....................... 38
transmit data register .............. 89
trap ........................................ 87
TRDR ...................................... 89
troubleshooting
 baud rate .............................. 67
cables .................................... 66
com port ................................. 66, 67
troubleshooting (continued)
  communication mode .......... 67
  expansion boards .......... 66
  grounds ......................... 66
  memory size ....................... 67
  operating mode .................. 67
  repeated resets ............... 67
  TX line .......................... 40

U
  UDN2985A ......................... 27, 28
  ULN2803 ............................ 27
  unregulated input voltage 80
  up_bEEP ....................... 55
  up_digin ....................... 52
  up_Setout ....................... 54
  uploading data ............... 38, 39

V
  VIOInit ............................... 52
  VIODrv ............................... 52

W
  watchdog .......................... 23
  WDOG ................................. 91
  WINTEK.LIB .......................... 55

write12data .................... 104
write24data .................... 105
write4data .................... 104
write8data .................... 105
writing data on the PLCBus .... 98, 104
wtDisplaySw ...................... 60

X
  XMODEM .......................... 38, 39
  XMODEM download ............. 63
  XMODEM upload .................. 63
  XP8100 .............................. 96
  XP8200 .............................. 96
  XP8300 .............................. 96
  XP8400 .............................. 96
  XP8500 .............................. 96
  XP8600 .............................. 96
  XP8700 .............................. 96, 97, 101
  XP8800 .............................. 96
  XP8900 .............................. 96

Z
  Z180 Port 0 .......................... 38–41
  interrupt handling ............ 41
  Z180 Port 1 ...................... 38, 86
Blank