

**Technical Reference Manual for  
the ASCBDEV Developer's Kit**

**from  
Pine Instrument Company  
Grove City, Pennsylvania**

**© Copyright 1996-1998 by Pine Instrument Company  
All Rights Reserved**

*Manual: LMCBP1T  
Revision: 004  
Date: Dec 1998*

## **PINE INSTRUMENT COMPANY SOFTWARE LICENSE AGREEMENT**

You should carefully read the terms and conditions on this page and the next page before opening this package. Opening the diskette envelope indicates your acceptance of these terms and conditions. If you do not agree with them, you must promptly return the unopened package and documentation.

**License:** Pine Instrument Company grants you a non-exclusive license to use the enclosed computer program subject to the terms and conditions set forth below. You assume responsibility for the selection of the program to achieve your intended results, and for the installation, use and results obtained with the program. You may use this program on a single computer only. You may copy the program for backup or archive purposes only, in support of its use on a single computer. You may transfer this program and license to another party if the other party agrees to accept the terms and conditions of this Agreement and if you transfer all copies of the program or documentation to that party.

**Restrictions:** You may not distribute copies of the program or documentation to others or electronically transfer the program or documentation from one computer to another over a network. The program contains trade secrets and in order to protect them, you may not decompile, reverse engineer, disassemble or otherwise reduce the program to a human-understandable form. You may not modify, adapt, translate, rent, lease, loan, resell for profit, distribute, network or create derivative works based on all or any part of the program or documentation

**Copyright:** The enclosed computer program and its manual are copyrighted. You may not copy or otherwise reproduce any part of the program or manual except for personal archive purposes.

**Title:** The enclosed computer program and its manual, including partial copies, and translations, are the property of Pine Instrument Company. You have only the limited rights granted by this license. You are not an owner of the program or documentation and 17 U.S.C. section 117 does not apply.

**Term:** The license is effective until terminated. You may terminate the license at any time by destroying the program and documentation, together with all copies, partial copies, updates, and translations. The license also terminates if you fail to comply with any term or condition of this License Agreement. Upon such termination, you agree to destroy the program and documentation, together with all copies, partial copies, updates, and translations. You agree that this License Agreement is the complete and sole statement of the agreement between us concerning the use of the program and supersedes any proposal, oral or written statement or prior agreement between us. No variation of the terms and conditions of this agreement will be enforceable against Pine Instrument Company unless Pine Instrument Company specifically agrees in writing.

**Use Limitation:** This software is intended for use only with electrochemical test equipment manufactured by Pine Instrument Company. You agree that you shall never use this software as part of any laboratory, clinical, or experimental study or procedure involving human test subjects, such as, but not limited to, studies in which human test subjects are in electrical contact with electrochemical test equipment being controlled by this software.

**Limitation of Liability:** Neither Pine Instrument Company nor anyone who has been involved in the creation, production, or distribution of this software and documentation shall be liable for any direct, incidental, or consequential damages, such as, but not limited to, loss of anticipated profits or benefits, resulting from the use of the program or as a result of a breach of warranty. Further, neither Pine Instrument company nor anyone who has been involved in the creation, production, or distribution of this software and documentation shall be liable for any direct, incidental, or consequential damages from use of this software as part of any laboratory, clinical, or experimental study or procedure involving human test subjects. Some states allow the exclusion or limitation of direct, incidental, or consequential damages, so the above limitation may not apply to you.

**Limited Warranty:** The program and documentation is provided "as is" without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability, title, and fitness for a particular purpose. The entire risk as to the quality and performance of the program and documentation is with you. Should the program prove defective, you (and not Pine Instrument Company nor its dealers) assume the entire cost of all necessary servicing, repair or correction. Some states do not allow the exclusion of implied warranties so the above exclusion may not apply to you. This warranty gives you specific legal rights and you may also have other rights which vary from state to state.

Pine Instrument Company does not warrant that the documentation or the functions contained in the program will meet your requirements or that the operation of the program will be uninterrupted or error free. However, Pine Instrument Company warrants the diskette(s) on which the program is furnished to be free from defects in materials and workmanship under normal use for a period of ninety (90) days from the date of delivery to you as evidenced by a copy of your receipt. Pine Instrument Company's entire liability and your exclusive remedy will be the replacement of any diskette not meeting this limited warranty and which is returned to Pine Instrument Company with a copy of your receipt, or if Pine Instrument Company is unable to deliver a replacement diskette which is free of defects in material or workmanship, you may terminate this Agreement by returning the program and documentation to Pine Instrument Company.

No oral or written information or advice given by Pine Instrument Company, its dealers, distributors, agents, or employees will create a warranty or in any way increase the scope of this warranty, and you may not rely on any such information or advice.

## Table of Contents

<b>INTRODUCTION .....</b>	<b>1</b>
<b>SCOPE OF THIS MANUAL .....</b>	<b>1</b>
<b>INTERNET RESOURCES .....</b>	<b>1</b>
<b>COPYRIGHT .....</b>	<b>1</b>
<b>TRADEMARKS.....</b>	<b>1</b>
<b>INTERFACE DESCRIPTION.....</b>	<b>2</b>
<b>THE INTERFACE BOARD .....</b>	<b>2</b>
<b>THE INTERFACE CABLE.....</b>	<b>3</b>
<b>ANALOG INPUT CHANNELS.....</b>	<b>5</b>
Measuring Potentials .....	5
Measuring Currents .....	6
Monitoring the Sweep Generator .....	6
Overload Conditions .....	6
<b>ANALOG OUTPUT CHANNELS.....</b>	<b>7</b>
<b>DIGITAL COMMUNICATIONS.....</b>	<b>7</b>
Integer Representation within the AFCBP1 Bipotentiostat .....	8
Generating Handshaking Strokes using NIDAQ.....	8
Communications Timing Parameters .....	8
Checksum Computation .....	9
<b>COMMUNICATIONS LIBRARY .....</b>	<b>12</b>
<b>THE FUNCTIONS.....</b>	<b>12</b>
PCOMM_Initialize.....	12
PCOMM_SetDefaults .....	12
PCOMM_ResetBoard.....	12
PCOMM_Hello .....	13
PCOMM_SendMessage .....	13
PCOMM_SetCommuncationsTiming .....	13
PCOMM_GetCommuncationsTiming .....	14
PCOMM_ResetCommuncationsTiming.....	14
<b>GENERAL VARIABLES .....</b>	<b>14</b>
pcomm_Command .....	14
pcomm_GalPot .....	15
pcomm_Open.....	15
pcomm_DummyNormal.....	15
pcomm_NoConnect.....	15
pcomm_K1SweepState, pcomm_K2SweepState.....	16
pcomm_K1OffsetState, pcomm_K2OffsetState.....	16
pcomm_K1Range, pcomm_K2Range .....	16
<b>SWEEP GENERATOR VARIABLES .....</b>	<b>17</b>
pcomm_PosSweepRate .....	17
pcomm_NegSweepRate.....	17
pcomm_ManSweepDir .....	17
pcomm_SweepHold .....	17
pcomm_StopAtLower, pcomm_StopAtUpper .....	17
pcomm_SweepZero .....	17
pcomm_InitPot .....	17
pcomm_UpperLimit, pcomm_LowerLimit .....	18
pcomm_FinalPot .....	18
pcomm_NumLegs .....	18
<b>TIMING VARIABLES .....</b>	<b>18</b>
pcomm_BeforeDelay .....	18
pcomm_AcqDelay .....	18

---

pcomm_AcqLength.....	19
pcomm_DisengageDelay.....	19
<b>RETURN CODES.....</b>	<b>20</b>
CBP_DEVICE_NUMBER_ERROR (-29).....	20
CBP_DEVICE_UNSUPPORTED (-35).....	20
CBP_XMT_CHECKSUM_ERROR (-40) .....	20
CBP_RCV_PKT_SHORT (-41) .....	20
CBP_RCV_PKT_LONG (-42) .....	20
CBP_RCV_PKT_TIMEOUT (-43) .....	20
<b>SOURCE CODE DISKETTE .....</b>	<b>21</b>
Using Other Languages.....	21



# ***INTRODUCTION***

## **SCOPE OF THIS MANUAL**

This manual is meant for a person who is experienced in both electrochemistry and computer programming. It describes how to control the AFCBP1 Bipotentiostat using a library of low-level software functions written in the C programming language.

To make effective use of the information in this manual, you need to have a personal computer equipped with a suitable INTERFACE BOARD. The board should be connected to the 50-pin connector on the back side of the AFCBP1 Bipotentiostat. In addition, you should be familiar with the information in the manuals which accompany the INTERFACE BOARD, especially those which describe the NI-DAQ data acquisition library created by *National Instruments Corporation*.

This manual describes the use of the CBPCOM communications library developed for use with the AFCBP1 Bipotentiostat. As of this printing, the CBPCOM library supports the use of about ten different interface boards offered by *National Instrument Corporation*.

## **INTERNET RESOURCES**

The latest versions of the *NIDAQ* device driver libraries can be downloaded from the *National Instruments Corporation* web site ([www.natinst.com](http://www.natinst.com)). The site also offers complete documentation and on-line help services for the *NIDAQ* library.

*Pine Instrument Company* also has a web site ([www.pineinst.com](http://www.pineinst.com)) that offers up-to-date information about our products and software. Technical help may be requested via electronic mail ([info@pineinst.com](mailto:info@pineinst.com)) as well.

## **COPYRIGHT**

Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of *Pine Instrument Company*.

## **TRADEMARKS**

Product and company names used in this manual are trademarks or trade names of their respective companies. *MS-DOS*, *Windows*, and *Plug and Play (PnP)* are all trademarks of *Microsoft Corporation* (Redmond, Washington). *NI-DAQ* is a trademark of *National Instruments Corporation* (Austin, Texas).

## ***INTERFACE DESCRIPTION***

The section of the manual gives a general overview of the connection between a personal computer system and the AFCBP1 BIPOTENTIOSTAT.

### **THE INTERFACE BOARD**

The AFCBP1 Bipotentiostat was originally designed to work with the AT-MIO-16H-9 INTERFACE BOARD manufactured by *National Instruments Corporation* (Austin, TX). This board offers eight analog input (ADC) channels as required to monitor the eight analog signals generated by the AFCBP1 Bipotentiostat. In addition, it has two analog output (DAC) channels that can be used to apply potentials to each of the working electrodes. Finally, it has eight digital I/O lines that can be used for communicating with the 8032 CPU located within the AFCBP1 Bipotentiostat.

Several years ago, *National Instruments Corporation* discontinued the original AT-MIO-16H-9 board because it was based on the Am9513 timing chip (which had also become obsolete). Other boards based on this chip, including the AT-MIO-16L-9, the AT-MIO-16F-9, and the AT-MIO16-X were also discontinued.

A new series of boards, based on the custom DAQ-STC chip, was introduced by *National Instruments Corporation* to replace the older Am9513-based boards. This newer "E-series" offered superior timing capabilities and a wider selection of interface boards. *Pine Instrument Company* chose the AT-MIO-16E-10 as the replacement for the older AT-MIO-16H-9 board. Most software current offered by Pine supports either interface board.

More recent developments in the personal computer industry have required support for even more interface boards. The AT-MIO-16E-10 board is designed for a computer system with an ISA bus slot, and does not work with the newer style PCI slots. To provide support for the newer PCI style interface boards, the communications library described in the manual (CBPCOM) was designed to support the widest possible range of boards. The boards currently supported are listed below:

***ISA boards based on the older Am9513 chip:***

AT-MIO-16H-9, AT-MIO-16L-9, AT-MIO-16F-5, AT-MIO-16X

***ISA boards based on the newer DAQ-STC chip:***

AT-MIO-1E-10, AT-MIO-16E-2, AT-MIO-16E-1, AT-MIO-16XE-10

***PCI boards based on the newer DAQ-STC chip:***

PCI-MIO-16E-1, PCI-MIO-16E-4, PCI-MIO-16XE-10

It is undoubtedly possible to use other types of boards with the AFCBP1 Bipotentiostat as well. If you wish to use some other board, make certain that it has enough ADC, DAC, and digital lines to fully communicate with the instrument. In addition, be prepared to make extensive modifications to the source code for the CBPCOM library to match the particular characteristics of your board. The information in this manual should help you in this process.

## **THE INTERFACE CABLE**

The connection between the INTERFACE BOARD and the AFCBP1 Bipotentiostat is made via a 50-conductor INTERFACE CABLE. There are both analog and digital signal lines in this ribbon cable. The analog lines are used mainly to convey current and potential signals from the AFCBP1 Bipotentiostat to the INTERFACE BOARD. However, two of the analog lines are used to connect the DAC outputs from the INTERFACE BOARD to the K1 and K2 working electrode potential inputs. Finally, eight digital lines allow for control of the AFCBP1 front panel settings and the internal SWEEP GENERATOR.

A complete description of the 50-pin ribbon cable connector is given in Table I. Note that not all of the signal lines are used. Also note that if an E-series interface board is being used, then a 68-to-50 pin CABLE ADAPTER is required to connect the 50-pin ribbon cable to the 68-pin connector on the board. The CABLE ADAPTER maps the appropriate E-series signal lines to the correct positions on the 50-pin ribbon cable. The adapter is available from either *Pine Instrument Company* or *National Instruments Corporation*.

**Table I: Cross-Index for Interface Cable Signal Lines**

pin #	E-series board signal*	Am9513 board Signal	AFCBP1 signal	signal direction	signal description
1	AI GND	AI GND	AGND		analog ground (for board inputs)
2	AI GND	AI GND	AGND		analog ground (for board inputs)
3	ACH00	ACH00	SW	CBP->MIO	output of CBP's internal sweep generator
4	ACH08	ACH08	AGND		analog ground
5	ACH01	ACH01	E1	CBP->MIO	potential of the K1 working electrode
6	ACH09	ACH09	AGND		analog ground
7	ACH02	ACH02	I1	CBP->MIO	current at the K1 working electrode
8	ACH10	ACH10	AGND		analog ground
9	ACH03	ACH03	E2	CBP->MIO	potential of the K2 working electrode
10	ACH11	ACH11	AGND		analog ground
11	ACH04	ACH04	I2	CBP->MIO	current at the K2 working electrode
12	ACH12	ACH12	AGND		analog ground
13	ACH05	ACH05	OH	CBP->MIO	positive overload if greater than +7.5V
14	ACH13	ACH13	AGND		analog ground
15	ACH06	ACH06	OL	CBP->MIO	negative overload if less than -7.5V
16	ACH14	ACH14	AGND		analog ground
17	ACH07	ACH07	SL	CBP->MIO	present target sweep limit
18	ACH15	ACH15	AGND		analog ground
19	AI SENSE	AI SENSE			
20	DAC0OUT	DAC0OUT	K1 INPUT	MIO->CBP	potential applied to K1 working electrode
21	DAC1OUT	DAC1OUT	K2 INPUT	MIO->CBP	potential for K2 or back panel rotator control
22	EXTREF	EXTREF			
23	AO GND	AO GND	AGND		analog ground (for board outputs)
24	DIG GND	DIG GND	DGND		digital signal ground
25	ADIO0	ADIO0	ED0	MIO<>CBP	used for encoded digital communication
26	BDIO0	BDIO0	ED4	MIO<>CBP	used for encoded digital communication
27	ADIO1	ADIO1	ED1	MIO<>CBP	used for encoded digital communication
28	BDIO1	BDIO1	ED5	MIO<>CBP	used for encoded digital communication
29	ADIO2	ADIO2	ED2	MIO<>CBP	used for encoded digital communication
30	BDIO2	BDIO2	ED6	MIO<>CBP	used for encoded digital communication
31	ADIO3	ADIO3	ED3	MIO<>CBP	used for encoded digital communication
32	BDIO3	BDIO3	ED7	MIO<>CBP	used for encoded digital communication
33	DIG GND	DIG GND	DGND		digital signal ground
34	+5V	+5V			
35	+5V	+5V			
36	SCANCLK	SCANCLK			
37	EXTSTROBE	EXTSTRB			
38	PFI0/TRIG1	STRT TRIG		CBP->MIO	signals beginning of sweep program
39	PFI1/TRIG2	STOP TRIG		CBP->MIO	signals the end of sweep program
40	CONVERT	EXTCONV			
41	GPCTR1 SOURCE	SOURCE1			
42	GPCTR1 GATE	GATE1			
43	GPCTR1_OUT	OUT1	XMT STROBE	MIO->CBP	strokes digital data sent to potentiostat
44	PFI5/UPDATE	SOURCE2			
45	PFI6/WFTRIG	GATE2			
46	STARTSCAN	OUT2			
47	GPCTR0 SOURCE	SOURCE5	RCV STROBE	CBP->MIO	strokes digital data sent from potentiostat
48	GPCTR0 GATE	GATE5			
49	GPCTR0_OUT	OUT5			
50	FREQ_OUT	FOUT			

\* Note: Pin numbers for E-series boards assume that a 68-to-50 pin CABLE ADAPTER is in use.

## ANALOG INPUT CHANNELS

Pins 3 through 18 are used for analog input from the AFCBP1 Bipotentiostat to the INTERFACE BOARD. There are eight ADC channels on the board numbered zero through seven (7), and eight analog signals generated by the AFCBP1 are directly connected to these ADC channels. Table II shows the use for each of the eight ADC channels.

The NI-DAQ library provided by *National Instruments Corporation* offers a wide variety of data acquisition functions that can be used to monitor one or more of these ADC channels. The channel scan functions are especially useful for acquiring large amounts of data using a limited number of function calls.

**Table II: Analog Signals Generated by the AFCBP1 Bipotentiostat**

Channel Number	Signal Name (abbreviation)	Description
0	Sweep Output (SW)	This signal always reflects the output level of the AFCBP1's internal analog SWEEP GENERATOR.
1	K1 Potential (E1)	This signal output from the AFCBP1 represents the voltage at the K1 working electrode.
2	K1 Current (I1)	This signal is a voltage proportional to the current flowing at the K1 working electrode. To convert this voltage to a current value requires knowledge of the present setting of the CURRENT CONVERTER.
3	K2 Potential (E2)	This signal output from the AFCBP1 represents the voltage at the K2 working electrode.
4	K2 Current (I2)	This signal is a voltage proportional to the current flowing at the K2 working electrode. To convert this voltage to a current value requires knowledge of the present setting of the CURRENT CONVERTER.
5	Positive Overload (OH)	If the voltage signal on this channel is greater than +7.5V, then a positive overload condition exists.
6	Negative Overload (OL)	If the voltage signal on this channel is less than -7.5V, then a negative overload condition exists.
7	Sweep Direction (SL)	When an analog sweep is in progress, this signal reflects the limit potential towards which the sweep is presently heading. It can be used to distinguish between the various segments in the sweep program.

Of course, the most important analog input signals are E1, I1, E2, and I2 as these are directly related to what is going on in the electrochemical cell. The other signals are used to monitor the status of the AFCBP1 Bipotentiostat and its internal SWEEP GENERATOR.

### ***Measuring Potentials***

The 12-bit ADC circuitry on most interface boards allows analog input signals to be measured with a precision of one part in 4096. Each channel on the board can be set up to measure signal levels over a given voltage range. The most useful voltage ranges are  $\pm 10V$ ,  $\pm 5V$ , and  $\pm 2.5V$ , although even more sensitive ranges are available. The voltage range for each channel is determined by the *gain setting* for that channel. The NI-DAQ software library offers functions that allow you to set the channel gains prior to acquiring data.

If you are monitoring an analog input signal using the  $\pm 10\text{V}$  range, then you can measure the signal with a precision of 4.88 mV. (This is computed by taking the full 20 volt input range and dividing it by 4096.) If you need to make measurements with more precision, then you must use a smaller input range.

Three of the supported boards offer 16-bit ADC circuitry. These boards are the AT-MIO-16X, the AT-MIO-16XE-10, and the PCI-MIO-16XE-10, all of which measure signals with a precision of one part in 65536.

### ***Measuring Currents***

When you are monitoring one of the current signals (I1 or I2), you must know both the ADC channel gain and the present setting of the CURRENT CONVERTER in order to translate the ADC signal level to a meaningful current value. Use the NI-DAQ library functions to set the channel gain, and use the CBPCOM library to set the CURRENT CONVERTER range.

### ***Monitoring the Sweep Generator***

If you use the AFCBP1's internal analog SWEEP GENERATOR during your experiment, you have the option of monitoring the sweep output (SW) directly using ADC channel zero. Of course, if you are applying the sweep output to one of the electrodes, simply monitoring either E1 or E2 usually provides the same information. However, the E1 and E2 signals may differ from the SW signal when an offset potential is being applied to either electrode.

The sweep output generally consists of repetitive sweeping back and forth between two sweep limits. The overall sweep program consists of one or more sweep segments with the direction of the sweep changing from one segment to the next. Depending on your particular application, it may be important for you to know exactly when the sweep direction changes. If this is the case, your program can monitor ADC channel seven (7), which provides a synchronization signal (SL) that is related to the sweep direction.

The SL signal line is held at the limit potential towards which the sweep is presently heading. Whereas the sweep output (SW) is essentially a triangle wave bounded by two limit potentials, the SL signal is a square wave bounded by the same two limit potentials. The SL signal and the SW signal are synchronized with each other so that when the sweep direction changes, the SL signal suddenly steps from one limit potential to the other.

### ***Overload Conditions***

The feedback circuitry at the heart of the AFCBP1 is prone to overloading or serious oscillations under certain experimental circumstances. The most common cause of an overload is when one of the CURRENT CONVERTERS is set at too sensitive a level for the amount of current flowing in the cell. The next most likely cause is when there is too great an impedance between the REF and CE inputs on the front panel. Oscillations can occur whenever rapidly changing waveforms are applied to a cell with peculiar impedance characteristics.

In any of the cases described above, the AFCBP1 warns you that it is in an overload state by activating the OVERLOAD indicator on the front panel. You can design your software to detect this condition by having it monitor ADC channels 5 and 6. If the signal on channel 5 exceeds +7.5 volts or if the signal on channel 6 falls below -7.5 volts, then an overload condition exists.

## **ANALOG OUTPUT CHANNELS**

The INTERFACE BOARD provides two digital-to-analog channels (DACs) that can be used to output potentials to the AFCBP1. The two channels are channel zero and channel one. The output of channel zero is connected directly to the potential input for the K1 working electrode, and the output of channel one is connected directly to the K2 working electrode.

The potential actually applied to a working electrode is the sum of the appropriate DAC output potential, the SWEEP GENERATOR output (if active), and any externally applied potential signal present on the front panel “IN” connectors. Thus, you can use the DAC output channel as an offset potential in conjunction with the SWEEP GENERATOR output and/or some external waveform generator.

In most cases, you will want to disable the OFFSET VOLTAGE controls located on the front panel of the AFCBP1. The CBPCOM library allows you to turn off these controls in order to prevent the user from summing yet another offset potential to the working electrode.

The NI-DAQ library offers a variety of functions that you can use to apply potentials to the working electrodes. The simple “AO\_VWrite” function can apply any potential between -10V and +10V to an electrode. More advanced waveform generation functions are offered by the NI-DAQ library if you wish to perform pulse or staircase voltammetry experiments.

## **DIGITAL COMMUNICATIONS**

The most complex aspect of controlling the AFCBP1 Bipotentiostat is establishing digital communications. Originally, the AFCBP1 was designed for use with the AT-MIO-16H-9 board. This board has only eight (8) digital I/O lines and a few triggering and strobe lines that can be used for handshaking. If you take a quick glance at the front panel of the AFCBP1, it is readily apparent that there are far more than just eight digital controls on it. Obviously, there are not enough digital lines to map each control to its own digital signal.

In order to control all aspects of the AFCBP1 along with its internal SWEEP GENERATOR, there had to be a way to transmit a large amount of digital information from the INTERFACE BOARD to the AFCBP1. It was decided that the best way to do this was to send the information as a message packet. Currently, this message packet consists of 62 octets (bytes) of information plus two additional checksum octets. (The checksum is used to check for transmission errors.)

The message packet is sent to the AFCBP1 one octet at a time using all eight digital lines as output lines. Each time an octet of information is placed on the output lines, a data strobe signal is sent to the AFCBP1 telling it to read the octet. The eight digital lines are on pins 25 to 32 of the INTERFACE CABLE. The strobe signal is sent on pin 43.

Upon receipt of the message packet, the AFCBP1 responds to any commands in the message and configures itself accordingly. It sends a brief reply message back to the INTERFACE BOARD. This reply is only eight (8) octets long. In order to receive this reply, the eight digital lines must be reconfigured for input (as soon as possible) after a message packet is sent to the AFCBP1. The reply message is read one octet at a time, and the AFCBP1 sends a strobe signal back to the INTERFACE BOARD (using pin 47) each time a octet is ready to be read.

### ***Integer Representation within the AFCBP1 Bipotentiostat***

The 8032 processor within the AFCBP1 Bipotentiostat represents 16-bit integers in “big-endian” form while *Windows*-based personal computers store such integers in “little-endian” form. As a result, all of the 16-bit integers in the message packet must be explicitly changed to big-endian form before transmitting the packet to the AFCBP1. Fortunately, this is a simple matter of swapping the order of the two octets that make up a 16-bit integer.

### ***Generating Handshaking Strokes using NIDAQ***

The strobes used for timing digital communications between the board and the AFCBP1 make use of the digital counters found on the Interface Board. There are remarkable differences between the older Am9513 counters and the newer DAQ-STC counters. The CBPCOM library deals with these differences by including two completely different sets of program functions, each designed to work with one of the two counters. The CBPCOM functions listed in the source code include either “Am9513” or “DAQSTC” in their names in order to explicitly indicate the timer chip for which they are intended.

On the older Am9513 boards, counter one (1) is used for sending strobes to the AFCBP1, and counter five (5) is used for receiving strobes. The NI-DAQ device driver functions used to control the counters on this chip are grouped together in the “CTR” family of functions. Consult the *NI-DAQ* documentation for more details on how to use these functions.

On the newer DAQ-STC based boards, general purpose counter one (GPCTR1) is used for sending strobes to the AFCBP1, and general purpose counter zero (GPCTR0) is used for receiving strobes. The NI-DAQ device driver functions used to control the counters on this chip are grouped together in the “GPCTR” family of functions. Consult the *NI-DAQ* documentation for more details on how to use these functions.

### ***Communications Timing Parameters***

Figure 1 and Table III summarize how the two strobe signals and the eight data lines are used during a typical packet transmission event. During periods of inactivity, the transmit strobe (XMT STROBE, pin 43) should be maintained in a logic HIGH (+5 V) state. A high-to-low transition on this line signals the AFCBP1 that a valid data octet is present on the digital lines. Likewise, the AFCBP1 normally maintains the receive strobe (RCV STROBE, pin 47) at a logic HIGH (+5V) state. A high-to-low transition on this line indicates that the AFCBP1 has placed a valid reply packet octet on the digital lines.

The very first octet of a message packet is sent to the AFCBP1 as follows. First, the digital lines on the INTERFACE BOARD are configured for output. Then, the data for the first octet is written to the digital output port on the board. Next, a high-to-low transition on the XMT STROBE line signals the AFCBP1 to read the octet. At this point, the AFCBP1 requires a certain amount of time (the XMT OCTET DWELL TIME,  $t_{x0}$ ) to read and store the octet in its internal buffers.

Subsequent octets sent to the AFCBP1 are transmitted in the same fashion. The total time taken to transmit the entire 64 octet message packet must not exceed 500 milliseconds, or else the AFCBP1 flushes its input buffers and ignores the incomplete packet. This effectively places an upper limit on the interval between octets (XMT OCTET INTERVAL,  $t_{x1}$ ) of about six milliseconds (6 msec).

The final octet in the message packet must remain on the digital output lines for a certain amount

of time (XMT OCTET DWELL TIME,  $t_{x0}$ ) before switching the lines over for use as digital input lines. The delay between transmission of the message packet to the AFCBP1 and the reply packet from the AFCBP1 is about 18 milliseconds (RCV Reply Time,  $t_{r0}$ ). Each of the eight octets in the reply packet is strobed by a high-to-low transition on the RCV STROBE line. Each reply octet remains on the digital lines for about 20 milliseconds (RCV OCTET DWELL,  $t_{r1}$ ). The strobe signal is a LOW (zero volts) pulse lasting about 6.5 milliseconds (RCV STROBE DURATION,  $t_{r3}$ ).

Receipt of the entire reply packet typically takes about 200 milliseconds. After that, the AFCBP1 still needs some additional time to process the message. For this reason, the overall interval between message packets should be at least 300 milliseconds (i.e., XMT PACKET INTERVAL,  $t_{x3}$ ).

**NOTE:** More recent versions of the CBPCOM library vary the length of the transmit strobe pulse to achieve the timing delays required for proper communication. The NIDAQ functions used to control and monitor the counters on the INTERFACE BOARD permit the duration of the HIGH and LOW parts of the XMT STROBE pulse to be of various lengths. By relying on these counters to keep time (rather than using arbitrary delay loops), the control of the communications timing is more deterministic and less prone to arbitrary latencies in code execution time.

### ***Checksum Computation***

There is always a chance that a message packet might be corrupted by electrical noise during transmission. The main purpose of the short reply message from the AFCBP1 is to verify that the message packet was transmitted without error. Each time the AFCBP1 processes an incoming message packet, it validates the packet using a checksum placed in the last two octets of the message packet. If the packet is invalid, then the reply from the AFCBP1 contains a code indicating an error condition.

The CRC checksum algorithm used by the CBPCOM library is somewhat complicated is implemented in the “PCOMM\_XMT\_COMPUTE\_CRC\_CHECKSUM” function. If you use this function without modification, you should have no trouble producing the proper checksum value to place at the end of a message packet.

If, however, you have to implement this function in another language or using a different programming platform, you will need to make certain that your function is producing the right checksum value. To help you in debugging your CRC checksum function, three complete message packets are given in Table IV in hexadecimal. These message packets, as shown, are ready for transmission to the AFCBP1 (that is, they contain big-endian integers and a properly computed checksum value.)

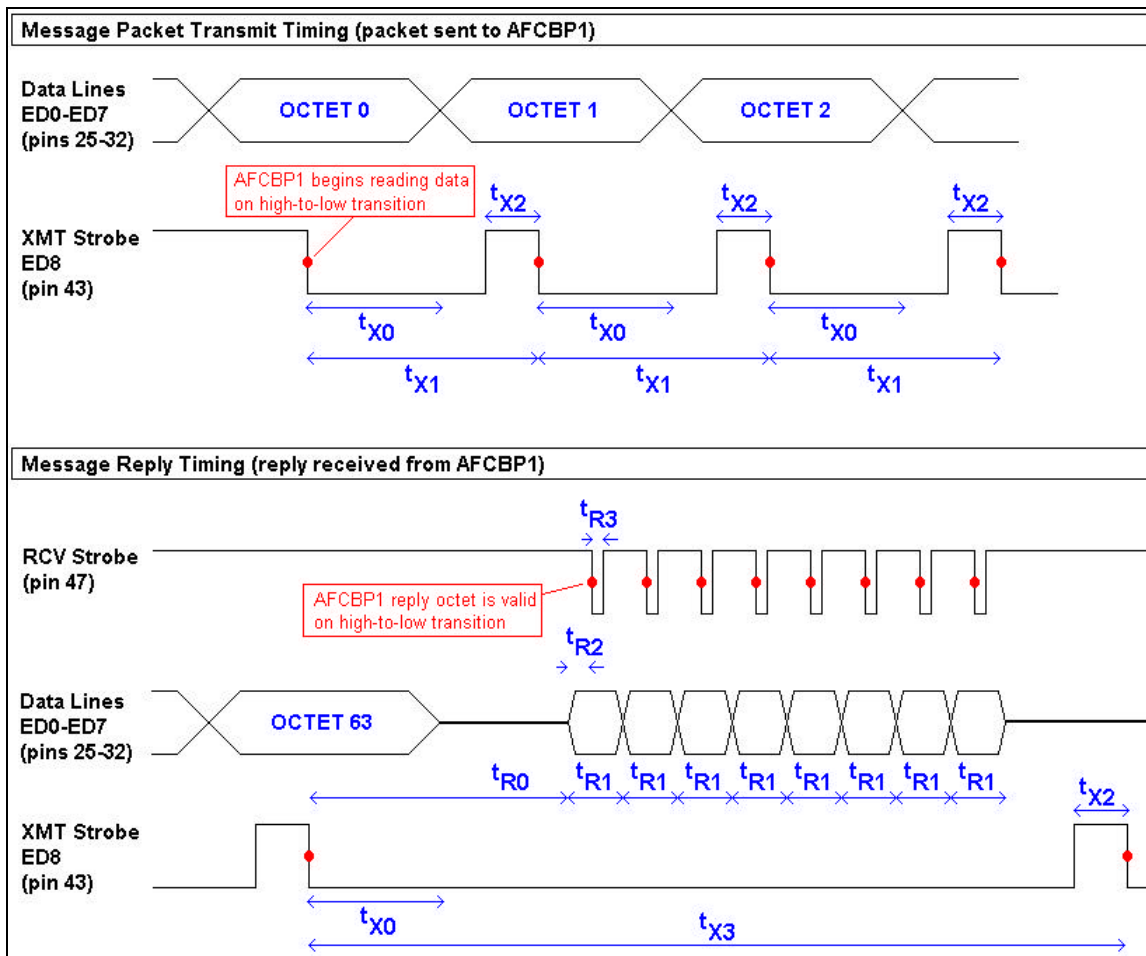


Figure 1: AFCBP1 Communications Timing Diagram

Table III: Critical Timing Parameters

Parameter	Symbol	Minimum	Typical	Maximum	Units
XMT OCTET DWELL	$t_{X0}$	150	150		$\mu$ sec
XMT OCTET INTERVAL	$t_{X1}$	155	180	6000	$\mu$ sec
XMT STROBE DURATION	$t_{X2}$	1	5		$\mu$ sec
XMT PACKET INTERVAL	$t_{X3}$	300	300		$\mu$ sec
RCV REPLY TIME	$t_{R0}$	18	18	20	msec
RCV OCTET DWELL	$t_{R1}$		20		msec
RCV OCTET LEAD TIME	$t_{R2}$		7		msec
RCV STROBE DURATION	$t_{R3}$		6.5		msec

**Table IV: Sample Message Packets**

Octet	Variable	Message #1	Message #2	Message #3
0	pcomm_Command	00	00	00
1		14	FF	14
2	pcomm_OpMode	00	00	00
3		00	00	00
4	pcomm_PosSweepRate	00	01	00
5		00	F4	00
6	pcomm_NegSweepRate	00	01	00
7		00	F4	00
8	pcomm_ManSweepDir	00	00	00
9		04	08	04
10	pcomm_NumLegs	00	00	00
11		00	07	00
12	pcomm_BeforeDelay	00	00	00
13		00	64	00
14	pcomm_AcqDelay	00	FF	00
15		01	FF	01
16	pcomm_K1OffsetState	00	00	00
17		02	02	02
18	pcomm_K2OffsetState	00	00	00
19		02	02	02
20	pcomm_K1Range	00	00	00
21		00	03	00
22	pcomm_K2Range	00	00	00
23		00	06	00
24	pcomm_K1SweepState	00	00	00
25		00	01	00
26	pcomm_K2SweepState	00	00	00
27		00	00	00
28	pcomm_GalPot	00	00	00
29		00	00	00
30	pcomm_OpenLoop	00	00	00
31		01	01	01
32	pcomm_DummyNormal	00	00	00
33		01	00	00
34	pcomm_SweepHold	00	00	00
35		01	00	01
36	pcomm_StopAtLower	00	00	00
37		01	01	01
38	pcomm_StopAtUpper	00	00	00
39		01	01	01
40	pcomm_AcqLength	00	FF	00
41		02	FF	02
42	pcomm_DisengageDelay	FF	FF	FF
43		FF	FF	FF
44	pcomm_SweepZero	00	00	00
45		01	01	01
46	pcomm_Range	00	00	00
47		00	00	00
48	pcomm_InitPot	00	00	00
49		00	00	00
50	pcomm_UpperLimit	00	00	00
51		01	00	01
52	pcomm_LowerLimit	FF	FD	FF
53		FF	44	FF
54	pcomm_FinalPot	00	FD	00
55		00	44	00
56	pcomm_NoConnect	00	00	00
57		01	00	00
58	pcomm_Unused1	00	00	00
59		00	00	00
60	pcomm_Unused2	00	00	00
61		00	00	00
62	pcomm_CheckSum	E0	1E	21
63		B4	CA	49

## **COMMUNICATIONS LIBRARY**

The CBPCOM communications library was developed by *Pine Instrument Company* to facilitate control of the AFCBP1 Bipotentiostat by programs written in the C programming language. The main function of this library is to provide a standard way to send message packets to the AFCBP1 over the digital I/O lines on the INTERFACE BOARD.

The library provides absolutely no data acquisition capability and no support for reading the analog input (ADC) channels. Ample support for data acquisition from the ADC channels is provided by the NI-DAQ library functions supplied by *National Instruments Corporation*. Similarly, no support is provided for analog output functions as the NIDAQ device driver library provides ample support for such functions.

The software interface to the CBPCOM library is a set of four public functions, a structure definition for the message packet, and a rather large set of constant definitions, all of which can be found in the "CBPCOMxx.H" header file. Many of the function and variable names begin with the "pcomm" prefix. This is a holdover from the earlier versions of the library which was itself named the "PCOMM" library.

### **THE FUNCTIONS**

The public functions exported by CBPCOM 3.2 are described below. Further details as to how each function works may be gleaned by reading through the source code for the CBPCOM library (included on the diskette).

#### ***PCOMM\_Initialize***

This function should be called once to initialize the library. It causes the library to build an internal list of all of the NIDAQ devices presently installed on the computer system.

This function has no parameters and no return value.

#### ***PCOMM\_SetDefaults***

This function can be used to set all of the PCOMM variables in a message packet to their default values. This provides a good starting point from which to build your own message packet. Use of this function is optional, and it is provided for convenience only.

The only parameter passed to this function is a pointer to the message packet's memory buffer. It is assumed that the memory for the buffer has already been allocated.

This function has no return value.

#### ***PCOMM\_ResetBoard***

This function resets the interface board and sends a "buffer flushing" pulse to the AFCBP1. This function requires one parameter.

The only parameter is a 16-bit integer which specifies the NI-DAQ device number for the INTERFACE BOARD. This device number is assigned to the board during the board installation process (consult the board manual for details).

This function returns an error code indicating the success or failure of the operation. All possible error codes are defined in the CBPCOMxx.H header file. Error codes with values less than -59 are NIDAQ device driver library error codes.

### ***PCOMM\_Hello***

This function should be called once to set up initial communication with the AFCBP1. This function initializes the interface board (by calling ***PCOMM\_ResetBoard***), sends a message packet to the AFCBP1, and waits for a reply. This function requires two parameters.

The first parameter is a 16-bit integer which specifies the NI-DAQ device number for the INTERFACE BOARD. This device number is assigned to the board during the board installation process (consult the board manual for details).

The second parameter is used to return the version number for the EPROM within the AFCBP1. This parameter is a 16-bit integer. The current EPROM version number is 2.02, so this parameter should be equal to 0202 (hexadecimal) upon return.

This function returns a boolean result which indicates whether or not the attempt to communicate with the AFCBP1 was successful.

### ***PCOMM\_SendMessage***

This function sends a message packet to the AFCBP1. This is the most important and most complicated function in the CBPCOM library. This function requires four parameters.

The first parameter is a 16-bit integer which specifies the NI-DAQ device number for the INTERFACE BOARD. This device number is assigned to the board during the board installation process (consult the board manual for details).

The second parameter is a pointer to the message packet that is to be sent to the AFCBP1.

The third parameter is an integer representing the number of attempts that should be made in trying to send the packet to the AFCBP1. A good value for this parameter is three (3).

The fourth parameter is used to return the version number for the EPROM within the AFCBP1. This parameter is a 16-bit integer. The current EPROM version number is 2.02, so this parameter should be equal to 0202 (hexadecimal) upon return.

This function returns an error code indicating the success or failure of the transmission operation. All possible error codes are defined in the CBPCOMxx.H header file. Error codes with values less than -59 are NIDAQ device driver library error codes.

### ***PCOMM\_SetCommuncationsTiming***

This function allows the caller to make changes to the fundamental communications timing parameters which govern message packet transmission. Normally, this function does not need to be used. This function requires three parameters.

The first parameter is a 16-bit integer which specifies the NI-DAQ device number for the INTERFACE BOARD. This device number is assigned to the board during the board installation process (consult the board manual for details).

The second parameter is a constant defining the communications timing parameter to be changed. These constants are defined in the CBPCOMxx.H header file.

The third parameter is the new value for the communications parameter.

This function has no return value.

### ***PCOMM\_GetCommunicationsTiming***

This function allows the caller to obtain the value of one of the fundamental communications timing parameters which govern message packet transmission. Normally, this function does not need to be used. This function requires two parameters.

The first parameter is a 16-bit integer which specifies the NI-DAQ device number for the INTERFACE BOARD. This device number is assigned to the board during the board installation process (consult the board manual for details).

The second parameter is a constant defining the communications timing parameter to be changed. These constants are defined in the CBPCOMxx.H header file.

This function returns an unsigned 32-bit integer that is the present value of the timing parameter.

### ***PCOMM\_ResetCommunicationsTiming***

This function allows the caller to reset all of the fundamental communications timing parameters which govern message packet transmission. Normally, this function does not need to be used. This function requires one parameter.

The only parameter is a 16-bit integer which specifies the NI-DAQ device number for the INTERFACE BOARD. This device number is assigned to the board during the board installation process (consult the board manual for details).

This function has no return value.

## **GENERAL VARIABLES**

The message packet can be viewed as a series of 64 octets, a set of 32 integers (16-bit), or as a set of 32 “pcomm” variables. The CBPCOM\_MSG\_PKT data structure is a union which permits the message packet to be treated in any of these three ways. In the description that follows, the message packet is described in terms of its 32 constituent “pcomm” variables. These 32 variables provide complete control of the AFCBP1 Bipotentiostat.

The many constants which define the allowed values for the various pcomm variables are listed in the CBPCOM31.H header file.

### ***pcomm\_Command***

This variable defines the action that you expect the AFCBP1 to take when it receives the message packet. Three commands are recognized: CBP\_START\_EXPERIMENT (10), CBP\_STOP\_EXPERIMENT (20), and CBP\_IDLE\_COMMAND (255). If the command code is invalid, then error code CBP\_COMMAND\_ERROR (-1) is returned.

If *pcomm\_Command* is CBP\_START\_EXPERIMENT (10), then the AFCBP1 BIPOTENTIostat will initiate the sweep once the message is processed. If the *pcomm\_Command* is CBP\_STOP\_EXPERIMENT (20), then the AFCBP1 BIPOTENTIostat will halt any sweep program that may already be in progress.

The CBP\_IDLE\_COMMAND (255) can be used when you want to send a message to the AFCBP1 without starting or stopping the sweep program. This does not necessarily mean that the sweep will be unaffected. If the message changes any of the sweep parameters, then the SWEEP GENERATOR will respond accordingly, even if *pcomm\_Command* is equal to CBP\_IDLE\_COMMAND.

The very first message sent to initialize the AFCBP1 should have the *pcomm\_Command* variable set equal to CBP\_STOP\_EXPERIMENT (20). This halts any sweep program that the SWEEP GENERATOR might presently be executing. As part of this same initial message to the AFCBP1, you should make certain that the variable *pcomm\_SweepHold* is set to HOLD (1) so that the sweep doesn't just start over again. You may also wish to set the variable *pcomm\_SweepZero* to TRUE (1) to assure that the sweep output is zero volts. If you do not intend to use the sweep at all, then every message you send to the AFCBP1 should always have *pcomm\_SweepHold* and *pcomm\_SweepZero* set in this manner.

To define and then execute a sweep program, you must send *two* messages to the AFCBP1. The first message should define the sweep program using the various global “pcomm” variables, and the *pcomm\_Command* variable should be set to the CBP\_IDLE\_COMMAND (255). This first message gives the AFCBP1 time to adjust the internal SWEEP GENERATOR settings appropriately. Then, send a second *identical* message, but change the value of the *pcomm\_Command* variable to CBP\_START\_EXPERIMENT (10). The sweep program will commence once the AFCBP1 has processed the second message.

#### ***pcomm\_GalPot***

This variable determines whether the K1 electrode circuit operates as a potentiostat or a galvanostat. The two valid settings for this variable are zero (0) for potentiostat mode or one (1) for galvanostat mode. Any other value results causes a CBP\_GALPOT\_ERROR (-15).

#### ***pcomm\_Open***

This variable determines whether the AFCBP1 operates in “open-loop” potential mode (see the AFCBP1 manual for more details). The two valid settings for this variable are CBP\_OPENLOOP\_CLOSED (1) for normal operation or CBP\_OPENLOOP\_OPEN (0) for “open-loop” mode. Any other value causes a CBP\_OPEN\_ERROR (-16).

#### ***pcomm\_DummyNormal***

This variable determines whether the AFCBP1 is connected to the external cell (normal) or an internal resistor network (dummy). The two valid settings for this variable are as follows:

CBP\_DUMMYNORMAL\_NORMAL (0) for normal operation, or  
CBP\_DUMMYNORMAL\_DUMMY (1) for dummy mode.

Any other causes a CBP\_DUMMYNORMAL\_ERROR (-17).

#### ***pcomm\_NoConnect***

This variable determines whether the AFCBP1 is connected to the internal dummy cell or the

actual external electrochemical cell *during the sweep program*. If the value of this variable is set equal to CBP\_NOCONNECT\_DUMMY\_CELL (1), then the AFCBP1 uses the dummy cell. If it is set equal to CBP\_NOCONNECT\_EXTERNAL\_CELL (0), then it uses the external cell. Any other value causes a CBP\_NOCONNECT\_ERROR (-28).

***pcomm\_K1SweepState, pcomm\_K2SweepState***

These variables determine whether or not the sweep output is applied to the K1 and/or K2 working electrode circuits. Setting either of these variables equal to one (1) causes the sweep to be applied to the appropriate electrode (K1 or K2). Setting both of them equal to one causes the sweep to be applied to both working electrodes.

If you will not be using the sweep, you should set both of these variables equal to zero. Note that when the sweep output is applied to an electrode circuit, it is summed with any offset potentials that are being applied at the same time.

If either variable is set to a value other than zero or one, a CBP\_SWEEPSTATE\_ERROR (-13) code is returned.

***pcomm\_K1OffsetState, pcomm\_K2OffsetState***

These variables determine whether or not the Offset Voltage controls physically located on the front panel of the AFCBP1 are active. Normally, you should deactivate these controls by setting these variables to CBP\_OFFSETSTATE\_DISABLED (2).

If for some reason, you want to make use of these controls, you can set this variable to either CBP\_OFFSETSTATE\_POSITIVE (1) or CBP\_OFFSETSTATE\_NEGATIVE (0) to apply either a positive or negative offset to the electrode potential.

Any other value for these variables causes a CBP\_OFFSETSTATE\_ERROR (-10).

***pcomm\_K1Range, pcomm\_K2Range***

These variables are used to set the CURRENT CONVERTER ranges for the K1 and K2 working electrode circuits on the AFCBP1 front panel. There are seven different ranges numbered zero through six (6) as summarized in Table IV.

**Table IV: Current Converter Ranges**

<b>range number</b>	<b>CURRENT CONVERTER</b>
0	100 mA / V
1	10 mA / V
2	1 mA / V
3	100 $\mu$ A / V
4	10 $\mu$ A / V
5	1 $\mu$ A / V
6	100 nA / V

## SWEEP GENERATOR VARIABLES

### ***pcomm\_PosSweepRate***

This variable is used to specify the sweep rate whenever the sweep is heading toward more positive (anodic) potentials. The value of this 16-bit integer must be non-zero, it must be a multiple of five (5), and it must be no higher than 9995. The sweep rate is specified in millivolts per second. If this variable is less than zero or greater than 10000, then a CBP\_SWEEP\_RATE\_ERROR (-3) is returned. The communications library does not check to make sure the value is a multiple of five.

### ***pcomm\_NegSweepRate***

This variable is used to specify the sweep rate whenever the sweep is heading toward more negative (cathodic) potentials. The value of this variable is subject to the same constraints as the *pcomm\_PosSweepRate* variable described above.

### ***pcomm\_ManSweepDir***

This variable is used to control the initial direction of the sweep. The two useful settings for this variable are CBP\_SWEEPDIR\_DOWN (8) for sweeping *down* towards more negative (cathodic) potentials or CBP\_SWEEPDIR\_UP (4) for sweeping *up* towards more positive (anodic) potentials. If the value of this variable is less than zero or greater than nine (9), it causes a CBP\_MANSWEEPDIR\_ERROR (-5).

### ***pcomm\_SweepHold***

This variable determines whether the sweep output is held constant or allowed to continue running. The two valid settings for this variable are zero (0) which allows the sweep to continue or one (1) which *holds* the sweep output constant. Any other value causes a CBP\_SWEEP\_HOLD\_ERROR (-18).

If you will not be using the sweep in your experiment, you should set this variable to one (1). If you wish to start the sweep, then you should set the variable to zero (0) and also set the *pcomm\_Command* to START\_EXPERIMENT.

### ***pcomm\_StopAtLower, pcomm\_StopAtUpper***

These variables determine whether or not the sweep stops when it reaches a limit potential. The two valid settings for this variable are zero (0) which allows the sweep to continue or one (1) which stops the sweep at the limit. Any other value causes a CBP\_LIMIT\_STOP\_ERROR (-19).

These variables are not very useful and should be set equal to one (1) for normal operation.

### ***pcomm\_SweepZero***

This variable is used to zero the sweep output. The two valid settings for this variable are zero (0) which allows the sweep to operate normally or one (1) which forces the sweep output to zero volts. Other values cause a CBP\_SWEEP\_ZERO\_ERROR (-23).

If you will not be using the sweep in your experiment, you should set this variable to one (1). If you will be using the sweep, then you can still set this variable to one (1) if you want to assure that the sweep starts from zero volts.

### ***pcomm\_InitPot***

This variable is used to set the initial potential for the sweep program. It is generally best to start the sweep program from zero volts by using a value of zero (0) for this variable. If you need the

sweep to start from some value other than zero, you can use an offset potential in conjunction with the sweep output to achieve the desired result.

#### ***pcomm\_UpperLimit, pcomm\_LowerLimit***

These variables are used to specify the upper and lower sweep limit potentials. The value of these 16-bit integers must be in the range from -9995 to 9995. The potentials are specified in millivolts. If one of the potentials is out of range, the a CBP\_SWEEPLIMIT\_ERROR (-26) is returned.

The value of *pcomm\_UpperLimit* must be at least 10 millivolts greater than the value of *pcomm\_LowerLimit* for proper sweep performance. If the value of *pcomm\_LowerLimit* is greater than *pcomm\_UpperLimit*, it causes a CBP\_SWEEPLIMIT\_INVERSION\_ERROR (-30) to be returned.

#### ***pcomm\_FinalPot***

This variable must be set equal to the potential at which the sweep is to stop. Normally, you will set the final potential equal to one of the two limit potentials (upper or lower, depending on the sweep direction). However, if you wish, you can have the sweep stop at some potential other than the usual limit potential.

The value of this 16-bit integer must be in the range from -9995 to 9995. The potential is specified in millivolts. If the potential is out of range, it causes a CBP\_FINALPOT\_ERROR (-27) to be returned.

In order to assure proper sweep operation, you must make sure that the final potential is valid. For example, if you know that the very last sweep segment in the sweep program starts at 800 mV and heads in a negative direction, then you should set the final potential to a value less than 800 mV.

#### ***pcomm\_NumLegs***

This variable specifies the total number of sweep segments in the sweep program. It should be set to zero if you will not be using the sweep. Otherwise, any value between 1 and 60,000 is acceptable. Values outside this range cause a CBP\_NUMLEGS\_ERROR (-6).

## **TIMING VARIABLES**

#### ***pcomm\_BeforeDelay***

This variable specifies a delay period before the sweep program begins. If this variable is set equal to zero, then the AFCBP1 starts the sweep as soon as possible after receiving the START\_EXPERIMENT command. Otherwise, it waits for a brief period of time before starting. The value of this variable determines the length of the delay period. The quantity of time is given in increments of 10 milliseconds. For example, if you set this variable equal to 200, then the delay period is two seconds long.

This variable must be set to a value in the range from zero to 60,000. Thus, the longest delay period is ten minutes. Values outside this range a CBP\_BEFOREDELAY\_ERROR (-7).

#### ***pcomm\_AcqDelay***

At some point during the sweep program, the AFCBP1 sends a triggering pulse to the INTERFACE BOARD. This pulse is sent on pin 38 and is meant to be used as a “start trigger” for the data acquisition process. The *NI-DAQ Function Reference Manual* describes how to set up the data

acquisition functions so that they wait for such an external trigger. Thus, you can use this start trigger to synchronize data acquisition with the sweep program.

If the *pcomm\_AcqDelay* variable is set equal to 65535 (which is 0xFFFF hexadecimal), then the trigger pulse is sent at exactly the same time that the sweep program starts.

If the *pcomm\_AcqDelay* variable is set to some value less than 65535, then the trigger pulse is sent at a given amount of time after receipt of the START\_EXPERIMENT command. This quantity of time is given in increments of 10 milliseconds. For example, if you set this variable equal to 200, then the time period is two seconds long.

If the value of *pcomm\_AcqDelay* is out of range, it causes a CBP\_ACQDELAY\_ERROR (-8).

### ***pcomm\_AcqLength***

At some point after the start trigger, the AFCBP1 sends a second triggering pulse to the INTERFACE BOARD. This pulse is sent on pin 39 and can be used as a stop trigger for halting data acquisition, if desired. The *NI-DAQ Function Reference Manual* describes how to set up the data acquisition functions so that they look for such an external stop trigger.

If the *pcomm\_AcqLength* variable is set equal to 65535 (which is 0xFFFF hexadecimal), then the trigger pulse is sent at exactly the same time as the sweep program stops.

If the *pcomm\_AcqLength* variable is set to some value less than 65535, then the *stop trigger* is sent at a given amount of time after *start trigger*. This quantity of time is given in increments of 10 milliseconds. For example, if you set this variable equal to 200, then the time period is two seconds long.

An out of range value causes a CBP\_ACQLENGTH\_ERROR (-21).

### ***pcomm\_DisengageDelay***

This variable determines whether or not the AFCBP1 automatically switches to the internal dummy cell after the stop trigger is sent. If you set the variable equal to 65535 (which is 0xFFFF hexadecimal), then the cell is not automatically switched to the dummy cell.

If this variable is set to some value less than 65535, then AFCBP1 switches to the internal dummy cell a certain amount of time after the *stop trigger*. This quantity of time is given in increments of 10 milliseconds. For example, if you set this variable equal to 200, then the time period is two seconds long.

An out of range value causes a CBP\_DISENGAGEDDELAY\_ERROR (-22).

## **RETURN CODES**

If an error occurs during a call to PCOMM\_Hello or PCOMM\_SendMessage, then an error code is returned to the caller. The most commonly encountered error codes are described below:

### ***CBP\_DEVICE\_NUMBER\_ERROR (-29)***

This code indicates that the NIDAQ device number passed to the function was invalid.

### ***CBP\_DEVICE\_UNSUPPORTED (-35)***

This code indicates that the requested NIDAQ device is an interface board that is not supported by this version of the CBPCOM library.

### ***CBP\_XMT\_CHECKSUM\_ERROR (-40)***

This code indicates that the AFCBP1 received a corrupted message packet.

### ***CBP\_RCV\_PKT\_SHORT (-41)***

This code indicates that the reply from the AFCBP1 did not contain enough octets.

### ***CBP\_RCV\_PKT\_LONG (-42)***

This code indicates that the reply from the AFCBP1 contained too many octets.

### ***CBP\_RCV\_PKT\_TIMEOUT (-43)***

This code indicates that no reply packet was received from the AFCBP1. The most common causes of this error are a disconnected interface cable or when the AFCBP1 is not in EXTERNAL control mode.

## ***SOURCE CODE DISKETTE***

The source code for several versions of the CBPCOM library are included on the source code diskette. It is recommended that you use the most recent version; the older versions are provided for backward compatibility only. This library is written using the C programming language. It is the very same library used by *Pine Instrument Company* to develop software for the AFCBP1.

The library consists of five different files. Four of these files are header files (CBPCOMxx.H, CRCTABLE.H, NIBOARDS.H, and SOMETYPE.H), and the fifth file (CBPCOMxx.C) contains the actual lines of code. These files are shipped to you on diskette along with this manual.

The CBPCOMxx.H header file contains important definitions and structure declarations that describe the message packet. Error code values are also defined in this file.

The CRCTABLE.H header file contains a large array of integer values used to generate CRC checksums.

The NIBOARDS.H header file contains the NIDAQ code numbers for all of the interface boards manufactured by *National Instruments Corporation*.

The SOMETYPE.H header file contains abbreviations for common C data types.

### ***Using Other Languages***

*Pine Instrument Company* does not directly support the use of languages other than the C language. However, the information in these source code listings should be sufficient for the average programmer to develop their own library using another language.