

***Not Recommended for New Installations.***

Please contact Technical Support for more information.

***Serial Port to Data Acquisition  
Control Module***

**Model SPIO**

**Documentation Number SPIO1495**

**B&B Electronics Mfg. Co. Inc.**

P.O. Box 1040 -- Ottawa, IL 61350

PH (815) 433-5100 -- FAX (815) 433-5105

**Internet:**

<http://www.bb-elec.com>

[orders@bb-elec.com](mailto:orders@bb-elec.com)

[support@bb.elec.com](mailto:support@bb.elec.com)

© 1993 B&B Electronics

Revised June 1993

# TABLE OF CONTENTS

<b>CHAPTER 1: INTRODUCTION</b> .....	<b>1</b>
SPIO FEATURES .....	1
PACKING LIST .....	1
FILES INCLUDED ON DISK .....	1
<b>CHAPTER 2: SPIO OPERATIONS</b> .....	<b>3</b>
OVERVIEW .....	3
COMMUNICATIONS.....	3
SETUP MODE .....	3
ACTIVE LED.....	6
POWER RESET.....	6
SERIAL PORT TO DATA ACQUISITION MODULE .....	7
<i>Overview</i> .....	7
<i>SPIO Specifications</i> .....	8
A/D INPUTS .....	8
INTERFACE MODULES FOR SPIO .....	9
<i>DAPBI</i> .....	9
<i>SDIOB8</i> .....	11
<i>SDIOB8 Specifications</i> .....	13
DIGITAL I/O - INPUTS.....	13
<i>Normal Inputs</i> .....	17
<i>Event Counters</i> .....	17
<i>Latches</i> .....	17
DIGITAL I/O - OUTPUTS.....	19
<i>Normal Outputs</i> .....	19
<i>Time Delayed Outputs</i> .....	19
<i>Pulse Width Modulation</i> .....	23
ANALOG SECTION .....	24
<i>Overview</i> .....	24
<i>A/D Channels</i> .....	24
<i>A/D Thermocouple Application</i> .....	25
<i>D/A Channels</i> .....	27
POWER SUPPLY.....	28
<b>CHAPTER 3: INSTALLATION</b> .....	<b>29</b>
OVERVIEW .....	29
POWER CONNECTIONS .....	29
RS-232 CONNECTIONS .....	29
SIGNAL NAME .....	30
SIGNAL NAME .....	30
UPDATING SPIO FIRMWARE.....	31

<i>Why Reprogram?</i> .....	31
<i>Setting Up An Update Session</i> .....	31
<i>Command Line Switches</i> .....	32
<i>What The Program Does</i> .....	32
<i>What If Something Goes Wrong?</i> .....	33
CONFIGURATION PROGRAM .....	34
<i>Configuring your module</i> .....	34
<i>Communication Port</i> .....	34
<i>Interrupt Request (IRQ) Number</i> .....	35
<i>Communication Errors</i> .....	35
<i>Installing the software</i> .....	42
<b>CHAPTER 4: COMMANDS</b> .....	<b>44</b>
OVERVIEW .....	44
COMMAND MESSAGE.....	44
REPLY MESSAGES .....	45
CHECKSUM .....	45
TWO-STEP & FOUR-STEP COMMUNICATIONS .....	46
<i>Two-Step Communications</i> .....	46
<i>Four-Step Communications</i> .....	47
INPUT/OUTPUT SECTION .....	48
ERROR CODES .....	49
COMMAND SUMMARY LIST .....	50
COMMAND FORMAT .....	52
GENERAL COMMANDS.....	53
<i>Read/Set SPIO Configuration</i> .....	53
<i>Power-Up Clear</i> .....	54
<i>Reset</i> .....	55
<i>Turn-Around Delay</i> .....	55
<i>Communication Timer</i> .....	56
<i>Set Communications Mode</i> .....	57
<i>Identify Module</i> .....	57
DIGITAL COMMANDS .....	58
<i>Read/Set Digital I/O Type</i> .....	58
INPUTS .....	58
<i>Read/Set Digital Power Up States</i> .....	61
<i>Read/Set Output Timer Values</i> .....	63
<i>Configure Digital I/O Lines</i> .....	64
<i>Configure Digital I/O as Input</i> .....	65
<i>Configure Digital I/O as Output</i> .....	66
<i>Set Digital Outputs ON or OFF</i> .....	66
<i>Turn Digital Output ON</i> .....	67
<i>Turn Digital Output OFF</i> .....	68
<i>Read Digital I/O States</i> .....	68

<i>Set Latch Edges</i> .....	69
<i>Set Off-to-On Latches</i> .....	70
<i>Set On-to-Off Latches</i> .....	71
<i>Read Latches</i> .....	72
<i>Read And Clear Latches</i> .....	72
<i>Clear Latches</i> .....	73
<i>Start/Stop Counters</i> .....	74
<i>Start Counters</i> .....	74
<i>Stop Counters</i> .....	75
<i>Read Counters</i> .....	76
<i>Read And Clear Counters</i> .....	77
<i>Clear Counters</i> .....	78
<i>Set Time Delay</i> .....	79
<i>Retrigger Time Delay</i> .....	80
<i>Read Digital I/O Configuration</i> .....	81
<b>ANALOG COMMANDS</b> .....	<b>82</b>
<i>Analog Data</i> .....	82
<i>Analog Inputs</i> .....	82
<i>Analog Outputs</i> .....	83
<i>Set Analog Output Level</i> .....	83
<i>Read Analog Output Level</i> .....	84
<i>Read Analog Input Level</i> .....	85
<i>Set Individual Analog Output Levels</i> .....	86
<i>Read/Set Analog Output Power Up Levels</i> .....	87
<b>CHAPTER 5: APPLICATION PROGRAM INTERFACE</b> .....	<b>89</b>
<b>PROGRAMMING WITH QUICKBASIC</b> .....	<b>89</b>
<b>PROGRAMMING WITH C</b> .....	<b>90</b>
<b>PROGRAMMING WITH PASCAL</b> .....	<b>92</b>
<b>STATUS CODE SUMMARY</b> .....	<b>93</b>
<b>FUNCTIONAL OVERVIEW</b> .....	<b>95</b>
<i>API_Version</i> .....	96
<i>Clear_Counters</i> .....	96
<i>Clear_Latches</i> .....	97
<i>Comm_Timer</i> .....	97
<i>Config_Digital_Lines</i> .....	98
<i>Config_Digital_Inputs</i> .....	99
<i>Config_Digital_Outputs</i> .....	100
<i>End_API</i> .....	101
<i>Get_Data</i> .....	101
<i>Get_Message</i> .....	102
<i>Get_Reply</i> .....	105
<i>Identify_Module</i> .....	105
<i>Module_Reset</i> .....	106

<i>Power_Up_Clear</i> .....	106
<i>Read_And_Clear_Counters</i> .....	107
<i>Read_And_Clear_Latches</i> .....	108
<i>Read_Counters</i> .....	108
<i>Read_Counters_With_Overflow</i> .....	109
<i>Read_Digital_Config</i> .....	110
<i>Read_Input_Level</i> .....	110
<i>Read_Latches</i> .....	111
<i>Read_Module_Config</i> .....	112
<i>Read_Output_Level</i> .....	113
<i>Read_Power_Up_Levels</i> .....	113
<i>Read_States</i> .....	115
<i>Read_Output_Timer_Values</i> .....	115
<i>Read_Digital_Types</i> .....	116
<i>Retrigger_Time_Delay</i> .....	117
<i>Set_Analog_Outputs</i> .....	117
<i>Set_API_Timeout</i> .....	118
<i>Set_Com_Port</i> .....	119
<i>Set_Communication_Mode</i> .....	120
<i>Set_Digital_Types</i> .....	121
<i>Set_Indiv_Levels</i> .....	123
<i>Set_Latch_Edges</i> .....	124
<i>Set_Module_Config</i> .....	125
<i>Set_Off_To_On_Latches</i> .....	126
<i>Set_On_To_Off_Latches</i> .....	127
<i>Set_Output_State</i> .....	127
<i>Set_Output_Timer_Values</i> .....	128
<i>Set_Power_Up_Levels</i> .....	129
<i>Set_Power_Up_States</i> .....	130
<i>Set_Protocol</i> .....	131
<i>Set_Retries</i> .....	131
<i>Set_Time_Delay</i> .....	132
<i>Specify_Indiv_Levels</i> .....	134
<i>Start_API</i> .....	134
<i>Start_Counters</i> .....	135
<i>Start_Stop_Counters</i> .....	136
<i>Stop_Counters</i> .....	136
<i>Turn_Around_Delay</i> .....	137
<i>Turn_Output_Off</i> .....	138
<i>Turn_Output_On</i> .....	138
<i>Unspecify_Indiv_Levels</i> .....	139

**APPENDIX A: ASCII CHARACTER CODES.....A-1**

**APPENDIX B: HEX/DECIMAL CONVERSIONS.....B-1**

**APPENDIX C: I/O PORT PIN FUNCTIONS..... C-1**

**APPENDIX D: DIAGRAMS.....D-1**

FIGURE D-1. SPIO.....D-1

FIGURE D-2. SDIOB8.....D-2

FIGURE D-3. DAPB1 .....D-3

**APPENDIX E: HANDLING UNUSED I/O LINES ..... E-1**

**APPENDIX F: QBASIC DEMO PROGRAM..... F-1**

# Chapter 1: INTRODUCTION

## SPIO Features

The SPIO is a general purpose control module that is connected to your computer's RS-232 Serial Port. It can be used to sense a variety of external conditions and control, via a host computer, a variety of external devices.

I/O lines are available via a female DB-25 connector. See Appendix C , Pin Functions. The SPIO communicates to a host computer using the RS-232 Standard. See to Figure 1-1. Standard baud rates from 300 to 9600 with adjustable data formats can be used.

All of the SPIO's configuration parameters can be set easily via the host computer. These parameters are then stored in non-volatile memory. This means that the module's setup information is retained even when power is removed from the module. The SPIO provides a setup mode that returns it to known factory defaults if the user configuration is lost.

## Packing List

Examine the shipping carton and contents for physical damage. If damage is found, file a claim with B&B Electronics immediately.

The following items should be in the shipping carton:

1. SPIO module
2. Manual with/diskette

If any of these items is missing contact B&B Electronics.

## Files Included on Disk

The following files are included on the disk that is shipped with your SPIO module:

- READ.ME - text file that lists, in chronological order, changes or revisions to the manual or programs.
- INSTALL.BAT - a batch file that will uncompress the ".ZIP" file and install it on your IBM PC or compatible computer. Refer to Chapter 4, Installation, for specific file names.
- SPIOV21.ZIP - a compressed file which includes all the programs for the SPIO module.

# Chapter 2: SPIO OPERATIONS

## Overview

The SPIO is a self-contained, intelligent, general purpose control module. It can be used to control a variety of devices from a host computer via a single RS-232 communication line. The module's program and setup parameters are stored in EEPROM. This means that the module's setup information is retained even when power is removed from the module. Setup parameters can be configured remotely from the host computer. As program updates become available the SPIO firmware can easily be updated on site by the user (refer to Chapter 3).

## Communications

The SPIO can communicate with the host over an RS-232 communication link. This requires the host to be RS-232 compatible. Standard baud rates from 300 to 9600 with a variety of data formats can be used (refer to Chapter 4, “%” Command) to match the user's requirements. See Figure 2-1.

## Setup Mode

The SPIO has a SETUP JUMPER that forces the SPIO to its factory default settings (refer to the particular SPIO module's factory default table in this chapter). This mode can be used to read the module's user defined parameters if they have been forgotten. It is important that you disconnect all the SPIO's I/O lines BEFORE entering the SETUP mode. This will prevent any external device from turning on accidentally. To enter the setup mode follow the steps below:

1. Disconnect power to SPIO Module.
2. Disconnect all external devices/interfaces from SPIO's I/O port.
3. Remove bottom cover of the SPIO.
4. Insert the SETUP JUMPER. See Figure 2-2.
5. Put bottom cover back on the SPIO.
6. Connect power to SPIO Module.

**IMPORTANT:** Do not touch any components while the cover is removed.

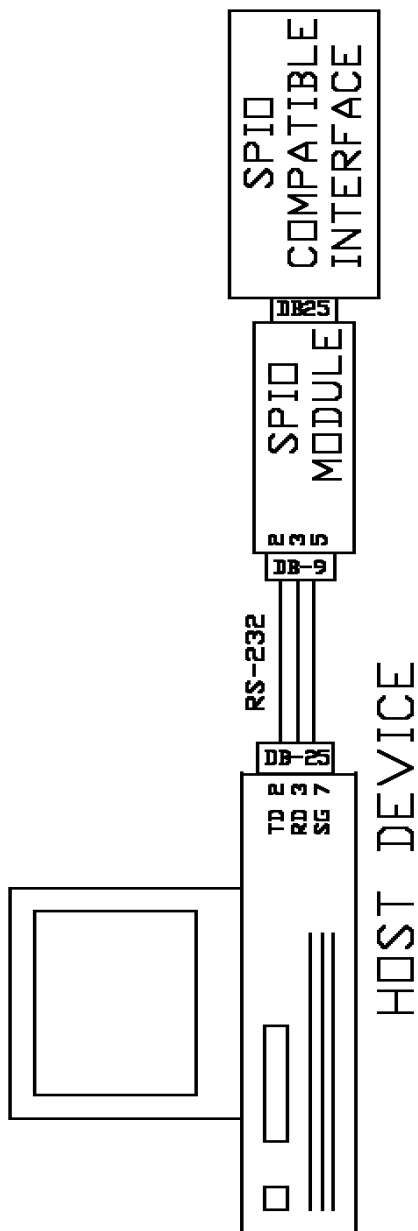
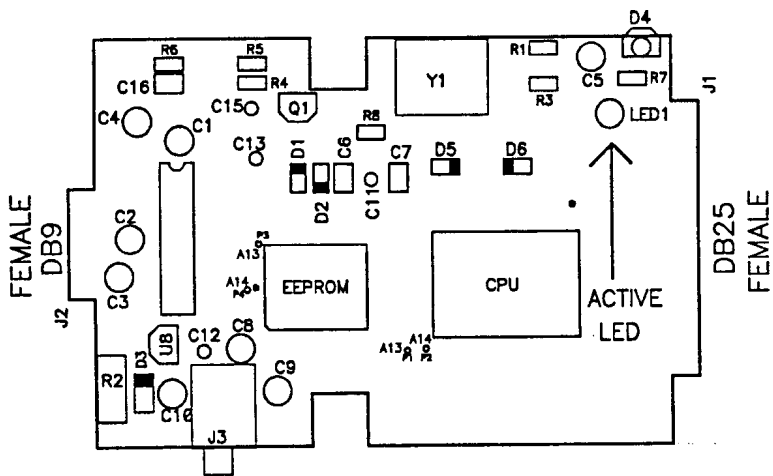
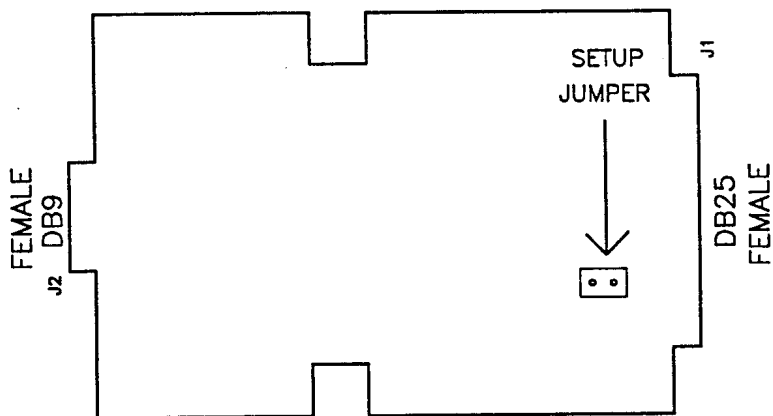


Figure 2.1



TOP VIEW



BOTTOM VIEW

Figure 2.2

Inserting the SETUP JUMPER will not alter the user defined parameters stored in the EEPROM. Set the host communications to match the default values of the SPIO. The module's user defined parameters can now be read by sending the appropriate commands (refer to Chapter 4). To exit the setup mode follow the steps below:

1. Disconnect power to SPIO Module.
2. Remove bottom cover of the SPIO.
3. Remove the SETUP JUMPER. See Figure 2-2.
4. Put bottom cover back on the SPIO.
5. Make all necessary connections of external devices interfaces to SPIO's I/O port.
6. Connect power to SPIO Module.

### **Active LED**

Located on the SPIO module is a red LED labeled "ACTIVE." (Refer to Figure 2-2). When power is applied to the SPIO, the ACTIVE LED will blink at a 1 second rate indicating it is operating properly. If, during normal operation, the ACTIVE LED stops blinking a problem has developed with the module and the level of outputs will be unknown.

### **Power Reset**

When an SPIO module initially has power applied to it, all output lines will be set to a predefined power up level (refer to POWER UP commands in Chapter 4). This also will happen anytime the SPIO experiences a power loss and power is then reapplied. This feature allows the user to define a level for each output that will leave the system in a safe operating condition at power up.

## Serial Port to Data Acquisition Module

### Overview

The SPIO is a general purpose control module that is connected to your computer's RS-232 Serial Port with a DB9S (female) connector. Configuration and manipulation of the SPIO are performed using a PC and an easy-to-use command set.

I/O lines are available through a female DB-25 connector. Digital input lines are CMOS/TTL compatible. Digital output lines are CMOS compatible.

B&B Electronics manufactures compatible interfaces for the SPIO (see DAPB1, SDIOB8 in this chapter). Listed in Table 2-1 are the factory default settings.

Baud rate	9600
Data bits	8
Parity	None
Stop bits	1
Comm mode	Two-Step
Turn-around delay	10 ms
Comm timer	Disabled
Digital I/O's	Input - normal type
D/A's Power-up level	0 volts

## SPIO Specifications

Size: 3.8"L x 2.4"W x 0.9"H  
Power Requirement: 10 - 18 VDC @ 50 mA  
Operating Temperature: 0 to +70 degrees C

Pin # 14 on the I/O port can be used as an unregulated +12VDC supply (250mA max.)

### Digital Lines: 8 programmable I/O lines.

#### Digital Inputs

Input voltage: 0 to +5VDC  
"0" input voltage: +2.0 to +5.0VDC  
"1" input voltage: 0.0 to +0.8VDC

#### Digital Outputs

Output voltage: +5.0VDC max.  
Output current: 10.0 mA max. per output

**CAUTION: Total output power cannot exceed 0.5 watts.**

**IMPORTANT: Power must be off before any connections are made to the digital lines. Make sure all external devices connected are electrically compatible with these lines. Failure to follow these guidelines could result in damage to the SPIO and void the warranty.**

### Analog I/O: Two D/A Outputs and Eight A/D Inputs

#### A/D Inputs

Resolution: 8 bits  
Absolute Accuracy: +/- 3 LSB max. (+/- 1.5 LSB typ.)  
A/D Input Voltage Range: ANALOG GND to A/D PWR  
A/D REF Voltage: +2VDC min. to +5.5VDC max.  
A/D PWR Voltage: +4.5V min. to +5.5VDC max.

#### D/A Outputs

Resolution: 8 bits  
Output Resistance: 1K min, 2K typical, 4K max.  
D/A REF Voltage: +4VDC min to +5.5VDC max.  
REF Input Current: 2.5 mA typical, 5ma max.  
Full Scale Deviation: 1% max. @ +5VDC D/A REF

**IMPORTANT: Power must be off before any connections are made to the analog section. Make sure all external devices connected are electrically compatible with the analog section. Failure to follow these guidelines could result in damage to the SPIO and void the warranty.**

## **INTERFACE MODULES FOR SPIO**

### **DAPB1**

The DAPB1 connects to your SPIO to provide easy access to the available I/O lines. The DAPB1 has a work space available for any necessary signal conditioning. See Figure 2- 3.

I/O types available:

Inputs - normal inputs, latches, and two event counters  
(software & I/O controlled)

Output - normal outputs, PWM, and time delayed on/off

When the DAPB1 is connected to the SPIO, the current version firmware for the SPIO (SPIOV21.HEX) should be loaded. Refer to Chapter 3 for how to update the SPIO's firmware.

For DAPB1 specifications see SPIO specifications.

**IMPORTANT: Power must be off before plugging the DAPB1 into the SPIO and when any connections are made to the DAPB1. Make sure all external devices connected are electrically compatible with DAPB1. Failure to follow these guidelines could result in damage to the SPIO or DAPB1 and will void the warranty.**

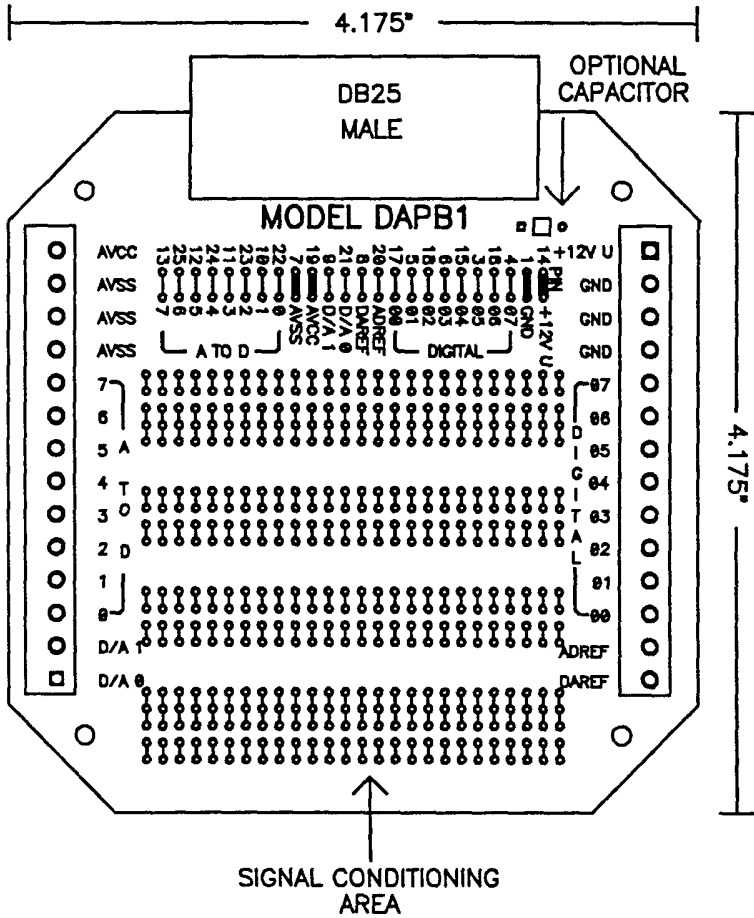


Figure 2-3

## SDIOB8

The SDIOB8 plugs into the B&B Electronics' SPIO module to provide 8 programmable digital input/output lines. See Figure 2-4. Each I/O has an LED status indicator.

I/O types available:

Inputs - normal inputs, latches, and two event counters  
(software & I/O controlled).

Outputs - normal outputs, and time delayed on/off.

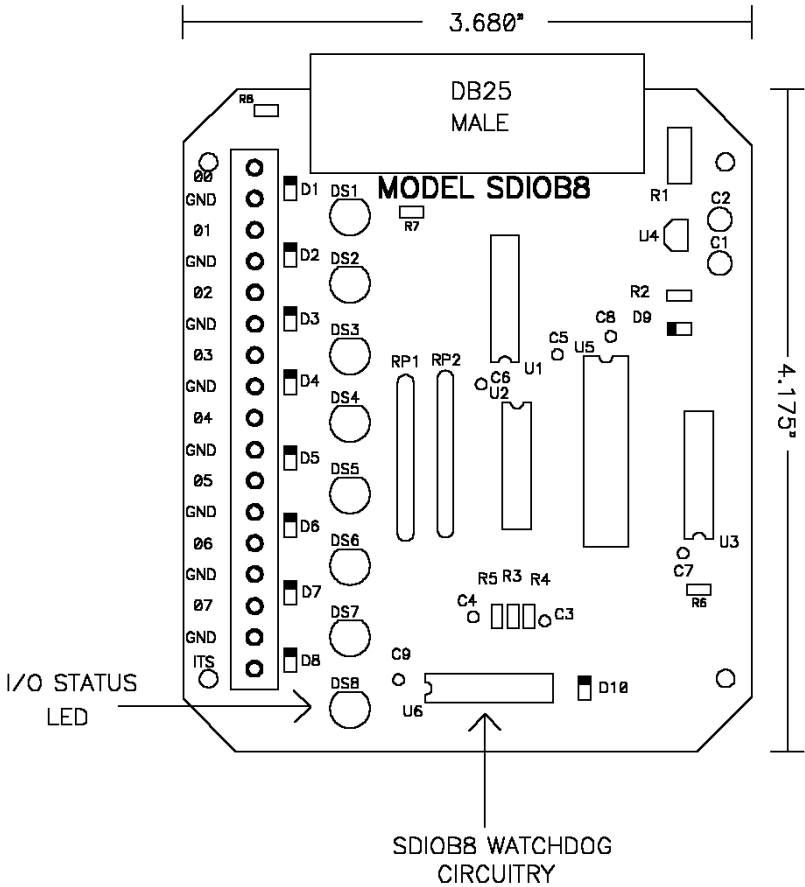
**NOTE:** The PWM is not available on the SDIOB8.

The SDIOB8 has a watchdog circuit that monitors the operation of the SPIO's microprocessor. If the microprocessor develops a problem the watchdog will disable all digital outputs indicating a fault.

All necessary setup and monitoring programs are included with the SPIO module.

When the SDIOB8 is connected to the SPIO, the current version firmware for the SDIOB8 (SDIB8V21.HEX is included with SPIO package) should be loaded. Refer to Chapter 3 for how to update the SPIO's firmware.

**NOTE:** The SDIOB8 is exclusively for use with the B&B Electronics' SPIO Module.



**Figure 2-4**

## SDIOB8 Specifications

Size: 4.2"L x 3.7"W  
Power requirement: About 50mA unregulated 12VDC  
(supplied by SPIO module)  
Operating temperature: 0 to +70 degrees C  
Digital I/O: 8 lines programmable as inputs or outputs

### Digital I/O used as Input

Input voltage: 0 to +50VDC  
Internal pull-up current: 2 mA  
"0" input voltage: +2.5 to +50VDC  
"1" input voltage: 0 to +1.5VDC

### Digital I/O used as Output

Output voltage: +50VDC max  
Output current: 350 mA max. - with only 1 output on  
100 mA max. - all outputs on  
Output leakage current: 50 ua max.  
Output saturation voltage: 1.1VDC max @100mA

**CAUTION: Total digital output power cannot exceed 2 watts.**

**IMPORTANT: Power must be off before plugging the SDIOB8 into the SPIO and when any connections are made to the SDIOB8. Make sure all external devices connected are electrically compatible with SDIOB8. Failure to follow these guidelines could result in damage to the SPIO or SDIOB8 and void the warranty.**

## DIGITAL I/O - INPUTS

Digital inputs are used to sense an "ON" or "OFF" state based on voltage levels. This can be accomplished via switch closures, contact closures or the state of a digital signal. When an input of the SPIO is pulled up (above +2.0 VDC) through an external resistor, a logic "0" will be returned when read by the host. When an input of the SPIO is grounded (below +0.8 Volts), it will be read as a logic "1" by the host. For typical digital input circuits refer to Figure 2-5A, 2-6 for the SPIO, and to Figure 2-5B for the SDIOB8. These input lines can be used to sense AC voltages by using solid state relays that are available from many manufacturers.

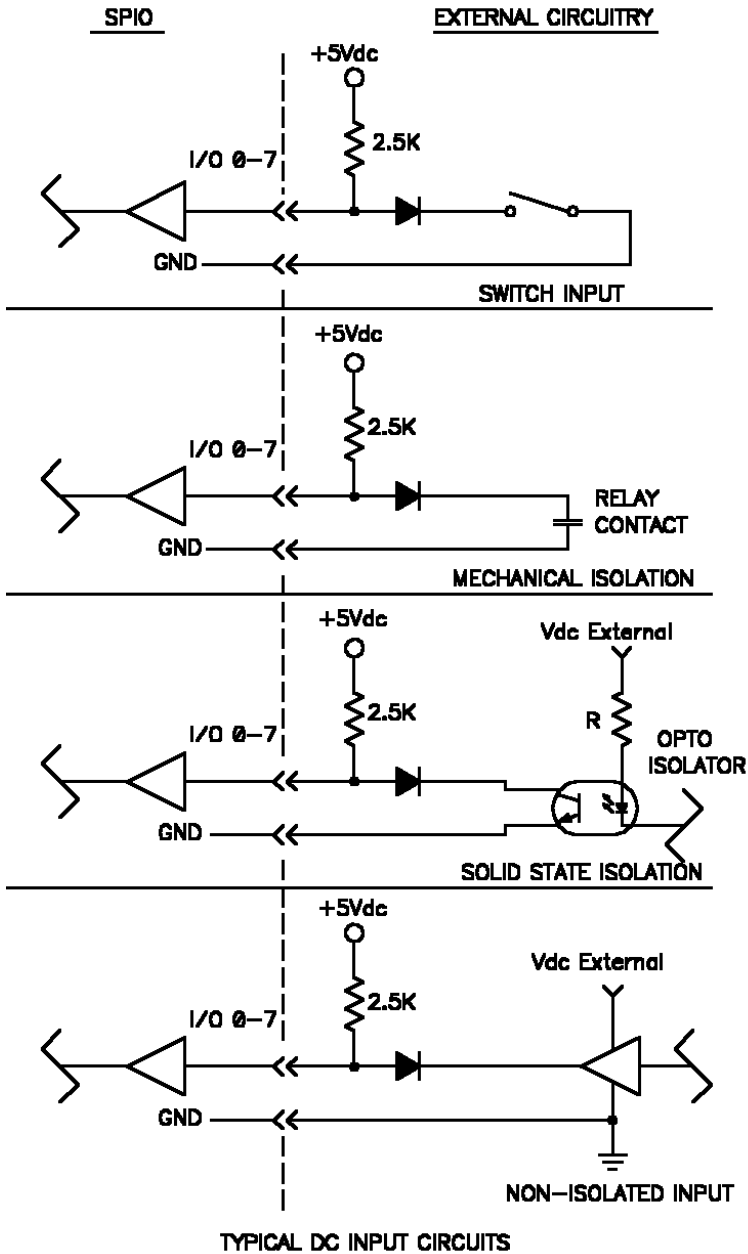


Figure 2-5A

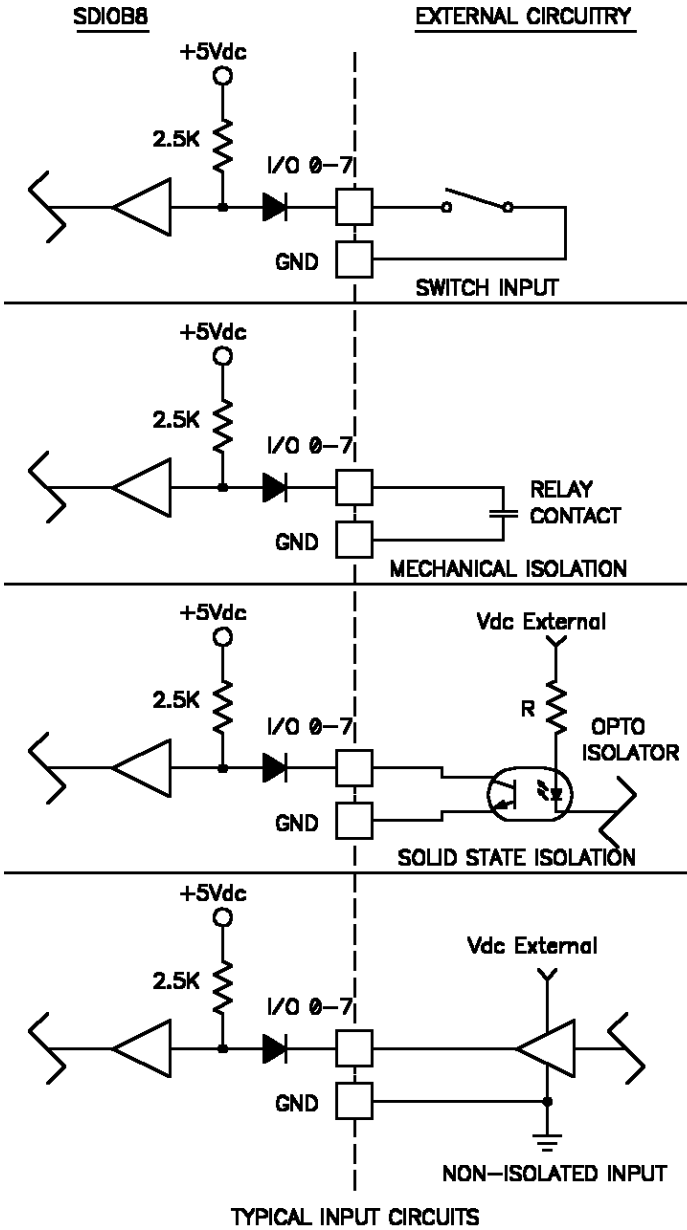


Figure 2-5B

INTERFACING SPIO WITH ISOLATED DIGITAL I/O MODULES

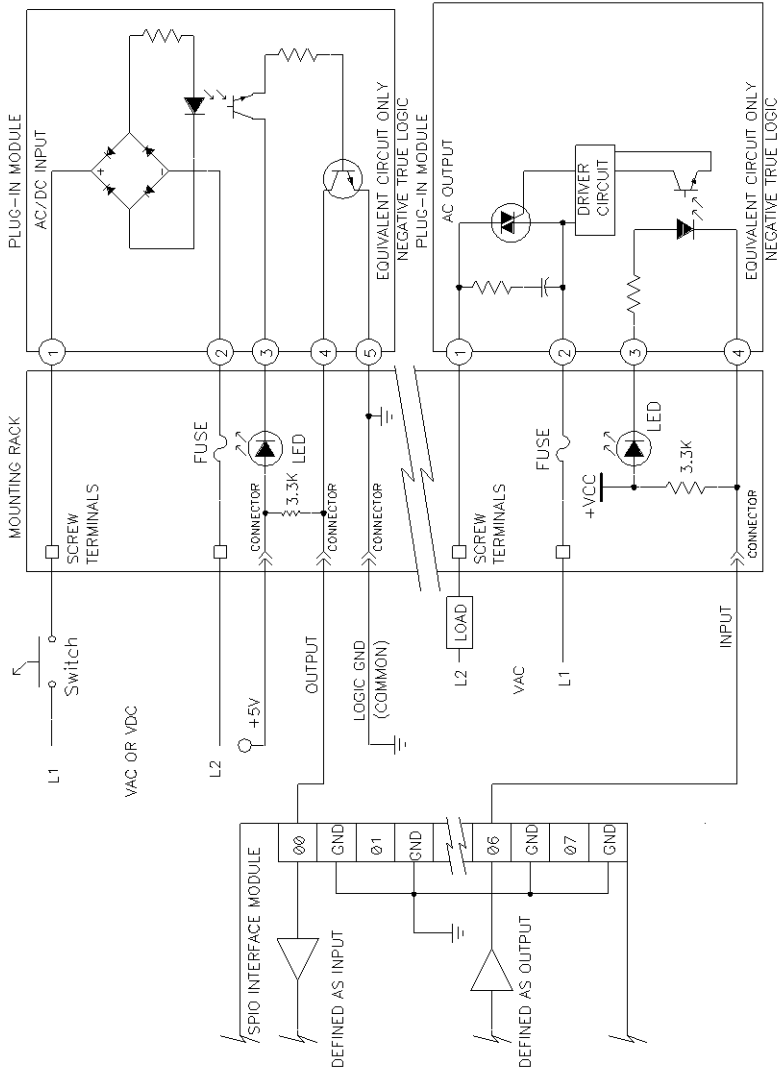


Figure 2-6

Listed below are the digital input characteristics:

Input voltage:	0 to +5.0VDC
"0" input voltage:	+2.0 to +5.0VDC
"1" input voltage:	0 to +0.8VDC

## Normal Inputs

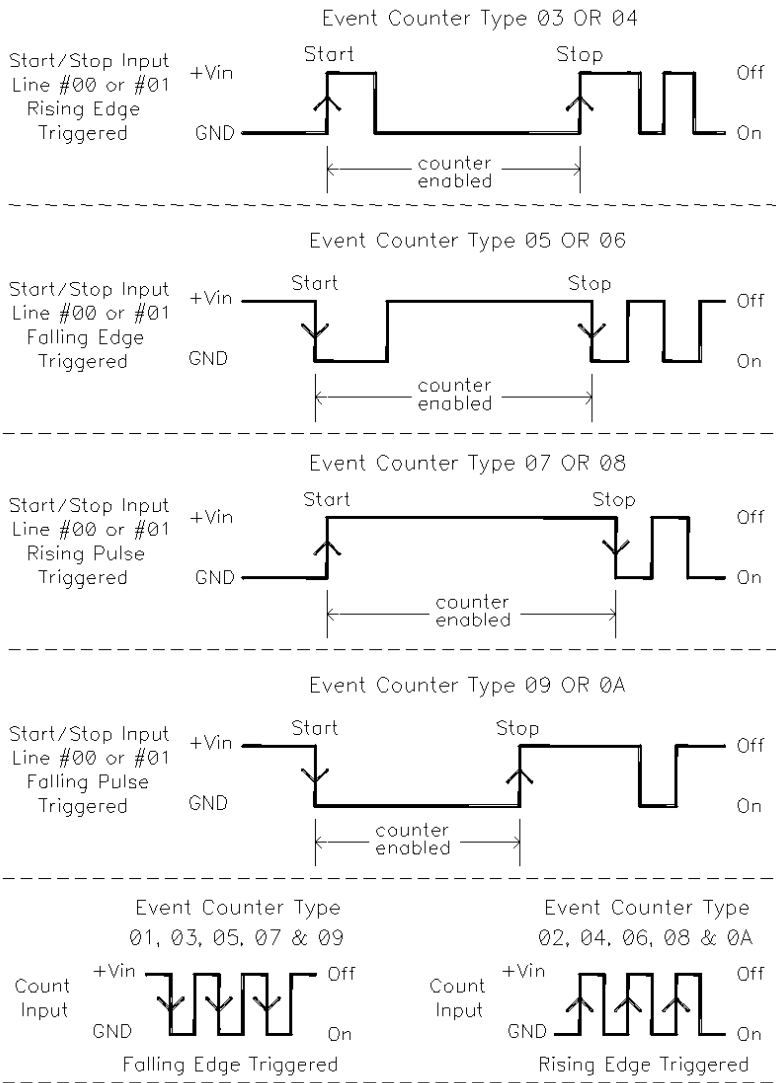
All available I/O lines can be used as normal inputs. These are a non-latched type inputs. The module is shipped with all I/O lines configured as normal inputs.

## Event Counters

Event counters can be used to record momentary events. Only two I/O lines can be defined as event counters. The starting and stopping of the counters can either be controlled from the host or by an input line. The first type uses I/O lines #00 and/or #01 as the count input with the counting being controlled via the counter stop and start commands. The second type uses I/O lines #00 & #01 as the start/stop lines with #04 & #05 as the count input lines respectively. Both the count and control lines can be configured as falling or rising edge triggered. Refer to Figure 2-7. The minimum pulse width that can be sensed is 0.65 microseconds when using I/O lines #00 & #01 as count inputs or as start/stop lines. The minimum pulse width that can be sensed is 0.93 microseconds when using I/O lines #04 & #05 as count inputs lines. The maximum count value is 16,777,215.

## Latches

All available I/O lines can be used as latched inputs. They can be defined as rising or falling edge triggered. The minimum pulse width that can be sensed is 10 milliseconds.



**Figure 2-7**

## DIGITAL I/O - OUTPUTS

Digital outputs are used by the host computer to turn "ON" or "OFF" external devices. The SPIO outputs are CMOS compatible which can be used to control solid state output modules, CMOS and TTL logic circuits. Refer to Figure 2-6, and 2-8A for typical digital output circuits for SPIO and Figure 2-8B for SDIOB8. Turning on an output, forces the output LOW. Conversely, turning off an output, forces the output HIGH.

Listed below are the digital output characteristics:

Output voltage: +5.0VDC max.

Output current: 10mA max. per output

**CAUTION:** Total digital output power cannot exceed 0.5 watts.

### Normal Outputs

I/O lines #00 - 07 can be defined as normal outputs. The host controls the immediate on or off state of these outputs.

### Time Delayed Outputs

#### Normal Outputs

I/O lines #00 - 07 can be defined as normal outputs. The host controls the immediate on or off state of these outputs.

I/O lines #00 - 07 can be defined as time delayed outputs. Each output can be defined as four types of time delays:

	OUTPUT STATE		
	Before Delay	During Delay	After Delay
a)	OFF	ON	OFF
b)	OFF	OFF	ON
c)	ON	ON	OFF
d)	ON	OFF	ON

Time delays have a resolution of 10 ms. Refer to Figure 2-9.

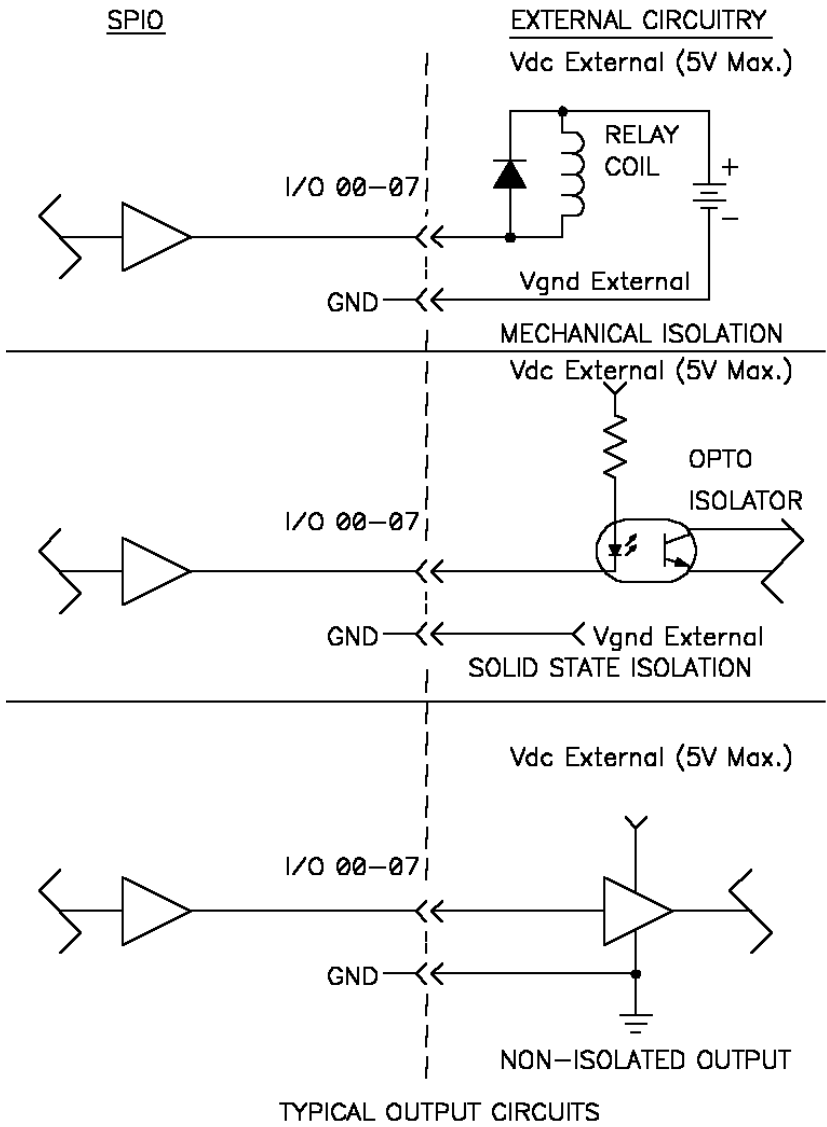


Figure 2-8A

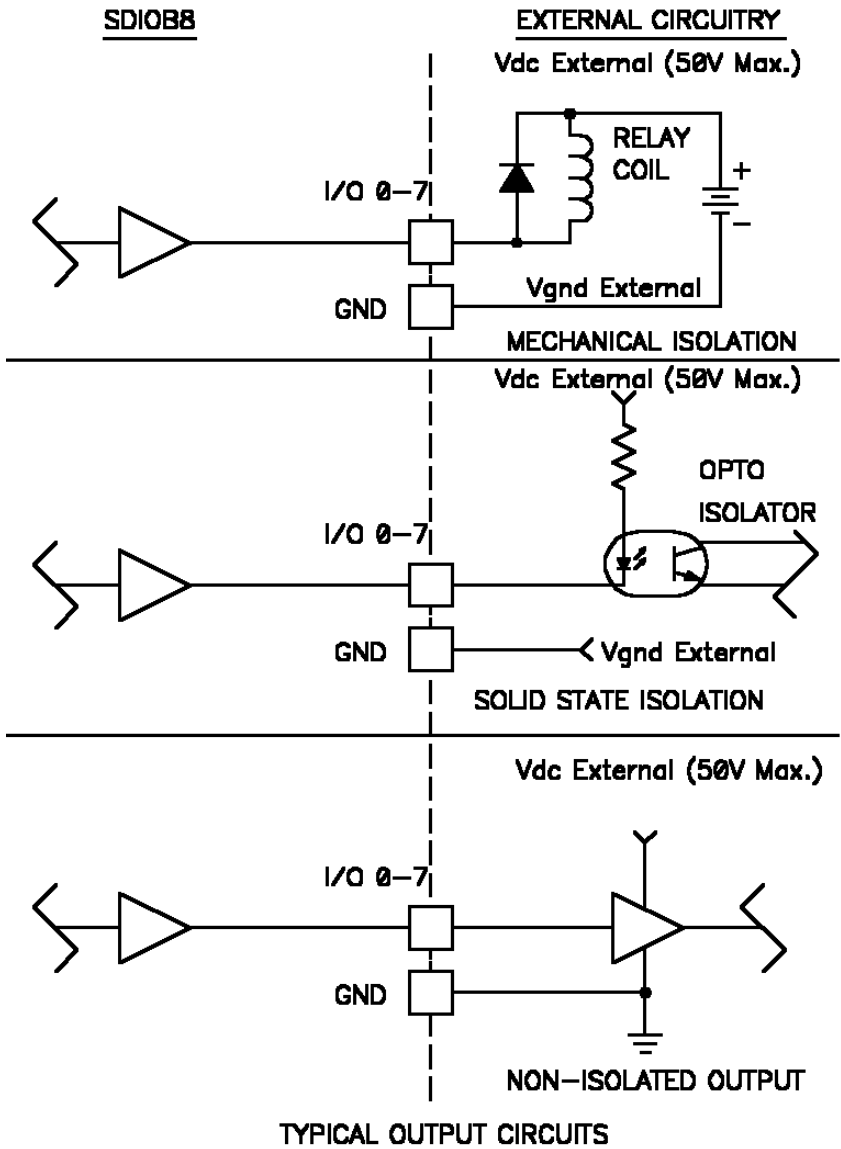
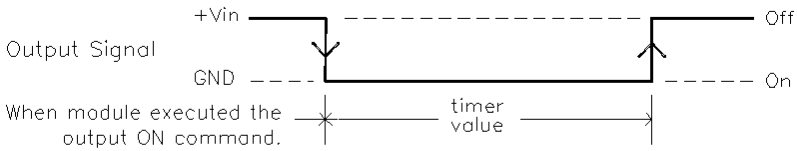
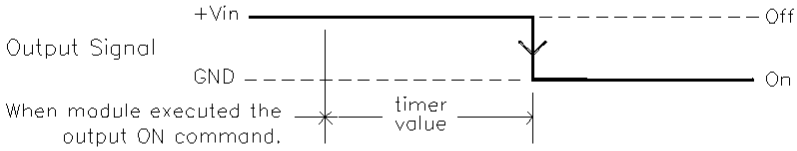


Figure 2-8B

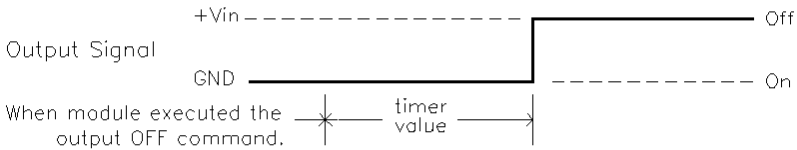
Time Delayed Output Type 81 (H)  
Turn on for designated time and then turn off.



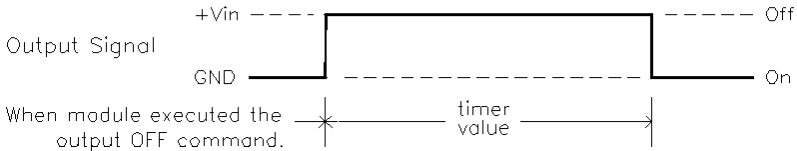
Time Delayed Output Type 82 (I)  
Stay off for designated time and then turn on.



Time Delayed Output Type 83 (J)  
Stay on for designated time and then turn off.



Time Delayed Output Type 84 (K)  
Stay off for designated time and then turn on.



**Figure 2-9**

## Pulse Width Modulation

The SPIO has a built-in Pulse Width Modulation (PWM) generator output that is available on I/O line #03 only. The PWM can be operated in two modes, high speed or high resolution.

The High Speed mode has a fixed cycle of 69 microseconds and a pulse resolution of 0.27 micro-seconds. In this mode the PWM has 8-bit resolution with 0.27 micro-seconds per bit. High Speed mode has a valid value range of 0000 - 00FF hex with 0000 being a constant "Low" output and 00FF being a constant "High" output.

The high resolution mode has a fixed cycle of 17.78 milliseconds and a pulse resolution of 0.27 microseconds. In this mode the PWM has 16-bit resolution with 0.27 microseconds per bit. High Resolution mode has a value range of 0000 - FFFF with 0000 being a constant "Low" output and FFFF being a constant "High" output.

To determine the value for a given pulse width divide the desired pulse width by 0.27 microseconds then round the result to the nearest whole number and convert it to hex.

### Examples:

High Speed mode:        34 microsecond pulse width.  
                              Data value =  
                               $34\mu s / .27\mu s = 125.9 = 126 = 7E$  hex.

High Resolution mode: 10 millisecond pulse width.  
                              Data value =  
                               $10ms / .27\mu s = 37,037.03 = 37,037 = 90AD$  hex.

The hex values are then sent to the SPIO using the "Read/Set Output Timer Value" command. To turn the PWM's output on or off use the "J", "K" or "L" command.

**NOTE:** The PWM is not available on the SDIOB8.

**IMPORTANT: Power must be off before any connections are made to the digital I/O lines. Make sure all external devices connected are electrically compatible with these lines. Failure to follow these guidelines could result in damage to the SPIO and void the warranty.**

## ANALOG SECTION

### Overview

The SPIO contains an 8-bit Analog to Digital (A/D) converter and an 8-bit Digital to Analog (D/A) converter. The D/A converter has two channels, a common ground, and operates with positive voltages. The A/D converter has eight channels, a common ground, and operates with positive voltages.

### A/D Channels

The A/D converters are successive approximation types with 8-bit resolution. They have absolute precision of +/- 3 least significant bits with a conversion speed of 26.9 microseconds (+/- 1.5 LSB typ.). To obtain the best accuracy it is recommended that a precision regulated power source (A/D PWR) and voltage reference (A/D REF) be used. Refer to Figure 2-10.

An easy way to use the A/D converters is to use a reference of +5.1VDC so that each bit will be equal to 20 millivolts ( $5.1\text{V}/255 = 20\text{mV}$ ). The round number makes it easy to make calculations. The range of actual input voltage is from the ANALOG GND terminal to the A/D REF terminal. If the A/D REF is lower than A/D PWR, the hex number read becomes "FF" when the input voltage is from A/D REF to A/D PWR. For lower A/D REF voltages the precision of the reading will increase. For instance, if you use a A/D REF of 3.00 volts the range will be from 0 to 3 volts and each bit will be equal to 11.76 millivolts instead of the 20 millivolt steps with a 5.1 volt A/D REF. You would be trading range for precision.

With the A/D REF connected to 5.1 volts and an input voltage of 3.20, the hex number obtained by the ideal A/D converter should be A0 hex. Since the converter can have an error of +/-3 LSB (Least Significant Bits) we may read a value anywhere from 9D to A3 hex.

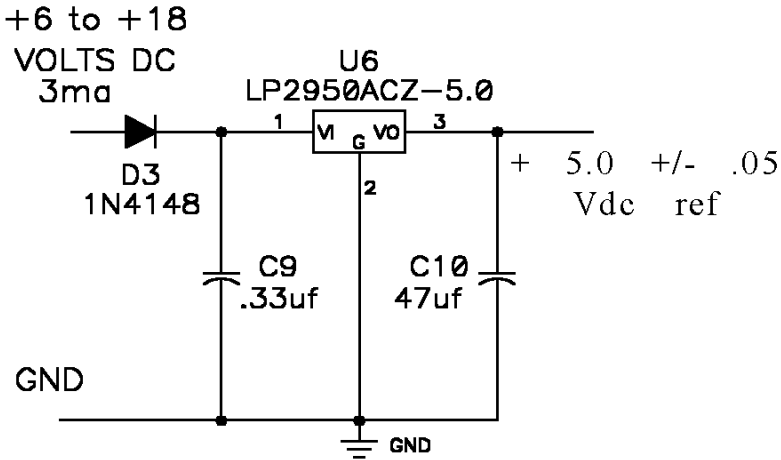
Listed below are the A/D characteristics:

Resolution:	8 bits
Absolute Accuracy:	+/- 3 LSB max. (+/- 1.5 LSB typ.)
A/D Input Voltage:	ANALOG GND to A/D PWR
A/D REF Voltage:	+2VDC min. to +5.5VDC max.
A/D PWR Voltage:	+4.5V min. to +5.5VDC max.

## A/D Thermocouple Application

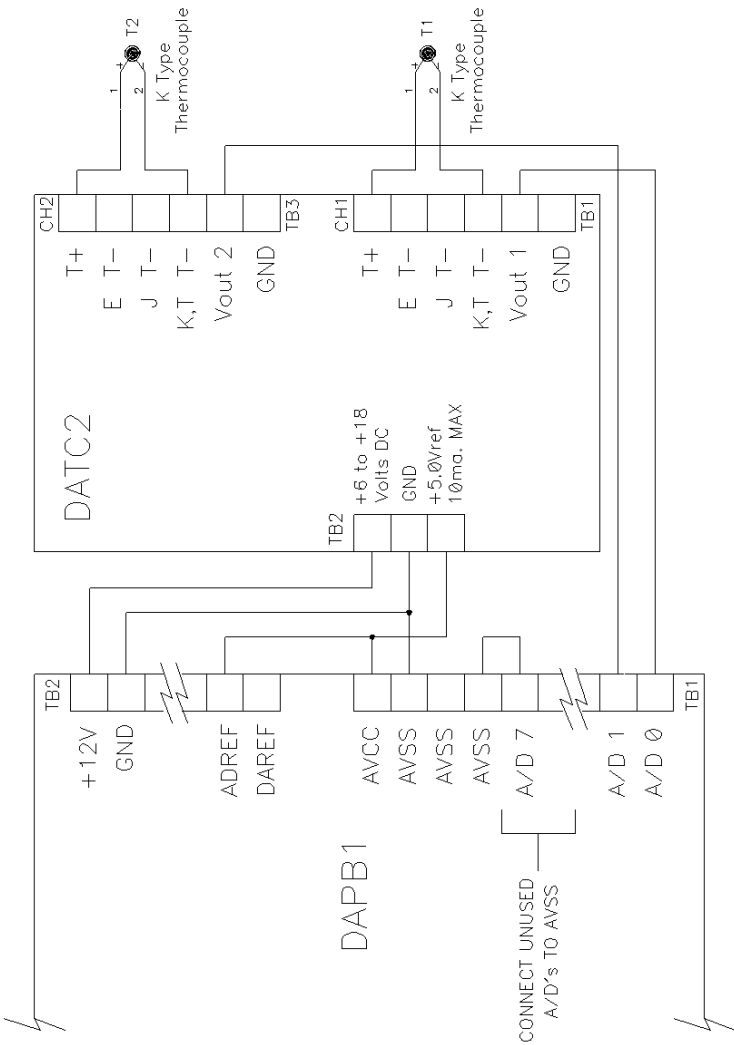
A/D channels can be used to read temperature values. A voltage from a conditioned signal from a thermocouple can be converted by the A/D to a hexadecimal value. This hexadecimal value can be converted to a temperature (Celsius or Fahrenheit).

A typical connection of two thermocouples using the DAPB1 and B&B's Model DATC2 is shown in Figure 2-11. B&B also manufactures a one channel version, Model DATC1. Both the DATC1 and the DATC2 come with demonstration programs for the SPIO. For further information refer to DATC's owner manual or call B&B Electronics.



Typical Precision Voltage Reference

Figure 2-10



CONNECTING SPIO's DAPB1 BOARD TO A DATC2  
(Disregard the CH2 connections when using a DATC1.)

Figure 2-11

## D/A Channels

The D/A converter is an R-2R resistive ladder type with 8-bit resolution. To obtain the best accuracy it is recommended that a precision regulated voltage reference (D/A REF) be used. See Figure 2-10. The range of the actual output voltage is from ANALOG GND to D/A REF. The D/A output voltage corresponds to the 8-bit hex number written to the D/A channel with respect to the D/A REF voltage. The output voltage will equal D/A REF multiplied by the hex# divided by 255. (Refer to Chapter 4 - Analog Outputs). For instance, if you connect the D/A REF to 5.00 volts and output a hex 7D (125 decimal) you will get an output of 2.45 volts.

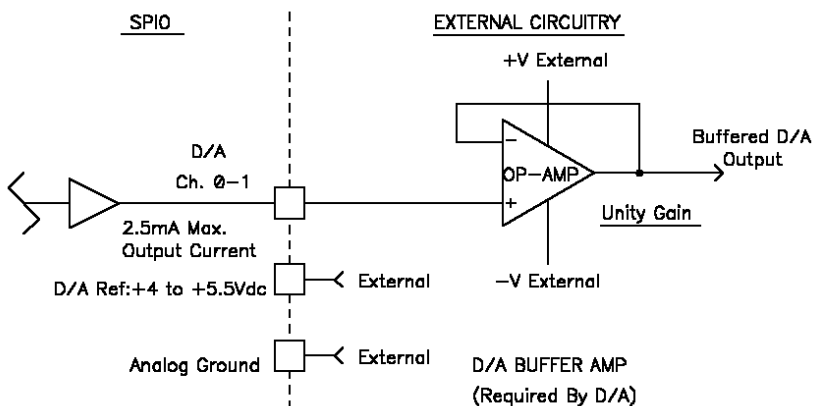
$$(5.00 * 124/255 = 2.45)$$

The D/A outputs can only drive a small amount of current so an external buffer amplifier will be required. Refer to Figure 2-12 for a suggested amplifier.

### D/A Characteristics:

Resolution:	8 bits
Output resistance:	1K min, 2K typ, 4K max
D/A REF voltage:	+4VDC min to +5.5VDC max.
REF input current:	2.5mA typ. 5ma max.
Full scale deviation:	1% max. @+5VDC D/A REF

**IMPORTANT: Power must be off before any connections are made to the analog section. Make sure all external devices connected are electrically compatible with the analog section. Failure to follow these guidelines could result in damage to the SPIO and void the warranty.**



**Figure 2-12**

## POWER SUPPLY

The SPIO module requires unregulated +12VDC at about 50 mA. This power requirement does not include power consumption of external sensors or devices/modules (such as the SDIOB8) connected to the SPIO module.

## Chapter 3: **INSTALLATION**

### OVERVIEW

This chapter will describe how to connect the power and the communication lines to the SPIO module. Reprogramming the SPIO's firmware using Update program will be covered in this chapter. It also will explain how the "SPCFG1.EXE" program is used to configure the SPIO.

### POWER CONNECTIONS

The SPIO requires unregulated +12VDC at about 50 ma. Simply plug a 2.5mm plug from an unregulated +12VDC source into the power jack on the side of the SPIO module. Power supplies are available through B&B. (Model 232PS3 is recommended)

The 50 mA which is required by the SPIO module does not include any external devices connected to the SPIO module. For example: If 100 mA is drawn from the unregulated +12VDC on the I/O port of the SPIO, the power supply must provide at least 150 mA.

**IMPORTANT:** Make sure that only 10 - 18 VDC is applied to the SPIO. Any other voltages could damage unit and void warranty.

### RS-232 CONNECTIONS

In order to communicate with the SPIO module it must be connected to the host via an RS-232 compatible communication link. (Refer to Chapter 2 - Communications.)

The SPIO module is to be connected as a DCE device. For communication the following connections must be present: Transmit Data (TD) from the host must be connected to Pin #2 on the SPIO module, Receive Data from the host must be connected to Pin #3 on the SPIO module, and Signal Ground from the host must be connected to Pin #5 on the SPIO module. See Figure 2-1.

Request to Send (RTS) and Clear to Send (CTS) connections from the host to the SPIO module are not required. (Current firmware does not support handshake lines).

Table 3.1 and 3.2 give the pin connections for a DB25P (Male) and DB9P (Male) to the SPIO's DB9S (Female) connector.

<b>Table 3.1</b>			
<b>To Computer</b>			<b>To SPIO</b>
<b>DB25S</b>			<b>DB9P</b>
<b>Pin#</b>	<b>Signal Name</b>	<b>Direction</b>	<b>Pin#</b>
2	Transmit Data	————>	3
3	Receive Data	<————	2
4	Request to Send		7*
5	Clear to Send		8*
6	Data Set Ready		NC
7	Signal Ground		5
8	Carrier Detect		NC
20	Data Terminal Ready		NC
22	Ring Indicator		NC

\*\_Not required

<b>Table 3.2</b>			
<b>To Computer</b>			<b>To SPIO</b>
<b>DB9S</b>			<b>DB9P</b>
<b>Pin#</b>	<b>Signal Name</b>	<b>Direction</b>	<b>Pin#</b>
1	Data Carrier Detect		NC
2	Receive Data	<————	2
3	Transmit Data	————>	3
4	Data Terminal Ready		NC
5	Signal Ground		5
6	Data Set Ready		NC
7	Request To Send		7*
8	Clear To Send		8*
9	Ring Indicator		NC

\* Not required

B&B Electronics offers a wide variety of cables and adapters to support various system needs.

With the use of modems, the SPIO may be used at a remote site. Contact B&B Electronics' technical support for additional information.

## **UPDATING SPIO FIRMWARE**

The SPIO uses an EEPROM to store the internal program. Because of this the SPIO's firmware can be updated very easily.

Included, on the disk that came with your SPIO, are the following files: UPDATE.EXE, SPIOV21.HEX, and a SDIB8V21.HEX file. The SPIOV21.HEX file is the current version of the SPIO firmware program in Intel Hex File Format. The SDIB8V21.HEX file is the current version of firmware program for the SDIOB8 interface module. The DAPB1 uses the same firmware as the SPIO. The UPDATE.EXE is used for updating the SPIO firmware program whenever the SPIO requires a firmware change.

### **Why Reprogram?**

You may be saying to yourself this is neat, but why? Each interface requires specific firmware, so the SPIO must be updated for each interface module change.

When B&B comes out with new features, we can send you a new .HEX file. By down loading the new file into the EEPROM you will have the latest firmware program.

Another reason you may need to reprogram is that it is possible that your EEPROM may become corrupt. If that happens you can fix it yourself. This is why we have included the current version of firmware on your SPIO disk.

You also may switch from version to version easily. For example, if for some reason there is a bug in a new version of the firmware, you could reprogram the SPIO with the old version until we fix the bug.

### **Setting Up An Update Session**

The SPIO module **MUST** have all connections to its I/O port connector unplugged to be reprogrammed. Disconnect power before removing any connections. All I/O lines on the SPIO must be disconnected from external equipment before reprogramming. The SPIO should be connected directly to an IBM compatible computer using a com port that is compatible with the SPIO. The Update program supports COM1, COM2, COM3, or COM4.

Also, it might be convenient for the SPIO to be close to the computer. You MUST enter the setup mode on the SPIO before reprogramming. Refer to Chapter 2 for complete instructions on entering the setup mode.

## **Command Line Switches**

The Update program uses command line switches to specify options.

UPDATE <COM Port> <File Name>

<COM Port> can be 1, 2, 3, or 4.

<File Name> the Intel Hex Format File Name (Without the .HEX extension)

Example 1:

UPDATE 1 SDIB8V21

This will reprogram the SPIO connected to COM1 with Version 2.1 of the firmware program which is compatible for the SDIOB8 interface module.

Example 2:

UPDATE 2 SPIOV22

This will reprogram the SPIO connected to COM2 with Version 2.2 of the firmware program which is compatible for an interface to SPIO module (DAPB1 for example).

## **What The Program Does**

The Update program uses the serial port specified on the command line to access the SPIO. Update also uses the information in the file specified on the command line to reprogram the SPIO.

When you run the program, you will be asked if the I/O port connector is unplugged and if the SETUP JUMPER is inserted. If you have prepared the module enter "Y" to continue, otherwise disconnect the power supply and perform the necessary preparations. The active light will then stop blinking and remain on until programming begins. Update will then look at the SPIO and report which version and module type of firmware the SPIO is currently running. Then, Update will check the .HEX file specified and report its version and module type to you. Update will then ask if you want to reprogram the SPIO. If you answer no, it will return you to DOS and everything in the SPIO will be returned to normal. The active light will start blinking again. If you answer yes, Update will start reprogramming the SPIO and the active LED light will blink at a fast rate until programming is complete. When update is complete it will report the current SPIO firmware version and module type. The SPIO will be returned to its normal operating mode and the active light will start blinking at a normal rate.

**IMPORTANT: Power must be off before any interface modules or devices are connected to the SPIO module. Failure to do so could result in damage to the SPIO and void the warranty.**

### **What If Something Goes Wrong?**

Once reprogramming has started this process should not be stopped for any reason. Because of this, the software was written to prevent any keyboard halting of the program. This does not prevent you from turning the power off or unplugging the serial cable. If for some reason the update program does not complete its reprogramming, the SPIO will be unusable. This will be shown by the active light remaining constantly on. If this happens, you **MUST** keep trying to reprogram the SPIO until it can finish the reprogramming. If this happens the SPIO will also return version number ???.

## CONFIGURATION PROGRAM

### Configuring your module

Your module's power-up configuration can be set with the configuration program, SPCFG1.EXE. The power-up configuration is the condition that the module returns to when one of the following occurs:

- \* The module is initially powered-up.
- \* A power failure occurs.
- \* A COMMUNICATION\_TIMEOUT error occurs.
- \* The module is reset with the "B" command (Module\_Reset).

**WARNING: Do not use the configuration program with devices connected to the module.**

The SPCFG1.EXE program will accept either one or two parameters as described below.

### Communication Port

The COM port number. This is a number between 1 and 4 for COM1 through COM4. The standard addresses and interrupt request (IRQ) numbers are shown in the table below.

Port	Address	IRQ
COM1	03F8h	4
COM2	02F8h	3
COM3	03E8h	4
COM4	02E8h	3

If you specify a number other than 1 through 4, the program assumes that you are specifying a serial port address (in hexadecimal notation). This is to support serial cards that have selectable addresses.

## Interrupt Request (IRQ) Number

If you specified a COM port number in the first parameter, this parameter is ignored and the standard interrupt request number is used.

If you specify a serial port address, then the IRQ number (in hexadecimal notation) must be given. This number tells the communication driver that the serial port has something it wants the driver to do.

Be careful when specifying the IRQ number. If you specify an IRQ number that is already used for something else, your computer may hang. A list of the IRQ numbers typically used in XT and AT type computers is shown below.

IRQ	Description
02h	Reserved (XT), INT 8 – 15 (AT)
03h	COM or SDLC
04h	COM or SDLC
05h	Hard disk drive (XT) LPT (AT)
06h	Floppy disk drive
07h	LPT
08h	Real time clock (AT only)
09h	Re-directed to IRQ2 (AT only)
0Ah	Unassigned (AT only)
0Bh	Unassigned (AT only)
0Ch	Unassigned (AT only)
0Dh	Math co-processor (AT only)
0Eh	Hard disk drive (AT only)
0Fh	Unassigned (AT only)

## Communication Errors

The communication driver used by the configuration program can report the following errors:

### The serial port is not found.

- If you specified a COM port number on the command line, then the COM port does not exist or it is using a non-

standard address. Check the address of your serial card and try specifying the serial port's address and IRQ number instead of the COM port's number.

- If you specified the serial port's address and IRQ number, check the address of the serial card and make sure that it is what you specified.

### **The serial port is not responding.**

- If you specified the COM port number on the command line, your card can be using a non-standard IRQ number. Check the IRQ number that your card is using and try specifying the serial port's address and IRQ number instead of the COM port number.
- If you specified the serial port's address and IRQ number, you probably specified a wrong IRQ number. Check the IRQ number that your serial card is using and make sure that it is what you specified.

### **Unsupported UART**

The serial card is using a UART (universal asynchronous receiver/transmitter) that the driver does not supported (8250 or 8250B). Try using another serial card.

**WARNING: If you have any devices connected to the module, do not execute the following example.**

For the rest of this section we will assume you that you have the module connected to COM1. Type **SPCFG1** at the DOS prompt and press the **ENTER** key. This will bring you to the program's title screen. It looks similar to the one below.

**F1 = Help SPIO Configuration Utility**

B&B Electronics Manufacturing Company  
P.O. Box 1040  
Ottawa, IL 61350

Voice (815) 433-5100  
Fax (815) 433-5105  
BBS (815) 434-2927

WARNING: DO NOT USE THIS PROGRAM IF YOU HAVE YOUR MODULE CONNECTED  
TO ANY DEVICES. IF IT IS, DO NOT FOLLOW THE INSTRUCTIONS  
BELOW AND PRESS ANY KEY OTHER THAN THE SPACE BAR.

NOTE: All numbers are in hexadecimal notation.

1. Remove power from the module and remove the back cover.
2. Insert the setup jumper.
3. Put the back cover on and restore power to the module.
4. Press the [ SPACE BAR ] to continue configuring the module.

Copyright © 1991, 1992 B&B Electronics Manufacturing Company

If you have any devices connected to the module, exit the program by pressing any key other than the space bar and do not continue.

Once you have inserted the setup jumper, press the **SPACE BAR** to continue configuring the module. This will bring you to an options selection screen. You will be able to select one of the following:

- Read the module's configuration.
- Reset the module to factory defaults.
- Exit the program.

Use the up and down arrow keys to highlight the option that you want, then press the **ENTER** key.

**F1 = Help**                      **SPIO Configuration Utility**

Read module's configuration
Reset module to factory
Exit the program

Copyright © 1991, 1992 B&B Electronics Manufacturing Company

If you did not exit the program, you will be on the power-up communication configuration screen. This configuration will either be the factory defaults or the module's power-up configuration, depending on which option you chose. You may change any of these values.

**F1 = Help**                      **SPIO Configuration Utility**

Baud Rate:	9600
Data Bits:	8
Protocol Passes:	Two-Pass
Turn-Around Delay:	10 ms
Communication Timer:	0000

Copyright © 1991, 1992 B&B Electronics Manufacturing Company

When you are finished specifying these values, press the **PAGE DOWN** key.

If your module has digital I/O lines, this will bring you to the digital I/O power-up configuration screen.

If your module does not have any digital I/O lines, you will be on the analog power-up configuration screen.

We will now describe the digital I/O power-up configuration screen. It displays the configuration of 32 I/O lines. Any line that is not available on your module will be marked "Not Available". It looks similar to the one below. The numbers in parenthesis is the I/O type number.

**F1 = Help**                      **SPIO Configuration Utility**

00	Normal Input (00h)	10	Not Available
01	Normal Input (00h)	11	Not Available
02	Normal Input (00h)	12	Not Available
03	Normal Input (00h)	13	Not Available
04	Normal Input (00h)	14	Not Available
05	Normal Input (00h)	15	Not Available
06	Normal Input (00h)	16	Not Available
07	Normal Input (00h)	17	Not Available
08	Not Available	18	Not Available
09	Not Available	19	Not Available
0A	Not Available	1A	Not Available
0B	Not Available	1B	Not Available
0C	Not Available	1C	Not Available
0D	Not Available	1D	Not Available
0E	Not Available	1E	Not Available
0F	Not Available	1F	Not Available

Copyright © 1991, 1992 B&B Electronics Manufacturing Company

Use the up and down arrow keys to select a line. When you have selected a line, you may change that line's I/O class with the right and left arrow keys. You can only select I/O classes that the line can be configured to. When you have selected an I/O class, press the **ENTER** key to select an I/O type within that class.

If a class contains more than one I/O type, you will see an I/O type selection list.

If the class contains only one I/O type and there is further configuration to be done on that I/O type, you will be able to specify that configuration.

If the class contains only one I/O type, and no further configuration is necessary, a message will appear to notify you of this. You may press any key to return to the digital I/O power-up configuration screen.

You use the up and down arrow keys to select the I/O type. When you have selected an I/O type, press the **ENTER** key.

If there is further configuration to be done on the selected I/O type, you will be able to specify that configuration.

If no further configuration is necessary, a message will appear to notify you of this. You may press any key to return to the digital I/O power-up configuration screen.

When you are finished with the digital I/O power-up configuration, press the **PAGE DOWN** key.

If your module has analog output channels available, you will now be on the analog power-up output configuration screen.

If your module does not have analog output channels available, you will be on the power-up communication configuration screen again.

We will now describe the analog power-up output configuration screen. It looks similar to the one following.

0. 0000	16. Not Available
1. 0000	17. Not Available
2. Not Available	18. Not Available
3. Not Available	19. Not Available
4. Not Available	20. Not Available
5. Not Available	21. Not Available
6. Not Available	22. Not Available
7. Not Available	23. Not Available
8. Not Available	24. Not Available
9. Not Available	25. Not Available
10. Not Available	26. Not Available
11. Not Available	27. Not Available
12. Not Available	28. Not Available
13. Not Available	29. Not Available
14. Not Available	30. Not Available
15. Not Available	31. Not Available

Copyright © 1991, 1992 B&B Electronics Manufacturing Company

You can now specify the analog power-up output levels for the available lines. When you are done, press the **PAGE DOWN** key. This will bring you the the power-up communication configuration screen again.

When you have returned to the power-up communication configuration screen, you may exit the program. You do this by pressing the **ESC** key.

You will then be asked if you want to write the configuration to the module.

If you decide to write the configuration, the module's configuration will be changed before exiting the program.

If you do not write the configuration, the module will retain the configuration that it had before running the SPCFG1.EXE program.

You will need to remove the setup jumper when you leave the program.

## Installing the software

- \* Put the disk that came with your module into your floppy disk drive.
- \* Make the floppy drive the default drive. If your disk is in drive A, then this can be done by typing **A:** at the DOS prompt and pressing the **ENTER** key.
- \* First, read the file READ.ME by typing **TYPE READ.ME** at the DOS prompt and pressing **ENTER** key.
- \* The install program requires three arguments and will accept up to five arguments. The first is the drive that you want to install the files on. The second is the directory that you want to put the files in. The third through fifth arguments are the programming languages that you will be using. These can be "BASIC", "C", and/or "PASCAL".
- \* If you want to install the files on drive C, in the directory C:\QBASIC\API, for the QuickBASIC programming language, type **INSTALL C: \QBASIC\API BASIC** at the DOS prompt and press the **ENTER** key.
- \* If you want to install the files on drive C, in the directory C:\BC\API, for the C programming language, type **INSTALL C: \BC\API C** at the DOS prompt and press the **ENTER** key.
- \* If you want to install the files on drive C, in the directory C:\TP\API, for the Pascal programming language, type **INSTALL C: \TP\API PASCAL** at the DOS prompt and press the **ENTER** key.
- \* When the installation is complete, remove the disk from the floppy drive and store it in a safe place.

The following lists the files that will be installed on your drive.

<b>C</b>	<b>Pascal</b>	<b>QuickBASIC</b>
UPDATE.EXE	UPDATE.EXE	UPDATE.EXE
UPDATE.DAT	UPDATE.DAT	UPDATE.DAT
SPIOV21.HEX	SPIOV21.HEX	SPIOV21.HEX
SDIB8V21.HEX	SDIB8V21.HEX	SDIB8V21.HEX
SPCFG1.EXE	SPCFG1.EXE	SPCFG1.EXE
SPCFG1.DAT	SPCFG1.DAT	SPCFG1.DAT
SPMON1.EXE	SPMON1.EXE	SPMON1.EXE
SPAPI1.OBJ	SPAPI1.OBJ	SPAPI1.OBJ
SPDEM1BC.EXE	SPDEM1TP.EXE	SPDEM1QB.EXE
SPDEM1BC.C	SPDEM1TP.PAS	SPDEM1QB.BAS
SPINC1BC.H	SPINC1TP.PAS	SPINC1QB.BI
.....	SPINC1TP.TPU	.....
.....	.....	SPQLB1QB.LIB
.....	.....	SPQBL1QB.QBL

# Chapter 4: **COMMANDS**

## Overview

This section discusses the architecture of an SPIO command message and each SPIO command in detail.

Command messages consist of three parts. The first part contains the start of message character and two characters representing the preamble of the command type (DIGITAL or ANALOG). The second part must contain the command character followed by, if required, the command parameters. The third part contains three characters, the first two characters represent the command checksum and the third indicates the end of the command message.

Reply messages are divided into 3 types: an acknowledgement response, an acknowledgement with data response, or an error response.

**NOTE:** All command messages and replies are transmitted as a series of ASCII characters. The hexadecimal numbering system is used to represent all numbers in commands and responses. These ASCII numbers are 0 - 9 and upper-case letters A - F. This representation of numbers with ASCII characters will be referred to as "ASCII-HEX" (Refer to Appendix B). A command function will be represented by a printable ASCII character (Refer to Appendix A).

## Command Message

The first three bytes received by the SPIO module must be the following characters for a digital command ">00". The first three bytes for an analog command must be ">FF". The fourth byte must contain the command character. The following bytes, if required by the command, must contain the command parameters. The following two bytes represent the checksum. The last byte is the end of message character and can be either a carriage return (cr) or a period (.). In command examples a carriage return will be represented as "cr".

Example:

<b>Part1</b>	<b>Part2</b>	<b>Part3</b>
DIGITAL >00 [cmd]	{command parameters}	[cksm]cr
ANALOG >FF [cmd]	{command parameters}	[cksm]cr

## **Reply Messages**

### *Acknowledgement -*

The first byte will be an "A" followed by the carriage return character.

Example: Acr

### *Acknowledgement with data-*

The first byte will be an "A" followed by the data, two ASCII-HEX characters representing the checksum and a carriage return. The data component of the message can be of variable length and this length is dependent on the command received. The data is always returned with the most significant data first in sequence with the least significant data being last.

Example: A[data][cksm]cr

### *Error message -*

The first character will be an "N". The second and third characters will be a two digit error code. The fourth character will be a carriage return character.

Example: N[error code]cr

## **Checksum**

The checksum value is calculated by summing the decimal value of all the ASCII characters of the message excluding the start of message character, the two checksum characters, and the end of message character. The sum is then divided by 256. The remainder is then converted to hexadecimal and becomes the checksum.

Example:

Command to be sent without checksum =

```
>00 %105 [cksm]cr
```

$$48+48+37+49+48+53 = 283$$

283/256 = 1 with a remainder of 27 or 1B hex  
Checksum = 1B

Command to be sent with checksum =

```
>00%1051Bcr
```

The checksum may be ignored by placing two “?” characters in the checksum field. This should only be used for debugging purposes and SHOULD NOT BE USED for the final application.

## Two-Step & Four-Step Communications

The SPIO modules can be configured to use one of two types of communication modes, two-step or four-step. The two-step mode allows for efficient communication between the host and the module while maintaining message integrity due to the message checksum calculation. The four-step mode adds an additional level of security when message integrity is of the utmost importance. These modes are described below.

### Two-Step Communications

In two-step mode, the host transmits a command message to an SPIO module, the module will execute the command and then send to the host the acknowledgement. However, if the command received by the module was in error, the error message will be sent to the host and the command will be ignored.

EXAMPLE:

Step 1) Command sent by host.

```
>00AA1cr
```

Step 2) Response from module upon execution.

Acr

### Four-Step Communications

In four-step mode, the host transmits a command message to an SPIO module. If there are no communication errors, the module echos the message back to the host for verification. The echo message will contain the same characters as the message received except for the start of message character ">" that will be replaced with an "A" and the end of message character "cr" or "." will always be a "cr". If an error occurs during this sequence the module will send an error message and the command will be ignored. If there are no errors the host must then send the execute command to the module.

The execute command syntax is:

Ecr

The module will then execute the command and send the appropriate acknowledgement to the host.

EXAMPLE:

Step 1) Command sent by host.

>00AA1cr

Step 2) Echo command from SPIO to host.

A00AA1cr

Step 3) Execute command sent by host.

Ecr

Step 4) Response from SPIO upon execution.

Acr

**NOTE:** If the decision is made not to execute the command simply send the next command and the previous command will be aborted.

## Input/Output Section

When constructing command messages that contain an input/output field it is necessary to understand how to select the desired input/output line/s. The input/output field will use the following terms: I/O, input, output or channel. When the terms "I/O", "input" or "output" are contained in the input/output field, the lines affected will be the digital lines. When the term "channel" is contained in the input/output field, the lines affected will be the analog lines. A single ASCII-HEX digit can represent a 4-bit binary number (0000 -1111), therefore each ASCII-HEX digit represents 4 input/output lines. The SPIO command set can accept up to 4 ASCII-HEX digits in the input/output field, which means that a command could handle up to 16 lines of digital, 16 lines of D-A (digital to analog) or 16 lines of A-D (analog to digital). The following table should help to explain how to select or read input/output lines.

Input/Output Lines:

	<b>Most Signf Digit</b>				<b>Digit</b>				<b>Digit</b>				<b>Least Signf Digit</b>			
<b>Line#</b>	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<b>Bit Pos.</b>	3	2	1	0	3	2	1	0	3	2	1	0	3	2	1	0
<b>Weight</b>	8	4	2	1	8	4	2	1	8	4	2	1	8	4	2	1

To select the desired input/output line, set the corresponding bit position to a "1". To convert this to an ASCII-HEX digit add the weight value of this bit to any of the other bit positions selected in that digit.

Example:

Select input/output lines 15 , 13, 10 & 08 = A500 hex.

Select input/output lines 05, 04, 03 & 02 = 3C hex.

**NOTE:** Some commands may not require all 4 ASCII-HEX digits in the input/output field, but only those digits whose lines are affected. However, these digits must be followed by any least significant ASCII-HEX digits in the field. (Refer to each command for the specific requirements.)

## Error Codes

There are two basic types of errors: a command message error and an SPIO generated error. A command message error occurs when: the SPIO received a command character that it did not recognize, an incorrect checksum was received, the command received was too long or too short, a non-printable ASCII character was received or an invalid value was received in one of the data fields. An SPIO error occurs when: power is first applied (which can signify a power failure), when the proper interface module is not connected to the SPIO module for the current SPIO firmware, or when communications has not been received by the SPIO within a pre-defined time. When an SPIO error occurs all outputs will be reset to power up levels.

<b>Hex Value</b>	<b>Description</b>
00H	Power-up Clear Expected. First time power-up or a power failure occurred.
01H	Undefined Command. Unrecognizable command character.
02H	Checksum Error. Calculated checksum does not match the received checksum.
03H	Input Buffer Overrun. Message had too many characters.
04H	Non-printable ASCII Character Received. Legal ASCII characters are 21-7F hex.
05H	Data Field Error. Not enough characters received.
06H	Comm Time-Out Error. SPIO did not receive communications within a defined time period.
07H	Invalid Specified Value. Values are out of SPIO's specified range.
10H	Module Interface error. Improper interface connection for current SPIO firmware.

**NOTE:** This error may not be available with all module interfaces! (DAPB1 for example).

## Command Summary List

<u>COMMAND NAME</u>	<u>COMMAND SYNTAX</u>
READ/SET SPIO CONFIGURATION	%[mod]{comm}
POWER UP CLEAR	A
RESET	B
TURN-AROUND DELAY	C{data}
COMMUNICATION TIMER	D{time}
SET COMMUNICATION MODE	E{data}
IDENTIFY MODULE	F
READ/SET DIGITAL I/O TYPE	![mod]{I/O}{type}
READ/SET DIGITAL OUTPUT POWER UP STATES	#[mod]{output}{state}
READ/SET OUTPUT TIMER VALUES	*[mod][output]{data}
CONFIGURE DIGITAL I/O LINES	G{I/O}
CONFIGURE DIGITAL I/O AS INPUTS	H{I/O}
CONFIGURE DIGITAL I/O AS OUTPUTS	I{I/O}
SET DIGITAL OUTPUTS ON OR OFF	J{output}
SET ANALOG OUTPUT LEVEL	J[channel][level]
TURN DIGITAL OUTPUT ON	K{output}
READ ANALOG OUTPUT LEVEL	K{channel}
TURN DIGITAL OUTPUT OFF	L{output}
READ ANALOG INPUT LEVEL	L{channel}
READ DIGITAL I/O STATES	M
SET LATCH EDGES	N{input}
SET OFF-TO-ON LATCHES	O{input}
SET ON-TO-OFF LATCHES	P{input}
READ LATCHES	Q
READ AND CLEAR LATCHES	R{input}

CLEAR LATCHES	S{input}
SET INDIVIDUAL ANALOG OUTPUT LEVELS	S[channel][level]
START/STOP COUNTERS	T{input}
START COUNTERS	U{input}
STOP COUNTERS	V{input}
READ COUNTERS	W{input}
READ AND CLEAR COUNTERS	X{input}
CLEAR COUNTERS	Y{input}
SET TIME DELAY	Z[output][mod]{data}
RETRIGGER TIME DELAY	h{output}
READ DIGITAL I/O CONFIGURATION	j
READ/SET ANALOG OUTPUT POWER UP LEVELS	&[mod]{channel}{level}

All commands are printable ASCII characters. (See Appendix A.)

[ ] = parameter that is required by the command.

{ } = parameter is optional or maybe required by the SET portion of the command.

For more information and examples refer to each individual command.

## Command Format

Listed below is the structure that will be used to describe each SPIO command.

- PURPOSE:** Describes the command.
- FIELD:** Describes in detail each parameter that is required by the command.
- FORMAT:** Shows the correct syntax for PART 2 of the command message. Fields surrounded by “[ ]” must contain the appropriate number of ASCII-HEX digits. Fields surrounded by “{ }” are optional and the number of ASCII- HEX digits may vary within a specified range.
- EXAMPLE:** Displays a command message sent by the host and the reply message sent by the SPIO module.

**NOTE:** These examples will be shown in the Two-Step communications mode.

## GENERAL COMMANDS

### Read/Set SPIO Configuration

This command reads or sets a SPIO module's current configuration. The "set" portion of this command can be executed only when the unit is in "SETUP MODE."

The following SPIO's parameters can be read in following order: Module Type, Communication Format, Turn-around Delay Value, and Communication Timer Value. The communication timer value consists of 4 ASCII-HEX digits, all of the other parameters consist of 2 ASCII-HEX digits.

**NOTE:** To convert the delay and timer values to real time, convert the ASCII-HEX value to decimal and multiply by ten milliseconds.

The Communication Format can be set by the user.

[modifier] must contain 1 ASCII-HEX digit; 0 = read configuration, 1 = set configuration.

[comm] must contain 2 ASCII-HEX digits. Valid settings are:

1st Digit Weight Value	2nd Digit Hex Value	Description
	0	300 Baud
	1	600 Baud
	2	1,200 Baud
	3	2,400 Baud
	4	4,800 Baud
	5	9,600 Baud
	0	8 Data Bits
	1	7 Data Bits
0		Parity Disabled
2		Parity Enabled
0		Even Parity
4		Odd Parity
0		Two-Step Comm
8		Four-Step Comm

Command Read Format:  
>00%0[cksm]cr

Command Set Format:  
>00%1[comm][cksm]cr

Example:  
Host - >00%1051Bcr  
SPIO - Acr

Set the configuration of SPIO module as follows: communications to 9,600 baud, 8 data bits, no parity, and Two-Step mode.

Example:  
Host - >00%0B5cr  
SPIO - A0205000000E7cr

Read the configuration of SPIO module. Configuration is as follows: module type is SDIOB8 (02); communications is set for 9,600 baud, 8 data bits, no parity, and Two-Step mode; turn-around delay is disabled (00); communications timer is disabled (0000).

Refer to the Turn-Around Delay and Communication Timer commands to set those parameters.

## **Power-Up Clear**

This command's only function is to reset the power-up clear error that occurs when the SPIO is first powered-up. This error will be reset upon receiving any command message from the host, however, the command received by the SPIO will be ignored and the SPIO will reply with the power up clear error. During normal operation this error indicates that the SPIO has experienced a power failure.

**NOTE:** Anytime the SPIO has a loss of power and power is reapplied to it, all outputs will be reset to their predefined level. To define these levels refer to the power up configuration commands.

Command format:

>00A[cksm]cr

Example:

Host - >00AA1cr

SPIO - Acr

Clear the power up clear error on the module after applying power to the module.

## Reset

This command resets the SPIO to predefined power up levels.

Command format:

>00B[cksm]cr

Example:

Host - >00BA2cr

SPIO - Acr

Reset module to its predefined power up state.

**NOTE:** To define the power up levels refer to the power up configuration commands.

## Turn-Around Delay

This command sets a specified time period before the SPIO transmits the reply message to the host. You must be in "SETUP MODE" to execute this command. To determine the current time delay use the Read/Set SPIO Configuration command.

{data} contains a single ASCII digit 0, 1, 2, 3 or 4.

0 = Delay Disabled

1 = 10ms

2 = 100ms

3 = 500ms

4 = 1000ms

If no digit is sent delay is disabled.

Command Format:

>00C{data}[cksm]cr

Example:

```
Host - >00C3D6cr  
SPIO - Acr
```

Set SPIO turn-around delay to 500 milliseconds.

## Communication Timer

When the communication timer is enabled the SPIO will monitor activity on the serial communication line and will reset all outputs to their power up levels if a command message has not been received within the time specified by this command. This time period is equal to [time] multiplied by 10 milliseconds. You must be in "SETUP MODE" to execute this command.

To determine the current status of the communication timer refer to the Read/Set SPIO Configuration command.

{time} can consist of 0 - 4 ASCII digits.

Valid time range is 200ms(14H) to 10.92 minutes (FFFFH). Other valid digits are:

0 or 4	=	timer disabled
1 or 5	=	10 seconds
2 or 6	=	1 minute
3 or 7	=	10 minutes.

If no digit is sent in the time field the timer will be disabled.

Command format:

```
>00D{time}[cksm]cr
```

Example:

```
Host - >00D2D6cr  
SPIO - Acr
```

Set SPIO's communication timer to 1 minute.

## Set Communications Mode

Sets the SPIO's communication mode to either Two-Step or Four-Step (Refer to the beginning of Chapter 4). You must be in "SETUP MODE" to execute this command. To determine the current mode use the Read/Set SPIO Configuration command.

[data] field must contain 1 ASCII-HEX digit.

0 = Two-Step Mode  
1 = Four-Step Mode

Command format:

>00E[data][cksm]cr

The new mode will be in affected after the reply message is sent to host from the SPIO.

Example:

Host - >00E1D6cr  
SPIO - Acr

Set SPIO to the Four-Step communication mode.

## Identify Module

This command allows the host to determine what type of interface can be used with the SPIO's current firmware.

### Interface Type

01 = SPIO (DAPB1)  
02 = SDIOB8

Command format:

>00F[cksm]cr

Example:

Host - >00FA6cr  
SPIO - A0161cr

Module identified its current firmware to be for the SPIO module.

## DIGITAL COMMANDS

This section deals with all of the commands required to configure, read and/or set the digital input and output lines.

### Read/Set Digital I/O Type

This command sets or reads the digital I/O line type. The "set" portion of this command can be executed only when the unit is in "SETUP MODE." Input types are: normal unlatched input, latched input, and event counter. Latches and counter types are also selected based on event polarity. Output types are: normal on/off, time delayed (4 types), and pulse width modulation (PWM). Time delayed outputs have a 10 millisecond resolution with a maximum time delay of 10.92 minutes. PWM output can be set in high speed or high resolution mode.

**Note:** PWM is not available when the SDIO8 is used with the SPIO.

When reading the digital I/O types, two characters will be returned for each of the 16 possible positions. The most significant I/O will be sent first in sequence with the least significant I/O last. The I/O type will consist of 2 ASCII-HEX digits. For I/O lines that are not available on the SPIO "??" will be sent in those positions.

[modifier] field must contain 1 ASCII-HEX digit; 0 = read I/O type, 1 = set I/O type.

[type] must contain 2 ASCII-HEX digits. Valid types are:

### INPUTS

Type	Description
00	Normal non-latched input.
01	Event counter with start/stop command controlled* - count input = falling edge triggered. Refer to Figure 2-7.
02	Event counter with start/stop command controlled* - count input = rising edge triggered. Refer to Figure 2-7.
03	Event counter with start/stop input** - start/stop input = rising edge controlled; count input = falling edge triggered. Refer to Figure 2-7.

- 04 Event counter with start/stop input\*\* - start/stop input = rising edge controlled; count input = rising edge triggered. Refer to Figure 2-7.
- 05 Event counter with start/stop input\*\* - start/stop input = falling edge controlled; count input = falling edge triggered. Refer to Figure 2-7.
- 06 Event counter with start/stop input\*\* - start/stop input = falling edge controlled; count input = rising edge triggered. Refer to Figure 2-7.
- 07 Event counter with start/stop input\*\* - start/stop input = rising pulse width controlled; count input = falling edge triggered. Refer to Figure 2-7.
- 08 Event counter with start/stop input\*\* - start/stop input = rising pulse width controlled; count input = rising edge triggered. Refer to Figure 2-7.
- 09 Event counter with start/stop input\*\* - start/stop input = falling pulse width controlled; count input = falling edge triggered. Refer to Figure 2-7.
- 0A Event counter with start/stop input\*\* - start/stop input = falling pulse width controlled; count input = rising edge triggered. Refer to Figure 2-7.
- 0B Latched input - rising edge triggered (on-to-off).
- 0C Latched input - falling edge triggered (off-to-on).

### READ ONLY TYPES

- 7E Start/Stop event counter input line - rising edge triggered. Automatically set when types 03, 04, 07 & 08 are set.
- 7F Start/Stop event counter input line - falling edge triggered. Automatically set when types 05, 06, 09 & 0A are set.

\* Only I/O lines #00 & #01 can be defined as this type of event counter.

\*\* Only I/O lines #04 & #05 can be defined as this type of event counter. When I/O #04 is defined as the count input, I/O #00 is automatically defined as its start/stop input. When I/O #05 is defined as the count input, I/O #01 is automatically defined as its start/stop input. Once this counter type is set I/O lines #0 & #1 cannot be reconfigured until its respective count source line is reconfigured.

## OUTPUTS

Type	Description
80	Normal on/off.
81	Time delayed - transition from OFF to ON, turn ON for desired time, then turn OFF. Refer to Figure 2-8.
82	Time delayed - transition from OFF to ON, stay OFF for desired time, then turn ON. Refer to Figure 2-8.
83	Time delayed - transition from ON to OFF, turn OFF for desired time, then turn ON. Refer to Figure 2-8.
84	Time delayed - transition from ON to OFF, stay ON for desired time, then turn OFF. Refer to Figure 2-8.
85	PWM - high resolution mode (I/O #3 only).
86	PWM - high speed mode (I/O #3 only.)

The digital output on and off commands (“J”, “K” & “L”) activate and deactivate time delay and PWM outputs. When a time delay or a PWM is configured, a default value of “FFFF” is loaded into the timer value. To change the timer value use the “\*” or the “Z” command.

[I/O] must contain 4 ASCII-HEX digits, which represents the 16 possible digital I/O lines. To change a digital I/O line to a new type, set its bit position to a “1” (refer to Chapter 4, Input/Output Selection). I/O lines with their bit position set to a “0” will retain their current type. Changing output types will turn the output OFF. All specified I/O lines will be set to the new type provided the following rules are met:

- 1) Input types CANNOT be changed to output types and vice versa.
- 2) Input type (00) can be changed to any input type except for “Read Only Types” (refer to type listing).
- 3) Output type (80) can be changed to any output type.
- 4) Like types can be changed among themselves. Like types are: 01 & 02; 03 - 0A; 0B & 0C; 81 - 84; 85 & 86.

To change an I/O to an unlike type it must be reconfigured using the “G”, “H” or “I” command before using this command. If these rules are not followed an invalid value error will be sent and the command will be ignored.

Command Read Format:  
>00!0[cksm]cr

Command Set Format:  
>00!1[I/O][type][cksm]cr

Example:  
Host - >00!1000101D4cr  
SPIO - Acr

Set SPIO module, to input type 01 event counter. All other I/O's will be left unchanged.

Example:  
Host - >00!0B1cr  
SPIO- A????????????????000B00008500810118cr

Read SPIO module digital I/O configuration types. The reply indicates I/O's: #00 is a type 01 event counter input; #01 is type 81 time delayed output; #02,04,05, & 07 are type 00 normal inputs; #03 is a type 85 PWM output; #06 is a type 0B latch input; #08-15 are not available on the module.

### Read/Set Digital Power Up States

This command can read or set the power up state for a specified digital output/s. The "set" portion of this command can be executed only when the unit is in "SETUP MODE." When reading the power up configuration the data returned will consist of two fields with both fields containing 4 ASCII-HEX digits. The first field will contain the pre-defined I/O configuration; a bit position set to a "1" represents an output, a bit position set to a "0" represents an input. The next field will contain the current output power up state for the digital outputs. If the I/O bit position is a "1" it represents an ON condition at power up. If the bit position is a "0" it represents an OFF condition at power up. If the I/O is defined as an input the corresponding power up bit position will be a "0". In both fields, the most significant I/O will be sent first in sequence with the least significant I/O last.

**NOTE:** The time cycle of time delayed outputs will not be executed at power up.

[modifier] field must contain 1 ASCII-HEX digit; 0 = read configuration, 1 = set configuration.

[output] must contain 4 ASCII-HEX digits, which represents the 16 possible digital output positions. To specify a digital output position set its bit position to a "1" (refer to Chapter 4, Input/Output Selection). All specified outputs will be set to STATE at power up. The unspecified outputs will retain their previously set power up state.

[state] must contain 1 ASCII-HEX digit. 0 = Off & 1 = On.

**NOTE:** Outputs must be defined first before the power up states can be defined. If I/O's defined as inputs are specified they will be ignored. If I/O #03 has been defined as a PWM output it will be ignored. On modules that do not have all 16 digital I/O positions available the unavailable I/O's bits will be ignored.

Command Read Format:

```
>00#0[cksm]cr
```

Command Set Format:

```
>00#1[output][state][cksm]cr
```

Example:

```
Host - >00#100501AAcr  
SPIO - Acr
```

Set SPIO module digital outputs #06 & 04 power up states to ON and all other outputs will be left unchanged.

Example:

```
Host - >00#0B3cr
```

```
SPIO - A00F000509Bcr
```

Read SPIO module power up states for the digital outputs. The reply shows that I/O lines #04, 05, 06 & 07 are defined as outputs with outputs #05 & 07 having an OFF power up state and outputs #04 & 06 having an ON power up state.

## Read/Set Output Timer Values

This command reads or sets digital output timer values. When reading output timer values the data returned will contain a field of 5 ASCII characters for each I/O position requested. If the I/O position requested does not exist on module or is an input "?????" will be returned. If the I/O position requested is normal output "00000" will be returned. However, if the I/O position is a timer output or a PWM output the first character indicates the state of the output. A "0" indicates the timer is inactive or the PWM is disabled. A "1" indicates the timer is active or the PWM is enabled. The remaining four ASCII-HEX characters will be the time delay value for timers or the pulse width value for the PWM. The most significant I/O specified will be sent first in sequence with the least significant I/O last. When setting values while the unit is in "SETUP MODE," the values are stored in EEPROM. When setting values while not in "SETUP MODE," the values will be stored in RAM. Values that are stored in RAM will be lost when the unit is reset.

[modifier] field must contain 1 ASCII-HEX digit; 0 = read timer value, 1 = set timer value.

[output] must contain 4 ASCII-HEX digits, which represents the 16 possible digital output positions. To specify a digital output position set its bit position to a "1" (refer to Chapter 4, Input/Output Selections). When reading, only specified values will be returned. When setting, only specified values will be changed.

[data] must contain 4 ASCII-HEX digits. This value will only be loaded into specified timer or PWM outputs. Any other specified output types or input types will be ignored. Unspecified timer/PWM outputs will be left unchanged.

**NOTE:** If the PWM is in high speed mode the most significant two characters of the data field will be ignored. When the PWM is configured and this command loads its pulse width value for the first time the value will be written to the EEPROM. This allows the user to vary the pulse output with this command without losing the initial starting value. To change the value in EEPROM reconfigure the PWM output using the "!" command.

Command Read Format:

```
>00*0[output][cksm]cr
```

Command Set Format:

```
>00*1[output][data][cksm]cr
```

Example:

```
Host - >00*11008006443cr  
SPIO - Acr
```

Set SPIO module PWM output #03 to output a positive 27 micro-second pulse width, which is a value of 64 Hex ( $100 \times .27\mu s = 27\mu s$ ). Set timer output #12 for a 1 second time delay ( $100 \times 10ms = 1s$ ).

Example:

```
Host - >00*0500988cr  
SPIO - A000000006400064?????1Fcr
```

Read SPIO module timer values for digital outputs #14, #12, #03 & #00. The reply shows that I/O line #14 is inactive with a timer value of 0. I/O lines #12 & 03 are inactive with timer values of 100 (64hex). I/O line #00 is an input.

### **Configure Digital I/O Lines**

This command configures the digital I/O lines as either inputs or as outputs. You must be in "SETUP MODE" to execute this command.

[I/O] can contain 0 - 4 ASCII-HEX digits, which represents 16 possible digital I/O lines. To define an I/O line as an output set its bit position to a "1". To define an I/O line as an input set its bit position to a "0" (refer to Chapter 4, Input/Output Selection). If this field is left empty all available I/O's will be changed to outputs. Any associated function for each I/O specified will be cleared (power up state, timer value, pulse duration etc.). The I/O lines not specified are left unchanged. An error will be sent if an attempt is made to change a counter start/stop input line without changing its associated count input line. When a count input line (#04, #05), having a start/stop input, is reconfigured the start/stop input (#00, #01) will be reconfigured as a normal input, if not otherwise specified.

**NOTE:** On SPIO modules that do not have all 16 I/O lines available, the unavailable I/O's will be ignored(refer to Chapter 2).

Command Format:

>00G{I/O}[cksm]cr

Example:

Host - >00G5DCcr

SPIO - Acr

Define SPIO digital lines 00 & 02 as outputs, lines 01 & 03 as inputs, and leave all other available lines unchanged.

### **Configure Digital I/O as Input**

This command configures the specified digital I/O lines as inputs. You must be in "SETUP MODE" to execute this command.

[I/O] can contain 0 - 4 ASCII-HEX digits, which represents the 16 possible digital I/O lines. To define an I/O line as an input set its bit position to a "1" (refer to Chapter 4, Input /Output Selection). All other I/O's are left unchanged. If this field is left empty all available I/O's will be changed to inputs. Any associated function for each of the I/O's specified will also be cleared (power up state, timer value, pulse duration, etc.). An error will be sent if an attempt is made to change a counter start/stop input line (#00, #01) without changing its associated count input line (#04, #05). When a count input line, having a start/stop input, is reconfigured the start/stop input will be reconfigured as a normal input.

**NOTE:** On modules that do not have all 16 I/O positions available, the unavailable I/O's will be ignored (refer to Chapter 2).

Command Format:

>00H{I/O}[cksm]cr

Example:

Host - >00H3DBcr

SPIO - Acr

Define SPIO digital lines #00 & 01 as inputs, and leave all other available lines unchanged.

## Configure Digital I/O as Output

This command defines the digital I/O lines as outputs. You must be in "SETUP MODE" to execute this command. [I/O] can contain 0 - 4 ASCII-HEX digits, which represents the 16 possible I/O positions. To define an I/O line as an output set its bit position to a "1" (refer to Chapter 4, Input /Output Selection). All other I/O's are left unchanged. If this field is left empty all available I/O's will be changed to outputs. Any associated function for each of the I/O's specified will also be cleared (power up state, timer value, pulse duration etc.). An error will be sent if an attempt is made to change a counter start/stop input line without changing its associated count input line. When a count input line (#04, #05), having a start/stop input, is reconfigured the start/stop input (#00, #01) will be reconfigured as a normal input, if not otherwise specified.

**NOTE:** On modules that do not have all 16 I/O positions available, the unavailable I/O's will be ignored. (Refer to Chapter 2.)

Command Format:

```
>00I{I/O}[cksm]cr
```

Example:

```
Host - >00IF01Fcr
```

```
SPIO - Acr
```

Define SPIO digital lines #04, 05, 06, & 07 as outputs and leave all other available lines unchanged.

## Set Digital Outputs ON or OFF

This command turns digital outputs "ON" or "OFF".

[output] field contains 0 - 4 ASCII-HEX digits, which represents 16 possible output positions. Each digit in the output field specifies the ON - OFF status of 4 I/O lines. To turn an output line on set its bit position to a "1". To turn an output off set its bit position to a "0" (refer to Chapter 4, Input/Output Selection). The I/O lines (ASCII-HEX digits) not specified are left unchanged. If this field is left empty all available outputs will be turned on. I/O line's that have been defined as inputs will be ignored.

**NOTE:** On modules that do not have all 16 I/O positions available, the unavailable I/O's will be ignored. (Refer to Chapter 2.)

Command Format:

>00J{output}[cksm]cr

Example:

Host - >00JA01Bcr

SPIO - Acr

Turn ON pre-defined outputs #07 & 05 and leave all other outputs OFF.

### Turn Digital Output ON

This command turns digital output/s "ON".

[output] field contains 0 to 4 ASCII-HEX digits, which represents 16 possible output positions. Each digit in the output field specifies the ON status of 4 I/O lines. To turn an output on, the output's corresponding bit position must be set to a "1" (refer to Chapter 4, Input/Output Selection). All other outputs will be left unchanged. If this field is left empty all available outputs will be turned on. I/O lines that have been defined as inputs will be ignored.

**NOTE:** On modules that do not have all 16 I/O positions available, the unavailable I/O's will be ignored. (Refer to Chapter 2.)

Command Format:

>00K{output}[cksm]cr

Example:

Host - >00KABcr

SPIO - Acr

Turn ON all pre-defined digital outputs.

## Turn Digital Output OFF

This command turns digital output/s "OFF".

[output] field contains 0 to 4 ASCII-HEX digits, which represents 16 possible digital output positions. Each digit in the output field specifies the OFF status of 4 I/O lines. To turn an output off the output's corresponding bit position must be set to a "1" (refer to Chapter 4, Input/Output Selection).

All other outputs will be left unchanged. If this field is left empty all available outputs will be turned OFF. I/O lines that have been defined as inputs will be ignored.

**NOTE:** On modules that do not have all 16 I/O positions available, the unavailable I/O's will be ignored. (Refer to Chapter 2.)

Command Format:

```
>00L[output][cksm]cr
```

Example:

```
Host - >00L300Fcr
```

```
SPIO - Acr
```

Turn OFF pre-defined digital outputs #04 & 05.

## Read Digital I/O States

Instructs SPIO to send to the host the current state of all its digital I/O lines. The body of the reply will contain 4 ASCII-HEX digits, which specify the "ON" or "OFF" state of 16 I/O lines. I/O lines with their corresponding bit position's set to a "1" are in the "ON" state. Available I/O lines with their corresponding bit positions set to a "0" are in the "OFF" state. All unavailable I/O lines will also be in the "OFF" state. The digits will be returned with the most significant I/O lines first in sequence with the least significant last.

Command format:

```
>00M[cksm]cr
```

Example:

Host - >00MADcr

SPIO - A00C4D7cr

Read the current state of all digital input/output lines. The reply indicates that digital lines #2, 6 & 7 are in the ON state and the remaining lines are in the OFF state.

## Set Latch Edges

Set the polarity of I/O lines, configured as normal inputs or as latch inputs, to latch either on the rising edge or the falling edge of the input signal. You must be in "SETUP MODE" to execute this command.

[input] can contain 0 - 4 ASCII-HEX digits, which represents all 16 possible input positions. To define an eligible input\* to latch on the rising edge (on-to-off) set its bit position to a "1". To define an eligible input\* to latch on the falling edge (off-to-on) set its bit position to a "0" (refer to Chapter 4, Input/Output Selection). I/O lines not specified are left unchanged. If the field is left empty all eligible inputs\* will be changed to latch on the rising edge. Ineligible inputs specified will generate an invalid value error. All outputs or unavailable I/O lines on the module will be ignored.

Command Format:

>00N{input}[cksm]cr

Example:

Host - >00N2E0cr

SPIO - Acr

Define SPIO digital input lines #00, 02, & 03 to be falling edge triggered latches and input line #01 to be a rising edge triggered latch. All other I/O lines are left unchanged.

\* Before a latch edge can be set, the I/O line must currently be defined as a latch input or as a normal input (The "!" command can be used to determine its I/O type). Otherwise, reconfigure the I/O line as a normal input using the "H" command.

### **Set Off-to-On Latches**

Set the polarity of I/O lines, configured as normal inputs or as latch inputs, to latch on the falling edge of the input signal. You must be in "SETUP MODE" to execute this command.

[input] can contain 0 - 4 ASCII-HEX digits, which represents all 16 possible input positions. To define an eligible input\* to latch on the falling edge (off-to-on) set its bit position to a "1" (refer to Chapter 4, Input/Output Selection). I/O lines not specified are left unchanged. If the field is left empty all eligible inputs\* will be changed to latch on the falling edge. Ineligible inputs specified will generate an invalid value error. All outputs or unavailable I/O lines on the module will be ignored.

Command Format:

```
>000{input}[cksm]cr
```

Example:

```
Host - >00040043cr
```

```
SPIO - Acr
```

Define SPIO digital input line #10 to be a falling edge triggered latch. All other I/O lines are left unchanged.

\* Before a latch edge can be set, the I/O line must currently be defined as a latch input or as a normal input (The "!" command can be used to determine its I/O type). Otherwise, reconfigure the I/O line as a normal input using the "H" command.

## Set On-to-Off Latches

Set the polarity of I/O lines, configured as normal inputs or as latch inputs, to latch on the rising edge of the input signal. You must be in "SETUP MODE" to execute this command.

[input] can contain 0 - 4 ASCII-HEX digits, which represents all 16 possible input positions. To define an eligible input\* to latch on the rising edge (on-to-off) set its bit position to a "1" (refer to Chapter 4, Input/Output Selection). I/O lines not specified are left unchanged. If the field is left empty all eligible inputs\* will be changed to latch on the rising edge. Ineligible inputs specified will generate an invalid value error. All outputs or unavailable I/O lines on the module will be ignored.

Command Format:

```
>00P{input}[cksm]cr
```

Example:

```
Host - >00P3A0084cr
```

```
SPIO - Acr
```

Define SPIO digital input line #13, 12, 11 & 09 to be rising edge triggered latches. All other I/O lines are left unchanged.

\* Before a latch edge can be set, the I/O line must currently be defined as a latch input or as a normal input (The "!" command can be used to determine its I/O type). Otherwise, reconfigure the I/O line as a normal input using the "H" command.

## Read Latches

Instructs SPIO to send to the host the current state of all its digital latches. The body of the message sent to the host will contain 4 ASCII-HEX digits. These digits represents the “latched” state of all the defined latches. When a bit is set to a “1” the proper latch transition has occurred and the input has latched. When a bit is set to a “0” the proper latch transition has not occurred and the input has not latched. I/O lines that are not configured as latches or that are not available on the module will also have their corresponding bit position’s set to a “0”. The digits will be returned with the most significant I/O lines first in sequence with the least significant last. The state of the latches are not affected by this command.

Command format:

```
>00Q[cksm]cr
```

Example:

```
Host - >00QB1cr
```

```
SPIO - A0006C6cr
```

Read the current state of all digital latch inputs. The reply indicates that latch inputs #01 & 02 are in the latched state and any remaining latch inputs are in the unlatched state.

## Read And Clear Latches

This commands sends to the host the current state of all latches and then clears the specified latch inputs. The message returned to the host will contain 4 ASCII-HEX digits, which specify the “latched” state of all defined latches. When a bit is set to a “1” the proper latch transition has occurred and the input has latched. When a bit is set to a “0” the proper latch transition has not occurred and the input has not not latched. I/O lines that are not configured as latches or that are not available on the module will also have their corresponding bit position’s set to a “0”. The digits will be returned with the most significant I/O lines first in sequence with the least significant last.

[input] can contain 0 - 4 ASCII-HEX digits, which represents all 16 possible latch input positions. To clear a latch set its bit position to a

“1” (refer to Chapter 4, Input/Output Selection). Bit positions set to a “0” will not be cleared. All I/O lines that are not latches or that are not available on the module will be ignored. If this field is left empty all latches will be cleared.

Command format:

```
>00R{input}[cksm]cr
```

Example:

```
Host - >00R4E6cr
```

```
SPIO - A0006C6cr
```

Read all latch inputs and clear latch input #02. The reply indicates that latch inputs #01 & 02 are in the latched state and any remaining latch inputs are in the unlatched state.

## Clear Latches

This commands clears specified latch inputs.

[input] can contain 0 - 4 ASCII-HEX digits, which represents all 16 possible latch input positions. To clear a latch set its bit position to a “1” (refer to Chapter 4, Input/Output Selection). Bit positions set to a “0” will not be cleared. All I/O lines that are not latches or unavailable on the module will be ignored. If this field is left empty all latches will be cleared.

Command format:

```
>00S{input}[cksm]cr
```

Example:

```
Host - >00S2E5cr
```

```
SPIO - Acr
```

Clear latch input #01.

## Start/Stop Counters

Starts and stops or enables specified defined counters inputs. An error will be returned if a counter input has not been defined and has not been specified in the input field.

If the counter input specified is type "01" or "02" (refer to the "READ/SET DIGITAL TYPES" command) the counting is started and stopped with this command.

If the counter specified uses a start/stop input line this command is only used to enable the counter. Once enabled the start/stop input controls the counting. Disabling of the counter occurs automatically when the start/stop input line stops the count. The counter must then be re-enabled to restart the count.

[input] can contain 0 - 4 ASCII-HEX digits, which represents all 16 possible I/O positions. To start/enable a counter input set its bit position to a "1" (refer to Chapter 4, Input/Output Selection). To stop a counter input set its bit position to a "0". All I/O lines that are not counter inputs or are non-existent on the module will be ignored provided a counter input has been specified.

Command Format:

```
>00T{input}[cksm]cr
```

Example:

Host - >00T1E5cr

SPIO - Acr

Start counting input signals from counter input #00 of SPIO.

## Start Counters

Starts or enables specified defined counters inputs.

If the counter input specified is type "01" or "02" (refer to the "READ/SET DIGITAL TYPES" command) the counting is started with this command. The count will be added to the previous count value unless it was previously cleared.

If the event counter specified uses a start/stop input line this command is used only to enable the counter. Once enabled, the start/stop input can start the counting. When the start signal is sensed the count value is cleared to zero and the counter is ready to count. The count value cannot be read until the start/stop input line stops the count. If the count value is read during the counting a value of 0000 will be returned. After the start/stop line stops the counter the count value can be read. The counter must then be re-enabled to restart the count.

[input] can contain 0 - 4 ASCII-HEX digits, which represents all 16 possible I/O positions. To start/enable a counter input set its bit position to a "1" (refer to Chapter 4, Input/Output Selection). All I/O lines that are not counter inputs or are nonexistent on the module will be ignored provided a counter input has been specified.

Command Format:

```
>00U{input}[cksm]cr
```

Example:

```
Host - >00U1016cr
```

```
SPIO - Acr
```

Enable the start/stop input line of counter input #04 of SPIO.

## Stop Counters

Stops specified defined counters inputs.

If the counter input specified is type "01" or "02" (refer to the READ/SET DIGITAL TYPES" command) the counting is stopped with this command. All other counter types will be not be affected.

[input] can contain 0 - 4 ASCII-HEX digits, which represents all 16 possible I/O positions. To stop a counter input set its bit position to a "1" (Refer to Input/Output Selection, Chapter 4). All I/O lines that are not counter inputs or are non-existent on the module will be ignored.

Command Format:

```
>00V{input}[cksm]cr
```

Example:

Host - >00V2E8cr

SPIO - Acr

Stop counting input signals from counter input #01 of SPIO.

## Read Counters

Returns the count value of all I/O lines specified.

When reading these count values the data returned will contain a field of 4 ASCII characters for each I/O position specified. If the I/O position requested is an event counter input, the count value will be returned as four ASCII-HEX digits. If the I/O position requested is not defined as an event counter or does not exist on the module "?????" will be returned. The most significant I/O specified will be sent first in sequence with the least significant I/O last. The maximum count value before the count rolls over is FFFF hex or 65,535. However, the SPIO counts the number of roll overs up to FF hex or 255. This means the maximum count that can be recorded is FFFFFFFF hex or 16,777,215. To read the roll over count along with the counter value a "\*" must be placed in the input field instead of the I/O position bits. The data returned will contain a field of six ASCII characters for each of the two possible event counters. The first two ASCII-HEX digits will be the roll over count and the remaining four will be the current count value. If a counter is not defined "??????" will be returned. The most significant event counter will be sent first with the least significant last. If the event counter input specified is type "01" or "02" (refer to the Read/Set Digital Types command) the current count value can be read at any time. If the event counter specified uses a start/stop input line the count value cannot be read until the start/stop input line stops the count. If the count value is read during the counting a value of 0000 will be returned.

[input] can contain 0 - 4 ASCII-HEX digits, which represents all 16 possible I/O positions. To read an I/O position set its bit position to a "1" (Refer to Input/Output Selection, Chapter 4). Bit positions set to a "0" will not be read. To read the roll over count along with the counter value for the two possible event counters place a "\*" in the input field instead of the I/O position bits. If this field is left empty all positions will be read.

Command format:

>00W{input}[cksm]cr

Example:

```
Host - >00W*E1cr  
SPIO - A??????0C01A8C7cr
```

Read the roll over count and the current count value of the two possible event counters on the SPIO. The reply indicates that the most significant counter is not used and the least significant counter has a count value of 786,856 (0C01A8 hex).

## Read And Clear Counters

Returns the current count value of any event counter specified and clears the count values of event counters defined as types "01" or "02" to zero. Event counters defined as types "03 - 0A" can be read using this command but they will not be cleared (their count values are cleared to zero only when the counter is started via its start/stop input line).

When reading these count values the data returned will contain a field of 4 ASCII characters for each I/O position specified. If the I/O position requested is an event counter input, the count value will be returned as four ASCII-HEX digits. If the I/O position requested is not defined as an event counter or does not exist on the module "?????" will be returned. The most significant I/O specified will be sent first in sequence with the least significant I/O last. The maximum count value before the count rolls over is FFFF hex or 65,535. However, the SPIO counts the number of roll overs up to FF hex or 255. This means the maximum count that can be recorded is FFFFFFFF hex or 16,777,215. To read the roll over count along with the counter value use the "READ COUNTER" command. If the event counter input specified is type "01" or "02" (refer to the "READ/SET DIGITAL TYPES" command) the current count value can be read at any time. If the event counter specified uses a start/stop input line the count value cannot be read until the start/stop input line stops the count. If the count value is read during the counting a value of 0000 will be returned.

[input] can contain 0 - 4 ASCII-HEX digits, which represents all 16 possible I/O positions. To read and clear an event counter set its bit position to a "1". Event counter types "03 - 0A" will not be cleared. All other I/O positions will not be affected.

Command format:

```
>00X{input}[cksm]cr
```

Example:

```
Host - >00X3EBcr
```

```
SPIO - A????01A8D6cr
```

Read I/O lines #01 & #00 and clear any defined event counter. The reply indicates that the most significant counter is not used and the least significant counter has a count value of 424 (01A8 hex).

## Clear Counters

This command clears the specified event counter types "01" & "02" count value to zero. Event counters types "03 - 0A" will not be cleared (their count values are cleared to zero only when the counter is started via its start/stop input line).

[input] can contain 0 - 4 ASCII-HEX digits, which represents all 16 possible I/O positions. To clear an event counter set its bit position to a "1". Event counter types "01" & "02" will be cleared, types "03 - 0A" will not be cleared. All other I/O's will be ignored.

Command format:

```
>00Y{input}[cksm]cr
```

Example:

```
Host - >00YB9cr
```

```
SPIO - Acr
```

Clear all defined event counters to zero on SPIO.

## Set Time Delay

This command configures output lines as time delayed outputs or as normal outputs. You must be in "SETUP MODE" to execute this command. Time delayed outputs have a 10 millisecond resolution with a maximum time delay of 10.92 minutes. The digital output on and off commands activate and deactivate time delays.

[output] must contain 1 - 4 ASCII-HEX digits, which represents the 16 possible output lines. To change an output line to a new type set its bit position to a "1" (refer to Chapter 4, Input/Output Selection). Output lines with their bit position set to a "0" will retain their current type. All input lines will be ignored. An error will be sent if the specified output line is currently configured as a PWM. The power up state for each output specified will be cleared (set to the off state).

[mod] contains 1 ASCII character that specifies the type of output to be configured. Valid modifiers are:

<b>mod</b>	<b>Type Code</b>	<b>Description</b>
G	80	Normal on/off.
H	81	Time delayed - transition from OFF to ON, turn ON for desired time, then turn OFF.
I	82	Time delayed - transition from OFF to ON, stay OFF for desired time, then turn ON.
J	83	Time delayed - transition from ON to OFF, turn OFF for desired time, then turn ON.
K	84	Time delayed - transition from ON to OFF, stay ON for desired time, then turn OFF.

[data] must contain 1 - 4 ASCII-HEX digits. This value will only be loaded into the specified time delay outputs. Any other output types or input types will be ignored. Unspecified timer delay outputs will be left unchanged. To determine the actual time delay: time delay = [data] x 10 milliseconds.

Command Set Format:

```
>00Z[output][mod][data][cksm]cr
```

Example:

Host - >00Z300H177064cr

SPIO - Acr

Configure output lines #09 & #08, of SPIO, to turn ON for 1 minute [6000(1770h) x .01 sec = 60 sec] and then turn OFF when going from OFF to ON. The modifier is represented by the non-ASCII- HEX DIGIT - the "H" is this case.

### **Retrigger Time Delay**

This command restarts an active time delay output. This command is used to extend the original time value. Time delays that are inactive (outputs that have completed their timed cycle) will not be affected by this command. The digital output on and off commands activate and deactivate time delay outputs.

By constantly sending this command to retrigger an active time delay a watchdog output can be created.

[output] can contain 0 - 4 ASCII-HEX digits, which represents the 16 possible output lines. To retrigger an active time delayed output line set its bit position to a "1" (refer to Chapter 4, Input/Output Selection). Time delayed output lines with their bit position set to a "0" will not be retrIGGERED. All other types of I/O lines are ignored.

Command format:

>00h{output}[cksm]cr

Example:

Host - >00h200B9cr

SPIO - Acr

Retrigger time delay output #9 on SPIO.

## Read Digital I/O Configuration

This command returns the current digital I/O configuration. I/O lines with their corresponding bit position's set to a "1" are configured as outputs. Available I/O lines with their corresponding bit position's set to a "0" are configured as inputs (refer to Chapter 4, Input/Output Selection). All unavailable I/O lines also will have their corresponding bit position's set to a "0". The I/O positions will be returned with the most significant I/O lines first in sequence with the least significant I/O lines last.

Command Format:

```
>00j[cksm]cr
```

Example:

Host - >00jCAcr

SPIO - A00C0D3cr

Read the current digital input/output line configuration. The reply indicates that digital lines #06 & 07 are outputs and the remaining lines are inputs or unavailable (refer to Chapter 2).

## ANALOG COMMANDS

### Analog Data

This section deals with the interpretation of the data received by and sent to analog channels. All analog commands dealing with reading and setting analog levels are capable of handling up to 12 bits of resolution. However, the circuitry of a particular SPIO module may only be capable of 8-bits of resolution, so it is important to know what module is being used. (Refer to Chapter 2.)

### Analog Inputs

All analog input values are made up of four ASCII-HEX digits. The most significant digit will be a "1".

**NOTE:** This digit will be used at a later date to indicate an over/under condition.

#### 12-bit resolution

zero scale = 1000 hex  
full scale = 1FFF hex

#### 8-bit resolution

zero scale = 1000 hex  
full scale = 10FF hex

To determine the level received the following steps must be used:

- 1) Convert value received to decimal and subtract 4096 from the value.
- 2) Divide the A/D reference voltage by the A/D's full scale resolution. (12-bit = 4095, 8-bit = 255)
- 3) Multiply the result of step 1 by the result in step 2.

Example 1:            Input value is 1BF7 hex. (This assumes a 12-bit A/D with a A/D reference voltage of +5.00 volts.)  
                          7159 - 4096 = 3063    5volts/4095 = 0.00122  
                          3063 x 0.00122 volts = 3.74 volts

Example 2: Input value is 1072 hex. (This assumes an 8-bit A/D with a A/D reference voltage of +5.00 volts.)  
 $4210 - 4096 = 114$   $5\text{volts}/255 = 0.01961$  volts  
 $114 \times 0.01961$  volts = 2.24 volts

## Analog Outputs

All analog output values are made up of three ASCII-HEX digits. These digits represent the voltage level that is to be sent to the output.

### 12-bit resolution

zero scale = 000 hex  
full scale = FFF hex

### 8-bit resolution

zero scale = 000 hex  
full scale = 0FF hex

To determine a value that must be sent to a SPIO module to obtain the desired output level use the following steps:

- 1) Divide desired value by D/A reference voltage.
- 2) Multiply the result by the D/A's full scale resolution.  
(12-bit = 4095, 8-bit = 255)
- 3) Convert the total to hexadecimal.

Example 1: Output a 3.74 volt level. (This assumes a 12-bit D/A with a D/A reference voltage of +5.00 volts.)

$$3.74 / 5 = 0.748 \times 4095 = 3063 = \text{BF7 hex.}$$

Example 2: Output a 2.24 volt level. (This assumes an 8-bit D/A with a D/A reference voltage of +5.00 volts.)

$$2.24 / 5 = 0.448 \times 255 = 114 = \text{072 hex.}$$

## Set Analog Output Level

This command can output a 12-bit analog level to all D/A channels specified. (Refer to Analog Data.)

[channel] field must contain 4 ASCII-HEX digits, which represents 16 possible D/A channels. Set the D/A channel/s bit position to a "1" to output the specified level (refer to Chapter 4, Input/Output Selection).

[level] must contain a 12-bit ASCII-HEX value (3 ASCII-HEX digits) to be written to each channel identified in the channel field. Level range for a 12-bit D/A is 000H to FFFH. Level range for an 8-bit D/A is 000H to 0FFH (refer to Chapter 4, Analog Commands).

**NOTE:** Channels specified in the channel field that are not available on a SPIO module will be ignored. If the module has only 8-bit D/A's, the most significant digit in the level field will be ignored.

Command Format:

```
>FFJ[channel][level][cksm]cr
```

Example:

```
Host - >FFJ000107F44cr
```

```
SPIO - Acr
```

Output +2.49 volts (7F hex) on D/A channel #0. (This assumes a voltage reference of +5.00 volts with an 8-bit D/A.)

## Read Analog Output Level

This command reads a 12-bit value from analog D/A outputs whose bit position is set to a "1" in the channel field (refer to Chapter 4, Input/Output Selection). Modules that have 12-bit D/A's will have a value range of 000H to FFFH. Modules that have 8-bit D/A's will have a value range of 000H to 0FFH (refer to Chapter 4, Analog Commands). The most significant output level will be returned first in order with the least significant output level last.

[channel] field contains 0 - 4 ASCII-HEX digits, which represents 16 possible output positions. If this field is left empty all 16 outputs will be read.

**NOTE:** Channels specified in the channel field that are not available on SPIO will return "???" in the corresponding value area.

Command Format:

```
>FFK[channel][cksm]cr
```

Example:

Host - >FFKF1Dcr

SPIO - A?????00007FB7cr

Read the D/A output levels for channels #0, 1, 2 & 3. The reply indicates that channel #0 has an output of +2.49 volts (07F)hex, channel #1 has an output of 0 volts (000), and channels #2 & #3 do not exist on that module. (This assumes a voltage reference of +5.00 volts with an 8-bit D/A.)

### Read Analog Input Level

Instructs SPIO to return the current levels of all the specified analog inputs. Modules that have 12-bit A/D's will have a normal value range of 1000H - 1FFFH. Modules that have 8-bit A/D's will have a normal value range of 1000H - 10FFH (refer to Chapter 4, Analog Data). The most significant input level will be returned first in order with the least significant output level last.

[channel] field contains 0 - 4 ASCII-HEX digits, which represents 16 possible analog input positions. If field is left empty all 16 A/D inputs will be read. To specify a channel set its corresponding bit position to a "1" (refer to Chapter 4, Input/Output Selection).

**NOTE:** Channels specified in the channel field that are not available on a SPIO module will return "?????" in the corresponding value area. (Refer to Chapter 2.)

Command format:

>FFL[channel][cksm]cr

Example:

Host - >FFL1030Fcr

SPIO - A????107F103FB4cr

Read the A/D input levels for channels #0, 1 & 8. The reply indicates that channel #0 has an input level of +1.24 volts (3F)hex, channel #1 has an input level of +2.49 volts (7F)hex and channel #8 does not exist on that module. (This assumes a voltage reference of +5.00 volts with an 8-bit A/D.)

## Set Individual Analog Output Levels

This command can output an individual 12-bit level to each D/A output specified.

[channel] field must contain 4 ASCII-HEX digits, which represents 16 possible D/A channels. To select a D/A channel set its bit position to a "1" (refer to Chapter 4, Input/Output Selection).

[level] field must contain a 12-bit value (3 ASCII-HEX digits) for each channel identified in the channel field. The level digits must be in order starting with the highest channel specified and ending with the lowest channel specified. Level range for a 12-bit D/A is 000H to FFFH. Level range for an 8-bit D/A is 000H to 0FFH (refer to Chapter 4, Analog Data).

**NOTE:** Channels specified in the channel field that are not available on a SPIO module will be ignored. The maximum of two channels may be specified in the level field. If the module only has 8-bit D/A's the most significant digit in the level field will be ignored. A data field error will occur if the command does not include three digits in the level field for each output represented in the channel field.

Command Format:

```
>FFS[channel][level][cksm]cr
```

Example:

```
Host - >FFS000307F03FF8cr
```

```
SPIO - Acr
```

Output +1.24 volts (3F hex) on D/A channel #0 and +2.49 volts (7F hex) on D/A channel #1. (This assumes a D/A reference voltage of +5.00 volts with an 8-bit D/A.)

## Read/Set Analog Output Power Up Levels

This command can read or set the current D/A output power up levels. The "set" portion of this command can be executed only when the unit is in "SETUP MODE."

When reading the power up levels the data returned to the host will contain 3 ASCII-HEX digits for each available D/A channel. The most significant D/A channel will be sent first in sequence with the least significant D/A channel last. All 16 D/A channels may not be available on some modules therefore only the available channels will be sent back to the host (refer to Chapter 2).

[modifier] must contain 1 ASCII-HEX digit; 0 = read configuration, 1 = set configuration.

[channel] must contain 4 ASCII-HEX digits, which represents the 16 possible D/A channels. To define a D/A channel set its bit position to a "1" (refer to Chapter 4, Input/Output Selection).

[level] must contain 3 ASCII-HEX digits. Level range for a 12-bit D/A is 000H to FFFH. Level range for an 8-bit D/A is 000H to 0FFH.

**NOTE:** If the D/A has only 8-bits of resolution the most significant digit will be ignored (refer to Chapter 4, Analog Data).

Command Read Format:  
>FF&0[cksm]cr

Command Set Format:  
>FF&1[channel][level][cksm]cr

Example:

Host - >FF&100010BF5Ccr

SPIO - Acr

Set SPIO module D/A output channel #0 to a power up level of +3.74 volts (BF hex). (This assumes a D/A reference voltage of +5.00 volts with an 8-bit D/A.)

Example:

Host - >FF&0E2cr

SPIO - A000BF48cr

Read SPIO module power up levels for the analog D/A outputs. The reply shows that channel #0 has a power up output level of +3.74 volts (0BF hex) and channel #1 has a power up output level of 0 volts (000 hex). (This assumes a D/A reference voltage of +5.00 volts with an 8-bit D/A.)

## Chapter 5: Application Program Interface

### PROGRAMMING WITH QuickBASIC

Included with your module is the source code of a demo program written in QuickBASIC. See Appendix F. This was compiled using QuickBASIC version 4.5 and linked with Microsoft's Link version 3.05.

There are two library files included with your module.

**SPQLB1QB.LIB** is necessary to create an executable version of a program. **SPQLB1QB.QLB** is necessary to run a program from the QuickBASIC editor in interpretive mode. To create or edit a program with the QuickBASIC editor that calls the API, use the following statement:

```
QB filename /L SPQLB1QB
```

This statement links the API QuickBASIC library with the program *filename* and allows the program to be run in the interpretive mode. If you are using another editor to create or edit your Basic programs, the program must be compiled and linked with the API. To compile your program, use the following command at the DOS prompt:

```
BC filename;
```

To link the object code with the API, use the following command:

```
LINK filename+SPAPI1;
```

Remember that your executable file will need access to the QuickBASIC file BRUN45.EXE.

The file SPINC1QB.BI is the header file for the API. To use this in your program put **'\$INCLUDE: 'SPINC1QB'** at the beginning of your program. This file contains all the DECLARE statements for the API routines. For example:

Put other '\$INCLUDE: '...' statements here.

```
'$INCLUDE: 'SPINC1QB.BI'
```

Put your program here.

## PROGRAMMING WITH C

Included with your module is the source code of a demo program written in C. This was compiled using Borland C++ version 2.0.

In this section, we assume that you are familiar with the Borland C++ compiler. If necessary, consult your manuals for more information.

The file SPINC1BC.H is the header file for the API. To use this in your program put **#include <SPINC1BC.h>** at the beginning of your program. For example:

```
/* Put other #include statments here */  
#include <SPINC1BC.h>  
int main (void)  
{ /* Put your program here */  
    return 0;  
}
```

To compile your program from the DOS command line, use the following commands. Remember to change the include and library directories to your own.

```
BCC -c -iC:\BC\INC -ml -v filename.c
```

```
TLINK /c /d /LC:\BC\LIB /v /x C0L filename SPAPI1,  
filename.exe,,CL,
```

Parameters for the BCC command:

-c	Compile to .OBJ but do not link
-I	Directories for include files
-ml	Compile using large memory model
-v	Source debugging ON
<i>filename.c</i>	Name of your source code file.

Parameters for the TLINK command:

/c	Treat case as significant in symbols
/d	Warn if duplicate symbols in library
/L	Specify library search paths
/v	Include full symbolic debug information
/x	Do not create file map
C0L	Object file that does C's setup before your program and the clean up after it.
<i>filename</i>	Your object file created with BCC.
SPAPI1	Our program interface object file.
<i>filename.exe</i>	Output file name.
CL	C's large memory model library.

You can create a make file to automatically compile and link any files that change. This saves time in the program development cycle. The following is an example of a make file. To run the make file you would use **MAKE -f*filename*.MAK**.

```
# Beginning of make file
.autodepend

CC = BCC
LN = TLINK
CFLAGS = -c -IC:\BC\INC -ml -v
LFLAGS = /c /d /LC:\BC\LIB /v /x

# Any other macro's go in here.

.c.obj:
    $(CC) $(CFLAGS) $*.C

# Any other implicit definitions go in here.

filename.exe: filename.obj SPAPI1.obj
    $(LN) $(LFLAGS) C0L filename API, filename.exe,CL,

filename.obj: filename.c

# Any other files go in here.

# End of make file.
```

## PROGRAMMING WITH PASCAL

Included with your module is the source code for a demo program written in Turbo Pascal. The source should compile using Turbo Pascal versions 4 or later.

Before going any farther, we will briefly describe the assumptions made while writing this section. First we assume that you are familiar with the Turbo Pascal compiler and the features that Borland added in version 4.0. Most notably, we assume that you are familiar with:

- the concept of a UNIT (files with the .TPU extension)
- the meaning of the USES statement

If necessary, consult the Turbo Pascal manual for more information.

The file SPINC1TP.PAS is the source code for the UNIT. To use this in your program put **Uses SPINC1TP** at the beginning of your program. For example:

```
Program Example;  
Uses SPINC1TP;  
Begin  
  { Put your program here }  
End.
```

The unit file SPINC1TP.TPU was compiled using Turbo Pascal version 6.0. If you have a different version of Turbo Pascal, recompile the unit file before using it.

To compile the Unit using the Turbo Pascal command line compiler type **TPC SPINC1TP**.

To compile a program that uses the API and the Unit together, you would type **TPC yourProg /B**.

## STATUS CODE SUMMARY

Most of functions in the API return a status code. They are described below. Note that the description to the status returned by the function **Set\_Com\_Port** is described in the functional overview.

- 00h **POWER\_UP\_CLEAR\_EXPECTED** The module returns this error when you use a function that sends a command message, other than **Power\_Up\_Clear**, immediately after powering up the module. A function that returns this error will not be executed by the module. After you clear the error at power-up, if it occurs again, you will know that the module has experienced a power failure and that corrective measures should be taken. When the module returns this error, it will be set to the user-defined power-up condition. Be careful when deciding your power-up condition.
- 01h **UNDEFINED\_COMMAND** The module returns this error when it receives a command message that it does not recognize.
- 02h **CHECKSUM\_ERROR** The module returns this error when the checksum that it calculates is not the checksum that the host sent in the command message.
- 03h **INPUT\_BUFFER\_OVERRUN** The module returns this error when the command message sent is too long.
- 04h **NON\_PRINTABLE\_ASCII** The module returns this error when the command message contains a byte that is not between 21h(33) and 7Fh(127).
- 05h **DATA\_FIELD\_ERROR** The module returns this error when there are not enough bytes in the command message.
- 06h **COMMUNICATION\_TIMEOUT** The module returns this error when the communication line between the host and it are inactive for longer than the specified time. This time is set with the **Comm\_Timer** function. This error can be caused by incorrectly calculating the amount of time between commands that the host sends to the module. For example, if you send one command

every minute and set the communication timer to 30 seconds, this error will occur. If you correctly calculated the amount of time between commands, it could mean that the module has lost the host. The module will be reset to the power-up condition when this error occurs. The function that returns this error will not be executed by the module.

- 07h **INVALID\_SPECIFIED\_VALUE** The module returns this error when an invalid data value is sent.
- 10H **MODULE\_INTERFACE\_ERROR** Improper interface connection for current SPIO firmware.

**NOTE:** This error may not be available with all module interfaces! (DAfor example).

- 60h **API\_NOT\_ACTIVE** The API returns error when a function is called before the **Start\_API** function is used or after and **End\_API** function is used.
- 61h **COM\_DRIVER\_NOT\_INSTALLED** The API returns this error when a function that sends a message to the module is called before using the **Set\_Com\_Port** function is called.
- 62h **REPLY\_TIME\_OUT** The API returns this error when the module does not respond in the specified amount of time. This can be set with the **Set\_API\_Timeout** function. When you specify this value, make sure you take into account the value of the turn-around delay. This is initially set to 1500 milliseconds.
- 63h **INVALID\_CHARACTER** The API returns this error when it expects a hexadecimal character and the one received is not one.
- 64h **INVALID\_REPLY** The API returns this error when the reply in four pass mode is not the same as the command message sent out.

- 65h    **INVALID\_REPLY\_CHECKSUM** The API returns this error when the checksum of the reply message is not the same as the calculated checksum.
  
- 66h    **REPLY\_TOO\_LONG** The API returns this error when the reply was longer than the internal reply buffer. This is 254 bytes.
  
- 1     **NO\_ERROR** The API returns this when no error occurred.

## FUNCTIONAL OVERVIEW

This section describes each of the functions that are included in the API.

Each function that returns data from a module will store that data in an internal data array. The data can be accessed by using the function **Get\_Data**. Any array element that contains a -1 indicates that the data for that line is undefined. Before the values of the array are read, the return status from the previously called function should be checked, to determine if it was executed properly. The values of the array will be retained until another function sends a message to the module. The data for a line will be in its corresponding array element, unless otherwise specified. For example: to access the data for line 4, pass a 4 to the function **Get\_Data**.

Many of the functions require you to specify lines or channels. To specify a line or channel, you must set its corresponding bit to one.

For example: Suppose you want to specify lines 0, 4, and 7.

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1

So you may pass any of the following equivalent values: 145 (decimal), 91 (hexadecimal), 0000000010010001 (binary).

## API\_Version

**Function:** Get the API version number.

**BASIC Syntax:** API.Version%

**C Syntax:** unsigned far pascal API\_Version (void);

**Pascal Syntax:** function API\_Version : word;

**Remarks:** Use this function to determine what version of the API you are using.

**Return Value:** The API's major version is returned in the most significant byte and the minor version number is in the least significant byte.

## Clear\_Counters

**Function:** Clears the specified software controlled event counters.

**BASIC Syntax:** Clear.Counters% (preamble%, lines%)

**C Syntax:** int far pascal Clear\_Counters (unsigned\_char preamble, unsigned lines);

**Pascal Syntax:** function Clear\_Counters (preamble : byte; lines : word) : integer;

**Remarks:** **Clear\_Counters** clears the event counters specified in lines. Event counter types 01h and 02h will be cleared by this function. Event counter types 03h - 0Ah will not be cleared, they are cleared when they are enabled. To specify which event counters to clear, set their corresponding bit in lines to one.

**Return Value:** The status code is returned.

**See Also:** Clear Counters in Chapter 4.

## Clear\_Latches

**Function:** Clears the specified latched inputs.

**BASIC Syntax:** Clear.Latches% (preamble%, lines%)

**C Syntax:** int far pascal Clear\_Latches (unsigned\_char preamble, unsigned lines);

**Pascal Syntax:** function Clear\_Latches (preamble: byte, lines: word) : integer;

**Remarks:** **Clear\_Latches** clears the latched inputs specified in lines. To specify which latched inputs to clear, set their corresponding bits in lines to one.

**Return Value:** The status code is returned.

**See Also:** Clear Latches in Chapter 4.

## Comm\_Timer

**Function:** Set the communication timer's maximum value. You must be in "SETUP MODE" to execute this command.

**BASIC Syntax:** Comm.Timer% (preamble%, value%)

**C Syntax:** int far pascal Comm\_Timer (unsigned char preamble, unsigned value);

**Pascal Syntax:** function Comm\_Timer (preamble: byte; value: word) : integer;

**Remarks:** **Comm\_Timer** sets the maximum amount of time the module will wait for activity on the communication line to value. The communication timer count starts at zero when the module is powered up and is reset to zero each time the module responds to the host. If

the module's communication timer reaches its maximum value before the host sends a message to that module, it assumes that it has lost its connection with the host. When this occurs, the module will reset itself to the user defined power-up condition. The return status of the next function executed will be **COMMUNICATION\_TIMEOUT**. The valid values are 0014h to FFFFh. To convert value to milliseconds, multiply it by 10. Other valid values are shown below.

- 0 - Disable timer.
- 1 - 10 seconds.
- 2 - 1 minute.
- 3 - 10 minutes.
- 4 - Disable timer.
- 5 - 10 seconds.
- 6 - 1 minute.
- 7 - 10 minutes.

**Return Value:** The status code is returned.

**See Also:** Communication Timer in Chapter 4.

## Config\_Digital\_Lines

**Function:** Configure digital I/O lines as inputs or outputs. You must be in "SETUP MODE" to execute this command.

**BASIC Syntax:** Config.Digital.Lines% (preamble%, lines%, value%)

**C Syntax:** int far pascal Config\_Digital\_Lines (unsigned char preamble, unsigned lines, unsigned value);

**Pascal Syntax:** function Config\_Digital\_Lines (preamble: byte; lines, value : word) : integer;

**Remarks:** **Config\_Digital\_Lines** configures the digital I/O lines to either inputs or outputs. The meaning of the number passed in value is described below.

- 0 - Change all 16 lines to outputs.
- 1 - Change lower 4 lines as specified.
- 2 - Change lower 8 lines as specified.
- 3 - Change lower 12 lines as specified.
- 4 - Change all 16 lines as specified.

To set an I/O line to an output, set its corresponding bit position in lines to a one. To set an I/O line to a input, set its bit position to a zero. Lines which do not exist on a module will be ignored. Any other configuration, such as the power-up output state and pulse duration, associated with a newly configured I/O line will be cleared.

- If you pass a 0005h in lines and a 1 in value, lines 0 and 2 will be configured as outputs, lines 1 and 3 will be configured as inputs, and all other lines will be unchanged.

- If you pass a 0005h in lines and a 2 in value, lines 0 and 2 will be configured as outputs, lines 1, 3, 4, 5, 6, and 7 will be configured as inputs, and all other lines will not be changed.

**Return Value:** The status code is returned.

**See Also:** Configure Digital I/O Lines in Chapter 4.

## Config\_Digital\_Inputs

**Function:** Configure digital I/O lines as inputs. You must be in "SETUP MODE" to execute this command.

**BASIC Syntax:** Config.Digital.Inputs% (preamble%, lines%)

**C Syntax:** int far pascal Config\_Digital\_Inputs (unsigned char preamble, unsigned lines);

**Pascal Syntax:** function Config\_Digital\_Inputs (preamble : byte; lines : word) : integer;

**Remarks:** **Config\_Digital\_Inputs** configures the digital I/O lines specified in lines to inputs. To configure a line as an input set its corresponding bit position to a one. Bit positions set to zero cause no change in that line's configuration. Lines that do not exist on a module will be ignored. Any other configuration, such as the power-up output state and pulse duration, associated with a newly configured I/O line will be cleared.

**Return Value:** The status code is returned.

**See Also:** Configure Digital I/Os As Inputs in Chapter 4.

**Function:** Configure digital I/O lines as outputs.

### **Config\_Digital\_Outputs**

**BASIC Syntax:** Config.Digital.Outputs% (preamble%, lines%)  
You must be in "SETUP MODE" to execute this command.

**C Syntax:** int far pascal Config\_Digital\_Outputs (unsigned char preamble, unsigned lines);

**Pascal Syntax:** function Config\_Digital\_Outputs (preamble : byte; lines : word) : integer;

**Remarks:** **Config\_Digital\_Outputs** configures the digital I/O lines specified in lines to outputs. To configure a line as an output set its corresponding bit position to a one. Bits positions that are set to zero will cause no change in that line's configuration. Lines that do not exist on a module will be ignored. Any other configuration, such as the power-up output state and pulse duration, associated with a newly configured I/O line will be cleared.

**Return Value:** The status code is returned.

**See Also:** Configure Digital I/O's As Outputs in Chapter 4.

## End\_API

**Function:** Signals the end of API usage and terminates the communication driver.

**BASIC syntax:** End.API

**C Syntax:** void far pascal End\_API (void);

**Pascal Syntax:** procedure End\_API;

**Remarks:** Signals the end of API usage and terminates the communication driver. If you do not use this command before exiting your program, it can cause your system to lock up. A good place to put this command is in an auto-exit procedure.

**Return Value:** None.

**See Also:** Start\_API.

## Get\_Data

**Function:** Returns the data received from the module.

**BASIC Syntax:** Get.Data& (value%)

**C Syntax:** long far pascal Get\_Data (unsigned\_char value);

**Pascal Syntax:** function Get\_Data (value : byte) : longint;

**Remarks:** **Get\_Data** returns the specified element of data that is stored in the API's internal array. The valid values range from zero to a maximum of fifteen, depending on the data retrieval function used prior to this one. If the data returned is a "1", then one of the following is true:

- 1) a line specified is not supported on your module.
- 2) you are accessing an invalid field for the particular function.

3) the line is not configured as the requested I/O type.  
values of the array will be retained until another function sends a message to the module. The contents of the data elements are described in their associated commands.

**Return Value:** The value of the specified array element.

**See Also:** The beginning of the Functional Overview in this Chapter.

## Get\_Message

**Function:** Retrieve the command message that is sent to the module.

**BASIC Syntax:** Get.Message  
(VARSEG(msg%(0)), VARPTR (msg%(0)))

**C Syntax:** void far pascal Get\_Message (char far \*msg);

**Pascal Syntax:** procedure Get\_Message (msg : pointer);

**Remarks:** **Get\_Message** sets the 254 bytes pointed to by msg to the value of the API's internal message buffer. This function allows you to display the command message that is sent to the module. The end of the message string is marked by a NULL character.

**Return Value:** Nothing.

### C Example:

```
#include <SPINC1BC.H>
char buffer[254];
int main(void)
{ int status;
  Start_Api();
  if ((Set_Com_Port(0x03F8, 4, 9600L,
    0, 8, 1, 1)) <= 2)
```

```

    {   cputs("Could not set up the COM"
            "port.\r\n");
        return 1;
    }
    status = Power_Up_Clear(0x00);
    Get_Message(buffer);
    cprintf("%s\r\n", buffer);
    if (status == NO_ERROR)
        Get_Reply(buffer);
        cprintf("%s\r\n", buffer);
    }
    else
        cputs("Power_Up_Clear
failed.\r\n");
    End_API();
    return 0;
}

```

### Pascal Example:

```

uses pas_api;
var buffer : array[0..253] of char;
    i, status : integer;
begin
    i := 0;
    Start_API;
    if ((Set_Com_Port($03F8, 4, 9600, 0, 8, 1, 1)) <= 2)
    then
        begin;
            writeln('Could not set up COM port. ');
            halt;
        end;
    status := Power_Up_Clear($00);
    Get_Message(@buffer);
    i := 0;
    while buffer[i] <> #0 do
        begin
            write(buffer[i]);
            i := i + 1;
        end;
    writeln;
    if (status = -1) then
        begin
            get_Reply(@buffer);

```

```

        while buffer[i] <> #0 do
            begin
                write(buffer[i]);
                i := i + 1;
            end;
        writeln;
    end
else
    writeln('Power_Up_Clear failed.');
```

End\_API;

end.

### QBASIC EXAMPLE:

```

'$INCLUDE: 'SPINC1QB.BI'
DIM msg%(127)
DIGITAL% = 00
Start.API
status% = Set.Com.Port% (&H3F8, 4, 9600, 0, 8, 1, 1)
IF status% <= 2 THEN
    PRINT "Could not install the COM"
    END
END IF
    status% = Power.Up.Clear% (DIGITAL%)
Call Get.Message
(VARSEG(msg%(0)),VARPTR(msg%(0)))
i% = 0
j% = 1
i% = 1
    WHILE (j% > 0) AND (k% > 0) AND (i% < 128)
        j% = msg%(i%) MOD 256
        k% = msg%(i%) / 256
        IF j% > 0 THEN
            PRINT CHR$(j%);
            IF k% > 0 THEN PRINT CHR$(k%);
        END IF
        i% = i% + 1
    WEND
PRINT
End.API
END
```

## Get\_Reply

**Function:** Get reply from module.

**BASIC Syntax:** Get.Reply (VARSEG(msg%(0)),  
VARPTR(msg%(0)))

**C Syntax:** void far pascal Get\_Reply (char far \*msg);

**Pascal Syntax:** procedure Get\_Reply (msg : pointer);

**Remarks:** **Get\_Reply** sets the 254 bytes pointed to by msg to the value of the API's internal reply buffer. This function retrieves the reply message that is sent from the module. The end of the reply string is marked by a NULL character.

**Return Value:** Nothing.

**Example:** See **Get\_Message**.

## Identify\_Module

**Function:** Identifies the module type.

**BASIC Syntax:** Identify.Module%(preamble%)

**C Syntax:** int far pascal Identify\_Module (unsigned\_char  
preamble);

**Pascal Syntax:** function Identify\_Module (preamble: byte) :  
integer;

**Remarks:** **Identify\_Module** is used to determine what type of interface can be connected to the module. The return value will be stored in the first element of the API's internal data array. The function **Get\_Data (0)** can be used to retrieve this value. The following are defined the defined module types.

01 = SPIO Module (DAPB1)

02 = SDIOB8

**Return Value:** The status is returned.

**See Also:** Identify Module in Chapter 4.

### **Module\_Reset**

**Function:** Reset the module to its power-up condition.

**BASIC Syntax:** Module.Reset%(preamble%)

**C Syntax:** int Module\_Reset (unsigned char preamble)

**Pascal Syntax:** function Module\_Reset (preamble:byte);

**Remarks:** **Module\_Reset** resets the module to its user defined power-up condition.

**Return Value:** The status is returned.

**See Also:** Reset in Chapter 4.

### **Power\_Up-Clear**

**Function:** Reset the power-up-clear expected error.

**Basic Syntax:** Power. Up. Clear% (preamble%)

**C Syntax:** int far pascal Power\_Up\_Clear (unsigned char preamble);

**Pascal Syntax:** function Power\_Up\_Clear (preamble: byte) : integer;

**Remarks:** **Power\_Up\_Clear** resets the **POWER\_UP\_CLEAR\_EXPECTED** error of the module. This error is returned by the module when a function other than **Power\_Up\_Clear** is used after the module is powered-up. A function returning a **POWER\_UP\_CLEAR\_EXPECTED** error will not be executed. Receiving this error indicates that the module has experienced a power failure

and is currently in the user defined power-up condition. After a power failure, proper actions should be taken.

**Return Value:** The status is returned.

**See Also:** Power Up Clear in Chapter 4.

## **Read\_And\_Clear\_Counters**

**Function:** Reads and clears event counter inputs.

**BASIC Syntax:** Read.And.Clear.Counters% (preamble%, lines%)

**C Syntax:** int far pascal Read\_And\_Clear\_Counters (unsigned char preamble, unsigned lines);

**Pascal Syntax:** function Read\_And\_Clear\_Counters (preamble : byte; lines : word) : integer

**Remarks:** **Read\_And\_Clear\_Counters** reads and clears the current count value of the event counter specified in lines. This clears only event counter types 01h and 02h. Event counter types 03h - 0Ah can be read using this command, but will not be cleared (their count values are cleared when the counter is enabled). To specify a line set its corresponding bit position to one. All other I/O lines will not be affected. The data returned will be stored in the corresponding element of the API's internal data array. Use the function **Get\_Data** to return the count value. The maximum count value before the counter overflows is FFFFh. To read the overflow count along with the counter value use **Read\_Counter\_With\_Overflow**.

**Return Value:** The status code is returned.

**See Also:** Read And Clear Counters in Chapter 4.

## Read\_And\_Clear\_Latches

**Function:** Reads and clears latched inputs.

**BASIC Syntax:** Read.And.Clear.Latches%(preamble%, lines%)

**C Syntax:** int far pascal Read\_And\_Clear\_Latches  
(unsigned char preamble, unsigned lines);

**Pascal Syntax:** function Read\_And\_Clear\_Latches (preamble :  
byte; lines : word) : integer;

**Remarks:** **Read\_And\_Clear\_Latches** reads and clears the latched inputs specified in lines. To clear a latched input, set its corresponding bit position to one. All I/O lines that are not latched inputs or that are not available on the module will be ignored. The data will be stored in the first element of the API's internal data array. Use **Get\_Data (0)** to return this value. When a bit position is set to a one the input has latched. When a bit position is set to a zero the proper latch transition has not occurred and the input has not latched. I/O lines that are not configured as latches or that are not available on the module will also have their corresponding bit positions set to a zero.

**Return Value:** The status code is returned.

**See Also:** Read and Clear Latches in Chapter 4.

## Read\_Counters

**Function:** Read the value of the event counters.

**BASIC Syntax:** Read.Counters%(preamble%, lines%)

**C Syntax:** int far pascal Read\_Counters (unsigned char  
preamble, unsigned lines);

**Pascal Syntax:** function Read\_Counters (preamble : byte; lines  
: word) : integer;

**Remarks:** **Read\_Counters** reads the event counter values of the lines specified in lines. If an event counter of type 03h - 0Ah is read during a count cycle a count of zero will be read. To specify a line set its corresponding bit position to a one. The values will be stored in the corresponding element of the API's internal data array. Use **Get\_Data** to return these values. The maximum count value before the counter overflows is FFFFh. To read the counters with overflow use the **Read\_Counters\_With\_Overflow** command.

**Return Value:** The status code is returned.

**See Also:** Read Counters in Chapter 4.

### **Read\_Counters\_With\_Overflow**

**Function:** Read the event counter values and overflow count.

**BASIC Syntax:** Read.Counters.With.Overflow% (preamble%)

**C Syntax:** int far pascal Read\_Counters\_With\_Overflow (unsigned char preamble);

**Pascal Syntax:** function Read\_Counters\_With\_Overflow (preamble : byte) : integer;

**Remarks:** **Read\_Counters\_With\_Overflow** reads the event counters and overflow count. If an event counter of type 03h - 0Ah is read during a count cycle a count of zero will be read. The values returned will be stored in the corresponding elements of the API's internal data array. Use **Get\_Data** to return these values. The maximum event count value before overflow is FFFFh. To determine the overflow count, divide the value returned by 10000h. The maximum for the overflow count is FFh.

**Return Value:** The status code is returned.

**See Also:** Read Counters in Chapter 4.

### Read\_Digital\_Config

**Function:** Read the current digital configuration.

**BASIC Syntax:** Read.Digital.Config% (preamble%)

**C Syntax:** int far pascal Read\_Digital\_Config (unsigned char preamble);

**Pascal Syntax:** function Read\_Digital\_Config (preamble:byte) : integer;

**Remarks:** **Read\_Digital\_Config** reads the digital configuration of the module. The digital I/O line configuration will be in the first element of the API's internal data array. The line state will be in the second element of the array. Use **Get\_Data** to return these values. The bits in the line configuration will be set to one for outputs and zero for input and unavailable lines. The bits in the line state value will be set to one if the digital output line is on, otherwise it will be set to zero. Note that on a PWM output, the line state will sometimes be read as on and at other times as off, because it is constantly changing states. Use **Read\_Output\_Timer\_Values** to determine if a PWM output is on or off.

**Return Value:** The status code is returned.

**See Also:** Read Digital I/O Configuration in Chapter 4.

### Read\_Input\_Level

**Function:** Read the analog input (A/D) values.

**BASIC Syntax:** Read.Input.Level% (preamble%, channels%)

**C Syntax:** int far pascal Read\_Input\_Level (unsigned char preamble, unsigned channels);

**Pascal Syntax:** function Read\_Input\_Level (preamble: byte; channels : word) : integer;

**Remarks:** **Read\_Input\_Level** reads the analog input lines specified in channels. Modules with 8-bit analog-to-digital converters have a normal range of 1000h to 10FFh (for 12-bit converters, 1000h to 1FFFh). The value of a line will be located in the corresponding array element of the API's internal data array. Use the function **Get\_Data** to retrieve these values. The voltage of the line is calculated by taking the returned value and subtracting 1000h from it, then multiply it by the reference voltage, and divide that by 00FFh (for 12-bit converters divide by 0FFFh).

**Return Value:** The status is returned.

**See Also:** Read Analog Input Level in Chapter 4.

## Read\_Latches

**Function:** Reads the state of latched digital inputs.

**BASIC Syntax:** Read.Latches% (preamble%)

**C Syntax:** int far pascal Read\_Latches (unsigned char preamble);

**Pascal Syntax:** function Read\_Latches (preamble: byte) :integer;

**Remarks:** **Read\_Latches** reads the state of all latched digital inputs. The result of this command will be stored in the first element of the API's internal data array. Use **Get\_Data (0)** to return this value. The bits set to one indicate the proper latch transition has occurred for the corresponding input line. The bits set to zero

indicate that the proper latch transition has not occurred for the corresponding input line. I/O lines that are not configured as latched inputs or that are not available on the module will also have their corresponding bit positions set to zero.

**Return Value:** The status code is returned.

**See Also:** Read Latches in Chapter 4.

### **Read\_Module\_Config**

**Function:** Reads a module's configuration.

**BASIC Syntax:** Read.Module.Config% (preamble%)

**C Syntax:** int far pascal Read\_Module\_Config (unsigned char preamble);

**Pascal Syntax:** function Read\_Module\_Config (preamble : byte) : integer;

**Remarks:** **Read\_Module\_Config** reads the module's configuration. The following values will be read: interface module type, communication format, turn around delay, and communication timer. These values will be stored in the API's internal data array. Use **Get\_Data** to return these values. The interface module type will be in the first element of the array, and communication timer value in the second, and so on. To convert communication format to milliseconds, multiply it by 10. See **Set\_Module\_Config** for a description of communication format.

**Return Value:** The status code is returned.

**See Also:** Read/Set SPIO Configuration in Chapter 4.

## Read\_Output\_Level

**Function:** Read the analog output (D/A) levels.

**BASIC Syntax:** Read.Output.Level% (preamble%, channels%)

**C Syntax:** int far pascal Read\_Output\_Level  
(unsigned char preamble, unsigned channels);

**Pascal Syntax:** function Read\_Output\_Level (preamble: byte;  
channels : word) : integer;

**Remarks:** **Read\_Output\_Level** reads the analog output lines specified in channels. Modules with 8-bit analog-to-digital converters have a normal range of 0000h to 00FFh (for 12-bit converters, 0000h to 0FFFh). To specify a line, set its corresponding bit position to one. The value of a line will be stored in corresponding element of the API's internal data array. Use **Get\_Data** to return these values. The voltage of the line is calculated by taking the data value in the array, multiply it by the reference voltage, and divide that by 00FFh (for 12-bit converters divide by 0FFFh).

**Return Value:** The status is returned.

**See Also:** Turn Digital Outputs On in Chapter 4.

## Read\_Power\_Up\_Levels

**Function:** Reads the current analog D/A output power up levels.

**BASIC Syntax:** Read.Power.Up.Levels% (preamble%)

**C Syntax:** int far pascal Read\_Power\_Up\_Levels  
(unsigned char preamble);

**Pascal Syntax:** function Read\_Power\_Up\_Levels (preamble :  
byte) : integer;

**Remarks:** **Read\_Power\_Up\_Levels** reads the power-up output levels of the module. The levels will be stored in the API's internal data array. The most significant line's value will be in the first element. Use **Get\_Data** to retrieve these values. Modules with 8-bit digital-to-analog converters have a normal range of 0000h to 00FFh (for 12-bit converters, 0000h to 0FFFh). The voltage of the line is calculated by taking the data value, multiply it by the reference voltage, then divide that by 00FFh (for 12-bit converters divide by 0FFFh).

**Return Value:** The status code is returned.

**See Also:** Read/Set Analog Output Power Up Levels in Chapter 4.

### **Read\_Power\_Up\_States**

**Function:** Reads the power-up state of the digital output lines.

**BASIC Syntax:** Read.Power.Up.States% (preamble%)

**C Syntax:** int far pascal Read\_Power\_Up\_States (unsigned char preamble);

**Pascal Syntax:** function Read\_Power\_Up\_States (preamble : byte) : integer;

**Remarks:** **Read\_Power\_Up\_States** reads the power-up state of the digital outputs. The data will be stored in the API's internal data array. Use the function **Get\_Data** to retrieve these values. The I/O configuration is stored in the first element of the data array. The bits set to one are configured as outputs and the bits set to zero are configured as inputs. The second element of the array contains the current state of the output lines. A bit set to one represents that its

corresponding line will be "ON" at power-up and a bit set to zero will be "OFF" at power-up.

**Return Value:** The status code will be returned.

**See Also:** Read/Set Digital Power Up States in Chapter 4.

### **Read\_States**

**Function:** Reads the current state of all its digital lines.

**BASIC Syntax:** Read.States% (preamble%)

**C Syntax:** int far pascal Read\_States (unsigned char preamble);

**Pascal Syntax:** function Read\_States (preamble : byte) integer;

**Remarks:** **Read\_States** read the state of the digital I/O lines. The values of the lines will be stored in the first element of the API's internal data array. Use the function **Get\_Data (0)** to retrieve this value. A bit set to one means that the corresponding digital I/O line is "ON". A bit set to zero means that the digital I/O line is "OFF" or not available.

**Return Value:** The status code is returned.

**See Also:** Read Digital I/O States in Chapter 4.

### **Read\_Output\_Timer\_Values**

**Function:** Reads digital output timer values.

**BASIC Syntax:** Read.Output.Timer.Values% (preamble%, lines%)

**C Syntax:** int far pascal Read\_Output\_Timer\_Values (unsigned char preamble, unsigned lines);

**Pascal Syntax:** function Read\_Output\_Timer\_Values (preamble: byte; lines : word) : integer;

**Remarks:** **Read\_Output\_Timer\_Values** reads the digital output timer values of the lines specified in lines. The data will be stored in the corresponding element of the API's internal data array. Use the function **Get\_Data** to return these values. If the I/O line is configured as normal output, the data value will be zero. If the I/O line is a time delayed output or a PWM output the data can be interpreted as follows:

1. If the data value is 10000h or greater, the timer is active or the PWM is enabled. The time delay value or the pulse width value will be the data value minus 10000h.
2. If the return value is less than 10000h then the timer is inactive or the PWM is disabled. The timedelay value for timers or the pulse width value will be the data value.

**Return Value:** The status code is returned.

**See Also:** Read/Set Output Timer Values in Chapter 4.

### Read\_Digital\_Types

**Function:** Reads the digital I/O types.

**BASIC Syntax:** Read.Digital.Types% (preamble%)

**C Syntax:** int far pascal Read\_Digital\_Types (unsigned char preamble);

**Pascal Syntax:** function Read\_Digital\_Types (preamble : byte) : integer;

**Remarks:** **Read\_Digital\_Types** reads the digital I/O types of the module. The data will be stored in the API's internal data array. The most significant line will be stored in the first element of the array. Use the function **Get\_Data** to access the data.

**Return Value:** The status code is returned.

**See Also:** **Set\_Output\_Timer\_Values**, **Set\_Digital\_Types**, and **Read\_Output\_Timer\_Values** in this chapter and Read/Set Digital I/O Type in Chapter 4.

### Retrigger\_Time\_Delay

**Function:** Resets a time delayed output's counter.

**BASIC Syntax:** Retrigger.Time.Delay% (preamble%, lines%)

**C Syntax:** int far pascal Retrigger\_Time\_Delay (unsigned char preamble, unsigned lines);

**Pascal Syntax:** function Retrigger\_Time\_Delay (preamble : byte; lines : word) : integer;

**Remarks:** **Retrigger\_Time\_Delay** resets the active time delays specified in lines to the beginning of its time cycle. This command is used to extend the original time value. Time delays that are inactive (outputs that have completed their timed cycle) will not be affected. Set the bits in lines to one to select the time delayed outputs to retrigger.

**Return Value:** The status code is returned.

**See Also:** Retrigger Time Delay in Chapter 4.

### Set\_Analog\_Outputs

**Function:** Set the level of analog outputs.

**BASIC Syntax:** Set.Analog.Outputs% (preamble%, channels%, value%)

**C Syntax:** int Set\_Analog\_Outputs (unsigned char preamble, unsigned channels, unsigned value);

**Pascal Syntax:** function Set\_Analog\_Outputs (preamble : byte; channels, value word) : integer;

**Remarks:** **Set\_Analog\_Outputs** sets the level of all analog outputs specified in channels to value. To specify an analog output line set its corresponding bit in channels to one. value has a range of 0000h to 00FFh for 8-bit digital-to-analog converters (0000h to 0FFFh for 12-bit converters). Any specified line that is not available will be ignored. To calculate the output voltage, multiply the value by the analog output reference voltage and divide that by 00FFh (for 12-bit converters divide by 0FFFh).

**Return Value:** The status code is returned.

**See Also:** Set Analog Output Level in Chapter 4.

### Set\_API\_Timeout

**Function:** Sets the API time out duration.

**BASIC Syntax:** Set.API.Timeout (value%)

**C Syntax:** void Set\_API\_Timeout (unsigned value);

**Pascal Syntax:** procedure Set\_API\_Timeout (value: word);

**Remarks:** **Set\_API\_Timeout** sets the amount of time that the API will wait for the start of the reply message from the module. This is initially set to 1500 milliseconds. This is done to accommodate the maximum selectable turn-around delay. You can change this value so your program returns faster when an error occurs. If you change this value, you must take into account the turn-around delay value. Setting this value too low will result in an **REPLY\_TIME\_OUT** error. value is in milliseconds.

**Returns:** Nothing.

**See Also:** Nothing.

## Set\_Com\_Port

**Function:** Sets the communication port's configuration.

**BASIC Syntax:** Set.Com.Port% (adr%, irq%, baud&, parity%, data.bits, stop.bits%, mode%)

**C Syntax:** int Set\_Com\_Port (unsigned adr, un-signed irq, long baud, unsigned parity, unsigned data\_bits, unsigned stop\_bits, unsigned mode);

**Pascal Syntax:** function Set\_Com\_Port (adr : word; irq : word; baud : longint; parity, data\_bits, stop\_bits, mode : word) : integer;

**Remarks:** Sets the serial communication port's configuration. This must be the same as the configuration of the module you intend to communication with.

adr = address of the serial port  
irq = interrupt request (IRQ) number of the serial port.  
baud = baud rate (1 – 115,200).  
parity = parity (0 = none, 1 = odd, 2 = even).  
data\_bits = number of data bits (7 or 8).  
stop\_bits = (always 1).  
mode = (0 = RS-232, 1 = RS-485).

The standard addresses and interrupt request numbers are given in the following table. Some serial cards, such as B&B Electronics' serial cards, have selectable port addresses and interrupt request numbers. If you are using a serial card with a selectable port address or interrupt request number, pass them in adr and irq instead of the standard values.

COM Port	Address	IRQ
----------	---------	-----

COM1	03F8h	4
COM2	02F8h	3
COM3	03E8h	4
COM4	02E8h	3

**Returns:** The communication driver status.  
0 = No UART (universal asynchronous receiver/transmitter) at the specified address.  
1 = Bad interrupt request number specified.  
2 = Unsupported UART found. (The communications driver supports the following UARTS: 8250A, 16450, 16550, and 16550AF. The driver does not support the 8250 or 8250B.)  
Any other number means that the communication driver was installed properly.

**See Also:** Nothing.

### Set\_Communication\_Mode

**Function:** Set the module's communication mode. You must be in "SETUP MODE" to execute this command.

**BASIC Syntax:** Set.Communication.Mode% (preamble%, value%)

**C Syntax:** int far pascal Set\_Communication\_Mode (unsigned char preamble, unsigned char value);

**Pascal Syntax:** function Set\_Communication\_Mode (preamble, value : byte) : integer;

**Remarks:** **Set\_Communication\_Mode** sets the communication mode to value. The value can be either 0 for two-step or 1 for four-step. Any value besides 1 will set the communication mode to two-step. This command only sets the module's communication mode. **Set\_Protocol** must be used to set the host's communication

mode before attempting to communicate to the module.

**Return Value:** The status code is returned.

**See Also:** Set Communication Mode in Chapter 4.

## Set\_Digital\_Types

**Function:** Sets the digital I/O line type. You must be in "SETUP MODE" to execute this command.

**BASIC Syntax:** Set.Digital.Types% (preamble%, lines%, value%)

**C Syntax:** int far pascal Set\_Digital\_Types (unsigned char preamble, unsigned lines, unsigned char value);

**Pascal Syntax:** function Set\_Digital\_Types (preamble: byte; lines : word; value byte); integer;

**Remarks:** **Set\_Digital\_Types** sets the digital I/O line type of the lines specified in lines. Input types are: normal, latched, and event counter. Latches and counters types are selected based on event polarity. Output types are: normal, time delayed, and pulse width modulated (PWM). The valid types are described below.

### INPUT TYPES

Type	Description
------	-------------

00h	Normal input.
-----	---------------

01h	Start/Stop command controlled event counter, the count input is falling-edge triggered.
-----	---

02h	Start/Stop command controlled event counter, count rising edges.
-----	--

03h	Rising edge controlled event counter, count falling edges. The count begins when a rising edge appears on the count control line and stops on the next rising edge.
-----	---

- 04h Rising edge controlled event counter, count rising edges. The count begins when a rising edge appears on the count control line and stops on the next rising edge.
- 05h Falling edge controlled event counter, count falling edges. The count begins when a falling edge appears on the count control line and stops on the next falling edge.
- 06h Falling edge controlled event counter, count rising edges. The count begins when a falling edge appears on the count control line and stops on the next falling edge.
- 07h Rising pulse-width controlled event counter, count falling edges. The count begins on the rising edge of the pulse and stops at the falling edge.
- 08h Rising pulse-width controlled event counter, count rising edges. The count begins on the rising edge of the pulse and stops at the falling edge.
- 09h Falling pulse-width controlled event counter, count falling edges. The count begins on the falling edge of the pulse and stops at the rising edge.
- 0Ah Falling pulse-width controlled event counter, count rising edges. The count begins on the falling edge and stops on the rising edge.
- 0Bh Rising edge triggered latched input.
- 0Ch Falling edge triggered latched input.

## **READ-ONLY TYPES**

### Type Description

- 7Eh Rising edge triggered start/stop event counter control line. Automatically configured when types 04h, 06h, 08h and 0Ah are configured.
- 7Fh Falling edge triggered start/stop event counter control line. Automatically

configured when types 03h, 05h, 07h, and 09h are configured.

## OUTPUT TYPES

### Type Description

80h	Normal output.
81h	Time delayed output. If the output is OFF, when turning the output ON, turn it ON for the specified time, then turn it OFF again.
82h	Time delayed output. If the output is OFF, when turning an output ON, stay OFF for the specified time before turning it ON.
83h	Time delayed output. If the output is ON, when turning an output OFF, turn it OFF for the specified time, then turn it back ON.
84h	Time delayed output. If the output is ON, when turning an output OFF, stay ON to the specified time before turning it OFF.
85h	High resolution pulse width modulated output.
86h	High speed pulse width modulated output.

**Return Value:** The status code is returned.

**See Also:** The **Set\_Output\_Timer\_Values** function in this chapter and Read/Set Digital I/O Type in Chapter 4.

## Set\_Indiv\_Levels

**Function:** Set the analog output levels.

**BASIC Syntax:** Set.Indiv.Levels% (preamble%)

**C Syntax:** int far pascal Set\_Indiv\_Levels (unsigned char preamble);

**Pascal Syntax:** function Set\_Indiv\_Levels (preamble: byte) : integer;

**Remarks:** **Set\_Indiv\_Levels** sets the analog output lines of the specified lines.

**Return Value:** The status code is returned.

**See Also:** **Specify\_Indiv\_Levels** and **Unspecify\_Indiv\_Levels** in this chapter and Set Individual Analog Output Levels in Chapter 4.

### Set\_Latch\_Edges

**Function:** Sets the polarity of the latched input lines. You must be in "SETUP MODE" to execute this command.

**BASIC Syntax:** Set.Latch.Edges% (preamble%, lines%, value%)

**C Syntax:** int far pascal Set\_Latch\_Edges (unsigned char preamble, unsigned lines, unsigned value);

**Pascal Syntax:** function Set\_Latch\_Edges (preamble: byte; lines: word; value: byte): integer;

**Remarks:** **Set\_Latch\_Edges** sets the polarity of the latched and normal input lines to rising or falling edge triggered. The meaning of the number passed in value is described below.

- 0 - Set all 16 lines as rising edge triggered latches.
- 1 - Set lower 4 lines as specified.
- 2 - Set lower 8 lines as specified.
- 3 - Set lower 12 lines as specified.
- 4 - Set all 16 lines as specified.

Set the bit in lines to one to configure the corresponding input to rising edge triggered. Set the bit to zero to configure the

corresponding line to falling edge triggered. If a specified input line is not configured as a normal or latched input, an **INVALID\_VALUE** error will be returned.

- If you pass a 0005h in lines and a 1 in value, lines 0 and 2 will be configured as rising edge triggered, lines 1 and 3 will be configured as falling edge triggered, and all other lines will not be changed.

- If you pass a 0005h in lines and a 2 in value, lines 0 and 2 will be configured as rising edge triggered, lines 1, 3, 4, 5, 6, and 7 will be configured as falling edge triggered, and all other lines will not be changed.

**Return Value:** The status code is returned.

**See Also:** Set Latch Edges in Chapter 4.

## Set\_Module\_Config

**Function:** Sets a module's configuration. You must be in "SETUP MODE" to execute this command.

**BASIC Syntax:** Set.Module.Config% (preamble%, comm%)

**C Syntax:** int far pascal Set\_Module\_Config (unsigned char preamble, unsigned char comm);

**Pascal Syntax:** function Set\_Module\_Config (preamble: byte; comm : word) : integer;

**Remarks:** **Set\_Module\_Config** is used to set the configuration of the module. comm is the new communication format. It is used to set the baud rate, the number of data bits (dbit), the parity, and the communication mode (proto). The following describes comm at the bit level.

proto:        0 - two step communication  
              1 - four step communication

parity:       00 - no parity

01 - even parity  
10 - no parity  
11 - odd parity

dbit : 0 - 8 data bits  
1 - 7 data bits

baud : 0000 - 300 baud  
0001 - 600 baud  
0010 - 1200 baud  
0011 - 2400 baud  
0100 - 4800 baud  
0101 - 9600 baud

**Return Value:** The status code will be returned.

**See Also:** Read/Set SPIO Configuration in Chapter 4.

### Set\_Off\_To\_On\_Latches

**Function:** Sets the polarity of the of I/O lines to latch on the falling edge of the input signal. You must be in "SETUP MODE" to execute this command.

**BASIC Syntax:** Set.Off.To.On.Latches% (preamble%, lines%)

**C Syntax:** int Set\_Off\_To\_On\_Latches (unsigned char preamble, unsigned lines);

**Pascal Syntax:** function Set\_Off\_To\_On\_Latches (preamble : byte; lines : word) : integer;

**Remarks:** **Set\_Off\_To\_On\_Latches** sets the normal and latch input line specified in lines to falling edge triggered. To specify the line to change, set its corresponding bit in lines to a one. All bits set to zero will cause no change in that line's configuration. If a specified input line is not configured as a normal or latched input, an **INVALID\_VALUE** error will be returned.

**Return Value:** The status code is returned.

**See Also:** Set Off To On Latches in Chapter 4.

### Set\_On\_To\_Off\_Latches

**Function:** Sets the polarity of the of I/O lines to latch on the rising edge of the input signal. You must be in "SETUP MODE" to execute this command.

**BASIC Syntax:** Set.On.To.Off.Latches% (preamble%, lines%)  
**C Syntax:** int Set\_On\_To\_Off\_Latches  
(unsigned char preamble, unsigned lines);

**Pascal Syntax:** function Set\_On\_To\_Off\_Latches (preamble : byte; lines : word) : integer;

**Remarks:** **Set\_On\_To\_Off\_Latches** sets normal and latched inputs specified in lines to rising edge triggered. To specify the line to change, set its corresponding bit in lines to a one. All bits set to zero will cause no change in that line's configuration. If a specified input line is not configured as a normal or latched input, an **INVALID\_VALUE** error will be returned.

**Return Value:** The status code is returned.

**See Also:** Set Off To ON Latches in Chapter 4.

### Set\_Output\_State

**Function:** Sets the output states ON or OFF.

**BASIC Syntax:** Set.Output.State% (preamble%, lines%, value%)

**C Syntax:** int far pascal Set\_Output\_State (unsigned char preamble, unsigned lines, unsigned char value);

**Pascal Syntax:** function Set\_Output\_State (preamble : byte; lines : word; value : byte) : integer;

**Remarks:** **Set\_Output\_State** sets the states of the output lines to ON or OFF. The meaning of the number passed in value is described below.

- 0 - Turn all 16 lines to ON.
- 1 - Set lower 4 lines as specified.
- 2 - Set lower 8 lines as specified.
- 3 - Set lower 12 lines as specified.
- 4 - Set all 16 lines as specified.

To specify a line to turn ON, set its corresponding bit to one. To specify a line to turn OFF set its bit position to zero. All lines not configured as outputs will be ignored.

- If you pass a 0005h in lines and a 1 in value, lines 0 and 2 will be turned ON, lines 1 and 3 will be turned OFF, and all other lines will be unchanged.
- If you pass a 0005h in lines and a 2 in value, lines 0 and 2 will be turned ON, lines 1, 3, 4, 5, 6, and 7 will be turned OFF, and all other lines will not be changed.

**Return Value:** The status is returns.

**See Also:** Set Digital Outputs On or Off in Chapter 4.

## Set\_Output\_Timer\_Values

**Function:** Sets digital output timer values. When setting values while the unit is in "SETUP MODE," the values are stored in EEPROM. When setting values while not in "SETUP MODE," the values will be stored in RAM. Values that are stored in RAM will be lost when the unit is reset.

**BASIC Syntax:** Set.Output.Timer.Vaules% (preamble%, lines%, value%)

**C Syntax:** int far pascal Set\_Output\_Timer\_Values (unsigned char preamble, unsigned lines, unsigned value);

**Pascal Syntax:** function Set\_Output\_Timer\_Values (preamble : byte; lines, value : word) : integer;

**Remarks:** **Set\_Output\_Timer\_Values** sets the digital output timer values of the lines specified in lines of the module. To specify an output line set its corresponding bit position to one. All lines with their bit positions set to zero will be left unchanged. The value will be only loaded into specified time delayed or PWM outputs. Any other specified output or input types will be ignored. Unspecified time delayed or PWM outputs will be left unchanged.

**Note:** Maximum values for the PWM.  
High Resolution Mode: FFFFh

High Speed Mode: 00FFh  
(High byte will be ignored if specified)

**Return Value:** The status code is returned.

**See Also:** Read/Set Output Timer Values in Chapter 4.

## Set\_Power\_Up\_Levels

**Function:** Sets the current analog D/A output power up levels. You must be in "SETUP MODE" to execute this command.

**BASIC Syntax:** Set.Power.Up.Levels% (preamble%, channels%, value%)

**C Syntax:** int far pascal Set\_Power\_Up\_Levels  
(unsigned char preamble, unsigned channels, unsigned value);

**Pascal Syntax:** function Set\_Power\_Up\_Levels (preamble : byte; channels, value : word) : integer;

**Remarks:** **Set\_Power\_Up\_Levels** sets the analog output power-up levels specified in channels to value. To specify an output line, set the corresponding bit in channels to one. For a 12-bit digital to 12-bit analog converters, value has a range of 0000h to 0FFFh (0000h to 00FFh for 8-bit converters). To convert value to a voltage, multiply it by the analog output reference voltage and divide it by 0FFFh (00FFh for 8-bit converters). Any line specified that is not available will be ignored. If you specify a value greater than 00FFh with an 8-bit converter, the most significant digits will be ignored.

**Return Value:** The status code is returned.

**See Also:** Read/Set Analog Power Up Levels in Chapter 4.

### Set\_Power\_Up\_States

**Function:** Sets the power-up output states for digital outputs. You must be in "SETUP MODE" to execute this command.

**BASIC Syntax:** Set.Power.Up.States% (preamble%, lines%, value%)

**C Syntax:** int far pascal Set\_Power\_Up\_States (unsigned char preamble, unsigned lines, unsigned char value);

**Pascal Syntax:** function Set\_Power\_Up\_States (preamble : byte; lines : word; value : byte) : integer;

**Remarks:** **Set\_Power\_Up\_States** sets the power-up states of the lines specified in lines to lines. To specify a line, set its corresponding bit to one. Set value to one to turn the specified lines ON at power-up. Set value to zero to turn the specified lines OFF at power-up. Any unavailable line will be ignored.

**Return Value:** The status code is returned.

**See Also:** Read/Set Digital Power Up States in Chapter 4.

### Set\_Protocol

**Function:** Sets the API's protocol.

**BASIC Syntax:** Set.Protocol (value%)

**C Syntax:** void far pascal Set\_Protocol (unsigned\_char value);

**Pascal Syntax:** function Set\_Protocol (value : byte) : integer;

**Remarks:** **Set\_Protocol** sets the API's protocol. value = 2 for two-step or = 4 for four-step. This should be done after **Set\_Communication\_Mode**.

**Return Value:** Nothing.

**See Also:** **Set\_Communication\_Mode**

### Set\_Retries

**Function:** Sets the number of times to retry sending a command message if an error occurs.

**BASIC Syntax:** Set.Retries (value%)

**C Syntax:** void far pascal Set\_Retries (unsigned value);

**Pascal Syntax:** procedure Set\_Retries (value : word);

**Remarks:** **Set\_Retries** sets the number of times the API will try to send out a command message if it finds an error. Two errors, **POWER\_UP\_CLEAR\_EXPECTED** and **COMMUNICATION\_TIMEOUT**, return without retrying. These errors would be cleared by resending the command, since they indicate special conditions that **MUST** be acted on by the program, they return immediately.

**Return Value:** None.

## Set\_Time\_Delay

**Function:** Sets output lines as time delayed outputs or as normal outputs. You must be in "SETUP MODE" to execute this command.

**BASIC Syntax:** Set.Time.Delay% (preamble%, lines%, modifier%, value%)

**C Syntax:** int far pascal Set\_Time\_Delay (unsigned char preamble, unsigned lines, unsigned char modifier, unsigned value);

**Pascal Syntax:** function Set\_Time\_Delay (preamble: byte; lines: word; modifier: byte; value: word): integer;

**Remarks:** **Set\_Time\_Delay** configures the output lines specified in lines as time delayed or normal outputs. To specify an output line set its corresponding bit position to one. Setting a bit position to zero will cause no change in that line's configuration. If the specified line is not configured as a normal or time delayed output, it will be ignored.

**INVALID\_VALUE** error will occur if the specified output line is currently configured as a PWM. The power-up state for each output specified will be set to the off state. modifier specifies the type of output to be configured. Valid modifiers are shown below.

<u>modifier</u>	<u>Type</u>	<u>Description</u>
G	80h	Normal output.
H	81h	Time delayed output. When turning the output ON, turn it ON for the specified time, then turn it OFF again.
I	82h	Time delayed output. When turning an output ON, stay OFF for the specified

J	83h	time before turning it ON. Time delayed output. When turning an output OFF, turn it OFF for the specified time, then turn it back ON.
K	84h	Time delayed output. When turning an output OFF, stay ON to the specified time before turning it OFF. <u>value</u> has a range of 0000h to FFFF. Any unspecified time delayed output will be left unchanged. To determine the actual time delay in milliseconds, multiply <u>value</u> by 10.

**Return Value:** The status code is returned.

**See Also:** Set Time Delay in Chapter 4.

## Specify\_Indiv\_Levels

- Function:** Adds a level to the levels array.
- BASIC Syntax:** Specify.Indiv.Levels (channels%,  
value%)
- C Syntax:** void far pascal Specify\_Indiv\_Levels  
(unsigned channels, unsigned value);
- Pascal Syntax:** procedure Specify\_Indiv\_Levels  
(channels, value: word);
- Remarks:** **Specify\_Indiv\_Levels** sets the elements of the internal individual levels array of the lines specified in channels to value. This function does not change the module's levels. It only sets up a table of values that will be set when you use the **Set\_Indiv\_Levels** function. This differs from **Set\_Output\_Levels** in that this function sets the specified lines to different levels, rather than to the same level.
- Return Value:** Nothing.
- See Also:** **Set\_Indiv\_Levels** and **Unspecify\_Indiv\_Levels** in this chapter.

## Start\_API

- Function:** Initialize the API.
- BASIC Syntax:** Start.API
- C Syntax:** void far pascal Start\_API (void);

**Pascal Syntax:** procedure Start\_API;

**Remarks:** **Start\_API** initializes the API's internal variables. This must be done before any of the API's commands can be executed. After you start the API, make sure that you use **End\_API** before exiting the program.

**Returns:** Nothing.

**See Also:** **End\_API**.

## Start\_Counters

**Function:** Starts the software controlled event counters.

**BASIC Syntax:** Start.Counters% (preamble%, lines%)

**C Syntax:** int far pascal Start\_Counters  
(unsigned char preamble, unsigned lines);

**Pascal Syntax:** function Start\_Counters (preamble:  
byte; lines : word): integer;

**Remarks:** **Start\_Counters** starts the software event counters or enables the I/O controlled counters, specified in lines. To specify an event counter line, set its corresponding bit to one. All lines that are not configured as event counter inputs or that are unavailable will be ignored.

**Return Value:** The status code is returned.

**See Also:** Start Counters in Chapter 4.

## Start\_Stop\_Counters

- Function:** Starts and stops the software controlled event counters.
- BASIC Syntax:** Start.Stop.Counters% (preamble%, lines%)
- C Syntax:** int far pascal Start\_Stop\_Counters (unsigned char preamble, lines);
- Pascal Syntax:** function Start\_Stop\_Counters (preamble : byte; lines : word) : integer;
- Remarks:** **Start\_Stop\_Counters** starts/stops the software controlled event counters or enables the I/O controlled counters, specified in lines. To specify an event counter to start, set its corresponding bit to one. To specify an event counter to stop, set its bit to zero. An I/O controlled counter can only be stopped via its control line. Lines that are not counter inputs or are unavailable on the module will be ignored. If a valid counter has not been specified, an **Invalid\_Value** error will occur.
- Return Value:** The status code is returned.
- See Also:** Start/Stop Counters in Chapter 4.

## Stop\_Counters

- Function:** Stops the software controlled event counters.
- BASIC Syntax:** Stop.Counters% (preamble%, lines%)
- C Syntax:** int far pascal Stop\_Counters (unsigned char preamble, unsigned lines);

**Pascal Syntax:**     function Stop\_Counters (preamble :  
byte; lines : word) : integer;

**Remarks:**           **Stop\_Counters** stops the software event counters, specified in lines. To stop a counter set its corresponding bit position to one. An I/O controlled counter can only be stopped via its control line. Lines that are not counter inputs or are unavailable will be ignored.

**Return Value:**       The status code is returned.

**See Also:**            Stop Counters in Chapter 4.

## Turn\_Around\_Delay

**Function:**           Sets the time period before the module transmits the reply message back to the host. You must be in "SETUP MODE" to execute this command.

**BASIC Syntax:**       Turn.Around.Delay% (preamble%, value%)

**C Syntax:**           int far pascal Turn\_Around\_Delay  
(unsigned char preamble, unsigned char value);

**Pascal Syntax:**      function Turn\_Around\_Delay (preamble,  
value : byte) : integer;

**Remarks:**           **Turn\_Around\_Delay** sets the time period that the module will wait before it transmits the reply back to the host to value.

value can be one of the following values:

0 = Delay Disabled

1 = 10 ms

2 = 100 ms

3 = 500 ms  
4 = 1000 ms

**Return Value:** The status code is returned.

**See Also:** Turn-Around Delay in Chapter 4.

## Turn\_Output\_Off

**Function:** Turns digital output lines OFF.

**BASIC Syntax:** Turn.Output.Off% (preamble%, lines%)

**C Syntax:** int far pascal Turn\_Output\_Off (unsigned char preamble, unsigned lines);

**Pascal Syntax:** function Turn\_Output\_Off (preamble: byte; lines : word) : integer;

**Remarks:** **Turn\_Output\_Off** turns the digital outputs lines, specified in lines, "OFF". To specify a line to turn "OFF" set its corresponding bit position to one. The state of all other lines will be left unchanged. Lines that have been defined as inputs and unavailable lines will be ignored.

**Return Value:** The status code will be returned.

**See Also:** Turn Digital Output Off in Chapter 4.

## Turn\_Output\_On

**Function:** Turns digital output lines ON.

**BASIC Syntax:** Turn.Output.On% (preamble%, lines%)

**C Syntax:** int far pascal Turn\_Output\_On  
(unsigned\_char preamble, unsigned lines);

**Pascal Syntax:** function Turn\_Output\_On (preamble:  
byte; lines : word) : integer;

**Remarks:** **Turn\_Output\_On** turns the digital  
output lines, specified in lines, "ON". To  
specify a line to turn "ON", set its corre-  
sponding bit to one. The state of all other  
lines will be left unchanged. Lines that  
have been defined as inputs or that are  
unavailable will be ignored.

**Return Value:** The status code will be returned.

**See Also:** Turn Digital Output On in Chapter 4.

## Unspecify\_Indiv\_Levels

**Function:** Removes a level from the levels array.

**BASIC Syntax:** Unspecify.Indiv.Levels (channels%)

**C Syntax:** void far pascal Unspecify\_Indiv\_Levels  
(unsigned channels);

**Pascal Syntax:** procedure Unspecify\_Indiv\_Levels  
(channels : word);

**Remarks:** **Unspecify\_Indiv\_Levels** removes a  
level specified with  
**Specify\_Invdiv\_Level** from the  
internal individual level array.

**Return Value:** Nothing.

**See Also:** **Specify\_Indiv\_Levels** and  
**Set\_Indiv\_Levels** in this chapter.

**DEC                      HEX                      BIN                      ASCII**

## **APPENDIX A: ASCII CHARACTER CODES**

<b>DEC</b>	<b>HEX</b>	<b>BIN</b>	<b>ASCII</b>
0	00	00000000	NUL
1	01	00000001	SOH
2	02	00000010	STX
3	03	00000011	ETX
4	04	00000100	EOT
5	05	00000101	ENQ
6	06	00000110	ACK
7	07	00000111	BEL
8	08	00001000	BS
9	09	00001001	HT
10	0A	00001010	LF
11	0B	00001011	VT
12	0C	00001100	FF
13	0D	00001101	CR
14	0E	00001110	SO
15	0F	00001111	SI
16	10	00010000	DLE
17	11	00010001	DC1
18	12	00010010	DC2
19	13	00010011	DC3
20	14	00010100	DC4
21	15	00010101	NAK
22	16	00010110	SYN
23	17	00010111	ETB
24	18	00011000	CAN
25	19	00011001	EM
26	1A	00011010	SUB
27	1B	00011011	ESC
28	1C	00011100	FS
29	1D	00011101	GS
30	1E	00011110	RS
31	1F	00011111	US
32	20	00100000	SP
33	21	00100001	!
34	22	00100010	"

DEC	HEX	BIN	ASCII
35	23	00100011	#
36	24	00100100	\$
37	25	00100101	%
38	26	00100110	&
39	27	00100111	'
40	28	00101000	(
41	29	00101001	)
42	2A	00101010	*
43	2B	00101011	+
44	2C	00101100	,
45	2D	00101101	-
46	2E	00101110	.
47	2F	00101111	/
48	30	00110000	0
49	31	00110001	1
50	32	00110010	2
51	33	00110011	3
52	34	00110100	4
53	35	00110101	5
54	36	00110110	6
55	37	00110111	7
56	38	00111000	8
57	39	00111001	9
58	3A	00111010	:
59	3B	00111011	;
60	3C	00111100	<
61	3D	00111101	=
62	3E	00111110	>
63	3F	00111111	?
64	40	01000000	@
65	41	01000001	A
66	42	01000010	B
67	43	01000011	C
68	44	01000100	D
69	45	01000101	E
70	46	01000110	F
71	47	01000111	G
72	48	01001000	H
73	49	01001001	I
74	4A	01001010	J
75	4B	01001011	K

DEC	HEX	BIN	ASCII
76	4C	01001100	L
77	4D	01001101	M
78	4E	01001110	N
79	4F	01001111	O
80	50	01010000	P
81	51	01010001	Q
82	52	01010010	R
83	53	01010011	S
84	54	01010100	T
85	55	01010101	U
86	56	01010110	V
87	57	01010111	W
88	58	01011000	X
89	59	01011001	Y
90	5A	01011010	Z
91	5B	01011011	[
92	5C	01011100	\
93	5D	01011101	]
94	5E	01011110	^
95	5F	01011111	_
96	60	01100000	`
97	61	01100001	a
98	62	01100010	b
99	63	01100011	c
100	64	01100100	d
101	65	01100101	e
102	66	01100110	f
103	67	01100111	g
104	68	01101000	h
105	69	01101001	i
106	6A	01101010	j
107	6B	01101011	k
108	6C	01101100	l
109	6D	01101101	m
110	6E	01101110	n
111	6F	01101111	o
112	70	01110000	p
113	71	01110001	q
114	72	01110010	r
115	73	01110011	s
116	74	01110100	t

DEC	HEX	BIN	ASCII
117	75	01110101	u
118	76	01110110	v
119	77	01110111	w
120	78	01111000	x
121	79	01111001	y
122	7A	01111010	z
123	7B	01111011	{
124	7C	01111100	
125	7D	01111101	}
126	7E	01111110	~
127	7F	01111111	DEL
128	80	10000000	
129	81	10000001	
130	82	10000010	
131	83	10000011	
132	84	10000100	
133	85	10000101	
134	86	10000110	
135	87	10000111	
136	88	10001000	
137	89	10001001	
138	8A	10001010	
139	8B	10001011	
140	8C	10001100	
141	8D	10001101	
142	8E	10001110	
143	8F	10001111	
144	90	10010000	
145	91	10010001	
146	92	10010010	
147	93	10010011	
148	94	10010100	
149	95	10010101	
150	96	10010110	
151	97	10010111	
152	98	10011000	
153	99	10011001	
154	9A	10011010	
155	9B	10011011	
156	9C	10011100	
157	9D	10011101	

DEC	HEX	BIN	ASCII
158	9E	10011110	
159	9F	10011111	
160	A0	10100000	
161	A1	10100001	
162	A2	10100010	
163	A3	10100011	
164	A4	10100100	
165	A5	10100101	
166	A6	10100110	
167	A7	10100111	
168	A8	10101000	
169	A9	10101001	
170	AA	10101010	
171	AB	10101011	
172	AC	10101100	
173	AD	10101101	
174	AE	10101110	
175	AF	10101111	
176	B0	10110000	
177	B1	10110001	
178	B2	10110010	
179	B3	10110011	
180	B4	10110100	
181	B5	10110101	
182	B6	10110110	
183	B7	10110111	
184	B8	10111000	
185	B9	10111001	
186	BA	10111010	
187	BB	10111011	
188	BC	10111100	
189	BD	10111101	
190	BE	10111110	
191	BF	10111111	
192	C0	11000000	
193	C1	11000001	
194	C2	11000010	
195	C3	11000011	
196	C4	11000100	
197	C5	11000101	
198	C6	11000110	

DEC	HEX	BIN	ASCII
199	C7	11000111	
200	C8	11001000	
201	C9	11001001	
202	CA	11001010	
203	CB	11001011	
204	CC	11001100	
205	CD	11001101	
206	CE	11001110	
207	CF	11001111	
208	D0	11010000	
209	D1	11010001	
210	D2	11010010	
211	D3	11010011	
212	D4	11010100	
213	D5	11010101	
214	D6	11010110	
215	D7	11010111	
216	D8	11011000	
217	D9	11011001	
218	DA	11011010	
219	DB	11011011	
220	DC	11011100	
221	DD	11011101	
222	DE	11011110	
223	DF	11011111	
224	E0	11100000	
225	E1	11100001	
226	E2	11100010	
227	E3	11100011	
228	E4	11100100	
229	E5	11100101	
230	E6	11100110	
231	E7	11100111	
232	E8	11101000	
233	E9	11101001	
234	EA	11101010	
235	EB	11101011	
236	EC	11101100	
237	ED	11101101	
238	EE	11101110	
239	EF	11101111	

<b>DEC</b>	<b>HEX</b>	<b>BIN</b>	<b>ASCII</b>
240	F0	11110000	
241	F1	11110001	
242	F2	11110010	
243	F3	11110011	
244	F4	11110100	
245	F5	11110101	
246	F6	11110110	
247	F7	11110111	
248	F8	11111000	
249	F9	11111001	
250	FA	11111010	
251	FB	11111011	
252	FC	11111100	
253	FD	11111101	
254	FE	11111110	
255	FF	11111111	



### **Example Dec to Hex conversion :**

A decimal number of 4348 is equal to :

#### **First Hex digit:**

$4348/4096 > 1$  so the first digit is a 1. There is a remainder of 252.

#### **Second Hex digit:**

$252/2840 < 1$  so the second digit is a 0. There is a remainder of 252.

#### **Third Hex digit:**

$252/16 > 15$  so the third digit is a F. There is a remainder of 12.

#### **Fourth Hex digit:**

$12 = C$  so the fourth digit is a C.

## APPENDIX C: I/O Port Pin Functions

TABLE C.1

DB25S PIN#	DAPB1 FUNCTION	SDIOB8 FUNCTION
1	GND	GND
2	NC	OE
3	I/O 5	IN5
4	I/O 7	CLOCK
5	I/O 1	IN1
6	I/O 3	WTOGGLE
7	AVSS	NC
8	DAVREF	NC
9	D/A 1	NC
10	A/D 1	ID1
11	A/D 3	IN3
12	A/D 5	ID3
13	A/D 7	IN7
14	+12VUNREG	+12VUNREG
15	I/O 4	IN4
16	I/O 6	STROBE
17	I/O 0	IN0
18	I/O 2	DATAOUT
19	AVCC	NC
20	ADVREF	NC
21	D/A 0	NC
22	A/D 0	ID0
23	A/D 2	IN2
24	A/D 4	ID2
25	A/D 6	IN6

# APPENDIX D: DIAGRAMS

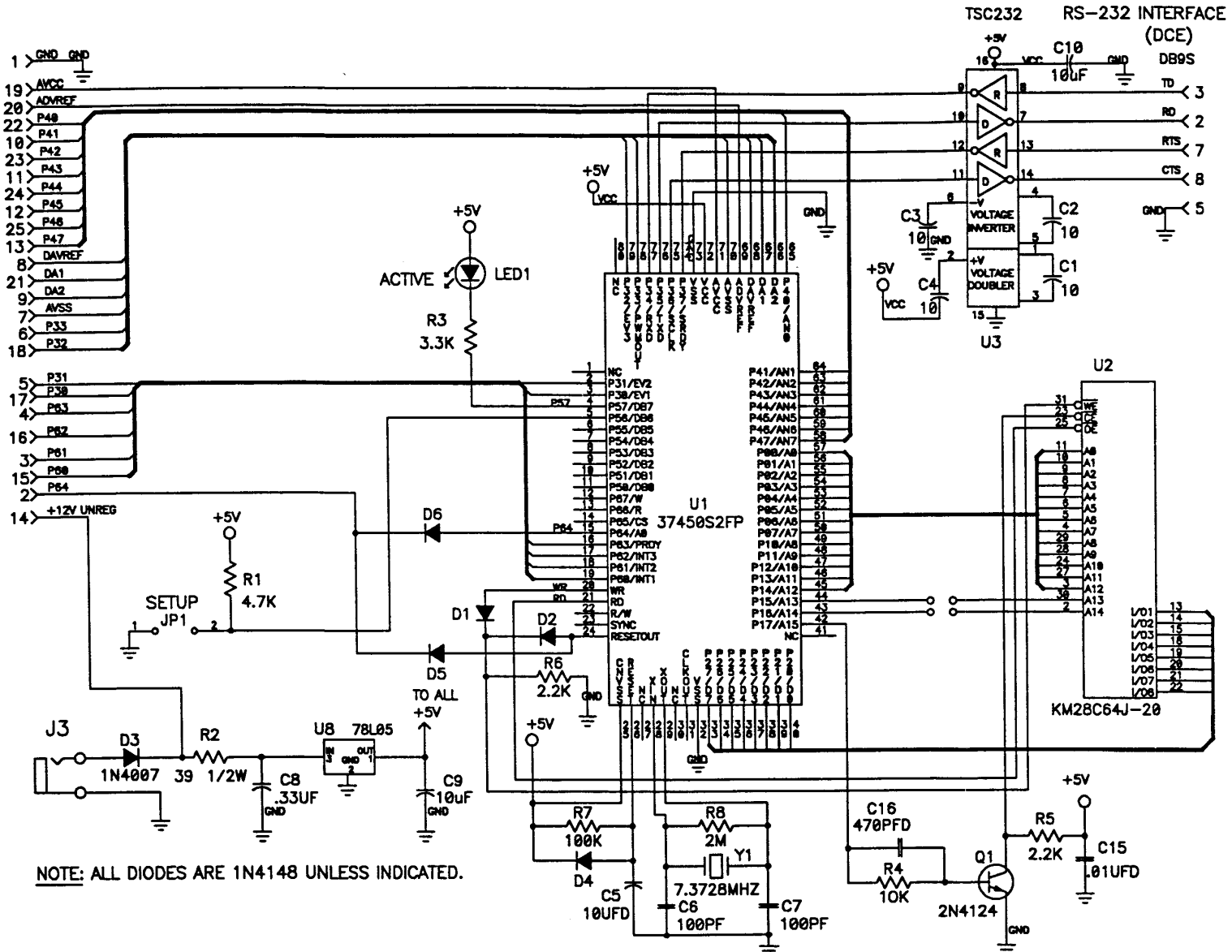


Figure D-1. SPIO

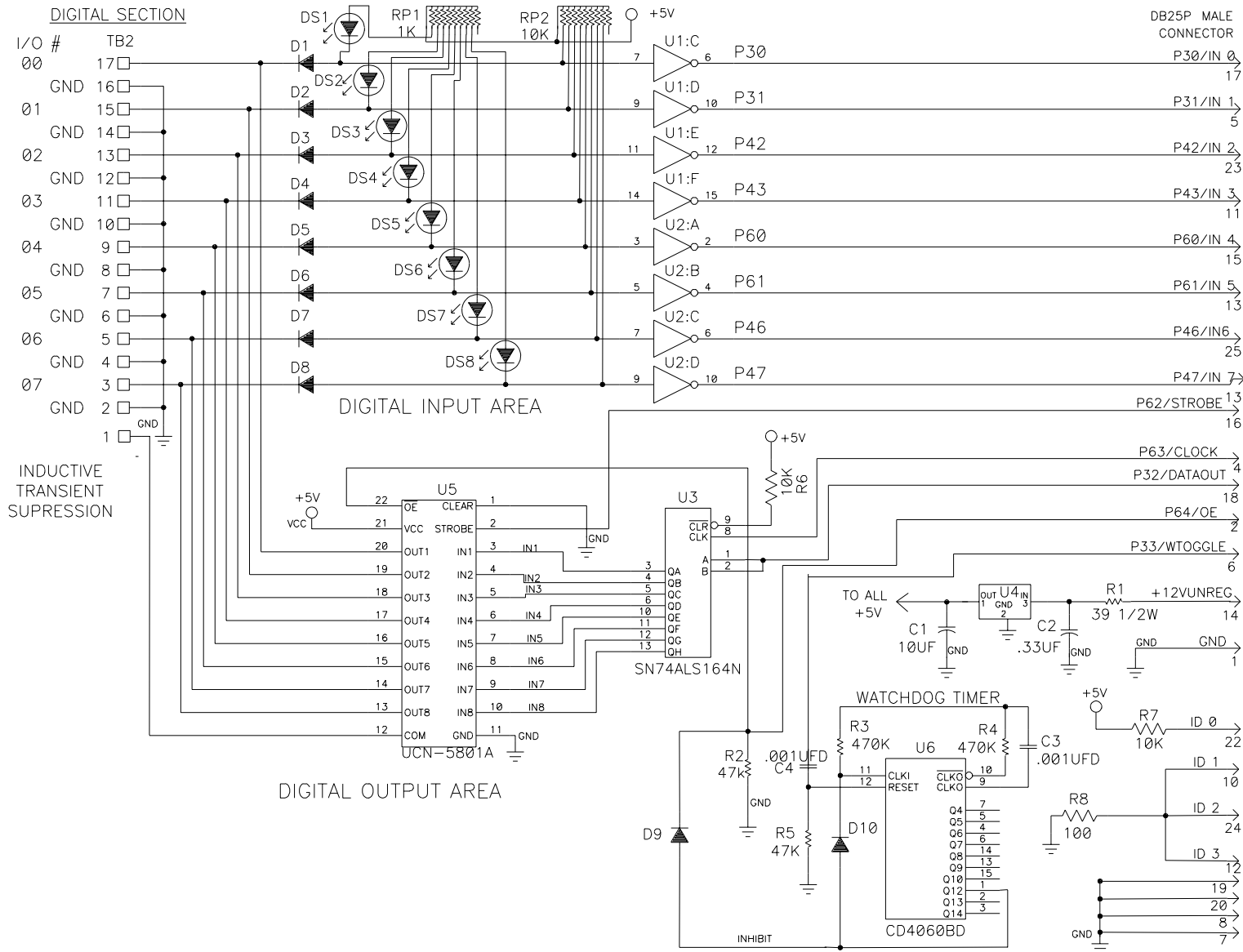


Figure D-2. SDIOB8

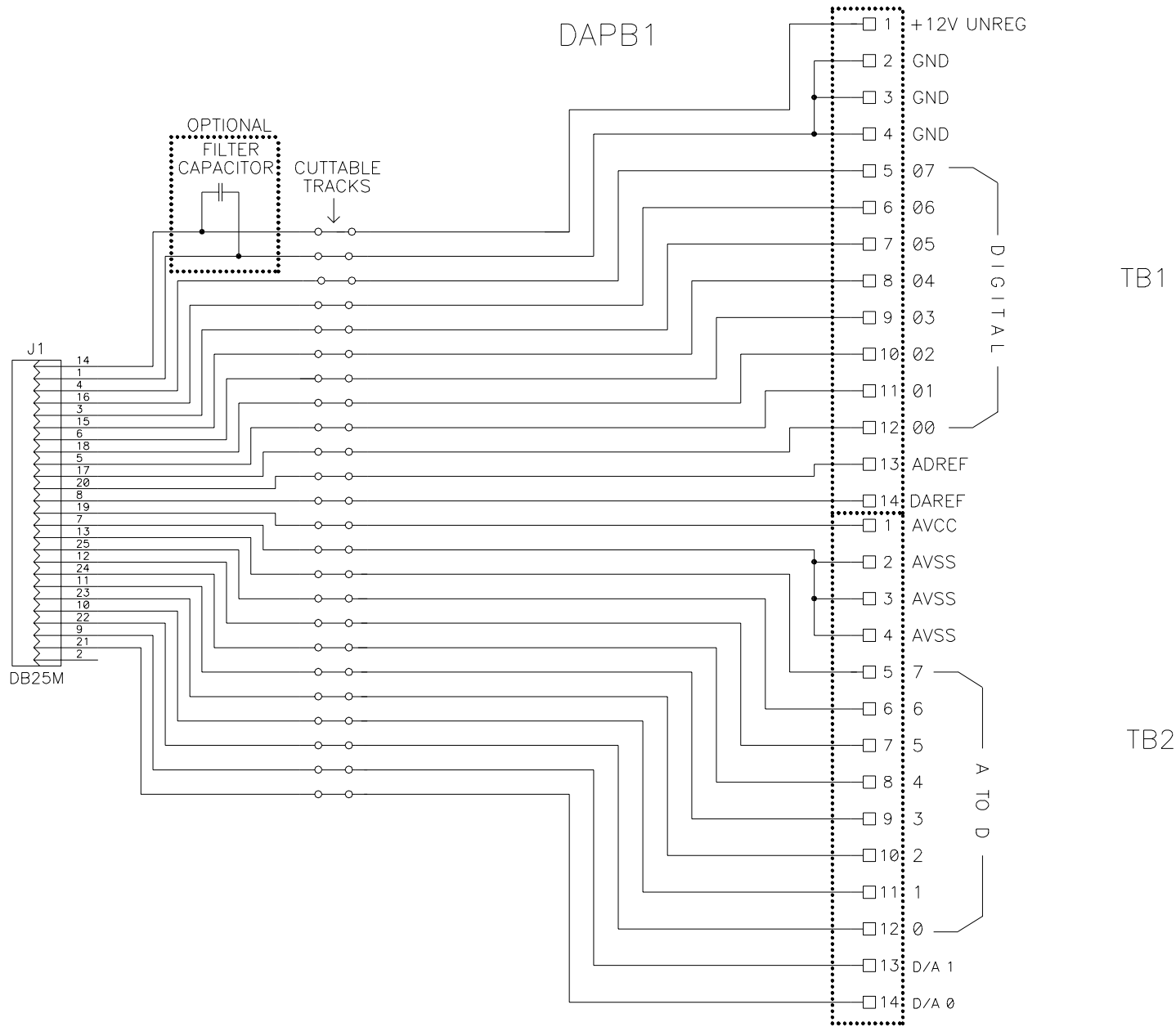


Figure D-3. DAPB1

## **APPENDIX E: Handling Unused I/O Lines**

To minimize the effect of noise, unused I/O lines should be handled as follows:

Digital I/O - Configure as Normal Outputs.

D/A Channels - Leave D/A channels open.

A/D Channels - Pull-down through 1k resistor.

## APPENDIX F: QBASIC DEMO PROGRAM

The QBASIC demonstration program for the SPIO shows how the module can be used as a temperature control unit. The source code and the executable file, SPDEM1QB.BAS and SPDEM1QB.EXE, are included in your SPIO package.

The first section of the program informs the user that the SPIO should be set up as follows:

Communications:	9600 baud, 8 data bits, no parity
Comm mode:	Two-Step
Comm timer:	Disabled
Turn around delay:	10 ms
Digital I/O:	All Normal Inputs
D/A Level:	0 Volts

The values above are the factory default settings of the SPIO mode. A normal exit from the demonstration program will return the SPIO module to factory default settings.

The power must now be disconnected from the SPIO module before continuing. The next section lists both the required and the optional connections for this program. For the program to be operational the following connections are required:

+5vdc	←-→	D/A Ref.	D/A Ref.	←-→	A/D Ref.
A/D Ref.	←-→	AVCC	A/D Ch.0	←-→	D/A Ch.0
GND	←-→	AVSS			

Refer to Figure F-1 for additional information.

**NOTE:** The +5vdc is not supplied by the SPIO! For a +5vdc source refer to Figure 2-10.

The optional connections provide the user some external input and output information. The following connections are optional:

Connect LED's in series with a 3.3k resistor from I/O's #00, #01, and #02 to +5vdc. This provides a visual indicator on each of the following: Furnace Status, Lower Alarm Status, and the Upper Alarm Status.

Connect 10k pull-up resistor between I/O #03 and +5vdc.

Connect a switch between I/O #03 and Ground. This switch can then be used to turn off an alarm.

Refer to Figure F-2 for additional information.

Once the connections have been made on the SPIO, the power can then be restored to the unit. The next section of the program will ask the user to enter which Comm Port is connected to the SPIO module. The program will set the Comm Port using the Set.Com.Port%(...) Command. If the Comm Port is not set properly, the program will display an error. Once the Comm Port has been properly set, the demonstration program will send out the Power.Up.Clear%(...). The program will then send out the Identify.Module%(...) Command. This command checks the SPIO's current firmware. If the proper firmware, SPIOV21.HEX, is not loaded into the SPIO, the program will be ABORTED!

The screen will show the demonstration setting up I/O #03 as an Input and I/O's #00 - #02 as Outputs. The latest messages to and from the SPIO are displayed on the screen.

## **MAIN MENU**

The main menu is the "active" part of the program. While the main menu is displayed on the screen the program is constantly reading the temperature. The program turns ON/OFF the furnace and/or the alarms based on the current temperature. The four options in the main menu are as follows:

- [1] Set Lower/Upper Alarm Limit.
- [2] Set Temp. Furnace Turns ON/OFF.
- [3] Disable Alarms. [ESC] EXIT Program.

The initial temperature values are as follows:

Furnace turns ON at: 68 F  
Furnace turns OFF at: 72 F  
Lower Alarm Limit: 60 F  
Upper Alarm Limit: 80 F

## ABOUT THE CURRENT TEMPERATURE

The current temperature is read from A/D channel 0. Normally, the A/D would be connected to thermocouple with some additional circuitry. For information connecting a thermocouple to the SPIO contact B & B. For this program the output from D/A channel 0 is used to simulate the thermocouple. With DAREF connected to +5vdc, the output of the D/A will range from 0 to +5vdc. The D/A will represent a temperature range from 45 F to 96 F. For the given temperature range: 1 step = 19.6mv = 0.2 F.

When the voltage is read from A/D channel 0, the voltage can be converted to a temperature using the given information as follows:

$$\text{temperature} = (\text{A/D value} * 0.2) + 45.0$$

Initially, the current temperature is set to 71 F. The temperature will then decrease beyond the furnace turn ON temperature. Next, the temperature will increase beyond the furnace turn OFF temperature. The temperature will then decrease beyond the lower alarm limit. Finally, the temperature will increase beyond the upper alarm limit. This cycle will repeat until the program is terminated. Note: Changing any temperature value will restart this process. Enabling/disabling alarms have no effect on this cycle.

## ENABLING/DISABLING ALARM

When an alarm is enable and the temperature exceeds the alarm limit, the corresponding alarm will be turned on until the alarm is turned off or disabled. The lower alarm is represented by I/O #00 and when activated the host computer will sound a single "BEEP" until the alarm is turned off. The upper alarm is represented by I/O #01 and when activated the host computer will sound two "BEEP's" until the alarm is turned off. An alarm can be turned off using I/O #03, by closing the switch shown in Figure F-2. An alarm may be disabled/enabled by choosing "3" in the main menu.

# REQUIRED CONNECTIONS

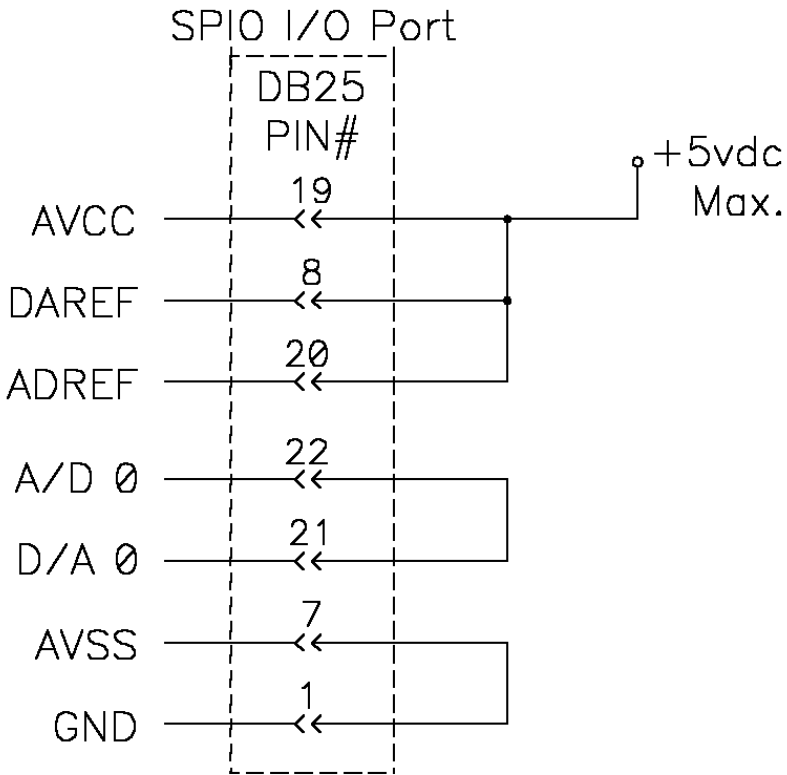


Figure F-1

# OPTIONAL CONNECTIONS

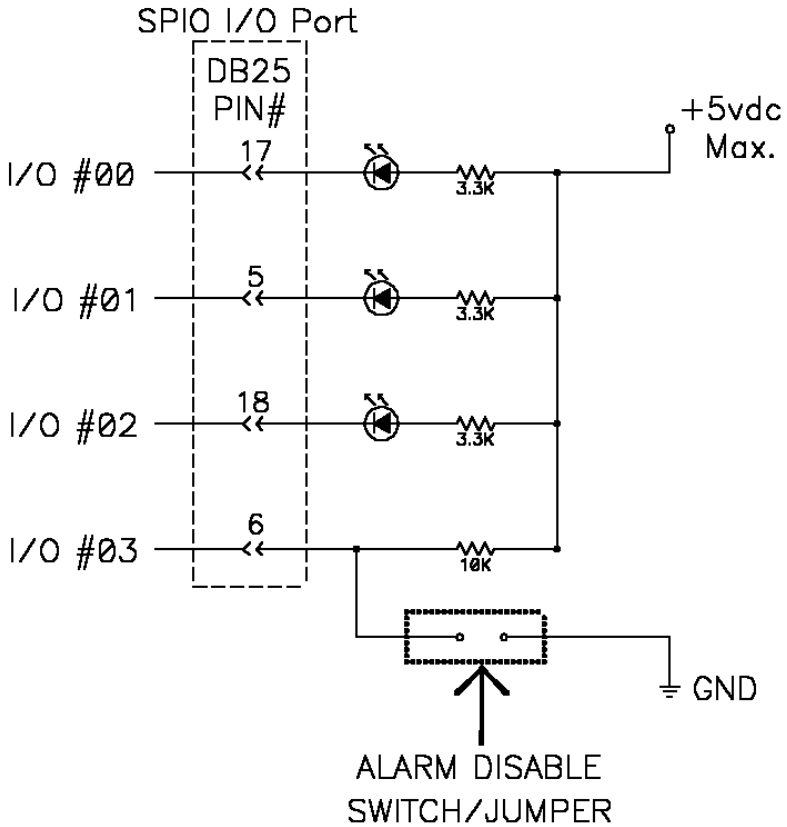


Figure F-2