AccuRange™ 200 KHz High Speed Interface
PCI Formats

User's Manual

Rev. 7.2
For use with AccuRange HSIF- PCI (200 KHz version)
August 8, 2007

Acuity
A product line of Schmitt Measurement Systems, Inc.
2765 NW Nicolai St.
Portland, OR 97210
www.acuitylaser.com

Limited Warranty

Acuity, a division of Schmitt Measurement Systems, Inc., makes the following limited warranties. These limited warranties extend to the original purchaser and to no other purchaser or transferee.

Limited One Year Parts and Labor Warranty

Acuity warrants this product and its parts against defects in materials or workmanship for a period of one year after the date of original retail purchase. During this period, Acuity will, at its option, repair or replace a defective product or part without charge to you.

Warranty Conditions

The above LIMITED WARRANTIES are subject to the following conditions:

1. Warranties extend only to products manufactured by Acuity.

2. Warranties extend only to defects in materials or workmanship as limited above. Warranties extend only to defects which occur during normal use and do not extend to damage to products or parts which results from alteration, repair, modification, faulty installation or service by anyone other than an authorized Acuity service center, damage to products or parts caused by accident, abuse, misuse or maintenance, mishandling, misapplication, or damage caused by acts of God.

3. You must retain your bill of sale or provide other proof of purchase.

4. Any replacement parts furnished at no cost to the purchaser in fulfillment of this warranty are warranted only for the unexpired portion of the original warranty.

ALL WARRANTIES REQUIRED TO BE IMPLIED BY STATE LAW ARE EXPRESSLY LIMITED TO THE DURATION OF THE LIMITED WARRANTIES SET FORTH ABOVE. Some states do not allow limitations on how long an implied warranty lasts, so the above limitation may not apply to you. **WITH THE EXCEPTION OF ANY WARRANTIES REQUIRED TO BE IMPLIED BY STATE LAW AS HEREBY LIMITED, THE FOREGOING EXPRESS WARRANTY IS EXCLUSIVE AND IN LIEU OF ALL OTHER WARRANTIES.**

IN NO EVENT SHALL ACUITY BE LIABLE FOR SPECIAL, INCIDENTAL, CONSEQUENTIAL OR PUNITIVE DAMAGES, INCLUDING, WITHOUT LIMITATION, INJURY OR DAMAGE TO PERSONS OR OTHER PROPERTY, INCONVENIENCE, LOSS OF GOODWILL, LOST PROFITS OR REVENUE, LOSS OF USE OF THIS PRODUCT OR ANY ASSOCIATED EQUIPMENT, COST OF SUBSTITUTIVE EQUIPMENT DOWNTIME COSTS OR CLAIMS OF ANY PARTY DEALING WITH PURCHASER FOR SUCH DAMAGES, RESULTING FROM THE USE OF THIS PRODUCT OR FROM DEFECTS IN THIS PRODUCT, OR ARISING FROM BREACH OF WARRANTY OR CONTRACT, NEGLIGENCE OR ANY OTHER LEGAL THEORY. Some states do not allow the exclusion or limitation of incidental or consequential damages, so the above limitation may not apply to you.

Procedures for Obtaining Warranty Service

1. Contact your Acuity distributor or call Acuity to obtain a return merchandise authorization (RMA) number within the applicable warranty period. Acuity will not accept any returned product without an RMA number.

2. Ship the product to Acuity, postage prepaid, together with your bill of sale or other proof of purchase. Include your name, address, and description of the problem(s). Print the RMA number you have obtained on the outside of the package.

This device complies with part 15 of the FCC Rules. Operation is subject to the following two conditions:

(1) This device may not cause harmful interference, and (2) this device must accept any interference received, including interference that may cause undesired operation.

Note: This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to part 15 of the FCC rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this device in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at their own expense.

TABLE OF CONTENTS

1. AccuRange PCI High Speed Interface (HSIF) Card

1.1 General Description

The AccuRange PCI High Speed Interface (PCI HSIF) is an interface computer board that takes samples from the AR4000 optical rangefinder. Samples come over the bus in a 16 byte format that includes a 32 bit range value, two 32 bit encoder values, and a 32-bit status value that includes signal strength, ambient light, sensor internal temperature, and general purpose input bits. These inputs, along with external enable/disable control of sampling, allow precise synchronization with external events.

The interface board operates by measuring the range-dependent frequency output of the AR4000. To use the AR4000 sensor with HSIF card, the current loop option must not be installed in the sensor. The sensor frequency output is divided down by the HSIF card and the resulting signal is measured with a resolution of 125 ps. The sample rate of the interface is controlled by the sample rate configuration of the AR4000 device and the HSIF card. The maximum sample rate is 200 KHz.

The raw data collected by the HSIF card is not scaled or calibrated in any way. This output is used to create calibrated distance output using software modules and tables supplied with the interface or though user-written algorithms. The data can be used to calculate distance as each sample is collected, although the more typical application will collect a batch of samples and post-process them to create distance readings from the entire group after high-speed collection is finished. Data is collected in a buffered streaming mode.

Other features of the interface include memory buffer indicators for the number of samples available, external sample start/stop control, and three general purpose input bits that allow synchronous recording of events while sampling.

The board comes standard with power control circuitry for two small motors. This is not full servo control, but it allows motor power to be programmed. If the motors have encoders, the encoders may be sampled with the sensor data to provide position information in the sample stream in scanning systems. Each motor can be driven with up to 2 amps at 12 to 15 volts. Power for the DC-level controlled motors must be supplied to the board.

The AR4000 PCI High Speed Interface can be ordered with either single-ended or differential encoders. If you are using your own motor and encoder with a line scanner application, you must specify which encoder type you will be using with your PCI HSIF.

2. HSIF Card Installation

The PCI HSIF card is compatible with Windows® 2000, XP, and Vista. While the PC is turned "off", install the card into an available PCI bus slot.

2.1 Windows 2000 driver installation

When you turn on the PC, the following dialog box appears.

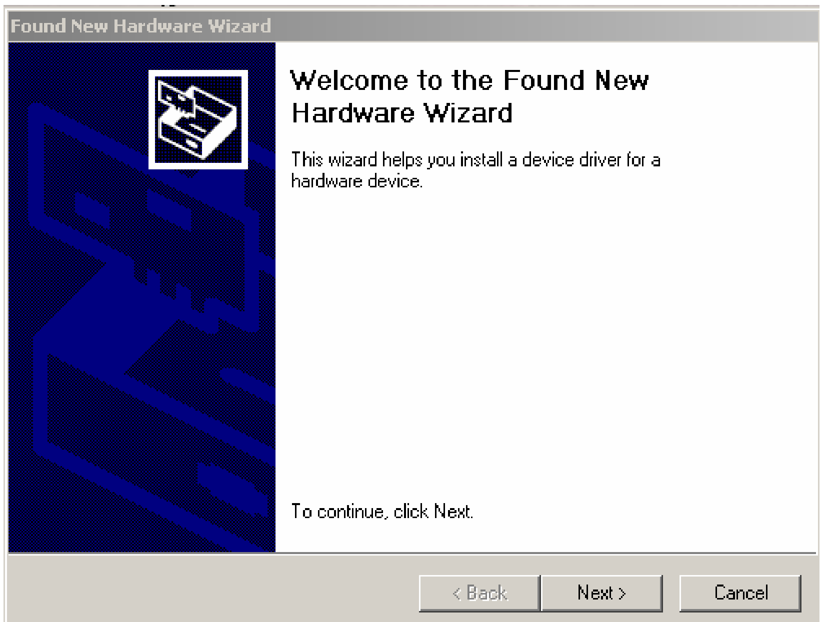

Figure 1 – Found New Hardware

Click *Next*. Then, the next dialog box appears.

Figure 2 - Install Hardware Device

Choose "Search for a suitable driver for my device" and click *Next*. The following box appears:

Figure 3 - Search for PCI Device Driver

Choose "CD-ROM drives" and make sure that the AccuRange PCI High Speed Interface installation CD-ROM is loaded into the CD-ROM drive prior to clicking *Next*.

The following box appears:

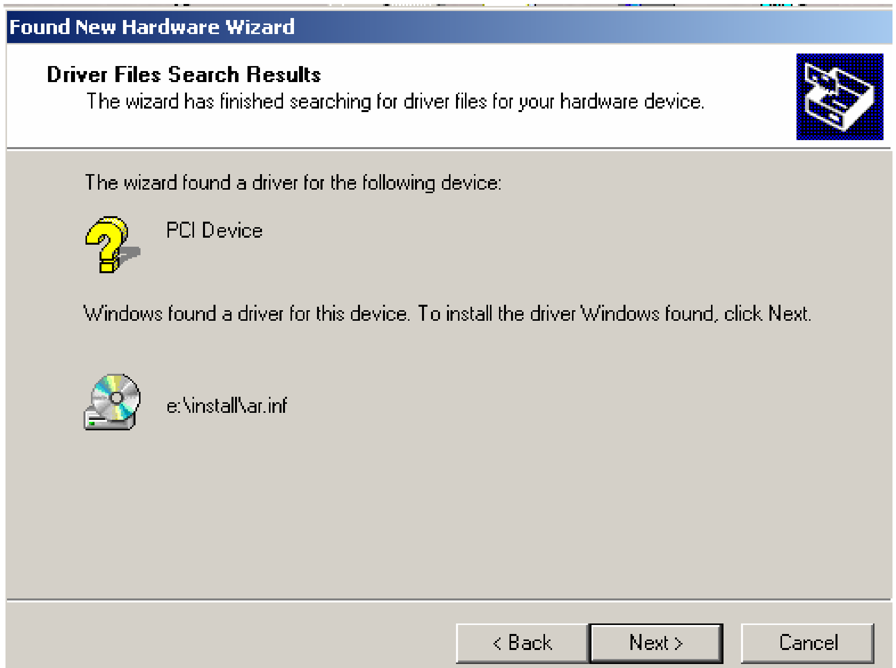

Figure 4 - Found Device Driver

Windows found the appropriate driver on the e:\ drive. Click *Next*.

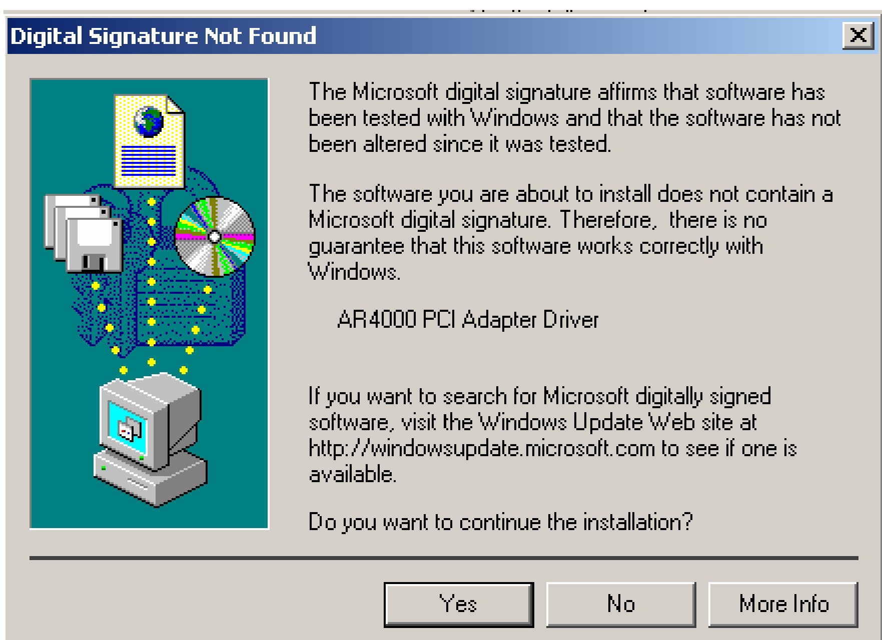

Figure 5 - Digital Signature

The driver should be successfully installed. Click *Yes*.

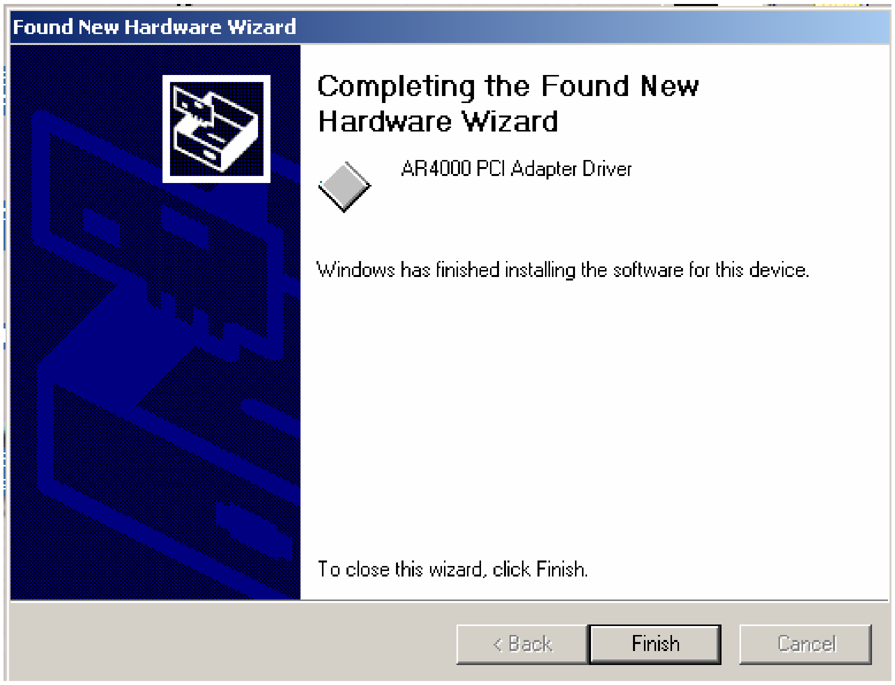

Figure 6 - Device Driver Installation Complete

Installation is complete. Click *Finish*.

2.2 Windows XP driver installation

Be sure the installation CD-ROM is in the drive and turn on the power to the PC.

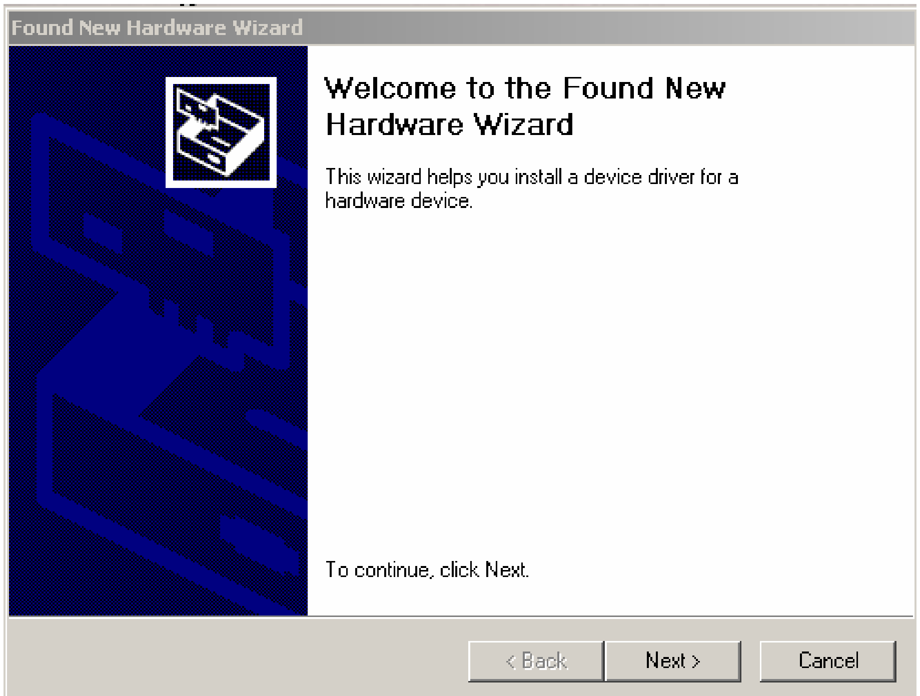

Figure 7 - Found New Hardware

Click *Next*. The following dialog box appears:

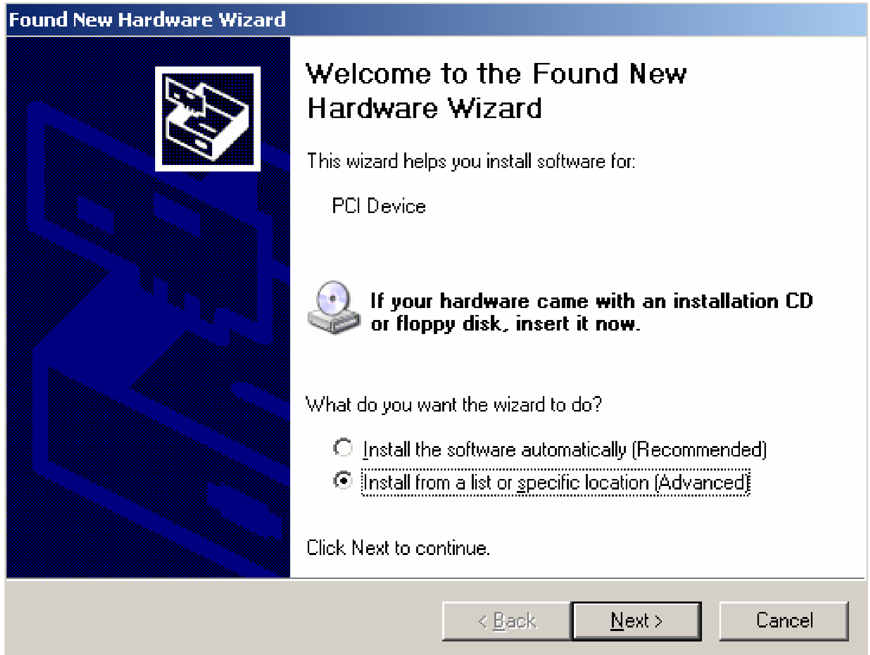

Figure 8 - Device Driver Location

Choose "Install from a list or specific location (Advanced)" and click *Next*. The following window appears:

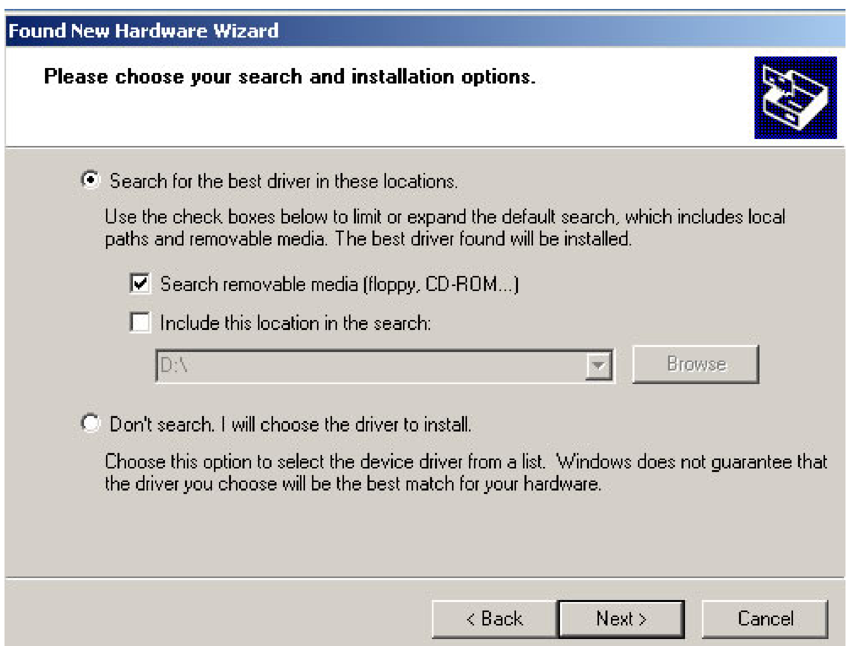

Figure 9 - Search Options for Driver

Choose the selection that will read from your CD-ROM drive and click *Next*.

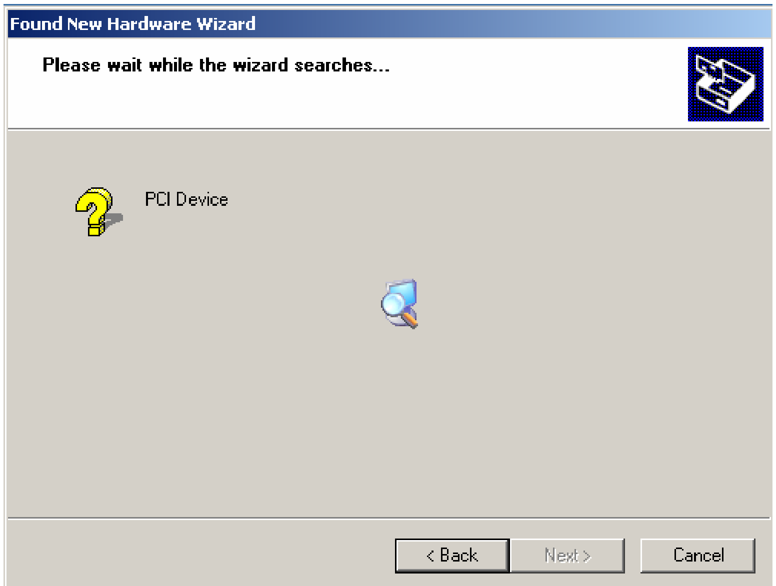

Figure 10 - Search for Driver

This window appears while the system searches for the PCI driver.

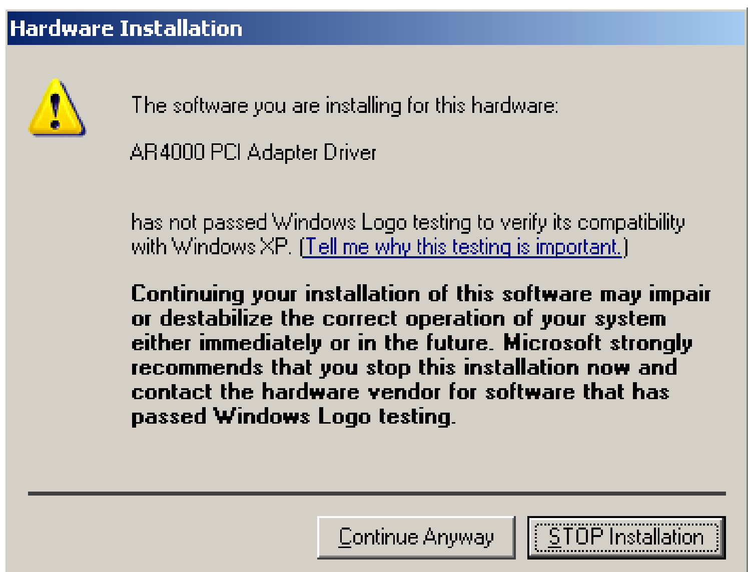

Figure 11 - Windows Logo Testing

This window confirms the hardware driver selection. Click *Continue Anyway*.

Figure 12 - Completing Driver Installation

When the hardware installation has completed, click *Finish*.

3. Included Software

3.1 CD Directory Tree

The HSIF product includes a CD with the following contents:

\install	- driver installation files
\dll	- dynamic link library (DLL) files
\docs	- API and Users Guide
\examples\bin	- precompiled programs and DLL, built under Windows XP
\examples\distance	- example *distance* project and source code
\examples\hsiftest	- example *hsiftest* project and source code
\examples\rt_cap	- example *rt_cap* project and source code
\examples\utilities	- serial interface and DLL utilities
\calibration	- calibration files for sensor and HSIF PCI card

3.2 Demonstration Programs

Three demonstration programs with C source code which run under Windows 2000/XP are supplied with the AccuRange High Speed Interface: *hsiftest*, *distance*, and *rt_cap.* The programs were compiled with Microsoft Visual C++ 6.0 under Windows XP and tested with Windows 2000, XP, and Vista.

3.2.1 hsiftest

hsiftest is a command-line program built from the project "hsiftest" located in directory *\examples\hsiftest* on the CD.

hsiftest exercises all off the features of the HSIF card. The program requires three command line arguments and can accept an optional one. The required arguments are: the serial port number the AccuRange is connected to (1-8); the HSIF card the AccuRange is connected to (0-4); and the calibration filename. An optional argument can be added to specify whether to run the program on multiple CPU cores (0) or a single CPU core (1). This option is useful for some multiprocessor/multi-core CPUs that have trouble keeping up with the sensor PCI bandwidth requirements when run in multiprocessor/multi-core mode.

hsiftest is called by entering the following command-line where required parameters are denoted as <*required parameter*> and optional parameters are denoted as [*optional parameter*]:

```
hsiftest <serialport (1-8)> <boardnumber (0-4)>
<calibration   filename> [CPUs (0 = all CPUs (default), 1 =
1 CPU only)]
```

As an example of using *hsiftest*, make a local copy of the CD's *\examples\bin* directory on your hard disk. This directory contains *hsiftest.exe* and *hsif.dll*. Next, copy the calibration file *lookuphs* from CD's *\calibration* to this directory. Open a Windows Command Prompt Window and change to the directory you created. Next, enter the following line, which assumes the AccuRange is connected to COM1 and that there is a single HSIF card installed:

```
hsiftest 1 0 lookuphs
```

Once *hsiftest* is started, follow the instructions on the screen.

If any of the sampling tests fail, run the program on a single CPU by entering the following command:

```
hsiftest 1 0 lookuphs 1
```

3.2.2 distance

distance is a command-line program built from the project "distance" located in directory *\examples\distance* on the CD included with the PCI HSIF card.

distance samples target distances using the HSIF card, averages the distances, and prints them to a console window. The program requires six arguments and can accept two optional ones. The required arguments are: the serial port number the AccuRange is connected to (1-8); the HSIF card the AccuRange is connected to (0-4); the sampling period in microseconds; the calibration filename; the number of samples to average per print out; and number of lines to print before the program ends.

Two optional arguments are also allowed. First, you can specify the maximum range the sensor. The valid max range should be set to a value between 2 and 650 inches. Second, you can set whether to run the program on one or multiple CPUs. The number of CPU cores the program will utilize is specified as 0 for multiple CPU cores or 1 for a single CPU core. This option is useful for some multiprocessor/multi-core CPUs that have trouble keeping up with the sensor PCI bandwidth requirements when in multiprocessor/multi-core mode.

```
distance <serialport (1-8)> <boardnumber (0-4)> <sampling
period 5 – 9999999 microseconds> <calibration filename>
<num samples to average> <num line to print> [max range 2
to 650 inches (default = 650)] [CPUs (0 = all CPUs
(default), 1 = 1 CPU only)]
```

As an example of using *distance*, make a local copy of the CD's *\examples\bin* directory on your hard disk. This directory contains *distance.exe* and *hsif.dll*. Next, copy the calibration file *lookuphs* from CD's *\calibration* to this directory. Open a Windows Command Prompt Window and change to the directory you created. Next, enter the following line, which assumes the AccuRange is connected to COM1 and that there is a single HSIF card installed:

```
distance 1 0 100 lookuphs 10000 500
```

This command line configures the sampling period to 100µs, the number of samples to average per output line to 1,000, and the number of output lines to print before exiting to 500.

Note that the program can be terminated early by hitting the ‘Q’ or ESC key. Also, if you see any overflow errors printed to the screen while the program executes, add the option to run the program on a single CPU core as follows:

```
distance 1 0 100 lookuphs 10000 500 1
```

To specify a maximum distance of 400 inches, you would enter:

```
distance 1 0 100 lookuphs 10000 500 1 400
```

Note that the last two optional command line arguments can be specified in any order.

3.2.3 rt_cap

rt_cap is a command-line program built from the project “rt_cap” located in directory *\examples\rt_cap* on the CD.

rt_cap is a real-time capture utility that captures raw samples from the HSIF card, post-processes them, and writes the post-processed samples to a data file. Unlike the previous two examples, all of the configuration options are specified in a separate input configuration file. The program is run from the command line with the configuration filename as its argument. The program can be terminated early by hitting the ‘Q’ or ESC key.

```
rt_cap <raw.cfg>
```

3.2.3.1. Configuration file

The configuration file, *raw.cfg,* is a text file that specifies one configuration parameter per line. There is an example configuration file, *raw_2.cfg*, included in the *\examples\bin* directory of the CD that covers version 2 of the configuration file. This file listing of *raw_2.cfg* and a line-by-line description is presented below:

```
2          ;Configuration file version 2
1          ;Set COM port sensor is connected to (first line of file)
1          ;HSIF PCI card number installed
5          ;Sampling period (us) (5 us -> 200kHz sampling rate)
650        ;Max range in inches
30000      ;Buffer size for storing samples
2000000    ;Number of samples to capture
```

```
190        ;Motor power 1 (0..255)
190        ;Motor power 2 (0..255)
4096       ;Motor 1 pulses per revolution
4096       ;Motor 2 pulses per revolution
lookuphs   ;Calibration file, place in same directory as capture executable
v2_data    ;Outuput capture data filename. A .raw extension is automatically appended
5000       ;Delay in (ms) before acquisition starts, allow motor speed to stabilize
0          ;Force process to run on 1 CPU.  Set to 1 to use 1 CPU, else 0 to use multiple CPUs
0          ;Set to 1 to save .raw file, else 0.  This option is used by rt_cap only.
```

Line 1 specifies the version number of the input configuration file. It should be set to 2.

Line 2 specifies the com port that the AccuRange sensor is connected to. For instance, specifying 1 will use COM1.

Line 3 specifies the HSIF card installed in the system. This parameter can be set to 1 because currently only one installed HSIF card is supported.

Line 4 specifies the sampling period in microseconds. This can be set from 5 to 9999999 microseconds.

Line 5 specifies the maximum range the target will be from the sensor. This can be set from 2 to 650 inches.

Line 6 specifies the maximum buffer size used for requesting samples from the driver. This can be set from 1 to 30,000. For fast sampling rates this should be set to 30,000.

Line 7 specifies the number of samples to save to the hard disk before exiting.

Line 8 specifies the motor power 1. It can take a value from 0 (off) to 255 (max power).

Line 9 specifies the motor power 2. It can take a value from 0 (off) to 255 (max power).

Line 10 specifies the motor 1 encoder counts per revolution. The high resolution encoders that come with the AR4000 line scanner have 4096 counts per revolution.

Line 11 specifies the motor 2 encoder counts per revolution.

Line 12 specifies the calibration file used for post-processing the samples from encoded raw distances to calibrated output distances.

Line 13 specifies the base name of the output files generated by the program. For instance, if *v2_data* is specified, the program will produce an output file *v2_data.prc* that contains the post-processed samples, and optionally *v2_data.raw*, that contains the raw samples.

Line 14 specifies the amount of delay in milliseconds between when the motor power is applied and the data capture begins. This allows the motors to achieve a constant angular velocity before sampling starts.

Line 15 specifies whether to run the *rt_cap* utility on multiple CPUs or a single CPU core. This is sometimes useful for some multiprocessor/multi-core systems that loose samples at faster sampling rates when multiple cores compete for the memory and PCI bus. When sample loss is detected, an error message is output to the console.

Line 16 specifies whether to output both the post-processed samples and the raw samples, or just the post-processed samples. Because it requires twice as many disk write accesses to create both files, samples could be lost at high sampling rates. To prevent this, it is recommended to set this value to 0 so that only the post-processed samples are written to disk. Setting this value to 1 will write both raw and post-processed samples to disk.

3.2.3.2. Post-processed Output

The post-processed output is a binary file that consists of two sections. First is a header that describes the sampling conditions when the post-processed file was created. Following that is the array of recorded samples. All of the binary data elements are stored in little-endian (Intel) format.

The header is a packed structure which consists of the following:

```
uint32 version;         // Version of file produced by post_proc
uint32 samp_period;     // sampling period in (us) (* used in post process)
uint32 max_range;       // inches (i.e. 650) (* used in post process)
uint32 buf_size;        // buffer size in RAM for samples (i.e. 30000 max)
uint32 num_samples;     // number of samples to capture
uint32 encoder_ppr[2];  // Encoder 1 and 2 pulses per revolution
int32  cal_val;         // calibration value  (* used in post process)
uint8  motor_power[2];  // Motor 1 and 2 power settings
uint8  comport;         // comport used to communicate with AR4000
uint8  card_num;        // PCI card number, generally 1
int8   cal_file[256];   // Zero-terminated calibration file (* used in post process)
```

version is a 32-bit unsigned value that indicates the version number of the post-processed sample file. This field allows for backwards compatibility if the file format changes in the future.

samp_period is a 32-bit unsigned value that indicates the sampling period in microseconds.

max_range is a 32-bit unsigned value that indicates the maximum range the sensor was configured for.

buf_size is a 32-bit unsigned value that indicates the buffer size used to retrieve samples from the HSIF driver.

num_samples is a 32-bit unsigned value that indicates the number of post-processed samples the file contains following the header.

encoder_ppr[2] is an array of two 32-bit unsigned values that indicate motor 1 and 2 encoder pulses (counts) per revolution.

cal_val is a 32-bit signed value that indicates the HSIF calibration value recorded when the samples were taken. This value is required to calculate an absolute range value from the encoded raw range.

motor_power[2] is an array of two 8-bit unsigned values that indicate motor 1 and 2 power settings.

comport is an 8-bit unsigned value that indicates the serial port used to communicate with the AR4000.

card_num is an 8-bit unsigned value that indicates the HSIF card number used.

cal_file[256] is a 256-byte array of signed 8-bit values that contains the calibration filename as a NULL-terminated string. Normally this field contains "*lookuphs*".

Following this, each post-processed sample is recorded as a packed structure:

```
uint32  rawRange;  // raw range after conversion from encoded raw value to an absolute value
float   distance;  // calbrated distance
float   caltemp;   // calibrated temperature, degrees F.
uint16  turns1;    // counted from encoder reset
uint16  count1;    // encoder counts from encoder zero mark (modulo counts/revolution)
uint16  turns2;    // counted from encoder reset
uint16  count2;    // encoder counts from encoder zero mark (modulo counts/revolution)
uint8   ambient;   // background illumination same as HSIF_SAMPLE
uint8   amplitude; // reflected signal strength same as HSIF_SAMPLE
uint8   status;    // FIFO and encoder index latches, same as HSIF_SAMPLE + timerout
uint8   timeout;   // TDC timeout while measuring range, sample invalid
```

rawRange is a 32-bit unsigned integer that represents an uncalibrated truncated distance to the target in arbitrary units. This value is an absolute distance generated from the encoded raw range. This distance, along with the ambient, amplitude, and temperature, could be used to generate a calibrated lookup table.

distance is a 32-bit floating-point calibrated distance that is generated from the encoded raw range based on the calibration file, temperature, ambient light, and amplitude measurements.

caltemp is a 32-bit floating-point calibrated temperature that represents the AR4000 hardware temperature in degrees Fahrenheit.

turns1 is a 16-bit unsigned value that represents the number of complete revolutions of motor 1 since the last time the counter was reset.

count1 is a 16-bit unsigned value that represents the number of counts that the motor 1 encoder has counted since the last complete revolution. *count1* increments from 0 to 1 – encoder counts per revolution as the motor turns.

turns2 is a 16-bit unsigned value that represents the number of complete revolutions of motor 2 since the last time the counter was reset.

count2 is a 16-bit unsigned value that represents the number of counts that the motor 2 encoder has counted since the last complete revolution. *count2* increments from 0 to 1 – encoder counts per revolution as the motor turns.

ambient is an 8-bit unsigned value that represents the background light illumination present when the sample was recorded.

amplitude is an 8-bit unsigned value that represents the amplitude of the reflected laser pulse.

status is an 8-bit unsigned value that contains the fields described in Table 1.

Bits	Description
0	Sticky bit set if the HSIF card L1 FIFO overflows. This indicates the driver is not reading samples fast enough from the card.
1	INPUT3
2	Motor 1 index pulse when used with an encoder with an index pulse. Otherwise, general purpose INPUT1
3	Motor 2 index pulse when used with an encoder with an index pulse. Otherwise, general purpose INPUT2
7-4	Sequential 4-bit count that increments with each sample and wraps to zero. Used internally by HSIF driver for determining the number of residual samples available in the HSIF card L1 FIFO.

Table 1 - Status Bits

timeout an 8-bit unsigned value that indicates that a range reading was not taken during this sample period if its value is non-zero. If the value is zero, a successful measurement was made. A timeout can occur if the target distance is outside of the maximum range or if the 9-pin AR4000 I/O connector cable is damaged or not connected properly to the HSIF card.

3.2.3.3. Raw Output

The optional raw output file consists of two sections. First is a header that describes the contents of the file. Second is an array of raw samples.

The header is a packed structure which consists of the following:

```
uint32 version;         // Version of file produced by raw_cap
uint32 samp_period;     // sampling period in (us) (* used in post process)
uint32 max_range;       // inches (i.e. 650) (* used in post process)
uint32 buf_size;        // buffer size in RAM for samples (i.e. 30000 max = 65000)
uint32 num_samples;     // number of samples to capture
uint32 encoder_ppr[2];  // Encoder 1 and 2 pulses per revolution
int32  cal_val;         // calibration value  (* used in post process)
uint8  motor_power[2];  // Motor 1 and 2 power settings
uint8  comport;         // comport used to communicate with AR4000
uint8  card_num;        // PCI card number, generally 1
int8   cal_file[256];   // Zero-terminated calibration file (* used in post process)
```

The header is identical to the one detailed in 3.2.3.2 *Post-processed Output*.

Following this, each raw sample is recorded as a packed structure:

```
uint32  status;    // FIFO and encoder index latches, temp, amb, amp
uint32  encoder1;  // Motor encoder 1 counts
uint32  encoder2;  // Motor encoder 2 counts
uint32  range;     // Encoded range reading
```

These fields are described in Table 2:

DWord #	Bit #	Contents
0	0	Hardware Buffer Overflow Indicator
	1	Input 3
	2	Motor 1 Encoder Index / Input 1
	3	Motor 2 Encoder Index / Input 2
	7-4	Sample Count 0-15
	15-8	8-bit Sensor Internal Temperature
	23-16	8-bit Ambient Light Sample
	31-24	8-bit Amplitude Sample
1	31-0	32-bit Motor 1 Encoder Position
2	31-0	32-bit Motor 2 Encoder Position
3	31-0	32-bit Encoded Raw Range

Table 2 - Sample Data Format

3.2.3.4. rt_cap Example

As an example of using *rt_cap*, make a local copy of the CD's *\examples\bin* directory on your hard disk. This directory contains *rt_cap.exe*, *hsif.dll*, and *raw_2.cfg*. Next, copy the calibration file *lookuphs* from CD's *\calibration* to this directory. Open a Windows Command Prompt Window and change to the directory you created. Next, enter the following line:

```
rt_cap raw_2.cfg
```

If the processor has any trouble keeping up with short sampling periods, such as 5 microseconds, a buffer overflow may occur in the HSIF card or the driver. The HSIF buffer is known as the L1 FIFO buffer and the driver buffer is known as the L2 FIFO buffer. If the program outputs an L1 FIFO overflow error and a multiprocessor/multi-core machine is in use, set the parameter on line 15 of configuration file, *raw_2.cfg,* to 1. This will run *rt_cap* on a single CPU. It that does not clear the problem, see Section 3.3 to boot the computer with a single core.

If an L2 FIFO buffer overflow error occurs, the program is not requesting samples quickly enough from the driver. To fix this, set the buffer size to a larger value, such as 30,000. Also, minimize disk write accesses by turning off the option to write the *.raw* file to disk. This is done by setting line 16 of the configuration file to 0.

3.3 Maximizing Sampling Performance

Some multiprocessor/multi-core CPU systems cannot achieve the maximum 200 KHz sampling rate because background processes compete for CPU utilization. Also, some multiprocessor/multi-core systems cannot achieve the maximum sampling rate because the cores compete for memory and PCI bus bandwidth.

For best results, multitasking should be kept to a minimum when sampling. However, many system background processes cannot be disabled. Therefore, each example boosts its priority to *Above Normal* before it starts sampling. This ensures background processes are preempted by the example when new distance samples become available.

For multiprocessor/multi-core systems, competition for the memory and PCI bus can be reduced by ensuring the examples run on only a single CPU. However, when multi-tasking, even this option may not be enough to ensure the HSIF card does not overflow its buffers and loose samples.

Therefore, to ensure that only one CPU has access to the memory and PCI bus, the computer can be booted in single CPU mode. To do this on a Windows 2000 or XP machine, add the option */numproc=1* to the end of the boot configuration line in the *boot.ini* file. This file is located in the root directory of the boot disk, normally *C:\boot.ini*. This file is a hidden system file and you will need to make it visible by making system files visible and making hidden files visible in the Explorer "Folder View" options.

An example *boot.ini* is shown below with the boot option "XP One CPU" added:

```
[boot loader]
timeout=30
default=multi(0)disk(0)rdisk(0)partition(1)\WINDOWS
[operating systems]
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Microsoft Windows XP Professional" /fastdetect
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="XP One CPU" /fastdetect /numproc=1
```

The next time you boot the Windows 2000/XP machine, Windows will ask you to select a boot option. Select the "XP One CPU" boot option.

For Windows Vista, run *msconfig.exe*. This will launch the "System Configuration" window shown below:

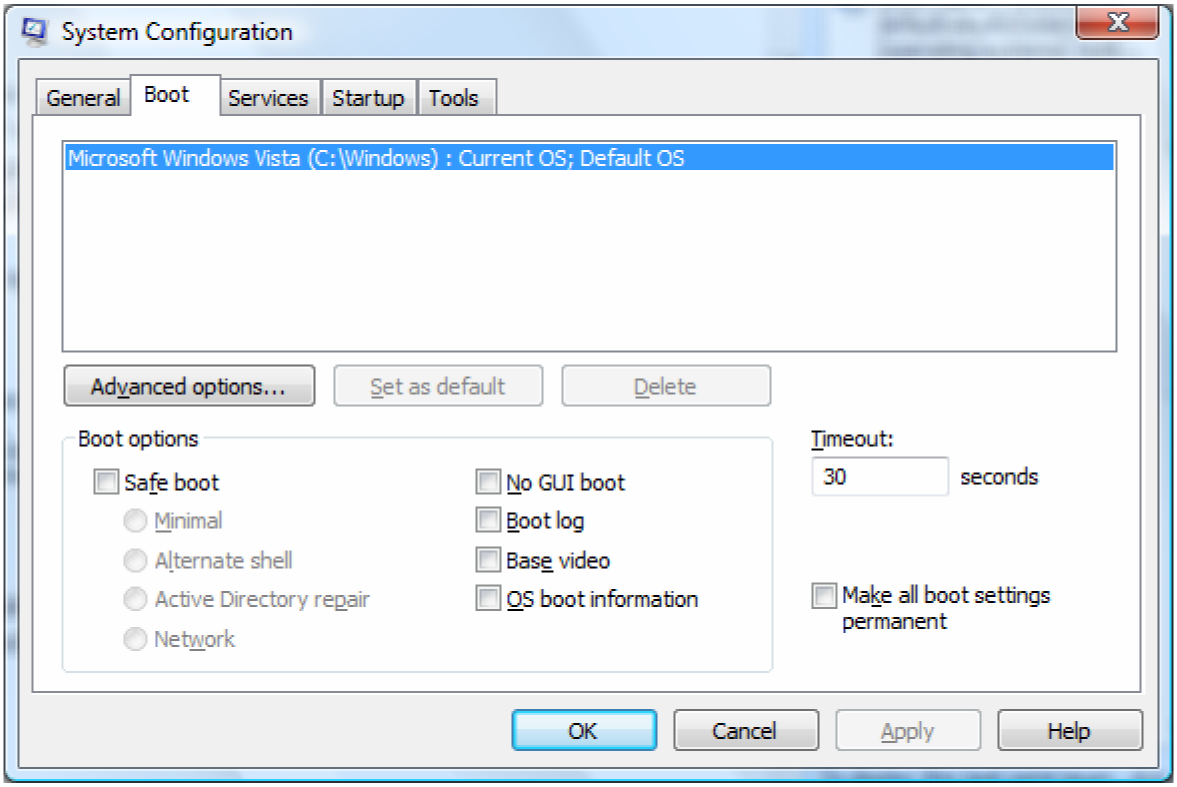

Figure 13 - Windows Vista System Configuration

Click *Advanced options…* to show the *BOOT Advanced Options* window.

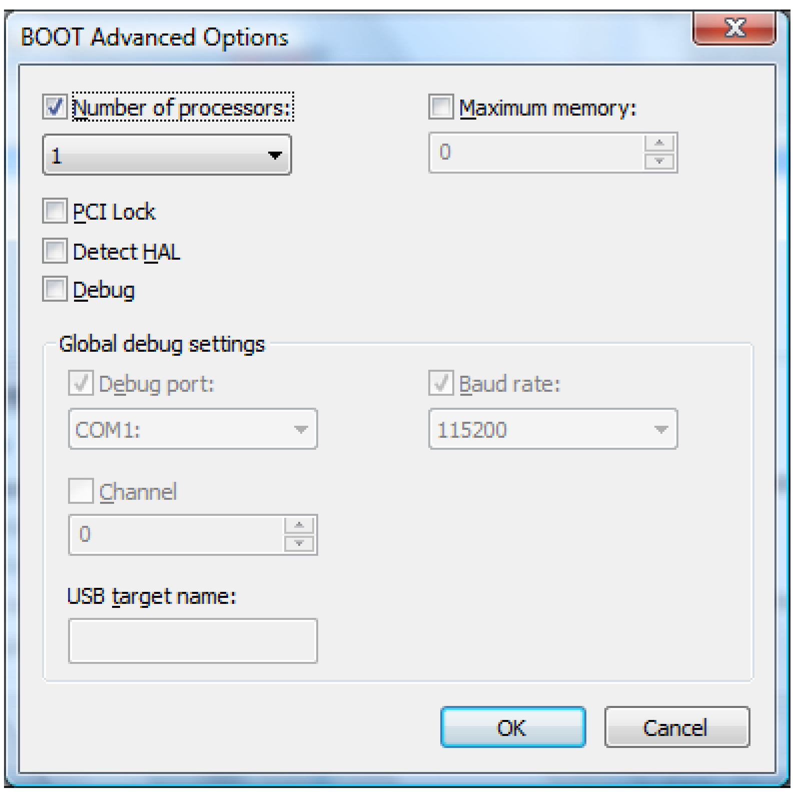

Figure 14 - BOOT Advanced Options

Acuity Laser Measurement

Check *Number of processors:* and select *1* from the list box. Click *OK* to close the window and then, click *OK* to close the System Configuration window.

The next time you reboot the Vista machine it will run on a single CPU.

4. Sensor Configuration and Sample Rate

When using the HSIF card, all configurations of the AR4000 are done via the serial port, just as it would be when using the sensor without the HSIF card. The communication path is one-way only from the AR4000 to the HSIF card; the sensor cannot be configured through the HSIF card. Therefore, the serial port must be used to communicate configuration data to the AR4000.

The sample rate of the interface is controlled by a 26-bit register in the HSIF card that is clocked at 40 MHz. The minimum sampling period that can be set is 5 microseconds. The maximum sample period is 1.6 seconds.

5. Motor Power

The HSIF card has two motor power control and encoder reading channels. Each motor may be set to one of 256 software controlled power levels via commands to the board. If the motors have quadrature encoders that are connected to the encoder inputs, two 32-bit encoder counts will be inserted into the data stream, giving the position of each motor. If the encoders provide index pulses, these can be applied to two of the general purpose input lines and used to determine the absolute positions of the motors. See the description of the 25 pin I/O connector for encoder connection details.

If motors are to be driven by the power amplifier on the board, the motors and motor power must be connected to P2. Motor 1 should be connected between pins 14 and 16, and motor 2 between pins 1 and 2. A separate power supply is required to drive the motors. Connect the motor power supply to pin 3 and the power supply ground to pin 15.

6. I/O Connectors

There are three connectors on the HSIF card. The 9 pin connector (P1) supplies power and receives signals from the AR4000 sensor. The 25 pin connector (P2) is used for powering the motors, reading the motor encoders, general purpose inputs, and sample control input.

6.1 9 Pin Power and Signal Connector P1

Pin	4000 Wire	Function	Direction
1	Red	Power, +5V (5-6V)	Out
2	Black	Ground	
3	Orange	Heater Power, +5V (4.5-7V)	Out
4	Brown	Heater Power Return	
5	Yellow	Temperature, 0-5 V	In
6	Blue	Range Frequency Signal	In
7	Green	Ambient light signal, 0-5 V	In
8	Purple	Amplitude signal, 0-5 V	In
9	Not Used	Laser Control, 0-5V	Out

Table 3 - P1: Power and Signal Connector Wiring

6.2 Power and Signal Connector Description

The line descriptions for P1 are the same as the descriptions of the power and signal lines in the AR4000 Power and Signal Cable Wire Description section. Pins 1-4 supply sensor power and sensor heater power and ground lines. The remaining lines are inputs for the signals from the AR4000 sensor. Pins 5, 7, and 8 are the inputs for the analog signals, with 2K impedance. Pin 6 is the input for the range frequency signal that is transmitted by the dedicated coaxial cable.

Figure 15 - P1 9-pin connector with 200 KHz range frequency cable

6.3 25 Pin I/O Connector P2

P2 includes general purpose input lines, a sample start/stop control line, quadrature encoder input lines, and power for encoders or other applications. There are two configurations, one for a *single-ended encoder* and one for an index pulse, and one for a *differential encoder* and index pulse.

6.3.1 P2 Pin Descriptions: Single-Ended Encoder

Below is a table of pin descriptions for a PCI HSIF card with Single-Ended Encoders. See section 1.4.3.2 for Differential Encoders.

	Top Row			***Bottom Row***	
Pin	**Function**	**Direction**	**Pin**	**Function**	**Direction**
1	Motor 2 Control	Out	14	Motor 1 Control	Out
2	Motor 2 Return	Out	15	Motor Power Ground	
3	Motor Power Supply	In	16	Motor 1 Return	Out
4	Ground		17	Laser Control	Out
5	+5V Power, 100 mA.	Out	18	+5V Power, 100 mA	Out
6	Ground		19	Motor 2 Encoder Ch A	In
7	Ground		20	Motor 2 Encoder Ch B	In
8	Ground		21	Motor 1 Encoder Ch A	In
9	Ground		22	Motor 1 Encoder Ch B	In
10	Ground		23	Ground	
11	Ground		24	General Purpose Input 1/ Encoder 1 Index Pulse	In
12	Start/Stop Sample Ctrl	In	25	General Purpose Input 3	In
13	General Purpose Input 2/ Encoder 2 Index Pulse	In			

Table 4 - P2: Single-Ended I/O Connector

Pin 1: Motor 2 Control

If used, motor 2 should be connected between this pin and pin 2. The output voltage level is varied as commanded to control the variable voltage motor.

Pin 2: Motor 2 Return.

If used, motor 2 should be connected between this pin and pin 1.

Pin 3: Motor Power.

The external power supply for the motors should be applied to this line, at +12 to +15 Volts, depending on the motor used. The line may draw up to 2 amps.

Pin 4: Ground

May be used as ground for encoders or other hardware powered by +5V on pins 5 and 18

Pin 5: +5V power output.

Primarily intended as power for the motor 1 encoder, but it may be used to drive other hardware, up to 100 mA maximum

Pins 6-10: Ground

May be used as ground for encoders or other hardware powered by +5V on pins 5 and 18

Pin 11: Ground.

May be used as ground for encoders or other hardware powered by +5V on pins 5 and 18

Pin 12: Start/Stop Sample Control Input.

When high, this input enables sampling and samples will be taken until the on-board buffer is full. When pulled low, sampling will stop. Samples are always completed, so that a full 8 byte sample is always buffered. This line is pulled up with an on-board 10Kohm resistor, so sampling is enabled when the input is left open. The first sample following resumption of sampling after stopping the sampling will not contain valid data, and must be read and discarded.

Pin 13: General purpose input bit 2 / Motor 2 index pulse input.

This may be used to sample external signals. The value of the bit is included in the sampled data stream. This pin is intended to sample motor encoder index pulses or other events to synchronize the sample data with the event. The signal is latched high so that any high signal of 100 nanoseconds or longer during a sample interval will appear as a high level following sample. This is intended for use with encoder index pulses.

Pin 14: Motor 1 Control.

If used, motor 1 should be connected between this pin and pin 14. The output voltage level is varied as commanded to control the variable voltage motor.

Pin 15: Motor Power Ground.

The external power supply ground for the motors should be connected to this pin.

Pin 16: Motor 1 Ground.

If used, motor 2 should be connected between this pin and pin 16.

Pin 17: Laser Control.

A 0-5V signal used to turn the laser on or off.

Pin 18: +5V power.

Primarily intended as power for the motor 2 encoder, but it may be used to drive other hardware, up to 100 milliamps maximum

Pin 19: Motor 2 Encoder Channel A.

If the motor control option is installed on the board, this input is decoded with pin 20 as a quadrature encoder signal from motor 2. The input should be a TTL-level signal and may switch at up to 1.5 MHz. The encoder positions are converted to 8 bit position values that ate included in the data stream. Each transition of pins 19 or 20 causes an up or down count in the position, so each quadrature cycle is effectively multiplied by 4 for the best possible resolution.

Pin 20: Motor 2 Encoder Channel B.

If the motor control option is installed on the board, this input is decoded with pin 19 as a quadrature encoder signal from motor 2. The input should be a TTL-level signal and may switch at up to 1.5 MHz.

Pin 21: Motor 1 Encoder Channel A.

If the motor control option is installed on the board, this input is decoded with pin 22 as a quadrature encoder signal from motor 1. The input should be a TTL-level signal and may switch at up to 1.5 MHz. The encoder positions are converted to 8 bit position values that are included in the data stream. Each transition of pins 21 or 22 causes an up or down count in the position, so each quadrature cycle is effectively multiplied by 4 for the best possible resolution.

Pin 22: Motor 1 Encoder Channel B.

If the motor control option is installed on the board, this input is decoded with pin 21 as a quadrature encoder signal from motor 1. The input should be a TTL-level signal and may switch at up to 1.5 MHz.

Pin 23: Ground

May be used as ground for encoders or other hardware powered by +5V on pins 5 and 18.

Pin 24: General purpose input bit 1 / Motor 1 index pulse input.

This may be used to sample external signals. The value of the bit is included in the sampled data stream. This pin is intended to sample motor encoder index pulses or other events to synchronize the sample data with the event. The signal is latched high so that any high signal of 100 nanoseconds or longer during a sample interval will appear as a high level following sample. This is intended for use with encoder index pulses.

Pin 25: General purpose input bit 3.

This may be used to sample external signals. The value of the bit will be inverted and inserted into the sample data stream. This may be used to sample events in order to synchronize the sample data with the event.

6.3.2 P2 Pin Descriptions: Differential Encoders

Below is a table of pin descriptions for a PCI HSIF card with Differential Encoders:

Pin	***Top Row*** **Function**	**Direction**	**Pin**	***Bottom Row*** **Function**	**Direction**
1	Motor 2 Control	Out	14	Motor 1 Control	Out
2	Motor 2 Return	Out	15	Motor Power Ground	
3	Motor Power Supply	In	16	Motor 1 Return	Out
4	Ground		17	Laser Control	Out
5	+5V Power, 100 mA.	Out	18	+5V Power, 100 mA	Out
6	Motor 2 Encoder Ch A-	In	19	Motor 2 Encoder Ch A+	In
7	Motor 2 Encoder Ch B-	In	20	Motor 2 Encoder Ch B+	In
8	Motor 1 Encoder Ch A-	In	21	Motor 1 Encoder Ch A+	In
9	Motor 1 Encoder Ch B-	In	22	Motor 1 Encoder Ch B+	In
10	Ground		23	General Purpose Input 1/ Encoder 1 Index Pulse-	In
11	General Purpose Input 2/ Encoder 2 Index Pulse-	In	24	General Purpose Input 1/ Encoder 1 Index Pulse+	In
12	Start/Stop Sample Ctrl,	In	25	General Purpose Input 3	In
13	General Purpose Input 2/ Encoder 2 Index Pulse+	In			

Pin 1: Motor 2 Control

If used, motor 2 should be connected between this pin and pin 2. The output voltage level is varied as commanded to control the variable voltage motor.

Pin 2: Motor 2 Return.

If used, motor 2 should be connected between this pin and pin 1.

Table 5 - P2: Differential I/O Connector

Pin 3: Motor Power.

The external power supply for the motors should be applied to this line, at +12 to +15 Volts, depending on the motor used. The line may draw up to 2 amps.

Pin 4: Ground

May be used as ground for encoders or other hardware powered by +5V on pins 5 and 18.

Pin 5: +5V power output.

Primarily intended as power for the motor 1 encoder, but it may be used to drive other hardware, up to 100 mA maximum.

Pin 6: Motor 2 Encoder Ch A-

Negative input of motor 2 encoder channel A differential pair. See pin 19 for functional description.

Pin 7: Motor 2 Encoder Ch B-

Negative input of motor 2 encoder channel B differential pair. See pin 20 for functional description.

Pin 8: Motor 1 Encoder Ch A-

Negative input of motor 1 encoder channel A differential pair. See pin 21 for functional description.

Pin 9: Motor 1 Encoder Ch B-

Negative input of motor 1 encoder channel B differential pair. See pin 22 for functional description.

Pin 11: General Purpose Input 2 / Encoder 2 Index Pulse-

Negative input of motor 2 index pulse differential pair or general purpose input 2. See pin 13 for functional description.

Pin 12: Start/Stop Sample Control Input.

When high, this input enables sampling and samples will be taken until the on-board buffer is full. When pulled low, sampling will stop. Samples are always completed, so that a full 8 byte sample is always buffered. This line is pulled up with an on-board 10Kohm resistor, so sampling is enabled when the input is left open. The first sample following resumption of sampling after stopping the sampling will not contain valid data, and must be read and discarded.

Pin 13: General purpose input bit 2 / Motor 2 index pulse input+.

Positive input of motor 2 index pulse differential pair or general purpose input 2. This may be used to sample external signals. The value of the bit will is included in the sampled data stream. This may be used to sample motor encoder index pulses or other events in order to synchronize the sample data with the event. The signal is latched high so that any high signal of 100 nanoseconds or longer during a sample interval will appear as a high level the following sample. This is intended for use with encoder index pulses.

Pin 14: Motor 1 Control.

If used, motor 1 should be connected between this pin and pin 14. The output voltage level is varied as commanded to control the variable voltage motor.

Pin 15: Motor Power Ground.

The external power supply ground for the motors should be connected to this pin.

Pin 16: Motor 1 Ground.

If used, motor 2 should be connected between this pin and pin 16.

Pin 17: Laser Control.

A 0-5V signal used to turn the laser on or off.

Pin 18: +5V power.

Primarily intended as power for the motor 2 encoder, but it may be used to drive other hardware, up to 100 milliamps maximum.

Pin 19: Motor 2 Encoder Channel A+

Positive input of motor 2 encoder channel A differential pair. If the motor control option is installed on the board, this input is decoded with pin 20 as a quadrature encoder signal from motor 2. The input should be a TTL-level signal and may switch at up to 1.5 MHz. The encoder positions are converted to 8 bit position values that ate included in the data stream. Each transition of pins 19 or 20 causes an up or down count in the position, so each quadrature cycle is effectively multiplied by 4 for the best possible resolution.

Pin 20: Motor 2 Encoder Channel B+.

Positive input of Motor 2 encoder channel B differential pair. If the motor control option is installed on the board, this input is decoded with pin 19 as a quadrature encoder signal from motor 2. The input should be a TTL-level signal and may switch at up to 1.5 MHz.

Pin 21: Motor 1 Encoder Channel A+.

Positive input of motor 1 encoder channel A differential pair. If the motor control option is installed on the board, this input is decoded with pin 22 as a quadrature encoder signal from motor 1. The input should be a TTL-level signal and may switch at up to 1.5 MHz. The encoder positions are converted to 8 bit position values that are included in the data stream. Each transition of pins 21 or 22 causes an up or down count in the position, so each quadrature cycle is effectively multiplied by 4 for the best possible resolution.

Pin 22: Motor 1 Encoder Channel B+.

Positive input of motor 1 encoder channel B differential pair. If the motor control option is installed on the board, this input is decoded with pin 21 as a quadrature encoder signal from motor 1. The input should be a TTL-level signal and may switch at up to 1.5 MHz.

Pin 23: General Purpose Input 1 / Encoder 1 Index Pulse-

Negative input of motor 1 index pulse differential pair or general purpose input 1. See pin 24 for functional description.

Pin 24: General purpose input bit 1 / Motor 1 index pulse input+.

Positive input of motor 1 index pulse differential pair or general purpose input 1. This may be used to sample external signals. The value of the bit will is included in the sampled data stream. This may be used to sample motor encoder index pulses or other events in order to synchronize the sample data with the event. The signal is latched high so that any high signal of 100 nanoseconds or longer during a sample interval will appear as a high level the following sample. This is intended for use with encoder index pulses.

Pin 25: General purpose input bit 3.

This may be used to sample external signals. The value of the bit will be inverted and inserted into the sample data stream. This may be used to sample events in order to synchronize the sample data with the event.

6.4 BNC Connector on AR4000 back panel

AR4000 laser rangefinders used with the 200 KHz PCI HSIF card have a special hardware configuration. The high-speed range frequency signal is transmitted on a dedicated coaxial cable to minimize the effects of interference. This coaxial cable attaches to the AR400 back panel at the mating BNC connector. If the system is ordered with an AC power supply, the coaxial cable is routed outside of the power supply while the dark-grey power and signal cable will be routed through the AC power supply box.

Figure 16 - Cabling of AR4000 with 200 KHz capability

7. HSIF Library

The AR4000 PCI HSIF card is accessed from user applications via *hsif.dll*, which is a dynamic-link library (DLL). This section describes the application program interface (API) exported by *hsif.dll*. For examples of how the API is used, see the source code included on the CD.

7.1 Sampling Architecture

The examples included on the CD demonstrate the necessary sequence of function calls required to configure and enable the AR4000 and HSIF card for capturing samples. The AR4000 HSIF card stores samples on the HSIF card in a buffer known as the L1 FIFO. These samples are read from the card by the driver and stored in the driver's much larger buffer known as the L2 FIFO. User applications read samples from the L2 FIFO.

Each time 2,048 samples have been stored in the L1 FIFO, the HSIF card issues an interrupt to the driver. The driver then reads 2,048 samples out of the L1 FIFO and stores them in the L2 FIFO. These samples are then available to be read by the user application.

At slow sampling rates, the delay between interrupts, and hence new samples, may be too long for an application. Therefore, the HSIF driver may be configured for polling mode. In this mode, the driver empties the L1 FIFO into the L2 FIFO each time the user application attempts to read samples or check the number of samples available.

Enabling polling has the downside that it requires more CPU processing. Therefore, polling at high sampling rates may lead to L1 FIFO and L2 FIFO overflows. It is recommended to only use polling with long sampling periods, such as 20 microseconds or greater.

7.2 HsifCalibrate() - Calibrate HSIF card

The PCI HSIF card must be calibrated to ensure accurate range readings in a given environment. This must be performed at least once after calling `HsifOpen`(). Calibrations should be performed periodically if the environmental conditions the HSIF card experiences vary. The procedure takes several seconds to perform.

```
HSIF_RESULT HsifCalibrate(
   HSIF_HANDLE hsifCard)
```

<u>Returns:</u>

HSIF_SUCCESS	Card was calibrated successfully
HSIF_FAIL	Failed to calibrate the card

7.3 HsifCalibrateEncoder() - Calibrate encoder

Calibrate the encoders to return the proper angle in radians based on an offset and the number of counts per revolution. The motor angle is calculated as:

$$2 * \pi * (\text{encoderCount} - \text{offset})/\text{countsPerRev}$$

```
HSIF_RESULT HsifCalibrateEncoder(
   HSIF_HANDLE hsifCard,
   DWORD encoder,
   DWORD offset,
   DWORD countsPerRev)
```

`encoder`	Encoder 1 or 2
`offset`	Encoder count offset
`countsPerRev`	Number of counts per revolution

<u>Returns:</u>

HSIF_SUCCESS	Calibrated the encoders successfully
HSIF_FAIL	Failed to calibrate the encoders.

7.4 HsifClearEncoder() - Clear motor position encoder accumulator

Clear either one or both encoders immediately or the next time the index pulse comes around.

```
HSIF_RESULT HsifClearEncoder(
   HSIF_HANDLE hsifCard,
   DWORD encoderMask,
   BOOL withIndex)
```

`encoderMask`	Set to the following value to select which encoder will be cleared: `ENCODER1` – clear encoder 1 `ENCODER2` – clear encoder 2 `ENCODER1 | ENCODER2` – clear both encoders
`withIndex`	Set to the following value to select when the encoder will be cleared: TRUE – clear the encoder when the index pulse is set FALSE – clear the encoder immediately

Returns:

HSIF_SUCCESS	Set the motor power successfully.
HSIF_FAIL	Failed to set the motor power.

7.5 HsifClearL1OverflowFlag() – Clear L1 FIFO overflow flag

This clears the hardware L1 FIFO overflow flag. This routine is automatically called by `HsifClearSampleBuffer()`.

```
HSIF_RESULT HsifClearL1OverflowFlag (
   HSIF_HANDLE hsifCard)
```

Returns:

HSIF_SUCCESS	L1 FIFO flag was successfully cleared.
HSIF_FAIL	Failed to clear L1 FIFO flag.

7.6 HsifClearSampleBuffer() - Clear sample buffer

Disables sampling, clears all samples that have been buffered into hardware and software, and clears the L1 and L2 FIFO overflow flags.

```
HSIF_RESULT HsifClearSampleBuffer(
```

```
HSIF_HANDLE hsifCard)
```

Returns:

HSIF_SUCCESS	Samples cleared successfully.
HSIF_FAIL	Sample clear failed

7.7 HsifClose() - Closing access to the card

Close the application's access to the PCI HSIF card. Calling the card with an invalid handle is not destructive.

```
HSIF_RESULT HsifClose(
   HSIF_HANDLE hsifCard)
```

Returns:

HSIF_SUCCESS	the HSIF card closed successfully and the resources are freed.
HSIF_FAIL	the HSIF card failed to close (because the specified card was not open)

7.8 HsifDataAvailable() - Get number of samples available

Return the number of samples available to be read.

```
HSIF_RESULT HsifDataAvailable(
   HSIF_HANDLE hsifCard,
   DWORD * numSamplesAvailable)
```

`numSamplesAvailable`	The number of samples available is returned through this pointer.

Returns:

HSIF_SUCCESS	Accessed number of samples available successfully.
HSIF_FAIL	Could not access number of samples available.

7.9 HsifDllInit() - Initialize the library

Initialize the library before using any of its functions.

```
BOOL HsifDllInit()
```

Returns:

TRUE	library initialization successful
FALSE	library failed to initialize

7.10 HsifGetBufferedSamples() - Get Buffered Samples

This function gets up to `numSamples` samples from the HSIF card or waits until `numSamples` have been received if wait is set to TRUE. The number of samples read is returned in `numRead`.

```
HSIF_RESULT HsifGetBufferedSamples (
   HSIF_HANDLE hsifCard,
   HSIF_SAMPLE * sampleBuf,
   DWORD numSamples,
   DWORD *numRead,
   BOOL wait)
```

`sampleBuf`	Storage buffer for the retrieved samples
`numSamples`	The number of samples to read from the card
`numRead`	The actual number of samples read.
`wait`	When TRUE, the function will block until `numSamples` have been read from the card. When FALSE, the function will return immediately with 0 to `numSamples` in the `sampleBuf` as indicated by numRead

Returns:

HSIF_SUCCESS	Read up to `numSamples` successfully
HSIF_OVERFLOW	L2 FIFO overflow occurred, samples lost.
HSIF_FAIL	Read failure. Bad HANDLE or card not responding.

Sample Contents:
HSIF_SAMPLE is defined as follows:

```
typedef struct {
     DWORD status;          index latches, temp, ambient,
                            amplitude, and other status info
     DWORD encoder1;        motor 1 position
     DWORD encoder2;        motor 2 position
     DWORD range;           uncalibrated range measurement
} HSIF_SAMPLE;
```

7.11 HsifGetCalValue() – Get the HSIF calibration value

Read the calibration value from the HSIF card that is used for converting the encoded raw range value into an absolute distance.

This function can be used for post-processing raw samples on slower CPU systems. The value returned by this function is intended to be loaded into the HSIF DLL via `HsifSetCalValue()` and the sampling configuration set `via` `HsifSetSamplePeriod()`. Afterwards, `HsifProcessSamples()` can be called to process the raw samples into calibrated distance samples.

```
HSIF_RESULT HsifGetCalValue (
   HSIF_HANDLE hsifCard,
   int * calVal)
```

`calVal`	calibration value the HSIF card uses to convert encoded raw range samples into absolute distances. The calibration value is returned via this pointer.

Returns:

HSIF_SUCCESS	Retrieved the calibration value successfully.
HSIF_FAIL	Failed to retrieve the calibration value.

7.12 HsifGetOverflowStatus() - Get overflow status

Return the L2 FIFO overflow status. If the overflow status is TRUE, the application isn't reading samples out of the driver buffer fast enough.

```
HSIF_RESULT HsifGetOverflowStatus(
   HSIF_HANDLE hsifCard,
   BOOL * overflowStatus)
```

`overflowStatus`	Logical and of the hardware and software overflow status.

Returns:

HSIF_SUCCESS	Returned the overflow status successfully.
HSIF_FAIL	Failed to return the overflow status.

7.13 HsifLaserOff() - Laser off

Disable the laser.

```
HSIF_RESULT HsifLaserOff(
   HSIF_HANDLE hsifCard)
```

Returns:

HSIF_SUCCESS	Laser on output successfully set
HSIF_FAIL	Laser on output set failed.

Note:
The current AR4000 sensors do not read this output from the PCI HSIF card. To turn the laser off, use the serial command “L”.

7.14 HsifLaserOn() - Laser on

Enable the laser.

```
HSIF_RESULT HsifLaserOn(
   HSIF_HANDLE hsifCard)
```

Returns:

HSIF_SUCCESS	Laser on output successfully set
HSIF_FAIL	Laser on output set failed.

Note:

The current AR4000 sensors do not read this output from the PCI HSIF card. To turn the laser on, use the serial command “H”.

7.15 HsifLoadCalibrationData() - Load calibration file

Load calibration file for the sensor/card pair. This file is shipped with each AR4000 and PCI HSIF card set. The calibration file is used by `HsifProcessSamples()` for generating calibrated distance measurements.

```
BOOL HsifLoadCalibrationData (
   HSIF_HANDLE hsifCard,
   LPCSTR filename)
```

`filename`	name location for calibration file

Returns:

HSIF_SUCCESS	Read successfully
HSIF_FAIL	File not found or File corrupted

7.16 HsifOpen() - Open communications with the card

Open communication with a PCI HSIF card.

```
HSIF_HANDLE HsifOpen(
    DWORD hsifNum,
    DWORD comHandle)
```

`hsifNum`	Open communication with a PCI HSIF card. hsifNum is a number 0 – 4, denoting the card number assignment. If hsifNum is 0, the first available HSIF device is returned as enumerated by Windows.
`comHandle`	If a serial port is dedicated to the AR4000 sensor, a handle to the opened COM device can be included here. The AR4000 will enable the PWM output and set the sample period and maximum range to their default values.

Returns:

HSIF_HANDLE	handle (0-4) to hsifCard resource
HSIF_INVALID_HANDLE	invalid or null handle

7.17 HsifProcessSamples() - Process samples

Generates calibrated range measurements from raw samples collected via `HsifGetBufferedSamples()` using the calibration file and calibration value determined through `HsifCalibrate()`.

This function also checks the L1 FIFO overflow status bit in each raw sample to determine if an L1 FIFO overflow occurred. If so, the driver is not retrieving samples fast enough from the HSIF card.

If you will be post-processing samples saved to disk during an earlier capture session, `HsifSetSamplePeriod()` and `HsifSetCalValue()` must be called before `HsifProcessSamples()`. These functions will restore the sampling environment.

```
HSIF_RESULT HsifProcessSamples (
   HSIF_HANDLE hsifCard,
   HSIF_SAMPLE * sampleBuf,
   HSIF_PROC_SAMPLE * procSampleBuf,
   DWORD numSamples)
```

`HSIF_SAMPLE * sampleBuf`	samples retrieved from HsifGetBufferedSamples call
`HSIF_PROC_SAMPLE * procSampleBuf`	buffer to store results after processing samples
`DWORD numSamples`	Number of samples to process

Returns:

HSIF_SUCCESS	Processed `numSamples` successfully
HSIF_OVERFLOW	Hardware L1 FIFO overflow bit was set in one or more of the samples
HSIF_FAIL	Read failure. Possible bad HANDLE or card not responding

Results Format:

`HsifProcessSamples` contains results in the following format

```
typedef struct {
     USHORT status;            FIFO and encoder index latches.
     double angle1;            motor 1 angle from offset in radians
     double angle2;            motor 2 angle from offset in radians
     double distance;          calibrated distance in inches
     double caltemp;           calibrated temperature
     double ambient;           ambient light
     double amplitude;         reflected signal strength
     BOOL timeout;             sample measurement timeout
```

```
    DWORD rawRange;                uncalibrated range
} HSIF_PROC_SAMPLE;
```

Notes:

A timeout will be set if no range reading from the AR4000 was detected before the sample period expires. This can occur if the range output cable from the AR4000 is disconnected or the target is beyond the maximum range.

7.18 HsifResetBoard() - Reset PCI HSIF card

This function performs a hardware reset of the PCI HSIF card. After calling this function, all initialization and sampling configuration setup must be performed again.

```
HSIF_RESULT HsifResetBoard(
   HSIF_HANDLE hsifCard)
```

Returns:

HSIF_SUCCESS	the HSIF card was successfully reset
HSIF_FAIL	the HSIF card reset failed

7.19 HsifSamplingDisable() - Sampling disable

Disable sampling of range data from the AR4000 sensor.

```
HSIF_RESULT HsifSamplingDisable(
   HSIF_HANDLE hsifCard)
```

Returns:

HSIF_SUCCESS	Sampling disabled successfully
HSIF_FAIL	Sampling disable failed

7.20 HsifSamplingEnable() - Sampling enable

Enable sampling of range data from the AR4000 sensor. This function must be called after `HsifClearSampleBuffer()`, since that function automatically disables sampling after clearing the sample buffers.

```
HSIF_RESULT HsifSamplingEnable(
   HSIF_HANDLE hsifCard)
```

Returns:

HSIF_SUCCESS	Sampling enabled successfully
HSIF_FAIL	Sampling enable failed.

7.21 HsifSamplingModeInit() - Sampling mode initialization

Set HSIF card to sampling mode. This must be done after a call to `HsifCalibrate()` to initialize the card for capturing samples.

```
HSIF_RESULT HsifSamplingModeInit(
   HSIF_HANDLE hsifCard)
```

Returns:

HSIF_SUCCESS	sampling mode entered successfully
HSIF_FAIL	unable to enter sampling mode

7.22 HsifSetCalValue() – Set the HSIF calibration value

Sets the calibration value used by `HsifProcessSamples()` to convert the encoded raw range value into an absolute distance. This function should be used with the value returned by `HsifGetCalValue()` as its argument when post-processing samples.

```
HSIF_RESULT HsifSetCalValue(
   HSIF_HANDLE hsifCard,
   int calValue)
```

`calVal`	calibration value the HSIF card uses to process encoded raw range samples into absolute distances.

Returns:

HSIF_SUCCESS	Set the calibration value successfully.
HSIF_FAIL	Failed to set the calibration value.

7.23 HsifSetMotorPower() - Set motor power

Set power for motors 1 and 2. The powers may be set from 0 to 255, corresponding to the range of zero power to maximum power.

```
HSIF_RESULT HsifSetMotorPower(
   HSIF_HANDLE hsifCard,
   DWORD motorNum,
   DWORD power)
```

`motorNum`	1 or 2 are valid motor numbers.
`Power`	0 (off) to 255 (maximum power)

Returns:

HSIF_SUCCESS	Set the motor power successfully.
HSIF_FAIL	Failed to set the motor power.

7.24 HsifSetPollMode() - Set poll mode

This function configures the driver to use polling mode to retrieve samples from the HSIF card. This is useful for slow sampling rates.

In interrupt mode, samples are offloaded into the L2 FIFO from the HSIF card when 2,048 samples have been captured into the L1 FIFO. Interrupt mode is best suited for fast sampling rates because it requires less processor overhead than polling mode.

When polling mode is enabled, all available samples are transferred from the L1 FIFO to the L2 FIFO whenever the user calls `HsifGetBufferedSamples`() or `HsifDataAvailable()`.

```
HSIF_RESULT HsifSetPollMode(
   HSIF_HANDLE hsifCard,
   BOOL mode)
```

`mode`	TRUE – enable polling mode FALSE – enable interrupt mode

Returns:

HSIF_SUCCESS	Poll mode was successfully set.
HSIF_FAIL	Failed setting poll mode

7.25 HsifSetSamplePeriod() - Set sample period and maximum range

Set the sample period and maximum range for the HSIF card.

```
HSIF_RESULT HsifSetSamplePeriod(
   HSIF_HANDLE hsifCard,
   HANDLE comHandle,
   DWORD range,
   DWORD samplePeriod)
```

`hsifCard`	The HSIF card's number from the above function
`comHandle`	Handle to a comport
`range`	Max measurement range, default is 650 inches.
`samplePeriod`	Period between samples in microseconds.

<u>Returns:</u>

HSIF_SUCCESS	Read up to `numSamples` successfully
HSIF_WARN_COM	Sample period updated internally, but an invalid com handle was specified and thus no serial communication with the AR4000 was attempted.
HSIF_FAIL	Read failure. Bad HANDLE or card not responding

8. Serial I/O Utilities

The Serial I/O Utilities comprise a set of functions for accessing the AR4000 sensor through the serial port. These functions are used internally by the HSIF Library for setting the resolution and range of the AR4000 sensor if the address of an open COMMINFO struct is passed to the `HsifOpen()` routine when opening access to the card, or during a call to `HsifSetSamplePeriod()`.

8.1 HsifAsciiReadLine() - Read a line of ASCII characters from the AR4000 sensor

Read a line of characters from the AR4000 sensor when the sensor is configured for ASCII output mode.

```
unsigned int HsifAsciiReadLine(
   COMMINFO *pCommInfo,
   char *buf,
   unsigned int nMaxLength,
   BOOL crlf,
   int timeout)
```

pCommInfo	pointer to a COMMINFO struct where relevant com port information is stored.
buf	storage space for characters read from the AR4000 sensor
nMaxLength	maximum number of characters that can be put in the buf storage space
crlf	TRUE - read a carriage return/line-feed pair as the end of line indicator. FALSE – read a line-feed as the end of line indicator. This should be set to TRUE for AR4000 sensor.
timeout	Number of milliseconds to wait before returning from the call if no data is detected.

Returns:

number of bytes read	If read was successful.
ASCII_TIMEOUT	If read times out.

8.2 HsifAsciiRead() - Read ASCII characters from the AR4000 sensor

Reads *numBytes* characters from the AR4000 sensor into *buf* when the sensor is configured for ASCII output mode.

```
unsigned int HsifAsciiRead(
   COMMINFO *pCommInfo,
   char *buf,
   int numBytes)
```

pCommInfo	pointer to a COMMINFO struct where relevant com port information is stored.
buf	storage space for characters read from the AR4000 sensor
numBytes	maximum number of characters that can be put in the buf storage space

Returns:

number of bytes read	If read was successful.
ASCII_TIMEOUT	If read times out.

8.3 HsifBinaryReadBytes() - Read binary data bytes from the AR4000 sensor

Reads up to *nMaxLength* data bytes into *buf* from the AR4000 sensor when it is configured in binary output mode.

```
unsigned int HsifBinaryReadBytes(
   COMMINFO *pCommInfo,
   char *buf,
   unsigned int nMaxLength,
   int timeout)
```

pCommInfo	pointer to a COMMINFO struct where relevant com port information is stored.
buf	storage space for characters read from the AR4000 sensor
nMaxLength	maximum number of characters that can be put in the *buf* storage space
timeout	Number of milliseconds to wait before returning from the call if no data is detected.

Returns:

number of bytes read	If read was successful.
BINARY_TIMEOUT	If read times out.

8.4 HsifClosePort() - Close com port

Close com port.

```
BOOL HsifClosePort(
   COMMINFO *pCommInfo)
```

pCommInfo	pointer to a COMMINFO struct where relevant com port information is stored.

Returns:

TRUE	Com port close was successful
FALSE	Com port close failed

8.5 HsifOpenPort() - Open a com port to communicate with an AR4000 sensor

Open a com port at the baud rate specified.

```
BOOL HsifOpenPort(
   COMMINFO *pCommInfo,
   int baudrate,
   int Portnum)
```

`pCommInfo`	pointer to a COMMINFO struct where relevant com port information is stored
`baudrate`	baud rate the sensor is configured for. Normally this is 9600.
`Portnum`	com port number to open

<u>Returns:</u>
TRUE Com port was successfully opened
FALSE Com port open failed.

8.6 HsifPurgePort() - Purge port

Clear out pending transmit and receive data

```
BOOL HsifPurgePort(
   COMMINFO *pCommInfo,
   int flags)
```

`pCommInfo`	pointer to a COMMINFO struct where relevant com port information is stored.
`flags`	Possible settings: PURGE_TX - clear transmit buffer PURGE_RX - clear receive buffer These values can be "bitwise OR'd" for simultaneous operation, i.e. PURGE_TX	PURGE_RX

<u>Returns:</u>
TRUE Purge was successful
FALSE Purge failed

8.7 HsifSendStr() - Send serial data to the AR4000 sensor

Send a string of characters to the AR4000 sensor.

```
BOOL HsifSendStr(
   COMMINFO *pCommInfo,
   char str[])
```

`pCommInfo`	pointer to a COMMINFO struct where relevant com port information is stored.
`str`	null-terminated string of characters to transmit to the AR4000 sensor

Returns:

TRUE	String was transmitted successfully
FALSE	String transmit failed.

9. Sample Formats

The interface board collects 16 byte samples in a sequential stream which are read as samples with function `HsifGetBufferedSamples()`. If an HSIF card L1 FIFO buffer overflow occurs, the board will always drop complete samples, so that synchronization is not lost. Similarly, if the user application does not retrieve samples fast enough from the driver, and an L2 FIFO buffer overflow occurs, complete samples will be dropped and synchronization will not be lost. If `HsifClearSampleBuffer()` is called, these overflow conditions will be cleared.

In general the values for amplitude and ambient light level, will correspond closely to the values from the 4000's serial interface, with the ASCII format serial data being 4 times the HSIF values for amplitude and ambient light. However, the values will not match exactly, and the calibration software supplied for use with the HSIF card must be used with the values obtained from the HSIF card, not serial data. The temperature and range have different scale factors from the serial data and must be scaled using algorithms found in the software supplied with the interface.

The raw samples gathered through `HsifGetBufferedSamples()` must to be processed via a call to `HsifProcessSamples()` to convert the encoded raw range values into absolute distances and calibrated distances. Both the raw sample data format (HSIF_SAMPLE) and the processed sample data format (HSIF_PROC_SAMPLE) are described below:

9.1 Raw sample format – HSIF_SAMPLE

The data structure that contains each sample returned by a call to `HsifGetBufferedSamples()` is:

```
typedef struct
{
      DWORD status;     // FIFO stat, inputs, temp, amb, amp
      DWORD encoder1;   // Motor 1 encoder counts
      DWORD encoder2;   // Motor 2 encoder counts
      DWORD range;      // Encoded raw range
} HSIF_SAMPLE;
```

Each elements of the HSIF_SAMPLE data structure is described in Table 6.

Element	Bit #	Description
status	0	Hardware Buffer Overflow Indicator
	1	Input 3
	2	Motor 1 Encoder Index / Input 1
	3	Motor 2 Encoder Index / Input 2
	7-4	Sample Count 0-15
	15-8	8-bit Sensor Internal Temperature
	23-16	8-bit Ambient Light Sample
	31-24	8-bit Amplitude Sample
encoder1	31-0	32-bit Motor 1 Encoder Position
encoder2	31-0	32-bit Motor 2 Encoder Position
range	31-0	32-bit Encoded Range

Table 6 – HSIF_SAMPLE

status: Status consists of the following sub-fields:

Hardware Buffer overflow indicator: 1 bit indicating whether a HSIF card L1 FIFO buffer overflow occurred and 1 or more samples were lost just prior to the first sample in which the flag is set. Once an overflow occurs, this bit will stay set until a `HsifClearOverflow()`, `HsifClearSampleBuffer()`, or `HsifResetBoard()` command is given or a power cycle occurs. Samples with the overflow flag set may contain inaccurate range data and should be discarded. Since the overflow flag is stored with the buffered data, resetting the flag will not become evident in the data until the data in the buffer has been read, or the buffer has been cleared with a board reset command. Note that if the buffer is full when the `HsifClearOverflow()` command is given, it will simply be set again immediately.

Inputs 1, 2, 3: Three general purpose input lines, CMOS logic levels. These may be used to determine the exact times of external events relative to the samples taken.

Inputs 1 and 2 are set on the positive rising edge of signals input to these pins. After the next sample is written, the inputs are automatically cleared. Input 3 is a general purpose, level-sensed input.

Motor 1, 2 Encoder Index: Indicates when a motor has completed a revolution. When Inputs 1 and 2 are used to indicate motor encoder index they will be unavailable for use as general purpose inputs.

Sample Count 0-15: This is a modulo-16 count that is incremented each time a new sample is stored in the L1 FIFO. This count is used internally by the HSIF driver to determine the number of samples available in the hardware buffer for retrieval when polling mode is enabled.

Sensor Internal temperature: 8-bit sample of the AR4000 internal temperature. The temperature is sampled in the first 5 microseconds of the data sample interval.

Ambient Light: 8-bit sample of the AR4000 ambient light output. The sample represents the ambient or background light sensed by the detector. It will also register the light transmitted by the sensor, so changing range signal strengths will affect this reading somewhat. The ambient light sample is taken in the first 5 microseconds of the data sample interval.

Amplitude: 8-bit sample of the AR4000 logarithmic signal strength output. The sample represents the amplitude of the modulated signal sensed by the detector. The amplitude sample is taken in the first 5 microseconds of the data sample interval.

encoder1: 32-bit unsigned sample of the position of motor encoder 1, if the motor control option is installed and a motor encoder is attached to the P2 motor 1 encoder inputs. The position will wrap to 0 on the 2^{32} count.

encoder2: 32-bit unsigned sample of the position of motor encoder 2, if the motor control option is installed and a motor encoder is attached to the P2 motor 2 encoder inputs. The position will wrap to 0 on the 2^{32} count.

range: 32-bit encoded range value which must be processed through `HsifProcessSamples()` to generate an absolute range value and calibrated range value. The absolute range value is proportional to the distance to the object being ranged, within the uncalibrated linearity of the AR4000.

9.2 Processed sample format – HSIF_PROC_SAMPLE

The processed sample returned via a call `to HsifProcessSamples()` is shown below:

```
typedef struct
{
  USHORT    status;     // First 16-bits of HSIF_SAMPLE status
  double    angle1;     // Motor 1 angle in radians
  double    angle2;     // Motor 2 angle in radians
  double    distance;   // Calibrated distance
  double    caltemp;    // Calibrated temperature, degrees F.
  double    ambient;    // Background illumination  same as HSIF_SAMPLE
  double    amplitude;  // Reflected signal strength
  BOOL      timeout;    // Range measurement timeout
  DWORD     rawRange;   // raw range before calibration lookup
} HSIF_PROC_SAMPLE;
```

status: 16-bit unsigned value that repeats bits 15-0 of the status register described in the HSIF_SAMPLE data structure above.

angle1: 64-bit double that indicates motor1's angle in radians minus the offset set via `HsifSetEncoderCalibration()`.

angle2: 64-bit double that indicates motor 2's angle in radians minus the offset set via `HsifSetEncoderCalibration()`.

distance: 64-bit calibrated target distance calculated via the lookup calibration table, temperature, ambient, and amplitude readings. This is the value that should normally be used as an indication of target distance.

caltemp: 64-bit temperature in degrees Fahrenheit.

ambient: 64-bit value indicating the background illumination.

amplitude: 64-bit value indicating the reflected signal strength.

timeout: 32-bit value that indicates the distance measurement timed out. When this value is non-zero the distance sample should be ignored.

rawRange: 32-bit unsigned encoded raw range value that is converted via the calibration value into an absolute distance value.

10. Interface Installation and Checkout

To install the AR4000 PCI High Speed Interface board, first install the computer drivers from the supplied CD according to the directions in section 2. When complete, physically install the PCI board into an available PCI slot in your computer. Attach the AR4000 Power and Signal cable to the 9 pin connector (P1). Turn on the computer power. Check out the operation of the AR4000 as described in the Initial Checkout section.

10.1 Diagnostics

Install the PCI High Speed Interface in a bus slot, connect the sensor to the HSIF board and to a serial port on the computer. Be sure that power is being supplied to the AR4000 sensor.

If the sensor's LED does not come on, check the connection of the sensor to the interface. The serial connection to the sensor may be tested separately using a program such as the Windows terminal to observe sensor output and send commands. If the sensor does not respond to serial communications, check the serial port connection.

After installing the board and connecting the AR4000 sensor, run the *hsiftest* example supplied with the board, following the instructions at the beginning of this manual.

If the motor control option is not installed, the encoder tests will not succeed. If you have not connected the input lines and external sample control line to 0/5 volt signals, the tests of those lines will not succeed. All other tests should succeed.

If one or more of the HSIF card tests fail, check that the serial port the sensor is connected to is the port number you have given to the *hsiftest* software. Also, check that the AR4000 sensor's serial port is configured for 9600 baud.

If the sensor stability tests fail, check that the laser comes on during those tests and that the sensor is pointed a white target 1 to 2 yards from the sensor.