Sensoray Model 7429
STDbus Sensor Coprocessor

TABLE OF CONTENTS

Limited Warranty

Sensoray Company, Incorporated (Sensoray) warrants the model 7429 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 which 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 OR 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.

Special Handling Instructions

The Mode 7429 board contains CMOS circuitry that is sensitive to Electrostatic Discharge (ESD). Special care should be taken in handling, transporting, and installing the 7429 to prevent ESD damage to the board. In particular:

1. Do not remove the 7429 from its protective antistatic bag until you are ready to install it in your computer.
2. Handle the 7429 only at grounded, ESD protected stations.
3. Always turn off the computer before installing or removing the 7429 board.

INTRODUCTION

FUNCTIONAL DESCRIPTION

The 7429 interfaces sixteen process sensors directly to the STD bus. Each of the sixteen sensor channels may be independently configured via software to accept thermocouple, RTD, strain gage, thermistor, current loop, or DC voltage inputs.

Pulsed sensor excitation is provided for thermistors, RTD's, and strain gages in order to minimize self-heating effects. Cold junction compensation is provided for thermocouples on the optional 7409TB sensor screw termination boards, and all inputs are protected from high common mode voltages.

The 7429's onboard microprocessor continuously scans through the sixteen channels. As each channel is scanned, the channel sensor signal is digitized, linearized, and converted to engineering units appropriate for the sensor type. The most recent sensor data from each channel is immediately accessible to the host processor.

Communication with the host processor is achieved via two contiguous I/O ports which may be mapped anywhere in the STD bus I/O address space. Communication interrupts are provided to support the demands of high performance real time systems. Interrupts may be routed onto the STD bus INTRQ* line or directly to an external interrupt controller for more complex interrupt-driven systems.

SPECIFICATIONS

Electrical

Input power	±12.0 to ±15 VDC, ±35 mA +5.0 VDC, 100 mA
Operating temperature range	-25°C to 85°C for specified accuracy
CMRR	80dB minimum
A/D converter	16-bit integrating

Measurement time per channel	16.66 msec/channel
Channel settling time	5 msec
Strain/Pressure gage excitation	10VDC, pulsed
Resistive sensor excitation	13mA DC, pulsed (0 to 400¾) 5VDC in series w/4K¾ (other ranges)
Total time slot per channel	22msec
Input protection	Protection to 70 VAC common mode voltage

Sensors

Sensor Type	Range	Resolution	Accuracy	Noise, 1-sigma
Thermocouples				
E	-270C to 990°C	0.1C	0.2°C	
J	-210C to 760°C	0.1°C	0.2°C	
K	-270C to 1360°C	0.1°C	0.2°C	
T	-270C to 400°C	0.1°C	0.2°C	
S	0C to 1760°C	0.1°C	0.6°C	
R	0C to 1760°C	0.1°C	1.0°C	
Thermistor				
Omega 44006	-55C to 145°C	0.01°C	0.05°C	
or 44031				
RTD (100 ¾)				
0.385 ¾/C	-200C to 800°C	0.05°C	0.20°C	
0.392 ¾/C	-200C to 800°C	0.05°C	0.20°C	

Strain/Pressure (4-wire bridge)				
Š 120¾	-100 to +100 mV	5 µV	10 µV	
DC Voltage Input				
Range 1	-5 to +5V	200 µV	0.40 mV	0.75 mV
Range 2	-500 to +500 mV	20 µV	40 µV	63 µV
Range 3	-100 to +100 mV	5 µV	10 µV	63 µV
Current Loop				
Range 1	4 to 20 mA	0.01%	0.02%	
Resistance				
Range 1	0 to 400¾	0.02¾	0.04¾	0.16¾
Range 2	0 to 3K¾	0.125¾	0.25¾	
Range 3	0 to 600K¾	31¾	62¾	

HARDWARE CONFIGURATION

DEFAULTS

The 7429 requires installation of programming shunts to select various options such as I/O port addresses, interrupts, and individual channel filters. This chapter describes these configuration options.

After configuring the option shunts, the sensor coprocessor may be installed in you STDbus backplane and programmed as explained in the next chapter.

Defaults as shipped from the factory

Command/Data port	B2H
Status port	B3H
I/O space	primary
Data available interrupt	disabled
Alarm interrupt	disabled
STDbus interrupt enable	disabled
Individual channel filters	disabled

I/O SPACE

Two separate I/O spaces exist on the STD bus: *primary* and *expanded*. Option jumper **E21** selects which I/O space the 7429 board will reside in.

Install E21 to map the board into primary I/O space, or remove E21 to map the board into expanded I/O space. E21 is factory set to map the board into primary space.

I/O PORT MAPPING

The 7429 occupies two consecutive port addresses in the selected STD I/O space. These ports may be mapped to any address from 00H to FFH. Jumpers are factory installed to locate the board at ports B2H and B3H.

Jumpers **E20** through **E27** are used to select the 7429 I/O address. Installing a shunt at any of these positions will cause an address match when the corresponding address line is low (logic 0). Conversely, removing a shunt results in a match when the address line is high (logic 1).

For example, to decode the board at ports B4H and B5H (port B4H is the board's base address), install shunts on E25, E24, and E20, and remove shunts from E23, E27, E26, and E22. Note that the board base port always resides at an even address.

Use the following tables to determine which shunts to use for your desired addresses:

MSN	E23	E25	E27	E26
0	I	I	I	I
1	I	I	I	R
2	I	I	R	I
3	I	I	R	R
4	I	R	I	I
5	I	R	I	R
6	I	R	R	I
7	I	R	R	R
8	R	I	I	I
9	R	I	I	R
A	R	I	R	I
B	R	I	R	R
C	R	R	I	I
D	R	R	I	R
E	R	R	R	I
F	R	R	R	R

LSN	E24	E22	E20

0	I	I	I
2	I	I	R
4	I	R	I
6	I	R	R
8	R	I	I
A	R	I	R
C	R	R	I
E	R	R	R

I indicates shunt should be inserted

R indicates shunt should be removed

MSN is the most significant nibble of the port address

LSN is the least significant nibble of the port address

INTERRUPTS

The 7429 may be configured to interrupt the host processor in the event that data is available in the data port for access by the host or a channel alarm limit has been violated.

Host processor requests for sensor data are given top priority by the 7429 — data becomes available in sufficiently short time to satisfy many realtime situations. Even so, some applications demand maximum performance from host-to-7429 communications. Interrupts are the key to optimal communication rates.

Shunt **E17** is provided to select *data available* interrupts. Install E17 to enable *data available* interrupts, or remove E17 to disable this interrupt.

As you will discover in the programming chapter, alarm limits can be declared for each channel. Shunt **E19** may be configured to interrupt the host processor when a programmed limit value is exceeded. Install E19 to enable the *alarm* interrupt, or remove E19 to disable this interrupt.

Any interrupt request originating on the 7429 board must be routed to your system interrupt controller. The signal may be routed either externally or over the STD bus INTRQ* line.

Install shunt **E18** to route the interrupt request over the STD bus INTRQ* line.

Connector **J1** is available to direct the active low interrupt request to an external interrupt controller circuit.

DATA AVAILABLE interrupt	E17	enables *data available* interrupt

ALARM interrupt	E19	enables *alarm* interrupt
System interrupt enable	E18	routes interrupt request to STD bus

HARDWARE FILTERS

Each channel is provided with a hardware filter which may be used to filter noise from the sensor signal. This feature is in addition to the software filter algorithm.

Sometimes it is desirable to filter a differential input signal to roll off high frequency noise. For example, thermocouple wires might be pulled alongside high current cables capable of coupling noise onto the thermocouple. This noise can often be reduced by inserting the channel hardware filter into the input circuit of the corresponding thermocouple channel.

In other cases, however, hardware filtering produces undesireable side effects. For example, RTD's and thermistors are powered by a pulsed current source. Filtering signals from these sensors can result in inaccurate measurements.

Jumpers **E1** through **E16** are used to connect hardware filters to channels 0 through 15, respectively. Install the appropriate shunt to connect a channel hardware filter, or remove the shunt to disconnect the filter.

Sensoray recommends the following configurations for the varied sensors supported by the 7429:

Sensor Type	Filter
Thermocouples	Installed
Direct voltage	Installed
Strain gages	Removed
Thermistors	Removed
RTD's	Removed
4-20 mA source	Removed

PROGRAMMING

COMMUNICATION

Programming of the 7429 is achieved via a set of built-in commands which the 7429's onboard microcomputer is programmed to recognize. These commands are sent from the host processor to the 7429.

Some commands produce responses from the 7429; these responses are sent from 7429 to the host processor. The following sections describe the mechanism by which this bi-directional communication operates.

Programming Model

The coprocessor occupies two contiguous I/O ports in the STDbus I/O space. Both ports may be written to and read from, but each port has a distinctly different function for read and write operations.

Programming model

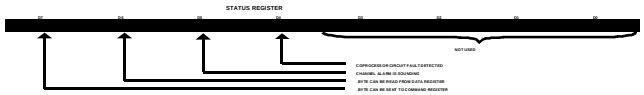
The base port (7429 board low port address) is the data port between 7429 and host processors. Commands are sent to the 7429 and command responses are passed back to the host through this port.

Physically, the data port consists of two hardware registers: *command* and *data*. When the host sends a command to the coprocessor, it is really storing a byte in the command register. When fetching a command response from the 7429, the host is reading a byte from the data register. Because the command register is write-only and the data register read-only, they can share the same port address.

The status port (7429 board high port address) may be read by the host to obtain coprocessor status information. This status information is discussed in the next section. When the host writes a byte into the status register, a hardware reset is invoked on the 7429. This can be useful during system software development. Data written to the status register is ignored.

Status Register

The status port provides the host with four status bits for monitoring various aspects of sensor coprocessor status. When the host inputs a byte from the status port, a byte of the following form is returned:



The CRMT and DAV bits are used for communication handshake control as described in the next section. During a coprocessor reset, CRMT is held low — this prevents the host from writing to the command register before the coprocessor is ready to communicate.

The ALARM bit indicates that one or more of the programmable channel limits was exceeded. This bit will not go to its active state until at least one channel limit value has been downloaded onto the coprocessor board. The ALARM bit is reset when the coprocessor is reset or when the *read alarm flags* command is executed (see *programming* chapter).

The FAULT bit indicates that the coprocessor self-test failed. This bit is set by either a system reset or soft reset by command from the host processor. As part of its initialization sequence the coprocessor performs a self-test, clearing the FAULT bit if all board functions are confirmed to be functioning properly. The self-test consumes approximately one-half second to complete. Note that the red LED near the top of the coprocessor board reflects the state of the FAULT bit.

Handshake Mechanism

Prior to writing a byte into the command register, the host must test the coprocessor status port CRMT bit. When CRMT contains a logic 1, the command register is ready to accept a new byte. The host should write to the command register only when CRMT contains a logic 1.

Similarly, the DAV status bit must be tested before reading a byte from the coprocessor data register. When DAV contains a logic 1 value, a new byte is available in the data register for reading by the host. The host should read from the data register only when DAV contains a logic 1.

Although these handshake rules are simple, failure to observe them will most likely result in communication errors.

Sample Drivers

We suggest incorporating procedures into the host processor's 7429 drivers which hide the communication handshake protocol from higher software layers. Sample QuickBasic drivers are shown below. These drivers are referenced in the programming examples throughout this manual.

```
'
' BasePort% is globally visible within this module
'
COMMON SHARED BasePort%'7429 base port address

'
' This subprogram establishes the 7429 base port address for subsequent
' driver calls.  Call this procedure before using any other sample
' drivers described in this manual.
'
SUB SetBasePort (Address%)
     BasePort% = Address%
END SUB

'
' This subprogram sends a byte to the 7429 command register:
'
SUB SendByte (ByteValue%)
     DO: LOOP WHILE (INP(BasePort% + 1) AND 128) = 0'wait for CRMT
     OUT BasePort%, ByteValue%                    'write command register
END SUB

 '
 ' This function reads a byte from the 7429 data register:
 '
FUNCTION ReadByte%
     DO: LOOP WHILE (INP(BasePort% + 1) AND 64) = 0'wait for DAV
     ReadByte% = INP(BasePort%)                   'read data register
END FUNCTION
```

You may also find the following procedures useful. The first procedure reads a 16-bit value from the coprocessor, adjusting the sign if necessary. The second procedure sends a 16-bit value to the coprocessor. Note that these routines call the above procedures.

```
'
' This function reads a 16-bit value from the coprocessor board.
'
FUNCTION ReadWord%
     Value& = ReadByte%(BasePort%)
     Value& = Value& * 256 + ReadWord%(BasePort%)
     IF Value& > 32767 THEN Value& = Value& - 65536&
     ReadWord% = Value&
END FUNCTION

'
' This subprogram sends a 16-bit value to the coprocessor board.
'
SUB SendWord (Value%)
     CALL SendByte(BasePort%, Value% \ 256)
     CALL SendByte(BasePort%, Value% AND 255)
END SUB
```

COMMANDS

The great versatility of the 7429 is due to its simple but powerful set of built-in commands. Commands vary in length depending on the need for supplementary parameters. Some commands cause the coprocessor to return data to the host processor. The first byte of each command typically adheres to this format:

Format of first command byte

Each command in the 7429's repertoire consists of at least one byte. Some commands require more than one byte because the information necessary to process the command will not fit into a single byte. Without exception, the first byte of each command contains a 4-bit opcode within the four most significant bits.

The remainder of this chapter discusses the coprocessor command set. Each command function is described, and the command and command-response byte strings affiliated with each command are detailed. These conventions are used in describing commands and responses:

1. A *byte* is represented by data contained in parenthesis.
2. Individual bits within a byte are separated by commas.
3. The suffix "H" on any datum designates hexadecimal notation.
4. *Byte strings* are depicted as bytes separated by commas.
5. A *command* consists of an ordered byte string to be written into the 7429's command register.
6. A command *response* consists of an ordered byte string to be read from the 7429's data register.
7. The ordering of bytes in any byte sequence (command or response) is sacred.
8. *Don't care* bits within a byte are designated by "x".

Define Sensor

This command declares the type of sensor connected to a particular channel. Each supported sensor type is assigned a unique 8-bit sensor definition code. See the sensor tables near the end of this manual for a list of valid sensor definition codes.

Typically, this command is executed once for each channel during the host processor initialization sequence (i.e., during power-on reset). Each execution declares the sensor type for a single channel; hence, it is necessary to execute the command sixteen times in order to set up all sixteen channels.

Any channel sensor having undeclared type will default to direct voltage in (500 µV/bit). You should not attempt to declare an invalid sensor type. There are three variations of this command — proper command form is dictated by the sensor type:

All forms of this command begin the same way. The first command byte contains the *define channel sensor* opcode and channel number. The second byte contains an appropriate sensor definition code (a unique 8-bit value assigned to each valid sensor type). This two-byte sequence is sufficient for most sensor types.

COMMAND: (16 + CHAN),(SENSOR DEFINITION CODE)

RESPONSE: NONE

Variation 2: Used to declare *custom resistive sensor* type (i.e., thermistors other than Omega 44006 type 10K). This form is used only when specifying sensor definition code 0CH. The 7429 will expect six additional bytes to follow the sensor definition code. These additional bytes constitute the coefficients of a polynomial designed to map sensor resistance into the desired output units. See the chapter on Custom Resistive Sensors for more detailed information.

COMMAND: (16 + CHAN),(0CH),
(HIGH COEFFICIENT MSB),(HIGH COEFFICIENT LSB),
(MID COEFFICIENT MSB),(MID COEFFICIENT LSB),
(LOW COEFFICIENT MSB),(LOW COEFFICIENT LSB)

RESPONSE: NONE

Variation 3: When declaring a *pressure gage/load cell* sensor type, the 7429 will expect six additional bytes to follow the sensor definition code. The first two bytes contain the gage mV/V rating times 10. The next two bytes specify the desired full-load output data value. The last two bytes declare the gage input impedance in ohms.

COMMAND: (16 + CHAN),(12H),
(V MSB),(V LSB),
(P MSB),(P LSB),
(R MSB),(R LSB)

RESPONSE: NONE

Example: A 350¾ pressure gage is to be connected to channel 7. The gage specification is 3 mV/V at 150 PSI. 3 (mV/V) x 10 = 30.

The application requires a resolution of one-tenth PSI. At full load, therefore, the 7429 should output a value of 150 PSI / 0.10 PSI = 1500. Since this is a 350¾ gage (the gage input impedance is 350¾), R = 350.

A suitable QuickBasic sequence would be:

```
CALL SendByte (16 + 7)'opcode + channel
CALL SendByte (&H12)'sensor definition code
CALL SendWord (30)'V
CALL SendWord (1500)'P
CALL SendWord (350)'R
```

Data will now be returned from the 7429 (when requested via the *read channel data* command) in units of 0.10 PSI/bit.

Note that pressure gage/load cell input impedance is not limited to 350¾. Any impedance greater than 120¾ is acceptable.

Example:

Configure channel 2 for interface to type K thermocouple. The QuickBasic sequence is:

```
CALL SendByte (16 + 2)'opcode + channel
CALL SendByte (3)'sensor definition code
```

Read Data

This command returns the most recently measured sensor data from the specified channel. The returned value is scaled according to the declared sensor type. Regardless of sensor type, the returned value is always represented in 16-bit two's complement form. Refer to the sensor tables near the end of this manual for a description of the data format used by each sensor type.

COMMAND: (CHAN)

RESPONSE: (HIGH DATA),(LOW DATA)

Example: Channel 6 is configured for the 5 Volt measurement range. This sequence will read and display the channel 6 voltage:

```
CALL SendByte (6)'opcode + channel
PRINT ReadWord% * .0002; " volts"
```

Example: Channel 0 is configured as a 350¾ strain gage rated at 2.7 mV/V at 150 PSI. Data scaling is programmed to 0.1 psi/bit. A suitable program segment to read and display channel 0 data:

```
CALL SendByte (0)'opcode + channel
PRINT ReadWord% * .1; " PSI"
```

Set Alarm Limits

This command declares upper and lower alarm limits for the specified channel. After declaring limits, an alarm will "sound" if sensor data for that channel strays outside of the range defined by the alarm limits.

For example, if a temperature sensor is being used to monitor the temperature of a milk storage tank, the upper alarm limit for that channel could be set to the highest "safe" temperature for the tank. If this "safe" limit is exceeded, the 7429 would notify the host so that appropriate action can be taken. This relieves the host of the burdensome task of reading the sensor temperature and comparing it to the limit value.

If limits are not specified for a channel the low limit defaults to -32768 and the high limit defaults to 32767.

Alarm limits may be changed via this command at any time. If only a lower limit is to be used, specify an upper alarm limit value of 32767. Similarly, specify a lower limit value of -32768 to use only an upper limit.

COMMAND: (32 + CHAN),
(HIGH LIMIT MSB),(HIGH LIMIT LSB),
(LOW LIMIT MSB),(LOW LIMIT LSB)

RESPONSE: NONE

Example: Channel 7 is connected to a type K thermocouple which is monitoring a process temperature. The temperature must not fall below 400°C.

For this example, assume that the upper temperature limit is of no concern. The 7429 scales K thermocouple to 0.1°C/bit, so the low limit value must also be scaled by the same factor. The upper limit is set to 32767 to disable the high alarm. The code segment is:

```
CALL SendByte (32 + 7)'command + opcode
CALL SendWord (32767)'high limit
CALL SendWord (400 / .1)'low limit
```

Read Alarms

This command returns the status the channel alarms. Alarm flags will be returned from either channels 0-7 or 8-15, depending on the command opcode used.

The first byte returned contains the status of the eight high alarms, and the second byte contains the status of the low alarms. Execution of this command will reset the status register alarm bit and clear the host processor alarm interrupt request.

Bit 7 (most significant bit) of each status byte corresponds to channel 7 or 15, bit 6 corresponds to channel 6 or 14, and so on for all other channels. If the bit corresponding to a channel is set (logic 1) then an alarm condition exists on that channel. If the bit is reset (logic 0) then the corresponding channel sensor is on the "safe" side of that channel's alarm limit.

COMMAND: (48)

RESPONSE: (CHANNEL 0-7 HIGH ALARMS),(CHANNEL 0-7 LOW ALARMS)

COMMAND: (49)

RESPONSE: (CHANNEL 8-15 HIGH ALARMS),(CHANNEL 8-15 LOW ALARMS)

Example: This code segment will display the channel alarm status on the system CRT:

```
PRINT "CHAN HIGH LOW"'print heading
CALL DisplayAlarms (48)'display channel 0-7 alarms
CALL DisplayAlarms (49)'display channel 8-15 alarms
. . . . . .

SUB DisplayAlarms (opcode%)
    CALL SendByte (opcode%)'opcode
    HiFlags% = ReadByte%'fetch high flag byte
    LoFlags% = ReadByte%'fetch low flag byte
    FOR channel% = 0 TO 7'print alarm list
        mask% = 2 ^ channel%
        PRINT channel% + 8 * (opcode% AND 1);
        IF (mask% AND HiFlags%) THEN PRINT "*"; ELSE PRINT "";
        IF (mask% AND LoFlags%) THEN PRINT "*"
    NEXT channel%
END SUB
```

Read Board Temperature

This command returns the temperature of the thermocouple cold-junction compensation sensors (if 7409TB termination boards are connected to the 7429). Depending on the command opcode used, temperature data will be returned from either the high 7409TB (channels 8-15) or low 7409TB (channels 0-7).

A 16-bit value is returned, scaled to 0.10°C/bit. Ordinarily, this command is used either as a diagnostic tool or to monitor the ambient temperature.

COMMAND: (64)

RESPONSE: (CHAN 0-7 7409TB TEMP MSB),(CHAN 0-7 7409TB TEMP LSB)

COMMAND: (65)

RESPONSE: (CHAN 8-15 7409TB TEMP MSB),(CHAN 8-15 7409TB TEMP LSB)

Example: Two 7409TB termination boards are connected to the 7429. This code segment will read and display the low 7409TB (connected to channels 0 through 7) temperature in degrees centigrade.

```
CALL SendByte (64)
PRINT "7409TB temperature = "; ReadWord% * .1; " degrees C"
```

Set Open Sensor Data Values

This command establishes the data value to be returned for each channel if an open sensor is detected. Depending on the form of the command, either channels 0-7 or 8-15 are affected.

Bits belonging to the second command byte are mapped to channels in the following way: bit 7 (most significant bit) to channel 7 or 15, bit 6 to channel 6 or 14, etc.

If a bit is set (logic 1) and the corresponding channel sensor is open, one of two data values will be returned by the *read channel data* command requesting data for that channel. If the bit is set (logic 1), the value 32767 will be returned. If the bit is reset, the value -32768 will be returned.

COMMAND: (80),(OPEN SENSOR FLAGS — CHANNELS 0-7)

RESPONSE: NONE

COMMAND: (81),(OPEN SENSOR FLAGS — CHANNELS 8-15)

RESPONSE: NONE

Example: An alarm is to sound if the K thermocouple on channel 7 fails. The thermocouple is regulating the temperature of an oven.

Assuming that we don't want the oven temperature to run away as a result of the sensor failure, the channel should be programmed to fail high so that the controller will shut off the heating control. This code segment will satisfy these requirements.

```
OpenFlags% = 128'flags:  (1,0,0,0,0,0,0,0)
CALL SendByte (80 + 7)'send opcode + channel
CALL SendByte (OpenFlags%)'send flags
```

Set Filter Time Constant

This command 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. This table shows some sample F values and the corresponding filter percentages.

Filter Factor	Filter Percentage
0	0.0
32	12.5
64	25.0
128	50.0
192	75.0
255	99.6

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

Some applications require knowledge of the filter time constant. This function describes the relationship between time constant and filter factor:

t = b(F(-N,45 •
lnB(F(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.

COMMAND: (96 + CHAN),(FILTER FACTOR)

RESPONSE: NONE

Example: The sensor on channel 4 is to have a 12.5% low-pass filter. A filter factor of 32 yields a 12.5% filter, so the command string is: (96 + 4),(32). Here is a sample QuickBasic implementation:

```
CALL SendByte (96 + 4)'opcode + channel
CALL SendByte (32)'filter factor
```

Assume that all sixteen channels are active. Using the above relationship between time constant and filter factor, it can be shown that the channel 4 software filter has a time constant of approximately 171 milliseconds.

Tare Gage

This command tares the strain/pressure gage connected to the specified channel. Gage taring is commonly used to subtract off the load offset created by the weight of an unloaded container. Offsets can also be caused by an imbalance in the gage bridge circuit.

Note that the *tare gage* command adjusts gage offset only and has no influence over gain (sometimes called "span"). Gage span may be altered by modifying the gage parameters specifed as part of the *define channel sensor* command.

After executing this command, all sensor data from the specified channel (accessed via the *read channel data* command) will be adjusted to compensate for the offset. Data following the first command byte is set to the current desired output value of the gage channel.

COMMAND: (112 + CHAN),(MSB DATA),(LSB DATA)

RESPONSE: NONE

Example: Channel 3 has been previously configured for a strain gage via the *define channel sensor* command. When configured as such, data scaling was established at 20 psi/bit. This particular gage exhibits a 60 psi offset at zero load, so it is necessary to correct for bridge imbalance when taring a load.

This code segment will tare the current gage load:

```
gageScalar% = 20
gageOffset% = 60 / gageScalar%
tareWeight% = 0 / gageScalar%
CALL SendByte (112 + 3)'opcode + chan
CALL SendWord (tareWeight% - gageOffset)'desired data value
```

Activate 50 Hz Rejection Filter

This command will cause the 7429 to reject 50 Hertz differential noise on all channels. Upon power-up reset, the 7429 will default to 60 Hertz rejection for all channels. This command should only be used in systems targeted for 50 Hertz power environments.

COMMAND: (128)

RESPONSE: NONE

Read All Channels

This command returns the most recently acquired data from a block of eight sensor channels. Depending on the command form used, data will be returned from either channels 0-7 or 8-15.

Sensor data is weighted as a function of the declared sensor type connected to each channel. Refer to the sensor tables near the end of the manual for sensor weights.

COMMAND: (144)

RESPONSE: (CH0 MSB),(CH0 LSB),
(CH1 MSB),(CH1 LSB),
(CH2 MSB),(CH2 LSB),
(CH3 MSB),(CH3 LSB),
(CH4 MSB),(CH4 LSB),
(CH5 MSB),(CH5 LSB),
(CH6 MSB),(CH6 LSB),
(CH7 MSB),(CH7 LSB)

COMMAND: (145)

RESPONSE: (CH8 MSB),(CH8 LSB),
(CH9 MSB),(CH9 LSB),
(CH10 MSB),(CH10 LSB),
(CH11 MSB),(CH11 LSB),
(CH12 MSB),(CH12 LSB),
(CH13 MSB),(CH13 LSB),
(CH14 MSB),(CH14 LSB),
(CH15 MSB),(CH15 LSB)

Example:This code segment will read data from all sixteen sensor channels into a data array:

```
CALL SendByte (144)'opcode:  read channels 0-7
FOR channel% = 0 to 7
    DataArray%(channel%) = ReadWord%
NEXT channel%

CALL SendByte (145)'opcode:  read channels 8-15
FOR channel% = 8 to 15
    DataArray%(channel%) = ReadWord%
NEXT channel%
```

Calibrate Board

This command is used to calibrate all 7429 internal standards. All measurements made by the coprocessor are referenced to these standards. To perform a board calibration, you must supply two reference voltages (5 volts and 500 millivolts) and one reference resistance (380 ohms).

References must be calibrated in the following order: 5 volts, 500 mV, and 380 ohms.

COMMAND: (224 + CHAN),(CAL CODE),(DATA HIGH),(DATA LOW)

RESPONSE: (GARBAGE)

Cal Code	Reference
0	5 volts
1	500 mV
2	380 ohms

Example: This code segment will assist you in calibrating the 7429.

```
FOR calCode% = 0 to 2

    SELECT CASE calCode%'setup cal parameters
        CASE 0
            refName$ = "5 volt"
            scalar% = 5000
            sensorCode% = 0
        CASE 1
            refName$ = "500 mV"
            scalar% = 50
            sensorCode% = 13
        CASE 2
            refName$ = "380 ohm"
            scalar% = 40
            sensorCode% = 9
    END SELECT

    INPUT "What channel is "; refName$; " reference connected to"; chan%

    CALL SendByte (16 + chan%)'declare channel sensor type
    CALL SendByte (sensorCode%)

    INPUT "What is exact value of "; refName$; " reference"; actualValue!

    CALL SendByte (224 + chan%)'issue calibrate command
    CALL SendByte (calCode%)
    CALL SendWord (actualValue! * scalar%)

    junk% = ReadByte%'wait for 7429 to calibrate

    OUT BasePort% + 1, 0'reset 7429 board
```

```
NEXT calCode%
```

CONNECTIONS

GENERAL

All sensors are connected to the 7429 through the 40-pin ribbon cable connectors P1 and P2. Optionally, sensors may be connected to the model 7409TB screw termination boards which in turn connect to the P1 and P2 connectors via 40-conductor ribbon cables. Sensors may be electrically connected and disconnected from the 7429 or 7409TB boards with power applied to the STDbus.

Each sensor has as many as five connections to a channel circuit. Two connections are universally used for all sensor types: SENSE+ and SENSE-. These two pins are the positive and negative differential voltage sense inputs, respectively. In addition to the sense pins, all resistive sensors (i.e., RTD's, thermistors, and strain/pressure gauges) require connection to the POWER+ and POWER- pins.

The POWER+ and POWER- pins provide positive and negative excitation, respectively, for all resistive sensors. This excitation is supplied in the form of a strobed DC voltage (~10V) for strain and pressure gages, and pulsed constant current for all other resistive sensors. The fifth lead is intended for connection to a cable shield.

The remainder of this chapter describes how to connect sensors to the 7429 for proper operation.

RTDS

RTD's can be connected to the 7429 in three possible configurations: two-wire, three-wire, and four-wire.

The simplest configuration is the two-wire circuit. As the name implies, this configuration requires only two wires. The SENSE+ and POWER+ pins are shorted together, and SENSE- and POWER- are shorted together. The drawback to this circuit is that a significant voltage may develop across the sense wires if you run a long cable out to the RTD. Depending on the length of cable, this voltage will introduce error into the RTD measurement. There are two solutions to this problem: use shorter cable or use more than two wires.

Two-wire connection

By using more than two wires it is possible to run one or both of the sense wires all the way out to the RTD. This enables the 7429 to sense RTD voltage at the RTD instead of at the 7429 connector. Since the SENSE pins are high impedance inputs, virtually

no current passes through the sense lines, therefore no voltage develops across the sense wires.

Three-wire connection

Using three wires has the advantage of eliminating current through one of the two sense leads. This yields a 50% reduction in lead-loss errors compared to the two-wire circuit. Using four wires eliminates all error due to lead-loss.

We recommend use of the four-wire circuit using four-conductor shielded cable. You should always use shielded cable for RTD's regardless of which circuit you choose for your application. Make sure you connect the cable shield only to the channel SHIELD pin.

Four-wire connection

THERMOCOUPLES AND DC VOLTAGE

Thermocouple and DC voltage inputs are connected directly to the channel SENSE+ and SENSE- pins. The positive wire is connected to SENSE+, while the negative wire is connected to SENSE-.

DC voltage connections

There is no need for thermocouples to connect to the POWER+ or POWER- pins because thermocouples contain their own built-in power source (i.e., the Seebeck voltage).

Thermocouple wires are color coded to indicate polarity. The positive thermocouple wire should always be connected to the SENSE+ pin, and the negative thermocouple wire should always be connected to the SENSE- pin. The following table lists the wire color coding for all 7429 supported thermocouple types.

TC Type	Positive	Negative
E	purple	red
J	white	red
K	yellow	red
T	blue	red
R	black	red
S	black	red

Thermocouple connections

THERMISTORS

Thermistors (type 10K) have much higher resistance values than RTD's over most of their operating range, therefore a connection identical to the RTD two-wire configuration is recommended for most applications.

If you are using a thermistor to measure temperatures near the high end of it's range, however, it may be advisable to implement a three- or four-wire hookup to prevent error due to lead-loss effects. As with RTD's, thermistors should be connected using shielded cable. The shield should be connected only to the 7429 channel SHIELD pin.

STRAIN AND PRESSURE GAGES

In a typical strain/pressure gage, four wires connect the gage to the gage interface system. Two wires supply gage excitation, while the other two wires carry the gage output signal. The 7429 is designed to support gages requiring constant voltage excitation.

Gage input impedances may range from 120 ohms on up, although most standard gages have nominal impedances of either 120 or 350 ohms. Like RTD's, strain and pressure gages should use shielded cable. The shield should be connected only to the channel SHIELD pin on the 7429.

Strain/pressure gage connections

4 TO 20 MILLIAMP CURRENT LOOPS

Coprocessor firmware supports 4-20 mA current source inputs on any channel; however, a 250-ohm 0.01% resistor must be installed on the channel inputs as shown in the following diagram. These resistors are available as an option for the 7429. Order Sensoray PN 7408R.

4-20 mA current loop connections

CUSTOM RESISTIVE SENSORS

The 7429 possesses the capability of interfacing custom resistive sensors to any sensor channel with the following limitations:

1. The maximum sensor resistance must not exceed 600K ohms.
2. You must provide three 16-bit integers which serve as coefficients of a second-degree sensor linearization polynomial.
3. A second-degree sensor linearization polynomial is sufficiently accurate for your application.

Custom resistive sensors are interfaced as follows:

1. If empirical sensor data is available, apply a least-squares fit (or other curve-fit) algorithm to the data to obtain a polynomial of the form

$$y = AR^2 + BR + C$$

where: R is the sensor resistance (ohms)
y is the channel data value in with appropriate scaling
A, B, and C are coefficients derived from the curve fit

If an equation is already available for the sensor, the equation can usually be worked into the form of the above expression via various numerical methods (if not already in the correct form).

2. Install the following command into your host processor's initialization sequence to define the custom sensor (see DEFINE CHANNEL SENSOR command):

(16 + CHAN),(0CH),
(A MSB),(A LSB),
(B MSB),(B LSB),
(C MSB),(C LSB)

SAMPLE APPLICATION

A displacement transducer is to be interfaced to the sensor coprocessor on channel 5. The sensor DC resistance varies as a function of displacement. The following data points have been measured:

Resistance (ohms)	Displacement (mm)
0	-3105
50	245
100	8595
150	21945

A least-squares fit algorithm is applied to the transducer data with the following results:

$$\mathbf{D} = 1\mathbf{R}^2 + 17\mathbf{R} - 3105$$

where:**D** is the displacement in millimeters
R is the transducer resistance in ohms

This QuickBasic code segment invokes the *define channel sensor* command for this resistive linear displacement transducer:

```
CALL SendByte (16 + 5)'DEFINE CHANNEL SENSOR opcode + channel
CALL SendByte (&HC)'sensor definition code
CALL SendWord (1)'A coefficient
CALL SendWord (17)'B coefficient
CALL SendWord (-3105)'C coefficient
```

After defining the sensor, invoking the READ SENSOR DATA command for channel 5 will return the displacement value scaled to 1 mm/bit. This code segment reads and displays the displacement:

```
CALL SendByte (5)
PRINT "Channel 5 displacement (mm) = "; ReadWord%
```

THERMOCOUPLE THEORY

Thermocouples are typically formed from two wires having dissimilar metals. The wires are built into a cable having a heat-resistant sheath and often have a metallic shield. At one end of the cable, the wires are electrically shorted together by crimping, welding, etc. This end of the thermocouple — the *hot junction* — is thermally attached to the measurement point. The other end — the *cold junction* — is connected to a measuring device. In this case, the measuring device is a 7429 coprocessor board.

A thermocouple generates an open-circuit voltage which is proportional to the temperature gradient between the hot and cold junctions (Re: the Seebeck effect). Since the thermocouple voltage is a function of the temperature difference between junctions, it is necessary to know the cold junction temperature in order to determine the temperature at the hot junction.

The sensor coprocessor periodically measures the temperature at the 7409TB termination board (optional accessory). Since all thermocouples are terminated at this board, thermocouple cold junctions will be very near to the temperature of the termination board. A correction voltage is computed for all supported thermocouple types based on the measured temperature. This correction voltage is the *cold junction compensation voltage*.

The cold junction compensation voltage is added to the differential voltage measured at the thermocouple cold junction. The result of this addition — the *corrected thermocouple voltage* — is the voltage one would measure if the cold junction were maintained at zero degrees C. The corrected thermocouple voltage is then converted to the appropriate temperature units using a non-linear mapping function.

COMMAND SUMMARY

DEFINE CHANNEL SENSOR - declare sensor type connected to a channel

Custom resistive sensors only:

command: (16 + CHAN),(SENSOR DEFINITION CODE),
(HIGH MSB),(HIGH LSB),
(MID MSB),(MID LSB),
(LOW MSB),(LOW LSB)

response: NONE

Strain/pressure gages only:

command: (16 + CHAN),(SENSOR DEFINITION CODE),
(V HIGH),(V LOW),
(P HIGH),(P LOW),
(R HIGH),(R LOW)

response: NONE

All other sensors:

command: (16 + CHAN),(SENSOR DEFINITION CODE)

response: NONE

READ CHANNEL DATA - read linearized sensor data from a single channel

command: (CHAN)

response: (HIGH DATA),(LOW DATA)

SET CHANNEL ALARM LIMITS - specify sensor limits for a single channel

command: (32 + CHAN),
(HIGH LIMIT MSB),(HIGH LIMIT LSB),
(LOW LIMIT MSB),(LOW LIMIT LSB)

response: NONE

READ ALARM FLAGS - read alarm status for all channels

Channels 0-7:

command: (48)

response: (HIGH ALARMS),(LOW ALARMS)

Channels 8-15:

command: (49)

response: (HIGH ALARMS),(LOW ALARMS)

READ BOARD TEMPERATURE - read 7409TB temperature

7409TB (channels 0-7):

command: (64)
response: (TEMP MSB),(TEMP LSB)

7409TB (channels 8-15):

command: (65)
response: (TEMP MSB),(TEMP LSB)

SET OPEN SENSOR VALUES - establish data value in event of open sensor

Channels 0-7:

command: (80),(OPEN SENSOR FLAGS)
response: NONE

Channels 8-15:

command: (81),(OPEN SENSOR FLAGS)
response: NONE

SET FILTER TIME CONSTANT - specify filter constant for a single channel

command: (96 + CHAN),(FILTER CONSTANT)
response: NONE

TARE GAGE - specify gage tare weight

command: (112 + CHAN),(MSB DATA),(LSB DATA)
response: NONE

50 HERTZ FILTER - set A/D sample interval for 50 Hz environment

command: (128)
response: NONE

READ ALL CHANNELS

Channels 0-7:

command: (144)
response: (CH0 MSB),(CH0 LSB),
(CH1 MSB),(CH1 LSB),
(CH2 MSB),(CH2 LSB),
(CH3 MSB),(CH3 LSB),
(CH4 MSB),(CH4 LSB),
(CH5 MSB),(CH5 LSB),
(CH6 MSB),(CH6 LSB),
(CH7 MSB),(CH7 LSB)

Channels 8-15:

command: (145)
response: (CH8 MSB),(CH8 LSB),
(CH9 MSB),(CH9 LSB),
(CH10 MSB),(CH10 LSB),
(CH11 MSB),(CH11 LSB),
(CH12 MSB),(CH12 LSB),
(CH13 MSB),(CH13 LSB),
(CH14 MSB),(CH14 LSB),
(CH15 MSB),(CH15 LSB)

SENSOR TABLES

Sensor definition codes and *data scale factors*:

SENSOR TYPE	CODE	DATA SCALING per bit
Voltage		
±100 mV f.s.	17H	5 µV
±500 mV f.s	16H	20 µV
±5 V f.s	15H	200 µV
Resistance		
400¾ f.s.	09H	0.02¾
3K¾ f.s.	0AH	0.125¾
600K¾ f.s.	20H	31¾
Custom resistive sensor	0CH	* defined by programmer
Thermocouple		
E	01H	0.10°C
J	1BH	0.10°C
K	1CH	0.10°C
T	1DH	0.10°C
S	1EH	0.10°C
R	1FH	0.10°C
RTD		
385	18H	0.05°C
392	19H	0.05°C
Thermistor		
Omega 44031, 10K¾	1AH	0.01°C
Current loop		

4-20 mA	11H	0.01% (4mA=0%, 20mA=100%)
Gage (pressure/strain)		
S120¾	12H	* defined by programmer
Disabled channel	13H	* not applicable

BACK COMPATIBILITY

In addition to the sensor definition codes listed above, several additional codes exist to maintain compatibility with Sensoray model 7408 and 7409 products. Use of these codes is not recommended for new designs:

SENSOR TYPE	CODE	DATA SCALING per bit
Voltage		
0 to 5 V f.s	00H	500 µV
0 to 1.65 V f.s.	0EH	100 µV
0 to 80 mV f.s.	0DH	10 µV
RTD		
385	07H	0.1°C
392	08H	0.1°C
Thermocouple		
J	02H	0.11°C
K	03H	0.17°C
T	04H	0.15°C
S	05H	0.60°C
R	06H	0.50°C
Thermistor		
Omega 44031, 10K¾	0BH	0.02°C

TIMING

Three timing parameters are important from a system integration viewpoint: channel scan rate, communication latency, and host processor speed. These timing elements are considered below.

CHANNEL SCAN RATE

Channel scan rate is defined as the number of conversions per channel per second. The scan rate is influenced primarily by the number of active channels in the scan loop. A secondary influence is the frequency of communication between host and 7429 processors.

The basic channel processing time is approximately 22 milliseconds. This is the amount of time required for the 7429 to configure its analog section, allow the front end to settle, and digitize the input signal. Note that linearization and filtering functions don't require any additional time as they execute concurrently with the digitization.

Clearly, the scan time decreases as the number of active channels decrease. Channels can be removed from the scan loop by disabling them with the *define channel sensor* command, using the *disabled channel* sensor definition code. The update rate for each channel is given by this function:

R Ý F(45,N)

R = samples/second
N = number of active channels

All host communication functions are interrupt-driven in the 7429's local environment. As a result, both communication latency and overhead are kept to a minimum. Even so, time spent communicating with the host is often time stolen from the scanning function. Keep in mind that excessive communication traffic between host and 7429 will tend to reduce the scan rate.

COMMUNICATION LATENCY

Communication latency is defined here as communication delays between host and 7429. Perhaps the most important measure of latency is the amount of time elapsed from a request for sensor data to the acquisition of that data. This is called the *acquisition latency*.

Acquisition latency can be viewed as having two parts: time from request to first response byte — *command response delay* — and time between response bytes — *data queue delay*.

Two commands return sensor data to the host processor, *read all channels* and *read channel data*. The worst case command response delay is the same for both commands: 140 microseconds. The worst case data queue delay is 40 microseconds.

The *read channel data* command consumes a maximum of 180 usec. This is computed as follows: 140 us from command issue to availability of the first response byte, plus 40 us from the time the host reads the first byte to availability of the second response

byte. This analysis ignores any additional time consumed by the host processor.

The *read all channels* command consumes 740 microseconds maximum as follows: 140 us command response delay plus 15 more data bytes with 40 us data queue delay.

Notice that reading a block of eight channels is faster when using the *read all channels* command rather than invoking the *read channel data* command eight times.

PROCESSOR SPEED

Host processor speed is limited only by the bus interface circuitry on the sensor coprocessor board. The bus interface is designed to work with hosts operating at up to 16MHz. If you have a faster processor or a heavily loaded backplane, you may need to insert wait states into your CPU's I/O cycles for reliable operation.

BOARD LAYOUT