INSTRUCTION MANUAL

Sensoray Models 418/419
ISA Smart A/D™ Cards

Revised March 11, 2004

Sensoray Co., Inc.
7313 SW Tech Center Dr., Tigard, Oregon 97223
voice: 503.684.8005, fax: 503.684.8164, e-mail: sales@sensoray.com
www.sensoray.com

Table of Contents

1 Prolog

1.1 Limited Warranty

Sensoray Company, Incorporated (Sensoray) warrants the Model 418/419 hardware to be free from defects in material and workmanship and perform to applicable published Sensoray specifications for two years from the date of shipment to purchaser. Sensoray will, at its option, repair or replace equipment that proves to be defective during the warranty period. This warranty includes parts and labor.

The warranty provided herein does not cover equipment subjected to abuse, misuse, accident, alteration, neglect, or unauthorized repair or installation. Sensoray shall have the right of final determination as to the existence and cause of defect.

As for items repaired or replaced under warranty, the warranty shall continue in effect for the remainder of the original warranty period, or for ninety days following date of shipment by Sensoray of the repaired or replaced part, whichever period is longer.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. Sensoray will pay the shipping costs of returning to the owner parts that are covered by warranty.

Sensoray believes that the information in this manual is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, Sensoray reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult Sensoray if errors are suspected. In no event shall Sensoray be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, SENSORAY MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OF FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF SENSORAY SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. SENSORAY WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

1.2 Special Handling Instructions

The model 418/419 Smart A/D™ circuit boards contain CMOS circuitry that is sensitive to Electrostatic Discharge (ESD).

Special care should be taken in handling, transporting, and installing the Smart A/D™ board to prevent ESD damage. In particular:

- Do not remove the circuit board from its protective anti-static bag until you are ready to configure the board for installation.
- Handle the circuit board only at grounded, ESD protected stations.
- Remove power from the target system before installing or removing the circuit board.

2 Introduction

2.1 Functional Description

The Sensoray model 418/419 Smart A/D™ boards provides eight/sixteen independent sensor channels that interface process sensors directly to the ISA bus.

An onboard microcomputer continuously scans the sensor channels. As each channel is scanned, the sensor is excited, digitized, linearized, converted to engineering units appropriate for the sensor type, and stored in onboard memory. The freshest sensor data from all channels are always available for instant access.

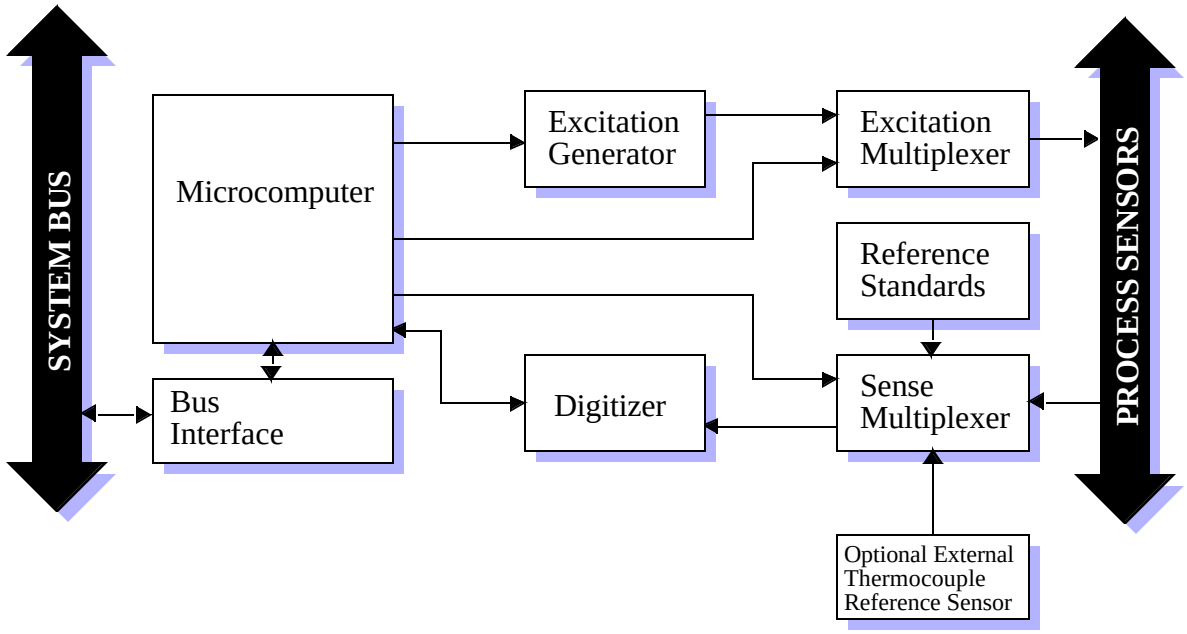
Standard 40-pin headers are provided for connecting the sensor channels to external circuitry. Optional mating termination boards and cables are available to facilitate connections to application sensors.

Smart A/D™ features include:

- Embedded processor off-loads the most demanding data acquisition functions from the host.
- Each sensor channel may be independently configured, via software, to accept thermocouples, RTD's, strain gages, thermistors, resistors, 4-to-20 mA current loops, or DC voltage inputs.
- Sensor channels employ fully differential sense inputs to help eliminate ground loops.
- All sense inputs are protected from high differential and common mode voltages.
- Pulsed excitation is provided for passive sensors, such as thermistors, resistors, RTD's, and strain gages. Pulsed excitation minimizes sensor self-heating effects and helps to reduce system power consumption.
- Sensor excitation signals are electrically separate from the sense inputs so that high accuracy three-wire or four-wire circuits may be implemented.
- Automatic cold junction compensation is provided for thermocouples. The remote compensation sensor—present on all Smart A/D™ termination boards—connects to a dedicated measurement channel so that all process sensor channels remain available for application use.
- Programmable alarm limits for each channel. A hardware Alarm status flag provides instant access to limit violation detection without cumbersome polling and evaluation of sensor data for limit violation detection.
- Fully electronic calibration—no trimpots to adjust. Highly stable internal standards provide excellent measurement accuracy without the need for frequent calibrations.

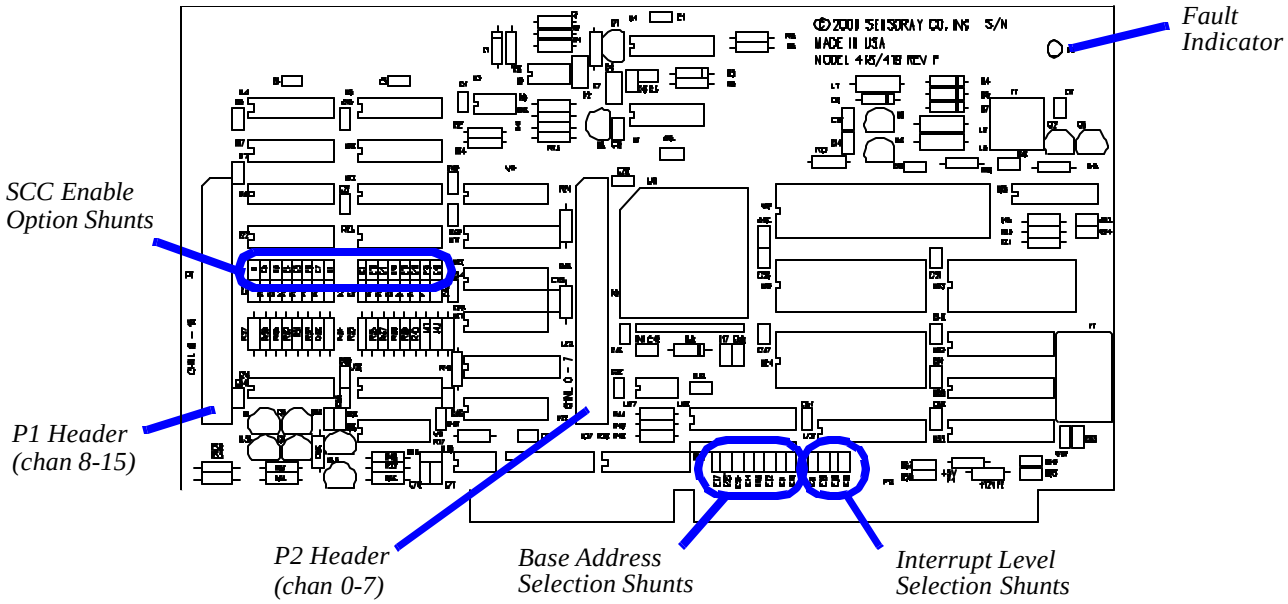
2.2 Block Diagram

Figure 1: Model 418/419 functional block diagram



2.3 Board Layout

Figure 2: Board Layout



2.4 Reset State

At power-up, or after a soft reset (see Section 5.1) or a system hard reset, the Smart A/D™ board is in its default state as follows:

- Standard speed mode (22ms/channel) is selected.
- 60Hz rejection mode is active.
- All channels are enabled for scanning.
- Sensor data values are reset to zero. Sensor data is not available immediately after a board reset. The maximum elapsed time that must pass before new sensor data becomes available is equal to 35 channel slot times. Details of channel slot timing can be found in Section 8.1
- Channels are configured for the 5-volt input range, with 500µV resolution.
- Software filter constants are set to zero (disabled).
- Alarm threshold limit values are indeterminate.
- All channel alarms are disabled.
- All channels are programmed to fail high upon open-sensor detection.

2.5 Fault Indicator

A red light-emitting diode (LED), located near the edge of the Smart A/D™ board, is used as a status indicator. This LED is energized under either of the following circumstances:

- A board reset is in progress.
- A board fault has been detected.

When the Smart A/D™ board is functioning normally, the LED is energized for approximately one-half second in response to a board reset, either hard or soft. In the case of power-up and hard resets, this time period may be extended as the result of a stretched hardware reset signal on the system bus.

If the LED is persistently energized, there may be a hardware fault on the A/D board. In this case, be sure to check all sensor connections, board orientation and seating, etc. before concluding that the board has a fault.

3 Hardware Configuration

The Smart A/D™ board requires the configuration of switches and programming shunts to select hardware options. These options must be configured before installing the Smart A/D™ board into the target system.

Programming shunts are configured by installing or removing shorting shunt plugs at various locations on the Smart A/D™ circuit board. A supply of programming shunts, of sufficient number to program all board options, is included with the Smart A/D™ board.

Table 1: default configuration from factory

Option	Default
Base address	0x02B2
Channel SCC's	All disabled

3.1 Base Address

The Smart A/D™ board may be mapped to any even base address in the ISA bus I/O space from 0x0200 to 0x03FF. The board occupies two consecutive addresses. To avoid address conflicts, you must not map any other devices into the address space occupied by the Smart A/D™.

Programming shunts E20 through E27, located on the board as shown in Figure 2, select the base address for the board. These shunts are factory set to locate the board at address 0x02B2. You must change the shunt positions if you require a different base address. Table 2 shows the shunt settings for all possible addresses.

Table 2: Address shunts: ● =installed, ❍ =removed.

Nibble Value	High Byte	Low Byte						
		High Nibble				Low Nibble		
	E27	E26	E25	E24	E23	E22	E21	E20
0		●	●	●	●	●	●	●
1		●	●	●	❍	●	●	
2	●	●	●	❍	●	●	●	❍
3	❍	●	●	❍	❍	●	●	
4		●	❍	●	●	●	❍	●
5		●	❍	●	❍	●	❍	
6		●	❍	❍	●	●	❍	❍
7		●	❍	❍	❍	●	❍	
8		❍	●	●	●	❍	●	●

Table 2: Address shunts: ● =installed, ❍ =removed.

Nibble Value	High Byte	Low Byte						
		High Nibble				Low Nibble		
	E27	E26	E25	E24	E23	E22	E21	E20
9		❍	●	●	❍	❍	●	
A		❍	●	❍	●	❍	●	❍
B		❍	●	❍	❍	❍	●	
C		❍	❍	●	●	❍	❍	●
D		❍	❍	●	❍	❍	❍	
E		❍	❍	❍	●	❍	❍	❍
F		❍	❍	❍	❍	❍	❍	

For example: select 0x0390 as the board's base address. In this case, the high address byte is "3," so shunt E27 must be removed. The next nibble (high nibble of low byte) is "9," so shunts E25 and E24 must be installed, and E26 and E23 must be removed. The next nibble (low nibble of low byte) is "0," so shunts E22, E21 and E20 must all be installed.

3.2 Signal Conditioning Circuit (SCC)

Each channel is provided with a signal conditioning circuit (SCC) which may be inserted into the sense signal path. The SCC performs these functions:

- **Common-mode tie-down.** This function helps to prevent the common mode voltage (CMV) from exceeding the input CMV range of the Smart A/D™ measurement section. This function is required when a sensor channel is driven from an isolated source, such as a battery, isolated power supply, or thermocouple.
- **Open-sensor detect.** If either of the two sense signals should become disconnected from the source, this function forces a differential voltage of 700mV, minimum, to appear across the sense inputs. This function is useful when a sensor is used in a control loop.
- **RF shunt.** This function, which shunts RF noise to ground, is essential when connecting to isolated sources, such as thermocouples.

3.2.1 Side Effects

Sense input impedance is reduced when a channel's SCC is enabled. See "General Specifications" on page35 for details.

3.2.2 Recommended Settings

The decision of whether or not to enable a channel SCC is influenced primarily by the type of sensor to be interfaced, and, in the case of active sources, whether or not the source is ground-referenced. Table 3 lists the recommendations for enabling vs. disabling the SCC:

Table 3: Recommended signal conditioner settings

Sensor Type	SCC
Ungrounded Thermocouple	Enabled
Isolated DC Voltage Source	
Grounded Thermocouple	Disabled
Ground-referenced DC Voltage Source	
4-to-20 mA loop	
Resistor	
RTD	
Strain/Pressure Gage	
Thermistor	

3.2.3 Shunt Mapping

Shunts E1 through E16, located on the circuit board as shown in Figure 2, control the enabling of the channel SCCs. Each channel is affiliated with one shunt. To enable a channel's SCC, you must install the affiliated shunt. Similarly, to disable the SCC, you must remove the shunt.

Table 4 and Table 5 shows the mapping between channels and affiliated shunts for SCC enables.

Table 4: SCC shunt mapping for channels 0-7

Channel	0	1	2	3	4	5	6	7
Shunt	E1	E2	E2	E3	E4	E6	E7	E8

Table 5: SCC shunt mapping for channels 8-15 (419 only)

Channel	8	9	10	11	12	13	14	15
Shunt	E9	E10	E11	E12	E13	E14	E15	E16

For example, install shunt E2 to enable the SCC for sensor channel 2.

3.3 Interrupts

The board may be configured to interrupt the ISA master if a channel alarm limit violation occurs. Alarm limits can be implemented for each channel as described in Section 6.5.

As detailed in Table 6, program shunts E29, E30, E19 and E28 may be used to enable the interrupt and to select the alarm interrupt level.

Table 6: Interrupt shunts

Level	IRQ 2	IRQ 3	IRQ 4	IRQ 5
Shunt	E28	E19	E30	E29

Install a shunt at the position corresponding to the desired interrupt level, or leave all shunts removed if alarm interrupts are not needed. If interrupts are used, make sure only one interrupt shunt is installed and all others are removed.

4 Connections

4.1 Overview

Sensors are connected to the Smart A/D™ by means of 40-pin headers P1 and P2, located on the circuit board as shown in Figure 2.

4.1.1 Termination Boards

It is recommended that sensors be indirectly connected to the Smart A/D™ board by means of optional screw termination boards (TB), such as Sensoray models 7409TB or 7409TC. These TBs have calibrated reference sensors for thermocouple compensation and are designed to mate directly to all Smart A/D™ boards.

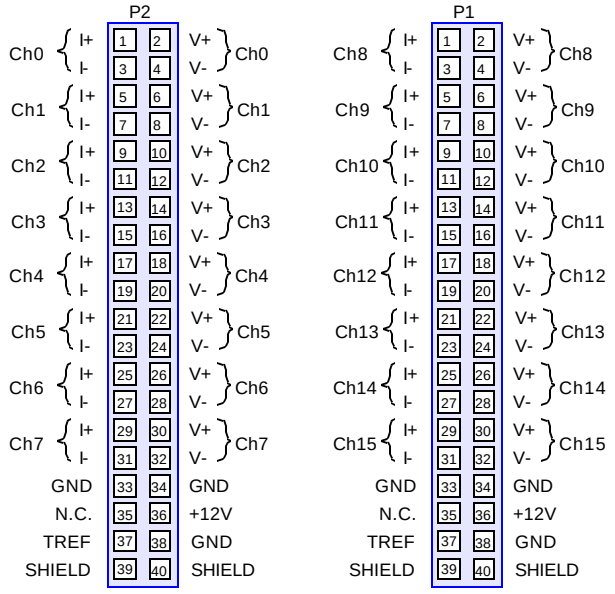
4.1.2 Sensor Hot Insertion

Sensors may be electrically connected and disconnected from the Smart A/D™ or termination board without removing the Smart A/D™ board or power from the backplane.

4.2 Connector Pinouts

The pinouts of headers P1 and P2 are shown in Figure 3.

Figure 3: Header P1 and P2 pinouts



4.2.1 Reference Junction Sensor

Three of the signals—+12V, GND and TREF— shown at the bottom of connector P1 are used for interfacing the Smart A/D™ to a remote thermocouple reference-junction temperature sensor. One such sensor is provided on all models of Sensoray Smart A/D™ termination boards.

The +12V and GND signals provide excitation to the reference sensor. The TREF connection provides a path from the sensor output to a dedicated input channel on the Smart A/D™ board.

If you are using thermocouples in your application and you will be implementing your own termination system, you must provide your own remote temperature sensor. The sensor, which produces an output of 10mV/°K, must have its output signal routed to the TREF pin.

4.3 Sensor Channels

Each sensor channel provides five signals for interfacing to a sensor. A sensor can have as many as five connections to a channel or as few as two.

4.3.1 Sense Terminals

Two of the channel signals, designated V+ and V-, are universally used for all sensor types. These two signals are, respectively, the positive and negative differential voltage sense inputs.

Since the sense inputs accept differential signals, it is not required for either of the two input signals to be at ground potential. The Smart A/D™ digitizer measures only the difference voltage across the two sense inputs.

Due to common-mode voltage (CMV) input constraints, however, you must ensure that neither sense input, with respect to system ground, exceeds the maximum input CMV permitted by the Smart A/D™.

Small excursions beyond the CMV limit may degrade measurement accuracy on the offending channel, while large excursions may cause measurement errors on other sensor channels, or worse yet, damage the board.

See “Sensor Specifications” on page36 for input CMV limit values.

4.3.2 Excitation Terminals

Passive sensors require connection to the I+ and I- signals. The I+ and I- signals supply positive and negative excitation, respectively, to passive sensors.

Different classes of excitation are applied to sensors; the class of excitation that is applied depends on the type of sensor to be measured.

Constant current excitation is used for high-accuracy, low resistance measurements. A current limited, constant voltage excitation is applied when measuring higher resistance values. Finally, a 10 Volt, high current excitation is applied to strain and pressure gages.

4.3.3 Shield Terminal

The fifth lead—named *S* for shield—connects to the cable shield, if present. The S signal is internally connected to the backplane ground. To avoid ground loops, the cable shield should never be connected to anything at the sensor end of the cable, nor should it be connected to another ground point.

A shield conductor is never absolutely required, but is sometimes an unavoidable noise-abatement measure.

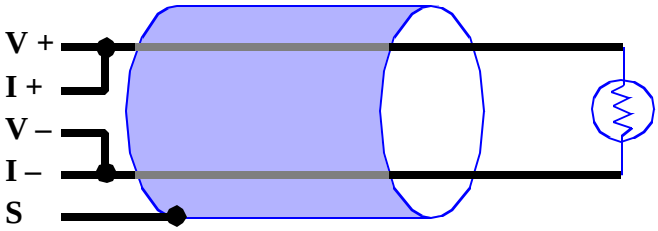
4.4 RTDs, Resistors and Thermistors

RTDs, resistors and thermistors can be connected to the TB in any of three possible configurations: two-wire, three-wire, and four-wire. In the following discussion, *sensor* is used interchangeably with *RTD*, *resistor* and *thermistor*.

4.4.1 Two-wire Circuit

The simplest configuration is the two-wire circuit. As the name implies, this configuration requires only two wires. The V+ and I+ terminals are shorted together at the TB, as are the V- and I- terminals.

Figure 4: Two-wire connection.

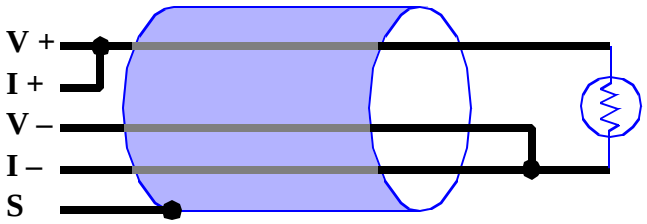


This circuit conserves wire, but it causes measurement error due to the voltage drop between the TB and sensor. There are two potential solutions for this problem: use short wires, or use more than two wires.

4.4.2 Three-wire Circuit

By using three wires, it is possible to reduce cable loss errors by 50 percent. Instead of shorting V- and I- together at the TB, discrete conductors are run from each of these terminals to the sensor. The wires are then shorted together at the sensor.

Figure 5: Three-wire connection

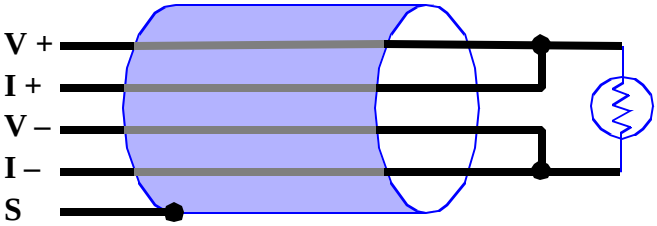


In this situation, the high impedance V- terminal detects the voltage at the sensor before any voltage drop can occur.

4.4.3 Four-wire circuit

Unlike the three-wire circuit, which exhibits half the cable losses of an equivalent two-wire circuit, a four-wire circuit completely eliminates cable losses.

Figure 6: Four-wire connection



Like the three-wire circuit, separate conductors are run from the V- and I- terminals to the sensor. In addition, separate conductors are run from V+ and I+ to the sensor, where they are tied together. This circuit eliminates cable loss effects from both the V+ and V- lines.

4.4.4 Recommended Practice

If sensor field wiring is exposed to electrical noise (i.e., the cable run is long or is close to noisy conductors) you should consider using shielded cable. The cable shield must be connected only to the S terminal on the TB and left unconnected at the sensor end of the cable.

The four-wire circuit should be used when accuracy is critical. This is especially important when making precision low-resistance value measurements, such as when using RTDs.

Thermistors have much higher resistance values than RTDs over most of their operating range. Consequently, the two-wire circuit may be used if your sensor will be operating exclusively at high resistance values. However,

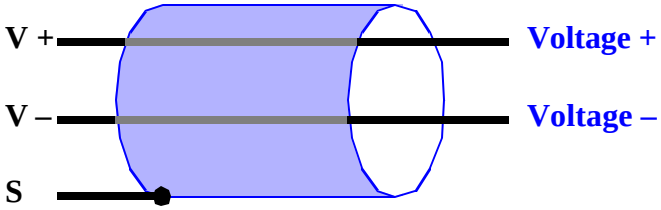
if your thermistor will be operating at lower resistance values, you should consider implementing a three-wire or four-wire circuit to reduce or eliminate cable-loss errors.

4.5 DC Voltage Sources

DC voltage sources are connected directly to the V+ and V- terminals. DC voltage sources should never be connected to the I+ or I- terminals.

Since all voltage input ranges are bipolar, DC voltages may be connected in either signal polarity. Although the diagram below shows Voltage+ connected to V+ and Voltage- connected to V-, there is no reason that Voltage+ must be positive with respect to Voltage-.

Figure 7: DC voltage connection



4.5.1 Recommended Practice

In order to assure accurate DC voltage measurements, high common-mode voltages (CMV) must not be permitted to appear on V+ or V-.

If your signal source is isolated (i.e., sourced from a battery or isolated power supply), you can connect V+ or V- to either S or the backplane power supply return. This connection can be implemented as a direct short or can be made through, for example, a 10KΩ resistor. There should be no significant DC current flowing through this connection; its purpose is to limit the common-mode voltage at the sense inputs.

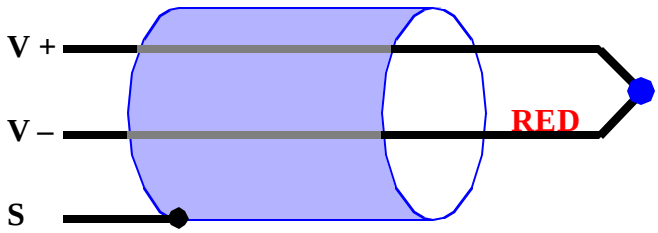
In the case of low source impedance voltage sources, another way to limit CMV is to install the channel signal conditioning circuit as described in Section 3.2.

4.6 Thermocouples

Thermocouple (TC) signals are connected directly to the V+ and V- terminals. TCs should never be connected to the I+ or I- terminals.

TC wires are color coded to indicate polarity. The red thermocouple wire is always negative, by convention. The positive TC wire should always be connected to the V+ terminal, and the negative wire should always be connected to the V- terminal.

Figure 8: Thermocouple connection



4.6.1 Recommended Practice

A TC reference-junction compensation sensor is required in order to obtain accurate TC measurements. This sensor is present on all Sensoray Smart A/D™ termination boards.

If you will be using a TB of your own design, you should physically arrange the compensation sensor so that it is thermally coupled to the TC reference junctions.

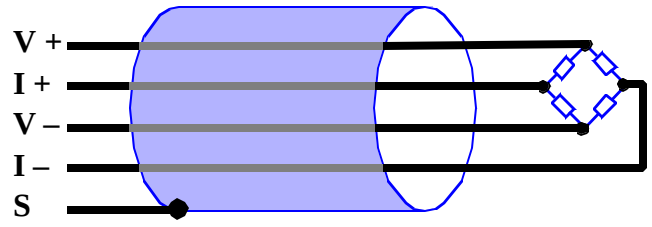
Significant measurement errors can occur if you allow the TB to be exposed to thermal transients, such as air flows from cooling fans or ambient breezes. For best results, the TB should be located within a protective enclosure after sensor terminations have been made.

If your TC is ungrounded, always enable the channel signal conditioning circuit. See Section 3.2 for details.

4.7 Strain and Pressure Gages

In a typical strain/pressure gage sensor, four wires are used to connect the gage to the sensor channel. Two of the wires supply gage excitation, while the other two wires transport the gage output signal to the Smart A/D™ measurement circuitry.

Figure 9: 4-wire gage connection



4.7.1 Recommended Practice

Due to the high measurement gain used, strain and pressure gages should use shielded cable. The shield should be connected only to the channel S terminal.

If you are using a six-wire gage, you should connect the gage's excitation source and excitation sense leads together at the I+ and I- terminals. If this is not practical (i.e., you are using cable that has only four couductors),

you may either connect the excitation source and excitation sense leads together at the gage or you may leave the excitation sense leads completely disconnected. Note that remote sensing of the excitation signals is not supported.

The gage impedance seen by the Smart A/D™ excitation source must be at least 120Ω to prevent automatic current limiting. If the input impedance is less than 120Ω, a resistor must be inserted in series with the gage excitation terminals to prevent excessive excitation circuit loading. Note that this will alter the mV/V rating of the sensor.

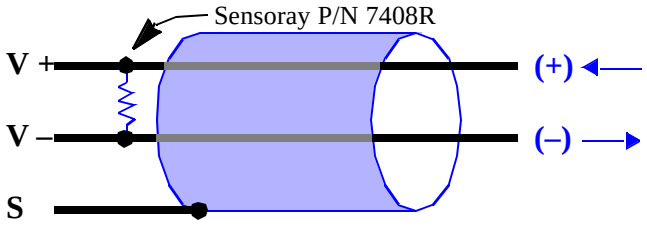
Gage output voltage must fall between -500mV and +500mV under all load conditions, including any offset caused by bridge imbalance. If the gage output voltage, at 10V excitation, might exceed these input limits, it may be necessary to configure the sensor channel for voltage input and supply external excitation to the gage.

4.8 4-to-20 mA Current Loops

All sensor channels support 4-to-20 mA current loop inputs, subject to two constraints:

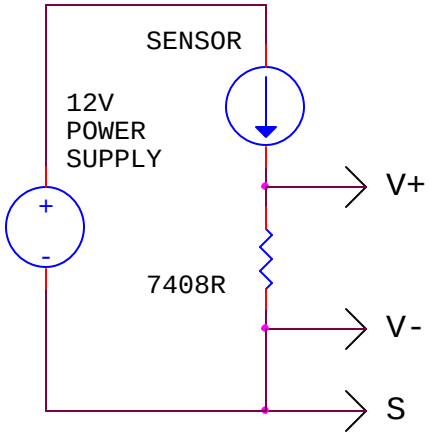
1. A 250Ω, 0.01% resistor must be installed as shown in the following diagram. This resistor, which converts the loop current to a voltage, is available as an option for the Smart A/D™; order Sensoray part number 7408R.
2. Only the *grounded* end of the current loop may be connected to a sensor channel. This ensures that the common-mode voltage will not exceed Smart A/D™ input limits.

Figure 10: 4-20 mA current loop connection



It is important to note that the Smart A/D™ does not provide excitation for current loops. Current loops must be energized from an external power source, such as the isolated 12V power supply in Figure 11.

Figure 11: Typical 4-20mA current loop connection



If the external power source is not isolated (i.e., its negative output terminal is electrically connected to the backplane power supply return), you should not connect to the S terminal as shown in the above diagram, as a potential ground loop would be created. In such a case, leave the S terminal disconnected and connect to the V+ and V- terminals only.

5 Programming

Smart A/D™ programming is accomplished by means of built-in commands. Commands may be sent from any ISA bus master (referred to as the *host*, or *client*) to the Smart A/D™. Some commands cause the Smart A/D™ to respond by returning data to the host.

This chapter explains the mechanism behind this bidirectional communication and provides sample drivers that are referenced throughout the command set descriptions. Because of its wide usage in the industry, the C/C++ programming language is used to code all programming examples.

Chapter 6 details the command set. A general-purpose driver function is shown for each command, along with one or more examples showing how to employ the driver in an application program.

5.1 Programming Model

The Smart A/D™ occupies two contiguous addresses in the ISA I/O space. Both of the addresses occupied by the board may be written to and read from, but each address has a distinct function for read and write operations.

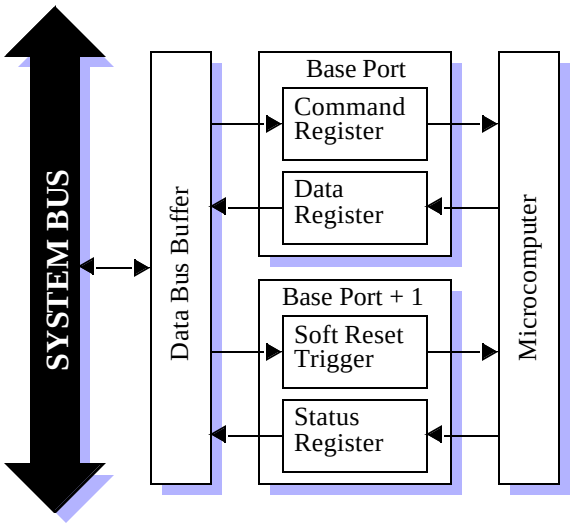
Table 7: Programming model

Port Address	Function	
	Read	Write
Base + 0	Data Register	Command Register
Base + 1	Status Register	Board Reset

The command/data port (base address) provides the data path between the Smart A/D™ and the host. The host "sends" a command byte to the Smart A/D™ by storing it in the command register. The host "reads" a command response byte by fetching the contents of the data register. Because the command register is write-only and the data register is read-only, both of these registers are able to share the same port address.

The status/reset port (base address + 1) also has a dual function. In the case of read operations, it provides Smart A/D™ status information to the host (see Section 5.1.1). When the host writes a byte to the status port a hardware reset is invoked on the Smart A/D™. The data value written to the port is "don't care," but should be set to zero to ensure compatibility with future product enhancements.

Figure 12: Bus interface block diagram



5.1.1 Handshake Mechanism

The status register provides the host with information used for Smart A/D™ status monitoring and communication handshake control. When the host reads the status register, a byte of the following form is returned:

Table 8: Status Register

D7	D6	D5	D4	D3	D2	D1	D0
CRMT	DAV	ALRM	FLT	X	X	X	X

Status register bits, which are all active-high, have the following functions:

- **CRMT** indicates Smart A/D™ is ready for a command byte. **The host should write to the command register only when CRMT is active.**
- **DAV** indicates Smart A/D™ data register contains data for the host. **The host should read from the data register only when DAV is active.**
- **ALRM** indicates one or more programmed channel limits were exceeded.
- **FLT** indicates board reset in progress or board fault detected. In normal operation this bit goes active for approximately one-half second following a board reset. FLT reflects the state of the red LED fault indicator. **When FLT is active, the CRMT, DAV and ALRM bits are not meaningful and should be ignored.**

5.2 Commands

The Smart A/D™ board is accessed and controlled by means of a simple, yet powerful, built-in command set.

Commands consist of byte strings that have varying lengths, due to differences in the amount of data embedded in each command. In some cases, invocation of a command will cause the Smart A/D™ to return data to the host.

5.2.1 Opcodes

Each command consists of at least one byte. The first byte of a command generally conforms to the format shown in Table 9.

The opcode, which is always present in the first command byte, specifies the function to be performed. The channel number is required only in commands that address a specific channel.

Table 9: Format of first command byte

D7	D6	D5	D4	D3	D2	D1	D0
OPCODE				CHANNEL			

5.2.2 Conventions

The following conventions are used to specify commands and command responses:

- The symbolic name `CHAN` represents a Smart A/D™ channel number. Valid channel numbers range from 0 through 7.
- A **byte** is represented as an expression contained by parenthesis. For example, `(16+CHAN)` specifies a byte that has a value equal to 16 plus a channel number.
- A **byte string** is an ordered sequence of bytes separated by commas. The sequence is ordered from left to right. For example, the string `(95+CHAN),(5),(0)` contains three bytes.
- A **command** is a byte string that is handshaked, one byte at a time, from the host to the Smart A/D™. All bytes of a command must be handshaked to the Smart A/D™ before fetching a response or issuing another command.
- A command **response** is a byte string that is handshaked, one byte at a time, from the Smart A/D™ to the host. All bytes of a pending response must be handshaked from the Smart A/D™ before issuing another command.

5.3 Low Level Drivers

We recommend that you incorporate procedures in your application program that will conceal the Smart A/D™ handshake protocol from higher host software layers. This will serve to simplify your programming efforts as well as help to reduce software maintenance costs.

This section contains sample low-level driver functions that implement the required handshake protocol. These functions are referenced throughout the command programming examples in Chapter 6.

5.3.1 Manifest Constants

Several constants are required by the sample drivers in Section 5.3.2 and the programming examples in Chapter 6. Instead of embedding constants in the examples in which they are referenced, all constants have been gathered here. By grouping the constants here, programming examples are easier to read and understand and unnecessary duplication is prevented.

```
#define MyBasePort  0x02B2        /* Base address for all examples */

// ====== COMMAND OPCODES ======

#define CMD_SET_TYPE        16    /* DECLARE CHANNEL SENSOR     */
#define CMD_SET_LIMITS      32    /* SET ALARM LIMITS           */
#define CMD_GET_ALARMS      48    /* READ ALARM FLAGS           */
#define CMD_GET_AMBIENT     64    /* READ AMBIENT TEMPERATURE   */
#define CMD_SET_REJECT50HZ  72    /* START 50HZ REJECTION MODE  */
#define CMD_SET_OPENFLAGS   80    /* SET OPEN-DETECT MODE FLAGS */
#define CMD_READ_ALL        88    /* READ ALL SENSOR CHANNELS   */
#define CMD_SET_FILT        96    /* SET FILTER TIME CONSTANT   */
#define CMD_SET_GAGETARE   112    /* TARE GAGE                  */
#define CMD_GET_GAGECAL    128    /* GET GAGE CALIBRATION       */
#define CMD_SET_GAGECAL    144    /* SET GAGE CALIBRATION       */
```

```
#define CMD_SET_GAGEZERO   176    /* SET GAGE ZERO              */
#define CMD_SET_COEFFS     192    /* SET COEFFICIENTS           */
#define CMD_SET_GAGESPAN   208    /* SET GAGE SPAN              */
#define CMD_EXTENDED       240    /* EXTENDED OPCODE PREFIX     */

// ====== EXTENDED COMMAND OPCODES ======

#define XCMD_GET_PRODUCTID   4    /* GET MODEL NUMBER           */
#define XCMD_GET_VERSION     5    /* GET FIRMWARE VERSION       */
#define XCMD_SET_HISPEED     8    /* SWITCH TO HIGH SPEED MODE  */

// ====== STATUS REGISTER BIT MASKS ======

#define STATUS_CRMT       0x80    /* Command Register eMpTy     */
#define STATUS_DAV        0x40    /* Data AVailable             */
#define STATUS_ALRM       0x20    /* ALaRM sounding             */
#define STATUS_FLT        0x10    /* board FauLT or reset       */

// ====== SENSOR DEFINITION CODES ======

#define TYPE_TC_K         0x1C    /* K thermocouple, 0.1 C/bit  */

// ====== SENSOR SCALARS ======

#define SCALAR_VOLTS_5   0.0002   /* 5V range, 200uV/bit        */
#define SCALAR_TC_K      0.1000   /* K thermocouple, 0.1C/bit   */
```

5.3.2 Sample Drivers

```
// Handshake a command byte to the command register.
VOID SendByte( USHORT BasePort, UCHAR cmd_byte )
{
    while ( !( inp( BasePort + 1 ) & STATUS_CRMT ) );  // wait for CRMT
    outp( BasePort, cmd_byte );                        // send command byte
}

// Handshake a response byte from the data register.
UCHAR ReadByte( USHORT BasePort )
{
    while ( !( inp( BasePort + 1 ) & STATUS_DAV ) );   // wait for DAV
    return inp( BasePort );                            // read response byte
}

// Send a 16-bit integer value to Smart A/D board.
VOID SendWord( USHORT BasePort, SHORT Value )
{
    SendByte( BasePort, ( Value >> 8 ) & 0xFF )        // send high byte
    SendByte( BasePort, Value & 0xFF )                 // send low byte
}

// Fetch a 16-bit integer value from Smart A/D board.
SHORT ReadWord( USHORT BasePort )
{
    SHORT val = (SHORT) ReadByte( BasePort ) << 8;     // read high byte
    return val | (SHORT) ReadByte( BasePort ) );       // read low byte
{

// Reset Smart A/D board.
VOID ResetBoard( USHORT BasePort )
{
    outp( BasePort + 1, 0 );                           // reset the board
    while ( inp( BasePort + 1 ) & STATUS_FLT );        // wait for FLT to clear
{
```

```
// Send a floating point value to the Smart A/D board.
VOID SendReal( USHORT BasePort, DOUBLE Value )
{
    int   exponent;
    DOUBLE mantissa = frexp( fabs( Value ), &exponent );

    if ( exponent < -127 )
    {
        SendWord(0);
        SendWord(0);
    }
    else
    {
        SendByte( (UCHAR)( mantissa * 16777216.0 ) );
        SendByte( (UCHAR)( mantissa * 65536.0 ) );
        SendByte( (UCHAR)( mantissa * 256.0 - ( Value > 0 ? 128.0 : 0.0 ) ) );
        SendByte( (UCHAR)( !Value ? 0 : exponent + 128 );
    }
}
```

6 Commands

6.1 Board Configuration

6.1.1 GetProductID

Command: `(240),(4),(0)`
Response: `(ModelNumMSB),(ModelNumLSB)`

Returns the Smart A/D model number. In the case of model 418, the value 418 (decimal) is returned.

This command can be used as a diagnostic tool to help you during application program development.

Driver

```
// Return the model number from a Smart A/D board.
USHORT GetProductID( USHORT BasePort )
{
    SendByte( BasePort, CMD_EXTENDED );
    SendByte( BasePort, XCMD_GET_PRODUCTID );
    SendByte( BasePort, 0 );
    return (USHORT) ReadWord( BasePort );
}
```

Example

```
// Get a board's model number.
USHORT MyModelNum = GetProductID( MyBasePort );
```

6.1.2 GetFWVersion

Command: `(240),(5),(0)`

Response: `(VersionMSB),(VersionLSB)`

Returns the Smart A/D firmware version number, times 100. The version number is printed on the EPROM device that is plugged into the Smart A/D board. This can be used to automatically determine if enhanced functions, which may only be implemented in specific firmware releases, are available.

Driver

```
// Return the firmware version number from a Smart A/D board.
DOUBLE GetFWVersion( USHORT BasePort )
{
    SendByte( BasePort, CMD_EXTENDED );
    SendByte( BasePort, XCMD_GET_VERSION );
    SendByte( BasePort, 0 );
    return (DOUBLE) ReadWord( BasePort ) * 0.01;
}
```

Example

```
// Get a board's firmware version number.
DOUBLE MyFWVersion = GetFWVersion( MyBasePort );
```

6.1.3 SetHiSpeedMode

Command: `(240),(8),(0)`
Response: `None`

Decreases the channel slot time (Section 8.1) to enable higher throughput.

Assuming, for example, that the board is operating in the default 60Hz rejection mode, this command will cause all channel slot times to be reduced from 22 milliseconds (default) to 9 milliseconds, and the board will be operating in the "high-speed mode."

Decreasing the slot time causes measurement accuracy to be reduced by approximately 75 percent and may cause increased measurement noise. Also, line frequency rejection is no longer possible as the digitization period is shorter than one line cycle.

A board reset will restore slot times to their default values. This is the only way to resume the default update rate after invoking the `SetHiSpeedMode` function.

When the high-speed mode is already active, any attempt to invoke this command again will be ignored.

Driver

```
// Switch a Smart A/D board to the high speed measurement mode.
VOID SetHiSpeedMode( USHORT BasePort )
{
    SendByte( BasePort, CMD_EXTENDED );
    SendByte( BasePort, XCMD_SET_HISPEED );
    SendByte( BasePort, 0 );
}
```

Example

```
// Configure a board to operate in the high speed mode.
SetHiSpeedMode( MyBasePort );
```

6.1.4 SetReject50Hz

Command: (72)
Response: None

Increases the digitizer's integration period from 16.7 milliseconds (default) to 20.0 milliseconds, enabling the Smart A/D to better reject 50Hz differential noise.

This function should be used in systems targeted for 50Hz power environments.

A board reset will restore the board to the default 60Hz rejection mode. This is the only way to return to the 60Hz rejection mode after invoking the SetReject50Hz function.

When the 50Hz rejection mode is active, any attempt to invoke this command again will be ignored.

Driver

```
// Switch a Smart A/D board to the 50Hz rejection mode.
VOID SetReject50Hz( USHORT BasePort )
{
    SendByte( BasePort, CMD_SET_REJECT50HZ );
}
```

Example

```
// Configure a board to operate in the 50Hz rejection mode.
SetReject50Hz( MyBasePort );
```

6.2 Channel Configuration

6.2.1 SetSensorType

Command: `(16 + Chan),(SensorDefinitionCode)`
Response: `None`

Declares the type of sensor connected to a channel. By declaring the sensor type, you are implicitly selecting the excitation mode, measurement gain setting and linearization curve. After executing this command, the onboard microcomputer will perform all subsequent scans of the specified channel as appropriate for the declared sensor type.

This function has several side-effects that should be noted:

- Sensor data is not available immediately after declaring a channel's sensor type. After executing this function, sensor data is reset to zero. The maximum elapsed time that must pass before new sensor data becomes available is equal to ten channel slot times. Details of channel slot timing can be found in Section 8.1.
- The channel filter constant (Section 6.2.3) is reset to zero, effectively disabling the software filter function.
- Limit alarms (Section 6.5.1) are disabled, as the new sensor type may have a new type of engineering units.

Typically, the host will execute this command once for each sensor channel. Each execution declares the sensor type for one channel, therefore eight invocations must occur to declare all channel sensor types.

Each sensor type supported by the Smart A/D board is assigned a unique value, called the Sensor Definition Code (SDC). This value, along with the target sensor channel number, is passed to the Smart A/D board as a function argument.

Upon reset of the Smart A/D board, all channel sensor types default to the ±5 Volt input range, with 500µV resolution, and all sensor data values are reset to zero. Channel sensor types remain at the default setting until changed by this function. If the default sensor type is suitable for your application, you need not execute this function at all.

Any attempt to declare an unsupported sensor type will result in selection of the default sensor type.

Declaring a sensor type as "Disabled" inhibits scanning of the corresponding channel, resulting in increased throughput for all remaining active channels.

See Table 12 on page 36 for a complete list of supported sensor types and SDCs.

Driver

```
// Configure a channel for the specified sensor type.
VOID SetSensorType( USHORT BasePort, UCHAR Channel, UCHAR SensorDefCode )
{
    SendByte( BasePort, Channel | CMD_SET_TYPE );
    SendByte( BasePort, SensorDefCode );
}
```

Example

```
// Configure channel 2 for interface to a K thermocouple.
SetSensorType( MyBasePort, 2, TYPE_TC_K );
```

6.2.2 SetFailMode

Command: `(80 + Group),(OpenFlags)`
Response: `None`

Establishes open sensor data values for a group of eight channels. The first byte consists of the base opcode plus the channel group number (0 for channels 0-7, or 1 for channels 8-15).

The second command byte contains eight bit flags, one for each channel. In the case of channels 0 to 7, flags are mapped to channels as follows: bit 7 (most significant) to channel 7, bit 6 to channel 6, etc. In the case of channels 8 to 15 (model 419 only), flags are mapped to channels as follows: bit 7 (most significant) to channel 15, bit 6 to channel 14, etc.

If a flag is set (logic 1) and the corresponding sensor is open, the value 32767 will replace the sensor data. On the other hand, if a flag is reset and the sensor is open, the value -32768 will replace the sensor data.

This function is useful for triggering alarms when open sensors are detected, and for forcing the desired system response to fault conditions in closed-loop control applications.

Driver

```
// Set Open Sensor data values for all eight channels on a model 418.
VOID SetFailMode_418( USHORT BasePort, UCHAR OpenFlags )
{
    SendByte( CMD_SET_OPENFLAGS );
    SendByte( OpenFlags );
}

// Set Open Sensor data values for all 16 channels on a model 419.
VOID SetFailMode_419( USHORT BasePort, UCHAR OpenFlagsHi, UCHAR OpenFlagsLo )
{
    // Program channels 0-7.
    SendByte( CMD_SET_OPENFLAGS );
    SendByte( OpenFlagsLo );

    // Program channels 8-15.
    SendByte( CMD_SET_OPENFLAGS + 1 );
    SendByte( OpenFlagsHi );
}
```

Example

```
// A controller uses a thermocouple on channel 1 to monitor furnace
// temperature. If the thermocouple fails, the Smart A/D must indicate
// a high temperature on channel 1 to ensure that the controller turns off
// the furnace heater. This code segment sets up channel 1 to fail high
// and all other channels to fail low.

SetFailMode_418( MyBasePort, 0x02 );   // OpenFlags = (0,0,0,0,0,0,1,0)
```

6.2.3 SetFilter

Command: `(96 + Chan),(FilterFactor)`
Response: `None`

Establishes a software single-pole low-pass filter for the specified channel. The second command byte specifies a filter factor, **F**, which may have any value from 0 to 255, inclusive. The approximate relationship between F and filter percentage, **P**, is:

$$P = 2.55 * F$$

All filter constants default to zero after a reset. This effectively disables the software filters, thereby maximizing the frequency response of all channels. Applying a non-zero factor in electrically noisy environments may help to reduce measurement noise.

Rather than computing the theoretical value of filter constants, we recommend that you experiment with different filter constants to find the best response-time/noise-immunity tradeoff point.

Some applications require the use of a specific filter time constant value. This function describes the relationship between time constant and filter factor:

$$t = (-N / (45 * \ln(F / 256)))$$

where: t is the time constant in seconds
F is the filter factor in the range 0 to 255
N is the number of active channels.

Driver

```
// Set the filter factor for the specified channel.

VOID SetFilter( USHORT BasePort, UCHAR Channel, UCHAR FiltFactor )
{
    SendByte( BasePort, Channel | CMD_SET_FILT );  // send channel & opcode
    SendByte( BasePort, FiltFactor );              // send filter factor
}
```

Example

```
// Set up channel 4 with a 12.5 percent low-pass filter.

SetFilter( MyBasePort, 4, (UCHAR)( 12.5 * 2.55 ) );
```

6.2.4 SetCoefficients

Command: `(192 + Chan),(A0),(A1),(A2),(A3),(B0),(B1),(B2),(B3),(C0),(C1),(C2),(C3)`
Response: `None`

Defines linearization coefficients for a channel that was previously declared for sensor type *Custom Resistive Sensor* (CRS).

Three coefficients are specified. The three values belong to a polynomial of the following form:

$$f(R) = aR^2 + bR + c$$

The Smart A/D™ measures CRS channels by performing an autoranging measurement of the sensor resistance. The measured resistance is then transformed into user-defined units by means of the above second-degree polynomial expression. The resulting value is converted to a 16-bit integer and stored in onboard memory. This is the value that is returned to the host when the `GetSensorData` or `GetAllSensors` commands are executed.

Declaring a channel as type CRS is practical for resistive sensors subject to the following constraints:

- Sensor resistance must not exceed the maximum resistance that can be measured by the Smart A/D™.
- A quadratic linearization polynomial is sufficiently accurate for the application.

Each coefficient is expressed in the 4-byte form used internally by the Smart A/D. The the `SendReal` function (see Section 5.3) takes care of translating the coefficients into the required internal format.

Driver

```
// Set the coefficients for the specified channel.  The coefficients are
// passed to the driver function through an array.

VOID SetCoefficients( USHORT BasePort, UCHAR Channel, DOUBLE *pCoeffs )
{
    SendByte( BasePort, Channel | CMD_SET_COEFFS );
    for ( USHORT nCoef = 0; nCoef < 3; nCoef++ )
        SendReal( BasePort, *pCoeffs++ );
}
```

Example

```
// Set the coefficients for channel 2, which is connected to a linear slide
// potentiometer whose position, P, is a function of resistance, R, as
// follows:  P = -0.0023 * R^2 + 12.6 * R + 0.45.  After executing this command,
// the GetSensorData and GetAllSensors commands will return the value P from
// this channel in the form of a signed 16-bit integer.

DOUBLE Coeffs[] = { -0.0023, 12.6, 0.45 };

SetCoefficients( MyBasePort, 2, Coeffs );
```

6.3 Sensor Acquisition

6.3.1 GetSensorData

Command: `(Chan)`
Response: `(DataMSB),(DataLSB)`

Returns the most recently acquired sensor data from a channel. The returned 16-bit integer value is scaled as a function of the declared channel sensor type.

Regardless of sensor type, the returned value is always represented in 16-bit two's complement integer form. The returned value must be multiplied by an appropriate scalar to convert it to floating point engineering units. Refer to Table12 on page36 for a list of sensor scalars.

Driver

```
// Read sensor data from specified channel in appropriate engineering units.
// Scalar converts integer sensor data to engineering units.

DOUBLE GetSensorData( USHORT BasePort, UCHAR Channel, DOUBLE Scalar )
{
    SendByte( BasePort, Channel );                  // request sensor data
    return Scalar * (DOUBLE) ReadWord( BasePort );  // return scaled data
}
```

Example

```
// Read voltage at Channel 6, which is configured for 5V, 200uV/bit range.
DOUBLE Chan6volts = GetSensorData( MyBasePort, 6, SCALAR_VOLTS_5 );
```

Example

```
// Read temperature (in degrees C) at channel 4, which is configured for
// a type K thermocouple.

DOUBLE Chan4temp = GetSensorData( MyBasePort, 4, SCALAR_TC_K );
```

6.3.2 GetAllSensors

Command: (88)

Response: (MSB0),(LSB0),(MSB1),(LSB1),(MSB2),(LSB2),(MSB3),(LSB3),
(MSB4),(LSB4),(MSB5),(LSB5),(MSB6),(LSB6),(MSB7),(LSB7)

Command: (89)

Response: (MSB8),(LSB8),(MSB9),(LSB9),(MSB10),(LSB10),(MSB11),(LSB11),
(MSB12),(LSB12),(MSB13),(LSB13),(MSB14),(LSB14),(MSB15),(LSB15)

Returns the most recent sensor data from a group of eight channels. Data is returned from all eight channels in the group, including disabled channels. The value returned from a disabled channel is indeterminate.

Sensor data is scaled as a function of the declared sensor type connected to each channel. Refer to Table12 on page36 for a list of sensor data scalars.

Driver

```
// Fetch sensor data from all Smart A/D channels into an array.
// The resulting integer data must be scaled to engineering units.

VOID GetAllSensors( USHORT BasePort, SHORT *pArray, int NumGroups )
{
    for ( int group = 0; group < NumGroups; group++ )
    {
        SendByte( BasePort, CMD_READ_ALL + group );
        for ( UCHAR ch = 0; ch < 8; *pArray++ = ReadWord( BasePort, ch++ ) );
    }
}
```

Example

```
// Read all sensor data from a model 418 board (8 channels) into an array, then
// return the voltage at channel 6, which is configured for 5V,200uV/bit range.

SHORT SensorData[8];              // Buffer to receive integer sensor data

GetAllSensors( MyBasePort, SensorData, 1 );    // get sensor data

// Fetch channel 6 integer sensor data from array & convert to volts.
DOUBLE Chan6volts = SCALAR_VOLTS * (DOUBLE) SensorData[6];
```

Example

```
// Read all sensor data from a model 419 board (16 channels) into an array, then
// return the voltage at channel 6, which is configured for 5V,200uV/bit range.

SHORT SensorData[16];             // Buffer to receive integer sensor data

GetAllSensors( MyBasePort, SensorData, 2 );    // get sensor data

// Fetch channel 6 integer sensor data from array & convert to volts.
DOUBLE Chan6volts = SCALAR_VOLTS * (DOUBLE) SensorData[6];
```

6.3.3 GetAmbientTemp

Command: `(64 + Group)`
Response: `(TemperatureMSB),(TemperatureLSB)`

Returns the temperature from an optional Smart A/D™ termination board such as Model 7409TB, 7409TF or 7409TC. Each of these termination boards, which connects to a group of eight sensor channels, has a reference junction sensor.

The returned temperature value is scaled to 0.10 °C/bit. If no Sensoray termination board is connected to the Smart A/D™, the `GetAmbientTemp` command will return a meaningless value.

Useful for monitoring the ambient temperature at thermocouple reference junctions, this function is also a good debugging aid during Smart A/D™ installation.

Driver

```
// Return the ambient temperature in degrees C.
DOUBLE GetAmbientTemp( USHORT BasePort, UCHAR Group )
{
    SendByte( BasePort, CMD_GET_AMBIENT + Group );
    return (DOUBLE) ReadWord( BasePort ) * 0.10;
}
```

Example

```
// Get the ambient temperature of the channel 0-7 termination board in degrees F.
DOUBLE AmbientDegF = GetAmbientTemp( MyBasePort, 0 ) * 1.8 + 32.0;
```

6.4 Pressure/Strain Gage

6.4.1 SetGageTare

Command: `(112 + Chan)`
Response: `None`

Tares the strain/pressure gage connected to a single channel. The `SetGageTare` command may be used to compensate for the weight of an empty container prior to measuring the container when it holds some substance.

Before invoking this command, the gage channel should be calibrated. See the `SetGageZero` and `SetGageSpan` commands for details.

Driver

```
// Tare the gage on the specified channel.
VOID SetGageTare( USHORT BasePort, UCHAR Channel )
{
    SendByte( BasePort, Channel | CMD_SET_GAGETARE );
}
```

Example

```
// An empty truck must be tared prior to loading. The truck scale
// is measured by means of a load cell connected to channel 7. Channel 7 has
// been previously calibrated using the SET GAGE ZERO and SET GAGE SPAN
// commands.

SetGageTare( MyBasePort, 7 );      // tare the empty truck
```

6.4.2 SetGageZero

Command: `(176 + Chan)`
Response: `None`

Informs the Smart A/D that there is presently no load applied to the target channel's gage. The Smart A/D registers this condition and uses it as a reference for all subsequent measurements on the target channel.

Typically, this command is executed just before invoking the `SetGageSpan` command. Acting together, the `SetGageZero` and `SetGageSpan` commands serve to completely calibrate a gage channel.

Driver

```
// Issue a Set Gage Zero command for the specified channel.
VOID SetGageZero( USHORT BasePort, UCHAR Channel )
{
    SendByte( BasePort, Channel | CMD_SET_GAGEZERO );
}
```

Example

```
See the programming example shown for the SetGageSpan command.
```

6.4.3 SetGageSpan

Command: `(208 + Chan),(DataMSB),(DataLSB)`
Response: `None`

Sets the effective gain of a strain gage channel. Many applications have the ability to simulate gage loads by applying simulated zero and full-scale load conditions to the gage channel.

To utilize this command, apply the zero-load signal onto the channel and issue the `SetGageZero` command. Next, switch the full-load signal onto the channel and issue the `SetGageSpan` command.

Driver

```
// Issue a Set Gage Span command for the specified channel.
VOID SetGageSpan( USHORT BasePort, UCHAR Channel, SHORT LoadValue )
{
    SendByte( BasePort, Channel | CMD_SET_GAGESPAN);  // send chan & command
    SendWord( BasePort, LoadValue );                  // send load value
}
```

Example

```
// Calibrate the load cell connected to channel 2. Full-load is
// 40,000 pounds, and the Smart A/D output units are to be scaled to 10
// lbs/bit. At full-load, therefore, Smart A/D output will be 4,000.

// ... Wait until zero load is applied to the gage ...

SetGageZero( MyBasePort, 2 );         // Set the gage zero

// ... Wait until full-scale load is applied to the gage ...

SetGageSpan( MyBasePort, 2, 4000 );   // Set the gage span
```

6.4.4 GetGageCal

Command: `(128 + Chan),(SensorDefinitionCode)`
Response: `(S0),(S1),(S2),(S3),(S4),(S5)`

Returns the Smart A/D™ internal calibration parameters for a strain gage channel. The response string is six bytes long.

The data byte values that are returned in the command response are represented in an internal format that is meaningful only to the Smart A/D™.

In order for the command response bytes to have legitimate values, the strain gage channel must already have been calibrated. This prior calibration may be accomplished in either of two ways: using the `SetGageZero` and `SetGageSpan` commands, or using the `SetGageCal` command.

Driver

```
// Copy the specified channel's gage parameters into an array.  The
// target channel must be configured for a strain gage sensor and
// must be calibrated before calling this procedure.

VOID GetGageCal( USHORT BasePort, UCHAR Channel, SHORT *pArray )
{
    SendByte( BasePort, Channel | CMD_GET_GAGECAL );

    for ( USHORT parm = 0; parm < 3; parm++ )
        *pArray++ = ReadWord( BasePort );
}
```

Example

```
// Fetch the gage calibration parameters from channel 6.

SHORT GageParms[3];

GetGageCal( MyBasePort, 6, GageParms );
```

6.4.5 SetGageCal

Command: `(144 + Chan),(S0),(S1),(S2),(S3),(S4),(S5)`
Response: `None`

Restores calibration parameters to a strain gage channel. These parameters must have been previously obtained via execution of the `ReadGageCal` command.

The data byte values that are embedded in this command string are represented in an internal format that is meaningful only to the Smart A/D™.

Driver

```
// Load a channel's gage calibration parameters from an array.

VOID SetGageCal( USHORT BasePort, UCHAR Channel, SHORT *pArray )
{
    SendByte( BasePort, Channel | CMD_SET_GAGECAL );

    for ( USHORT parm = 0; parm < 3; parm++ )
        SendWord( BasePort, *pArray++ );
}
```

Example

```
// Restore gage calibration parameters back onto channel 6.
// Assume that the GetGageCal() function previously stored the gage
// calibration parameters in the GageParms[] array as shown in the
// Read Gage Calibration example.

SetGageCal( MyBasePort, 6, GageParms );
```

6.5 Alarms

6.5.1 SetAlarmLimits

Command: `(32 + Chan),(HiLimitMSB),(HiLimitLSB),(LoLimitMSB),(LoLimitLSB)`
Response: `None`

Specifies high and low alarm limits for a channel. An alarm will sound if sensor data is more positive than the high limit or more negative than the low limit. Limits are specified as 16-bit signed integers.

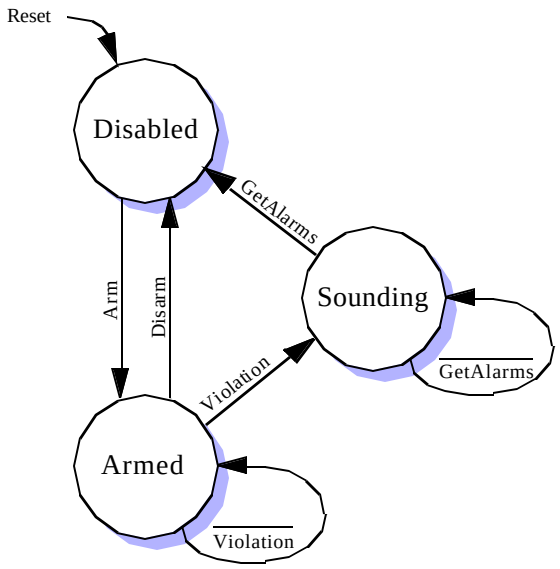
The high limit may be disabled by setting it to 32767, and the low limit may be disabled by setting it to -32768.

Limits must be converted from engineering units to the required integer values by scaling as appropriate for the channel's sensor type. In the case of a K thermocouple (0.1°C/bit) for example, divide the desired limit (expressed in °C) by 0.1 to obtain the integer value.

As shown in Figure 13, all channel alarms are initially in the Disabled state. To switch to the Armed state, the `SetAlarmLimits` command is invoked with the desired limit values specified. To return to the Disabled state, invoke `SetAlarmLimits` with both limits disabled.

If a limit violation is detected while the alarm is Armed, the channel switches to the Sounding state. This causes the Smart A/D™ status register `ALRM` flag to be asserted. A channel remains in the Sounding state until the `GetAlarms` command is executed. When `GetAlarms` executes, the channel switches to the disabled state, where it remains until the host again arms the alarm.

Figure 13: Alarm state diagram for an individual channel



When an alarm 'sounds', both violated limits are reset to the default value. This prevents the alarm from sounding again after the host has acknowledged the alarm. The host must reprogram the violated limit if the Smart A/D™ is to continue monitoring it.

Driver

```
// Set alarm limit values for the specified channel.
// Scalar converts limit values to Smart A/D integer units.

VOID SetAlarmLimits( USHORT BasePort, UCHAR Channel, DOUBLE LoLim, DOUBLE HiLim, DOUBLE Scalar )
{
    SendByte( BasePort, Channel | CMD_SET_LIMITS );     // send chan & opcode
    SendWord( BasePort, (SHORT)( HiLim / Scalar ) );    // send high limit
    SendWord( BasePort, (SHORT)( LoLim / Scalar ) );    // send low limit
}
```

Example

```
// Channel 7 is measuring a K thermocouple (0.1 C/bit) that monitors a process
// temperature which must fall between 400 and 450 degrees C. Program alarm
// limits so that the Smart A/D alarm flag will be asserted if the temperature
// strays outside operating limits.

SetAlarmLimits( MyBasePort, 7, 400.0, 450.0, SCALAR_TC_K );
```

6.5.2 GetAlarms

Command: `(48 + Group)`
Response: `(HighAlarmFlags),(LowAlarmFlags)`

Returns the status of all channel alarms for a group of eight channels.

The first response byte indicates the status of all high alarms while the second byte indicates the status of all low alarms. In the case of group 0, Bits 7 (most significant) through bit 0 (least significant) of each response byte corresponds to channels 7 through 0, respectively. In the case of group 1, Bits 7 (most significant) through bit 0 (least significant) of each response byte corresponds to channels 15 through 8, respectively. A status bit containing logic 1 indicates a violated alarm limit while a logic 0 indicates no limit violation.

In addition to returning alarm flag data, this command also resets all channel alarm flags as well as the Smart A/D™ status register alarm bit.

Driver

```
// Read alarm flags for an 8-channel group.  Low alarm flags are returned
// in the LSB, and high alarm flags are returned in the MSB.

USHORT GetAlarms( USHORT BasePort, UCHAR Group )
{
    if ( inp( BasePort + 1 ) & STATUS_ALRM )    // If at least one chan alarm sounding ...
    {
        SendByte( CMD_GET_ALARMS + Group );     //   request all alarm flags
        return (USHORT)ReadWord(MyBasePort);    //   fetch and return all alarm flags
    }
    else                                        // Otherwise ...
        return 0;                               //   indicate no alarms are sounding
}
```

Example

```
// Read alarm flags and service any channels that are sounding an alarm.

USHORT AlarmFlags = GetAlarms( MyBasePort );

for ( UCHAR Channel = 0; Channel < 8; Channel++ )
{
    if ( AlarmFlags & 0x0100 )
    {
        // Service high limit violation on Channel ...
    }

    if ( AlarmFlags & 0x0001 )
    {
        // Service low limit violation on Channel ...
    }

    AlarmFlags >> 1;
}
```

7 Theory of Operation

7.1 Firmware

The Smart A/D's internal microcomputer executes a firmware-resident control program. At the heart of this program is a high-performance, event-driven kernel.

The following paragraphs describe the functions of the major tasks that are managed by the kernel.

7.1.1 Digitizer

This task performs the fundamental data acquisition function central to the Smart A/D's purpose.

The Digitizer begins a conversion by selecting the next channel to be digitized. The sense and excitation multiplexers are switched to the desired channel, and, if the channel is connected to a passive sensor, the pulsed excitation source is configured and activated. The scanner then sleeps while the sensor signal stabilizes.

When the sensor signal has stabilized, the Digitizer initiates an A/D conversion. Other tasks are allowed to run while the conversion is taking place.

Upon completion of the conversion, the digitized value is forwarded to the Cruncher task.

7.1.2 Cruncher

The Cruncher task is responsible for polishing raw A/D data into finished form. Typically, the Cruncher task runs while the scanner is digitizing the next channel in the scan loop.

This task begins when it receives a digitized value from the Digitizer task. The data is normalized to the internal standards, then is linearized and converted to engineering units appropriate for the declared sensor type. The resulting value is passed through a single-pole low-pass filter.

Next, the filtered data is checked for alarm limit violations. If a limit has been exceeded, the channel alarm is disabled and the Alarm status flag is set.

Finally, the processed sensor data is posted to internal dual-ported RAM for rapid client access.

7.1.3 Command Processor

The Command Processor receives and executes commands from the host. Top priority is given to this task in order to minimize communication latency.

This task is run when a command is received from the client.

7.2 Analog Circuits

The Smart A/D analog circuitry is functionally partitioned into Measurement and Excitation sections.

7.2.1 Measurement Section

The Measurement section selects and conditions a sensor input signal for digitizing.

The Measurement section begins sensor signal processing by routing a pair of channel sense signals through a differential analog multiplexer. The selected channel is then conditioned by a programmable gain amplifier. Finally, the amplified signal is applied to the input of an integrating A/D converter.

Channel selection, amplifier gain, and A/D functions are all under control of the onboard microcomputer. None of these circuits are directly accessible by the client.

7.2.2 Excitation Section

The Excitation section sources a pulsed DC current or voltage that is used to excite passive sensors.

The Excitation section supplies one of three pulsed signals to passive sensors, depending on the declared sensor type. The length of the excitation pulse is equal to the channel slot time (see Section 8.1 for an explanation of slot time).

In the case of strain and pressure gages, 10VDC excitation is applied. This voltage is current limited so that shorting the excitation signals together or to ground will not damage the excitation source.

In the case of RTD's and the 400Ω and 3KΩ resistance ranges, a constant current of approximately 1.2mADC is forced through the sensor.

5VDC is applied, in series with a reference resistor, to sensors measured on the 600KΩ resistance range.

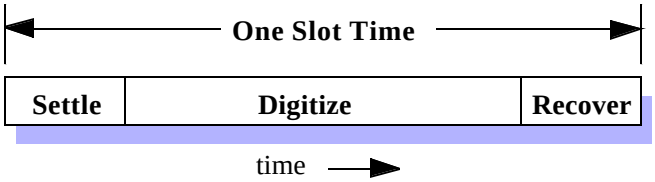
8 Timing

Three timing parameters are important from the application developer's viewpoint: Channel Update Rate, Data Age and Communication Latency. This section discusses these parameters along with some important related issues.

8.1 Slot Time

Channel Slot Time is the length of time required for the Smart A/D to completely process one sensor channel.

Each channel slot time encompasses three functional phases as shown in the following diagram:



In the first phase, a sensor channel is excited and its sense signals are routed through to the Smart A/D measurement section. The embedded microcomputer configures the signal path as appropriate for the sensor type, then the analog front end is allotted time to stabilize.

The digitizer acquires the sensor data value in the second phase. The Update Rate, which is described in the next section, can be increased by reducing the duration of the time slot Digitize phase. See Section 6.1.3 for details.

A Recover phase occurs at the end of the time slot. The function of this phase is to reset the analog front end in preparation for the next conversion.

Computationally intensive processes, such as linearization, alarm processing and software filtering, don't affect the slot time because they execute concurrently during the subsequent channel slot time.

8.2 Update Rate

Update Rate is defined as the number of times each second that a sensor channel acquires new sensor data.

8.2.1 Primary Influences

The two primary influences on update rate are the number of active channels and the channel slot time.

Due to timing uncertainties, the update rate is expressed as a range of values bounded by minimum and maximum times. The minimum and maximum update rate for any active channel is given by these functions:

$$UpdateRate_{min} = 1 / ((NumActiveChans + 1) * SlotTime)$$

$$UpdateRate_{max} = 1 / (NumActiveChans * SlotTime)$$

The time difference between the minimum and maximum update rate is due to the automatic, interleaved measurement of internal Smart A/D reference standards. Other than their timing impact, these measurements—which are scheduled and executed autonomously by the embedded processor—are transparent to the application program. At most, one internal standard will be measured per every sixteen sensor channel measurements.

Clearly, the update rate increases as the number of active channels decrease. Channels can be removed from the scan loop by disabling them with the SetSensorType command with the Disabled sensor definition code.

For example, a Smart A/D that has ten active channels and is running at the default channel slot time (22 milliseconds) would have a channel update rate ranging from 4.1 to 4.5 samples per second:

$$UpdateRate_{min} = 1 / ((10 + 1) * 0.022) = 4.1\ Hz$$

$$UpdateRate_{max} = 1 / (10 * 0.022) = 4.5\ Hz$$

8.2.2 Secondary Influences

Another influence on the update rate is the frequency of communication between host and Smart A/D.

The Smart A/D firmware is designed to minimize the impact of communications on the update rate. Even so, communication activity will sometimes interrupt scanning, which in turn will stretch the current channel's slot time in a non-deterministic way.

Because of the unpredictable effect on update rate, excessively frequent communication traffic between host and Smart A/D should be avoided.

8.3 Data Age

Data Age is the measure of time that has elapsed since a channel's data was acquired by the Smart A/D board.

The age of any channel's data is directly related to the update rate.

The minimum possible data age is, of course, zero. This would be the case if the host retrieves sensor data immediately after new data is posted by the Smart A/D.

The maximum possible data age would apply if the host retrieves sensor data immediately before new data is posted by the Smart A/D. In this case, the data age is the reciprocal of the minimum update rate.

In practice, data age almost always lies between the minimum and maximum possible values.

8.3.1 Example

As an example, take the case shown at the end of Section 8.2.1. The minimum update rate is approximately 4.1 Hertz, so the maximum data age would be:

$$DataAge_{max} = 1 / 4.1 = 244 \text{ milliseconds}$$

In this example, the age of sensor data from any given channel may range from zero to 244 milliseconds.

8.4 Communication Latency

Communication Latency is the time elapsed from a host request for sensor data to the acquisition of that data.

This latency can be viewed as having two components: host overhead and Smart A/D response time.

Host overhead is the time spent executing those parts of the Smart A/D interface (API) functions that are in the computational domain of the host (i.e., not pending on a Smart A/D response). Since this overhead—which is purely a function of host clock rate and architecture—varies from one system to another, it must be independently determined for each system.

Smart A/D response time is the time spent in an API function while waiting for a Smart A/D response. For obvious reasons, short communication latency is a high-priority design objective in many applications.

Communication functions are event-driven in the Smart A/D's local environment and given highes priority in order to minimize the communication latency. As a result, Smart A/D response time is predictable with a good degree of accuracy:

Table 10: Command response times

Command	Response Time (microseconds)
`GetSensorData`	`100`
`GetAllSensors`	`130`

9 Specifications

9.1 General Specifications

Table 11: General Specifications

Category	Parameter	Condition	Specification
Power	Quiescent	+5VDC, ±5%	100mA, typical.
		+12VDC, ±5%	75mA, typical.
Temperature	Operating range	Guaranteed accuracy	0°C to +70°C
Sense inputs	CMRR	CMV < 5V, f < 1KHz	80dB, minimum
	CMV	Guaranteed accuracy	±5.0V, maximum
	Input impedance	SCC disabled	1000 MΩ, minimum
		SCC enabled	2.35 MΩ, typical
	Input protection	Continuous	±20V CMV, maximum
		2 seconds, max.	±70V CMV, maximum
A/D converter	Type		Integrating, 16-bit resolution.
	Measurement period, per channel	60Hz reject mode (default mode)	16.7 milliseconds, typical (default mode). 4.0 milliseconds, typical (high-speed mode).
		50Hz reject mode	20 milliseconds, typical (default mode). 4.8 milliseconds, typical (high-speed mode).
	Total time slot, per channel	60Hz reject mode (default mode)	22 milliseconds, maximum (default mode). 9 milliseconds, maximum (high-speed mode).
		50Hz reject mode	25.3 milliseconds, maximum (default mode). 11 milliseconds, maximum (high-speed mode).
Excitation source	Pulse width		Applied for duration of channel time slot.
	Type	Strain/pressure gage	10VDC, 40mA maximum.
		400Ω/3KΩ ranges	1.2mADC.
		600KΩ range	5VDC in series with 4KΩ.
Bus interface	Type		I/O mapped, 2 consecutive byte addresses
	Address range		0x0200 to 0x03FF.

9.2 Sensor Specifications

Table 12: Sensor Specifications

Sensor Type		Measurement			SDC [7] (hex)
		Range	Resolution [1]	Accuracy [1]	
Thermocouple	B	0 to +1820°C	0.1°C	3.3°C	0x24
	C	0 to +1820°C	0.1°C	2.1°C	0x23
	E	-270 to +990°C	0.1°C	0.8°C	0x01
	J	-210 to +760°C	0.1°C	0.6°C	0x1B
	K	-270 to +1360°C	0.1°C	1.0°C	0x1C
	N	-270 to +1347°C	0.1°C	0.9°C	0x22
	T	-270 to +400°C	0.1°C	0.6°C	0x1D
	S	0 to +1760°C	0.1°C	3.0°C	0x1E
	R	0 to +1760°C	0.1°C	2.8°C	0x1F
Thermistor	Omega 44006 or 44031	-55°C to +145°C	0.01°C	0.05°C	0x1A
RTD	Cu 10Ω, 0.0367Ω/°C	0 to +119°C	0.1°C	0.6°C	0x2C
	Pt 100Ω, 0.385Ω/°C	-200 to +800°C	0.05°C	0.2°C	0x18
	Pt 100Ω, 0.392Ω/°C	-200 to +800°C	0.05°C	0.2°C	0x19
	Ni 200Ω, 1.10Ω/°C	-58.9 to +151.6°C	0.02°C	0.07°C	0x28
	Ni 1KΩ, 5.60Ω/°C	-50.0 to +129°C	0.025°C	0.08°C	0x29
Gage [5]		-500 to +500 mV	5µV [6]	30µV	0x0F
DC Voltage		-10 to +10V	500µV	600µV	0x09
		-5 to +5V [3]	500µV	600µV	0x00
		-5 to +5V	200µV	600µV	0x15
		-500 to +500mV	20µV	40µV	0x16
		-100 to +100mV	5µV	30µV	0x17
Current Loop [2]		4 to 20mA	0.01%	0.08%	0x11
Resistance		0 to 400Ω	0.02Ω	0.04Ω	0x0A
		0 to 3KΩ	0.125Ω	0.25Ω	0x14
		0 to 600KΩ	31Ω	130Ω	0x20
Disabled [4]		Not Applicable — sensor removed from scan loop			0x13

Notes

1. Measurement resolution and accuracy is specified for the default (versus high-speed) measurement mode. Derate resolution and accuracy by 75% when using the high-speed measurement mode.
2. Channels configured for 4-20mA current loops return data values representing a percentage of full-scale current (4mA= 0%, 20mA=100%). For example, a 12mA current would produce the data value 50.0, indicating 50% of full-scale.
3. This is the default sensor type after a board reset. It is provided for back-compatibility with earlier Sensoray Smart A/D products. Use of this type is not recommended for new designs, as higher resolution is available on the 5V, 200uV resolution range.
4. Declaring a channel's sensor type as Disabled removes the channel from the scan loop. Data values returned from disabled channels are indeterminate and should be ignored.
5. Gage specifications assume 4-wire bridge circuit, 120Ω minimum impedance, 10V excitation applied by sensor channel.
6. Use the following function to express gage measurement resolution in load units:

$$Resolution = \frac{RatedLoad}{(2 \times 10^6) \times RatedVoltageOut}$$

For example, take the case of a pressure gage that is rated at 3mV/V at 100 PSI full scale. The Smart A/D would be able to resolve a load to:

$$Resolution = \frac{100PSI}{(2 \times 10^6) \times 0.003V} = 0.017PSI$$

7. SDC is the Sensor Definition Code that is used by the SetSensorType command to declare a channel's sensor type. See Section 6.2.1 for details.