Sensoray Model 417
ISAbus Thermocouple Interface

July 13, 2000

Table of Contents

Chapter

1

Basics

Limited Warranty

Sensoray Company, Incorporated (Sensoray) warrants the Model 417 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 existance 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 Model 417 circuit board contains CMOS circuitry that is sensitive to Electrostatic Discharge (ESD). Special care should be taken in handling, transporting, and installing the 417 in order to prevent ESD damage to the board. In particular:

1. Do not remove the 417 from its protective antistatic bag until you are ready to configure the board for installation.
2. Handle the 417 only at grounded, ESD protected stations.
3. Remove power from the ISAbus before installing or removing the 417 circuit board.

Chapter

2

Introduction

Functional Description

The Model 417 circuit board interfaces 32 process sensors directly to the ISAbus. Each of its 32 sensor channels may be independently configured, via software, to accept thermocouples, 4-to-20 mA current loops, or DC voltage. Sensor channels utilize true differential inputs to allow for wide common mode variation.

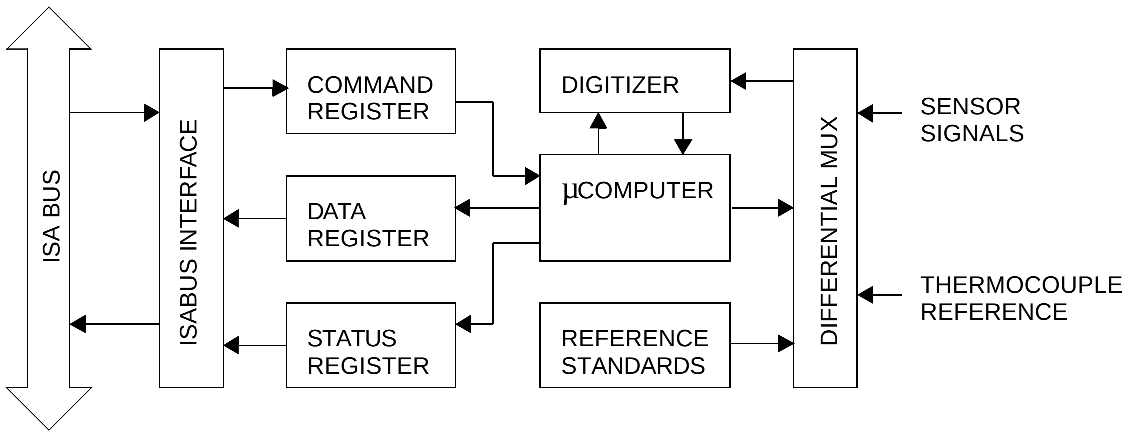
Cold junction compensation and open-sensor detection is provided for thermocouples, and all inputs are protected from high differential and common mode voltages.

Under firmware control, an onboard microprocessor continuously scans the 32 channels. As each channel is scanned, the sensor signal is digitized, linearized, and converted to engineering units as appropriate for the declared sensor type. Each channel's most recently acquired data may be quickly accessed by the ISAbus host.

Alarm limits are programmable for each channel. A status flag is set when limit violations occur, enabling the host to quickly detect alarm conditions.

Communication with the ISAbus host takes place over two contiguous ports which may be mapped anywhere in the system I/O address space. Local communication and scanning tasks execute in a high-performance multitasking environment, optimizing both throughput and host communication latency.

Block Diagram



Specifications

Table 1: General Specifications

Parameter	Condition	Specification			
		Min	Typ	Max	Units
DC Input power	+5V, ±5%		79		mA
	+12V, ±5%		32		
	-12V, ±5%		22		
Operating temperature		-25		+85	°C
Common-mode rejection	$V_{CM} < 5V$ $f < 1KHz$		80		dB
Input Impedance		0.95	1.0		MΩ
Noise	3 sigma		5.7	7.5	µV
Channel measurement time			16.67		msec
Channel total time slot				22	
Input overvoltage (with or without applied pwr)	2 sec. max	-70		+70	V
	continuous	-20		+20	
A/D converter	17.5-bit integrating				
Data format	16-bit integer, signed				
ISAbus Interface	Two contiguous 8-bit I/O ports Full 16-bit address decode				

Table 2: Sensor Specifications

Sensor Type		Range	Resolution	Accuracy
Thermocouple	B	0°C to 1820°C	0.1°C	3.3°C
	C	0°C to 1820°C	0.1°C	2.1°C
	E	-270°C to 990°C	0.1°C	0.8°C
	J	-210°C to 760°C	0.1°C	0.6°C
	K	-270°C to 1360°C	0.1°C	1.0°C
	N	-270°C to 1347°C	0.1°C	0.9°C
	T	-270°C to 400°C	0.1°C	0.6°C
	S	0°C to 1760°C	0.1°C	3.0°C
	R	0°C to 1760°C	0.1°C	2.8°C
DC Voltage		0 to +5V	200µV	600µV
		0 to +500mV	20µV	40µV
		0 to +100mV	5µV	30µV
Current Loop		4-to-20 mA	0.01%	0.08%

Chapter

3 Hardware Configuration

Base Address

Before you install the coprocessor board in your system, its base I/O address must be set. The base address may be set to any even value in the range 0000 to FFFE (hexadecimal) by means of option shunts A1 through A15. These option shunts are factory set to locate the board at address 02B2 and 02B3. To avoid conflicts, you must map the 417 into a two-port address range that is unoccupied by other devices.

Option Shunt Table

If you require the 417 board to have a base address different from the default address , use the following table to help to determine the proper shunt settings:

Table 3: I/O address jumpers: ● = install shunt, ❍ = remove shunt

Nibble Value	High Address Byte								Low Address Byte						
	Nibble3 (high)				Nibble2				Nibble1				Nibble0 (low)		
	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1
0	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
1	●	●	●	❍	●	●	●	❍	●	●	●	❍			
2	●	●	❍	●	●	●	❍	●	●	●	❍	●	●	●	❍
3	●	●	❍	❍	●	●	❍	❍	●	●	❍	❍			
4	●	❍	●	●	●	❍	●	●	●	❍	●	●	●	❍	●
5	●	❍	●	❍	●	❍	●	❍	●	❍	●	❍			
6	●	❍	❍	●	●	❍	❍	●	●	❍	❍	●	●	❍	❍
7	●	❍	❍	❍	●	❍	❍	❍	●	❍	❍	❍			
8	❍	●	●	●	❍	●	●	●	❍	●	●	●	❍	●	●
9	❍	●	●	❍	❍	●	●	❍	❍	●	●	❍			
A	❍	●	❍	●	❍	●	❍	●	❍	●	❍	●	❍	●	❍
B	❍	●	❍	❍	❍	●	❍	❍	❍	●	❍	❍			
C	❍	❍	●	●	❍	❍	●	●	❍	❍	●	●	❍	❍	●
D	❍	❍	●	❍	❍	❍	●	❍	❍	❍	●	❍			
E	❍	❍	❍	●	❍	❍	❍	●	❍	❍	❍	●	❍	❍	❍
F	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍			

Example

Set the base address to 0390:

Nibble3 = 0				Nibble2 = 3				Nibble1 = 9				Nibble0 = 0		
A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1
●	●	●	●	●	●	❍	❍	❍	●	●	❍	●	●	●

Nibble3 = 0, so install A15-A12.
Nibble2 = 3, so install A11-A10 and remove A9-A8.
Nibble1 = 9, so install A6-A5 and remove A7 and A4.
Nibble0 = 0, so install A3-A1.

Application Worksheet

Use this worksheet to record and configure the Model 417 base address for your application.

1. Enter your target base address, in hexadecimal, in the space provided below:

Base Address:	

2. Copy the hexadecimal address digits from the above box into the corresponding nibbles in the spaces provided below. As in the example shown above, and using the Option Shunt Table as a guide, darken in the circles representing option shunts that must be installed for each nibble.

Nibble3 =				Nibble2 =				Nibble1 =				Nibble0 =		
A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1
❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍	❍

3. Program the Model 417 circuit board option shunts as shown in the above table.

Chapter 4

Programming

The Model 417 is programmed by means of commands. Commands are sent from the ISAbus master—the "host" processor—to the 417. Some commands cause responses from the 417, which are sent from 417 to host. This chapter explains the 417-to-host communication mechanism and describes the command set.

Programming Model

The coprocessor occupies two consecutive port addresses in ISAbus I/O space. Each port has a unique function for read and write operations.

The base port (417 board low port address) forms the data path between 417 and host. Commands are sent to the 417 and responses are passed back to the host through this port, which consists of two physical hardware registers: *command* and *data*. When the host sends a command to the 417, it is really storing a byte in the command register. When the host reads a command response from the 417, it is reading a byte from the data register. Because the command register is write-only and the data register read-only, they are able to share the same I/O address.

The control port (base address + 1) consists of a read-only status register and write-only control register. As the name implies, the status register supplies coprocessor status information to the host. The control register is used to invoke soft resets.

Table 4: Programming model

Port Address	Read	Write
Base + 0: Data	Data Register	Command Register
Base + 1: Control	Status Register	Control Register

Status Register

The status register provides the host with information needed for status monitoring and communication handshake control. When the host reads the status register, a byte of the following form is returned:

Table 5: Status Register

D7	D6	D5	D4	D3	D2	D1	D0
CRMT	DAV	ALARM	FAULT	X (DISREGARD THESE BITS)			

Status register bits have the following functions:

Table 6: Status Register bit functions

Bit	Function
`CRMT`	**C**ommand **R**egister e**M**p**T**y, when set, indicates the 417 is ready to accept a byte into its command register. Before writing a byte to the command register, the host must test `CRMT` and verify that it is set to logic 1.
`DAV`	**D**ata **AV**ailable, when set, indicates that the 417 data register contains a new data byte for the host. Before reading a byte from the data register, the host must test `DAV` and verify that it is set to logic 1.
`ALARM`	When active (logic 1), indicates one or more programmed channel limits were exceeded. Activated by any limit violation. Cleared by executing the *Read Alarms* command.
`FAULT`	Indicates (1) a board reset is in progress, or (2) a board fault has been detected. In normal operation `FAULT` is active (logic 1) for approximately one-half second following a board reset. `FAULT` reflects the state of the visible red LED fault indicator on the Model 417 circuit board. **IMPORTANT NOTE**: The `CRMT`, `DAV`, and `ALARM` flags are not valid when `FAULT` is active. After resetting the 417 board by means of either soft or hard reset, the host should not attempt to handshake with the 417 until `FAULT` changes to its inactive (logic 0) state.

Control Register

The control register provides a means for forcing a coprocessor soft reset. A soft reset is invoked when any value is written to the control register port. The value of the data written to the control port is ignored. A soft reset has the same effect on the coprocessor as a hard system reset.

Reset State

Following a soft or hard reset, the board state is initialized as follows:

- All channel sensor types switch to the 5-Volt, 500uV/bit range.
- All channel filter factors are reset to zero.
- All channels are programmed to fail high on open-sensor detection.
- All channel alarm limits are set to min/max values that disable the alarms.
- 60 Hertz Rejection Mode is active.
- Low-speed/High-resolution Mode is active.

Sample Drivers

We recommend that you incorporate procedures in your application software that will conceal the coprocessor handshake protocol from higher host software levels. This will serve to simplify your programming requirements and reduce software maintenance costs.

This section contains simple software drivers that implement the required handshake protocol. In addition, the Visual Basic routines listed below are referenced throughout this chapter in numerous programming examples. Feel free to plagiarize and adapt these routines for your application.

Visual Basic Drivers

Note to Microsoft Visual Basic programmers: the Visual Basic programming language does not support direct I/O access. You will need a third-party tool, or a simple DLL of your own design, to access the Model 417 from within a Visual Basic application.

All Visual Basic code examples assume the availability of external procedures, “INP” and “OUT,” for implementing direct I/O access.

```
‘*******************************************************************
‘ Status Port mask constants
‘*******************************************************************

Const CRMT = &H80     ‘Command Register eMpTy
Const DAV = &H40      ‘Data AVailable
Const ALARM = &H20    ‘Limit alarm(s) sounding
Const FAULT = &H10    ‘Board fault

‘*******************************************************************
‘ Storage declarations
‘*******************************************************************

Dim BaseAdrs As Integer      ‘ Base address of 417 board

‘*******************************************************************
‘ Handshake a byte into the 417 command register
‘*******************************************************************
Sub SendByte (BasePort As Integer, CommandByte As Integer)
   Do: Loop Until (INP(BasePort + 1) And CRMT)     ‘wait for CRMT
   Call OUT(BasePort, CommandByte)                 ‘send the byte
End Sub

‘*******************************************************************
‘ Handshake a byte from the 417 data register
‘*******************************************************************
Function ReadByte%(BasePort As Integer)
   Do: Loop Until (INP(BasePort + 1) And DAV)      ‘wait for DAV
   ReadByte = INP(BasePort)                        ‘read the byte
End Function
```

```
'*******************************************************************
' Read a signed, 16-bit integer value from the 417, MSB first.
'*******************************************************************
Function ReadWord%(BasePort As Integer)

   Dim Lval As Long

   ' Handshake the high byte (MSB) from Model 417
   Lval = ReadByte(BasePort)

   ' Handshake LSB from Model 417 & concatenate with high byte
   Lval = Lval * 256 + ReadByte(BasePort)

   ' Adjust sign, if necessary
   If Lval > 32767 Then Lval = Lval - 65536

   ' Set return value
   ReadWord = Lval

End Function


'*******************************************************************
' Send a signed, 16-bit integer value to the 417, MSB first.
'*******************************************************************
Sub SendWord(BasePort As Integer, Value As Integer)

   ' Handshake the high byte (MSB) to Model 417
   Call SendByte(BasePort, Value \ 256)

   ' Handshake the low byte (LSB) to Model 417
   Call SendByte(BasePort, Value)

End Sub
```

C Language Drivers

```
//======================================================
// Write a command byte to the 417 command register.
//======================================================

void adSendByte ( short base_port, short cmd_byte )
{
   while ( ( inp( base_port + 1 ) & 128 ) == 0 ) ;    // wait for CRMT
   outp ( base_port, cmd_byte ) ;                     // send the byte
}


//======================================================
// Read a data byte from the 417 data register
//======================================================

short adReadByte ( short base_port )
{
   while ( ( inp ( base_port + 1 ) & 64 ) == 0 ) ;    // wait for DAV
   return ( inp ( base_port ) ) ;                     // read the byte
}


//======================================================
// Write 16-bit value to the 417
//======================================================

void adSendByte ( short base_port, short cmd_word )
{
   adSendByte ( base_port, cmd_word >> 8 ) ;          // send high byte
   adSendByte ( base_port, cmd_word ) ;               // send low byte
}


//======================================================
// Read 16-bit value from the 417
//======================================================

short adReadWord ( short base_port )
{
   short hiByte = adReadByte ( base_port ) << 8 ;     // read hi byte
   return ( hiByte | adReadByte ( base_port ) ) ;     // rd/concat lo byte
}
```

80x86 Assembly Language Drivers

```
;***********************************************************
; Handshake a command byte into the 417 command register.
; Entry:DX points to 417 base port address.
;     AL contains command byte to send to 417.
;***********************************************************

                   ;INITIALIZE DRIVER
XMIT:  PUSH AX     ;  Save command byte to be sent to 417
       INC DX      ;  Set address pointer to status port
                   ;WAIT FOR “COMMAND REGISTER EMPTY”
XLOOP: IN AL,DX    ;  Read status port
       TEST AL,80H ;   and test CRMT status flag
       JE XLOOP    ;  Loop until command register is empty
                   ;SEND COMMAND BYTE TO 417 BOARD
       DEC DX      ;  Set address pointer to command register
       POP AX      ;  Restore command byte
       OUT DX,AL   ;   and write it into command register
       RET         ;EXIT DRIVER


;***********************************************************
; Handshake a byte from the 417 data register.
; Entry:DX points to 417 base port address.
; Exit:AL contains data byte read from 417.
;***********************************************************

                   ;INITIALIZE DRIVER
RCV:   INC DX      ;  Set address pointer to status port
                   ;WAIT FOR “DATA AVAILABLE”
RLOOP: IN AL,DX    ;  Read 417 status port
       TEST AL,40H ;   and test DAV status flag
       JE RLOOP    ;  Loop until data register is full
                   ;READ DATA BYTE FROM 417 BOARD
       DEC DX      ;  Set address pointer to 417 data register
       IN AL,DX    ;  Read byte from data register
       RET         ;EXIT DRIVER
```

Commands

The 417 is accessed through a simple yet powerful built-in command set. Commands vary in length depending on the size of encapsulated data. In some cases invocation of a command will cause the coprocessor to return data to the host. In such cases the host must read the command response before issuing another command.

Table 7: Format of first command byte

D7	D6	D5	D4	D3	D2	D1	D0
OPCODE			CHANNEL				

Each command consists of at least one byte. The first byte of each command adheres to the above format. The opcode—always present in the first command byte—specifies the function to be performed. A channel number, present in the Channel field, is required by commands that address a particular channel. In other commands, the Channel field is sometimes used as an extension of the Opcode field.

Definitions

The following definitions and conventions are used throughout this chapter when describing commands and responses:

- The symbolic name “`Chan`” represents a coprocessor channel number. Valid channel numbers range from 0 through 31, inclusive.
- A *byte* is represented as a number or expression contained by parenthesis. For example, `(16 + Chan)` represents a byte having a value equal to 16 plus a channel number.
- A *byte string* is represented as an ordered sequence of bytes, separated by commas. The sequence is defined from left to right. For example, the byte string `(95 + Chan),(5),(0)` contains three bytes and begins with the leftmost byte.
- A *command* consists of a byte string which is passed from host to 417.
- A command *response* consists of a byte string which is passed from 417 to host.

Command Set

The remainder of this chapter discusses the coprocessor command set. Each command function is described along with the associated command and response byte strings.

Sample code snippets are provided to illustrate command usage. All programming examples are written in Visual Basic. If you are employing a different developer’s environment for your application, these programming examples can be easily converted to the language of your choice.

Set Sensor Type

This command declares the type of sensor connected to a channel. Typically, this command is executed once for each channel during the coprocessor initialization sequence (i.e., after a hard or soft reset). Each execution of this command specifies the sensor type for a single channel, therefore 32 invocations must occur to declare sensor types for all 32 channels.

A two-byte sequence forms the command string. The first byte contains the *Set Sensor Type* command opcode and target channel number. The second byte contains a Sensor Definition Code (SDC). Each sensor type is assigned a unique 8-bit SDC. Refer to the *Sensor Table* at the end of this chapter for a complete list of sensor definition codes.

Notes:

1. Undeclared channel sensor types default to the 5-Volt input range, SDC 0.
2. Declaring an invalid sensor type may result in unpredictable behavior.
3. Declaring a sensor type as “Disabled” inhibits scanning of the corresponding channel, resulting in increased throughput for all remaining active channels.
4. Sensor data is not available immediately after declaring a channel’s sensor type; you must wait for one complete scan loop before valid sensor data can be accessed.

COMMAND: (32 + Chan), (SensorDefinitionCode)

RESPONSE: NONE

VB Code Example:

```
‘**********************************************************
‘ Generic procedure used to declare any sensor type.
‘**********************************************************
Sub SetSensorType(BasePort%, Channel%, TypeCode%)

   Const SET_TYPE = 32   ‘ SET SENSOR TYPE command opcode

   ‘ Send the command string to the coprocessor board
   Call SendByte(BasePort, SET_TYPE + Channel)
   Call SendByte(BasePort, TypeCode)

End Sub

‘**********************************************************
‘ Configure channel 2 for interface to K thermocouple.
‘**********************************************************

Const TYPE_K = 3      ‘ K thermocouple SDC (from Sensor Tables)

Call SetSensorType(BaseAdrs, 2, TYPE_K)
```

Set Filter

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

$$F = Int(2.55 \times P)$$

All channel filter factors default to zero after a reset. This effectively disables the filters and maximizes channel response. 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 values to find the best response/noise tradeoff. Some applications, however, require knowledge of the filter time constant. This expression describes the relationship between time constant t, expressed in seconds, and filter factor F:

$$t \cong \left(\frac{-\text{NumActiveChannels}}{45 \times \ln\left(\frac{F}{256}\right)} \right)$$

COMMAND: (160 + Chan), (FilterFactor)

RESPONSE: NONE

VB Code Example:

```
'**********************************************************************
' Generic procedure to set filter factor on any channel.
'**********************************************************************
Sub SetFilter(BasePort%, Channel%, FiltFactor%)

   Const SET_FILT = 160        ' SET FILTER CONSTANT command opcode

   ' Send command string to coprocessor board
   Call SendByte(BasePort, SET_FILT + Channel)
   Call SendByte(BasePort, FiltFactor)

End Sub

'**********************************************************************
' Set up channel 4 with a 12.5% low-pass filter. To convert filter
' percentage (0 to 100%) to filter factor, multiply by 2.55.
'**********************************************************************

Call SetFilter(BaseAdrs, 4, 12.5 * 2.55)
```

Read Reference

This command returns the reference temperature from optional Sensoray termination boards (TBs), such as Models 7409TB, 7409TC and 7409TDIN. This function is useful for monitoring thermocouple reference junctions and can also serve as a debugging aid during 417 installation and application development.

Since two TBs can be connected to the 417, you must specify which TB you are addressing; the command byte includes a TB address code for this purpose. Set the TB address code to 0 to access the TB connected to channels 0-15, or to 1 to access the TB connected to channels 16-31.

The returned, signed 16-bit integer, temperature value is scaled to 0.10ºC per bit. If the addressed TB is not connected to the 417 board, this command will return a meaningless value.

COMMAND: (96 + TBcode)

RESPONSE: (BoardTemp MSB), (BoardTemp LSB)

VB Code Example:

```
‘***************************************************************
‘ Generic procedure to read temperature (in degrees C) from a TB.
‘***************************************************************
Function GetTbTemperature!(BasePort%, TbCode%)

   Const READ_TB = 96    ‘ READ BOARD TEMPERATURE command opcode

   ‘ Send command opcode to 417 board
   Call SendByte(BasePort, READ_TB + TbCode)

   ‘ Read TB temperature data and convert to degrees C
   GetTbTemperature = ReadWord(BasePort%) * 0.1

End Function

‘***************************************************************
‘ Read temperatures from both TBs and convert to degrees F
‘***************************************************************

Dim TbCode            As Integer   ‘ TB address code
Dim DegreesF(0 to 1)  As Single    ‘ Array to hold temperature data

For TbCode = 0 To 1
   DegreesF(TbCode) = 1.8 * GetTbTemperature(BaseAdrs, TbCode) + 32.0
Next TbCode
```

Read Channel

This command returns the most recently acquired sensor data from one channel. The returned value is scaled as a function of the channel's sensor type and is always represented as a signed, 16-bit integer. Refer to the *Sensor Table* at the end of this chapter for a list of sensor scaling factors.

COMMAND: (Chan)

RESPONSE: (Data MSB), (Data LSB)

VB Code Example:

```
‘****************************************************************
‘ Return the most recent Engineering Units data from any channel.
‘****************************************************************
Sub ReadChannel!(BasePort%, Channel%, Scalar!)

   ‘ Send command to 417 (Opcode = 0)
   Call SendByte(BasePort, Channel)

   ‘ Read chan data from 417 and convert to Engineering Units
   ReadChannel = ReadWord(BasePort) * Scalar

End Sub


‘**************************************************************
‘ Assume channel 6 has been configured for 5-Volt input signal,
‘ 200uV per bit. Read the voltage measured at channel 6.
‘**************************************************************

Dim ChanSixVolts   As Single
Dim Scalar         As Single

Scalar = 0.0002    ‘ From Sensor Tables, scalar = 200uV/bit

ChanSixVolts = ReadChannel(BaseAdrs, 6, Scalar)


‘**************************************************************
‘ Assume channel 4 has been configured for a K thermocouple,
‘ 0.1C/bit. Read the temperature measured at channel 4.
‘**************************************************************

Dim DegreesC As Single

DegreesC = ReadChannel(BaseAdrs, 4, 0.1)
```

Read Channel Group

This command returns the most recent sensor data from a group of eight channels. Sensor data is scaled as a function of the declared sensor type for each channel, just as it is scaled when using the *Read Channel* command. Refer to the *Sensor Table* at the end of this chapter for a list of sensor data scalars.

A Group Code is included in the command to specify which eight-channel group is to be returned. Group Code 0 returns channels 0-7, 1 returns channels 8-15, 2 returns channels 16-23, and 3 returns channels 24-31. The response string consists of channel data ordered by increasing channel number, beginning with the lowest channel number in the group. For example, Group Code 2 will return data in the following channel order: 16, 17, 18, ... , 23.

COMMAND:	**(104 + GroupCode)**
RESPONSE:	**(GroupChan 0 MSB), (GroupChan 0 LSB), (GroupChan 1 MSB), (GroupChan 1 LSB), (GroupChan 2 MSB), (GroupChan 2 LSB), (GroupChan 3 MSB), (GroupChan 3 LSB), (GroupChan 4 MSB), (GroupChan 4 LSB), (GroupChan 5 MSB), (GroupChan 5 LSB), (GroupChan 6 MSB), (GroupChan 6 LSB), (GroupChan 7 MSB), (GroupChan 7 LSB)**

VB Code Example:

```
‘*****************************************************************
‘ Read integer data from 8-channel group into DataArray().
‘*****************************************************************
Sub ReadChannelGroup(BasePort%, GroupCode%, DataArray%())

   Dim Channel As Integer    ‘ ChanNum and DataArray() index
   Const READ_GROUP = 104    ‘ READ CHANNEL GROUP command opcode

   ‘ Send command to 417 and read channel data into DataArray()
   Call SendByte(BasePort, READ_GROUP + GroupCode)
   For Channel = (8 * GroupCode) TO (8 * GroupCode + 7)
      DataArray(Channel) = ReadWord(BasePort)
   Next Channel

End Sub

‘****************************************************************
‘ Fetch integer data from all 32 channels.
‘****************************************************************
Dim Group                  As Integer   ‘ Channel Group specifier
Dim IntDataArray(0 To 31)  As Integer   ‘ Array to hold chan data

For Group = 0 to 3
   Call ReadChannelGroup(BaseAdrs, Group, IntDataArray())
Next Group
```

Set Limits

This command declares signed, 16-bit integer, upper and lower alarm threshold limits for a channel. An alarm will “sound” if the scaled sensor data strays below the lower limit or above the upper limit. The alarm “sounds” by setting the `ALARM` bit in the status register.

All limits assume default values when the coprocessor is reset. Low limits reset to -32768, and high limits to 32767. These default values effectively disable the alarm function. To disable the lower limit, specify a lower limit of -32768. Similarly, declare an upper limit of 32767 to disable the upper limit.

When a channel limit is exceeded, both channel limits are reset to their default values. This prevents the alarm from sounding again after the host has acknowledged the violation. The host must reprogram the violated limit to re-arm it.

COMMAND:	**(64 + Chan), (HiLimit MSB), (HiLimit LSB), (LoLimit MSB), (LoLimit LSB)**
RESPONSE:	**NONE**

VB Code Example:

```
'*********************************************************************
' Generic procedure to set alarm limits for one channel.
'*********************************************************************
Sub SetLimits(Channel%, LoLimit%, HiLimit%)

   Const SET_LIMITS = 64       'SET LIMITS command opcode

   ' Set channel alarm limit values on 417 board
   Call SendByte(BasePort, SET_LIMITS + Channel)
   Call SendWord(BasePort, HiLimit)
   Call SendWord(BasePort, LoLimit)

End Sub

'*********************************************************************
' Channel 7 is configured for a K thermocouple, 0.1C/bit. Program limits
' so that alarm will "sound" if temperature strays outside 400C-to-450C.
'*********************************************************************

Dim Scalar   As Single       'Data scalar appropriate for sensor type
Dim HiLim    As Integer      'High alarm limit threshold
Dim LoLim    As Integer      'Low alarm limit threshold

Scalar = 0.1                 'K thermocouple data scalar (from table)
HiLim = 450.0 / Scalar       'compute high alarm limit value
LoLim = 400.0 / Scalar       'compute low alarm limit value

Call SetLimits(BasePort, LoLim, HiLim)'Set channel 7 alarm limits
```

Set Fail Mode

This command establishes open sensor data values for a group of eight channels. The first command byte contains the command opcode and a Group Code that specifies which 8-channel group is being addressed. The second byte contains eight Mode Flag bits, one for each channel. Mode Flags are mapped to channels as shown in the following table:

Table 8: Channel mapping for ModeFlags byte

Group Code		ModeFlags Byte							
		D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
0	Addressed Channel	7	6	5	4	3	2	1	0
1		15	14	13	12	11	10	9	8
2		23	22	21	20	19	18	17	16
3		31	30	29	28	27	26	25	24

If a flag is set to logic 1 and the corresponding sensor is open, the value 32767 will replace the sensor data, causing it to "fail high." If the flag is reset, the value -32768 will replace the open-sensor data, causing it to "fail low." Following a hard or soft board reset, all channels automatically default to fail high.

Example: to program channel 21 to fail low, set GroupCode to 2 and reset bit 5 of the Mode Flags to 0. Note that channels 16 through 23 are simultaneously programmed.

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

COMMAND: (128 + GroupCode), (ModeFlags)

RESPONSE: NONE

VB Code Example:

```
‘************************************************************************
‘ Generic procedure to set fail mode for 8 channels.
‘************************************************************************
Sub SetFailMode(GroupCode%, ModeFlags%)
   Call SendByte(128 + GroupCode)  ‘Send Opcode & Channel Group to 417
   Call SendByte(ModeFlags)        ‘Send Mode Flags to 417
End Sub

‘************************************************************************
‘ A thermocouple is monitoring a process temperature on channel 24. If
‘ the thermocouple fails, the 417 must indicate a high temperature to
‘ ensure that the control heater is turned off. Program channels 24-26
‘ to fail high, and channels 27-31 to fail low.
‘************************************************************************

Call SetFailMode(3, 7)‘Set fail mode for chans 24-31: (0,0,0,0,0,1,1,1)
```

Read Alarms

This command (1) returns alarm status from a group of eight channels, and (2) clears the alarm status of all channels within the 8-channel group, and (3) resets the status register ALARM bit. A Group Code is used to designate which 8-channel block is being addressed. The first response byte reflects the status of high limit violations and the second byte reflects the status of low limit violations. Any status bit that is set to logic 1 indicates a violated limit.

Table 9: Channel mapping for Read Alarms command response bytes

Group Code		Either Response Byte							
		D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
0	Addressed Channel	7	6	5	4	3	2	1	0
1		15	14	13	12	11	10	9	8
2		23	22	21	20	19	18	17	16
3		31	30	29	28	27	26	25	24

COMMAND: (108 + GroupCode)

RESPONSE: (HighLimitFlags), (LowLimitFlags)

VB Code Example:

```
‘*********************************************************************
‘ Fetch alarm flags from one 8-channel group.
‘*********************************************************************
Sub ReadAlarms(BasePort%, GroupCode%, HiFlags%, LoFlags%)
   Call SendByte(108 + GroupCode)  ‘request group alarm flags
   HiFlags = ReadByte(BasePort)    ‘read high flags
   LoFlags = ReadByte(BasePort)    ‘read low flags
End Sub

‘*********************************************************************
‘ Check alarm status. If alarm(s) are sounding, test channels 8-15 and
‘ return the number of channels sounding in AlarmCount and a list
‘ of alarming channels in AlarmList(1 to 8). Assume all other channels
‘ have disabled alarms.
‘*********************************************************************
Sub GetAlarmList(BasePort%, AlarmCount%, AlarmList%())
   Dim HiFlags%, LoFlags%, Chan%
   AlarmCount = 0
   If (INP(BasePort + 1) And ALARM) Then
      Call ReadAlarms(BasePort, 1, HiFlags, LoFlags)
      For Chan = 8 to 15
         If (HiFlags Or LoFlags) And (2 ^ (Chan And 7)) Then
            AlarmCount = AlarmCount + 1
            AlarmList(AlarmCount) = Chan
         End If
      Next Chan
   End If
End Sub
```

50Hz Reject Mode

This command changes the digitizer’s integration period from 16.7ms to 20.0ms, enabling the coprocessor to reject 50 Hertz differential line noise. This command should be used in systems targeted for 50Hz power environments.

After a hard or soft reset, the digitizer defaults to 16.7ms in order to reject 60Hz line noise. After invoking this command, the coprocessor can be restored to the 60Hz rejection mode only by a hard or soft reset.

Note: all timing specifications assume the 417 board is operating in the 60Hz rejection mode.

COMMAND: (100)

RESPONSE: NONE

VB Code Example:

```
‘*****************************************************************
‘ This procedure switches the 417 to the 50Hz Reject Mode.
‘*****************************************************************
Sub Set50HzRejectMode(BasePort As Integer)
   Call SendByte (BasePort, 100)
End Sub
```

High Speed Mode

This command increases the the channel scan rate to allow higher throughput. Channel processing time is decreased from 22 milliseconds/channel (default) to 9 milliseconds/channel. Increasing the scan rate reduces accuracy by approximately sixty percent and will sometimes result in higher noise levels, due primarily to shortened signal integration time.

A hard or soft board reset will always restore the board to the default scan rate. A board reset is the only way to resume the default scan rate after invoking the *High Speed Mode* command.

COMMAND:	**(224), (8), (0)**
RESPONSE:	**NONE**

VB Code Example:

```
'*****************************************************************
' This procedure switches the 417 to the High Speed Mode.
'*****************************************************************
Sub HighSpeedMode(BasePort As Integer)
   Call SendByte (BasePort, 224)
   Call SendByte (BasePort, 8)
   Call SendByte (BasePort, 0)
End Sub
```

Read Version

This command returns the coprocessor firmware version number, times 100. The version number is printed on the EPROM device that is plugged into the coprocessor board.

COMMAND: (224), (5), (0)

RESPONSE: (Version_MSB), (Version_LSB)

By means of this command, the host can automatically determine any enhanced coprocessor functions that may be present in future firmware releases.

VB Code Example:

```
‘*****************************************************************
‘ This function returns the coprocessor firmware version number.
‘*****************************************************************
Function GetVersion!(BasePort As Integer)

   Call SendByte(BasePort, 224)
   Call SendByte(BasePort, 5)
   Call SendByte(BasePort, 0)

   GetVersion = ReadWord(BasePort) / 100.0

End Function

‘*****************************************************************
‘ Fetch the coprocessor firmware version number.
‘*****************************************************************

Dim Version As Single

Version = GetVersion(BaseAdrs)
```

Read Identifier

This command returns a product identifier specific to the Model 417 board. The command response consists of the decimal value *417*.

COMMAND: **(224), (4), (0)**
RESPONSE: **(1), (161)**

This command is useful both as a diagnostic and as a mechanism for automating the determination of the product type residing at a particular port address.

VB Code Example:

```
‘**********************************************************************
‘ This function returns the coprocessor product identifier.
‘**********************************************************************
Function GetProductID%(BasePort As Integer)

   Call SendByte(BasePort, 224)
   Call SendByte(BasePort, 4)
   Call SendByte(BasePort, 0)

   GetProductID = ReadWord(BasePort)

End Function

‘******************************************************************
‘ Fetch the coprocessor product identifier. In the case of Model
‘ 417, the returned ProductID value should be decimal 417.
‘******************************************************************

Dim ProductID As Integer

ProductID = GetProductID(BaseAdrs)
```

Calibrate

This command calibrates one of the coprocessor's internal standards. You must supply two stable references, 5V and 500mV, which are applied to external channels during board calibration. External channels used for the calibration must be properly configured and allowed to settle before issuing the calibrate command. A board reset must be invoked after each range is calibrated.

COMMAND: (192 + Chan), (RangeCode), (CalData MSB), (CalData LSB)

RESPONSE: (IgnoreThisValue)

Table 10: Range codes for the Calibrate command

RangeCode	Range
0	5 V
1	500 mV

VB Code Example:

```
'*********************************************************************
' Calibrate one voltage range on a Model 417 board.
'*********************************************************************
Sub CalRange(BasePort%, Chan%, RangeCode%, AppliedVolts!)
   Dim TypeCode    As Integer
   Dim Scalar      As Single
   Dim Junk        As Integer

   Select Case RangeCode
      CASE 0: Scalar = 5000:    TypeCode = &H15
      CASE 1: Scalar = 50000:   TypeCode = &H16
   End Select

   Call SetSensorType(BasePort, Chan, TypeCode)
   Call Delay(900)

   Call SendByte(192 + Chan)       'Issue Calibrate command to 417
   Call SendByte(RangeCode)
   Call SendByte(AppliedVolts * Scalar)
   Junk = ReadByte(BasePort)       'Wait for 417 to calibrate

   Call OUT(BasePort + 1, 0)       'Reset the 417 board
   Do: Loop While (INP(BasePort + 1) And FAULT)
End Sub

'*********************************************************************
' Calibrate a Model 417 board. This snippet assumes 5.0000V is
' applied to channel 3 and 0.50000V is applied to channel 7.
'*********************************************************************

Call CalRange(BaseAdrs, 3, 0, 5.0000)    'Calibrate 5V range
Call CalRange(BaseAdrs, 7, 1, 0.50000)   'Calibrate 500mV range
```

Sensor Table

Table 11: Sensor Definition Codes and Scale Factors

Sensor Type		Code	Resolution
Voltage	+100mV	17H	5µV
	+500mV	16H	20µV
	+5V	15H	200µV
	+5V	00H	500µV
Thermocouple	B	24H	0.10°C
	C	23H	
	E	01H	
	J	1BH	
	K	1CH	
	N	22H	
	T	1DH	
	S	1EH	
	R	1FH	
Current Loop	4-20mA	11H	0.01%
Disabled		13H	N/A

Notes:

Scale Factors: The Scale Factor for each sensor type is shown above in the Resolution column, above. Example: when a channel is configured for the +500mV measurement range, data is scaled to 20µV per bit. In this case, convert the integer data returned by the *Read Channel* command to Volts by multiplying times 0.00002.

Current Loops: Channels configured for current loops return data values that are scaled to 0.01% per bit (4mA=0%, 20mA=100%). Correspondingly, decimal value 0 will be returned for a 4mA current, and decimal 10000 (100% / 0.01% per bit) will be returned for a 20mA current.

“Disabled” type: Declaring a channel’s sensor type as *Disabled* removes the channel from the scan loop. Data values returned from such channels are meaningless and should be ignored. In the case of the *Read Channel Group* command, data is always returned for eight channels, regardless of whether any of the channels are disabled.

Type 0: Sensor Definition Code 00H—the default sensor type after a reset—is provided for back-compatibility with earlier Sensoray products. Use of this type is not recommended for new designs, as higher resolution is available from type 15H. If you are utilizing the +5V measurement range, specify type 15H for the best performance.

Chapter

5 Sensor Connections

Connectors

Analog input signals connect to the 417 board by means of header connectors P1 and P2. Header P1 provides connections to channels 0-15 and P2 to channels 16-31 as shown in the table below:

Table 12: Connector pinouts for headers P1 and P2

Connector P1								Connector P2							
Channel	**Input Polarity**	**Pin Number**	**TB Marking**	**Channel**	**Input Polarity**	**Pin Number**	**TB Marking**	**Channel**	**Input Polarity**	**Pin Number**	**TB Marking**	**Channel**	**Input Polarity**	**Pin Number**	**TB Marking**
0	+	2	V+0	8	+	1	I+0	16	+	2	V+0	24	+	1	I+0
	-	4	V-0		-	3	I-0		-	4	V-0		-	3	I-0
1	+	6	V+1	9	+	5	I+1	17	+	6	V+1	25	+	5	I+1
	-	8	V-1		-	7	I-1		-	8	V-1		-	7	I-1
2	+	10	V+2	10	+	9	I+2	18	+	10	V+2	26	+	9	I+2
	-	12	V-2		-	11	I-2		-	12	V-2		-	11	I-2
3	+	14	V+3	11	+	13	I+3	19	+	14	V+3	27	+	13	I+3
	-	16	V-3		-	15	I-3		-	16	V-3		-	15	I-3
4	+	18	V+4	12	+	17	I+4	20	+	18	V+4	28	+	17	I+4
	-	20	V-4		-	19	I-4		-	20	V-4		-	19	I-4
5	+	22	V+5	13	+	21	I+5	21	+	22	V+5	29	+	21	I+5
	-	24	V-5		-	23	I-5		-	24	V-5		-	23	I-5
6	+	26	V+6	14	+	25	I+6	22	+	26	V+6	30	+	25	I+6
	-	28	V-6		-	27	I-6		-	28	V-6		-	27	I-6
7	+	30	V+7	15	+	29	I+7	23	+	30	V+7	31	+	29	I+7
	-	32	V-7		-	31	I-7		-	32	V-7		-	31	I-7

Since all 32 channels have true differential inputs, each channel must have both a positive and negative connection, shown as “+” and “-”, respectively, under *Input Polarity* in the above table. Headers P1 and P2 also make available ISAbus ground (sometimes referred to as the “shield”) at pins 33, 34, and 38-40, +12V at pin 36, and a thermocouple reference sensor input at pin 37.

Live Connections—Model 417 circuitry is designed to permit modifications to sensor connections without disturbing system operation. Sensors may be electrically connected and disconnected from the coprocessor without physically removing the 417 board or electrically removing power from the ISAbus.

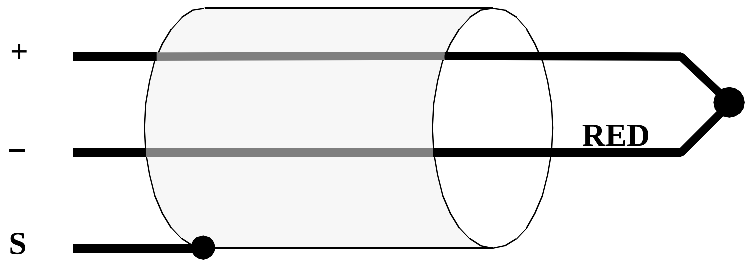
Termination Boards

To simplify field wiring, sensors may be connected to optional Sensoray screw termination boards, such as Models 7409TB, 7409TC or 7409TDIN. All of these termination boards are designed to connect directly to Model 417 headers P1 and P2 by means of standard 40-conductor flat cables.

Markings—the above *connector pinout* table lists termination board (TB) markings for screw terminations as they appear on a Model 7409TB. Example: channel 5 connects to the CH5 V+ and V- terminals on the TB connected to P1. Another example: channel 28 connects to the CH4 I+ and I- terminals on the TB connected to P2. ISAbus ground—labeled “S” on the 7409TB—connects to the cable shield, if present.

Thermocouples

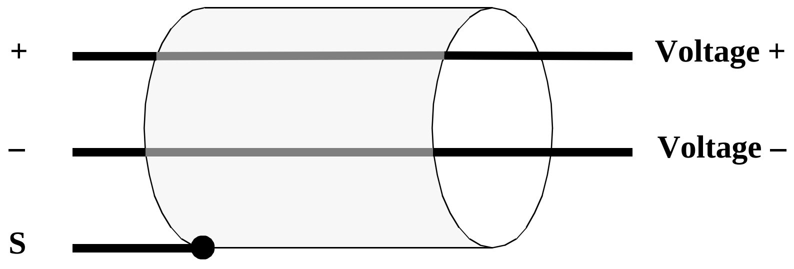
The positive thermocouple wire must be connected to the “+” terminal, and negative wire to the “-” terminal. Regardless of thermocouple type, the negative wire is always colored red.



For best results, use either a grounded or isolated thermocouple. If the thermocouple is not grounded or isolated, high common-mode voltages may be present that can disrupt measurement accuracy or damage the 417 circuit board.

DC Voltage

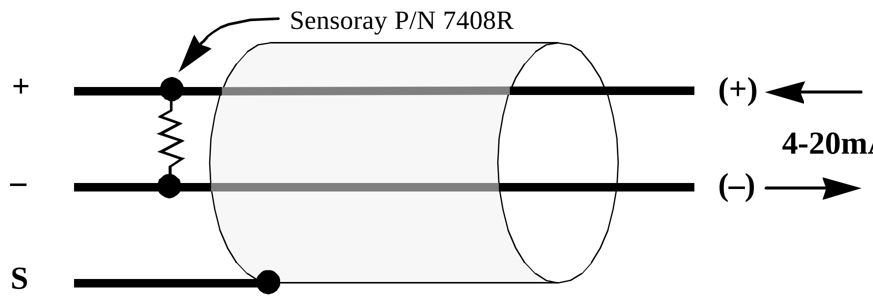
DC voltage sources must be arranged so that the positive wire is connected to the “+” terminal, and the negative wire to the “-” terminal:



If the voltage source is not referenced to ISAbus ground, connect either the “+” or “-” lead to “S” to prevent high common-mode voltages.

4-to-20 mA Current Loops

A 250Ω, 0.01% resistor must be installed as shown below. These resistors are available as an option for the Model 417; order Sensoray part number 7408R. Make sure the resistor is at the *grounded* end of the loop so that the common-mode voltage does not exceed 5 Volts.



Solving Noise Problems

Electrical noise can sometimes interfere with measurements. Fortunately, there are a handful of simple procedures that will solve most noise problems:

1. *Examine the input signal* with a DVM and/or oscilloscope to see if you really have the signal you think you have. Measurement “noise” may actually be the result of incorrect wiring or a fault at the signal source.

2. *Make sure the input is referenced to ISAbus ground.* Inputs that are not ground-referenced are, in effect, high-impedance “antennas” that pick up ambient noise from the environment. Such noise can elevate the common-mode voltage amplitude beyond the Model 417’s allowable common-mode input limits.

   Thermocouples—try grounding the “hot” end of the TC.

   DC Voltage inputs—if the input is isolated, try connecting one of the input terminals to ISAbus ground, either directly or by means of a resistor (i.e., 10KΩ).

   Current Loops—make sure one end of the 7408R resistor is connected to ISAbus ground.

3. *Shield the input wires.* Both input wires should be surrounded by the shield conductor, if present. Terminate the shield conductor only at the 417 end of the sensor cable assembly. Make sure the remote end of the shield conductor is electrically insulated from all other electrical conductors, otherwise you may end up with a ground-loop error.

4. *Invoke the channel software filter.* This should be tried as a last resort if everything else fails. If it becomes necessary to do this, it probably indicates that (a) the noise is fundamentally differential, and (b) the noise is produced at the signal source and is not caused by the interface between source and measurement system. In such a case, it may be worthwhile to evaluate the signal source itself for potential noise reduction solutions.

Chapter

6

Thermocouples

Overview

A thermocouple (TC) consists of a conductor pair constructed from dissimilar metals. At one end of the pair, the two conductors are electrically shorted together. This end of the TC—the *hot* junction—is thermally attached to the point to be measured. The other end of the pair—the *cold*, or *reference* junction—is connected to a measuring device. In this discussion, the measuring device is assumed to be a termination board—any of Sensoray Models 7409TB, 7409TC or 7409TDIN—in conjunction with a Model 417 thermocouple interface board.

Thermocouples generate an open-circuit voltage that is proportional to the thermal gradient between hot and cold junctions. Since this voltage is a function of the temperature *difference* between junctions, it is necessary to know the temperature at the cold junction in order to determine the temperature at the hot junction.

Reference Sensor

An integrated circuit temperature transducer resides on Sensoray termination boards. This transducer, because of its close physical proximity to the TC terminations, represents an accurate measurement of the TC cold junction. Periodically, the 417 measures the transducer signal and computes the corresponding TC voltage, known as the *compensation voltage*.

When a TC is measured, the 417 adds the TC voltage to the compensation voltage to obtain the *corrected voltage*. This is the voltage one would measure if the cold junction were maintained at zero degrees C. Finally, the corrected voltage is converted into the appropriate temperature units by means of a non-linear mapping function.

Ensuring Accuracy

To get the very best accuracy from TCs, the reference sensor must have the same temperature as the TC cold junction. Some manufacturers attempt to impose a constant temperature on both cold junction and reference sensor by means of an “isothermal” heat conductor. While this may appear to be a reasonable approach, it often fails to perform adequately in practice. The principal failure of the Isothermal Conductor approach is this: “isothermal” conductors are rarely isothermal. Any air flow across the heat conductor has the potential to produce a thermal gradient.

So what is the solution to this problem? Force the termination system to be isothermal. Make sure your termination system is insulated from outside air flows, and avoid exposing the termination system to sudden thermal transients. Example: enclose the termination system in a sealed box.

Chapter

7

Theory of Operation

Software

Execution of the embedded coprocessor firmware is managed by a multitasking kernel. The kernel is responsible for scheduling and executing tasks, processing intertask communication requests, and servicing real-time communication interrupts from the system bus interface. The following paragraphs describe the inner-workings of tasks that are controlled by the kernel.

Scanner

This task performs the fundamental data acquisition function central to the coprocessor's purpose. By order of increasing channel numbers, this task repetitively scans all active channels.

The scanner begins a data conversion by selecting the next channel to be digitized. Analog multiplexers are switched to select the target input channel, then the scanner sleeps while the sensor signal stabilizes. When the front end has settled, the scanner awakens and initiates an analog-to-digital conversion. Other processes are allowed to run during the conversion. Upon completion of the conversion, the digitized data is passed to the post-processor.

Post-processor

The post-processor is responsible for refining uncorrected A/D data into finished form.

This task begins by receiving uncorrected channel data from the scanner. The data is corrected for circuit offset and gain errors, then linearized and converted to engineering units appropriate for the declared sensor type. The resulting value is passed through a programmable, single-pole low-pass filter, then is stored in RAM for rapid host access.

Filtered data is checked for limit violations. If a violation is detected , both of the violated channel's limits are reset to their default values, the channel's high or low alarm flag is asserted and the status register Alarm bit is set.

Command Processor

The command processor fetches and executes commands from the host. Top priority is given to this task in order to minimize communication latency between the host and coprocessor.

This task is awakened when a command is received from the host. The command processor decodes the command opcode and invokes the appropriate command processing function. This task manages command and data port communication handshake.

Hardware

Although the coprocessor's analog circuitry is proprietary to Sensoray, an overview of the circuitry is provided here to provide some insight into the inner-workings of the board. Analog circuitry is logically partitioned into three sections: reference, conditioning and measurement.

Reference Standards

The reference section consists of a set of standards that are used to enhance the stability and accuracy of all sensor measurements. These standards are very stable over time and temperature. The exact values of the standards are stored in non-volatile memory when the board is calibrated.

Reference standards are measured periodically by the onboard CPU. To make standards measurements appear as transparent as possible, these measurements are interleaved with external sensor measurements. The values of the digitized standards are stored in onboard RAM and used to null offset and gain errors in the external sensor measurements.

A phenomenon known as *warm-up noise* occurs when the 417 is warming up from cold start or when subjected to a thermal transient. When the coprocessor is exposed to changing ambient temperatures, sensor data will appear somewhat noisy. This is caused by circuit gain and offset characteristics changing faster than reference standards are measured. Warm-up noise subsides when the 417 circuit board reaches thermal stability.

Signal Conditioning

Each channel is provided with a signal conditioning circuit. This circuit serves to (1) protect input circuitry from high common-mode voltages, and (2) detect thermocouple open-circuit failures, and (3) condition millivolt-level thermocouple signals for measurement.

Measurement Section

The measurement section is responsible for selecting and digitizing the external sensor input signals, as well as the internal reference standards.

The measurement section starts a digitization by selecting a channel to be measured. The selected channel is switched through a differential analog multiplexer. Reference standards are routed through dedicated switches so as to not rob inputs from external sensors.

Next, the selected channel is passed through a programmable gain amplifier. Gain level is controlled by the coprocessor's onboard CPU. The onboard CPU sets amplifier gain based on the declared sensor type in the case of external sensor channels, or based on the reference type in the case of internal reference standard channels.

Finally, the amplified signal is applied to the input of an integrating A/D converter. The converter circuit, based on a voltage-to-frequency converter, directly produces a floating-point output value. Converter resolution increases with decreasing signal amplitude so that the best resolution is provided for the lowest-level signals. Note: the specified converter resolution is the worst-case resolution across the entire input range.

Channel selection, amplifier gain, and A/D functions are all under direct control of the coprocessor's CPU. Consequently, none of these circuits are accessible to the host.

Timing

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

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 417 coprocessor.

The basic channel process time—the *scan time*—is approximately 22 milliseconds at the default scan rate, or 9 milliseconds when operating in the High Speed Mode. This is the amount of time required for the 417 to configure its analog section, allow the front end to stabilize and digitize the input signal. Note that the linearization, limit processing and filtering functions don't require any additional time as they execute concurrently in the digital domain.

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

$$\text{SamplesPerSecond} \cong \frac{1}{(\text{NumActiveChannels}) \times (\text{ChannelScanTime})}$$

Another important parameter is the *refresh time*—the time elapsed between updates of a channel's measured data. This parameter may be critical in control applications that require new data every time channel data is read. In such applications, it is necessary to know the maximum refresh time: the minimum time between successive reads from a channel that guarantees fresh data. The maximum refresh time is given by this function:

$$\text{RefreshTime}_{max} = (\text{NumActiveChannels} + 1) \times (\text{ChannelScanTime}) + \text{CommTime}$$

Note that time spent communicating with the host typically translates into time stolen from the scanning function, as denoted by CommTime in the above expression. Communication between host and 417 will tend to reduce the scan rate and increase the maximum refresh time. See the next section for details of communication timing.

Communication Latency

Communication latency is a measure of communication delays between host and 417. Perhaps the most important aspect of latency is the time elapsed from a host request for sensor data to the acquisition of that data. This is called the a*cquisition latency*.

Acquisition latency can be viewed as having two parts. *Response delay* is the time elapsed between writing to the command port and availability of the first response byte in the data port. *Data delay* is the time elapsed between reading a response byte from the data port and availability of the next response byte in the data port.

Two commands may be used to return sensor data to the host processor, *read channel* and *read channel group*. For both of these commands, the maximum response delay is 70µs and the maximum data queue delay is 20µs.

The *read channel* command consumes 90µs, maximum: 70µs response delay for the first data byte, plus 20µs delay from the time the host reads the first byte to availability of the second response byte.

The *read channel group* command consumes 370µs, maximum: 70µs response delay plus 15 more data bytes with 20µs data delay each.

Note that the above analyses do not consider host processor overhead. In reality, acquisition latency exceeds the theoretical maximums discussed here. Additional time is consumed by the host when it polls the 417's handshake status, writes to the command register and reads from the data register.

Processor Speed

The coprocessor's I/O bus interface circuitry is designed to operate at the standard ISAbus 8MHz rate.

There is no bottom limit on host processor speed except for the following: read and write strobes to the 417 must not exceed 5µs in duration. Handshake integrity cannot be guaranteed if host read/write strobes exceed 5µs. This restriction is typically not relevent unless you are exercising the ISAbus with an emulator.