

HSI-24 OPERATOR'S MANUAL

**HSI-24 Communications device driver
32 bit Dynamic Link Library
for use with Windows 95/98
VERSION 1.0**

Probe Products Corporation
1763 Baseline Road
Grand Island, NY 14072
(716) 773-5554
(716)-773-5336 FAX
www.probeproducts.com

Table of Contents

The HSI-24, an Overview	1
System Components	1
Large Systems	1
Auxiliary Inputs	1
High Speed Operation	2
Quick Start	3
Files on Diskette	4
File Required to run HSI-24	4
File Required to Interface to HSI-24	4
hsicom32.dll Software Interface	5
Overview	5
Communication Functions	8
Overview	8
Status Code	8
General Arguments	8
General Return Values	8
Floating Point Arguments	9
Floating Point Return Values	9
Funtion HsiCommInit	10
Funtion HsiMsgBox	11
Funtion HsiFirmware	12
Funtion HsiDLL	13
Function HsiStartScan	14
Function HsiStopScan	15
Function HsiDefineChannel	16
Function HsiReadFormula	17

Function HsiClearAllChannels	18
Function HsiClearChannel	19
Function HsiReadChannel	20
Function HsiSetCzero	21
Function HsiReadCzero	22
Function HsiResetMM	23
Function HsiReadLvdT	24
Function HsiSetTfsv	25
Function HsiReadTfsv	26
Function HsiSetTzero	27
Function HsiReadTzero	28
Function HsiReadAnalog	29
Function HsiSetAfsv	30
Function HsiReadAfsv	31
Function HsiSetAzero	32
Function HsiReadAzero	33
Function HsiSetScanTime	34
Function HsiGetScanTime	35
Function HsiGetScanFlag	36
Function HsiReadDirect	37
Appendix A: HSI-24 Formula Syntax	38
Peak Hold Functions	38
Mathematical Functions	38
Miscellaneous Functions	38
Mathematical Operators	39
Constants	39
Input Terms	39

Channel Terms	39
Example Formulae	39
Appendix B: Startup Settings	40
Appendix C: Error Codes	41
Command Error	41
Channel Definition Errors	41

The HSI-24, an Overview

Probe Product Corporation's HSI-24 is a high speed signal conditioning/data processing system designed to allow direct connection of gaging and displacement transducers to an IBM or IBM compatible computer.

To acquaint a new user with the powerful features of the HSI-24, an application program is furnished with the system. This program can be used "as-is" to set up a measurement system, or it can be used as a basis for customized software. Source code for this application program is included as part of the system. This application program called 'HSIDEMO' is available in a DOS version that may also be used through Windows.

System Components

The system consists of four parts:

- 1- a master system board with onboard 16-bit processor, A/D converter and all other active analog and digital devices, residing within the computer and occupying one slot,
- 2- a cable assembly to connect the system board to a passive external junction box with transducer sockets,
- 3- a passive junction box (normally located near the transducers) with sockets suitable for the transducers in use, and
- 4- a diskette containing the internal operating software for the system, 16/32 bit DLL Software and sample application and setup programs.

Some systems may contain special cable assemblies for specific applications, or no cable and junction box at all, when those connections are to be made on-site.

Large Systems

For those applications requiring conditioning for more than 24 transducers, slave system boards, cable assemblies and boxes are added to the master system. Up to 96 transducers can be conditioned by one such master/slave system.

Auxiliary Inputs

Besides the transducer inputs, each system board (master or slave) has four +/-5VDC inputs which may be used for any analog signal which has (or can be modified to have) suitable values. Digital inputs from TTL devices, Hall-effect proximity switches and switch closures are typical inputs which can also be used.

A fully expanded HSI-24 system will, therefore, have the capacity for 96 transducers and 16 voltage sources.

High Speed Operation

It is important to note that all signal conditioning for the transducers is accomplished within the system -- no external gaging amplifiers, A/D converters or communication ports are required.

Transducer signals, or even complex measurement functions (using the unique formulae construction techniques) can be passed to the host at up to **2,500** readings per second.

The HSI-24 lends itself to dynamic gaging, since complex functions can be completed by the on-board co-processor, with results being passed to the host computer. And, since all communications to the host computer are passed directly to its bus, data transfer rates are many times faster than those possible using RS-232, RS-422 or IEEE- 488 interfaces. These two factors combine to allow very high speed gaging operations.

Quick Start

For those users who are sure they are familiar with the installation of adapter cards in PC type equipment, this section provides a brief setup procedure.

1. Remove power and cover from PC.
2. Install HSI-24 card. Be sure to install the rear bracket hold-down screw, otherwise the force from flexing the large interface cable will pop the HSI-24 card out of the bus connector.
3. Install the large interface cable between the HSI-24 rear connector and the transducer junction box. Unless you have special cables the ends of the large cable are interchangeable.
4. Replace the computer covers.
5. Start the computer.
6. Make a subdirectory for the HSI-24 software.
7. Copy all the files from the supplied diskette to the HSI-24 subdirectory.
8. Run the batch file called **RUN**.

Files on Diskette

File Required to run HSI-24

PCA10C.BIN firmware which is loaded into coprocessor

File Required to Interface to HSI-24

HSICOM32.DLL program which interfaces commands to hardware.

hsicom32.dll Software Interface

Overview

The HSI Communications device driver (hsicom32.dll) is written in "C" and compiled as a 32 bit Dynamic Link Library (dll) file for use with Windows 95/98. The device drivers for Windows NT, Windows 3.1 and DOS are different so the target platform should be known or defined before using the driver.

To use the HSI Communication functions in your software, include the supplied hsi-com32.h file in the C source files. Link the file hsi-com32.lib with your application. The application must be able to locate the hsi-com32.dll file at run time. There are various strategies to accomplishing this. Check the documentation with the Windows development package for details about locating dll files. Put the dll file (hsicom32.dll) in the same path as your application or in the windows/system path.

All of the functions in the hsi-com32.dll have the same interface as their counterparts in the non-Windows (DOS) hsi-comm.c file. See the documentation on using the non-Windows hsi-comm.c driver.

The functions that differ from the non-Windows (hsi-comm.c) functions are:

```
int HsiCommInit(unsigned IOBaseAddr,const char *FirmwareFile);
```

This function **must** be the first function called before any of the other functions in HSICOM32.DLL are used.

The HsiCommInit function will set the IO address as given and attempt access to the HSI-24 board. If the access fails the HsiCommInit function will attempt to locate the given firmware file and load it into the HSI-24 board.

```
void HsiMsgBox(int Enable);
```

The HsiMsgBox function enables/disables a dialog box which appears whenever there is an error detected in an hsi-comm command. The dialog box translates the interface error code into a brief descriptive error message.

The enabling/disabling of the error dialog box does not affect the error codes returned by the hsi-comm functions.

```
int HsiFirmware(char *FirmwareVersion);
```

The HsiFirmware() function will copy the version stamp read from the firmware file to the area pointed to by the parameter.

```
int HsiDLL(char *DLLVersion);
```

The HsiDLL function will copy the version stamp from the DLL file to the area pointed to by the parameter.

Typical use:

STEP 1 - Initialize

```
.
int HsiLoadStatus=0;
.
// load firmware
HsiLoadStatus = HsiCommInit(0,"pca10c.bin");
if(HsiLoadStatus==0){
    HsiSetScanTime(100);           // scan time set to 10.0 ms
    // scan time must be set or it will default
    // to zero (meaning no scanning)
    HsiStartScan();
}
.
.
TimerID = SetTimer(NULL,0,400,NULL);
```

STEP 2 - define channels

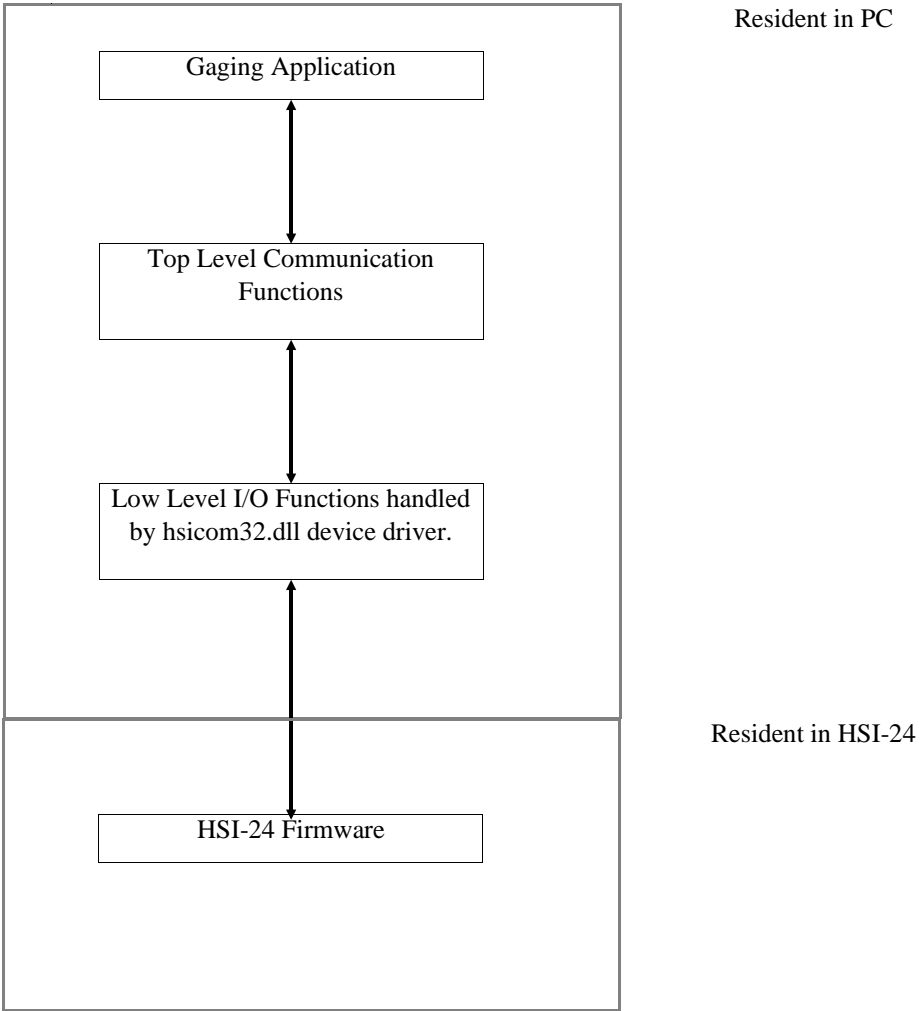
```
HsiDefineChannel(1,"t1+t2");
HsiDefineChannel(2,"max(t1+t2)");
HsiDefineChannel(3,"min(t1+t2)");
```

STEP 3 - read values

```
.
.
float values[3];
.
.
HsiReadChannel(1,&values[0]);      // pass adr of space for float
HsiReadChannel(2,&values[1]);      // pass adr of space for float
HsiReadChannel(3,&values[2]);      // pass adr of space for float
.
.
```

```
// display or store values
```

```
.
.
// to reset max/min/tir functions used in formulae call HsiResetMM()
if(ResetPushedMessage)
    HsiResetMM();
```



Communication Functions

Overview

There is one function for each of the HSI-24 commands. To use the functions, call the function with its arguments. The function will issue the proper command code and arguments to the HSI-24. The function will return the HSI-24's response.

Status Code

All HSI-24 functions return a status code which indicates the success or failure of the command execution. The status code is returned as the value of the function. The functions that return values in addition to the status code require arguments which are a pointer to a place to put the returned value.

The following rules pertain to all top level functions:

All functions return an integer which will be:

zero If the command was completed successfully.

-1 If the low level routines were unable to communicate.

+value If the command was not executable. (see Error Codes in Appendix C)

General Arguments

When functions require a channel number as an argument the channel number must be between 1 and 96.

When functions require a transducer number as an argument the transducer number must be between 1 and 96. (Transducer numbers greater than 24 will be accepted but are only meaningful if slave HSI-24s are installed.)

When functions require an analog number as an argument the analog number must be between 1 and 16. (Analog numbers greater than 4 will be accepted but are only meaningful if slave HSI-24s are installed.)

General Return Values

All functions return the status as the function return value. When other values must be returned the calling routine must supply a pointer to a variable which will have the return value stored into it.

Floating Point Arguments

When a function requires a floating point value, the value supplied is dependent on the condition of the floating point input mode. If the floating point input mode is set to the value `FP_IMODE_NATIVE` (see file `PCACMD.H`), then the number must be a single precision number in 8087 format. If the floating point input mode is set to the value `FP_IMODE_ASCII` then the number must be an ASCII string which represents the value. In the ASCII mode the conversion will terminate when an invalid character is detected. The string `'ABC'` will produce a value of `'0.0'`. Scientific notation is not allowed.

Floating Point Return Values

When a function returns a floating point value, the value returned is dependent on the condition of the floating point output mode. If the floating point output mode is set to the value `FP_OMODE_NATIVE`, then the number will be a single precision number in 8087 format. If the `FLOATING POINT OUTPUT MODE` is set to the value `FP_OMODE_ASCII`, then the number will be an ASCII string which represents the value. The number of decimal places in ASCII output mode is controlled by an HSI-24 command.

Important Note. The functions designed for IEEE floats move the floating point number to the variable pointed to by the argument. The functions designed for ASCII floats only set the passed pointer to point to the result string. The result string still located in the receive buffer. If you want to save the result string you must copy it out of the buffer before another HSI-24 function is called.

Function HsiCommInit

For 'C' applications use:

```
int HsiCommInit(HSIBasePort,HSIBinFile)
unsigned HSIBasePort; //Base IO address of HSI-24 system.
char *HSIBinFile; //char *HSIBinFile="pca10c.bin";
```

For 'Basic' applications use:

```
Declare Function HsiCommInit Lib "hsicom32" ( _
    ByVal HSIBasePort As Integer, _
    ByVal HSIBinFile As String _
) As Integer
```

PARAMETERS:

unsigned IOBaseAddr - Base IO address of HSI-24 system. Use the value zero here. Using a value of zero will cause the dll to use the default HSI-24 address of 0x310. See notes below if an I/O bus conflict exists with the address 0x310.

char *FirmwareFile - Pointer to the name (including path if not in the application's directory) of the firmware file. The firmware file is the file with a name like "PCA10C.BIN".

RETURNS:

Return value is zero if firmware loaded OK.

zero If the command was completed successfully.
-1 If the low level routines were unable to communicate.
+value If the command was not executable. (see Error Codes in Appendix C)

COMMENTS:

This function **must** be the first function called before any of the other functions in HSICOM32.DLL are used.

The HsiCommInit function will set the IO address as given and attempt access to the HSI-24 board. If the access fails the HsiCommInit function will attempt to locate the given firmware file and load it into the HSI-24 board.

I/O Address Conflicts:

The HSI-24 uses I/O addresses 0x0310 through 0x0313 inclusive. It decodes all 16 bits of the I/O address so it will not reappear at intervals throughout the I/O address space. Other values can be used if a conflict exists on the ISA bus where the HSI-24 is installed. Using other address values to resolve bus conflicts requires that the addressing jumpers on the HSI-24 board be changed.

Function HsiMsgBox

For 'C' applications use:

```
void HsiMsgBox(int Enable);  
int Enable;           // Value of zero (FALSE) disables the error dialog box.
```

For 'Basic' applications use:

```
Declare Function HsiMsgBox Lib "hsicom32" (_  
    ByVal Enable As Integer, _  
    ) As Integer
```

PARAMETERS:

int Enable - Value of one (TRUE) enables the error dialog box.

RETURNS:

Return status.

zero If the command was completed successfully.

-1 If the low level routines were unable to communicate.

+value If the command was not executable. (see Error Codes in Appendix C)

COMMENTS:

The HsiMsgBox function enables/disables a dialog box which appears whenever there is an error detected in an hsiComm command. The dialog box translates the interface error code into a brief descriptive error message.

The enabling/disabling of the error dialog box does not affect the error codes returned by the hsiComm functions.

Function HsiFirmware

For 'C' applications use:

```
int HsiFirmware(VersionString);  
char *VersionString; //Version number of firmware loaded into the HSI-24.
```

For 'Basic' applications use:

```
Declare Function HsiFirmware Lib "hsicom32" ( _  
    ByVal VersionString As String, _  
    ) As Integer
```

PARAMETERS:

char *VersionString - Version number of firmware loaded into the HSI-24. A pointer to char array 40 bytes long.

RETURNS:

Return status.

zero If the command was completed successfully.

-1 If the low level routines were unable to communicate.

+value If the command was not executable. (see Error Codes in Appendix C)

COMMENTS:

The HsiFirmware function returns the version number of the current firmware that is loaded into the HSI-24. The version number returned is an array 40 bytes long.

Function HsiDLL

For 'C' applications use:

```
int HsiDLL(VersionString);
```

```
char *VersionString; //Version number of current Dynamic Link Library.
```

For 'Basic' applications use:

```
Declare Function HsiDLL Lib "hsicom32" ( _
```

```
    ByVal VersionString As String, _
```

```
) As Integer
```

PARAMETERS:

char *VersionString - Version number of current Dynamic Link Library. A pointer to char array 20 bytes long.

RETURNS:

Return status.

zero If the command was completed successfully.

-1 If the low level routines were unable to communicate.

+value If the command was not executable. (see Error Codes in Appendix C)

COMMENTS:

The HsiDLL function returns the version number of the current Dynamic Link Library.

Function HsiStartScan

For 'C' applications use:
int HsiStartScan()

For 'Basic' applications use:
Declare Function HsiStartScan Lib "hsicom32" (_
) As Integer

PARAMETERS:

RETURNS:

Return status.

zero If the command was completed successfully.
-1 If the low level routines were unable to communicate.
+value If the command was not executable. (see Error Codes in Appendix C)

COMMENTS:

This function enables the scanning of formulae which contain MAX, MIN or TIR functions.

Function HsiStopScan

For 'C' applications use:
int HsiStopScan()

For 'Basic' applications use:
Declare Function HsiStopScan Lib "hsicom32" (_
) As Integer

PARAMETERS:

RETURNS:
Return status.

COMMENTS:
This function disables the scanning of formulae which contain MAX, MIN or TIR functions. The functions will be unaffected by changes in their arguments.

Function HsiDefineChannel

For 'C' applications use:

```
int HsiDefineChannel(int channel,char *formula)
int channel;          //A channel number.
char *formula;       //Pointer to formula string.
```

For 'Basic' applications use:

```
Declare Function HsiDefineChannel Lib "hsicom32" ( _
    ByVal channel As Integer, _
    ByVal Formula As String _
) As Integer
```

PARAMETERS:

int channel - A channel number must be between 1 and 96, inclusive.

char *formula - The formula argument is a pointer to a string which contains a valid formula.

RETURNS:

Return status.

COMMENTS:

This function will define a formula for a channel. The formula argument is a pointer to a string which contains a valid formula. The formulae are supplied to the HSI-24 in the form of an ASCII string terminated by a null byte. See the description of formulae in the section **HSI-24 FORMULA SYNTAX** for details of allowed formats.

Function HsiReadFormula

For 'C' applications use:

```
int HsiReadFormula(int channel,char *formula)
int channel;           //A channel number.
char **formula;       //Points *formula to formula string in the rx_buffer.
```

For 'Basic' applications use:

```
Declare Function HsiReadFormula Lib "hsicom32" ( _
    ByVal channel As Integer, _
    ByRef Formula As String _
) As Integer
```

PARAMETERS:

int channel - A channel number must be between 1 and 96, inclusive.

char **formula - The formula argument is a pointer to a string which contains a valid formula.

RETURNS:

Return status.

COMMENTS:

This function will set the pointer (char **formula) to point to the ASCIIZ string (an ASCII string terminated by a null byte) which is the formula for the specified channel.

Note: This function does not copy the string to a destination string. The formula string is located in the receive buffer and formula points to the receive buffer location.

The formula argument is a pointer to a string which contains a valid formula. See the description of formulae in the section **HSI-24 FORMULA SYNTAX** for details of allowed formats.

Function HsiClearAllChannels

For 'C' applications use:
int HsiClearAllChannels()

For 'Basic' applications use:
Declare Function HsiClearAllChannels Lib "hsicom32" (_
) As Integer

PARAMETERS:

RETURNS:
Return status.

COMMENTS:
This function will erase the formulae for all channels.

Function HsiClearChannel

For 'C' applications use:

```
int HsiClearChannel(int channel)
int channel;           //A channel number.
```

For 'Basic' applications use:

```
Declare Function HsiClearChannel Lib "hsicom32" ( _
    ByVal channel As Integer _
) As Integer
```

PARAMETERS:

int channel -The channel number must be between 1 and 96, inclusive.

RETURNS:

Return status.

COMMENTS:

This function will erase the formula for one channel.

Function HsiReadChannel

For 'C' applications use:

```
int HsiReadChannel(int channel, float *cvalue)
int channel;           //A channel number.
float *cvalue;        //Pointer to the channel value.
```

For 'Basic' applications use:

```
Declare Function HsiReadChannel Lib "hsicom32" ( _
  ByVal channel As Integer, _
  ByRef Value As Single _
) As Integer
```

PARAMETERS:

int channel - The channel number must be between 1 and 96, inclusive.
float *cvalue - Pointer to the returned value read.

RETURNS:

Return status.

COMMENTS:

Reads the value of a channel and places the result in the location pointed to by cvalue.

All functions return the status as the function return value. When other values must be returned the calling routine must supply a pointer to a variable which will have the return value stored into it.

Function HsiSetCzero

For 'C' applications use:

```
int HsiSetCzero(int channel, float *czero)
int channel;           //A channel number.
float *czero;         //Pointer to the channel's new zero offset value.
```

For 'Basic' applications use:

```
Declare Function HsiSetCzero Lib "hsicom32" ( _
  ByVal channel As Integer, _
  ByRef ChanZero As Single _
) As Integer
```

PARAMETERS:

int channel - The channel number must be between 1 and 96, inclusive.

float *czero - Pointer to the channel's new zero offset value to be set by function call.

RETURNS:

Return status.

COMMENTS:

This function sets the zero offset for a channel.

The channel's zero offset value is added to the channel value when the HsiReadChannel function is used and when the channel is used in a formula for another channel.

All functions return the status as the function return value. When other values must be returned the calling routine must supply a pointer to a variable which will have the return value stored into it.

Function HsiReadCzero

For 'C' applications use:

```
int HsiReadCzero(int channel, float *czero)
int channel;           //A channel number.
float *czero;         //Pointer to the channel's current zero offset value.
```

For 'Basic' applications use:

```
Declare Function HsiReadCzero Lib "hsicom32" ( _
    ByVal channel As Integer, _
    ByRef ChanZero As Single _
) As Integer
```

PARAMETERS:

int channel - The channel number must be between 1 and 96, inclusive.

float *czero - Pointer to the channel's current zero offset value returned by function call.

RETURNS:

Return status.

COMMENTS:

This function reads the current zero offset for a channel.

The channel's zero offset value is added to the channel value when the HsiReadChannel function is used and when the channel is used in a formula for another channel.

All functions return the status as the function return value. When other values must be returned the calling routine must supply a pointer to a variable which will have the return value stored into it.

Function HsiResetMM

For 'C' applications use:
int HsiResetMM()

For 'Basic' applications use:
Declare Function HsiResetMM Lib "hsicom32" (_
) As Integer

PARAMETERS:

RETURNS:
Return status.

COMMENTS:
This function will reset all MIN, MAX and TIR functions.

Function HsiReadLvdt

For 'C' applications use:

```
int HsiReadLvdt(int lvdt, float *tvalue)
int lvdt;           //An lvdt number.
float *tvalue;     //Pointer to the transducer reading.
```

For 'Basic' applications use:

```
Declare Function HsiReadLvdt Lib "hsicom32" ( _
  ByVal lvdt As Integer, _
  ByRef tvalue As Single _
) As Integer
```

PARAMETERS:

int lvdt - The transducer number must be between 1 and 96, inclusive.
float *tvalue - Pointer to the transducer reading returned by function call.

RETURNS:

Return status.

COMMENTS:

This function reads the current value of a transducer.

All functions return the status as the function return value. When other values must be returned the calling routine must supply a pointer to a variable which will have the return value stored into it.

Function HsiSetTfsv

For 'C' applications use:

```
int HsiSetTfsv(int lvdt, float *tfsv)
int lvdt;           //An lvdt number.
float *tfsv;       //Pointer to thefull scale value.
```

For 'Basic' applications use:

```
Declare Function HsiSetTfsv Lib "hsicom32" ( _
    ByVal lvdt As Integer, _
    ByRef tfsv As Single _
) As Integer
```

PARAMETERS:

int lvdt - The transducer number must be between 1 and 96, inclusive.
float *tfsv - Pointer to the transducer reading returned by function call.

RETURNS:

Return status.

COMMENTS:

This function sets the full scale value of a transducer.

All functions return the status as the function return value. When other values must be returned the calling routine must supply a pointer to a variable which will have the return value stored into it.

Function HsiReadTfsv

For 'C' applications use:

```
int HsiReadTfsv(int lvdt, float *tfsv)
int lvdt;           //An lvdt number.
float *tfsv;       //Pointer to thefull scale value.
```

For 'Basic' applications use:

```
Declare Function HsiReadTfsv Lib "hsicom32" ( _
    ByVal lvdt As Integer, _
    ByRef tfsv As Single _
) As Integer
```

PARAMETERS:

int lvdt - The transducer number must be between 1 and 96, inclusive.
float *tfsv - Pointer to the full scale value returned by function call.

RETURNS:

Return status.

COMMENTS:

This function reads the current full scale value of a transducer.

All functions return the status as the function return value. When other values must be returned the calling routine must supply a pointer to a variable which will have the return value stored into it.

Function HsiSetTzero

For 'C' applications use:

```
int HsiSetTzero(int lvdt, float *tzero)
int lvdt;           //An lvdt number.
float *tzero;      //Pointer to the transducer zero value.
```

For 'Basic' applications use:

```
Declare Function HsiSetTzero Lib "hsicom32" ( _
    ByVal lvdt As Integer, _
    ByRef tzero As Single _
) As Integer
```

PARAMETERS:

int lvdt - The transducer number must be between 1 and 96, inclusive.
float *tzero - Pointer to the zero value.

RETURNS:

Return status.

COMMENTS:

This function sets the zero offset for a transducer. The zero offset is added to the transducer value when the read transducer function is used and also when the transducer is used in a formula.

All functions return the status as the function return value. When other values must be returned the calling routine must supply a pointer to a variable which will have the return value stored into it.

Function HsiReadTzero

For 'C' applications use:

```
int HsiReadTzero(int lvdt, float *tzero)
int lvdt;           //An lvdt number.
float *tzero;      //Pointer to the zero value.
```

For 'Basic' applications use:

```
Declare Function HsiReadTzero Lib "hsicom32" ( _
    ByVal lvdt As Integer, _
    ByRef tzero As Single _
) As Integer
```

PARAMETERS:

int lvdt - The transducer number must be between 1 and 96, inclusive.
float *tzero - Pointer to the zero value returned by function call.

RETURNS:

Return status.

COMMENTS:

This function reads the current zero offset for a transducer.

All functions return the status as the function return value. When other values must be returned the calling routine must supply a pointer to a variable which will have the return value stored into it.

Function HsiReadAnalog

For 'C' applications use:

```
int HsiReadAnalog(int analog, float *avalue)
int analog;           //An analog number.
float *avalue;       //Pointer to the analog value.
```

For 'Basic' applications use:

```
Declare Function HsiReadAnalog Lib "hsicom32" ( _
  ByVal analog As Integer, _
  ByRef avalue As Single _
) As Integer
```

PARAMETERS:

int analog - The transducer number must be between 1 and 96, inclusive.
float *avalue - Pointer to the analog value returned by function call.

RETURNS:

Return status.

COMMENTS:

This function reads the current value of an analog input.

All functions return the status as the function return value. When other values must be returned the calling routine must supply a pointer to a variable which will have the return value stored into it.

Function HsiSetAfsv

For 'C' applications use:

```
int HsiSetAfsv(int analog, float *afsv)
int analog;           //An analog number.
float *afsv;         //Pointer to theanalog full scale value.
```

For 'Basic' applications use:

```
Declare Function HsiSetAfsv Lib "hsicom32" ( _
    ByVal analog As Integer, _
    ByRef afsv As Single _
) As Integer
```

PARAMETERS:

int analog - The analog number must be between 1 and 16, inclusive.
float *afsv - Pointer to the analog full scale value reading.

RETURNS:

Return status.

COMMENTS:

This function sets the full scale value of an analog input.

All functions return the status as the function return value. When other values must be returned the calling routine must supply a pointer to a variable which will have the return value stored into it.

Function HsiReadAfsv

For 'C' applications use:

```
int HsiReadAfsv(int analog, float *afsv)
int analog;           //An analog number.
float *afsv;          //Pointer to theanalog full scale value.
```

For 'Basic' applications use:

```
Declare Function HsiReadAfsv Lib "hsicom32" ( _
    ByVal analog As Integer, _
    ByRef afsv As Single _
) As Integer
```

PARAMETERS:

int analog - The analog number must be between 1 and 16, inclusive.
float *afsv - Pointer to the analog full scale value reading.

RETURNS:

Return status.

COMMENTS:

This function reads the full scale value of an analog input.

All functions return the status as the function return value. When other values must be returned the calling routine must supply a pointer to a variable which will have the return value stored into it.

Function HsiSetAzero

For 'C' applications use:

```
int HsiSetAzero(int analog, float *azero)
int analog;           //An analog number.
float *azero;         //Pointer to theanalog zero offset value.
```

For 'Basic' applications use:

```
Declare Function HsiSetAzero Lib "hsicom32" ( _
    ByVal analog As Integer, _
    ByRef azero As Single _
) As Integer
```

PARAMETERS:

int analog - The analog number must be between 1 and 16, inclusive.
float *azero - Pointer to the analog zero offset reading.

RETURNS:

Return status.

COMMENTS:

This function sets the zero offset for an analog input. The zero offset is added to the analog input value when the read analog function is used or when the analog input is used in a formula.

All functions return the status as the function return value. When other values must be returned the calling routine must supply a pointer to a variable which will have the return value stored into it.

Function HsiReadAzero

For 'C' applications use:

```
int HsiReadAzero(int analog, float *azero)
int analog;           //An analog number.
float *azero;        //Pointer to theanalog zero offset value.
```

For 'Basic' applications use:

```
Declare Function HsiReadAzero Lib "hsicom32" ( _
    ByVal analog As Integer, _
    ByRef azero As Single _
) As Integer
```

PARAMETERS:

int analog - The analog number must be between 1 and 16, inclusive.
float *azero - Pointer to the analog zero offset reading.

RETURNS:

Return status.

COMMENTS:

This function reads the zero offset for an analog input.

All functions return the status as the function return value. When other values must be returned the calling routine must supply a pointer to a variable which will have the return value stored into it.

Function HsiSetScanTime

For 'C' applications use:

```
int HsiSetScanTime(int time)
int time;           //time' is in units of tenth's of milliseconds.
```

For 'Basic' applications use:

```
Declare Function HsiSetScanTime Lib "hsicom32" ( _
    ByVal ScanTime As Integer _
) As Integer
```

PARAMETERS:

int time - The argument 'time' is in units of tenth's of milliseconds.

RETURNS:

Return status.

COMMENTS:

Sets scanning period for MAX, MIN and TIR functions.

Function HsiGetScanTime

For 'C' applications use:

```
int HsiGetScanTime(int *time)
int time;           // 'time' is in units of tenth's of milliseconds.
```

For 'Basic' applications use:

```
Declare Function HsiGetScanTime Lib "hsicom32" ( _
    ByVal ScanTime As Integer _
) As Integer
```

PARAMETERS:

int *time - Pointer to current scan period.

RETURNS:

Return status.

DESCRIPTION:

A pointer is returned in the time parameter to the current scan period.

COMMENTS:

The argument 'time' is in units of tenth's of milliseconds.

Function HsiGetScanFlag

For 'C' applications use:

```
int HsiGetScanFlag(int *flag)
int flag; //Read the current scan flag setting.
```

For 'Basic' applications use:

```
Declare Function HsiGetScanFlag Lib "hsicom32" ( _
    ByVal ScanFlag As Integer _
) As Integer
```

PARAMETERS:

int *flag - Pointer to current scan flag setting.

RETURNS:

Return status.

DESCRIPTION:

A pointer is returned in the flag parameter to the current scan flag.

COMMENTS:

Function HsiReadDirect

For 'C' applications use:

```
int HsiReadDirect(int lvd_t_count,unsigned char *lvd_t_list,int **values)
int lvd_t_count;           //number of lvd_t's to be read.
unsigned char *lvd_t_list; //pointer to a list of lvd_t numbers.
int **values;             //pointer to a table of integer values.
```

For 'Basic' applications use:

```
Declare Function HsiReadDirect Lib "hsicom32" ( _
    ByVal inputcount As Integer, _
    ByRef inputlist As Byte, _
    ByRef values As Integer _
) As Integer
```

PARAMETERS:

int lvd_t_count - is the number of lvd_t's to be read by this call.

unsigned char *lvd_t_list - is a pointer to a list of the lvd_t numbers.

int **values - is a pointer to an integer pointer. The calling routine passes the address of a variable where the pointer to the table of values is to be stored.

RETURNS:

Return status.

DESCRIPTION:

The function will set the supplied integer pointer to address the list of returned integers.

COMMENTS:

The lvd_t numbers are 8 bit integers starting at "0" through "23" for the first HSI-24, "24" through "47" for the second HSI-24, etc. The first four DC inputs are "96" through "99", the next four DC inputs are "100" through "103", etc.

Each of the returned integers or values will be in the range of -8192 to +8191.

Appendix A: HSI-24 Formula Syntax

When a channel is read, its value is computed by a formula supplied to the HSI-24 by the host software. The formulae are supplied to the HSI-24 in the form of an ASCII string terminated by a null byte. Formulae are constructed in much the same way as they are in assignment statements in programming languages such as BASIC.

Formulae consist of combinations of functions, operators, constants, input terms and channel terms.

Peak Hold Functions

MAX returns the largest value it sees as an argument while scanning.

MIN returns the smallest value it sees as an argument while scanning.

TIR returns MAX-MIN.

The MAX, MIN and TIR functions continue to be scanned although the channel which uses them is not being read by the host software. They will 'freeze' at their current value if scanning is disabled. The scanning is controlled by the functions start_scan and stop_scan. The period of scanning is set by the set_scan_time function. They are reset by the reset_mm function.

Mathematical Functions

ABS returns the absolute value of the argument.

ACOS returns the arc-cosine of the argument. Result in radians.

ASIN returns the arc-sine of the argument. Result in radians.

ATAN returns the arctangent of the argument. Result in radians.

COS returns the cosine of the argument. Argument in radians.

GOF returns the greatest (most positive) of the argument list.

GOR returns the greatest (most positive) of the range set by the argument list.

LOF returns the least (most negative) of the argument list.

LOR returns the least (most negative) of the range set by the argument list.

SIN returns the sine of the argument. Argument in radians.

SQRT returns the square root of the argument.

SQR returns the argument multiplied by itself.

TAN returns the tangent of the argument. Argument in radians.

Miscellaneous Functions

PI2 has the value 3.141592654/2.0

PI has the value 3.141592654

RAD returns the argument converted from degrees to radians.

DEG returns the argument converted from radians to degrees.

(and) are allowed to impose computation order.

Mathematical Operators

^ exponentiation
* multiplication.
/ division. (division by zero returns zero
+ addition.
- subtraction.

Constants

Constants in formulae are specified as ASCII strings. Scientific notation is not allowed for constants, (i.e., '.125' is allowed, '1.25E-01' is not allowed).

Input Terms

T_n returns the value of transducer number n. The number n must be in the range 1 to 96. The HSI-24 will read the transducer, multiply by the transducer full scale value and add the transducer offset. Each transducer has an individual full scale value and offset.
A_n causes the value of analog number n to be returned. The number n must be in the range 1 to 16. The HSI-24 will read the analog input, multiply by the analog full scale value and add the analog offset. Each analog input has an individual full scale value and offset.

Channel Terms

C_n returns the value of channel number n. The number n must be in the range 1 to 96. The HSI-24 will compute the value of channel nn, multiply by the channel full scale value and add the channel offset. Each channel has an individual full scale value and offset. Circular references to channels are not allowed and will cause the HSI-24 to return an error code.

Example Formulae

T1
T1+T2
MAX(T2-T1)
(T1+T2+T3)/3
1.0034*(T1+T2)
(MAX(T1)+MIN(T1))/2
GOF(T1,T2,T3) returns the greatest of T1, T2 and T3.
LOF(T1,T2,T3,T4) returns the least of T1, T2, T3 and T4.
GOR(T1,T8) returns the greatest of T1,T2,T3,T4,T5,T6,T7 and T8.
LOR(C9,C13) returns the least of C9,C10,C11,C12 and C13.

Appendix B: Startup Settings

Immediately after loading the HSI-24 firmware the following settings will be valid.

All 96 channel scale factors are set to 1.

All 96 channel zero offsets are set to zero.

All 96 transducer full scale values are set to .08.

All 96 transducer zero offsets are set to zero.

The first 4 analog full scale values are set to 1.

The remaining 12 analog full scale values are set to zero. (Although these are accessible they are physically located on slave boards.

All 16 analog zero offsets are set to zero.

The floating point input and output modes are both set to the native (IEEE) mode.

Appendix C: Error Codes

Command Error

- **1** An invalid command or parameter was entered.
- **2 - 9:** Reserved.

Channel Definition Errors

When defining channels, error codes **10-22** may be returned. They may be used to help determine the part of the formula which is causing the problem.

- **10** An invalid channel number was entered.
- **11** Internal error..
- **12** Invalid opcode mnemonic.
A built-in function name has been improperly typed in. If "SINE(T4)" is entered instead of "SIN(T4)" or "T1+S3" instead of "T1+T3".
- **13** Not enough operands for operator.
If "T1+" is entered the HSI-24 cannot determine what value to add to T1. If "SIN()" is entered the HSI-24 cannot determine what value to use as the argument of the sine function.
- **14** Node table full.
The HSI-24 reduces the entered formulae into a form which can be quickly computed when required. The reduced formulae are stored in a table which has a predetermined size. When the table is filled this error is returned. If this error occurs, the number or the complexity of the entered formulae must be reduced. Determining the number of node entries in the node table that a particular formula consumes is difficult due to certain optimizations which the HSI-24 applies during the formula reduction process. The following guidelines will allow determining the maximum number of nodes which a given formula will consume.
 1. Each constant consumes one node. The formula ".0023" consumes one node.
 2. Each function use consumes one node for the function. Additional nodes will be consumed for the function's arguments. The formula "SIN(.0023)" consumes one node for the SIN function plus one node (Rule 1) for the constant .0023, for a total of 2 nodes. The TIR function is a special case which consumes 2 nodes just for the function plus the additional nodes for the function arguments. The formula "TIR(.0023)" consumes 2 nodes for the TIR function and 1 node for the constant, for a total of 3 nodes.
 3. Arithmetic operators (+ - * /) consume one node. The formula "1 + 2 + 3" consumes 1 node for each of the "+" operators plus 1 node for each of the constants (Rule 1), for a total of 5 nodes.
 4. Channel references consume one node. The formula "C1" consumes 1 node.
 5. Transducers and analog inputs consume one node. The formula "T1" consumes 1 node. The formula "T1 + T2" consumes 1 node for each of the transducers plus 1 node for the "+" operator (Rule 3), for a total of 3 nodes. The first appearance of a transducer in a formula consumes 1 node however subsequent appearances of the same transducer in any formula do not consume additional nodes.
Channel 1 is "T1+T2". This uses 3 nodes.

Channel 2 is "T1+T3". This uses 2 nodes. T1 has already been allocated a node by the channel 1 formula.

The total number of nodes available is 400 in firmware version 'PCA10.BIN'. The HSI-24 cannot conserve nodes by recognising common subexpressions in formulae. Node table space can be conserved by defining a channel with the subexpression and then referencing that channel in the formulae which need the subexpression value.

Channel 1 is "T5 - (T1 + T2 + T3 + T4)". (9 nodes)
Channel 2 is "T6 - (T1 + T2 + T3 + T4)". (5 nodes)
Channel 3 is "T7 - (T1 + T2 + T3 + T4)". (5 nodes)
Channel 4 is "T8 - (T1 + T2 + T3 + T4)". (5 nodes)
For a total of 9+5+5+5 or 24 nodes.

Alternately define channel 50 with the subexpression and use channel 50 in place of the subexpression:

Channel 50 is "T1 + T2 + T3 + T4". (7 nodes)
Channel 1 is "T5 - C50". (3 nodes)
Channel 2 is "T6 - C50". (3 nodes)
Channel 3 is "T7 - C50". (3 nodes)
Channel 4 is "T8 - C50". (3 nodes)
For a total of 7+3+3+3 or 16 nodes.

- **15** Bad transducer or analog input number.
If a formula contains "T175" or "A32". The highest transducer number allowed is "T96". The highest analog input number allowed is "A16". Note: Although the HSI-24 accepts transducer numbers up to 96 the slave HSI-24's must be present for these to produce any meaningful measurements.
- **16** Too many operands for operator.
If "T1 + T2 T3" is entered, the HSI-24 cannot determine what to do with the T3 part of the formula.
- **17** Bad numeric value in expression.
When entering constants in a formula "scientific notation" is not allowed. In some computer languages the value .0015 may be entered as "1.5E-3". The HSI-24 requires that the value be entered as ".0015".
- **18** Bad token. An invalid symbol was entered.
- **19** Formula too complex to parse.
This error occurs when a formula has more levels of parenthesis than the HSI-24 can handle
- **20** Recursive channel definition.
This error occurs when channels reference each other in a circular fashion.
Channel 1 has the formula "T1+T2-C2".
Channel 2 has the formula "T3+C1".
To determine the value of channel 1 the HSI-24 must first determine the value of channel 2, which is dependent on the value of channel 1.

- **21** No memory left to allocate.
Besides storing the formulae in a reduced form (as described under error 14), the original text of the formula is also saved within the HSI-24 memory. The formula text is stored in a memory pool along with other items which are necessary during formula entry. Should this pool of memory become filled this error is issued.
- **22** General formula error.
The formula scanner in the HSI-24 will issue this error when the formula is bad and no other formula error codes apply.