WINSYSTEMS®

PCM-518

Intelligent PC/104 8-Channel Sensor Interface Module

Product Manual

WINSYSTEMS, INC. | 2890 112th Street, Grand Prairie, TX 75050 | 817-274-7553 | info@winsystems.com | www.winsystems.com

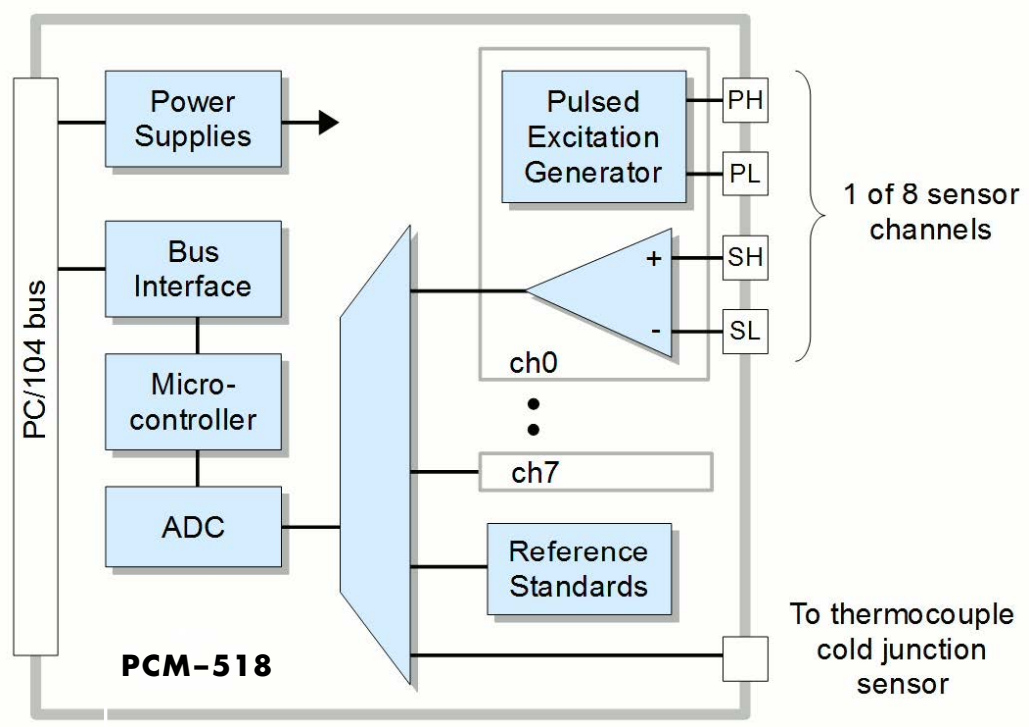
Table of Contents

Chapter 1: Introduction

1.1 Functional overview

PCM-518 is a PC/104 circuit board that interfaces any combination of up to eight sensors to a host computer. Each sensor channel provides complete signal conditioning for a variety of sensor types, thus allowing any channel to be directly connected to a thermocouple, RTD, strain gauge, thermistor, resistor, 4-to-20 mA current loop, or DC voltage.

Block diagram



Pulsed excitation is provided for thermistors, resistors, RTDs, and strain gauges, which minimizes sensor self-heating and reduces power consumption. Excitation signals are routed to dedicated connector pins to allow four-wire sensor circuits that completely eliminate lead-loss errors.

Cold junction compensation is automatically applied to thermocouples (requires optional, external sensor), and fully differential inputs are used to provide common-mode voltage rejection.

The onboard controller continuously scans the eight sensor channels. As each channel is scanned, the sensor is excited, digitized, normalized to precision standards, linearized, and converted to engineering units as appropriate for the sensor type. The most recent measurement from each channel is immediately accessible to the host.

Alarm limits may be programmed for each channel. A status flag is set when any limit is violated, eliminating the need for host polling.

Chapter 2: Configuration and installation

2.1 Handling instructions

The PCM-518 board contains electronic circuitry that is sensitive to electrostatic discharge (ESD). Special care should be taken in handling, transporting, and installation to prevent ESD damage to the board. In particular:

- Do not remove the board from its protective packaging until you are ready to configure and install it.
- Handle the board only at grounded, ESD protected stations.

2.2 Configuration

Hardware configuration must be completed before the board is installed. This is accomplished by installing shunts to configure various hardware options. Some shunts are factory installed, which results in the following configuration:

Default configuration:

Base address	0x2B0
PC/104 interrupts	Disabled
Channel filters	Disabled

2.2.1 Address mapping

The board may be mapped to any four-byte address block within the range 0x000 to 0x3FF. The board occupies a four-byte address range but uses only the first two address locations in the range.

To avoid address conflicts, you must map the board into an address range that is not used by other devices. PCM-518 does not decode the full PC/104 16-bit I/O address; only the low ten address bits are decoded. Consequently, aliased “images” of the 518 will appear throughout the 16-bit address range at intervals of 0x400 bytes. You must ensure that these images do not conflict with other devices.

Shunts E1 through E8 are used to select the board's base address. Shunts are factory set to locate the board at base address 0x2B0. If you require a different base address, use the following table to determine the shunts needed for your base address.

Address selection shunts:

Address bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Shunt	0	0	0	0	0	0	E8	E7	E6	E5	E4	E3	E2	E1	0	0

You must install a shunt to match address bit = '0', or remove the shunt to match address bit = '1'. For example, to configure the board's base address to 0x390 (binary 0000 0011 1001 0000), install shunts E5, E4, E2 and E1.

2.2.2 Interrupts

The board may be configured to interrupt the host computer in response to various alarm and communication events. All of the board's interrupt requests share a common IRQ line on the PC/104 bus.

If you will be using interrupts, you must install a shunt to select the desired IRQ line. Any interrupt line from IRQ2 to IRQ7 may be selected (be sure to choose an IRQ line that will not conflict with other devices in your system). Do not install any IRQ shunts if you will not be using interrupts.

2.2.3 Thermocouple shunts

Each sensor channel is associated with two shunts which provide a ground-reference and facilitate open-sensor detection for thermocouples. If a channel is connected to a thermocouple or other floating voltage source then you must install both of its shunts, otherwise both shunts should be removed.

Thermocouple shunts:

Channel	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Shunts	E9, E17	E10, E18	E11, E19	E12, E20	E13, E21	E14, E22	E15, E23	E16, E24

2.3 Installation

Before installing or removing the circuit board from the system:

- Remove power from the system.
- Disconnect the board's P1 connector.

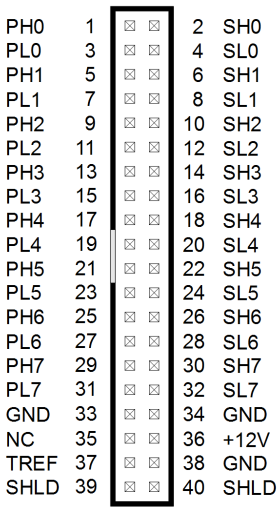
To install the board:

- Plug the board into the PC/104 stack and secure it with the included hardware.
- Connect an external field-wiring termination board to connector P1 with a 40-conductor flat ribbon cable. We recommends the model 7409TB termination board and 7409C cable for this purpose.
- Connect your sensors to the termination board as required. See Sensor connections for details.

Chapter 3: Sensor connections

3.1 Connector P1

P1 connector pinout



All sensors connect to the board via connector P1. Optionally, sensors may be connected to a WinSystems model 7409TB screw termination board, which in turn connects to P1 via ribbon cable. A 7409TB (or equivalent) is required if you are measuring thermocouples as it provides an essential temperature transducer for cold junction compensation.

Each sensor may have as many as five connections to a sensor channel. SH and SL are are used for all sensor types; these are the positive and negative differential sense inputs. Passive sensors also require connections to the PH/PL excitation terminals. The GND and SHLD pins may be connected to cable shields.

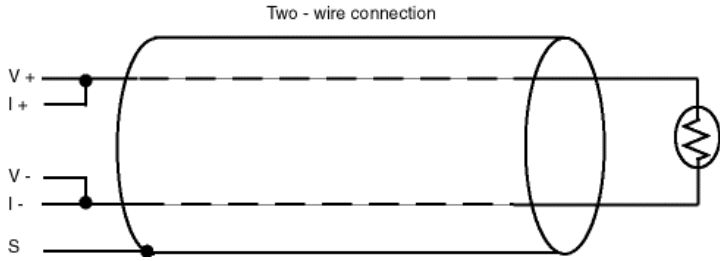
The following sections explain how to connect sensors when using a 7409TB termination board.

Pin functions

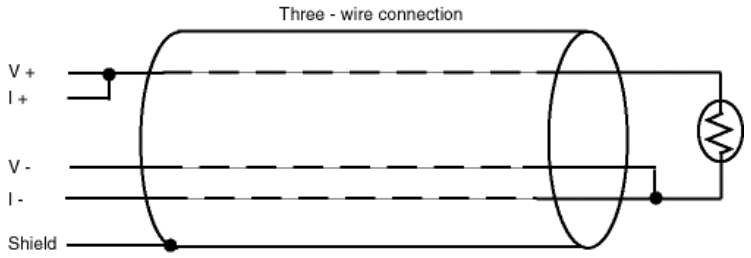
P1 Signal Name	7409TB Legend	Function
SHx	V+	Positive analog input for sensor channel x (half of differential pair SH/SL).
SLx	V-	Negative analog input for sensor channel x (half of differential pair SH/SL).
PHx	I+	Positive excitation output for sensor channel x (half of excitation pair PH/PL).
PLx	I-	Negative excitation output for sensor channel x (half of excitation pair PH/PL).
GND / SHLD	S	Cable shields; power return for cold junction temperature sensor.
+12V		+12 VDC power for cold junction temperature sensor (5 mA maximum).
TREF		Cold junction temperature sensor output (10 mV/°K).
NC		No connect.

3.2 RTDs, thermistors and resistors

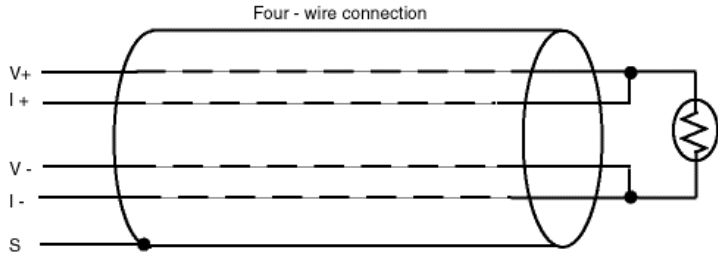
RTDs, thermistors and other resistors can be connected using a two-, three- or four-wire circuit. The simplest of these is the two-wire circuit which, as the name implies, requires only two wires. The V+ and I+ terminals are shorted together, and V- and I- are shorted together at the 7409TB. This conserves wire but introduces error as a result of the voltage difference (a "voltage loss") between the V/I junctions and sensor.



By using three wires, it is possible to reduce voltage loss errors by 50 percent. Instead of shorting V- and I- together at the 7409TB, separate wires can be run from each of these terminals out to the sensor. The wires are then shorted together at the sensor so that the high impedance V- terminal detects the voltage at the sensor, before losses can occur.



A four-wire circuit will entirely eliminate voltage losses. Like the three-wire circuit, separate wires are run from the V- and I- terminals to the sensor. In addition, separate wires are run from V+ and I+ to the sensor, where they are shorted together.

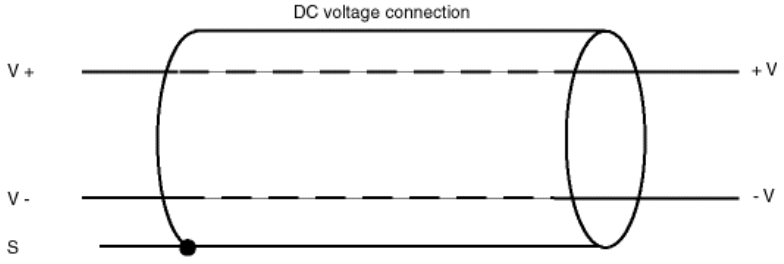


Thermistors have much higher resistance than RTDs over most of their operating range. As a result, a two-wire circuit may be used if your sensor will be operating only at higher resistance values. If it will be operating at lower resistance values, it is recommended to use a three- or four-wire circuit to prevent significant voltage losses.

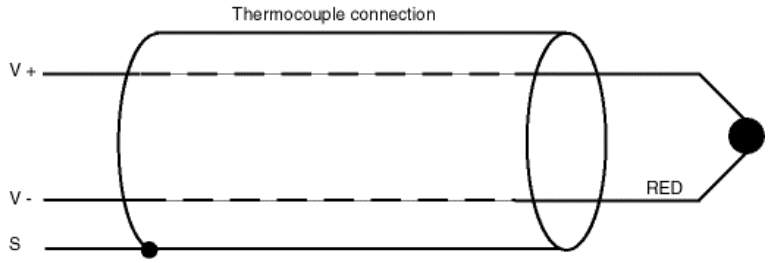
It is recommended to use shielded cable for all passive resistive devices. The cable shield must be connected only to the S terminal on the 7409TB; leave it unconnected at the sensor end of the cable.

3.3 Thermocouples and voltage

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



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

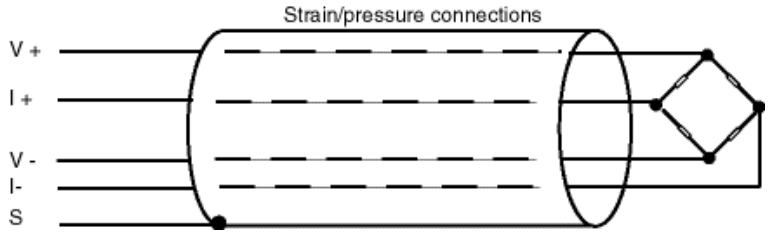


Electrical noise may be present on thermocouple and voltage signals. It is not within the scope of this manual to discuss all treatments of such noise, however, a few simple techniques are available which will solve many noise problems:

1. Inspect the sensor signal with an oscilloscope to confirm that it matches your expectations. “Noise” can often be the result of an incorrect sensor wiring or other external influences.
2. Install the hardware filter jumper on the offending channel.
3. In the case of thermocouples, try grounding the hot junction. Sometimes the thermocouple is not referenced to the PC/104 return. Noise induced into the thermocouple wire from the environment can elevate common mode voltage beyond the board input range. Make sure that the ground is referenced to the PC/104 bus ground. In the case of a DC voltage source, try grounding either the V+ or V- signal (but not both) to limit the common-mode voltage.

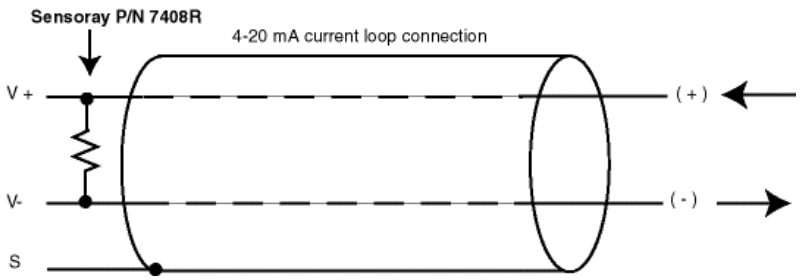
3.4 Strain and pressure gauges

Four wires are used to connect a gauge to the measurement system. Two wires supply excitation and the other two wires convey the gauge output signal. If you have a six-wire gauge, connect its excitation source and sense leads together at the 7409TB I+ and I- terminals. Strain and pressure gauges should use shielded cable. The shield should be connected only to the channel S terminal on the 7409TB.



3.5 Current loops

The board supports 4-20 mA current loop inputs on any channel when an optional 250-ohm 0.01% resistor is installed as shown below. These resistors are available as an option for PCM-518: order WinSystems part number 7408R. Note: The sensor channel must be located at the ground end of the current loop so that common mode voltage will not exceed 5 V.



Chapter 4: Register-level programming

Programming of the 518 is achieved via built-in commands. Commands are sent from the PC/104 master (the “host”) to the 518. Some commands cause responses to be sent to the host. This chapter describes the board's register architecture and explains how the host communicates with the board at the register level.

4.1 Programming model

The board occupies four consecutive addresses in system I/O space (only the lower two addresses are used). Each address has a different function for read and write operations.

Registers at the base address are used to exchange commands and data between 518 and host. Commands are sent to the 518 and responses are passed back to the host through the base address. Registers at base address + 1 consist of the read-only status register and write-only control register.

The base address consists of two hardware registers: command and data. When the host sends a command byte to the 518, it is storing a byte in the board's command register. When the host reads a reply byte from the 518, it is fetching a byte from the board's data register.

The status register exposes board status information to the host. The control register is used to invoke soft resets and to enable and disable host interrupts.

Register map:

Address	Read	Write
Base + 0	Data register	Command register
Base + 1	Status register	Control register

4.2 Status register

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

Status register:

Bit	7	6	5	4	3	2	1	0
Function	CRMT	DAV	ALARM	FAULT	x	x	x	x

Status bits:

CRMT	(Command Register eMpTy) indicates the 518 is ready to accept a command byte into its command register.
DAV	(Data AVailable) indicates that the 518 data register contains a data byte ready for access by the host.
ALARM	Indicates one or more programmed channel limits were exceeded.
FAULT	Indicates board reset is in progress. If persistently set to '1', this indicates a board hardware fault. In normal operation, FAULT is briefly active (less than one-half second) following a board reset. FAULT reflects the state of the board's red LED fault indicator. Note: CRMT, DAV, and ALARM status bits are not valid when FAULT = '1'. After resetting the board, the host should not attempt to execute commands until FAULT bit changes to '0'.

4.3 Control register

The control register provides the means for host management of interrupts and soft reset.

Control register:

Bit	7	6	5	4	3	2	1	0
Function	SET/CLR	0	0	INT/RST	0	ICMD	IDAT	IALARM

Control bits:

Name	Function
INT/RST	Specifies the control function to be performed. When set to logic zero, the board is reset and all other control register bits are ignored. When set to '1', the other control register bits behave as described below.
SET/CLR	Specifies whether selected interrupts are to be enabled or disabled. When set to '1', all selected interrupts are enabled. When set to '0', selected interrupts are disabled.
ICMD	Selects CRMT interrupt. When set to logic one, the CRMT interrupt is enabled or disabled as determined by the state of the SET/CLR bit. When set to logic zero, the CRMT interrupt enable is unchanged. While enabled, the host will be interrupted whenever the status register CRMT bit is asserted.
IDAT	Selects DAV interrupt. When set to logic one, the DAV interrupt is enabled or disabled as determined by the state of the SET/CLR bit. When set to logic zero, the DAV interrupt enable is unchanged. While enabled, the host will be interrupted whenever the status register DAV bit is asserted.
IALARM	Selects ALARM interrupt. When set to logic one, the ALARM interrupt is enabled or disabled as determined by the state of the SET/CLR bit. When set to logic zero, the ALARM interrupt enable is unchanged. While enabled, the host will be interrupted whenever the status register ALARM bit is asserted.

4.4 Handshaking

If you will not be using PCM-518 API (e.g., you are developing for DOS, etc.), it is recommended that you create functions that mask the handshake protocol from higher software layers. The following C language examples show how to do this. These example functions are used in the command descriptions in later sections of this manual.

4.4.1 Byte handshake

These functions handshake individual bytes to/from the board:

```
#define CRMT  0x80   // command register empty flag
#define DAV   0x40   // data available flag

// Send a byte to the command register
void SendByte(int base_port, int cmd_byte)
{
  while ((inp(base_port + 1) & CRMT) == 0); // wait for empty command register
  outp(base_port, cmd_byte);                // send the byte
}

// Fetch a byte from the data register
int ReadByte(int base_port)
{
  while ((inp(base_port + 1) & DAV) == 0);  // wait for full data register
  return (inp(base_port));                  // read the byte
}
```

4.4.2 Endianness

Unless otherwise specified, multi-byte values are big endian. As a result, functions such as the following can be used to handshake 16-bit values to/from the board:

```
// Send a 16-bit value to the board
void SendWord(int base_port, int cmd_word)
{
  SendByte(base_port, cmd_word >> 8);     // send high byte
  SendByte(base_port, cmd_word & 0xFF);   // send low byte
}

// Fetch a 16-bit value from the board
int ReadWord(int base_port)
{
  int hiByte = ReadByte(base_port) << 8;  // fetch high byte
  return (hiByte | ReadByte(base_port));  // fetch & concatenate low byte
}
```

Chapter 5: Commands

5.1 Overview

The 518 board is controlled and monitored by sending it commands. Each command consists of at least one byte, but commands vary in length depending on the size of associated data. In some cases, the board will respond to a command by sending data back to the host. In such cases, the host must read the response before issuing another command.

5.1.1 Conventions

These conventions are used when specifying commands and responses:

1. CHAN represents a channel number. Valid channel numbers range from 0 through 7.
2. A byte is indicated by a number or expression contained by parenthesis. For example, (16 + CHAN) represents a byte value equal to 16 plus a channel number.
3. A byte string is a sequence of bytes separated by commas. For example, the byte string (95+CHAN),(5),(0) contains three bytes sent in order from left to right.
4. A command consists of a byte string sent from host to 518.
5. A response consists of a byte string sent from 518 to host.

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

5.2 Basic operation

5.2.1 DeclareSensorType

This two-byte command configures a channel for a particular sensor type. Typically, it is executed once per channel during application initialization (i.e., after a reset). Each invocation specifies the sensor type for one channel, so eight invocations are required to declare all eight channel sensors. No response is issued by the board.

After invoking this function, sensor data will not be available from the channel until the board has been allowed time to scan all channels.

The first command byte contains the opcode and channel number. The second byte contains a sensor type code (see Sensor specifications for a list of sensor type codes). Declaring a sensor type as “disabled” inhibits scanning of the channel, resulting in increased throughput for other channels.

Command: (16 + CHAN), (SENSOR_TYPE_CODE)

Response: NONE

Example:

```
// Configure channel 2 for a K thermocouple
int chan = 2;                // Sensor channel 2
int port = 0x2B0;            // 518 base address
SendByte(port, 0x10 + chan); // send DeclareSensorType command and channel number
SendByte(port, 0x1C);        // specify K thermocouple type code
```

5.2.2 ReadChannel

This command returns the most recently acquired sensor data from a channel. The returned integer value is scaled according to the previously declared sensor type. See Sensor specifications for a list of sensor scale factors.

Command: (CHAN)

Response: (HIGH_DATA), (LOW_DATA)

Example:

```
// Channel 6 is configured for +/- 5 Volt measurement.
// Read and display the voltage measured at channel 6.
int chan = 6;
double scalar = 0.0005;  // from Sensor specifications (scalar = 500 uV)
SendByte(port, chan);    // send ReadChannel command and channel number
printf("Channel 6 = %f Volts\n", ReadWord(port) * scalar);
```

Example:

```
// Channel 4 is configured for a type K thermocouple.
// Read and display the temperature measured at channel 4.
SendByte(port, 4);
printf("Channel 4 = %f degrees C\n", ReadWord(port) * 0.17);
```

5.2.3 ReadAllChannels

This command returns the most recently acquired sensor data from all eight channels. Sensor data is scaled according to the previously declared sensor type on each channel. See Sensor specifications for a list of sensor scale factors. In the case of disabled channels, the returned value is meaningless.

Command: (88)

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)

Example:

```
// Read scaled sensor data from all channels into array.
int data[8];
int chan;
SendByte(port, 88); // send ReadAllChannels command
for (chan = 0; chan < 8; chan++)
  data[chan] = ReadWord(port);
```

5.3 Alarm/fault commands

5.3.1 SetOpenValues

This command establishes open-sensor data values for all thermocouple channels (the board will detect open sensor condition on channels that have thermocouple shunts installed). This is useful for triggering alarms when open sensors are detected, and for forcing the desired system response to fault conditions in closed-loop process control applications.

The second command byte contains bit flags (one per channel) that specify whether each channel will indicate the minimum or maximum possible value upon open-sensor detection:

FLAGS byte:

Bit	7	6	5	4	3	2	1	0
Channel	7	6	5	4	3	2	1	0

When an open sensor is detected, a channel's FLAGS bit determines the returned data value:

FLAGS bit	Channel data
0	0x8000 (-32768)
1	0x7FFF (32767)

Command: (80), (FLAGS)

Response: NONE

Example:

```
// A closed-loop heat controller monitors temperature via a thermocouple
// on channel 1. If the thermocouple opens, the 518 should indicate a high
// temperature to ensure that the heater turns off. This code sets up
// channel 1 to fail high (all other channels will fail low).
SendByte(port, 80);     // send first command byte
SendByte(port, 0x02);   // send FLAGS byte: (0,0,0,0,0,0,1,0)
```

5.3.2 SetLimits

This command declares upper and lower alarm limits for a channel. An alarm will “sound” if a channel's sensor data strays outside of the specified limits.

Alarm limits assume default values when the board is reset. All low limits default to -32768 (0x8000), and high limits default to 32767 (0x7FFF); these values effectively disable the alarm function. You may specify -32768 to disable the lower limit and 32767 to disable the upper limit.

When an alarm sounds, the status register ALARM bit is set and the violated limit is reset to its default value, thus preventing the limit from triggering another alarm. The host must reprogram the violated limit to re-enable it.

Command: (32 + CHAN), (HIGH_LIM_MSB), (HIGH_LIM_LSB), (LOW_LIM_MSB), (LOW_LIM_LSB)

Response: NONE

Example:

```
// Channel 7 is monitoring a K thermocouple that must measure between 400 and
// 450 degrees C. The following code programs limits so that the ALARM flag will
// be set to '1' if the temperature strays outside the normal operating range.
double scalar = 0.17;                 // K thermocouple scalar (from Sensor specifications table)
SendByte(port, 7 + 32)                // send SetLimits command and channel number
SendWord(port, (int)(450.0 / scalar)); // send high limit
SendWord(port, (int)(400.0 / Scalar)); // send low limit
```

5.3.3 ReadAlarms

This command reports the alarm status of all channels and then resets all channel alarm flags and the ALARM status bit. The first response byte indicates high alarms and the second byte indicates low alarms. Bit 7 (most significant) through bit 0 (least significant) of each byte corresponds to channel 7 through 0, respectively. A status bit containing '1' indicates a violated limit; '0' indicates no limit violation.

Command:(48)

Response: (HIGH ALARM LIMIT FLAGS),(LOW ALARM LIMIT FLAGS)

Example:

```
// Read and display alarms.
int chan, mask, hiflags, loflags;
SendByte(port, 48); // send ReadAlarms command
hiflags = ReadByte(port); // read high alarm flags
loflags = ReadByte(port); // read low alarm flags
for (chan = 0, mask = 1; chan < 8; chan++, mask <<= 1) {
  if (hiflags & mask) printf("Channel %d high alarm\n", chan);
  if (loflags & mask) printf("Channel %d low alarm\n", chan);
{
```

5.4 Gauge commands

5.4.1 SetGaugeZero

This command informs the board that there currently is no load on the channel's load cell. Typically, it is invoked just prior to a SetGaugeSpan command.

Command: (176 + CHAN)

Response: NONE

Example: See Setting zero and span.

5.4.2 SetGaugeSpan

This command establishes gauge sensitivity. It is typically invoked just after executing a SetGaugeZero command. Before invoking this command you must apply (or simulate) a known load. For best accuracy, apply the maximum rated load; if this is not possible, apply a significant load. The two data bytes comprise the signed 16-bit integer value expected from a ReadChannel command at the applied load.

Command: (208 + CHAN), (DATA MSB), (DATA LSB)

Response: NONE

Example: See Setting zero and span.

5.4.3 TareGauge

This command will tare a strain/pressure gauge channel. Typically, this is used to compensate the weight of a container by invoking it the container is empty. Before invoking this command, the gauge channel should be calibrated. See the SetGaugeZero and SetGaugeSpan commands for details.

Command: (112 + CHAN)

Response: NONE

Example:

```
// An empty truck is to be tared prior to loading. The truck scale
// is monitored by load cell connected to channel 7. Channel 7 has been
// previously calibrated using the SET gauge ZERO and SET gauge SPAN commands.
SendByte(port, 112 + 7);  // issue command: (opcode + channel)
```

5.4.4 ReadGaugeCalibration

This command returns a strain gauge's calibration. The strain gauge must have been previously calibrated via SetGaugeZero and SetGaugeSpan commands, or by invoking a SetGaugeCalibration command.

Command: (128 + CHAN)

Response: (S0), (S1), (S2), (S3), (S4), (S5)

Example:

```
// Reads a channel's gauge calibration (into array). The channel must be configured
// for a strain gauge sensor and must be calibrated before calling this function.
ReadGaugeCal(int port, int chan, unsigned char *array)
{
  int i;
  SendByte(port, 128 + chan);
  for (i = 0; i < 6; i++)
    array[i] = ReadByte(port);
}
```

5.4.5 SetGaugeCalibration

This command activates a predetermined gauge calibration on a channel that has been configured for gauge measurement. The calibration data must have been previously obtained via a ReadGaugeCalibration command.

Note: A SetGaugeCalibration command should not be immediately followed by ReadGaugeCalibration command. The minimum interval between these commands should be at least 500 milliseconds to ensure proper operation of the ReadGaugeCalibration command.

Command: (144 + CHAN), (S0), (S1), (S2), (S3), (S4), (S5)

Response: NONE

Example:

```
// Send a channel's gauge calibration (in array) to the board.
// The channel must be configured for a strain gauge sensor.
WriteGaugeCal(int port, int chan, unsigned char *array)
{
  int i;
  SendByte(port, 144 + chan);
  for (i = 0; i < 6; i++)
    SendByte(port, array[i]);
}
```

5.5 Miscellaneous commands

5.5.1 ReadFirmwareVersion

This command returns the firmware version number (times 100).

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

Response: (VERS_MSB), (VERS_LSB)

Example:

```
// Read and display the firmware version.
SendByte(port, 240);
SendByte(port, 5);
SendByte(port, 0);
printf("Vers %f", ReadWord(port) / 100.0); // should display something like "Vers 2.25"
```

5.5.2 ReadBoardTemperature

This command returns the temperature of an optional, external reference sensor (e.g., on a 7409TB termination board), which is required when thermocouples are being measured. This function can also be used as a debug aid during 518 installation. The returned integer value is scaled to 0.10 °C/bit. If no reference temperature sensor is connected, this command will return a meaningless value.

Command: (64)

Response: (TEMP_MSB), (TEMP_LSB)

Example:

```
// Read and display the reference temperature
SendByte(port, 64);   // send ReadBoardTemperature command
printf("Ref temp = %f degrees C", ReadWord(port) * 0.1);
```

5.5.3 ReadModel

This command returns the board's model number.

Command: (240), (4), (0)

Response: (ID_MSB), (ID_LSB)

Example:

```
// Read and display the model number.
SendByte(port, 240);   // send ReadModel command
SendByte(port, 4);
SendByte(port, 0);
printf("Model %d", ReadWord(port));  // should display "Model 518"
```

5.5.4 Calibrate

This command calibrates one of the board's three internal standards. To perform a calibration, you must supply two reference voltages (5 V and 500 mV) and a reference resistance (380 ohms). Standards must be calibrated in the following order: 5 V, 500 mV, 380 ohms. The board must be reset following each invocation to activate the calibration.

The first command byte specifies the sensor channel that is connected to your external reference. The second byte specifies the standard to calibrate:

CALCODE	Standard to calibrate
0	5 V
1	500 mV
2	400 ohm

A single byte is returned when the calibration has completed; it's value has no meaning.

Command: (224 + CHAN), (CALCODE), (DATA_MSB), (DATA_LSB)

Response: (DON'T_CARE)

Example:

```
// Connect 5V to channel 0, 500mV to channel 1, and 380 ohms to channel 2.

// Enter the actual reference values in the ref[] array:
double ref[] = {5.0000, 500.00, 380.00};  // CHANGE AS REQUIRED

int scalar[] = {5000, 50, 40};       // scalars: V, mV, ohms
int sensor[] = {0x15, 0x16, 0x0A};   // sensor type codes
int chan;     // channel number == CALCODE

for (chan = 0; chan < 3; chan++) {
  SendByte(port, 16 + chan);                        // declare channel sensor type
  SendByte(port, sensor);
  Sleep(500);                                       // wait for valid scan data
  SendByte(port, 224 + chan);                       // issue Calibrate command:
  SendByte(port, chan);                             //   CALCODE
  SendWord(port, (int)(ref[chan] * scalar[chan]));  //   cal data
  ReadByte(port);                                   // wait for calibrate to complete
  outp(port + 1, 0);                                // reset 518 board
  Sleep(500);                                       // wait for board to reboot
}
```

Chapter 6: Sensor specifics

6.1 Thermocouples

A thermocouple consists of two conductors made of dissimilar metals, which are typically built into a cable. At one end of the cable the conductors are shorted together and thermally coupled to the point to be measured; this is known as the *hot junction*. At the other end the conductors are electrically connected to a measurement system; these connections are known as the *cold junction*.

A thermocouples generates a voltage that is proportional to the temperature difference between its hot and cold junctions. Consequently, the cold junction temperature must be known in order to determine the hot junction temperature. To facilitate this, the measurement system must provide a means for measuring the cold junction temperature. In the case of PCM-518, this function is provided by an external termination board such asWinSystem's model 7409TB.

A temperature transducer resides on the 7409TB termination board for the purpose of measuring thermocouple cold junctions. PCM-518 periodically measures this transducer and computes the corresponding thermocouple voltage (the *compensation voltage*) for all thermocouple types. When a thermocouple is scanned, the measured voltage is first "corrected" by adding the compensation voltage (thus calculating the voltage one would measure if the cold junction was 0 °C) and then converted to temperature.

6.1.1 Accuracy

Measurement accuracy depends in large part on accurately measuring thermocouple cold junction temperature. Consequently, it is essential that cold junctions be maintained at the same temperature as the termination board's temperature transducer, to the extent possible. For best results, insulate the cold junctions from thermal transients and from breezes that might cause temperature gradients across the termination board.

6.2 Gauges

A strain gauge is a device that changes its electrical resistance in response to deformation. Typically it is one of the resistance elements in a Wheatstone bridge and is mounted to a load cell. In operation, a voltage is applied to the resistance bridge and a force is applied to the load cell that causes gauge deformation, which results in an output voltage from the bridge. In the case of a pressure gauge, the magnitude of the force is related to fluid (e.g., gas, liquid) pressure.

When a sensor channel is configured for gauge measurement, the board will automatically apply excitation in the form of a DC voltage strobe, with an amplitude of approximately 10 V. Excitation is applied only while the channel is being scanned; no excitation is applied when other channels are being scanned.

Model 518 requires the gauge to be part of a full bridge circuit, with a bridge input resistance of at least 120 Ω. Furthermore, the bridge output voltage must fall within the range -500 mV to +500 mV under all load conditions (including offset due to bridge imbalance).

6.2.1 Resolution

Every gauge has a characteristic ratio R of output voltage to input voltage at a particular applied force F. The following formula can be used to calculate measurement resolution:

$$resolution = \frac{F \cdot 5 \cdot 10^{-6}}{R}$$

For example, in the case of a gauge having 3 mV/V gain at 100 N force, the board can resolve the load to:

$$resolution = \frac{100\text{N} \cdot 5 \cdot 10^{-6}}{0.003} = 0.017\text{N}$$

6.2.2 Setting zero and span

Gauge channels must undergo a two-point calibration procedure in which zero and maximum loads are physically applied or electrically simulated. The board "forgets" gauge calibrations when reset or powered down, so initial calibration is required for each gauge channel. A channel should be calibrated once after the application program has configured it for gauge measurement (via DeclareSensorType command), and again as necessary to compensate for drift due to temperature, time and mechanical changes.

Follow this procedure to calibrate a gauge channel:

1. Create a zero load condition by either physically applying or simulating a zero load, then invoke a SetGaugeZero command to establish the gauge zero point.
2. Create a full load condition by either physically applying or simulating a maximum load, then invoke a SetGaugeSpan command to establish the gauge span.

Example:

```
// Calibrate the load cell connected to channel 1. In this example the full rated load
// is 40,000 pounds. Since 40,000 exceeds the range of a 16-bit integer, we will scale
// the output units to 10 pounds/bit, resulting in a full-load output of 4,000 counts.

CMD_ZERO = 176; // SetGageZero opcode
CMD_SPAN = 208; // SetGageSpan opcode
int chan = 1;
int maxload = 4000; // scaled data at full load

printf("Apply zero load to gauge and press any key when ready .. ");
WaitForKeypress();
SendByte(port, CMD_ZERO + chan); // send ZERO command

printf("Now apply 40,000 pound load and press any key when ready .. ");
WaitForKeypress();
SendByte(port, CMD_SPAN + chan); // send SPAN command
SendWord(port, maxload);
```

After completing the above procedure, a ReadGaugeCalibration command may be invoked to obtain the resulting gauge calibration. The data returned by this command may be saved for future use; for example, it may be used for the next initial calibration (via SetGaugeCalibration, in lieu of SetGaugeZero and SetGaugeSpan) following a board reset or power-up.

6.2.3 Taring

The board will automatically subtract the tare weight of a load container from the measured gross weight. To enable this, do the following:

1. Position the empty container/vehicle on the load cell.
2. Issue a TareGauge command to the board. Subsequent channel data will indicate the corrected payload weight.

Chapter 7: Timing

7.1 Warm-up

Sensor data may be noisy when the board is warming up or subjected to thermal transients. This is caused by rapidly changing circuit gains and offsets that cannot be compensated quickly enough by internal standardization cycles. It is characterized by sensor data that cyclically "jumps" and then drifts for approximately seven seconds.

Warm-up noise will subside when the board reaches thermal stability. For best accuracy, it is recommended to allow time for the board to reach thermal stability and to protect it from sudden temperature changes.

7.2 Sampling

The channel sampling rate (samples per second per channel) depends on the number of active channels: it will increase as the number of active channels decrease. A channel may be deactivated by declaring it disabled (sensor type code 0x13) with the DeclareSensorType command.

To avoid reading the same sample twice, it is necessary to allow enough time between reads for the board to scan all active channels plus one internal reference standard:

$$t_{minimum} = 22 \times (A + 1)$$

where: t = milliseconds between reads
A = number of active channels

7.3 Communication

Communication timing is determined by two parameters:

Parameter	Value	Description
Command execution time	30 µs	Time needed to perform the requested operation.
Byte handshake time	20 µs	Time needed to exchange each command/response byte with the board.

Excluding any host computer overhead, the maximum time needed to execute a command is:

$$t_{maximum} = 30 + 20 \times (C + R)$$

where: t = communication time in microseconds
C = number of command bytes
R = number of response bytes

Examples:

Command	Number of bytes		Total time
	Command	Response	
ReadChannel	1	2	30 + 20 * (1 + 2) = 90 µs
ReadAllChannels	1	16	30 + 20 * (1 + 16) = 350 µs

Chapter 8: Specifications

8.1.1 General specifications

Parameter		Specification
Sensor channels	Number	8
	Type	Independently configurable for any sensor type
ADC	Type	Integrating, V/F
	Resolution	Variable
	Conversion time	16.67 ms, nominal
Sampling rate	Aggregate	45 samples/s
	Per channel	5.6 samples/s (all channels active)
Analog inputs	Type	Differential
	Differential voltage range	-5 to +5 V
	CMV range	-5 to +5 V
	CMRR	78 dB @ ≤ 60 Hz, min
	Absolute maximum voltage	±20 V
	Input impedance	1000 MΩ, nominal
Sensor excitation (pulsed)	RTD / 400 Ω range	1.3 mADC
	Other resistance ranges	5 VDC || 4 K Ω
	Gauge	10 VDC
System bus	Type	PC/104
Input power (nominal)	Rev A-U boards	+5 VDC ±5% @ 100 mA +12 VDC ±5% @ 45 mA -12 VDC ±5% @ 35 mA
	Rev V+ boards	+5 VDC ±5% @ 300 mA
Temperature	Operating	0 to +70 °C
	Storage	-25 to +85 °C

8.1.2 Sensor specifications

Sensor			Range	Accuracy	Output data scaling/bit
Type		Code			
Thermocouple	B	0x24	0 to 1820 °C	3.3 °C	0.1 °C
	C	0x23	0 to 1820 °C	2.1 °C	
	E	0x01	-270 to 990 °C	0.8 °C	
	J	0x1B	-210 to 760 °C	0.6 °C	
	K	0x1C	-270 to 1360 °C	1.0 °C	
	N	0x22	-270 to 1347 °C	0.9 °C	
	T	0x1D	-270 to 400 °C	0.6 °C	
	S	0x1E	0 to 1760 °C	3.0 °C	
	R	0x1F	0 to 1760 °C	2.8 °C	
Thermistor	10 KΩ 44006 / 44031	0x25	-55 to 62 °C 62 to 126 °C 126 to 150 °C	0.01 °C 0.025 °C 0.008 °C	0.01 °C
RTD	Pt 100, α = 0.385	0x18 0x2A	-200 to 800 °C -200 to 400 °C	0.2 °C 0.2 °C	0.05 °C 0.0125 °F (~0.007 °C)
	Pt 100, α = 0.392	0x19 0x2B	-200 to 800 °C -200 to 400 °C	0.2 °C 0.2 °C	0.05 °C 0.0125 °F (~0.007 °C)
	Ni 200, α = 1.098	0x28	-60 to 180 °C	0.08 °C	0.05 °C
	Ni 1000, α = 4.4	0x29	-50 to 70 °C	0.02 °C	0.05 °C
	Cu 10, α = 0.039	0x2C	0 to 125 °C	0.5 °C	0.1 °C
Gauge	Full bridge, ≥ 120 Ω	0x0F	±500 mV	30 µV	Programmable (see section 6.2.1)
DC voltage		0x15	±5 V	400 µV	200 µV
		0x16	±500 mV	30 µV	20 µV
		0x17	±100 mV	30 µV	5 µV
Current loop		0x11	4-to-20 mA	0.02 %	0.01 % (4 mA=0%, 20 mA=100%)
Resistance		0x0A	0 to 400 Ω	0.04 Ω	0.02 Ω
		0x14	0 to 4 KΩ	0.25 Ω	0.125 Ω
		0x20	0 to 600 KΩ	130 Ω	31 Ω
Disabled channel		0x13	n/a	n/a	n/a

Chapter 9: Limited warranty

Full warranty information can be found at https://winsystems.com/company-policies/warranty/.