GE
Intelligent Platforms

Programmable Control Products

Single-Slot
PC Interface Module (PCIM)

User's Manual

GFK–0881 March 2010

Warnings, Cautions, and Notes as Used in this Publication

Warning

Warning notices are used in this publication to emphasize that hazardous voltages, currents, temperatures, or other conditions that could cause personal injury exist in this equipment or may be associated with its use.

In situations where inattention could cause either personal injury or damage to equipment, a Warning notice is used.

Caution

Caution notices are used where equipment might be damaged if care is not taken.

Note: Notes merely call attention to information that is especially significant to understanding and operating the equipment.

This document is based on information available at the time of its publication. While efforts have been made to be accurate, the information contained herein does not purport to cover all details or variations in hardware or software, nor to provide for every possible contingency in connection with installation, operation, or maintenance. Features may be described herein which are not present in all hardware and software systems. GE Intelligent Platforms assumes no obligation of notice to holders of this document with respect to changes subsequently made.

GE Intelligent Platforms makes no representation or warranty, expressed, implied, or statutory with respect to, and assumes no responsibility for the accuracy, completeness, sufficiency, or usefulness of the information contained herein. No warranties of merchantability or fitness for purpose shall apply.

Contact Information

If you purchased this product through an Authorized Channel Partner, please contact the seller directly.

General Contact Information

Online technical support and GlobalCare	http://www.ge-ip.com/support
Additional information	http://www.ge-ip.com/
Solution Provider	solutionprovider.ip@ge.com

Technical Support

If you have technical problems that cannot be resolved with the information in this guide, please contact us by telephone or email, or on the web at www.ge-ip.com/support

Americas

Online Technical Support	www.ge-ip.com/support
Phone	1-800-433-2682
International Americas Direct Dial	1-780-420-2010 (if toll free 800 option is unavailable)
Technical Support Email	support.ip@ge.com
Customer Care Email	customercare.ip@ge.com
Primary language of support	English

Europe, the Middle East, and Africa

Online Technical Support	www.ge-ip.com/support
Phone	+800-1-433-2682
EMEA Direct Dial	+352-26-722-780 (if toll free 800 option is unavailable or if dialing from a mobile telephone)
Technical Support Email	support.emea.ip@ge.com
Customer Care Email	customercare.emea.ip@ge.com
Primary languages of support	English, French, German, Italian, Czech, Spanish

Asia Pacific

Online Technical Support	www.ge-ip.com/support
Phone	+86-400-820-8208
	+86-21-3217-4826 (India, Indonesia, and Pakistan)
Technical Support Email	support.cn.ip@ge.com (China)
	support.jp.ip@ge.com (Japan)
	support.in.ip@ge.com (remaining Asia customers)
Customer Care Email	customercare.apo.ip@ge.com
	customercare.cn.ip@ge.com (China)

Preface

The intent of this manual is to supply the user with enough information to establish the GE Single-slot PC Interface Module (PCIM) as an entry point into the Genius I/O System. The PCIM is designed to be integrated into a user–developed IBM PC microprocessor–based system. It provides a low cost 'tap' on the Genius I/O bus, allowing a host system to monitor and control remote I/O utilizing the extensive diagnostics, high reliability and noise immunity of GEs Genius I/O System.

This manual is intended for design engineers and systems or applications programmers who are already familiar with Basic or C programming in the IBM PC environment. Readers are further assumed to be familiar with the Genius I/O System.

Content of this Manual

This manual contains 7 chapters and 1 appendix:

Chapter 1: Introduction. Chapter 1 provides a Genius I/O system overview, description and specifications of the PCIM, and information about the PCIM software.

Chapter 2: Operation. Chapter 2 describes PCIM electrical characteristics, and explains how the host can interact directly with a PCIM, reading status information and setting control bits.

Chapter 3: Getting Started. Chapter 3 provides procedures for installing and configuring a PCIM. Chapter 3 also describes an external connector that can be added to the serial bus for interface to a Genius Hand-held Monitor.

Chapter 4: C Programming for the PCIM. Chapter 4 is a programmer's reference for creating a C language interface to the PCIM.

Chapter 5: BASIC Programming for the PCIM. Chapter 5 is a programmer's reference for creating a BASIC language interface to the PCIM.

Chapter 6: Communications. Chapter 6 describes Global Data and datagram communications for a PCIM.

Chapter 7: Troubleshooting Chapter 7 lists basic diagnostic procedures.

Appendix A: Example Applications. Appendix A shows three programming examples for the PCIM.

Related Publications

For more information, refer to these publications:

Genius I/O System User's Manual (GEK–90486–1). Reference manual for system designers, programmers, and others involved in integrating Genius I/O products in a PLC or host computer environment. This book provides a system overview, and describes the types of systems that can be created using Genius products. Datagrams, Global Data, and data formats are defined.

Genius Discrete and Analog Blocks User's Manual (GEK–90486–2). Reference manual for system designers, operators, maintenance personnel, and others using Genius discrete and analog I/O blocks. This book contains a detailed description, specifications, installation instructions, and configuration instructions for all currently–available discrete and analog blocks.

Series 90–70 Remote I/O Scanner User's Manual (GFK–0579). Reference manual for the Remote I/O Scanner, which interfaces a drop containing Series 90–70 modules to a Genius bus. Any CPU capable of controlling the bus can be used as the host. This book describes the Remote I/O Scanner features, configuration, and operation.

Series Six™ Bus Controller User's Manual (GFK–0171). Reference manual for the Bus Controller, which interfaces a Genius bus to a Series Six PLC. This book describes the installation and operation of the Bus Controller. It also contains the programming information needed to interface Genius I/O devices to a Series Six PLC.

Series Five™ Bus Controller User's Manual (GFK–0248). Reference manual for the Bus Controller, which interfaces a Genius bus to a Series Five PLC. This book describes the installation and operation of the Bus Controller. It also contains the programming information needed to interface Genius I/O devices to a Series Five PLC.

We Welcome Your Comments and Suggestions

At GE Intelligent Platforms, we strive to produce quality technical documentation. After you have used this manual, please take a few moments to complete and return the Reader's Comment Card located on the next page.

Jeanne L. Grimsby

Senior Technical Writer

Chapter 1 Introduction 1-1

Description 1-2
Daughterboard 1-3
Motherboard 1-3
Faceplate 1-3
Genius I/O System Overview 1-4
Specifications 1-5
PCIM Software 1-6

Chapter 2 Operation 2-1

PCIM Electrical Characteristics 2-1
Signal Conditioning 2-1
PCIM Status and Control 2-2
Daughterboard Shared RAM 2-5

Chapter 3 Getting Started 3-1

Introduction 3-1
Hardware Required 3-1
Software Required 3-1
Setting the Board Address DIP Switch 3-2
PCIM Installation 3-2
Connecting the Bus 3-3
Removing the PCIM from the Bus 3-4
Installing a Hand-held Monitor Connector 3-5
PCIM Startup 3-7
Using the Configuration Software 3-8
Notifying the Configuration Software of DIP Switch Change 3-8
Running the Configuration Software 3-9
Configuration Entries 3-10
Configuration Example 3-13

Chapter 4 C Programming for the PCIM 4-1

Compiling your Application with Microsoft 4-1
Software File Linkage 4-1
Software Driver Function Calls 4-2
Using Software Driver Function Calls 4-3
C Software Driver Function Call Parameters 4-4
InitIM – Setup and Activate PCIM 4-15
ChgIMSetup – Change PCIM Configuration 4-18

GetIMState – Get Configuration and Status Information 4-20
GetBusConfig – Get Serial Bus Configuration 4-22
GetDevConfig – Get Device Configuration 4-24
DisableOut – Disable/Enable Device Outputs 4-26
GetBusIn – Read all Input Values 4-28
PutBusOut – Write all Output Values 4-30
GetDevIn – Read Device Data Only 4-32
PutDevOut – Write Device Data Only 4-34
GetIMIn – Read Directed Input Table 4-36
PutIMOut – Write the Global Output Table 4-37
GetCir – Read Input Circuit Value 4-38
PutCir – Write Output Circuit Value 4-40
GetWord – Read Input Word Value 4-42
PutWord – Write Output Word Value 4-44
SendMsg – Send a Message 4-46
SendMsgReply – Send a Message Requesting a Reply 4-48
ChkMsgStat – Read Message Progress Status 4-50
GetMsg – Read Received Message 4-52
GetINTR – Read Interrupt Status Table 4-54
PutINTR – Write to the Interrupt Disable Table 4-56

Chapter 5 BASIC Programming for the PCIM 5-1

Basic Software Driver Installation 5-1
Software Driver Function Calls 5-2
Using Software Driver Function Calls 5-3
Basic Software Driver Function Call Parameters 5-4
Basic Data Array Structures 5-4
Error Status Indication 5-10
Access from BASIC 5-11
Coding Basic Function Calls 5-12
INITIM CALL Statement 5-13
CHGIMSETUP CALL Statement 5-16
GETIMSTATE CALL Statement 5-18
GETBUSCONFIG CALL Statement 5-20
GETDEVCONFIG CALL Statement 5-22
DISABLEOUT CALL Statement 5-24
GETBUSIN CALL Statement 5-26
PUTBUSOUT CALL Statement 5-28
GETDEVIN CALL Statement 5-30
PUTDEVOUT CALL Statement 5-32
GETIMIN CALL Statement 5-34
PUTIMOUT CALL Statement 5-35
GETCIR CALL Statement 5-36
PUTCIR CALL Statement 5-38
GETWORD CALL Statement 5-40
PUTWORD CALL Statement 5-42
SENDMSG CALL Statement 5-44

SENDMSGREPLY CALL Statement 5-46
CHKMSGSTAT CALL Statement 5-48
GETMSG CALL Statement 5-50
GETINTR CALL Statement 5-52
PUTINTR CALL Statement 5-54

Chapter 6 Communications 6-1

Introduction 6-1
Global Data 6-1
Datagram Data 6-3

Chapter 7 Troubleshooting 7-1

Introduction 7-1
Replacement Module Concept 7-1
PCIM Troubleshooting 7-2
LEDS 7-2
Fault Isolation and Repair 7-2

Appendix A Example Applications G-1

Example Application 1 ..... G-1
Example Application 2 ..... G-5
Example Application 3 ..... G-10

Chapter 1 Introduction

This manual provides a description of the GE Genius I/O IBM PC Interface Module (PCIM). It includes procedures for setup, programming, operation, and troubleshooting in conjunction with the Genius I/O System.

This manual also describes the PCIM Software Library, software which is supplied with the PCIM. The Software Library provides a high level interface between applications software you develop and the PCIM. The PCIM Software Library consists of easy to use macro–oriented function calls you code appropriately in your C language or Basic language applications routines.

Suitable Computers

The PCIM has been tested successfully in many types of IBM PC XT and AT-type computers. It is fully compatible with the ISA backplane, and provides host system address decoding over the full PC, XT or AT memory maps.

However, it has not been possible to test the PCIM with all computers that may be available. Therefore, proper operation of the PCIM in every type of host computer cannot be assured.

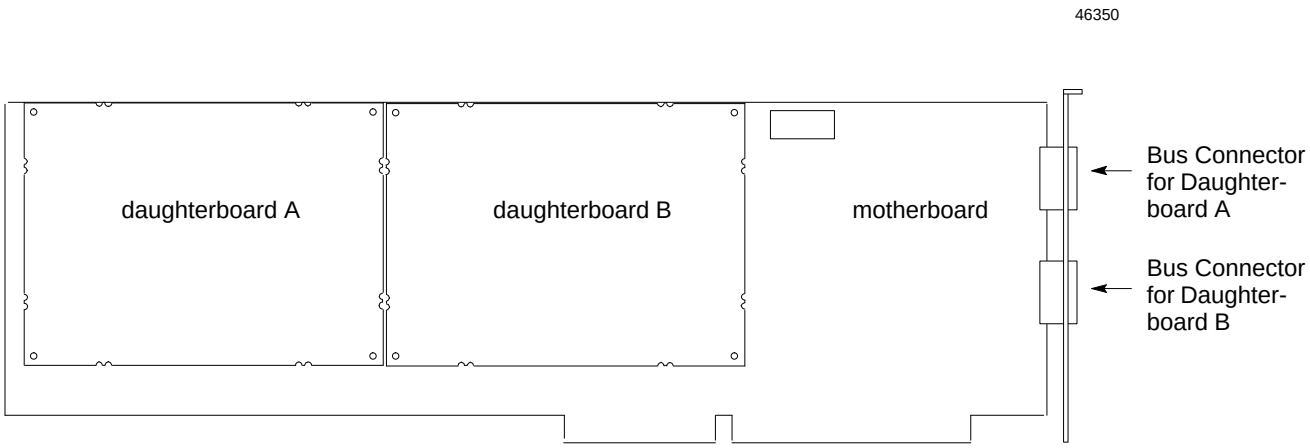
Using Other Interface Software

A number of companies have developed software for the PCIM, to be used in place of the PCIM Software Library.

If you are using such software, and have questions or are experiencing problems, please contact the software company. GE cannot provide customer support for other companies' PCIM software products.

Description

The Single-Slot PC Interface Module (PCIM) is an entry point into the Genius I/O System for the IBM PC / AT / XT™ family. The PCIM is an "AT" style board, designed to be integrated into a user–developed microprocessor system. It is fully compatible with all Genius protocols, mechanical, electrical levels, and communications timing.



The PCIM is available with either one or two daughterboard(s). Each PCIM daughterboard provides a low cost 'tap' on a Genius I/O bus, allowing a host system to control remote I/O utilizing the extensive diagnostics, high reliability and noise immunity of the Genius I/O System. Each daughterboard is independently configurable using the configuration software supplied with the PCIM.

Board-edge connectors are used to connect the PCIM to the Genius bus. If the PCIM has two daughterboards, they can be connected to the same bus or to independent busses.

Daughterboard

A PCIM daughterboard is a general purpose I/O Controller for the Genius I/O System. It provides a convenient method to control devices on the Genius serial bus. The PCIM daughterboard performs the housekeeping tasks of initialization and fault management for up to 30 bus devices, keeps up–to–date images of the I/O controlled by each device (whether the device is a Genius I/O Block or other bus device), and can communicate with other Controllers on the Genius bus by passing background messages not associated with I/O commands or Global Data. The interface to this RAM is optimized for the IBM personal computer bus.

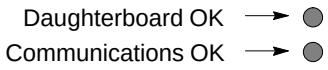
Motherboard

The PCIM motherboard provides a convenient way to interface an Open Architecture daughterboard like the PCIM daughterboard to an IBM compatible Host system. All the signals necessary to communicate to a daughterboard are buffered through the motherboard to the Host bus. In addition to the normal interface lines, the motherboard provides the following daughterboard control and monitoring functions:

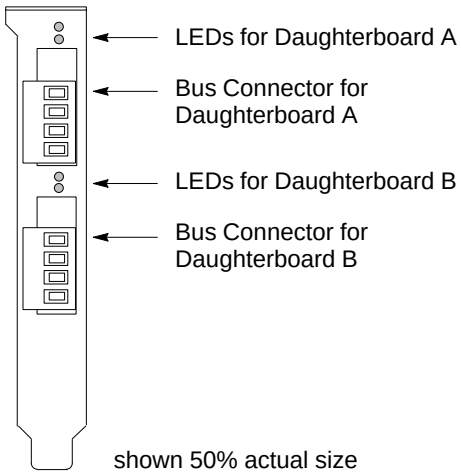
- A standard 'unit load' to the IBM bus.
- Works in ISA–compatible backplanes.
- Low supply voltage detection.
- Power up RESET signal sequencing.
- Host system address decoding over the full PC, XT or AT memory maps.
- A built–in watchdog timer that can monitor system operation and shut down the daughterboard if the Host system faults, preventing any conflicts on the Genius bus. Note that this timer is not used with the Software Library.

Faceplate

For each daughterboard, two LEDs (Board OK and Communications OK are provided in the PCIM faceplate. For each daughterboard, the LEDs are as shown below:



Openings in the faceplate accommodate the serial bus connectors for the PCIM daughterboard(s).



Genius I/O System Overview

The Genius I/O is a system of inherently distributed inputs and outputs, which consists of:

- Genius I/O Blocks AC, DC, Isolated, Analog, and others (mounted at the point of control),
- A Bus Controller (which serves as the interface between the Genius I/O system and a programmable controller),

  and/or

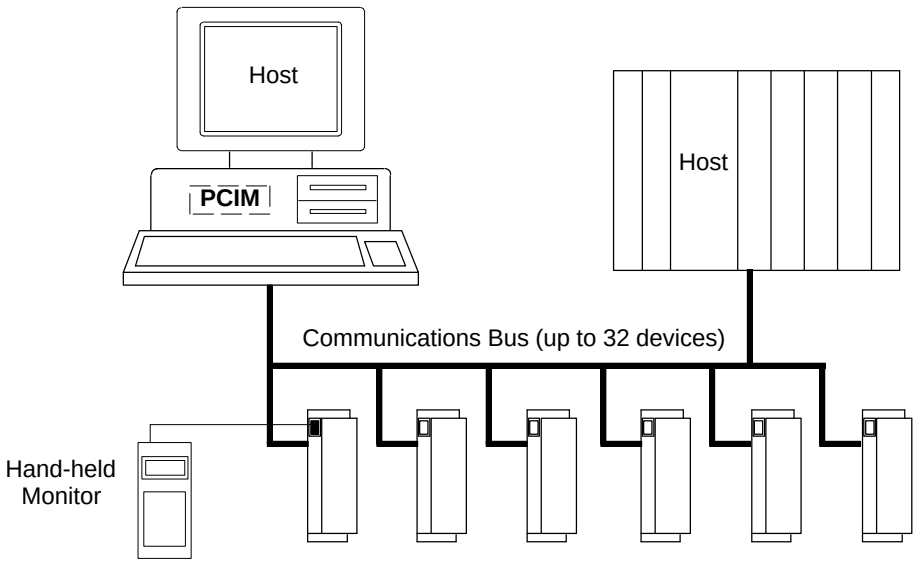
- A PCIM for interface with IBM PC ATs, XTs, or CIMSTAR I,
- A Hand Held Monitor (the portable diagnostic and configuration tool used for addressing, trouble–shooting, monitoring, scaling and configuring the I/O Blocks),
- And the Genius Serial Bus, which provides communications between the Bus Controller, Hand Held Monitor, and up to 30 I/O Blocks over a single shielded twisted wire pair.

Normally, Genius I/O will be controlled by a PLC in machine control and fast closed loop control applications. There are various applications, however, where systems based on Genius I/O blocks will be utilized with IBM PC products.

Genius I/O Blocks provide superior, built–in Diagnostics which detect open circuits, short circuits, overloads, and a variety of other malfunctions which are beyond the power of conventional PLCs to detect.

A simplified diagram of a typical Genius I/O System is shown below.

46201



Specifications

Catalog Numbers	
Single-channel PCIM	IC660ELB921
Dual-channel PCIM	IC660ELB922
MicroGENI daughterboard	IC660ELB912
LEDs (2 for each daughterboard)	GENI OK, COMMS OK (Communications OK)
Electrical	
Power Requirements	5 volts DC + / – 10%, 400 ma (maximum)
Bus Loading	1 LS TTL load per input line
Bus Drive Capability	10 LS TTL loads per output line
Mechanical	
PCIM board type	Single-slot "AT" style board
Hand-held Monitor connection	External connector with HHM and bus terminals
Serial bus connection	Board-edge terminals or external connector. Board-edge terminals accept two AWG #20 (avg .55mm^2 cross section) wires or three AWG #22 (avg .36mm^2 cross section) wires.
Host backplane interface	fully ISA compatible
Memory Requirements	
Motherboard	4 bytes
Each daughterboard	16K bytes
Environmental Requirements – Operating	
Temperature	0 to 60 degrees C (ambient temperature at board)
Humidity	5% to 95% non–condensing
Altitude	10,000 feet
Vibration	0.2 inch displacement 5 to 10 Hz 1 G 10 to 200 Hz
Shock	5 G, 10 ms duration per MIL–STD 810C, method 516.2
Environmental Requirements – Non–operating	
Temperature	–40 to 125 degrees C (ambient temperature at board)
Humidity	5% to 95% non–condensing
Altitude	40,000 feet
Vibration	0.2 inch displacement 5 to 10 Hz 1 G 10 to 200 Hz
Shock	5 G, 10 ms duration per MIL–STD 810C, method 516.2

PCIM Software

The PCIM is supplied with two types of software:

- Configuration software
- Interface software: the PCIM Software Library

PCIM Configuration Software

The configuration software is used to set up the characteristics of one or more PCIMs that will be installed in the computer.

Setup parameters include the base addresses used by the Single-Slot PCIM and its daughterboard(s), baud rate, serial bus address, outputs enable, and Watchdog Timer enable. Configuration data is stored in EPROM memory on the PCIM, and is retained if power is removed.

The configuration software can be run from diskette or the configuration files can be copied to a hard disk.

PCIM Software Library

The PCIM Software Library provides a high level interface between applications software you develop and the PCIM; and through the PCIM, devices on the Genius serial bus. The PCIM Software Library is accessed through a set of subroutine calls.

The PCIM Software Library is provided in versions compatible with C language and Basic language, specified as a set of function calls in order to allow a consistent interface with both languages. Library software is delivered in the form of object code in a single .exe (.COM) file. This user's guide covers both C language and Basic language applications.

The PCIM Software Library is supplied in a version compatible the MSDOS operating system, as follows:

C/MSDOS

BASIC/MSDOS

Chapter 2

Operation

PCIM Electrical Characteristics

Power Supply Requirements

The PCIM requires a 5 volt DC source for logic power. Supply voltage should not vary more than 10% above or below nominal (below 4.5 V DC or above 5.5 V DC), or the PCIM will not function correctly. The PCIM with one daughterboard (single-channel PCIM) typically draws 1.0 Amps. The PCIM with two daughterboards (dual-channel PCIM) typically draws 1.5 Amps.

Bus Loads/Drive Capability

All input lines to the PCIM present no more than one standard LSTTL load to the host interface connector.

All output lines from the PCIM are capable of driving 10 standard LSTTL loads. These lines, with the exception of the /INT and /PCIM OK lines, are tri–state outputs. The /INT line is an open–collector output that can be wired–ORed to a single interrupt input. The /PCIM OK and /COMM OK lines are low–true open collector type outputs with built–in current limiting to 10 ma suitable for driving LEDs directly.

All input signals to the PCIM from the Host system look like one LSTTL load to the host system. These signals are TTL compatible and switch at TTL levels.

Control output signals to the host are open–collector LSTTL drivers with 10K resistive pull–ups, capable of sinking 4 mA while maintaining an output voltage of 0.4V or lower.

The data transceiver is a tri–state LSTTL device capable of sourcing or sinking 12 mA with VOL = 0.4V and VOH = 2.0V.

The PCIM is fully compatible with ISA backplanes.

Signal Conditioning

The PCIM has two connectors that you can access when the PCIM is installed in a PC type rack. Both connectors are for the standard twisted pair connection to a serial bus.

The Hand-held Monitor can be connected through an interface cable to the separate Genius connectors.

All of the lines in from both connectors are either isolated or impedance limited to protect the PCIM from voltage spikes or the misapplication of high voltages on the serial bus connections.

PCIM Status and Control

The PCIM motherboard uses four bytes of mapped I/O memory space. These four bytes start at the I/O base address and are configured by the software utility, and have the functions shown below. Only the first two bytes are used.

Byte #	A7 – A0	Description
0	XXXXXXX00	PCIM Status byte
1	XXXXXXX01	PCIM Control byte

For example, if the the address selected is 3E0 (hex), you can perform operations on these addresses:

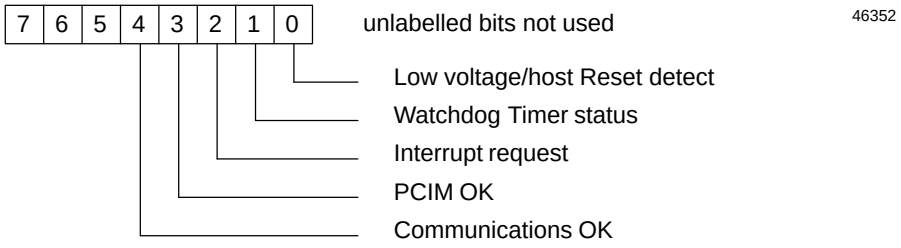
(3E0 + 0) = 3E0 PCIM Status Byte

(3E0 + 1) = 3E1 PCIM Control Byte

Bit definitions for the Status Byte and Control Byte are given on the following pages.

PCIM Status Byte: Bit Definitions

The individual bits in the PCIM Status byte have the following functions:



0 – Low Voltage/Host RESET Detect

This input goes to 0 and stays 0 (until reset) whenever the voltage on the motherboard drops below 3.12 volts or the Host system has gone into RESET. This bit is reset by the '1' bit of the PCIM Output byte (see next page). During normal operation this bit should be 1.

Note

Do not enable interrupts, or read / write to the PCIM for 2 seconds (the period of time required for hardware / software initialization) after reset. One false interrupt occurs within this time period. Reading or writing to the PCIM during this time may cause the watchdog timer to time out. The PCIM OK flag will be invalid during this period of time.

1 – Watchdog Timer Status

(Not used with the software library). This bit is 1 if the watchdog timer as been enabled by the configuration software and is being pulsed every 727mS by PCIM Control bit 0 (see next page). If the timer expires, this bit goes to 0. It will also go to 0 if the voltage detector detects low voltage. If not enabled by configuration, the timer does not need to be pulsed.

2 – Interrupt Request

When the daughterboard generates an interrupt to the motherboard, this goes to 1 and stays 1 until reset by Control bit 2 (see next page).

3 – PCIM OK

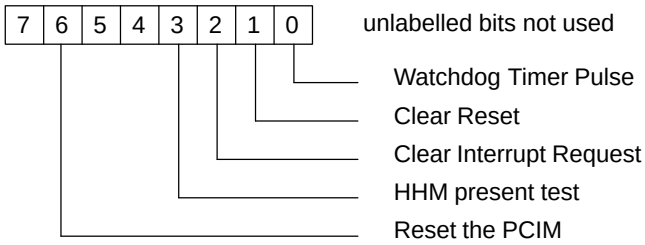
The state of this bit follows the condition of the PCIM OK LED on the daughterboard. If the LED is lit, the PCIM OK bit is 0.

4 – COMM (Communications) OK

Like the BOARD OK bit, this bit follows the output of one of the LEDs on the daughterboard. This bit is 0 if the COMM OK LED on the daughterboard is lit.

PCIM Control Byte: Bit Definitions

The individual bits in the PCIM Control byte are used for the following functions:



0 – Watchdog Timer Pulse

The watchdog timer is a hardware timer that can be enabled by the configuration software. If the watchdog timer is enabled, it must be reset periodically or it will put the PCIM into RESET. You can toggle the watchdog timer and use it as a failsafe timer to ensure that if the Host system 'hangs up', the PCIM will not send any erroneous messages to the serial bus. If the watchdog timer is disabled by by the configuration software, you do not have to toggle it; it will stay turned off and will not put the PCIM into RESET.

If the watchdog timer is enabled by the configuration software, this bit must be pulsed at least every 727mS to keep the watchdog timer from expiring. This bit must be pulsed at least once to allow the daughterboard to come out of RESET.

1 – Clear RESET Request

When the system is Reset, or when the voltage detector on the motherboard detects a low voltage condition, status bit 0 (see previous page) goes to 0. Command bit 1 (Clear Reset) clears the reset request when set to 0. To prepare for the next detection of RESET or low voltage condition, it must be reset to 1.

2 – Clear Interrupt Request

This bit is used to clear an interrupt to the motherboard from a daughterboard. Setting the bit to 0 clears the interrupt. It must then be set back to 1 to prepare it for the next interrupt.

3 – HHM Test

An HHM present can be indicated even when one is not plugged in by raising this bit to 1. After power up and under normal conditions, this bit should be 0.

6 – PCIM RESET

When this bit is 0 it resets the PCIM. Under normal conditions, it should be left high.

Daughterboard Shared RAM

Each PCIM daughterboard uses 16K bytes of host memory. For information purposes only, the structure of this area is illustrated below.

You do not need to understand how this Shared RAM memory works if your application program will use the PCIM Software Driver functions (described in chapters 4 and 5) to interact with this memory area.

Relative Location dec.	Relative Location hex.	Content	Size in Bytes	
0000	0000	Request Queue	(2176)	**Request Queue**: Queue for incoming Read Device, Write Device, and Write Point datagrams to the host.
2176	0880	Request Queue Head Pointer*	(1)	**Request Queue Head Pointer**: Number of the Request Queue buffer currently being read.
2177	0881	Request Queue Tail Pointer	(1)	**Request Queue Tail Pointer**: Indicates the most recent entry in the Request Queue.
2178	0882	µGENI Setup Table	(16)	**µGENI Setup Table**: Characteristics of µGENI and the bus.
2194	0892	µGENI Status Table	(16)	**µGENI Status Table**: Diagnostics for µGENI and the bus.
2210	08A2	Interrupt Status Table	(16)	**Interrupt Status Table**: Current status of interrupts to host.
2226	08B2	Interrupt Disable Table	(16)	**Interrupt Disable Table**: Used to enable / disable host interrupts.
2242	08C2	Command Block*	(16)	**Command Block**: Used by host to send Read Datagram, Transmit Datagram, Transmit Datagram with Reply, and Configuration Change commands to µGENI.
2258	08D2	Transmit Datagram Buffer	(240)	**Transmit Datagram Buffer**: Temporary location for sending datagrams.
2498	09C2	Read Datagram Buffer	(134)	**Read Datagram Buffer**: Location where host may read incoming datagrams.
2632	0A48	I/O Table Lockout Request *	(1)	**I/O Table Lockout** Request / Relinquish: Used to set or release µGENI lockout of I / O Tables.
2633	0A49	I/O Table Lockout State	(1)	**I/O Table Lockout State:** Actual lockout state.
2634	0A4A	Host Clear	(1)	**Host Clear:** Used by mGENI to clear Interrupts from the host.
2635	0A4B	Reserved	(64)	**Reserved Area:** The host should NOT read or write here.
2699	0A8B	Auxiliary Request Queue	(48)	**Auxiliary Request Queue:** Used in conjunction with Request Queue.
2747	0ABB	Heartbeat Enable	(2)	**Heartbeat Enable:** Used to enable host tomGENI heartbeat monitoring.
2749	0ABD	Heartbeat Timeout Multiplier	(1)	**Heartbeat Timeout Multiplier:** Sets the heartbeat interval.
2751	0ABF	Reserved	(4930)	**Reserved Area:** The host should NOT read or write here.
7680	1E00	Device Configuration Table	(256)	**Device Configuration Table:** Location of device ID, status, and setup information.
7936	1F00	Directed Control Input Table	(128)	**Directed Control Input Table:** Location for receiving Directed Control Data.
8064	1F80	Broadcast Control Output Table	(128)	**Broadcast Control Output Table:** Buffer for sending Global Data.
8192	2000	Device I/O Table	(8192)	**Device I/O Table:** Contains all device inputs and outputs, and incoming Global Data.
16383	3FFF			

* Host write causes interrupt to µGENI daughterbaord

Chapter 3

Getting Started

Introduction

In order for you to interface the PCIM with the Genius serial bus, you must first perform the following steps:

- Set the configuration I/O address on the DIP switches.
- Install the PCIM in the host.
- Connect the PCIM to the serial bus.
- Run the configuration software.

Hardware Required

In addition to the devices normally considered part of the Genius I/O system, the following hardware is required to effect a Genius I/O – PCIM – Host communications interface:

- An IBM PC/AT, IBM PC/XT, or compatible computer
- A PCIM

Software Required

The following software is required to effect Genius I/O – PCIM – Host communications:

- MS DOS® version 3.0 or higher

and

- dpcimcfg.exe (configuration software)
- pcim.lib (C Software Driver – small memory model)
- lpcim.lib (C Software Driver – large memory model)
- pcim.h (C Software Driver – include file)

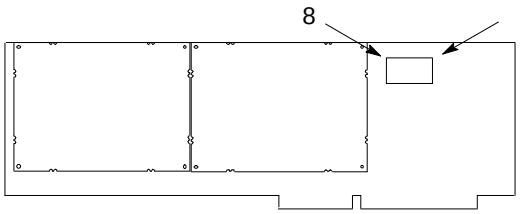
or

- pcimx.exe (BASIC Software Driver)
- pcim.bas (BASIC startup sequence)

Setting the Board Address DIP Switch

Before installing the PCIM in the computer, it may be necessary to set its address-selection DIP switches. The default setting is 222hex. The board address must be unique for each module; if there are multiple PCIMs or if the address conflicts with addresses used by other modules in the system, you must change it to an address in the range 102hex to 3FE hex.

Switch positions are numbered 1 through 8. Use switches 1 and 2 to set the high hex digit, switches 3, 4, 5, and 6 to set the middle hex digit, and switches 7 and 8 to set the low hex digit.



switches 1 2	high digit
# #	0
# "	1
" #	2
" "	3

switches 3 4 5 6	middle digit
# # # #	0
# # # "	1
# # " #	2
# # " "	3
# " # #	4
# " # "	5
# " " #	6
# " " "	7
" # # #	8
" # # "	9
" # " #	A
" # " "	B
" " # #	C
" " # "	D
" " " #	E
" " " "	F

switches 7 8	low digit
# #	2
# "	6
" #	A
" "	E

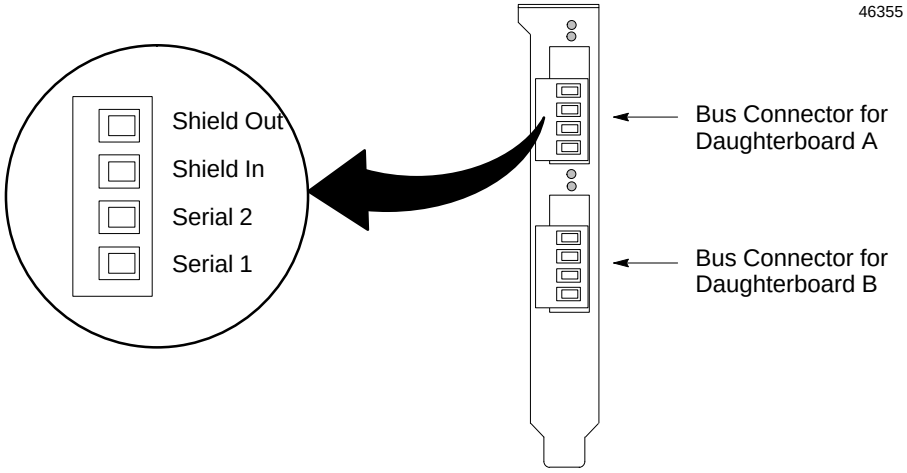
" means down toward board
means up away from board

PCIM Installation

1. Power OFF the Host computer and unplug from power source.
2. Install the PCIM according to the computer manufacturer's instructions for option cards.
3. Connect the bus (see next page) to the PCIM.
4. DO NOT
   - Mount the PCIM where air flow across it is obstructed
   - Mount the PCIM nearer than 1/8″(.318cm) to any other boards or rack components
   - Use adhesives or conformal coatings on any part of the PCIM

Connecting the Bus

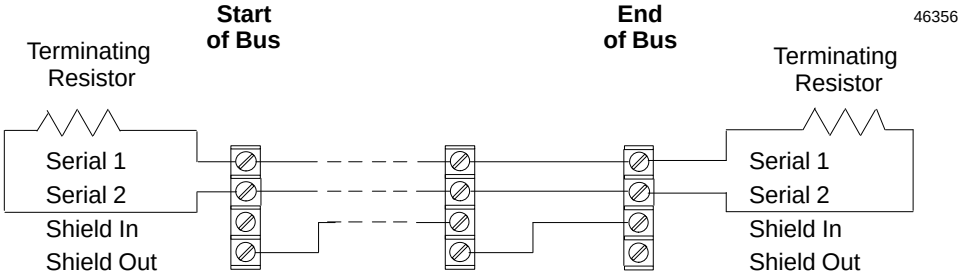
Devices can be placed in any physical sequence on the bus. Each connector on the PCIM has four terminals for the bus cable (Serial 1, Serial 2, Shield In, and Shield Out). Note that the sequence of these terminals on a PCIM connector is not the same as on other bus devices (for example, I/O blocks).



These terminals accept two AWG #20 wires (each avg .54mm^2 cross-section) plus one lead of a quarter-Watt resistor (optional: used for bus termination). The minimum recommended wire size is AWG #22 (avg .36mm^2 cross-section).

Connect the Serial 1 terminal of each connector to the Serial 1 terminals of the previous device and the next device. Connect the Serial 2 terminal of each connector to the Serial 2 terminals of the previous device and the next device. If the PCIM has two daughterboards, they may be connected to different busses or to the same bus.

Shield In of each connector must be connected to Shield Out of the preceding device. For the first device on the bus, Shield In can be left unconnected. For the last device on the bus, Shield Out can be left unconnected.



When making bus connections, the maximum exposed length of bare wires should be two inches. For added protection, each shield drain wire should be insulated with spaghetti tubing to prevent the Shield In and Shield Out wires from touching each other.

Bus Termination

A bus must be terminated at each end by impedance that is correct for that cable type. Impedance will be 75, 100, 120, or 150 ohms. If a PCIM connector is at either end of its bus, install the appropriate terminating resistor across the Serial 1 and Serial 2 terminals. The *Genius I/O System and Communications User's Manual* lists appropriate terminating resistors for each recommended bus cable type.

Removing the PCIM from the Bus

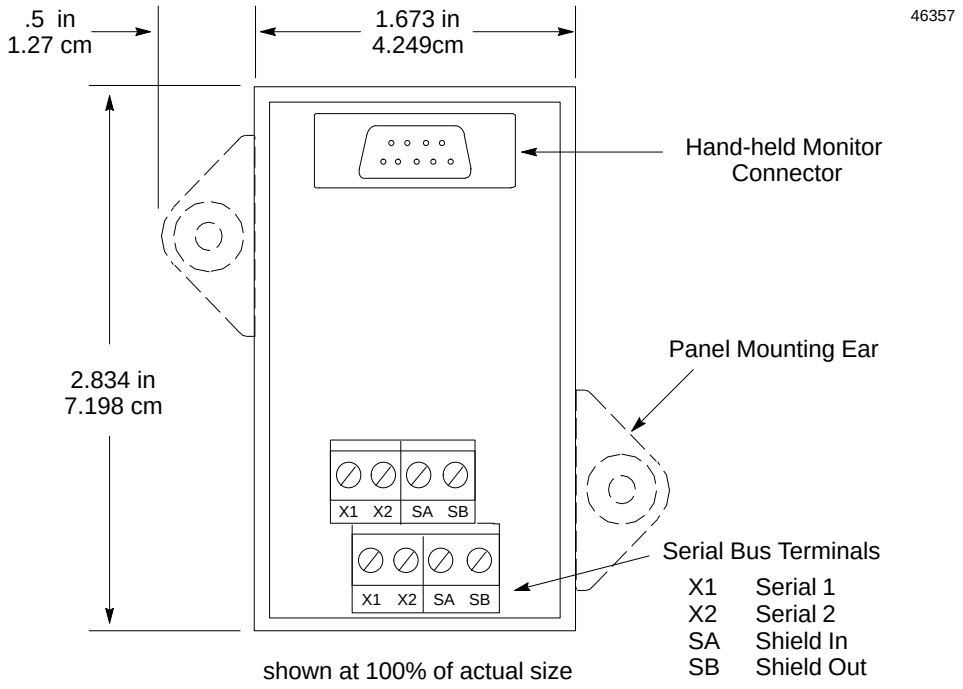
The PCIM's bus connectors are removable; they can be removed while the system is operating without compromising data integrity on the bus. To remove a bus connector, hold it carefully by its top and bottom sides and pull it away from the PCIM. If an operating cable is presently attached to the bus, be very careful not to touch the bus wires to each other or to anything else. Do not put the connector down on a conductuve surface.

Individual bus wires should never be removed from the connector terminals while the bus is in operation; the resulting unreliable data on the bus could cause hazardous control conditions.

Installing a Hand-held Monitor Connector

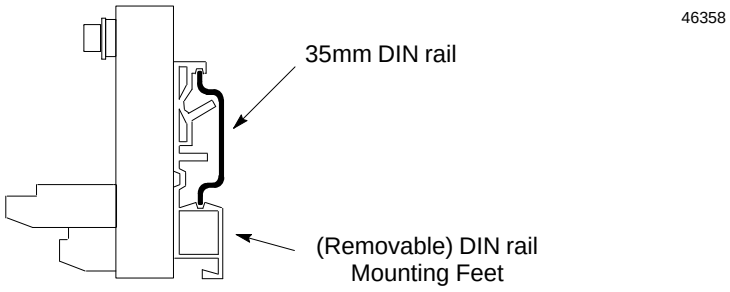
The PCIM does not have a built-in connector for a Genius Hand-held Monitor. However, a Hand-held Monitor connector can be added directly to the serial bus at any location.

The unit shown below (catalog number 44A736310–001–R001) is provided with the PCIM. It provides a Hand-held Monitor connector and serial bus terminals.



Mounting the HHM Connector

This unit can be easily mounted on a rail such as a standard 35mm (shown below) or 15mm DIN rail. The panel-mounting ears are not used if the unit is installed on a DIN rail.

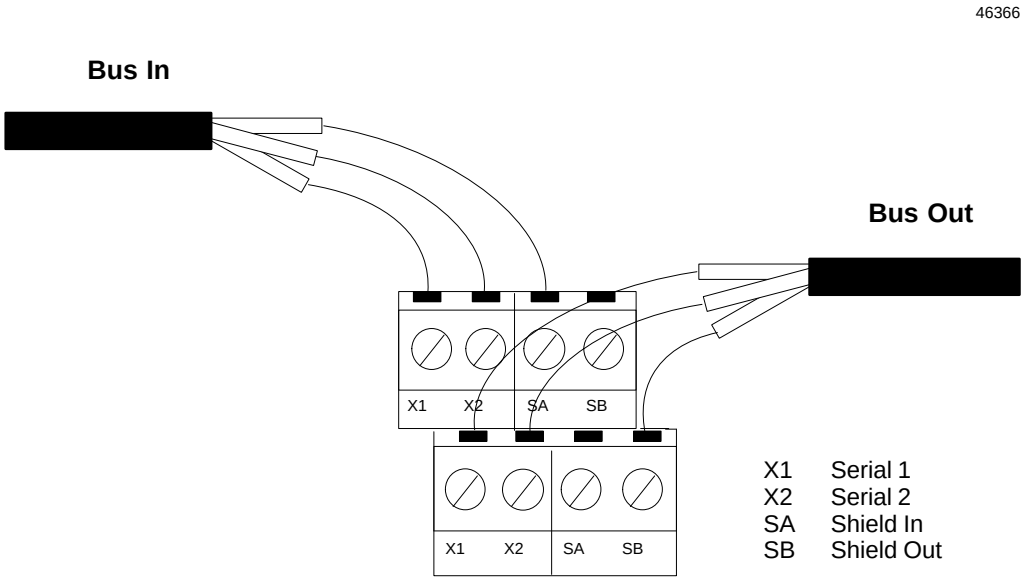


Alternatively, it can be installed directly on a panel using screws through its mounting ears. The DIN rail feet on the back of the unit are removed when the unit is panel-mounted.

Making the Bus Connections

The HHM connector has two sets of terminals; one for incoming cable and the other for outgoing cable.

Connect the Serial 1, Serial 2, and Shield In terminal of either connector to the previous device. Connect the Serial 1, Serial 2, and Shield Out terminal of the other connector to the next device. (For the first device on the bus, Shield In can be left unconnected. For the last device on the bus, Shield Out can be left unconnected.)



When making bus connections, the maximum exposed length of bare wires should be two inches. For added protection, each shield drain wire should be insulated with spaghetti tubing to prevent the Shield In and Shield Out wires from touching each other.

As with other devices, if the HHM Connector is at either end of its bus, install an appropriate terminating resistor across the Serial 1 and Serial 2 terminals. The *Genius I/O System and Communications User's Manual* lists appropriate terminating resistors for each recommended bus cable type.

PCIM Startup

You may now activate the PCIM as follows:

1. Plug in and power ON the Host computer.
2. If the PCIM has not been configured, insert the diskette containing the Software Driver and associated files into Drive B.
3. Set the disk drive to B.
4. Run the Configuration Software, as described on the next page.

   Beyond the self tests, the PCIM will do nothing until it is explicitly taken out of RESET. This is accomplished via the application program code you write – specifically, through the INITIM Software Driver function call.

Using the Configuration Software

The configuration software is used to set up the characteristics of one or more PCIMs. The PCIM(s) must be present in the same computer to complete the configuration.

Setup parameters include the base addresses used by the PCIM and its daughterboard(s), baud rate, serial bus address, outputs enable, and Watchdog Timer enable. Configuration data is stored in EPROM memory on the PCIM, and is retained if power is removed.

The configuration software can be run from diskette or the configuration files can be copied to a hard disk.

It has a tutorial mode that can be used to practice entering data without actually changing the parameters of any installed PCIM.

The software configures one PCIM at a time. Follow the instructions to configure the first PCIM. Write the configuration to the first PCIM then exit the configuration software. Configurate the next PCIM by restarting the configuration software, specifying the second PCIM's DIP switch address in the command, as described below.

Notifying the Configuration Software of DIP Switch Change

The configuration software for the PCIM expects the default DIP Switch setting of 222hex. If you change the DIP switch setting (for example, when using multiple PCIMs), you must also inform the configuration software in either of the following ways:

D. For one or more PCIMs, you can set an address flag when entering the command to execute the configuration software, as described on the next page.

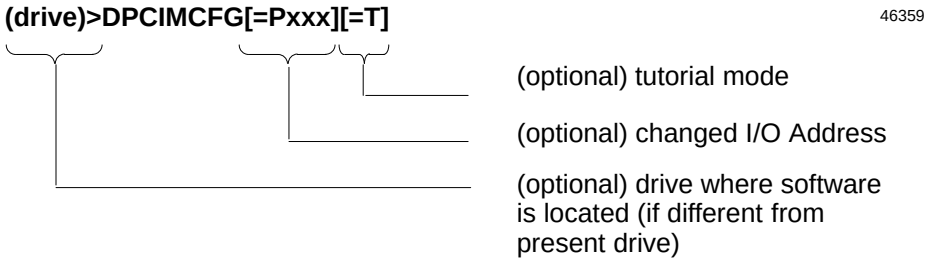
E. For just one PCIM, you can set a variable in the DOS startup file, AUTOEXEC.BAT. The format of the command is:

SET DPCIMCFG=xxx

where **XXX** is the new address.

Running the Configuration Software

With the configuration software diskette installed, or with the software files copied to your hard disk, type the following command at the DOS prompt:



The I/O Address, if entered, must match the DIP Switch setting on the PCIM board. If this flag is not used, or if the value entered is not in the range 100 hex to 3FF hex, the configuration software will use the default setting of 222hex.

Running the Program Normally

In normal mode, the software establishes communications with the PCIM. Therefore, the PCIM must already have its I/O Address assigned using the DIP switches. The software will look for it at its assigned address.

To run the configuration software in normal (not tutorial) mode, type:

DPCIMCFG to run the software without specifying a new address

or **DPCIMCFG=Pxxx** to run the software with a new PCIM address

The software establishes communications with the PCIM at the specified address. Complete the configuration entries as explained on the following pages.

Lack of Communications

If communications cannot be established, check the DIP switch settings. The DIP Switch address should match the address shown on the screen. For the default setting 222 hex, the correct DIP Switch settings are:

" # # # " # # #

If communications are disrupted, press any key to continue.

If you then want to exit the configuration software, press the ESC key.

If you want to reset the PCIM, press F10. Communications should be re-established.

Writing the Configuration Data to the PCIM

After you finish making the configuration entries for a PCIM, if you want to write them to the PCIM, press the F5 key. The configuration software then reads the new configuration data back from the PCIM and displays it. The "Last Update" field displays the time the configuration was updated.

If the two daughterboards have the same enabled memory or I/O Base Address, the update will NOT be performed.

If you want to read a configuration previously stored to the PCIM, press F3. If you want to reset the PCIM, press F10.

Use the ESC key when you are ready to exit the configuration utility.

Running the Program in Tutorial Mode

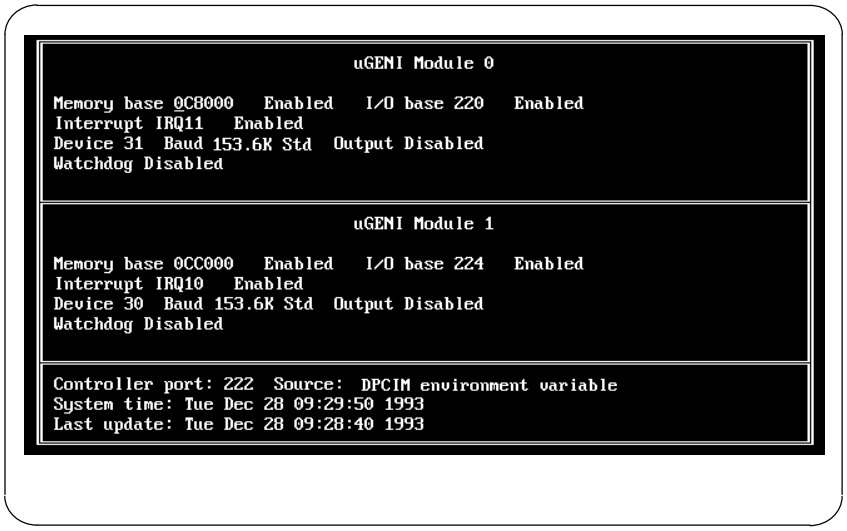
If you want to run the configuration software in tutorial mode but you do not want to change the I/O Address, type:

DPCIMCFG=T

In tutorial mode, the software supplies default values for the entries, and does not send entries you make to the PCIM.

Configuration Entries

The software displays the configuration for the specified PCIM's two daughterboards.



Use the Up Arrow and Down Arrow keys to move between configuration fields and between daughterboards.

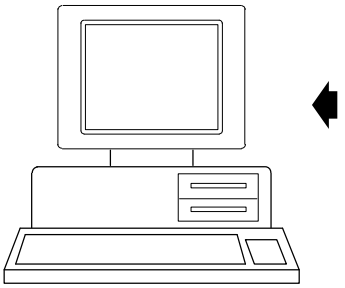
Type in entries where required (see the following table), or use the Right Arrow and Left Arrow keys to display the available choices for each field. A configuration example follows the table.

Definitions of Configuration Entries

Option	Entries / Choices	Comment
Memory base: Address Enabled / Disabled	If the daughterboard will be used, enter a hexadecimal address. The next field should be set to "Enabled". If the daughterboard will not be used, it is not necessary to enter an address. Select "Disabled".	Each daughterboard on a PCIM requires 16K of system memory. This memory is used to store I / O data, buffers for communications data, and a variety of other information. You can locate this memory anywhere space is available. The memory base address is truncated down to the nearest 16K boundary. That is, the fourth, fifth, and sixth digits of the hexadecimal address for the start of the memory space must be 0. The third digit must always be 0, 4, 8, or C. So valid memory addresses for the start of the block could be F4C000, 288000, 0E0000, etc...
I / O base: Address Enabled / Disabled	If the daughterboard will be used, enter a hexadecimal address. The next field should be set to "Enabled". If the daughterboard will not be used, it is not necessary to enter an address. Select "Disabled".	The I / O Base Address is a hexadecimal address used for PCIM command and status data (described in chapter 2). The address is truncated down to the nearest 4 byte boundary. That is, the third digit of the address must be a 0, 4, 8 or C.
Interrupt	Not used with the Software Library; select "Disable" when using the library routines. If not using the Software Library, select "Enabled" and enter the Interrupt: IRQ2 IRQ3 IRQ4 IRQ5 IRQ6 IRQ9 IRQ10 IRQ11 NMI (non-maskable interrupt)	IRQ2 and IRQ9 are the same pin on the motherboard connector. IRQ is used by PC / XTs. IRQ9 is used by PC.ATs and all new PCs. Either may be selected for configuration. Do not select IRQ3 if serial port 2 is installed. Do not select IRQ4 if serial port 1 is installed. Do not select IRQ5 if parallel port 2 is installed. Do not select IRQ6 if a diskette controller is present. IRQ9, 10, and 11 are only present on 286, 386, and 486 PCs. NMI is normally not selected. The two daughterboards on the PCIM may have different interrupt levels. They may also have the same interrupt level, if you plan to poll both boards with the same interrupt service routine.
Device	This is the Serial Bus Address. Enter a (decimal) number from 0 to 31.	Each device on a Genius bus must have a unique serial bus address. If two daughterboards on the same PCIM will be connected to the same bus, they are considered independent devices, and each must have a unique bus address.

Option	Entries / Choices	Comment
Baud	Specify the baud rate of the serial bus: 38.4K, 76.8K, 153.6K standard, or 153.6K extended.	All devices on the same bus must be set to use the same baud rate. If two daughterboards on the same PCIM are connected to separate busses that operate at different baud rates, their configuration should match the baud rates of the busses to which they are attached.
Outputs	Enabled or Disabled	This entry selects whether outputs from a PCIM daughterboard to other devices on the bus will be enabled at startup.
Watchdog Timer	Enabled or Disabled	This timer is not used with the Software Library, and should be disabled for Software Library applications. For other applications, it can be used to monitor the Host system and shut off the PCIM when the Host malfunctions. When the timer is enabled, you must pulse the timer input every 727 ms or the motherboard will reset the daughterboard. Chapter 2 explains how this is done. With the watchdog timer disabled no input from the Host system is needed. The other portions of the RESET circuit, the voltage detection and Host RESERDRV monitor, still provide RESET capability, even with the watchdog timer disabled.

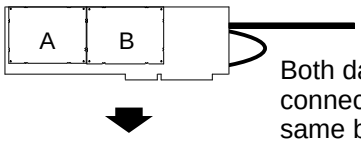
Configuration Example



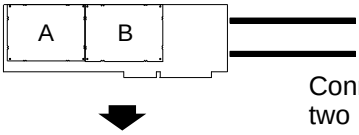
46360

Serial port 2 is installed.
Serial port 1 is installed.
Parallel port 2 is installed.
Diskette controller is present.

(These devices generate host interrupts, so their interrupt lines cannot be assigned to a PCIM.)



Both daughterboards connected to the same bus



Connected to two busses

PCIM 1 Configuration

Daughterboard A

Memory Base Address	0E0000
	Enabled
I/O Base Address	3E0
	Enabled
Interrupt	Disabled
Device	31
Baud	153.6K ext
Outputs	Enabled
Watchdog	Disabled

Daughterboard B

Memory Base Address	0E4000
	Enabled
I/O Base Address	3E4
	Enabled
Interrupt	Disabled
Device	30
Baud	153.6K ext
Outputs	Enabled
Watchdog	Disabled

PCIM 2 Configuration

Daughterboard A

Memory Base Address	0E8000
I/O Base Address	3E8
	Enabled
Interrupt	Disabled
	Enabled
Device	28
Baud	76.8K
Outputs	Enabled
Watchdog	Disabled

Daughterboard B

Memory Base Address	0EC000
	Enabled
I/O Base Address	3EC
	Enabled
Interrupt	Disabled
Device	28
Baud	153.6K std
Outputs	Enabled
Watchdog	Disabled

Chapter 4 C Programming for the PCIM

This chapter explains programming for a PCIM in C. Programming requires: C / MSDOS

Compiling your Application with Microsoft

In order to make your C application compatible with the PCIM library, you must first invoke the Microsoft compiler with the following switch (option):

```
/Zp
```

This option permits user–packed data structures and is required for the GetIMState, GetBusConfig, and GetDevConfig calls. For example:

```
C> msc application /Zp;      (small model)
```

OR

```
C> msc application /Zp/AL;  (large model)
```

Software File Linkage

It is necessary to link and load the file named "SPCIM.LIB" (small model) or "LPCIM.LIB" (large model) to use the C Software Drivers in your programs. There are several ways to link the PCIM.LIB using the Microsoft Linker.

1. The simplest way is to type all of the necessary module information on the command line:

   ```
   'LINK PROGRAM+MODULE,,,\SEARCH\PATH\SPCIM.LIB;'  (small model)
   ```

   OR

   ```
   'LINK PROGRAM+MODULE,,,\SEARCH\PATH\LPCIM.LIB;'  (large model)
   ```

2. However, if the program is divided up into several modules too numerous to fit on the command line, you can set up a response file to link all of the associated object files. The contents of a response file might look like:

   ```
   program+module1+module2+module3+
   module4+....+moduleN,
   program.exe,
   program.map,
   \search\path\pcim.lib
   ```

   The command to link the response file is:

   ```
   LINK @RESPONSE.FIL
   ```

Software Driver Function Calls

The PCIM Software Driver consists of easy to use macro–oriented function calls you code appropriately in your C language or Basic language applications routines. Function calls are summarized below.

Functions that deal with PCIM configuration:

InitIM – assigns PCIM numbers and Global data parameters to all PCIMs. Performs any required hardware activation and initialization of the PCIMs (such as Reset).

ChgIMSetup – writes to the Setup Table of the selected PCIM from the Host memory to change PCIM parameters.

GetIMState – reads PCIM configuration and status from the selected PCIM Status Table and Setup Table into Host memory.

Functions that deal with bus configuration:

GetBusConfig – reads all Device Configuration Tables from the selected PCIM into Host memory.

GetDevConfig – reads one device's configuration from the selected PCIM into Host memory.

DisableOut – writes to the Device Configuration Table of the selected PCIM to enable / disable outputs to selected devices or to all devices.

Functions that deal with control data movement:

GetBusIn – reads the entire Input Table (control data inputs) from a selected PCIM into Host memory.

PutBusOut – writes the entire Output Table (control data outputs) to a selected PCIM from Host memory.

GetDevIn – read control data inputs from a selected bus device into Host memory.

PutDevOut – write control data outputs to a selected bus device from Host memory.

GetIMIn – reads all PCIM control data from Directed Control Input Table of selected PCIM into Host memory.

PutIMOut – writes all PCIM control data to Global Data Table of selected PCIM from Host memory.

GetCir – reads an input circuit value (variable) into the Host memory from the Input Table of a selected PCIM.

GetWord – reads an input word value (variable) into the Host memory from the Input Table of a selected PCIM.

PutCir – writes an output circuit value (variable) from the Host memory to the Output Table of a selected PCIM.

PutWord – writes an output word value (variable) from the Host memory to the Output Table of a selected PCIM.

Functions that deal with communications:

GetMsg – reads a received message from a selected PCIM into Host memory.

SendMsg – writes a message from Host memory to the PCIM for transmission onto the bus.

SendMsgReply – writes a message from Host memory to the PCIM for transmission onto the bus and expects a specified reply message from the destination.

ChkMsgStat – allows the Host to detect when a transmitted message has actually been completed, or if transmission is incomplete or has failed.

Functions that deal with interrupt processing:

GetINTR – reads the entire Interrupt Status Table from a selected bus device into Host memory.

PutINTR – writes the entire Interrupt Status Table to a selected PCIM from Host memory.

Using Software Driver Function Calls

When coding the PCIM Software Drivers in your application programs, you should have at hand the following:

- Starting Address (Segment Address) of the Shared RAM Interface (or address of daughterboard).
- I/O Port Base Address.
- Status Table Address (PCIMs) or Reference Address (Series Six or Series Five PLC).
- Serial Bus Address of each bus device.
- Global, Input, Output Data lengths.

It is also helpful to have the *Genius I/O System and Communications User's Manual* (GEK–90486–1) handy for reference.

C Software Driver Function Call Parameters

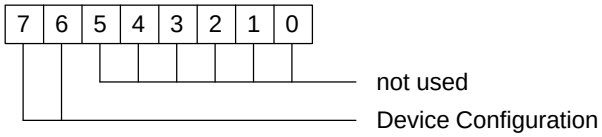
C Software Driver function calls require that you specify a number of parameters for each call. The data structures for each parameter, which are linked and loaded from your "pcim.h" file. Parameters are summarized on the pages following the pcim.h file.

PCIM.H File

The pcim.h file defines the data structures and macros used with the PCIM Software Library. The pcim.h file is listed below.

```
typedef
struct
{

    unsigned    int    Segment;        /* Starting address of SRI */
    unsigned    int    IOPort;         /*  I/O Port Base Address */

} IMBOARD;

/* The following template is to be used with
     the functions InitIM and the ChgIMSetup.
*/

typedef
struct
{
    IMBOARD     im;                    /* Board values for PCIM */
    unsigned    int    IMRef;          /* Status Table or
                                          Reference Address */
    unsigned    char   OutputLength;   /* Broadcast I/O Data Length */
    unsigned    char   InputLength;    /* Directed I/O Data Length */
    unsigned    char   Active;         /* Turn ON or OFF PCIM.  */

} IMPARMS;

/* The following Macros are to be used with
   the functions InitIM and the ChgIMSetup.
*/
 #define ON             1      /* Active set ON will enable the PCIM */
 #define OFF            0      /* Active set OFF will disable the PCIM  */
```

```
/* The following template is to be used with
    the function GetIMState
*/

typedef
struct   {
         unsigned    char     DipSwitch;      /* GENI Board Dip Switch value */
         unsigned    int      IMRef;          /* Reference Address */
         unsigned    char     OutputLength;   /* Output Control Data Length */
         unsigned    char     InputLength;    /* Input Control Data Length */
         unsigned    char     Revision;       /* GENI Revision Number */
         unsigned    char     GENI_OK;        /* every 200mS, set to one */
         unsigned    char     Fault;          /* Overall fault byte  */
         unsigned    char     Active;         /* Hand Held Monitor Present */
         unsigned    int      SBerr;          /* Serial Bus error count */
         unsigned    int      ScanTime;       /* Bus Scan Time in milliseconds */

} IMSTATE;

/*
     The following Macros are to be used with
     the function GetIMStatus and define the position of the Fault byte.
*/

#define RAMERR                  0
#define EPROMERR                1
#define CPUERR                  2
#define COMMERR                 3
#define SBAMASK                 0x1F
#define BAUDMASK                0x60
#define OUTPUTMASK              0x80

/*
     The following template is to be used with
     the function GetBusConfig.
*/

typedef
struct   {
         unsigned    char     Model;          /* Model Number of device */
         unsigned    char     OutputDisable;  /* Output Disable flag */
         unsigned    char     Present;        /* Device Present flag */
         unsigned    int      Reference;      /* Status Table or Reference Address */
         unsigned    char     InputLength;    /* Control Input Data Length */
         unsigned    char     Config          /* Device Configuration */
```




```
} DEVICE;
```

```
/*
     The following Macros are to be used with the function GetBusConfig.

*/
#define ENABLE              0       /* Enable Outputs to a Node */
#define DISABLE             1       /* Disable Outputs to a Node */
#define ALL                 32      /* Value to select all Nodes */
#define MAXDEVICE           32      /* Maximum devices per PCIM */
#define MAXIMS              64      /* Maximum number of PCIMs */
#define PRESENT             1       /* Device currently present on PCIM  */
#define NOTPRESENT          0       /* Device currently offline from PCIM */
#define INPUTO              1       /* Input Data Only Device */
#define OUTPUTO             2       /* Output Data Only Device */
#define COMBO               3       /* Combination of Input and Output Data */

/* The following templates are to be used with
   all of the MSG functions.
*/

typedef
struct   {
         unsigned    char    Source;         /* Source Address of Serial Bus Device */
         unsigned    char    Function;       /* Function Code */
         unsigned    char    SubFunction;    /* Sub Function Code */
         unsigned    char    DB Indicator;   /* Flag for Directed (1) or Broadcast (0) */
         unsigned    char    Length;         /* Length of Buffer */
         unsigned    char    Data[134];      /* Message buffer which will */

} READ_MESSAGE

typedef
struct   {
         unsigned    char    Destination;    /* Destination Address of Serial Bus Device */
         unsigned    char    Function;       /* Function Code */
         unsigned    char    SubFunction;    /* Sub Function Code */
         unsigned    char    Priority    ;   /* Priority (0 (normal) or 1 (high)) */
         unsigned    char    Length;         /* Data Buffer (134 bytes max) */
         unsigned    char    Data[134];      /* Message buffer which will */

} SEND_MESSAGE;

typedef
struct   {
         unsigned    char    Destination;    /* Destination Address of Serial Bus Device */
         unsigned    char    Function;       /* Function Code */
         unsigned    char    T_SubFunction;/* Sub Function Code (transmitted) *
         unsigned    char    R_SubFunction;/* Sub Function Code (received) */
         unsigned    char    Priority    ;   /* Priority (0 (normal) or 1 (high)) */
         unsigned    char    Length;         /* Trans. Data Buff. Length (134 bytes max) */
         unsigned    char    Data[134];      /* Message buffer which will */

} SEND_MESSAGE_REPLY;
```

```
/* The following Macros are used with all the _MESSAGE_ templates
*/

#define BROADCAST           255
#define NORMALP             0           /* Normal Priority */
#define HIGHP               1           /* High Priority */

 /*
     The following describes the tables necessary to read and write the PCIM's Interrupt
     Status Table and Interrupt Disable Table.
*/
#define I_ENABLE            0       /* Enable the interrupt level */
#define I_DISABLE           1       /* Disable the interrupt level */

#define I_SUMMARY           0       /* Summary set if interrupt occurred */
#define I_REQUEST_Q         1       /* Received memory datagram */
#define I_PCIM_STAT         2       /* PCIM Status Change – usually fatal */‘

#define  I_DEV_STAT         3       /* Device Status Change */
#define I_OUT_SENT          4       /* Outputs sent – end of bus access */
#define I_CCOMPLETE         5       /* Command Block complete */
#define I_RECEIVE_D         6       /* Received Datagram */
#define I_LOCKOUT           7       /* Lockout granted */

/*
The following Macros are used as Return values for all functions. The FAIL codes should
be listed with each individual function description.
*/

#define SUCCESS             0       /* Successful completion of function */
#define INITFAIL            1       /* Initialization Failure */
#define IMFAIL              2       /* PCIM Failure */
#define BADSEG              3       /* Invalid Segment address */
#define BADPORT             4       /* Invalid I/O Port Address */
#define BADCFG              5       /* Invalid Configuration parameter */
#define NOCFG               6       /* No Configuration changes found */
#define NOINIT              7       /* PCIM selected is not initialized */
#define NODATA              8       /* No data found */
#define UNDERFLOW           9       /* Insufficient device data length */
#define OVERFLOW            10      /* Exceeds device data length */
#define OFFLINE             11      /* Device is offline */
#define IMBUSY              12      /* PCIM busy */
#define BADPARM             13      /* Invalid message parameter */
#define TXERR               14      /* Message transmit failure */
#define NOMSG               15      /* No Message available */
#define IMFREE              16      /* No message activit */
#define BADSBA              17      /* Invalid Serial Bus Address */
#define BADIMNUM            18      /* Invalid PCIM Number */
#define PCIMERR             19      /* PCIM firmware problem */
#define DUPSEG              20      /* Duplicate segment values given */
#define DUPPORT             21      /* Duplicate I/O port values given */
```

Data Structures for Initialization

The following data structures may be defined for use with the IMPARMS function call:

Type	Name	Range	Description
unsigned int	im.Segment;	0–FFFE(h)	Starting address defined by configurationsoftware
unsigned int	im.IOport;	100(h) – 3FC(h)	I/OPort base address (DIP switch SW1)
unsigned int	IMRef;	0–8001 – 0–FFFF(h)	Global Data Address of PCIM daughterboard.
unsigned char	OutputLength;	0–128	Global Data length in BYTEs
unsigned char	InputLength;	0–128	Directed Input Data length (normally 0)
unsigned char	Active;	ON / OFF	Turn PCIM on or off (see ChgIMSetup)

Macros for Initialization and Setup Change

The following Macros are to be used with the variable Active in the functions InitIM and the ChgIMSetup. Any structures which do not indicate setting by Dipswitch (hardware actuated) are set by the Software Drivers (software actuated).

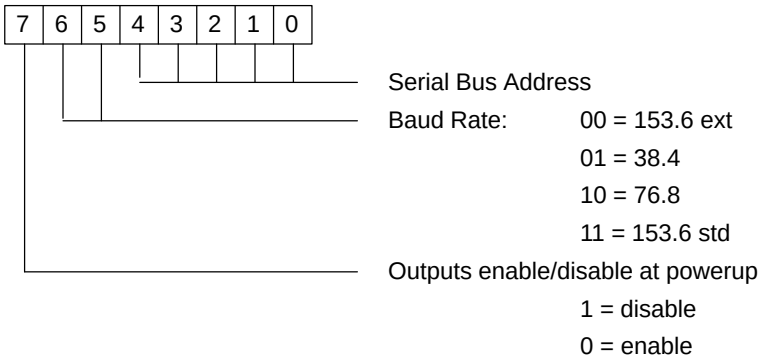
Macro	Value	Explanation
#define ON	1	Active set ON will enable the PCIM
#define OFF	0	Active set OFF will disable the PCIM

Data Structures for PCIM Configuration

The following data structures are to be used for the **GetIMState** function call.

Type	Name	Range	Description
unsigned char	DipSwitch;	0–255(d)	Daughterboard Dip Switch value. See below.
unsigned int	IMRef;	0–8001(d) or 0–FFFF(h)	Global Data Reference: Beginning address of the Global Data of the broadcasting CPU.
unsigned char	OutputLength;	0–128	Global Data Length: Number of bytes of Global Data to be broadcast by the PCIM.
unsigned char	InputLength;	0–128	Directed Input Data Length; normally 0.
unsigned char	Revision;		PCIM Firmware Revision Number.
unsigned char	GENI OK;	1 / 0	PCIM OK; every 200 ms, set to '1'.
unsigned char	Fault;	0–15	Overall fault byte: any PCIM fault shown below.
unsigned char	Active;	0–5	Hand Held Monitor Present – one or combination of bit positions: bit 0 = HHM present (1=pres) bit 1 = reserved bit 2 = 10 CRC errors in 10 seconds. On for one second, does not stop PCIM. This bit is et of 10 errors occur in 10 seconds.
unsigned int	SBerr;	0–FFFF FFFF–0	Serial Bus error count; roll over counter. Goes from 0 to FFFF to 0.
unsigned int	ScanTime;		Bus Scan Time in mS.

Content of DipSwitch is:



Macros for GetIMState

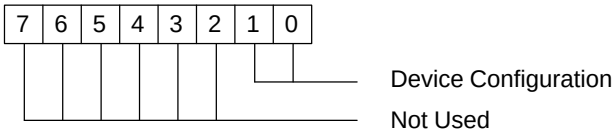
The following Macros are to be used with the variable **Fault** in the function **GetIMState**.

Macro	Value	Explanation
#define RAMERR	0	Random Access Memory error
#define EPROMERR	1	EPROM error
#define CPUERR	2	CPU error
#define COMMERR	3	Communications (Bus) error
#define SBAMASK	0x1F	Serial Bus Address mask
#define BAUDMASK	0x60	Baud Rate Mask
#define OUTPUTMASK	0x80	Output Enable / Disable mask

Data Structures for Bus Configuration

The following data structures are to be used for the **GetBusConfig** function call.

Type	Name	Range	Description
unsigned char	Model;	4–139	Model Number of serial bus device.
unsigned char	OutputDisable;	1 / 0	Output Disable flag values shown below.
unsigned char	Present;	1 / 0	Device Present flag shown below
unsigned int	Reference;	0–8001(d) 0–FFFF(h)	Global Data Reference: Beginning address of the Global Data of the broadcasting CPU.
unsigned char	InputLength;	0–128	Device input data length.
unsigned char	OutputLength;	0–128	Device output data length.
unsigned char Config;	1–3		Device Configuration as shown below.



Macros for GetBusConfig

The following Macros are to be used with the function **GetBus** Config.

Macro	Value	Explanation
In the variable Output Disable:		
#define ENABLE	0	Enable Outputs to a device
#define DISABLE	1	Disable Outputs to a device
#define ALL	32	Value to select all devices
#define MAXDEVICE	32	Maximum devices per PCIM
#define MAXIMS	64	Maximum number of PCIMs
In the value Present:		
#define PRESENT	1	Device Present on PCIM
#define NOTPRESENT	0	Device Offline from PCIM
In the value Config:		
#define INPUT	1	Input Data Only Device
#define OUTPUT	2	Output Data Only Device
#define COMBO	3	Input and Output Data Device

Data Structures for Communications: Read Message

The following data structures are to be used for the **ReadMsg** function call.

Type	Name	Range	Description
unsigned char	Source;	0–31	Serial Bus Address of device.
unsigned char	Function;	normally 20(h) for Genius messages	Function Code
unsigned char	SubFunction;	(hex value)	Sub Function Code. See GEK–90486–2
unsigned char	DB Indicator;	1 / 0	Message type: Directed (1) Broadcast(0).
unsigned char	Length;	0–134	Length of message in bytes.
unsigned char	Data[134];		Actual Message Data in bytes.

Data Structures for Communications: Send Message

The following data structures are to be used for the **SendMsg** function call.

Type	Name	Range	Description
unsigned char	Destination;	0–31 or broadcast	Serial Bus Address of device.
unsigned char	Function;	normally 20(h) for Genius messages	Function Code
unsigned char	SubFunction;	(hex value)	Sub Function Code. See GEK–90486–2
unsigned char	Priority;	1 / 0	Priority at which message is to be sent: Normal Priority(0). High priority (1)
unsigned char	Length;	0–134	Length of message in bytes.
unsigned char	Data[134];		Actual message data in bytes.

Data Structures for Communications: Send Message with Reply

The following data structures are to be used for the **SendMsgReply** function call.

Type	Name	Range	Description
unsigned char	Destination;	0–31	Serial Bus Address of device.
unsigned char	Function;	10 / 20(h)	Function Code
unsigned char	T_SubFunction;	(hex value)	Sub Function Code (transmitted message). See GEK–90486–2
unsigned char	R_SubFunction;	(hex value)	Sub Function Code (expected reply). See GEK–90486–2
unsigned char	Priority;	1 / 0	Priority at which message is to be sent: Normal Priority (0). High priority (1)
unsigned char	T_Length;	0–134	Length of message in bytes.
unsigned char	Data[134];		Actual message data in bytes.

Macros for the Message Functions

The following Macro is to be used by the **Destination** variable in the message structures.

Macro	Value	Explanation
#define BROADCAST	255	Message to be sent in broadcast mode.

The following Macros are to be used by the **Priority** variable in the message structures.

Macro	Value	Explanation
#define NORMALP	0	Message to be sent at normal priority.
#define HIGHP	1	Message to be sent at high priority.

Macros for Interrupts

The following Macros are used in the **PutINTR** and **GetINTR** function calls.

Macro	Value	Explanation
#define I_ENABLE	0	Enable the interrupt level.
#define I_DISABLE	1	Disable the interrupt level.
#define I_SUMMARY	0	Summary if interrupt occurred.
#define I_REQUEST_Q	1	Received memory datagram.
#define I_PCIM_STAT	2	PCIM Status Change – unless initiated by the host, this is usually fatal.
#define I_DEV_STAT	3	Device Status Change.
#define I_OUT_SENT	4	Outputs sent – end of bus access.
#define I_CCOMPLETE	5	Command Block completed.
#define I_RECEIVE_D	6	Received Datagram.

Miscellaneous Character Buffers and Integers

The following character buffers and integers are used in various calls:

Type	Name	Range	Description
int	IMcount;	1–64	Total number of PCIMs.
int	IMnum;	1–64	Relative number of PCIM.
int	Devicenum;	0–31	Specifies device on Serial Bus.
unsigned int	Offset;	1–1024	Specifies device on Serial Bus.
unsigned int	Worddata;	0–FFFF	Pointer to store the word requested.
char	IMflags;	0–63	Tells you which PCIMs initialized properly (or improperly).
char	Flag;	0 / 1	Enable / Disable outputs.
char	Datalngth;	0–128	Character pointer to size of data buffer.
char	DevData;	0–128	Character pointer to a buffer where data to be written will be located.
char	State;	0 / 1	ON or OFF condition of circuit read from PCIM.

Macros for Return Values for All Calls

The following Macros are used as as return values for all calls.

Macro	Value	Explanation
#define SUCCESS	0	Successful completion of function.
#define INITFAIL	1	Initialization Failure.
#define IMFAIL	2	PCIM Failure.
#define BADSEG	3	Invalid Segment address.
#define BADPORT	4	Invalid I/O Port Address.
#define BADCFG	5	Invalid Configuration parameter.
#define NOCFG	6	No Configuration changes found.
#define NOINIT	7	PCIM selected is not initialized.
#define NODATA	8	No data found.
#define UNDERFLOW	9	Insufficient device data length.
#define OVERFLOW	10	Exceeds device data length.
#define OFFLINE	11	Device is offline.
#define IMBUSY	12	PCIM busy.
#define BADPARM	13	Invalid message parameter.
#define TXERR	14	Message transmit failure.
#define NOMSG	15	No Message available.
#define IMFREE	16	No message activity.
#define BADSBA	17	Invalid Serial Bus Address.
#define BADIMNUM	18	Invalid PCIM Number.
#define PCIMERR	19	PCIM firmware problem.
#define DUPSEG	20	Duplicate segments given.
#define DUPPORT	21	Duplicate I/O port values given.

InitIM – Setup and Activate PCIM

Code Summary

```
#include <pcim.h>

int
InitIM (IMcount, IMparms, IMflags)

unsigned  int IMcount;
IMPARMS IMparms[ ];
unsigned char *IMflags;
```

Description

The Initialize PCIM call specifies the total number of PCIMs in the host system through the parameter "IMcount", and the characteristics of each PCIM through the parameter "IMparms".

InitIM resets the IMcount of PCIMs in the host system and initializes each PCIM as defined by IMparms. You must create a separate IMparms entry for each PCIM in IMcount.

The format of "IMPARMS" is:

IM 1 – Segment Address of PCIM shared RAM (two bytes LSB – MSB)
IM 1 – I/O Port Address (two bytes LSB – MSB)
IM 1 – PCIM Global Reference (two bytes LSB – MSB)
IM 1 – Global data length (one byte)
IM 1 – Input data length (input directed data length, normally 0)
IM 1 – Active (one byte) 1 = ON, 0 = OFF)

IM 2 – Segment Address of PCIM shared RAM (two bytes LSB – MSB)
IM 2 – I/O Port Address (two bytes LSB – MSB)
IM 2 – PCIM Global Reference (two bytes LSB – MSB)
IM 2 – Global data length (one byte)
IM 2 – Input data length (reserved – one byte always set to '0')
IM 2 – Active (one byte) 1 = ON, 0 = OFF)

etc...

Note

The memory pointer and I/O port assignments must correspond to the configuration of the PCIM.

The last parameter, "IMflags" is used by InitIM to tell you which of the PCIMs initialized properly (or improperly, as the case may be). The number of flags should equal IMcount.

Parameters are summarized as follows:

Parameter	Values	Function
IMcount	1–64	Total number of PCIMs
IMparms	varies	shows the characteristics of each module – see above
IMflags	varies	tells you which PCIMs initialized properly – see above

The InitIM call performs the following sequence of actions:

1. Brings each defined PCIM out of reset, or (if the PCIM is already running), into reset then out of reset.
2. Downloads Global data parameters to each PCIM after its PCIM OK LED turns ON (may take up to two seconds).
3. After all PCIMs have been downloaded or a two second timeout has occurred, returns with a 64 byte Status array (one byte for each defined PCIM). If any syntax or execution errors detected, the status returned is Fail. An example of an execution error is failure of the PCIM OK flag to be ON within two seconds after Reset.

Return Value (Status)

InitIM returns SUCCESS if all resets and data parameters are accepted by each PCIM. The following failure codes may be returned:

BADIMNUM	IMnum is out of range (a count of 64 or greater). No more InitIM processing is performed.
INITFAIL	An initialization problem occurred in one or more PCIM. The individual status for each PCIM on the bus is located in the IMflags parameter.

One of the following status codes will be returned in the appropriate location in the IMflags parameter if the return code is INITFAIL. Each status value in the IMflags array is unique to the associated PCIM and does not reflect the status of any other PCIM.

SUCCESS	This PCIM has been powered up and configured as specified.
IMFAIL	This PCIM never powered up.
BADCFG	This PCIM rejected the configuration because a parameter was out of range.
BADSEG	The segment value in IMparms is set to the illegal value 0 (zero)
BADPORT	The I/O port address is set to some illegal value less than 256.

Note

If any of the PCIMs fail to initialize as you have specified in IMparms, InitIM turns OFF the failed PCIM.

Coding Example

In this example there are two PCIMs.

```
 #include <pcim.h>

#define COUNT 2

int status; char IMflags[COUNT];
IMPARMS IMparms[COUNT];

    IMparms[0].im.Segment = 0xD000;      /* Shared RAM begins at D000(h) */
    IMparms[0].im.IOPort = 0x3E4;        /* Port Base Address at 3E4(h) */
    IMparms[0].IMRef = 0x7000;           /* PCIM Global Reference – 7000(h) */
    IMparms[0].OutputLength = 0;         /* No Global Data */
    IMparms[0].InputLength = 0;          /* No Directed Input data */
    IMparms[0].Active = ON;              /* Turn PCIM #1 ON by default */

    IMparms[1].im.Segment = 0xCC00;      /* Shared RAM begins at CC00(h) */
    IMparms[1].im.IOPort = 0x3E0;        /* Port Base Address at 3E0(h) */
    IMparms[1].IMRef = 0x8001;           /* PCIM Global Reference – 8001(h) */
    IMparms[1].OutputLength = 20;        /* No Global Data */
    IMparms[1].InputLength = 0;          /*  No Directed Input data */
    IMparms[1].Active = ON;              /* Turn PCIM #2 ON by default */

    status = InitIM (COUNT, IMparms, IMflags);
```

ChgIMSetup – Change PCIM Configuration

Code Summary

```
#include <pcim.h>

int
ChgIMSetup (IMnum, IMparms)

unsigned int IMnum;
IMPARMS *IMparms;
```

Description

Following initialization, any changes you make to the configuration of a specific PCIM must use the Change PCIM Setup call. This call allows you to make configuration changes to a specific PCIM Setup Table by writing the IMparms parameter from Host memory to it.

The "IMnum" parameter is an index to the IMparms array which, after initialization, indicates the specific PCIM in the host system for which configuration changes are intended.

Note

Configuration changes to any PCIM while online causes that PCIM to stop transmitting on the serial bus for 1.5 seconds.

The format of "IMPARMS" is:

IM 1 – Segment Address of PCIM shared RAM (two bytes LSB – MSB)
IM 1 – I/O Port Address (two bytes LSB – MSB)
IM 1 – PCIM Global Reference (two bytes LSB – MSB)
IM 1 – Global data length (one byte)
IM 1 – Input data length (reserved – one byte always set to '0')
IM 1 – Active (one byte) 1 = ON, 0 = OFF)

IM 2 – Segment Address of PCIM shared RAM (two bytes LSB – MSB)
IM 2 – I/O Port Address (two bytes LSB – MSB)
IM 2 – PCIM Global Reference (two bytes LSB – MSB)
IM 2 – Global data length (one byte)
IM 2 – Input data length (reserved – one byte always set to '0')
IM 2 – Active (one byte) 1 = ON, 0 = OFF)

etc...

Parameters are summarized as follows:

Parameter	Values	Function
IMnum	1–64	Relative number of PCIM
IMparms	varies	shows the characteristics of each module – see above

Return Value (Status)

ChgIMSetup will return SUCCESS if all changes were accepted by the target IM. If the PCIM fails to change to the new parameters, the following FAIL indications will be returned:

BADIMNUM	IMnum is out of range (a count of 64 or greater).
NOINIT	Indicated PCIM has not been initialized (InitIM).
IMFAIL	The indicated PCIM has failed (PCIM OK = 0), or never completed processing the config change command.
IMBUSY	The PCIM is otherwise engaged and cannot accept the config change command.
BADCFG	This PCIM rejected the configuration because a parameter was out of range.
NOCFG	The PCIM, after examining the received the config change command, found no changes to make.
INITFAIL	Change of Global Data output or Global Data reference or Directed Data input length required a reset of PCIM daughterboard and the daughterboard failed to reinitialize.

Coding Example

Change the PCIM Global Reference for PCIM #1.

```
#include <pcim.h>

#define COUNT 2

int status;
IMPARMS IMparms[COUNT];

        IMparms[0].IMref = 0x8010;

        status = ChgIMSetup (1, &IMparms );
```

Turn OFF PCIM #2.

```
#include <pcim.h>

int status;
IMPARMS IMparms[COUNT];

        IMparms[1].Active = OFF;

        status = ChgIMSetup (2, &IMparms );
```

GetIMState – Get Configuration and Status Information

Code Summary

```
#include <pcim.h>

int
GetIMState (IMnum, IMstate)

unsigned int IMnum;
IMSTATE *IMstate;
```

Description

The Get PCIM State call allows you to access configuration and status information about a specific PCIM by reading its Setup Table and Status Table into the "IMstate" parameter in Host memory.

The format of IMstate is:

DipSwitch	– Daugherboard Dip Switch Value
IMRef	– Reference Address
OutputLength	– Output Control Data Length
InputLength	– Input Control Data Length
Revision	– PCIM Firmware Revision Number
GENI OK	– PCIM OK – every 200 ms, set to '1'. If 0, board has faulted.
Fault	– Overall fault byte – any PCIM fault
Active	– Hand Held Monitor Present
SBerr	– Serial Bus error count
ScanTime	– Bus Scan Time in ms

Since the PCIM periodically sets its PCIM OK flag, this call allows the implementation of a PCIM OK heartbeat procedure.

Parameters are summarized as follows:

Parameter	Values	Function
IMnum	1–64	Relative number of PCIM
IMstate	varies	PCIM Configuration and Status – see above

Return Value (Status)

GetIMState will almost always return SUCCESS. If the target PCIM is currently offline, has not been initialized, or is out of range, the following FAIL indications will be returned:

BADIMNUM	– IMnum is out of range (a count of 64 or greater).
NOINIT	– Indicated PCIM has not been initialized (InitIM).
IMFAIL	– The indicated PCIM has failed (PCIM OK = 0).

Coding Example

Examine the state of PCIM #1.

```
#include <pcim.h>

int status;
IMSTATE IMstate;

    status = GetIMState (1, &IMstate);
```

GetBusConfig– Get Serial Bus Configuration

Code Summary

```
#include <pcim.h>

int
GetBusConfig (IMnum, Config)

unsigned int IMnum;
DEVICE Config [ ];
```

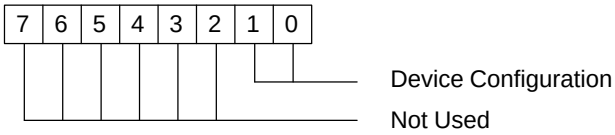
Description

The Get Bus Configuration call allows you to read device configuration information about all devices on a serial bus (except the PCIM). GetBusConfig reads all 32 Device Configuration Tables from the PCIM selected by IMnum into the Host memory"Config" parameter. This information is not packed and will fill the entire Config parm – 256 bytes in length.

The format of Config is:

unsigned char Model	– Model Number of device
unsigned char OutputDisable	– Output disable flag
unsigned char Present	– Device Present flag
unsigned int Reference	– Status Table or Reference Address
unsigned char InputLength	– Control Input Data Length
unsigned char OutputLength	– Control Output Data Length
unsigned char Config	– Device Configuration

1 = all inputs
2 = all outputs
3 = combination



Parameters are summarized as follows:

Parameter	Values	Function
IMnum	1–64	Relative number of PCIM
Config	256 bytes	Device configuration information about all devices on a serial bus – see above

Return Value (Status)

GetBusConfig will almost always return SUCCESS. If the target PCIM is currently offline, has not been initialized, or is out of range, the following FAIL indications will be returned:

BADIMNUM	IMnum is out of range (a count of 64 or greater).
NOINIT	Indicated PCIM has not been initialized (InitIM).
IMFAIL	The indicated PCIM has failed (PCIM OK = 0).
OFFLINE	No devices are currently active on the bus. However, the appropriate buffer is still returned and will contain configuration data for devices once logged in. Zeros will be returned if no device has logged in to a particular slot.

Coding Example

Examine the configuration of the devices on PCIM #1.

```
#include <pcim.h>

int status;
DEVICE Config[MAXDEVICE];

        status = GetBusConfig (1, Config);
```

GetDevConfig – Get Device Configuration

Code Summary

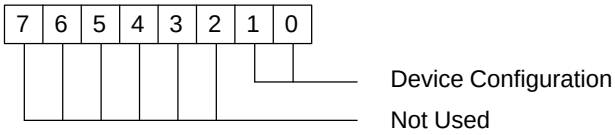
```
#include <pcim.h>

int
GetDevConfig (IMnum, Devicenum, Config)

unsigned int IMnum;
unsigned int Devicenum;
DEVICE *Config;
```

Description

The Get Device Configuration call allows you to read device configuration information about a specific device on the serial bus. GetDevConfig reads this information from the PCIM selected by IMnum into the Host memory "Config" parameter, which should point to a character buffer with the format of one DEVICE structure.

Again, the format of Config is:

```
unsigned char Model          – Model Number of device
unsigned char OutputDisable  – Output disable flag
unsigned char Present        – Device Present flag
unsigned int  Reference      – Status Table or Reference Address
unsigned char InputLength    – Control Input Data Length
unsigned char OutputLength   – Control Output Data Length
unsigned char Config         – Device Configuration
                                 1 = all inputs
                                 2 = all outputs
                                 3 = combination
```



Parameters are summarized as follows:

Parameter	Values	Function
IMnum	1–64	Relative number of PCIM
Devicenum	0–31	Specifies device on serial bus
Config	8 bytes	Device configuration information about all devices on a serial bus – see above

Return Value (Status)

GetDevConfig will almost always return SUCCESS. If the target PCIM is currently offline, has not been initialized, or is out of range, the following FAIL indications will be returned:

BADIMNUM	IMnum is out of range (a count of 64 or greater).
BADSBA	Specified Devicenum is not in the range for Genius bus devices (0 –31 decimal).
NOINIT	Indicated PCIM has not been initialized (InitIM).
IMFAIL	The indicated PCIM has failed (PCIM OK = 0), or never completed processing the config change command.
OFFLINE	The device requested is currently not on the bus, however, the appropriate buffer is still returned and will contain configuration data for devices once logged in.

Coding Example

Examine the configuration of device #30 on PCIM #1.

```
#include <pcim.h>

int status;
DEVICE Configbuf;

    status = GetDevConfig (1, 30, Configbuf);
```

DisableOut – Disable/Enable Device Outputs

Summary

```
#include <pcim.h>

 int
DisableOut (IMnum, Devicenum, Flag)

unsigned int IMnum, Devicenum;
char Flag;
```

Description

The Disable (/Enable) Outputs call allows you to selectively disable (or enable) outputs to a specific device, or to all devices, on a serial bus.

If Flag is non–zero ('1'), outputs to the device will be disabled; if Flag is zero ('0'), outputs will be enabled to that device. If you code the Devicenum value equal to 'ALL', then the outputs to all devices will be set to the value of Flag. If Devicenum is a serial bus address value between 0 – 31 decimal, then the flag value will only affect that device. PCIM.H contains macros defined for ON or OFF values for Flag.

Parameters are summarized as follows:

Parameter	Values	Function
IMnum	1–64	Relative number of PCIM
Devicenum	0–31 32	Specifies device on serial bus on which circuit resides Specifies all devices
Flag	0 or 1	Enable/disable outputs

Return Value (Status)

DisableOut will return SUCCESS if the device specified by IMnum is present on the serial bus. Otherwise, DisableOut will return FAIL. If Devicenum indicates ALL, then DisableOut will almost always return SUCCESS. The following FAIL indications will be returned:

BADIMNUM	IMnum is out of range (a count of 64 or greater).
BADSBA	Specified Devicenum is not in the range for Genius bus devices (0 – 31 decimal), or is the serial bus address of the daughterboard.
NOINIT	Indicated PCIM has not been initialized (InitIM).
IMFAIL	The indicated PCIM has failed (PCIM OK = 0).

Coding Example

Enable outputs to device #8 on PCIM #1.

```
#include <pcim.h>
int status;
    status = DisableOut (1, 8, ENABLE);
```

Disable outputs to all devices on PCIM #1.

```
#include <pcim.h>
int status;
    status = DisableOut (2, ALL, DISABLE);
```

GetBusIn – Read all Input Values

Code Summary

```
#include <pcim.h>

int
GetBusIn (IMnum, IOdata)

unsigned int IMnum;
unsigned char *IOdata;
```

Description

A Get Bus Inputs call allows you to read input values from all active devices in the Input Table of the specified PCIM. Active inputs are those for which the Device Present flag is set to '1' (it is the application's responsibility to know which devices are present on the bus via the GetBusConfig call). Active input values are placed into the Host memory "IOdata" parameter. IOdata must point to a 4096–byte buffer where the I/O information will be saved. The IOdata parm has the same format as the Input Table – 32 slots of 128 bytes each. Slots are in serial bus address order.

When GetBusIn is called, it begins by "locking out" the PCIM from updating its Input Table (ensures data coherency across bus scans). GetBusIn then transfers the entire Input Table to the IOdata parameter, even for devices that are not active. When the entire PCIM Input Table has been searched, the PCIM is "unlocked".

Parameters are summarized as follows:

Parameter	Values	Function
IMnum	1–64	Relative number of PCIM
IOdata	4096 bytes	Data parameter will be copied from Host memory to specified PCIM

Return Value (Status)

GetBusIn will return SUCCESS if any of the devices specified by the IMnum are active and data was transferred. If no devices are present on the target IM, if the target PCIM is currently offline, has not been initialized, or is out of range, the following FAIL indications will be returned:

BADIMNUM	IMnum is out of range (a count of 64 or greater).
NOINIT	Indicated PCIM has not been initialized (InitIM).
IMFAIL	The indicated PCIM has failed (PCIM OK = 0).
OFFLINE	No devices are currently on the bus, however, the appropriate buffer is still returned and will contain input data for devices once logged in.

Coding Example

Read all inputs from all active devices on PCIM #1.

```
#include <pcim.h>

int status;
unsigned char IOdata;

    status = GetBusIn (1, IOdata);
```

PutBusOut – Write all Output Values

Code Summary

```
#include <pcim.h>

int
PutBusOut (IMnum, IOdata)

unsigned int IMnum;
unsigned char *IOdata;
```

Description

The Put Bus Outputs call allows you to update outputs to all devices in the Output Table of the specified PCIM. All output values are written from the Host memory IOdata parameter. IOdata must point to a 4096–byte buffer where the I/O information is saved. The IOdata parm has the same format as the Output Table – 32 slots of 128 bytes each. Slots are in serial bus address order.

When PutBusOut is called, it begins by "locking–out" the PCIM from updating its Output Table (ensures data coherency across PCIM scans). PutBusOut then transfers all data from IODATA to the Output Tables. When the entire PCIM Output Table has been searched, the PCIM is "unlocked".

Parameters are summarized as follows:

Parameter	Values	Function
IMnum	1–64	Relative number of PCIM
IOdata	4096 bytes	Data parameter will be copied from Host memory to specified PCIM

Return Value (Status)

PutBusOut will return SUCCESS if any of the devices specified by the IMnum are active and data was transferred. If no devices are present on the target IM, if the target PCIM is currently offline, has not been initialized, or is out of range, the following FAIL indications will be returned:

BADIMNUM	– IMnum is out of range (a count of 64 or greater).
NOINIT	– Indicated PCIM has not been initialized (InitIM).
IMFAIL	– The indicated PCIM has failed (PCIM OK = 0).
OFFLINE	– Data was transferred to the Output Tables, however, no devices were found on the bus.

Coding Example

Write all outputs to all active devices on PCIM #1.

```
#include <pcim.h>

int status;
unsigned char IOdata[4096];

  IOdata[128] = 1;
  IOdata[256] = 2;
  IOdata[384] = 4;
  IOdata[512] = 8;
  IOdata[640] = 0x10h;

  status = PutBusOut (1, IOdata);
```

GetDevIn – Read Device Data Only

Code Summary

```
#include <pcim.h>

int
GetDevIn (IMnum, Devicenum, DataLngth, Devdata)
unsigned int IMnum, Device;
unsigned char *DataLngth, *Devdata;
```

Description

The GetDevIn function allows you to read the control data inputs received from a single serial bus device into the Host memory "Devdata" parameter.

IMnum is the PCIM number configured during initialization. The Devicenum parameter specifies the serial bus address of the device from which input data is to be written. The "DataLngth" parameter points to the location where the number of data bytes read from device is to be stored. The "Devdata" parameter is a character pointer to a buffer where the data to be written will be located. The size of this buffer is determined by the "InputLength" parameter located in the device's configuration data.

Parameters are summarized as follows:

Parameter	Values	Function
IMnum	1–64	Relative number of PCIM
Devicenum	0–31	Specifies device on serial bus from which input word will be read
DataLngth	0–128	Character pointer to size of data buffer
Devdata	variable	Character pointer to a buffer where the data to be read from device is written – see above

Return Value (Status)

GetDevIn will return SUCCESS if the device specified by IMnum is present on the serial bus, and after the data is transferred to the DevData buffer. If the target device is not present, or is out of range, the following FAIL indications will be returned:

BADIMNUM	IMnum is out of range (a count of 64 or greater).
BADSBA	Specified Devicenum is not in the range for Genius bus devices (0 –31 decimal), or is that of the PCIM itself.
NOINIT	Indicated PCIM has not been initialized (InitIM).
IMFAIL	The indicated PCIM has failed (PCIM OK = 0).
OFFLINE	The device requested is currently not on the bus, and data is NOT transferred.

Coding Example

Get the inputs from device #8 on PCIM #1.

```
#include <pcim.h>

int status;
unsigned char Devdata[expected data length];
              Length;

  status = GetDevIn (1, 8, &Length, Devdata);
```

PutDevOut – Write Device Data Only

Code Summary

```
#include <pcim.h>

int
PutDevOut (IMnum, Devicenum, DataLngth, Devdata)

unsigned int IMnum, Device;
unsigned char DataLngth, *Devdata;
```

Description

The PutDevOut call allows you to write all of the control data outputs to a single serial bus device from the Host memory Devdata parameter.

IMnum is the PCIM number configured during initialization. The Devicenum parameter specifies the serial bus address of the device to which output data is to be written. The DataLngth parameter is the number of data bytes to write. If the value differs from the PCIMs current data base, an Overflow or Underflow error will be returned. The Devdata parameter is a character pointer to a buffer where the data to be written is located. The size of this buffer is determined by the "OutputLength" parameter located in the device's configuration data.

Parameters are summarized as follows:

Parameter	Values	Function
IMnum	1–64	Relative number of PCIM
Devicenum	0–31	Specifies device to which output word will be written
Datalngth	0–128	Character size of data buffer in bytes
Devdata	variable	Character pointer to a buffer where the data to be written will be located – see above

Return Value (Status)

PutDevOut will return SUCCESS if the device indicated is present on the given IMnum and after the data is transferred to that device. If the target device is not present, or is out of range, the following FAIL indications will be returned:

BADIMNUM	IMnum is out of range (a count of 64 or greater).
BADSBA	Specified Devicenum is not in the range for Genius bus devices (0 –31 decimal), or is that of the PCIM.
NOINIT	Indicated PCIM has not been initialized (InitIM).
IMFAIL	The indicated PCIM has failed (PCIM OK = 0).
OFFLINE	The device requested is currently not on the bus, and data is NOT transferred.
OVERFLOW	Specified DataLngth is greater than actual device's output length.
UNDERFLOW	Specified DataLngth is less than actual device's output length.

Coding Example

Write 2 bytes of output data to device #8 on PCIM #1.

```
#include <pcim.h>

int status;
unsigned char Devdata[data length];

  Devdata[0]  = 1;
  Devdata[1]  = 0x10;

  status = PutDevOut (1, 8, 2, Devdata);
```

GetIMIn – Read Directed Input Table

Code Summary

```
#include <pcim.h>

int
GetIMIn (IMnum, IMdata)

unsigned int IMnum;
unsigned char *IMdata;
```

Description

The Get IM Inputs call allows you to read the Directed Control Input Table of a specified PCIM and write its contents into the Host memory "IMdata" parameter.

IMnum is the PCIM number configured during initialization. The "IMdata" parameter is a buffer where the data to be read will be located. The size of this buffer is determined by the "InputLength" parameter located in the PCIMs configuration data.

When GetIMIn is called, it begins by "Locking–out" the PCIM from updating the Directed Control Input Table (ensures data coherency across bus scans). GetIMIn then transfers all the data in this table into Host memory. Once the transfer is complete, the PCIM is "unlocked".

Parameters are summarized as follows:

Parameter	Values	Function
IMnum	1–64	Relative number of PCIM
IMdata	variable	Buffer where the data read will be located – see above

Return Value (Status)

GETIMIN will return SUCCESS if the InputLength is non–zero and the data transfer is complete. The following FAIL indications will be returned:

BADIMNUM	– IMnum is out of range (a count of 64 or greater).
NOINIT	– Indicated PCIM has not been initialized (InitIM).
IMFAIL	– The indicated PCIM has failed (PCIM OK = 0).
UNDERFLOW	– The InputLength of the PCIM is set to zero (0).

PutIMOut – Write the Global Output Table

Code Summary

```
#include <pcim.h>

int
PutIMOut (IMnum, IMdata)

unsigned int IMnum;
char *IMdata;
```

Description

The PutIMOut call allows you to write Global Data from the Host memory IMdata parameter to the Global Data Output Table of a specified PCIM. This data is subsequently broadcast to all CPUs on the bus every bus scan.

IMnum is the PCIM number configured during initialization. The IMdata parameter is a character pointer to a buffer where the data to be written is located. The size of this buffer is determined by the "OutputLength" (Global Data Length) parameter located in the PCIM's configuration data.

When PutIMOut is called, it begins by "locking–out" the PCIM from reading from its Output Table (ensures data coherency across bus scans). PutIMOut then transfers all the data from this parm to the PCIMs Global Output buffer. Once the transfer is complete, the PCIM is "unlocked".

Parameters are summarized as follows:

Parameter	Values	Function
IMnum	1–64	Relative number of PCIM
IMdata	variable	Character pointer to a buffer where the data is located. Length of buffer is equal to output length as specified in InitIM.

Return Value (Status)

PutIMOut will return SUCCESS if the Global Data Length is non–zero and the transfer is complete. The following FAIL indications will be returned:

BADIMNUM	– IMnum is out of range (a count of 64 or greater).
NOINIT	– Indicated PCIM has not been initialized (InitIM).
IMFAIL	– The indicated PCIM has failed (PCIM OK = 0).
UNDERFLOW	– The GlobalLength parameter in IMPARMS is set to zero (0).

Coding Example

Write the specified Global Data to PCIM #1.

```
#include <pcim.h>

 int status;
char IMdata[128];

        IMdata [2] = 0x10;
        status = PutIMOut (1, IMdata);
```

GetCir – Read Input Circuit Value

Code Summary

```
#include <pcim.h>

int
GetCir (IMnum, Devicenum, Offset, State)

unsigned int IMnum, Devicenum;
unsigned int Offset;
char *State;
```

Description

A Get Circuit call allows the state of a single input circuit to be read from the specified PCIMs Input Table and be placed into the Host memory "State" parameter.

IMnum is the PCIM number configured during initialization. The Devicenum parameter specifies the serial bus address of the device which contains the input circuit. The "Offset" parameter indicates which bit of Devicenum is to be read. This value ranges from 1 through 1024 (in bits).

"State" is a character pointer in which GetCir will store the value of the circuit as indicated by the above parameters. The contents of State will be either a '1' or '0' (ON or OFF).

Parameters are summarized as follows:

Parameter	Values	Function
IMnum	1–64	Relative number of PCIM
Devicenum	0–31	Specifies I/O device from which input circuit will be read
Offset	1–1024	Input circuit offset in specified I/O device, in bits
State	0/1	ON or OFF condition of circuit read from PCIM

Return Value (Status)

GetCir will return SUCCESS if the target device is present on the given IMnum. If the target device is not present, or is out of range, GetCir will return FAIL. If SUCCESS is returned, then the character pointed to by State will contain the value of the circuit requested. The following FAIL indications will be returned:

BADIMNUM	IMnum is out of range (a count of 64 or greater).
BADSBA	Specified Devicenum is not in the range for Genius bus devices (0 –31 decimal), or is that of the PCIM.
NOINIT	Indicated PCIM has not been initialized (InitIM).
IMFAIL	The indicated PCIM has failed (PCIM OK = 0).
OFFLINE	The device requested is currently not on the bus, and data is NOT transferred.
OVERFLOW	The Offset specified is greater than the devices InputLength in circuits.
UNDERFLOW	The Offset is specified as zero (0).

Coding Example

Get the State value of circuit 2 of device #8 on PCIM #1.

```
#include <pcim.h>

int status;
char State;

    status = GetCir (1, 8, 2, &State);
```

PutCir – Write Output Circuit Value

Code Summary

```
#include <pcim.h>

int
PutCir (IMnum, Devicenum, Offset, State)

unsigned int IMnum, Devicenum;
unsigned int Offset;
char State;
```

Description

A Put Circuit call allows the state of a single output circuit to be changed from ON to OFF or vice–versa. In this call, the State parameter is written from the Host memory to the specified PCIMs Output Table.

IMnum is the PCIM number configured during initialization. The Devicenum parameter specifies the serial bus address of the device which contains the target output circuit. The Offset parameter indicates which bit of Devicenum is to be written. This value ranges from 1 through 1024 (in bits).

State is a character in which PutCir will use as desired the value of the circuit. The contents of State should be either a '1' or '0' (ON or OFF).

Parameters are summarized as follows:

Parameter	Values	Function
IMnum	1–64	Relative number of PCIM
Devicenum	0–31	Specifies I/O device to which output circuit will be written.
Offset	1–1024	Output circuit offset in specified I/O device, in bits
State	0 /1	Variable "State" is written from the Host memory to the specified PCIM

Return Value (Status)

PutCir will return SUCCESS if the target device is present on the given IMnum. If the target device is not present, or is out of range, PutCir will return FAIL. The following FAIL indications will be returned:

BADIMNUM	IMnum is out of range (a count of 64 or greater).
BADSBA	Specified Devicenum is not in the range for Genius bus devices (0 –31 decimal), or is that of the PCIM.
NOINIT	Indicated PCIM has not been initialized (InitIM).
IMFAIL	The indicated PCIM has failed (PCIM OK = 0).
OFFLINE	The device requested is currently not on the bus, and data is NOT transferred.
OVERFLOW	The Offset specified is greater than the devices Output-Length in circuits.
UNDERFLOW	The Offset is specified as zero (0).

Coding Example

Set the State value of circuit 2 of device #8 on PCIM #1 to '1'.

```
#include <pcim.h>

int status;

    status = PutCir (1, 8, 2, (Char) 1);
```

GetWord – Read Input Word Value

Code Summary

```
#include <pcim.h>

int
GetWord (IMnum, Devicenum, Offset, Worddata)

unsigned int IMnum, Devicenum;
unsigned int Offset;
unsigned int *Worddata;
```

Description

A Get Word call allows you to read the value of a single input word from the specified PCIM's Input Table into the Host memory "Worddata" parameter. The "Worddata" parameter is an integer pointer which GetWord uses to store the word requested.

IMnum is the PCIM number configured during initialization. The Devicenum parameter specifies the serial bus address of the device where the input word is located. The Offset parameter indicates which word of the specified device is to be read. This value ranges from 1 through 64 (in word quantities).

When GetWord is called, it begins by "locking–out" the PCIM from updating the Shared RAM (ensures data coherency across bus scans). GetWord then transfers the word data into Host memory. Once the transfer is complete, the PCIM is "unlocked".

Parameters are summarized as follows:

Parameter	Values	Function
IMnum	1–64	Relative number of PCIM
Devicenum	0–31	Specifies I/O device from which input word will be read
Offset	1–64	Input word offset in specified I/O device, in words
Worddata	2 bytes	Integer pointer used to store the word requested – see above

Return Value (Status)

GetWord will return SUCCESS if the device specified by IMnum is present on the serial bus, and after the data is transferred to the DevData buffer. If the target device is not present, or is out of range, GetWord will return FAIL. If SUCCESS is returned, then the requested word value will be saved in the location pointed to by Worddata. The following FAIL indications will be returned:

BADIMNUM	IMnum is out of range (a count of 64 or greater).
BADSBA	Specified Devicenum is not in the range for Genius bus devices (0 –31 decimal), or is that of the PCIM.
NOINIT	Indicated PCIM has not been initialized (InitIM).
IMFAIL	The indicated PCIM has failed (PCIM OK = 0).
OFFLINE	The device requested is currently not on the bus, and data is NOT transferred.
OVERFLOW	The Offset specified is greater than the devices InputLength in circuits.
UNDERFLOW	The Offset is specified as zero (0).

Coding Example

Get the first word of device #8 on PCIM #1.

```
#include <pcim.h>

int status;
unsigned int Worddata;

        status = GetWord (1, 8, 1, &Worddata);
```

PutWord – Write Output Word Value

Code Summary

```
#include <pcim.h>

int
PutWord (IMnum, Devicenum, Offset, Worddata)

unsigned int IMnum, Devicenum;
unsigned int Offset, Worddata;
```

Description

A Put Word call allows you to write a single output word from the Host memory Worddata parameter to the specified PCIMs Output Table. The Worddata parameter is an integer which PutWord uses for the word to be transmitted.

IMnum is the PCIM number configured during initialization. The Devicenum parameter specifies the serial bus address of the device where the output word is to be sent. The Offset parameter indicates which word of the specified device is to be written. This value ranges from 1 through 64 (in word quantities).

When PutWord is called, it begins by "locking–out" the PCIM from updating the Shared RAM (ensures data coherency across bus scans). PutWord then transfers the word data to the device. Once the transfer is complete, the PCIM is "unlocked".

Parameters are summarized as follows:

Parameter	Values	Function
IMnum	1–64	Relative number of PCIM
Devicenum	0–31	Specifies device to which output word will be written
Offset	1–64	Output word offset in specified device, in words
Worddata	2 bytes	Integer used to store the word requested – see above

Return Value (Status)

PutWord will return SUCCESS if the device specified by IMnum is present on the serial bus. If the target device is not present, or is out of range, PutWord will return FAIL. The following FAIL indications will be returned:

BADIMNUM	IMnum is out of range (a count of 64 or greater).
BADSBA	Specified Devicenum is not in the range for Genius bus devices (0 –31 decimal), or is that of the PCIM.
NOINIT	Indicated PCIM has not been initialized (InitIM).
IMFAIL	The indicated PCIM has failed (PCIM OK = 0).
OFFLINE	The device requested is currently not on the bus, and data is NOT transferred.
OVERFLOW	The Offset specified is greater than the devices OutputLength in circuits.
UNDERFLOW	The Offset is specified as zero (0).

Coding Example

Set the second word of device #8 on PCIM #1 to 10 hex.

```
#include <pcim.h>

int status;

    status = PutWord (1, 8, 2, 0x10);
```

SendMsg – Send a Message

Code Summary

```
#include <pcim.h>

int
SendMsg (IMnum, Msg)

int IMnum;
SEND_MESSAGE *Msg;
```

Description

The Send Message call allows you to write a memory or non–memory message from the Host to the selected PCIM for transmission onto the serial bus (using the Transmit Datagram command). SendMsg will return control to the calling program without delay, before the message has been processed or transmitted by the PCIM.

IMnum defines the PCIM, as configured during initialization, from which to transmit the message. The Msg parameter is a pointer to the buffer where the transmit message is stored.

The format of SEND MESSAGE is:

Destination (0–31/255 brdcst)	– Destination address of Device
Function code (0–111)	– Function Code (normally 20 hex)
SubFunction code (0–255)	– Sub Function Code
Priority	– 0 – Normal, 1 – High
Length	– Data field length/length of message
Data (0–134)	– Message Data – depends on length parm

You should check the status of the message using ChkMsgStat to determine if the message completed processing properly.

Parameters are summarized as follows:

Parameter	Values	Function
IMnum	1–64	Relative number of PCIM
Msg	see above	Pointer to the buffer where the transmitted message will be stored – see above

Return Value (Status)

SendMsg will return SUCCESS if a message has been transferred from the Host memory to the PCIM. Otherwise, one of the following FAIL indications will be returned:

BADIMNUM	– IMnum is out of range (a count of 64 or greater).
NOINIT	– Indicated PCIM has not been initialized (InitIM).
IMFAIL	– The indicated PCIM has failed (PCIM OK = 0).
IMBUSY	– The PCIM is otherwise engaged and cannot accept the command.

Note

You are responsible for defining the device, the Function code, the Sub–F unction code and the length of the transmit Datagram.

Note

You cannot issue a SendMsg call or read a received unsolicited message while a SendMsgReply call is in progress. If this presents a timing problem, use the SendMsg call.

See Also: SendMsgReply, GetMsg and ChkMsgStat

Coding Example

Send a Read Diagnostics message to device #8 on PCIM #1. This message will read 10 bytes of diagnostic data beginning at offset 0.

```
#include <pcim.h>

int status;
SEND_MESSAGE Msg;

        Msg.Destination = 8;          /*Device #8*/
        Msg.Function = 0x20;          /*Genius Function Code*/
        Msg.SubFunction = 8;          /*Read Diagnostic Subfunction Code*/
        Msg.Priority = NORMALP;       /*Transmit at Normal priority*/
        Msg.Length = 2;               /*Length of data in Data Buffer*/
        Msg.Data[0] = 0;              /*Offset of 0*/
        Msg.Data[1] = 0xA;            /*Length of 10 decimal*/

status = SendMsg (1, &Msg);
```

To see how message function calls work together, see Appendix A, Example 2.

SendMsgReply – Send a Message Requesting a Reply

Summary

```
#include <pcim.h>

int
SendMsgReply (IMnum, Msg)

int IMnum;
SEND_MESSAGE_REPLY *Msg;
```

Description

The Send Message Reply call allows you to write a memory or non–memory message from the Host to the selected PCIM for transmission onto the bus (using the Transmit Datagram With Reply command). SendMsgReply will return control to the calling program without waiting for the reply. You must call ChkMsgStat or GetMsg to check for completion or to read the reply message.

IMnum defines the PCIM, as configured during initialization, from which to transmit the message. The Msg parameter is a pointer to the buffer where the transmit message is stored.

The format of SEND MESSAGE REPLY is:

Destination (0–31/255 brdcst)	– Destination address of Device
Function code (0–111)	– Function Code
T_SubFunction code (0–255)	– Transmitted Reply Sub Function Code
R_SubFunction code (0–255)	– Expected Reply Sub Function Code
Priority	– 0 – Normal, 1 – High
T_Length	– Data field length/length of message
Data (0–134)	– Message Data – depends on length parm

You can check the status of the message using ChkMsgStat to determine if the message completed processing properly.

Parameters are summarized as follows:

Parameter	Values	Function
IMnum	1–64	Relative number of PCIM
Msg	see above	Pointer to the buffer where the transmitted message will be stored – see above

The advantage of the SendMsgReply call over the SendMsg call is that it reduces user programming since a 10 second timeout to a non–responding device is automatically provided by the PCIM for a SendMsgReply call.

The Host program sequence for a SendMsgReply is as follows:

1. Host sends a SendMsgReply to the PCIM.
2. Host issues GetMsg calls until the Status indicates completion. GetMsg will also return the reply message into Host memory.

Return Value (Status)

SendMsgReply will return SUCCESS if a message has been transferred from the Host memory to the PCIM. Otherwise, one of the following FAIL indications will be returned:

BADIMNUM – IMnum is out of range (a count of 64 or greater).

NOINIT – Indicated PCIM has not been initialized (InitIM).

IMFAIL – The indicated PCIM has failed (PCIM OK = 0).

IMBUSY – The PCIM is otherwise engaged and cannot accept the command.

Note

You are responsible for defining the device, the Function code, the Sub–F unction code and the length of the transmit Datagram.

It is also your responsibility to interpret the Function code, the Sub–F unction code and the meaning of the Reply message. See GEK–90486–1 for predefined message codes.

Note

You cannot issue a SendMsg call or read a received unsolicited message while a SendMsgReply call is in progress. If this presents a timing problem, use the SendMsg call.

See Also: SendMsg, GetMsg and ChkMsgStat

Coding Example

This example sends a Read Diagnostics Message to device #8 on PCIM #1 and expects a reply message of Read Diagnostic Reply. This message requests 10 bytes of diagnostic data beginning at offset 0.

```
#include <pcim.h>

int status;
SEND_MESSAGE_REPLY Msg;

Msg.Destination = 8;          /*Device #8*/
Msg.Function = 0x20;          /*Genius Function Code*/
Msg.T SubFunction = 8;        /*Read Diagnostic Subfunction Code*/
Msg.R SubFunction = 9;        /*Read Diagnostic Reply Subfunction Code*/
Msg.Priority = NORMALP;       /*Transmit at Normal priority*/
Msg.T Length = 2;             /*Length of data in Data Buffer*/
Msg.Data[0] = 0;              /*Offset of 0*/
Msg.Data[1] = 0xA;            /*Length of 10 decimal*/

status = SendMsgReply (1, &Msg);
```

To see how message function calls work together, see Appendix A, Example 2.

ChkMsgStat – Read Message Progress Status

Summary

```
#include <pcim.h>

int         ChkMsgStat (IMnum, Replystatus)

int IMnum;
char *Replystatus;
```

Description

The Check Message Status call allows you to determine the status of a previous SendMsg call – that is, to determine when a transmitted message has actually been received, and its completion status.

IMnum is the PCIM number configured during initialization. The "Replystatus" parameter is a pointer to a buffer where the Status will be stored.

The "Replystatus" parameter will contain the following Macro values:

IMFREE	There is currently no activity.
IMBUSY	Message is still in progress.
SUCCESS	Message has successfully completed.
BADPARM	Message contained a syntax error.
TXERR	Message was not transmitted successfully.

Parameters are summarized as follows:

Parameter	Values	Function
IMnum	1–64	Relative number of PCIM
Replystatus	0/1	Pointer to a buffer where the Status will be stored –see above

Return Value (Status)

ChkMsgStat will normally return the Status requested and a SUCCESS indication. Other wise, one of the following FAIL indications will be returned:

BADIMNUM – IMnum is out of range (a count of 64 or greater).

NOINIT – Indicated PCIM has not been initialized (InitIM).

IMFAIL – The indicated PCIM has failed (PCIM OK = 0).

PCIMERR – There may be a problem with the PCIM firmware.

See Also: SendMsgReply, SendMsg and GetMsg

Coding Example

Check the message status area of PCIM #1.

```
#include <pcim.h>

int status;
char Status;

    status = ChkMsgStat (1, &Status);

    switch [STATUS]
    {
    case SUCCESS:
            ___;
        break;
    case IMFREE:
            ___;
        break;
    case IMBUSY:
            ___;
        break
    case BADPARM:
            ___;
        break
    case TXERR:
            ___;
        break
    case PCIMERR:
            ___;
        break
    default:
            ___;
        break
}
```

GetMsg – Read Received Message

Code Summary

```
#include <pcim.h>

int
GetMsg (IMnum, Msg)

int IMnum;
READ_MESSAGE *Msg;
```

Description

The Get Message call allows you to read a received memory or non–memory message (or a reply to a previous SendMsgReply call) from the selected PCIM into the Host memory "Msg" parameter.

IMnum is the PCIM number configured during initialization. The "Msg" parameter is a pointer to the buffer where the received message will be stored.

The format of READ MESSAGE is:

```
Source (0–31      – Source address of Device
Function code (0–111)      – Function Code
SubFunction code (0–255)   – Sub Function Code
DB Indicator (0–134)       – Directed (1)/Broadcast (0)
Length                     – Data field length/length of message
Data (0–134)               – Message Data –depends on length parm
```

Parameters are summarized as follows:

Parameter	Values	Function
IMnum	1–64	of PCIM
Msg	see above	Pointer to the buffer where the received message will be stored – see above

GetMsg performs the following sequence:

1. If there is a previous call to SendMsgReply, GetMsg checks to see if the transmission has successfully completed, and transfers the response back to you. If the response completed with an error, or if in progress, GetMsg will return a FAIL indication.
2. If there is no previous call to SendMsgReply, GetMsg checks to see if there is a memory message, and transfers that message back to you.
3. If no memory messages exist, then GetMsg checks to see if there is a non–memory message, and transfers that message back to you.
4. If no messages are present, GetMsg returns with a FAIL status.

Note

Unsolicited memory or non–memory Datagrams received by the PCIM may not be read by the Host while a SendMsg/Reply is in progress. This significantly affects Host response time to service received Datagrams. If this is a problem, use the SendMsg call instead of SendMsgReply.

Return Value (Status)

GetMsg will return SUCCESS if a memory or non–memory message is returned to you. Other wise, one of the following FAIL indications will be returned:

BADIMNUM – IMnum is out of range (a count of 64 or greater).

NOINIT – Indicated PCIM has not been initialized (InitIM).

IMFAIL – The indicated PCIM has failed (PCIM OK = 0).

IMBUSY – The PCIM is otherwise engaged and cannot accept the command.

NOMSG – No message is available to be received at this time.

TXERR – A message transmission has failed or response to SendMsgReply has not arrived after 10 seconds.

PCIMERR – There may be a problem with the PCIM firmware.

BADPARM – Syntax error from previous SendMsgReply

See Also: SendMsgReply, SendMsg and ChkMsgStat

Coding Example

Check to see if any messages exist on PCIM #1 and if so, store them into the location 'Msg'.

```
#include <pcim.h>

int status;
READ_MESSAGE Msg;

  status = GetMsg (1, &Msg);
```

GetINTR – Read Interrupt Status Table

Code Summary

```
#include <pcim.h>

int
GetINTR (IMnum, Intr)

unsigned int IMnum;
unsigned char *Intr;
```

Description

The Get Interrupt call allows you to read the selected PCIMs Interrupt Status Table. You can read this table to:

- See why an interrupt in the Host system has occurred
- Report the event in a non–interrupt environment, as is the default state of the Software Driver concept (the PCIM will still report the event even though the interrupt is disabled).

Thus, the Interrupt Status Table can be polled (by reading and interpreting it) to determine what is causing an interrupt from the PCIM.

When GetINTR is called, it transfers the data from the PCIMs Interrupt Status Table to the Host memory "Intr" parameter. The format of the Interrupt Status Table and its associated macros (shown below) is defined in the summary of data structures in this chapter and in <pcim.h>.

IMnum defines the PCIM, as configured during initialization, from which the Interrupt Status Table is to be read. The Intr parameter is a pointer to the buffer where the Interrupt Status Table information is stored.

The format of the Intr table is:

```
unsigned char  Intr;
```

The following Macros are used as shown in the Interrupt Status Table.

Macro	Position	Explanation
#define I ENABLE	0	– Enable the interrupt level.
#define I DISABLE	1	– Disable the interrupt level.
#define I SUMMARY	0	– Summary if interrupt occurred.
#define I REQUEST Q	1	– Received memory datagram.
#define I PCIM STAT	2	– PCIM Status Change – usually fatal.
#define I DEV STAT	3	– Device Status Change.
#define I OUT SENT	4	– Outputs sent – end of bus access.
#define I CCOMPLETE	5	– Command Block completed.
#define I RECEIVE D	6	– Received Datagram.

After data transfer to the Host is complete, GetINTR clears all of the PCIMs Interrupt Status Table bytes each time it is called. This way, you can see the lastest event that has occurred each call.

Parameters are summarized as follows:

Parameter	Value	Function
IMnum	1–64	Relative number of PCIM
Intr	see above	Pointer to the buffer where the table data will be stored

Return Value (Status)

GetINTR will return SUCCESS if the device specified by IMnum is present on the serial bus. If the target device is not present, or is out of range, GetINTR will return FAIL. The following FAIL indications will be returned:

BADIMNUM – IMnum is out of range (a count of 64 or greater).

NOINIT – Indicated PCIM has not been initialized (InitIM).

IMFAIL – The indicated PCIM has failed (PCIM OK = 0).

Coding Example

This example shows how, if an interrupt occurs on PCIM #1, to transfer the contents of that PCIMs Status Table. Interpretation of bits will depend on which interrupt is Enabled, and which application is to be run.

```
#include <pcim.h>

int status;
unsigned char Intr[8];

  if ((status = GetINTR (1, Intr)) !=SUCCESS)
       report–err (1, status);
 else
 {  /*do what is necessary for interrupt processing*/
  }
```

PutINTR – Write to the Interrupt Disable Table

Code Summary

```
#include <pcim.h>

int
PutINTR (IMnum, DisableIntr)

unsigned int IMnum;
unsigned char *DisableIntr;
```

Description

The Put Interrupt call allows you to write to the selected PCIM's Interrupt Disable Table. The PutINTR call first initializes a table to Enable and Disable individual interrupts as you require. The PutINTR call then writes this table to the Interrupt Disable Table on the PCIM. You can Enable or Disable interrupts in any mix; that is, on a single call, some interrupts may be Enabled and some Disabled, all may be Enabled, or all of the interrupts may be Disabled.

When PutINTR is called, it transfers the data from the Host memory "DisableIntr" parameter to the PCIMs Interrupt Disable Table. The format of the Interrupt Disable Table and its associated macros (shown below) is defined in the summary of data structures in this chapter and in <pcim.h>

IMnum defines the PCIM, as configured during initialization, to which DisableIntr will be read. The DisableIntr parameter is a pointer to the buffer where the Interrupt Disable Table information is stored.

The format of the DisableIntr table is:

```
unsigned char  DisableIntr;
```

The following Macros are used as shown in the Interrupt Disable Table.

Macro	Position	Explanation
#define I ENABLE	0	– Enable the interrupt level.
#define I DISABLE	1	– Disable the interrupt level.
#define I SUMMARY	0	– Summary if interrupt occurred.
#define I REQUEST Q	1	– Received memory datagram.
#define I PCIM STAT	2	– PCIM Status Change – usually fatal.
#define I DEV STAT	3	– Device Status Change.
#define I OUT SENT	4	– Outputs sent – end of bus access.
#define I CCOMPLETE	5	– Command Block completed.
#define I RECEIVE D	6	– Received Datagram.

Parameters are summarized as follows:

Parameter	Values	Function
IMnum	1–64	Relative number of PCIM
DisableIntr	see above	Pointer to the buffer from which enable/disable data is sent

Return Value (Status)

PutINTR will return SUCCESS if the device specified by IMnum is present on the serial bus. If the target device is not present, or is out of range, PutIntr will return FAIL. The following FAIL indications will be returned:

BADIMNUM	– IMnum is out of range (a count of 64 or greater).
NOINIT	– Indicated PCIM has not been initialized (InitIM).
IMFAIL	– The indicated PCIM has failed (PCIM OK = 0).

Coding Example

This example enables the Receive Datagram interrupt.

```
#include <pcim.h>

int status;
        x;
unsigned char DisableIntr[8];

/*Initialize the Disable Table*/
     for (x = 0; x < 8; x++)
                DisableIntr [x] = I_DISABLE     /* Disable all Interrupts*/

/*Enable Receive Datagram Interupt*/
                DisableIntr [I_RECEIVE_D] = I_ENABLE;

/*Now call use the call*/
     if ((status = PitiNTR (1, DisableIntr )) ! = SUCCESS)
                report_err (1, status)
```

Chapter 5 BASIC Programming for the PCIM

This chapter explains programming for a PCIM in BASIC. Programming requires: BASIC / MSOS

Basic Software Driver Installation

The Basic Software driver function call subroutines are made resident in your system when you execute the driver code file once under MS / OS as follows:

- Type 'PCIMX' in response to the DOS prompt 'A'> (if disk is in drive A).
  - The Driver code file is loaded into memory.
  - A short initialization sequence inside the Driver is executed.
- The Driver code displays the message 'PCIM Drivers Version x.x are Resident' and exits to DOS.
  - The Driver is resident in memory and available for use.
  - BASICA or GWBASIC can be loaded and calls to the Drivers performed.

If you need to recover the memory space occupied by the Driver, you must perform a system reset. In most cases, this will not be necessary since Driver code occupies only a small amount of memory (13K). If you plan to access the Driver frequently, the Driver code file can be moved to your system disk and executed from inside your AUTOEXEC.BAT file at startup. This will automatically make the Driver resident.

Software Driver Function Calls

The PCIM Software Driver consists of easy to use macro–oriented function calls you code appropriately in your C language or Basic language applications routines. Function calls are summarized below.

Functions that deal with PCIM configuration:

InitIM – assigns PCIM numbers and Global data parameters to all PCIMs. Performs any required hardware activation and initialization of the PCIMs (such as Reset).

ChgIMSetup – writes to the Setup Table of the selected PCIM from the Host memory to change PCIM parameters.

GetIMState – reads PCIM configuration and status from the selected PCIM Status Table and Setup Table into Host memory.

Functions that deal with bus configuration:

GetBusConfig – reads all Device Configuration Tables from the selected PCIM into Host memory.

GetDevConfig – reads one device's configuration from the selected PCIM into Host memory.

DisableOut – writes to the Device Configuration Table of the selected PCIM to enable / disable outputs to selected devices or to all devices.

Functions that deal with control data movement:

GetBusIn – reads the entire Input Table (control data inputs) from a selected PCIM into Host memory.

PutBusOut – writes the entire Output Table (control data outputs) to a selected PCIM from Host memory.

GetDevIn – read control data inputs from a selected bus device into Host memory.

PutDevOut – write control data outputs to a selected bus device from Host memory.

GetIMIn – reads all PCIM control data from Directed Control Input Table of selected PCIM into Host memory.

PutIMOut – writes all PCIM control data to Global Data Table of selected PCIM from Host memory.

GetCir – reads an input circuit value (variable) into the Host memory from the Input Table of a selected PCIM.

GetWord – reads an input word value (variable) into the Host memory from the Input Table of a selected PCIM.

PutCir – writes an output circuit value (variable) from the Host memory to the Output Table of a selected PCIM.

PutWord – writes an output word value (variable) from the Host memory to the Output Table of a selected PCIM.

Functions that deal with communications:

GetMsg – reads a received message from a selected PCIM into Host memory.

SendMsg – writes a message from Host memory to the PCIM for transmission onto the bus.

SendMsgReply – writes a message from Host memory to the PCIM for transmission onto the bus and expects a specified reply message from the destination.

ChkMsgStat – allows the Host to detect when a transmitted message has actually been completed, or if transmission is incomplete or has failed.

Functions that deal with interrupt processing:

GetINTR – reads the entire Interrupt Status Table from a selected bus device into Host memory.

PutINTR – writes the entire Interrupt Status Table to a selected PCIM from Host memory.

Using Software Driver Function Calls

When coding the PCIM Software Drivers in your application programs, you should have at hand the following:

- Starting Address (Segment Address) of the Shared RAM Interface.
- I/OPort Base Address.
- Status Table Address (PCIMs) or Reference Address (Series Six).
- Serial Bus Address of each bus device.
- Global, Input, Output Data lengths for all devices.

It is also helpful to have the *Genius I/O System and Communications User's Manual* (GEK–90486–1) handy for reference.

Basic Software Driver Function Call Parameters

Software Driver function calls require that you specify a number of parameters for each call. The data structures for each parameter, which are linked and loaded from the Software Driver .exe file, are summarized below.

IBM PC BASICA interpreter does not allow the passing of constants in the parameter list of a CALL statement. Only variables may be passed. You must load all variables which supply information to the Driver before performing a function call. In the parameter lists which follow, all parameters are either single integers or are arrays of integers.

Note

BASICA interpreter requires that all arrays be called with subscript. If this is violated, incorrect data and / or system crash is the usual result.

Basic Data Array Structures

IMPARMS

The user–supplied IMPARMS() array sets parameters for the initialization of each IM.

The format of "IMPARMS()" is:

Variable, depending on how many IMs are to be initialized, (can be up to 383)

0	– Segment address of 1st PCIM daughterboard
1	– I/O Port address (dip switch setting)
2	– Starting Ref addr for global data
3	– Global data length (0–127)
4	– Input data length (0–127)
5	– Active (1=ON, 0=OFF)
6	– Segment address of 2nd PCIM SIR
7	– I/O Point address (DIP switch setting)
8	•
•	•
•	•

IMFLAGS

The IMFLAGS() array is a system return used by INITIM to tell you which PCIMs initialized properly (on improperly, as the case may be). The length of IMFLAGS should be equal to the number of IMs or IMCOUNT.

The format of "IMFLAGS()" is:

Variable depending on the number of IMs (can be up to 64)

0	– Flag for the 1st IM
1	– Flag for the 2nd IM
2	– Flag for the 3rd IM
3	– Flag for the 4th IM
•	•
•	•
•	•
•	•

IMSTATE

The IMSTATE() array is a system return used for accessing configuration and status information about a specific PCIM by reading its Setup Table and Status Table.

The format of "IMSTATE()" is:

" 10 #

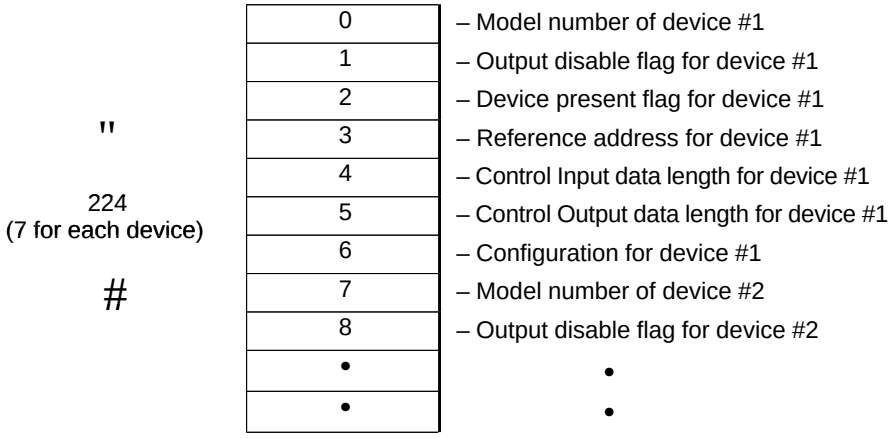
0	– Daughterboard configuration
1	– Global Data Reference
2	– Global Data Length
3	– Normally set to 0
4	– PCIM Firmware Revision number
5	– PCIM Hardware OK flag
6	– PCIM Fault Description
7	– PCIM Present/Excess Bus Errors flag
8	– HHM Present/Excess Bus Errors flag
9	– Serial Bus Error Count
10 *	– Bus Scan Time in mS

For more information about the content of the IMSTATE( ) array, particularly the daughterboard configuration parameters, see page 4-9.

BUSCONFIG

The BUSCONFIG() array is a system return used to access the configuration of all 32 devices from the PCIM selected by the IMNUM parameter.

The format of "BUSCONFIG()" is:

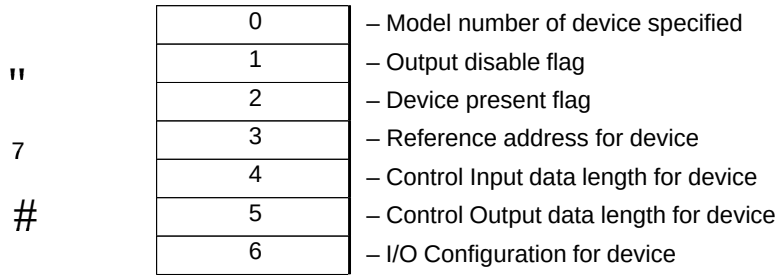


For more information about the content of the BUSCONFIG( ) array, see page 4-10.

DEVCONFIG

The user–supplied DEVCONFIG() array is a system return very similar to BUSCONFIG array, except that it can only read the configuration of 1 device at a time.

The format of "DEVCONFIG()" is:



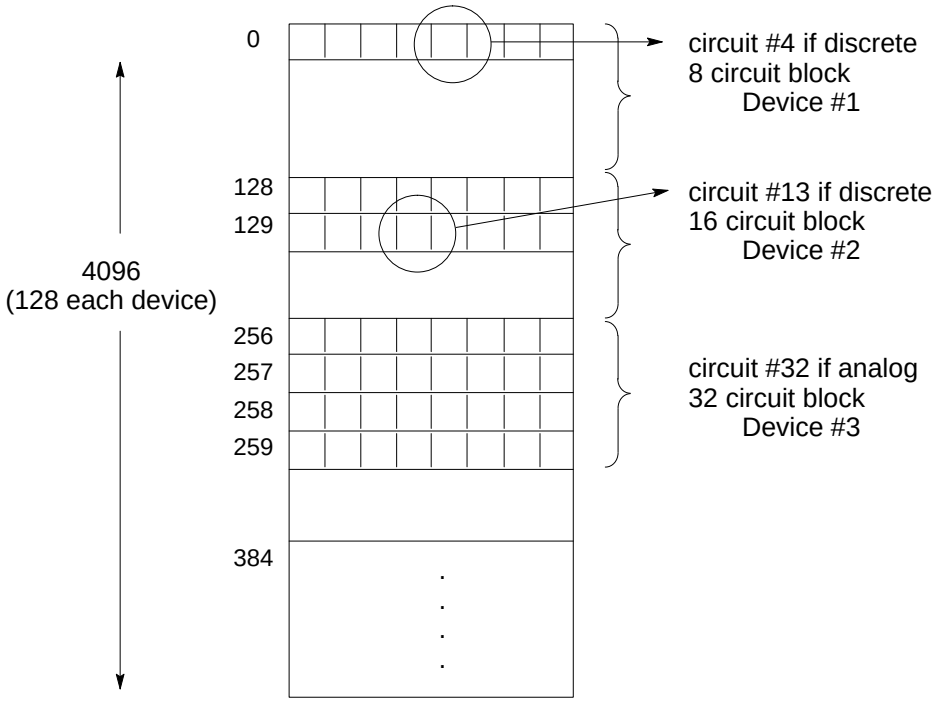
For more information about the content of the BUSCONFIG( ) array, see page 4-10.

IODATA

The IODATA() array is used to read and / or write I / O data to and from the PCIM input / output tables to all the devices on the bus (User supplied for PUTBUSOUT call / Systemreturned for GETBUSIN call).

The format of "IODATA()" is:

46361

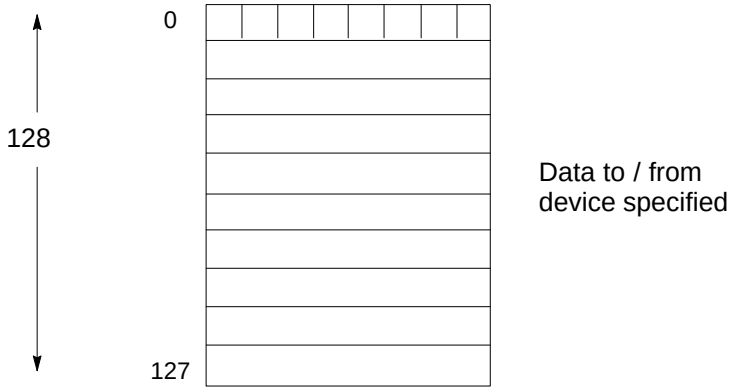


DEVDATA

The DEVDATA() array is very similar to IODATA() except that it is used to read and / or write I / O data to and from the PCIM input / output tables to a device on the bus (User supplied for PUTBUSOUT call / System returned for GETBUSIN call).

The format of "DEVDATA()" is:

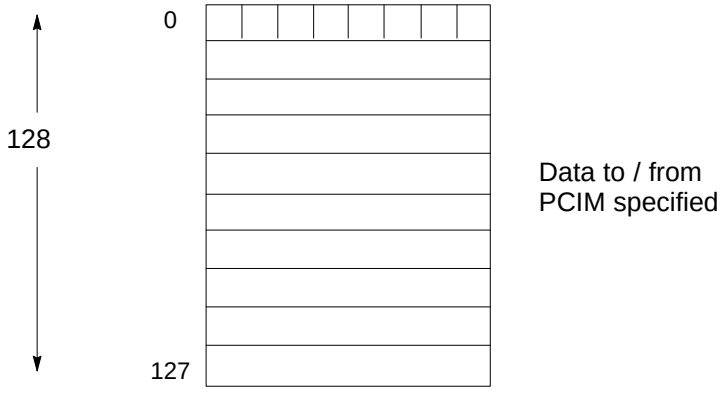
46362



IMDATA

The IMDATA() array is a buffer where Global Data to be read will be located. The size of this parameter is determined by the "Inputlength" parameter located in the PCIMs configuration data.

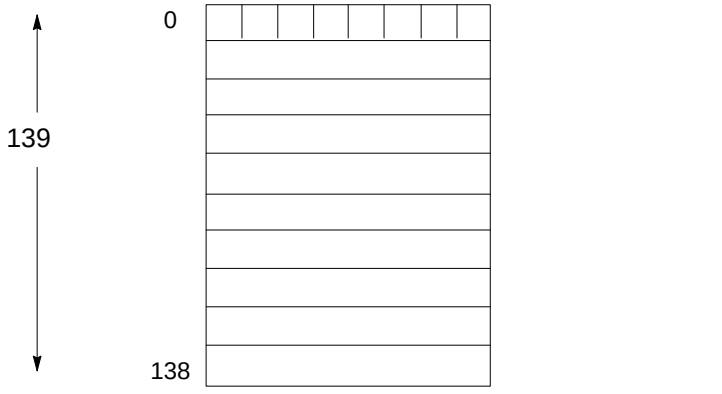
The format of "IMDATA()" is:



MSG

The MSG() array is a buffer where the message to be sent (SENDMSG) or message to be received (GETMSG) will be stored.

The format of "MSG()" is:

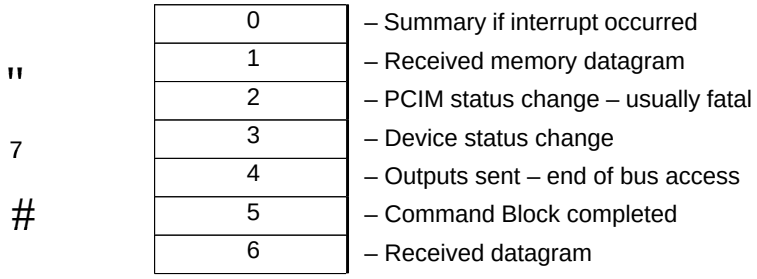


For explanations of the content of the MSG( ) array, see pages 4-11 and 4-12.

INTR/DISABLEINTR

The INTR and DISABLEINTR arrays are used to read the selected PCIMs Interrupt Status Table and write to the selected PCIMs Interrupt Disable Table, respectively.

The format of “INTR” and “DISABLEINTR” is:



Error Status Indication

Any function call may return an error condition. You are informed of error conditions by a non–zero error code returned in the STATUS variable included as the first parameter in every call. Normal completion of a function call is indicated by a zero STATUS returned. The table of error codes that follows will help you interpret these codes. A simple check for non–zero STATUS must be performed after each driver call to detect error conditions.

The following error codes are returned for all calls:

	Error Code	Explanation
SUCCESS	0	Successful completion of function.
INITFAIL	1	Initialization Failure.
IMFAIL	2	PCIM Failure.
BADSEG	3	Invalid Segment address.
BADPORT	4	Invalid I/O Port Address.
BADCFG	5	Invalid Configuration parameter.
NOCFG	6	No Configuration changes found.
NOINIT	7	PBIM selected is not initialized.
NODATA	8	No data found.
UNDERFLOW	9	Insufficient device data length.
OVERFLOW	10	Exceeds device data length.
OFFLINE	11	Device is offline.
IMBUSY	12	PCIM busy.
BADPARM	13	Invalid message parameter.
TXERR	14	Message transmit failure.
NOMSG	15	No Message available.
IMFREE	16	No message activity.
BADSBA	17	Invalid Serial Bus Address.
BADIMNUM	18	Invalid PCIM Number.
PCIMERR	19	PCIM firmware problem.
DUPSEG	20	Duplicate segment values given.
DUPPORT	21	Duplicate IO Port values given.

Access from BASIC

Every BASIC program which accesses the PCIM Software Driver must perform a short startup sequence to let BASIC know where each of the function call subroutines is located. This startup sequence is listed below. It is also included on the Driver diskette in the file PCIM.BAS so you can copy it at the beginning of new programs rather than re–code it every time you need it.

```
10  OPTION BASE 0
20  DEFINT A–Z
30  DIM IMPARMS(383),IMFLAGS (63),IMSTATE (9),IMDATA(127),BUSCONFIG(223)
40  DIM DEVDATA(127),IODATA(4095),MSG(139),DEVCONFIG(7)
50  DIM INTR(6),DISABLEINTR(7)
60  DEF SEG=0
70  SUBSEG=(PEEK(&H4F1)*256) + PEEK(&H4F0)
80  DROFFSET=(PEEK(&H4F3)*256) + PEEK(&H4F2)
90  IF SUBSEG0 THEN 180
100 '
110 '    Non–resident return
120 '
130 PRINT "PCIM Drivers not resident."
140 SYSTEM
150 '
160 '    Continue normally
170 '
180 DEF SEG=SUBSEG
190 INITIM=0+DROFFSET
200 GETDEVIN=4+DROFFSET
210 PUTDEVOUT=8+DROFFSET
220 GETBUSIN=12+DROFFSET
230 PUTBUSOUT=16+DROFFSET
240 GETIMIN=20+DROFFSET
250 PUTIMOUT=24+DROFFSET
260 GETCIR=28+DROFFSET
270 GETWORD=32+DROFFSET
280 PUTCIR=36+DROFFSET
290 PUTWORD=40+DROFFSET
300 CHGIMSETUP=44+DROFFSET
310 GETIMSTATE=48+DROFFSET
320 GETBUSCONFIG=52+DROFFSET
330 GETDEVCONFIG=56+DROFFSET
340 DISABLEOUT=60+DROFFSET
350 GETMSG=64+DROFFSET
360 SENDMSG=68+DROFFSET
370 SENDMSGREPLY=72+DROFFSET
380 CHKMSGSTAT=76+DROFFSET
390 GETINTR=80+DROFFSET
400 PUTINTR=84+DROFFSET
410 '
420 '    Get inputs for initialization function call INITIM.
430 '    INITIM must be called first to initialize PCIMs and
440 '    check that they were initialized.
450 '
460 CALL INITIM (STATUS,IMCOUNT,IMPARMS(0),IMFLA)
```

In the above sequence:

- Line 10 forces array indexing to start at zero since this is more convenient when using the Driver,
- Line 20 defaults all variables to integer type (use the type overrides for single and double precision reals),
- Lines 60 through 180 find the segment address in memory where the Driver has previously been installed and ensures that it is present,
- And Lines 190 through 400 define the offsets in the segment for each of the function call subroutines.
- Lines 410 through 460 are simply a reminder to call for initialization first (see the INITIM call).

Coding Basic Function Calls

There are two ways to call a function in Basic, as shown below:

1. Segment relocation – first relocate the segment, perform the the call, then restore the segment. For example, to call INITIM, code:

```
1000 DEF SEG=SUBSEG
1010 CALL INITIM(parameters)
1020 DEF SEG
```

2. No relocation – if you know in advance that other BASICA statements which depend on segment relocation (PEEK, POKE, BLOAD, BSAVE, DEF USR, or CALLs to other user routines) will not be used, then the code in line 1000 above can be executed once at startup to set the segment to the Driver. Function calls can then be coded on a single line without segment relocation. Using the same example:

```
1010 CALL INITIM(parameters)
```

INITIM CALL Statement

Syntax

CALL INITIM (STATUS, IMCOUNT, IMPARMS(0), IMFLAGS(0))

Action

Setup and Activate PCIM

Description

The Initialize IM call specifies the total number of PCIMs in the Host system through the parameter "IMCOUNT", and the characteristics of each PCIM through the parameter "IMPARMS".

INITIM resets the IMcount of PCIMs in the Host system and initializes each PCIM as defined by IMPARMS. You must create a separate IMPARMS entry for each PCIM in IMCOUNT. Each PCIM requires the entries in IMPARMS array.

The format of "IMPARMS" is:

IMPARMS(0) IM 1 – Segment Address of PCIM shared RAM (two bytes LSB – MSB)
IMPARMS(1) IM 1 – I/O Port Address (two bytes LSB – MSB)
IMPARMS(2) IM 1 – PCIM Global Reference (two bytes LSB – MSB)
IMPARMS(3) IM 1 – Global data length (one byte)
IMPARMS(4) IM 1 – Input directed data length (normally 0)
IMPARMS(5) IM 1 – Active (one byte) 1 = ON, 0 = OFF)

IMPARMS(6) IM 2 – Segment Address of PCIM shared RAM (two bytes LSB – MSB)
IMPARMS(7) IM 2 – I/O Port Address (two bytes LSB – MSB)
IMPARMS(8) IM 2 – PCIM Global Reference (two bytes LSB – MSB)
IMPARMS(9) IM 2 – Global data length (one byte)
IMPARMS(10) IM 2 – Input directed data length (normally 0)
IMPARMS(11) IM 2 – Active (one byte) 1 = ON, 0 = OFF)

etc...

Note

The memory pointer and I/O port assignments must correspond to the dip switch settings on the PCIM.

The last parameter, "IMFLAGS", is an array the size of IMCOUNT, used by INITIM to tell you which PCIMs initialized properly (or improperly, as the case may be). The number of flags should equal IMCOUNT.

Parameters are summarized as follows:

Parameter	Values	Function
IMCOUNT	1–64	Total number of PCIMs
IMPARMS	varies (6 entries/IM)	shows the characteristics of each module – see above
IMFLAGS	varies	tells you which PCIMs initialized properly – see above
STATUS	0 / 1	success / fail

The INITIM call performs the following sequence of actions:

1. Brings each defined PCIM out of reset, or (if the PCIM is already running) into reset then out of reset.
2. downloads Global data parameters to each PCIM after its PCIM OK LED turns ON (may take up to two seconds).
3. After all PCIMs have been downloaded or a two second timeout has occurred, returns with an IMFLAGS array (one for each defined PCIM). Status returned will be Fail for any syntax or execution errors detected. An example of an execution error is failure of the PCIM OK flag to be ON within two seconds after Reset.

Status Value

INITIM will return SUCCESS if all resets and data parameters were accepted by each PCIM. The following failure codes are returned:

BADIMNUM	IMcount is out of range (a count of 64 or greater). No more InitIM processing is performed.
INITFAIL	An initialization problem occurred in one or more PCIM. The individual status for each PCIM on the bus is located in the IMflags parameter.

One of the following status codes will be stored in the appropriate location in the IMFLAGS parameter if the return code is INITFAIL. Each status value in the IMFLAGS array is unique to the associated PCIM and does not reflect the status of any other PCIM.

INITFAIL	This PCIM failed to power up. (Incorrect segment address or port address).
SUCCESS	This PCIM has been powered up and configured as specified.
IMFAIL	This PCIM never powered up.
BADCFG	This PCIM rejected the configuration because a parameter was out of range.
BADSEG	The segment value in IMparms is set to the illegal value 0 (zero)
BADPORT	The I/O port address is set to some illegal value less than 256.
DUPSEG	The segment address is a duplicate of another PCIM.
DUPPORT	The port address is a duplicate of another PCIM.

Note

If any of the PCIMs fail to initialize as you specified in IMPARMS, INITIM turns OFF the failed PCIM.

Coding Example

In this example are two PCIMs.

```
410 IMCOUNT      = 2 ; 2 PCIMs
420 IMPARMS  (0) = &HD000        ;IM1 – PCIM #1 Segment address
430 IMPARMS  (1) = &H3E4         ;IM1 – Port address
440 IMPARMS  (2) = &H7000        ;IM1 – Reference address
450 IMPARMS  (3) = 0             ;IM1 – No global data
460 IMPARMS  (4) = 0             ;IM1 – No Directed input data
470 IMPARMS  (5) = 1             ;IM1 – Turn PCI on by default
480 IMPARMS  (6) = &HCC00        ;IM2 – PCIM #2 Segment address
490 IMPARMS  (7) = &H3E0         ;IM2 – Port address
500 IMPARMS  (8) = &H8001        ;IM2 – Global Data Reference address (Series 6
                                        register 1)
510 IMPARMS  (9) = 0             ;IM2 – No global data
520 IMPARMS (10) = 0             ;IM2 – No Directed input data
530 IMPARMS (11) = 1             ;IM1 – Turn PCI on by default
540 Call INITIM(STATUS,IMCOUNT,IMPARMS(0),IMFLAGS(0))
```

CHGIMSETUP CALL Statement

Syntax

CALL CHGIMSETUP (STATUS, IMNUM, IMPARMS(0))

Action

Change PCIM Configuration

Description

Following initialization, any changes you make to the configuration of a specific PCIM must use the Change IM Setup call. This call allows you to make configuration changes to a specific PCIM Setup Table by writing the IMPARMS parameter from Host memory to it.

The "IMNUM" parameter is an offset of the IMPARMS parameter which, after initialization, indicates the specific PCIM in the host system for which configuration changes are intended.

Note

Configuration changes to any PCIM while online causes that IM to stop transmitting on the serial bus for 1.5 seconds.

The format of "IMPARMS" is the same as shown in the INITIM call. However only four of the parameters should be allowed to be changed. These are as follows:

IMPARMS(I+2) – Reference Address
IMPARMS(I+3) – Global data length
IMPARMS(I+4) – Input data length
IMPARMS(I+5) – Active (1 = ON, 0 = OFF)

I = (IMNUM–1)*6

Parameters are summarized as follows:

Parameter	Values	Function
IMNUM	1–64	Relative number of PCIM
IMPARMS	varies	shows the characteristics of each module – see above
STATUS	0 / 1	success / fail

Status Value

CHGIMSETUP will return SUCCESS if all changes were accepted by the target IM. If the IM fails to change to the new parameters, the following FAIL indications will be returned:

BADIMNUM	IMnum is out of range (a count of 64 or greater).
NOINIT	Indicated PCIM has not been initialized (InitIM).
IMFAIL	The indicated PCIM has failed (PCIM OK = 0), or never completed processing the config change command.
IMBUSY	The PCIM is otherwise engaged and cannot accept the config change command.
BADCFG	This PCIM rejected the configuration because a parameter was out of range.
NOCFG	The PCIM, after examining the received the config change command, found no changes to make.
INITFAIL	Change of Global Data output or Directed Data input length required a reset of PCIM daughterboard and the daughterboard failed to reinitialize.

Coding Example

Change the reference address for PCIM #1.

```
600 IMNUM = 1
610 IMPARMS (2) = &H6000 ;new reference address
620 Call CHGIMSETUP(STATUS, IMNUM, IMPARMS(0))
```

Turn off PCIM #2.

```
690 IMNUM = 2
700 IMPARMS (5) = 0
720 Call CHGIMSETUP(STATUS,IMNUM,IMPARMS(0))
730 'Check status for next action
740 If STATUS = 0  Then 760 else 800
```

GETIMSTATE CALL Statement

Syntax

CALL GETIMSTATE (STATUS, IMNUM, IMSTATE(0))

Action

Get Configuration and Status Information

Description

The Get IM State call allows you to access configuration and status information about a specific PCIM by reading its Setup Table and Status Table into the "IMSTATE" parameter in Host memory.

The format of IMSTATE is:

IMSTATE(0) DipSwitch	– See page 4-9.
IMSTATE(1) IMRef	– Reference Address
IMSTATE(2) OutputLength	– Output Control Data Length
IMSTATE(3) InputLength	– Input Control Data Length
IMSTATE(4) Revision	– PCIM Firmware Revision Number
IMSTATE(5) PCIM OK	– PCIM OK – every 200 ms, set to 1. If PCIM OK=0, board has failed.
IMSTATE(6) Fault	– Overall fault byte – any PCIM fault
IMSTATE(7) Active	– Hand Held Monitor Present
IMSTATE(8) SBerr	– Serial Bus error count
IMSTATE(9) ScanTime	– Bus Scan Time in ms

Since the PCIM periodically sets its PCIM OK flag, this call allows the implementation of a PCIM OK heartbeat procedure.

Parameters are summarized as follows:

Parameter	Values	Function
IMNUM	1–64	Relative number of PCIM
IMSTATE	varies	PCIM Configuration and Status – see above
STATUS	0/1	Success/Fail

Status value

GETIMSTATE will almost always return SUCCESS. If the target IM is currently offline, has not been initialized, or is out of range, the following FAIL indications will be returned:

BADIMNUM	– IMNUM is out of range (a count of 64 or greater).
NOINIT	– Indicated PCIM has not been initialized (INITIM).
IMFAIL	– The indicated PCIM has failed (PCIM OK = 0).

Coding Example

Examine the state of PCIM #1.

```
1000 IMNUM = 1
1010 CALL GETIMSTATE(STATUS,IMNUM,IMSTATE(0))
```

GETBUSCONFIGCALL Statement

Syntax

CALL GETBUSCONFIG (STATUS, IMNUM, BUSCONFIG(0))

Action

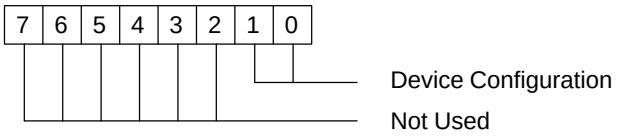
Get Serial Bus Configuration

Description

The Get Bus Configuration call allows you to read device configuration information about all devices on a serial bus (except the PCIM). GETBUSCONFIG reads all 32 Device Configuration Tables from the PCIM selected by IMNUM into the Host memory "BUSCONFIG" parameter. BUSCONFIG parm – 224 in length, 7 entries per device.

The format of BUSCONFIG is:

BUSCONFIG (0) Model – Model Number of device
BUSCONFIG (1) OutputDisable – Output disable flag
BUSCONFIG (2) Present – Device Present flag
BUSCONFIG (3) Reference – Status Table or Reference Address
BUSCONFIG (4) InputLength – Control Input Data Length
BUSCONFIG (5) OutputLength – Control Output Data Length
BUSCONFIG (6) Config – Device Configuration
1 = all inputs
2 = all outputs
3 = combination



46365

Parameters are summarized as follows:

Parameter	Values	Function
IMNUM	1–64	Relative number of PCIM
BUSCONFIG	224 entries (7 per device)	Device configuration information about all devices on the bus – see above.
STATUS		Success/Fail

Status Value

GETBUSCONFIG will almost always return SUCCESS. If the target IM is currently offline, has not been initialized, or is out of range, the following FAIL indications will be returned:

BADIMNUM	IMNUM is out of range (a count of 64 or greater).
NOINIT	Indicated PCIM has not been initialized (InitIM).
IMFAIL	The indicated PCIM has failed (PCIM OK = 0).
OFFLINE	None of the devices specified are currently active on the bus. However, the appropriate buffer is still returned and will contain configuration data for devices once logged in. Zeros will be returned if no device has logged in to a particular slot.

Coding Example

Examine the configuration of the devices on PCIM #1.

```
1100 IMNUM = 1
1110 Call GETBUSCONFIG (STATUS,IMNUM,BUSCONFIG(0))
```

GETDEVCONFIGCALL Statement

Syntax

CALL GETDEVCONFIG (STATUS, IMNUM, DEVICENUM, DEVCONFIG(0))

Action

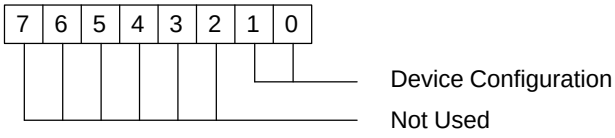
Get Device Configuration

Description

The Get Device Configuration call allows you to read device configuration information about a specific device on the serial bus. GETDEVCONFIG reads this information from the PCIM selected by IMNUM into the Host memory "DEVCONFIG" parameter.

Again, the format of DEVCONFIG is:

DEVCONFIG(0) Model – Model Number of device
DEVCONFIG(1) OutputDisable – Output disable flag
DEVCONFIG(2) Present – Device Present flag
DEVCONFIG(3) Reference – Status Table or Reference Address
DEVCONFIG(4) InputLength – Control Input Data Length
DEVCONFIG(5) OutputLength – Control Output Data Length
DEVCONFIG(6) Config – Device Configuration
1 = all inputs
2 = all outputs
3 = combination



46365

Parameters are summarized as follows:

Parameter	Values	Function
IMNUM	1–64	Relative number of PCIM
DEVICENUM	0–31	Specifies device on serial bus
DEVCONFIG	7 entries	Device configuration of DEVICENUM
STATUS		Success/Fail

Status Value

GETDEVCONFIG will almost always return SUCCESS. If the target IM is currently offline, has not been initialized, or is out of range, the following FAIL indications will be returned:

BADIMNUM	– IMNUM is out of range (a count of 64 or greater).
BADSBA	– Specified DEVICENUM is not in the range for Genius bus devices (0 –31 decimal).
NOINIT	– Indicated PCIM has not been initialized (INITIM).
IMFAIL	– The indicated PCIM has failed (PCIM OK = 0), or never completed processing the config change command.
OFFLINE	– The device requested is currently not on the bus, however, the appropriate buffer is still returned and will contain configuration data for devices once logged in.

Coding Example

Examine the configuration of device #30 on PCIM #1.

```
1200 IMNUM = 1
1210 DEVICENUM = 30
1220 Call GETDEVICECONFIG (STATUS,IMNUM,DEVICENUM,DEVCONFIG(0))
```

DISABLEOUTCALL Statement

Syntax

CALL DISABLEOUT (STATUS, IMNUM, DEVICENUM, FLAG)

Action

Disable / EnableDevice Outputs

Description

The Disable (/ Enable) Outputs call allows you to selectively disable (or enable) outputs to a specific device, or to all devices, on a serial bus.

If FLAG is n n (‘1’), outputs to the device will be disabled; if FLAG is zero (‘0’), outputs will be enabled to that device. If you code the DEVICENUM value equal to ‘ALL’(32), then the outputs to all devices will be set to the value of FLAG. If DEVICENUM is a serial bus address value between 0 – 31 decimal, then the flag value will only affect that device.

Parameters are summarized as follows:

Parameter	Values	Function
IMNUM	1–64	Relative number of PCIM
DEVICENUM	0–31 32	Specifies device on serial bus on which circuit resides Specifies all devices
FLAG	0 or 1	Enable/disable outputs
STATUS		Success/Fail

Status Value

DISABLEOUT will return SUCCESS if the device specified by IMNUM is present on the serial bus. Otherwise, DISABLEOUT will return FAIL. If DEVICENUM indicates ALL, then DISABLEOUT will almost always return SUCCESS. The following FAIL indications will be returned:

BADIMNUM – IMNUM is out of range (a count of 64 or greater).

BADSBA – Specified DEVICENUM is not in the range for Genius bus devices (0 – 31 decimal).

NOINIT – Indicated PCIM has not been initialized (INITIM).

IMFAIL – The indicated PCIM has failed (PCIM OK = 0).

Coding Example

Enable outputs to device #8 on PCIM #1.

```
1600 DEVICENUM = 8
1610 IMNUM = 1
1620 FLAG = 0
1630 Call DISABLEOUT(STATUS,IMNUM,DEVICENUM,FLAG)
```

Disable outputs to all devices on PCIM #1.

```
1700 DEVICENUM = 32
1710 IMNUM =  1
1720 FLAG =  1
1730 Call DISABLEOUT(STATUS,IMNUM,DEVICENUM,FLAG)
```

GETBUSIN CALL Statement

Syntax

CALL GETBUSIN (STATUS, IMNUM, IODATA(0))

Action

Read all Input Values

Description

A Get Bus Inputs call allows you to read input values from all active devices in the Input Table of the specified PCIM. Active inputs are those for which the Device Present flag is set to '1' (it is the application's responsibility to know which devices are present on the bus via the GETBUSCONFIG call). Active input values are placed into the Host memory "IODATA" parameter. IODATA must be an array buffer where the I/O information will be saved. The IODATA parm is 4096 in length, 128 entries / device, times 32 devices. Slots are in serial bus address order.

When GETBUSIN is called, it begins by "locking out" the PCIM from updating its Input Table (ensures data coherency across bus scans). GETBUSIN then transfers the entire Input Table to the IODATA parameter, even for devices that are not active. When the entire PCIM Input Table has been searched, the PCIM is "unlocked".

Parameters are summarized as follows:

Parameter	Values	Function
IMNUM	1–64	Relative number of PCIM
IODATA	4096 bytes	Data parameter will be copied to Host memory from specified PCIM
STATUS		Success/Fail

Status Value

GETBUSIN will return SUCCESS if any of the devices specified by the IMNUM are active and data was transferred. If no devices are present on the target IM, if the target IM is currently offline, has not been initialized, or is out of range, the following FAIL indications will be returned:

BADIMNUM	– IMNUM is out of range (a count of 64 or greater).
NOINIT	– Indicated PCIM has not been initialized (INITIM).
IMFAIL	– The indicated PCIM has failed (PCIM OK = 0).
OFFLINE	– No devices are currently on the bus, however, the appropriate buffer is still returned and will contain data for devices once logged in.

Coding Example

Read all inputs from all active devices on PCIM #1.

```
2000 IMNUM = 1
2010 Call GETBUSIN(STATUS,IMNUM,IODATA(0))
```

PUTBUSOUT CALL Statement

Syntax

CALL PUTBUSOUT (STATUS, IMNUM, IODATA(0))

Action

Write all Output Values

Description

The Put Bus Outputs call allows you to update outputs to all active devices in the Output Table of the specified PCIM. Active outputs are those with the Device Present flag set to '1' (it is the application's responsibility to know which devices are present on the bus via the GETBUSCONFIG call). Active output values are written from the Host memory IODATA parameter. IODATA must be an array buffer where the I/O information is saved. The IODATA parm is 4096 in length, 128 entries / device, times 32 devices. Slots are in serial bus address order.

When PUTBUSOUT is called, it begins by "locking–out" the PCIM from updating its Output Table (ensures data coherency across PCIM scans). PUTBUSOUT then transfers all data from IODATA to the Output Table. When the entire PCIM Output Table has been searched, the PCIM is "unlocked".

Parameters are summarized as follows:

Parameter	Values	Function
IMNUM	1–64	Relative number of PCIM
IODATA	4096 bytes	Data parameter will be copied from Host memory to specified PCIM.
STATUS		Success/Fail

Status Value

PUTBUSOUT will return SUCCESS if any of the devices specified by the IMNUM are active and data was transferred. If no devices are present on the target IM, if the target IM is currently offline, has not been initialized, or is out of range, the following FAIL indications will be returned:

BADIMNUM – IMNUM is out of range (a count of 64 or greater).

NOINIT – Indicated PCIM has not been initialized (INITIM).

IMFAIL – The indicated PCIM has failed (PCIM OK = 0).

OFFLINE – Data was transferred to the output tables, however, no devices were found on the bus.

Coding Example

Write all outputs to all active devices (4) on PCIM #1.

```
2100 IMNUM = 1
2110 IODATA (0) = 1
2120 IODATA (128) = 2
2130 IODATA (256) = 4
2140 IODATA (384) = 8
2150 Call PUTBUSOUT(STATUS,IMNUM,IODATA(0))
```

GETDEVIN CALL Statement

Syntax

CALL GETDEVIN (STATUS, IMNUM, DEVICENUM, LENGTH, DEVDATA(0))

Action

Read Device Data Only

Description

The GETDEVIN function allows you to read the control data inputs received from a single serial bus device into the Host memory "DEVDATA" parameter.

IMNUM is the PCIM number configured during initialization. The DEVICENUM parameter specifies the serial bus address of the device from which input data is to be read. The "LENGTH" parameter is the length of the input data the device sent. This way, the function can determine whether or not it should update its current data base. The "DEVDATA" parameter is a buffer data read will be located. The size of this buffer is determined by the "InputLength" parameter located in the device's configuration data.

Parameters are summarized as follows:

Parameter	Values	Function
IMNUM	1–64	Relative number of PCIM
DEVICENUM	0–31	Specifies device on serial bus from which input data will be read
LENGTH	0–128	Size of data buffer
DEVDATA	variable	Buffer where data in Host stored – see above
STATUS		Success/Fail

Status Value

GETDEVIN will return SUCCESS if the device specified by IMNUM is present on the serial bus, and after the data is transferred to the DEVDATA buffer. If the target device is not present, or is out of range, the following FAIL indications will be returned:

BADIMNUM	IMNUM is out of range (a count of 64 or greater).
BADSBA	Specified Devicenum is not in the range for Genius bus devices (0 –31 decimal), or is that of the PCIM.
NOINIT	Indicated PCIM has not been initialized (InitIM).
IMFAIL	The indicated PCIM has failed (PCIM OK = 0).
OFFLINE	The device requested is currently not on the bus, and data is NOT transferred.

Coding Example

Get the inputs from device #8 on PCIM #1.

```
2300 IMNUM = 1
2310 DEVICENUM = 8
2320 Call GETDEVCONFIG(STATUS,IMNUM,DEVICENUM,DEVDATA(0))
2330 LENGTH = DEVCONFIG(4))
2340 Call GETDEVIN(STATUS,IMNUM,DEVICENUM,LENGTH,DEVCONFIG(0))
```

PUTDEVOUT CALL Statement

Syntax

CALL PUTDEVOUT (STATUS, IMNUM, DEVICENUM, LENGTH, DEVDATA(0))

Action

Write Device Data Only

Description

The PUTDEVOUT call allows you to write all of the control data outputs to a single serial bus device from the Host memory DEVDATA parameter.

IMNUM is the PCIM number configured during initialization. The DEVICENUM parameter specifies the serial bus address of the device to which output data is to be written. The LENGTH parameter is length of data to be sent to the device. If the value differs from the PCIMs current data base, an Overflow or Underflow error will be returned. The DEVDATA parameter is a buffer where the data to be written is located. The size of this buffer is determined by the "LENGTH" parameter located in the device's configuration data.

Parameters are summarized as follows:

Parameter	Values	Function
IMNUM	1–64	Relative number of PCIM
DEVICENUM	0–31	Specifies device to which output word will be written
LENGTH	0–128	Size of data buffer in bytes
DEVDATA	variable	Buffer where the data to be written will be located – see above
STATUS		Success/Fail

Status Value

PUTDEVOUT will return SUCCESS if the device indicated is present on the given IMNUM and after the data is transferred to that device. If the target device is not present, or is out of range, the following FAIL indications will be returned:

BADIMNUM – IMNUM is out of range (a count of 64 or greater).

BADSBA – Specified DEVICENUM is not in the range for Genius bus devices (0 –31 decimal), or is that of the PCIM, which has its own function.

NOINIT – Indicated PCIM has not been initialized (INITIM).

IMFAIL – The indicated PCIM has failed (PCIM OK = 0).

OFFLINE – The device requested is currently not on the bus, and data is NOT transferred.

OVERFLOW – The Offset specified is greater than the device's InputLength in circuits.

UNDERFLOW – The Offset is specified as zero (0).

Coding Example

Write 2 bytes of output data to device #8 on PCIM #1.

```
2500 IMNUM = 1
2510 DEVICENUM = 8
2520 DEVDATA (0) = 1
2530 DEVDATA (1) = &H10
2540 LENGTH = 2
2550 Call PUTDEVOUT(STATUS,IMNUM,DEVICENUM,LENGTH,DEVDATA(0))
```

GETIMIN CALL Statement

Syntax

CALL GETIMIN (STATUS, IMNUM, IMDATA(0))

Action

Read Directed Input Table

Description

The Get IM Inputs call allows you to read the Directed Control Input Table of a specified PCIM and write its contents into the Host memory "IMDATA" parameter.

IMNUM is the PCIM number configured during initialization. The "IMDATA" parameter is a buffer where the data to be read will be located. The size of this buffer is determined by the "InputLength" parameter located in the PCIMs configuration data.

When GETIMIN is called, it begins by "Locking–out" the PCIM from updating the Directed Control Input Table (ensures data coherency across bus scans). GETIMIN then transfers all the data in this table into Host memory. Once the transfer is complete, the PCIM is "unlocked".

Parameters are summarized as follows:

Parameter	Values	Function
IMNUM	1–64	Relative number of PCIM
IMDATA	variable	Buffer where the data read will be located – see above
STATUS		Success/Fail

Status Value

GETIMIN will return SUCCESS if the InputLength is non–zero and the data transfer is complete. The following FAIL indications will be returned:

BADIMNUM	– IMNUM is out of range (a count of 64 or greater).
NOINIT	– Indicated PCIM has not been initialized (INITIM).
IMFAIL	– The indicated PCIM has failed (PCIM OK = 0).
UNDERFLOW	– The InputLength of the PCIM is set to zero (0).

Coding Example

Get the directed input data from PCIM #1.

```
2700 IMNUM = 1
2710 Call GETIMIN(STATUS,IMNUM,IMDATA(0))
```

PUTIMOUT CALL Statement

Syntax

CALL PUTIMOUT (STATUS, IMNUM, IMDATA(0))

Action

Write the Global Output Table

Description

The PUTIMOUT call allows you to write Global Data from the Host memory IMdata parameter to the Global Data Output Table of a specified PCIM. This data is subsequently broadcast to all CPUs on the bus every bus scan.

IMNUM is the PCIM number configured during initialization. The IMDATA parameter is a buffer where the data to be written is located. The size of this buffer is determined by the "GlobalLength" parameter located in the PCIM's configuration data.

When PUTIMOUT is called, it begins by "Locking–out" the PCIM from reading from its Control Output Table (ensures data coherency across bus scans). PUTIMOUT then transfers all the data from this parm to the PCIMs Global Output buffer. Once the transfer is complete, the PCIM is "unlocked".

Parameters are summarized as follows:

Parameter	Values	Function
IMNUM	1–64	Relative number of PCIM
IMDATA	variable	Buffer where the data to be written will be located. see above
STATUS	0/1	Success/Fail

Status Value

PUTIMOUT will return SUCCESS if the Global Data Length is non–zero and the transfer is complete. The following FAIL indications will be returned:

BADIMNUM – IMNUM is out of range (a count of 64 or greater).
NOINIT – Indicated PCIM has not been initialized (INITIM).
IMFAIL – The indicated PCIM has failed (PCIM OK = 0).
UNDERFLOW – The GlobalLength parameter in IMPARMS is set to zero (0).

Coding Example

Write the specified global data to PCIM #1.

```
2800 IMNUM = 1
2810 IMDATA (0) = &H10
2820 Call PUTIMOUT(STATUS,IMNUM,IMDATA(0))
```

GETCIR CALL Statement

Syntax

CALL GETCIR (STATUS, IMNUM, DEVICENUM, CIROFFSET, STATE)

Action

Read Input Circuit Value

Description

A Get Circuit call allows the state of a single input circuit to be read from the specified PCIMs Input Table and be placed into the Host memory "STATE" parameter.

IMnum is the PCIM number configured during initialization. The DEVICENUM parameter specifies the serial bus address of the device which contains the input circuit. The "CIROFFSET" parameter indicates which bit of DEVICENUM is to be read. This value ranges from 1 through 1024 (in bits).

"STATE" is a variable in which GETCIR will store the value of the circuit as indicated by the above parameters. The contents of STATE will be either a '1' or '0' (ON or OFF).

Parameters are summarized as follows:

Parameter	Values	Function
IMNUM	1–64	Relative number of PCIM
DEVICENUM	0–31	Specifies I/O device from which input circuit will be read
DIROFFSET	1–1024	Input circuit offset in specified I/O device in bits
STATE	0/1	ON or OFF condition of circuit read from PCIM
STATUS		Success/Fail

Status Value

GETCIR will return SUCCESS if the target device is present on the given IMNUM. If the target device is not present, or is out of range, GETCIR will return FAIL. If SUCCESS is returned, then STATE will contain the value of the circuit requested. The following FAIL indications will be returned:

BADIMNUM	–	IMNUM is out of range (a count of 64 or greater).
BADSBA	–	Specified DEVICENUM is not in the range for bus devices (0 –31 decimal), or is that of the PCIM.
NOINIT	–	Indicated PCIM has not been initialized (INITIM).
IMFAIL	–	The indicated PCIM has failed (PCIM OK = 01).
OFFLINE	–	The device requested is currently not on the bus, and data is NOT transferred.
OVERFLOW	–	The OFFSET specified is greater than the devices InputLength in circuits.
UNDERFLOW	–	OFFSET is specified as zero (0).

Coding Example

Get the state value of circuit 2 of device #8 on PCIM #1.

```
3000 IMNUM = 1
3010 DEVICENUM = 8
3020 CIROFFSET = 2
3030 Call GETCIR(IMNUM,DEVICENUM,CIROFFSET,STATE)
```

PUTCIR CALL Statement

Syntax

CALL PUTCIR (STATUS, IMNUM, DEVICENUM, CIROFFSET, STATE)

Action

Write Output Circuit Value

Description

A Put Circuit call allows the state of a single output circuit to be changed from ON to OFF or vice–versa. In this call, the STATE parameter is written from the Host memory to the specified PCIMs Output Table.

IMNUM is the PCIM number configured during initialization. The DEVICENUM parameter specifies the serial bus address of the device which contains the target output circuit. The CIROFFSET parameter indicates which bit of DEVICENUM is to be written. This value ranges from 1 through 1024 (in bits).

STATE is a variable containing the value of the circuit as indicated by the above parameters. The contents of STATE should be either a '1' or '0' (ON or OFF).

Parameters are summarized as follows:

Parameter	Values	Function
IMNUM	1–64	Relative number of PCIM
DEVICENUM	0–31	Specifies I/O device to which output will be written.
CIROFFSET	1–1024	Output circuit offset in specified I/O device, in bits
STATE	0/1	Variable "STATE" is written from the Host memory to the specified PCIM
STATUS		Success/Fail

Status Value

PUTCIR will return SUCCESS if the target device is present on the given IMNUM. If the target device is not present, or is out of range, PUTCIR will return FAIL. If SUCCESS is returned, then the character pointed to by STATE will contain the value of the circuit changed. The following FAIL indications will be returned:

BADIMNUM	IMNUM is out of range (a count of 64 or greater).
BADSBA	Specified Devicenum is not in the range for Genius bus devices (0 –31 decimal), or is that of the PCIM.
NOINIT	Indicated PCIM has not been initialized (InitIM).
IMFAIL	The indicated PCIM has failed (PCIM OK = 0).
OFFLINE	The device requested is currently not on the bus, and data is NOT transferred.
OVERFLOW	The Offset specified is greater than the devices Output-Length in circuits.
UNDERFLOW	The Offset is specified as zero (0).

Coding Example

Set the state value of circuit 2 of device #8 on PCIM #1 to '1'.

```
3200 IMNUM = 1
3210 DEVICENUM = 8
3220 STATE = 1
3230 CIROFFSET = 2
3240 Call PUTCIR(STATUS,IMNUM,DEVICENUM,CIROFFSET,STATE)
```

GETWORD CALL Statement

Syntax

CALL GETWORD (STATUS, IMNUM, DEVICENUM, CIROFFSET, WORDDATA)

Action

Read Input Word Value

Description

A Get Word call allows you to read the value of a single input word from the specified PCIMs Input Table into the Host memory ”WORDDATA” parameter. The ”WORDDATA” parameter is an integer.

IMNUM is the PCIM number configured during initialization. The DEVICENUM parameter specifies the serial bus address of the device where the input word is located. The CIROFFSET parameter indicates which word of the specified device is to be read. This value ranges from 1 through 64 (in word quantities).

When GETWORD is called, it begins by “Locking–out” the PCIM from updating the Shared RAM (ensures data coherency across bus scans). GETWORD then transfers the word data into Host memory. Once the transfer is complete, the PCIM is ”unlocked”.

Parameters are summarized as follows:

Parameter	Values	Function
IMNUM	1–64	Relative number of PCIM
DEVICENUM	0–31	Specifies I/O device from which input word will be read
OFFSET	1–64	Input word offset in specified I/O device, in words
WORDDATA	1 entry	Word requested
STATUS		Success/Fail

Status Value

GETWORD will return SUCCESS if the device specified by IMNUM is present on the serial bus, and after the data is transferred to WORDDATA. If the target device is not present, or is out of range, GETWORD will return FAIL. If SUCCESS is returned, then the requested word value will be saved in the location WORDDATA. The following FAIL indications will be returned:

BADIMNUM	IMNUM is out of range (a count of 64 or greater).
BADSBA	Specified Devicenum is not in the range for Genius bus devices (0 –31 decimal), or is that of the PCIM.
NOINIT	Indicated PCIM has not been initialized (InitIM).
IMFAIL	The indicated PCIM has failed (PCIM OK = 0).
OFFLINE	The device requested is currently not on the bus, and data is NOT transferred.
OVERFLOW	The Offset specified is greater than the devices InputLength in circuits.
UNDERFLOW	The Offset is specified as zero (0).

Coding Example

Get the first word of device #8 on PCIM #1.

```
3300 IMNUM = 1
3310 DEVICENUM = 8
3320 CIROFFSET = 1
3330 Call GETWORD(STATUS,IMNUM,DEVICENUM,CIROFFSET,WORDDATA)
```

PUTWORD CALL Statement

Syntax

CALL PUTWORD (STATUS, IMNUM, DEVICENUM, CIROFFSET, WORDDATA)

Action

Write Output Word Value

Description

A Put Word call allows you to write a single output word from the Host memory WORDDATA parameter to the specified PCIMs Output Table. The WORDDATA parameter is an integer which PUTWORD uses for the word to be transmitted.

IMNUM is the PCIM number configured during initialization. The DEVICENUM parameter specifies the serial bus address of the device where the output word is to be sent. The CIROFFSET parameter indicates which word of the specified device is to be written. This value ranges from 1 through 64 (in word quantities).

When PUTWORD is called, it begins by “Locking–out” the PCIM from updating the Shared RAM (ensures data coherency across bus scans). PUTWORD then transfers the word data to the PCIM. Once the transfer is complete, the PCIM is ”unlocked”.

Parameters are summarized as follows:

Parameter	Values	Function
IMNUM	1–64	Relative number of PCIM
DEVICENUM	0–31	Specifies device to which output word will be written
CIROFFSET	1–64	Output word offset in specified device, in words
WORDDATA	1 entry	Word requested
STATUS		Success/Fail

Status Value

PUTWORD will return SUCCESS if the device specified by IMNUM is present on the serial bus. If the target device is not present, or is out of range, PUTWORD will return FAIL. The following FAIL indications will be returned:

BADIMNUM	IMNUM is out of range (a count of 64 or greater).
BADSBA	Specified Devicenum is not in the range for Genius bus devices (0 –31 decimal), or is that of the PCIM.
NOINIT	Indicated PCIM has not been initialized (InitIM).
IMFAIL	The indicated PCIM has failed (PCIM OK = 0).
OFFLINE	The device requested is currently not on the bus, and data is NOT transferred.
OVERFLOW	The Offset specified is greater than the devices OutputLength in circuits.
UNDERFLOW	The Offset is specified as zero (0).

Coding Example

Set the second word of device #8 on PCIM #1 to 10 hex (circuit #21 if discrete block).

```
3400 IMNUM = 1
3410 DEVICENUM = 8
3420 CIROFFSET = 2
3430 WORDDATA = &H10
3440 Call PUTWORD(STATUS,IMNUM,DEVICENUM,CIROFFSET,WORDDATA)
```

SENDMSG CALL Statement

Syntax

CALL SENDMSG (STATUS, IMNUM, MSG(0))

Action

Send a Message

Description

The Send Message call allows you to write a memory or non–memory message from the Host to the selected PCIM for transmission onto the serial bus (using the Transmit Datagram command). SENDMSG will return control to the calling program without delay, before the message has been processed or transmitted by the PCIM.

IMNUM defines the PCIM, as configured during initialization, from which to transmit the message. The MSG parameter is the buffer where the transmit message is stored.

The format of SENDMSG is:

MSG(0)	Destination (0–31/255 brdcst)	– Destination address of Device
MSG(1)	Function code (0–111)	– Function Code (normally 20 hex)
MSG(2)	SubFunction code (0–255)	– Sub Function Code
MSG(3)	Priority	– 0 – Normal, 1 – High
MSG(4)	Length	– Data field length/length of msg
MSG(5)	Data (variable)	– Message Data – length per MSG(4)

You should check the status of the message using CHKMSGSTAT to determine if the message completed processing properly.

Parameters are summarized as follows:

Parameter	Values	Function
IMNUM	1–64	Relative number of PCIM
MSG	see above	Buffer where message to be sent is stored – see above
STATUS		Success/Fail

Status Value

SENDMSG will return SUCCESS if a message has been transferred from the Host memory to the PCIM. Otherwise, one of the following FAIL indications will be returned:

BADIMNUM – IMNUM is out of range (a count of 64 or greater).

NOINIT – Indicated PCIM has not been initialized (INITIM).

IMFAIL – The indicated PCIM has failed (PCIM OK = 0).

IMBUSY – The PCIM is otherwise engaged and cannot accept the command.

Note

You are responsible for defining the device, the Function code, the Sub–F unction code and the length of the transmit Datagram.

Note

You cannot issue a SENDMSG call or read a received unsolicited message while a SENDMSGREPLY call is in progress. If this presents a timing problem, use the SENDMSG call.

See Also: SENDMSGREPLY, GETMSG and CHKMSGSTAT

Coding Example

Send a Read Diagnostics message to device #8 on PCIM #1. This message will read 10 bytes of diagnostic data beginning at offset 0.

```
3800 IMNUM = 1
3810 MSG(0) = 8 'Destination
3820 MSG(1) = &H20 'Function Code
3830 MSG(2) = 8 'Sub Function Code
3840 MSG(3) = 0 'Priority
3850 MSG(4) = 2 'Message Length Sent
3860 MSG(5) = 0 'Offset
3870 MSG(6) = 10 'Length to be Read
3880 Call SENDMSG(STATUS,IMNUM,MSG(0))
```

To see how the message function calls work together, see Appendix A, Example 2.

SENDMSGREPLY CALL Statement

Syntax

CALL SENDMSGREPLY (STATUS, IMNUM, MSG(0))

Action

Send a Message requesting a Reply

Description

The Send Message Reply call allows you to write a memory or non–memory message from the Host to the selected PCIM for transmission onto the bus (using the Transmit Datagram With Reply command). SENDMSGREPLY will return control to the calling program without waiting for the reply. You must call CHKMSGSTAT or GETMSG to check for completion or to read the reply message.

IMNUM defines the PCIM, as configured during initialization, from which to transmit the message. The MSG parameter is a pointer to the buffer where the transmit message is stored.

The format of SENDMSGREPLY is:

MSG(0)	Destination (0–31/255 brdcst)	– Destination address of Device
MSG(1)	Function code (0–111)	– Function Code
MSG(2)	T SubFunction code (0–255)	– Transmitted Reply SubFunction Code
MSG(3)	R SubFunction code (0–255)	– Expected Reply SubFunction Code
MSG(4)	Priority	– 0 – Normal, 1 – High
MSG(5)	Length (0–134)	– Data field length/length of msg
MSG(6)	Data (variable)	– Message Data – length per MSG(5)

You can check the status of the message using CHKMSGSTAT to determine if the message completed processing properly.

Parameters are summarized as follows:

Parameter	Values	Function
IMNUM	1–64	Relative number of PCIM
MSG	see above	Pointer to the buffer where the received message will be stored – see above
STATUS		Success/Fail

The advantage of the SENDMSGREPLY call over the SENDMSG call is that a 10 second timeout to a non–responding device is automatically provided by the PCIM for a SENDMSGREPLY call.

The Host program sequence for a SENDMSGREPLY is as follows:

1. Host sends a SENDMSGREPLY to the PCIM.
2. Host issues GETMSG calls until the Status indicates completion. GETMSG will also return the reply message into Host memory.

Status Value

SENDMSGREPLY will return SUCCESS if a message has been transferred from the Host memory to the PCIM. Otherwise, one of the following FAIL indications will be returned:

BADIMNUM – IMNUM is out of range (a count of 64 or greater).

NOINIT – Indicated PCIM has not been initialized (InitIM).

IMFAIL – The indicated PCIM has failed (PCIM OK = 0).

IMBUSY – The PCIM is otherwise engaged and cannot accept the command.

Note

You are responsible for defining the device, the Function code, the Sub–F unction code and the length of the transmit Datagram.

It is also your responsibility to interpret the Function code, the Sub–F unction code and the meaning of the Reply message. See GEK–90486–1 for message codes.

Note

You cannot issue a SENDMSG call or read a received unsolicited message while a SENDMSGREPLY call is in progress. If this presents a timing problem, use the SENDMSG call.

See Also: SENDMSG, GETMSG and CHKMSGSTAT

Coding Example

This example sends a Read Diagnostics message to device #8 on PCIM #1 and expects a reply message of Read Diagnostics Reply. This message requests 10 bytes of diagnostic data beginning at offset 10.

```
4000  IMNUM   = 1
4010  MSG(0)  = 8 'Destination
4020  MSG(1)  = &H20 'Function Code
4030  MSG(2)  = 8 'Transmit SubFunction Code
4040  MSG(3)  = 9 'Excepted Reply SubFunction Code
4050  MSG(4)  = 0 'Priority
4060  MSG(5)  = 2 'Message Length Transmitted
4070  MSG(6)  = 16 'Offset
4080  MSG(7)  = 10 'Message Length to be Read
4090  Call SENDMSGREPLY(STATUS,IMNUM,MSG(0))
```

To see how the message function calls work together, see Appendix A, Example 2.

CHKMSGSTAT CALL Statement

Syntax

CALL CHKMSGSTAT (STATUS, IMNUM, MSGSTATUS(0))

Action

Read Message Progress Status

Description

The Check Message Status call allows you to determine the status of a previous SENDMSG call – that is, to determine when a transmitted message has actually been received, and its completion status.

IMNUM is the PCIM number configured during initialization. The "MSGSTATUS" parameter is the returned message status.

The "MSGSTATUS" parameter will contain the following values:

IMFREE	There is currently no activity.
IMBUSY	Message is still in progress.
SUCCESS	Message has successfully completed.
BADPARM	Message contained a syntax error.
TXERR	Message cannot be transmitted.

Parameters are summarized as follows:

Parameter	Values	Function
IMNUM	1–64	Relative number of PCIM
MSGSTATUS	0/1	Returned message status
STATUS		Success/Fail

Status Value

CHKMSGSTAT will normally return the Status requested and a SUCCESS indication. Other wise, one of the following FAIL indications will be returned:

BADIMNUM	–	IMNUM is out of range (a count of 64 or greater).
NOINIT	–	Indicated PCIM has not been initialized (INITIM).
IMFAIL	–	The indicated PCIM has failed (PCIM OK = 0).
PCIMERR	–	There may be a problem with the PCIM firmware.

See Also: SENDMSGREPLY, SENDMSG and GETMSG

Coding Example

Check the message status area of PCIM #1.

```
4200 IMNUM = 1
4210 Call CHKMSGSTATUS(STATUS,IMNUM,MSGSTATUS)
```

To see how the message function calls work together, see Appendix A, Example 2.

GETMSG CALL Statement

Syntax

CALL GETMSG (STATUS, IMNUM, MSG(0))

Action

Read Received Message

Description

The Get Message call allows you to read a received memory or non–memory message (or a reply to a previous SENDMSGREPLY call) from the selected PCIM into the Host memory "MSG" parameter.

IMNUM is the PCIM number configured during initialization. The "MSG" parameter is the buffer where the received message will be stored.

The format of GETMSG is:

MSG (0)	Source (0–31)	– Source address of Device
MSG (1)	Function code (0–111)	– Function Code
MSG (2)	SubFunction code (0–255)	– Sub Function Code
MSG (3)	DB Indicator	– Directed (1)/Broadcast (0)
MSG (4)	Length (0–134)	– Data field length/length of message
MSG (5)	Data (variable)	– Message Data – length per MSG(4)

Parameters are summarized as follows:

Parameter	Values	Function
IMNUM	1–64	Relative number of PCIM
STATUS		Success/Fail
MSG	see above	Buffer where the received message will be stored

GETMSG performs the following sequence:

1. If there is a previous call to SENDMSGREPLY, GETMSG checks to see if the transmission has successfully completed, and transfers the response back to you. If the response completed with an error, or if in progress, GETMSG will return a FAIL indication.
2. If there is no previous call to SENDMSGREPLY, GETMSG checks to see if there is a memory message, and transfers that message back to you.
3. If no memory messages exist, then GETMSG checks to see if there is a non–memory message, and transfers that message back to you.
4. If no messages are present, GETMSG returns with a FAIL status.

Note

Unsolicited memory or non–memory Datagrams received by the PCIM may not be read by the Host while a SENDMSGREPLY is in progress. This significantly affects Host response time to service received Datagrams. If this is a problem, use the SENDMSG call instead of SENDMSGREPLY.

Status Value

GETMSG will return SUCCESS if a memory or non–memory message is returned to you. Otherwise, one of the following FAIL indications will be returned:

BADIMNUM	– IMNUM is out of range (a count of 64 or greater).
NOINIT	– Indicated PCIM has not been initialized (INITIM).
IMFAIL	– The indicated PCIM has failed (PCIM OK = 0).
IMBUSY	– The PCIM is otherwise engaged and cannot accept the command.
NOMSG	– No message is available to be received at this time.
TXERR	– A message transmission has failed or response to SendMsgReply has not arrived after 10 seconds.
PCIMERR	– There may be a problem with the PCIM firmware.
BADPARM	– Syntax error from previous SendMsgReply

See Also: SENDMSGREPLY, SENDMSG and CHKMSGSTAT

GETINTR CALL Statement

Syntax

CALL GETINTR (STATUS, IMNUM, INTR(0))

Action

Read Interrupt Status Table

Description

The Get Interrupt call allows you to read the selected PCIMs Interrupt Status Table. You can read this table to:

- See why an interrupt in the Host system has occurred.
- Report the event in a non–interrupt environment, as is the default state of the Software Driver concept (the PCIM will still report the event even though the interrupt is disabled).

Thus, the Interrupt Status Table can be polled (by reading and interpreting it) to determine what is causing an interrupt from the PCIM.

When GETINTR is called, it transfers the data from the PCIMs Interrupt Status Table to the Host memory "INTR" parameter. The format of the Interrupt Status Table is shown below.

IMNUM defines the PCIM, as configured during initialization, from which the Interrupt Status Table is to be read. The INTR parameter is the buffer where the Interrupt Status Table information is stored. The values in the table below are: 0 = No interrupt occurred, 1 = Interrupt occurred.

The format of the INTR table is:

Position	Explanation
INTR(0)	– Summary if interrupt occurred.
INTR(1)	– Received memory datagram.
INTR(2)	– PCIM Status Change – usually fatal.
INTR(3)	– Device Status Change.
INTR(4)	– Outputs sent – end of bus access.
INTR(5)	– Command Block completed.
INTR(6)	– Received Datagram.

After data transfer to the Host is complete, GETINTR clears all of the PCIMs Interrupt Status Table bytes each time it is called. This way, you can see the latest event that has occurred each call.

Parameters are summarized as follows:

Parameter	Value	Function
IMNUM	1–64	Relative number of PCIM
INTR	see above	Buffer where the table data will be stored
Status		Success/fail

Status Value

GETINTR will return SUCCESS if the device specified by IMNUM is present on the serial bus. If the target device is not present, or is out of range, GETINTR will return FAIL. The following FAIL indications will be returned:

BADIMNUM – IMUM is out of range (a count of 64 or greater).

NOINIT – Indicated PCIM has not been initialized (INITIM).

IMFAIL – The indicated PCIM has failed (PCIM OK = 0).

Coding Example

This example shows how, if an interrupt occurs on PCIM #1, to transfer the contents of that PCIMs status table. Interpretation of bits will depend on which interrupt is enabled, and which application is to be run.

```
4300 IMNUM = 1
4310 Call GETINTR(STATUS,IMNUM,INTR(0))
4320 'Do what is necessary for interrupt processing
```

PUTINTR CALL Statement

Syntax

CALL PUTINTR (STATUS, IMNUM, DISABLEINTR(0))

Action

Write to the Interrupt Disable Table

Description

The Put Interrupt call allows you to write to the selected PCIM's Interrupt Disable Table. The PUTINTR call first initializes a table to Enable and Disable individual interrupts as you require. The PUTINTR call then writes this table to the Interrupt Disable Table on the PCIM. You can Enable or Disable interrupts in any mix; that is, on a single call, some interrupts may be Enabled and some Disabled, all may be Enabled, or all of the interrupts may be Disabled.

When PUTINTR is called, it transfers the data from the Host memory "DISABLEINTR" parameter to the PCIMs Interrupt Disable Table. The format of the Interrupt Disable Table is shown below.

IMNUM defines the PCIM, as configured during initialization, to which DISABLEINTR will be read. The DISABLEINTR parameter is the buffer where the Interrupt Disable Table information is stored. The values in the table below are: 0 = Enable, 1 = Disable.

The format of the DISABLEINTR table is:

Position	Explanation
DISABLINTR(0)	– Summary if interrupt occurred.
DISABLINTR(1)	– Received memory datagram.
DISABLINTR(2)	– PCIM Status Change – usually fatal.
DISABLINTR(3)	– Device Status Change.
DISABLINTR(4)	– Outputs sent – end of bus access.
DISABLINTR(5)	– Command Block completed.
DISABLINTR(6)	– Received Datagram.

Parameters are summarized as follows:

Parameter	Values	Function
IMNUM	1–64	Relative number of PCIM
DISABLEINTR	see above	Buffer from which enable/disable data is sent
STATUS		Success/fail

Status Value

PUTINTR will return SUCCESS if the device specified by IMNUM is present on the serial bus. If the target device is not present, or is out of range, PUTINTR will return FAIL. The following FAIL indications will be returned:

BADIMNUM	– IMNUM is out of range (a count of 64 or greater).
NOINIT	– Indicated PCIM has not been initialized (InitIM).
IMFAIL	– The indicated PCIM has failed (PCIM OK = 0).

Coding Example

This example enables the Receive Datagram Interrupt.

```
7000 IMNUM = 1
7010 For I = 0 to 6
7020 DISABLEINTR(I) = 0
7030 NEXT I
7040 DISABLEINTR(6) = 1
7050 Call PUTINTR(STATUS,IMNUM,DISABLEINTR(0))
```

Chapter 6 Communications

Introduction

PCIM applications may be considered on two levels; 'basic' operation, consisting of that which is necessary to set up the PCIM and use it as a simple I/O controller; and 'advanced' operation. Advanced operation details the use of expanded diagnostics, message handling, and other more sophisticated features – a class of applications dependent on the Genius I/O Network for low cost, peer–to–peer moderate performance communications between Hosts and I/O devices.

Chapters 4 and 5 outlined the 'basic' operational level – providing you with enough information to code the PCIM Software Driver function calls and run a system consisting of I/O blocks. Chapter 6 explains the 'advanced' communications features of the PCIM.

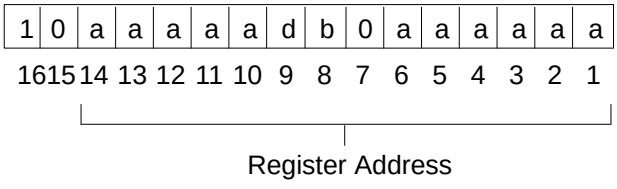
Global Data

Global Data is data used for communicating data between CPUs simply, automatically, and repetitively. Once set up by the user at power up, assigned data is automatically and periodically routed among CPUs without further user programming. Such data is termed "Global Data" since it is broadcast to all other CPUs on the bus and thus allows the formation of a global data base. Up to 128 bytes may be broadcast by each PCIM or Bus Controller. The PCIM or Bus Controller will broadcast these bytes once per bus scan.

A block of data is assigned to be broadcast by downloading a Global Data Reference and Global Data Length. The Global Data Reference is the beginning address of the Global Data where a receiving Series Six or Series Five PLC will place the data. A Series 90-70 or Series 90-30 PLC does not use this reference. If no PLCs are involved, this reference can be defined for any suitable application purpose. This reference is called IMRef in the PCIM. The Global Data length is the number of bytes of Global Data to be broadcast by the PCIM.

Global Data Length is called OutputLength. You will use the Software Driver function call InitIM to set IMRef and OutputLength parameters. Always set the MSB (Most Significant Bit) of the IMRef to '1', if it is to be used with a Series Six or Series Five PLC.

The location where the receiving host will place the Global Data can be specified using IMRef. The 16 bit register address must have the two upper bits set as shown below. A Series Six or Series Five PLC will only use the bottom 14 bits.



46233

For example:

> IMRef = 8005 hex will send Global Data to all Series Six / Series Five CPUs on the bus starting at Register 5. The Global Data Length (OutputLength) is always specified in bytes. Therefore, if 15 Registers of Global Data are to be sent, OutputLength should be set to 1E hex, 30 decimal.

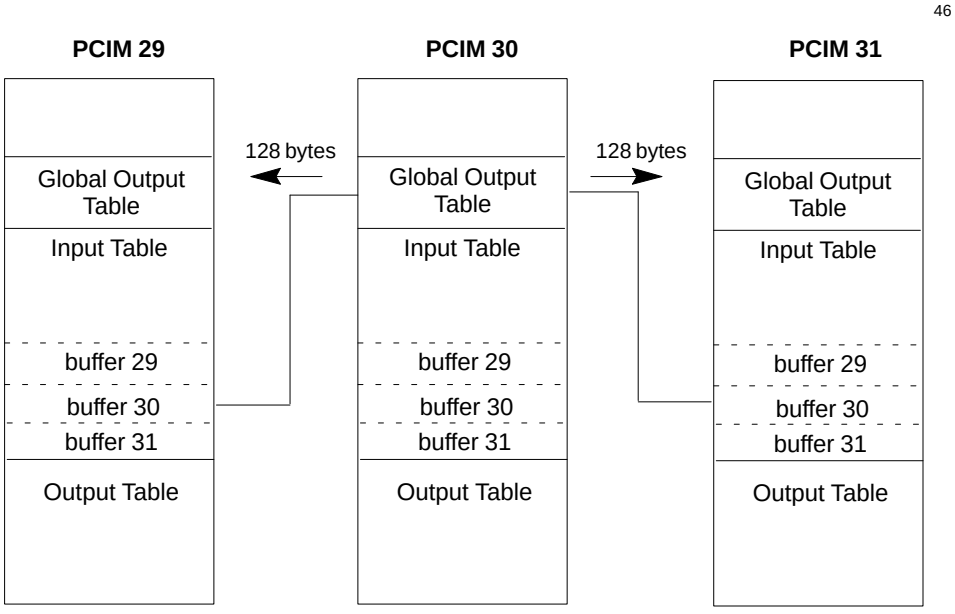
Global Data is automatically broadcast by the PCIM every serial bus scan. The user application program updates the PCIM with the latest Global Data by using the Software Driver PutIMOut.

If the program sweep time is longer than the bus scan time, new Global Data may not be available each bus scan; in that case, the same data may be sent more than once. If the program sweep time is much shorter than the bus scan time, it is possible that Global Data might change more quickly than the bus controller can send it. If that happens, some data could be lost. The host must accommodate these timing issues to assure data integrity. *Bus scan time increases by approximately 72µS for each byte of Global Data transmitted.*

When the PCIM receives Global Data, it appears in the Input Table slot corresponding to the Serial Bus Address (device number) of the device that sent the Global Data. You will use the Software Driver function call GetBusIn or GetDevIn to read this data.

Example:

In a three–host system, the PCIM with serial bus address 30 broadcasts 128 bytes of Global Data to PCIMs with serial bus addresses 29 and 31.



For More Information

For more information about Genius datagrams, refer to the *Genius I/O System and Communications User's Manual* (GEK-90486-1).

Datagram Data

A Datagram is a message comprised of application–specific information with up to 128 bytes of user supplied data. Datagrams may be directed from one bus device to another, or broadcast to all devices.

A directed Datagram is secure in that the data link control layer of the protocol ensures it will be received at the destination device once and only once, or aborted and alarmed after retry.

Datagram Service should be considered instead of Global Data if any of the following are true:

1. Global Data takes up too much serial bus scan time for the application
2. More than 128 bytes of data are to be sent from one CPU to another
3. The data does not need to be sent every serial bus scan
4. The PLC CPU sweeptime receiving Global Data becomes too large for the application.

A Datagram may be transmitted with High Priority or at Normal Priority. Normal Priority ensures that the bus scan time will only be modestly affected. High priority will be sent as soon as the token is held by the PCIM. Normal priority requires that no datagrams be sent for 1 bus scan prior to transmission of this datagram. Bus scan time affects the response time of any I/O data on the bus.

> **Using the same serial bus for CPU to CPU communications and I/O block control may result in variable I/O service times unless Normal Priority datagrams are used.**

Your application must service the Datagram queue at least once every 10 milliseconds to ensure that the Datagram queue will not fill up, causing datagrams to be dropped without Host notification.

Use the Software Driver function calls GetMsg, SendMsg, SendMsgReply, and ChkMsgStat to transmit Datagrams. For the the bit/byte format of the following specific Genius I/O Datagrams, see the *Genius I/O System and Communications Manual*, GEK–90486–1.

The following Datagrams are transmitted to and from I/O blocks:

Report Fault – faults are reported as they occur to the defined Controller of a specific I/O block or device. The controller of a device is the device which sends outputs to the device. The GetMsg call is used to access this message from the PCIM.

Clear Circuit Fault – the Host may clear a single circuit or controller fault using this message. The Host requires a SendMsg call to transmit this message to an I/O Block.

Clear All Circuit Faults – the Host may clear all circuit faults using this message. The Host again requires a SendMsg call to transmit this message.

Write Configuration – downloads either partial or full configuration from the Host to an I/O Block or other bus device. The Host requires a SendMsg call to transmit this message to an I/O Block.

Read Diagnostics, Read Diagnostics Reply – allows the Host to read the current diagnostic state of all circuits or controllers. Use a SendMsgReply call, then a GetMsg call to perform this function using the PCIM.

Read Configuration, Read Configuration Reply – allows the Host to read the current configuration of an I/O Block or I/O device. Use a SendMsgReply call, then a GetMsg call to perform this function with the PCIM.

Switch BSM – allows the Host to switch a Bus Switching Module (BSM) to a specified bus and therefore test redundant bus operation while a system is running. The Host requires a SendMsg call to send this message to an I/O block.

Assign Monitor – allows the Host to receive a Report Fault message from an I/O block even though it is not defined as the controller of (is not sending data to) that device. Use a SendMsg call to send this message to the block.

Pulse Test, Pulse Test Complete – allows the Host to toggle all outputs on a specific discrete I/O block briefly to the opposite state. Any faults are reported from the block to the Host through a Report Fault message, and the block will reply with a Pulse Test Complete message when the test is finished. The Host uses a SendMsgReply call to transmit this message to the block, and a GetMsg call to retrieve the reply and any fault reports.

Configuration Change – I/O blocks and other I/O devices will report any configuration changes of I/O circuit configuration, Status Table (Reference) Address, HHM forces, filter values, etc. The Host requires a GetMsg call to access this message from the PCIM.

Memory-Access Datagrams

Communications applications of the PCIM will for the most part be established between devices such as PLC CPUs and PCIM Hosts (IBM PC AT/XT). These applications will use four memory access Datagrams.

Read Device – the CPU may read the memory of another CPU on the bus through this message. The CPU may use the SendMsgReply call, then the GetMsg call, in order to send the Read Device message and access the eventual reply, respectively.

Read Device Reply – When a Read Device message is received, the PCIM (and Host) will service it by returning a Read Device Reply to the requesting CPU through the SendMsg call.

Write Device – the CPU may write the memory of another CPU using this message. Write Device allows byte writes. Use the SendMsg call to transmit this message.

Bit Write – the CPU may write the memory of another CPU using this message. Bit Write is for setting or resetting a single circuit. Use the SendMsg call to transmit this message.

These Datagrams allow the registers or I/O Tables of a PLC CPU to be read or written from other bus devices.

If a Host wishes its internal database to be accessible, user application programming must supply GetMsg calls to service Read Device and Write Device messages received by the PCIM. The PCIM Host need not allow Write Device access to its memory. This can be accomplished by rejecting all or specific Write Device messages.

Software Driver function calls are also used to transmit Datagram data.

The SendMsg call is used to send Read Device and Write Device datagams.

For More Information

For applications using datagrams, refer to the *Genius I/O System and Communications User's Manual* (GEK-90486-1) for detailed information.

Chapter 7 Troubleshooting

Introduction

As with program debugging, hardware / firmware troubleshooting is accomplished by thinking logically of the function of each part of the system and how these functions interrelate. A basic understanding of the various indicator lights will help you quickly isolate the problem to the PCIM, a Bus Controller, an I / O rack, an I / O Block, or the CPU.

The total system has to be considered when problems occur. The CPU, Host computer, I / O Blocks and external devices connected to or controlled by the Genius I / O system must all be operating and connected properly. All cable connections as well as all screw–down or soldered connections should be checked carefully.

Sometimes you need someone to talk to who can answer your questions. When you do, first call your local authorized GE distributor. After business hours, please do not hesitate to call the Programmable Control Emergency Service Number, (804) 978–5747.

Replacement Module Concept

When a problem arises, first isolate it to the major assembly, then to the defective module within that assembly. The defective module is then replaced from a duplicate set of modules maintained on site. Your production line or system is back up fast.

The defective module can be returned through normal channels under warranty or for service without keeping your production line or system down for an extended period of time. The replacement concept minimizes downtime to minutes as contrasted (potentially) to days. The potential savings far outweigh the comparatively small cost of duplicate modules.

If you did not purchase a duplicate set of modules with your initial system, we recommend that you contact your authorized GE distributor and do so. Then, with the help of this manual and the staff of your local authorized GE distributor, you will be able to troubleshoot and repair just about any problem that may arise.

PCIM Troubleshooting

LEDS

A malfunction causing the improper operation of a PCIM can generally be isolated by checking the condition of the status indicator LEDs on the PCIM. The normal condition of the status indicator LEDs is the ON state. If a LED is not ON, check the troubleshooting sequence in this section for the proper course of action.

Indicator	Status	Definition
BOARD OK	ON	Power is available to the PCIM (adequate power must be available for it to function properly), and the on–board self–diagnostics test was passed.
	OFF	The watchdog timer has timed out, indicating a board failure or improper address assignment or /RST input line is low.
COMM OK	ON	Power is available, the controller's communications hardware is functional, and it can send data (receives the token) every serial bus scan.
	OFF	(or FLASHING) means an error has been detected in the communications hardware or access to the Genius serial bus.

Fault Isolation and Repair

If the status indicator LEDs are in the correct state but the bus is not functioning properly, the malfunctions below may describe the problem. If so, follow the procedures listed under the appropriate malfunction.

- An LED does not come ON when a PCIM is plugged in and powered up and /RST input is high.

  If Board OK OFF / Comm OK ON –

  - □ Check the parameters entered using the configuration software.

  If set different then the InitIM parameter, the BOARD OK LED will not come on.

  - □ Check to see if the PCIM is completely inserted in the host backplane connector, and that all connector pins are properly aligned.

  If all appears to be in order, assume hardware failure – replace PCIM.

  If Board OK ON / Comm OK OFF –

  - □ Check for correct cable type and length (see *Genius I/O System and Communications User's Manual*, GEK–90486–1).
  - □ See if correct terminating resistors (see *Genius I/O System and Communications User's Manual*, GEK–90486–1) are installed at both ends of bus.
  - □ Determine if serial bus wiring has been completed in a daisy chain fashion.
  - □ Make sure cabling is not in proximity to high voltage runs.
  - □ Look for a broken cable.

If both LEDs off –

- □ Check to see if the PCIM is plugged in, seated properly, and receiving power.
- □ Check voltage receiving level of /RST. It must remain at 2.4 volts or higher (TTL logic 1).

If both LEDs flashing together –

- □ Two devices on the same bus have probably been configured with the same device number (serial bus address).

Check using the HHM.

- Repeated bus errors
  - □ Ensure that cable shielding is properly installed and grounded (see *Genius I/O System and Communciations User's Manual*, GEK–90486–1).
  - □ Unplug bus communications cable from PCIM, refer to the Device number sheets from which you configured the system, and use the HHM to read configuration / compare device numbers and I/O reference numbers.
  - □ If all appears to be in order, replace PCIM.
- System shuts down with parity errors.

  Duplicate or overlapping PCIM / I/O References.
  - □ Input duplicated on same bus.
  - □ Input references from other PCIMs overlap.
- Bus Errors – cannot get PCIM up and running
  - □ Serial 1 / Serial 2 crossed
- Intermittent or total lack of communications.
  - □ Mixed Baud Rates

  Power up blocks one at a time and confirm baud rate. Any change to baud rate in block will not take effect until block power is cycled.
- No Global Data.
  - □ Destination device off–line

  Verify destination on–line.
- Unsuccessful Datagram completion.
  - □ Destination device off–line

  Verify destination on–line.

Appendix A Example Applications

Example Application 1

This programming example uses the InitIM and GetDevConfig function calls. Example devices include two PCIMs connected to a serial bus. The PCIMs have the following Configurations (The IMPARMS Structure is defined in PCIM.H):

PCIM #2

Serial Bus Address:	30 Dec
IMPARMS.Segment:	D000 Hex
IMPARMS.IOPort:	3E4 Hex
IMPARMS.IMRef:	3434 Hex
IMPARMS.OutputLength:	0
IMPARMS.InputLength:	0
IMPARMS.Active:	ON

PCIM #1

Serial Bus Address:	31 Dec
IMPARMS.Segment:	CC00 Hex
IMPARMS.IOPort:	3E0 Hex
IMPARMS.IMRef:	1212 Hex
IMPARMS.OutputLength:	0
IMPARMS.InputLength:	0
IMPARMS.Active:	ON

These are the only two devices on our example Genius bus. The GetBusConfig function can be used for any device on the bus by giving the Serial Bus Address (device number) of the device desired. If the Device given is not online, GetDevConfig will return OFFLINE (11).

This example can be built using MicroSoft C Compiler Ver 4.0 or greater with the following syntax:

```
      C> MSC gdctst  /Zp;
      C> LINK gdctst, , , pcim;
*/

#include      <stdio.h>
#include      <pcim.h>            /* PCIM header file */

extern    int
        InitIM( ),
        ChgIMSetup( ),
        GetDevConfig( );

IMPARMS local[2];    /* PCIM Configuration Structure. . .
                        allocate an element per PCIM in your PC */

char    flags[2];    /* Error return for PCIM Init. . .allocate an
                        element per PCIM in your PC. */

DEVICE  config;      /* Device Config Structure. . .32 may be allocated */

#define PCIM1   &local[0]         /* Macro for easier remembering */
#define PCIM2   &local[1]         /* Macro for easier remembering */

*/

main( )
{
    int       ret,
              x,
              y,
              loop = 1;

          printf("\n\nThis is a test of the GetDevConfig function. . .\n");

    printf("\nTurning on two PCIMs\n\n");

/* Initialize the PCIM #1 Parameters */
    local[0].im.Segment = 0xCC00;
    local[0].im.IOPort = 0x3E0;
    local[0].IMRef = 0x1212;
    local[0].OutputLength = 0;
    local[0].InputLength = 0;
    loca[0]l.Active = ON;

/* Initialize the PCIM #2 Parameters */
    local[1].im.Segment = 0xD000;
    local[1].im.IOPort = 0x3E4;
    local[1].IMRef = 0x3434;
    local[1].OutputLength = 0;
    local[1].InputLength = 0;
    local[1].Active = ON;

    if ( (ret = InitIM ( 2, local, flags ) ) != SUCCESS )
    {
          printf("\nInitIM returned %d\ntest exit",ret);
          loop = 0;
     }
    while(loop)
    {
```

```
/*
    From PCIM #1 (which is SBA 31), GetDevConfig(uration) for SBA 30, which
    in this case is PCIM #2.  This can be used for any devices on the bus.
*/
        if ( (ret = GetDevConfig ( 1, 30, &config) ) != SUCCESS)
        {
/*
        returned an error code...probably 7 or 11...look in PCIM.H
        for Error Return MACROS
*/
              printf("\nGetDevConfig returned %d\ntest exit",ret);
              loop = 0;
        }/*

*/


        else
        {
              printf("\n\nFor Serial Bus Address 30");
              printf("\nModel = %2d", config.Model);
              printf("\nOutputs are %s", ((config.OutputDisable) ? "DI
              SABLED" : "ENABLED" ) );
              printf("\nDevice is%spresent", ((config.Present) ? " " :
              " NOT ") );
              printf("\nInput Length = %2d", config.InputLength);
              printf("\nOutput Length = %2d", config.OutputLength);
              printf("\nDevice type is ");
              switch (config.Config)
              (
              case 1:
                   printf("Input Only");
                   break;
              case 2:
                   printf("Output Only");
                   break;
              case 3:
                   printf("Combination");
                   break;
              }
        }
```

```
/*
    From PCIM #2 (which is SBA 30), GetDevConfig(uration) for SBA 31, which
    in this case is PCIM #1.  This can be used for any devices on the bus.
*/
        if ( (ret = GetDevConfig ( 2, 31, &config) ) != SUCCESS)
        {
/*
        returned an error code...probably 7 or 11...look in PCIM.H
        for Error Return MACROS.

*/
             printf("\nGetDevConfig returned %d\ntest exit",ret);
             loop = 0;
        }
        else
        {
             printf("\n\nFor Serial Bus Address 31");
             printf("\nModel = %2d", config.Model);
             printf("\nOutputs are %s", ((config.OutputDisable) ? "DI
             SABLED" : "ENABLED" ) );
             printf("\nDevice is%spresent", ((config.Present) ? " " :
             " NOT ") );
             printf("\nInput Length = %2d", config.InputLength);
             printf("\nOutput Length = %2d", config.OutputLength);
             printf("\nDevice type is ");
             switch (config.Config)
             {
*/
             case 1:
                  printf("Input Only");
                  break;

             case 2:
                  printf("Output Only");
                  break;/*

             case 3:
                  printf("Combination");
                  break;
            }
        }
        printf("\n\nPress return to continue ");
        x = getchar();
        if (x == 'q' || x == 'Q')
             loop = 0;
    }
    printf("\n\nThat is all");
/*
 These next instructions turn the two PCIMs off
*/
    local[0].Active = OFF;
    local[1].Active = OFF;
/*
 These next two function calls may be checked for
 Error Returns
*/
    ChgIMSetup (1, PCIM1 );
    ChgIMSetup (2, PCIM2 );
}
```

Example Application 2

This example provides uses the most common call routines for the PCIM. Each call routine will be provided with a section of C code showing the proper use of the driver.

These call routines have been set up using a discrete block connected to the PCIM in the following configuration:

Serial Bus Address – 1
Reference Address – 65
Point Configuration – Pt 1 Input
Pt 5 Output
Pt 2 Output
Pt 6 Input
Pt 3 Output
Pt 7 Input
Pt 4 Output
Pt 8 Input

Any failures by the call routines will be displayed with the returned failure code.

Time delays are inserted within the program to visually verify the correct operation of the driver where appropriate.

```
*/

#include      <stdio.h>
#include      ”pcim.h”

extern  int
    InitIM(),
    ChgIMSetup(),
    GetIMState(),
    GetBusConfig(),
    GetDevConfig(),
    DisableOut(),
    GetBusIn(),
    PutBusOut(),
    GetDevIn(),
    PutDevOut(),
    GetCir(),
    GetWord(),
    PutCir(),
    PutWord();

/*  Using the PCIM.H library, declare the following variables.
*/
IMPARMS imparm;
IMSTATE imstate;
DEVICE device[32];
DEVICE config;

/*  The following arrays are declared for use as data storage
      in the program.
*/

unsigned char   INdata [4096],
                OUTdata [4096],
                INDdata [128];
```

```
main()
{
        int     ret,
                x = 0,
                y = 0,
                pnum = 1,
                dnum = 1,
                offset = 3;

        char    val,
                valword [1],
                flags;

        unsigned char   lgth,
                        length;

/*      Define the PCIM parameters.  This assignment reflects the hardware
        setup of the PCIM and it is DIP switches.
*/
        imparm.im.Segment = 0xD000;
        imparm.im.IOPort = 0x3E4;
        imparm.IMRef = 0x0000;
        imparm.OutputLength = 0;
        imparm.InputLength = 0;
        imparm.Active = ON;

/*      Use the InitIM driver to initialize the PCIM.
*/
        for ( x=0; x<0xFFF; x++ );
        if ((ret = InitIM ( pnum, &imparm, &flags)) != SUCCESS )
        {
                printf("\nInitIM failure, returned %d\n",ret);
         }
        else
         {
                printf("\nInitIM driver successful\n");
          }
        for ( x=0; x<0xFFFF; x++ )
                for ( y=0; y<0xF; y++);

/*      Use the ChgIMSetup driver to change the IMREF value from
        0 to 0x1212.  Note that all parameters in the imparm array
        are transferred to the PCIM.
*/
        imparm.IMRef = 0x1212;
        if ((ret = ChgIMSetup ( pnum, &imparm )) != SUCCESS )
        {
                printf("\nChgIMSetup failure, returned %d\n",ret);
                printf("\nSegment %x",imparm.im.Segment);
                printf("\nIOPort %x",imparm.im.IOPort);
                printf("\nIMRef %d",imparm.IMRef);
                printf("\nOutLength %d",imparm.OutputLength);
                printf("\nInputLength %d",imparm.InputLength);
                printf("\nActive %d",imparm.Active);
        }
        else

        {
                printf("\nChgIMSetup driver successful\n");
         }

/*      Use the GetIMState driver to read the Status Table and Setup
        Table of the PCIM.  Display the DIP Switch value which is
        returned as part of this call.
```

```
*/
        if (( ret = GetIMState ( pnum, &imstate) ) != SUCCESS )
        {
                printf("\nGetIMState failure, returned %d\n",ret );
        }
        else
        {
                printf("\nGetIMState driver successful\n");
                printf("        DipSwitch value %x\n",imstate.DipSwitch);
        }
/*
        Use the GetBusConfig driver to display the configuration of
        the Genius Bus.  Display a subset of the information returned.
*/
        for ( x=0; x<0xFFFF; x++ )
                for (y=0; y<0xF; y++);

        if (( ret = GetBusConfig ( pnum, device) ) != SUCCESS)
        {
                printf("\nGetBusConfig failure, returned %d\n", ret);
        }
        else
        {
                printf("\nGetBusConfig successful\n");
                printf("        Model # Device 1 = %d", device[1].Model);
                printf("\n      Device Present = %d", device[1].Present);
                printf("\n      Device Configuration = %x\n", device[1].Config);
        )
/*      Use the GetDevConfig driver to display the configuration of a
        specific block.  Display the reference address of the block.
*/
        if (( ret = GetDevConfig ( pnum, dnum, &config) ) != SUCCESS )
        {
                printf("\nGetDevConfig failure, returned %d\n");
        }
        else
        {
                printf("\nGetDevConfig successful\n");
                printf("     Device Present = %d",config.Present);
                printf("\n   Device reference address = %d\n",config.Reference);
        }

/*      Use the PutCir driver to turn on pt 3 of the Genius I/O block.
*/

        for ( x=0 ; x<0xFFFF; x++ )
                for ( y=0; y<0xF; y++);
        if (( ret = PutCir ( pnum, dnum, offset, (char) 1 )) != SUCCESS )

        {
                printf("\nPutCir failure, returned %d\n", ret );
        }
        else
        {
                printf("\nPutCir driver successful.  Pt 3 should be ON.\n");
        }
        for ( x=0 ; x<0xFFFF; x++ )
                for (y=0; y<0xF; y++);

/*      Use the DisableOut driver to disable the updating of the block
        thus turning pt 3 off.
*/
        if (( ret = DisableOut ( pnum, dnum, DISABLE) ) != SUCCESS)
```

```
        {
                printf("\nDisableOut failure, returned %d\n", ret);
        }
        else
        {
                printf("\nDisableOut driver successful – Outputs shd be off\n");
       }
        for ( x=0 ; x<0xFFFF; x++)
                for (y=0; y<0xF; y++);

        DisableOut (pnum,dnum,ENABLE);

/*      Use the GetBusIn driver to read all input data on the PCIM bus.
        Display input data for device 1.


*/
        if (( ret = GetBusIn (pnum, INdata) ) != SUCCESS )
        {
                printf("\nGetBusIn failure, returned %d\n", ret);
        }
        else
        {
                printf("\nGetBusIn successful");
                printf("\n      Input data = %X\n",INdata);
        )
/*      Use the PutBusOut driver to write output data to the discrete
        block.  Turn on pt 3,4,5 .
*/
        OUTdata[128] = 0x1C;

        if (( ret = PutBusOut ( pnum, OUTdata ) ) != SUCCESS )
        {
                printf("\nPutBusOut failure, returned %d\n", ret);

        }
        else
        {
                printf("\nPutBusOut successful");
                printf("\n      Output data[128] = %X\n",OUTdata);
                printf("        Pt 3, 4, and 5 should be ON\n");
        }
        for ( x=0; x<0xFFFF; x++ )
                for ( y=0; y<0xF; y++);

/*      Use the GetDevIn driver to read input data from the discrete
        block.  Value should indicate 0x1C.
*/
        if (( ret = GetDevIn ( pnum, dnum, &length, INDdata )) != SUCCESS )
        {
                printf("\nGetDevIn failure, returned %d\n", ret );
        }
        else
        {
                printf("\nGetDevIn successful");
                printf("\n      Discrete Block Input Data = %X\n",INDdata[0]);
       }

/*      Use the PutDevOut driver to turn on pt 3 and 5 on the discrete
        block.
*/
        lgth=1;
```

```
        OUTdata[0]=0x14;

        if (( ret = PutDevOut ( pnum, dnum, lgth, OUTdata )) != SUCCESS )
        {
                printf("\nPutDevOut failure, returned %d\n", ret);
        }
        else
        {
                printf("\nPutDevOut successful");
                printf("\n      Pt 3 and Pt 5 should be ON\n");
        }

/*      Use the GetCir and GetWord drivers to read the input status of
        the discrete block.
*/
        offset = 3;

        if (( ret = GetCir ( pnum, dnum, offset, &val)) != SUCCESS )
        {
                printf("\nGetCir failure, returned %d\n", ret);
        }
        else
        {
                printf("\nGetCir successful");
                printf("\n      Value read should be 1, val= %x\n", val );
        {
        offset = 1;

        if (( ret = GetWord ( pnum, dnum, offset, valword)) != SUCCESS )
        {
                printf("\nGetWord failure, returned %d\n", ret);
        }
        else
        {
                printf("\nGetWord successful");
                printf("\n    Value read should be x14, val= %x\n", valword[0]);
        }
        for ( x=0; x<0xFFFF; x++)
                for ( y=0; y<0xF; y++);

/*      Use the PutWord driver to turn on pt 4 on the discrete block.
*/
        offset = 1;
        valword[1] = 0x08;


        if (( ret = PutWord ( pnum, dnum, offset, valword[1] )) != SUCCESS )
        {
                printf("\nPutWord failure, returned %d\n", ret);
        }
        else
        {
                printf("\nPutWord successful");
                printf("\n      Pt 4 should be ON");
        }

        for ( x=0; x<0xFFFF; x++)
                for ( y=0; y<0xF; y++);

/*      Exit the program by turning off the module.
*/
        imparm.Active = OFF;
        ChgIMSetup (pnum, &imparm);
)
```

Example Application 3

This example shows in BASIC the way the SENDMSG (or SENDMSGREPLY) and CHKMSGSTATUS message functions must be used together. The comments in the text provide a running commentary for the use of each driver.

```
010     CALL SENDMSG (or SENDMSGREPLY) (STATUS, IMNUM, MSG(0))
2020    IF STATUS = 12 THEN 2050 ;IF PCIM is busy go to 2050
2030    IF STATUS <> 0 THEN 2170 ;IF STATUS is anything other then ”0”;
2040        ;something is wrong – go to 2170
2050    GO TO 2110     ;SENDMSG was executed O.K.; go to 9110 to
2060        ;check msg status
2070    CALL CHKMSGSTAT (STATUS,IMNUM,MSGSTATUS)
2080    IF STATUS <> 0 THEN 2170 ;If STATUS is anything other then ”0”;
2090        ;something is wrong – go to 2170
2100    If MSGSTATUS = 12 THEN 2050   ;If PCIM busy, stay in this loop and go
2110        ;back to 2050
2120    IF MSGSTATUS = 16 THEN 2010   ;If PCIM is free; go back to 2010 and
2130        ;execute SENDMSG
2140    IF MSGSTATUS <> 0 THEN 2170   ;If MSGSTATUS is anything else; go to 2170
2150        ;and decode
2160    ,
2170    CALL CHKMSGSTAT (STATUS, IMNUM, MSGSTATUS) ;Did SENDMSG get
         ;on the bus
2180    IF STATUS <> 0 THEN 2170 ;If STATUS is anything other than ”0”; go to
2190        ;2170 and decode
2200    IF MSGSTATUS = 12 THEN 2110   ;PCIM is busy; stay in this loop and go
2210        ;back to 2110
2220    If MSGSTATUS <> 0 THEN 2170   ;If MSGSTATUS is anything other than ”0”;
2230        ;go to 2170 and decode
2240    RETURN    ;The SENDMSG call was executed properly; If
2250        ;SENDMSGREPLY the reply msg is ready to
2260        ;read with GETMSG
2270    CLS  ;Clear Screen
2280    PRINT STATUS, MSGSTATUS  ;Interpret the code for STATUS and/or
2290        ;message status
```

A

Address selection, 2-2 , 3-10
  DIP switches, 3-2
  in software, 3-8

Auxiliary Request Queue, 2-5

B

BASIC programming, 5-1

Baud Rate, 3-12 , 7-3

Broadcast Control Output Table, 2-5

Bus
  configuration, reading, 4-2 , 4-22 , 5-2 , 5-20
  connection to, 1-2 , 1-3 , 3-3
  errors, 7-3
  overview, 1-4
  termination, 3-3 , 3-4

Bus controller, 1-4

C

C programming, 4-1

Catalog numbers, 1-5

Changing the setup parameters, 4-2 , 4-18 , 5-2 , 5-16

ChgIMSetup, 4-2 , 4-18 , 5-2 , 5-16

ChkMsgStat, 4-3 , 4-50 , 5-3 , 5-48

Clear Reset Request, 2-4

COMM OK status, 2-3

Command Block, 2-5

Compiling a C program, 4-1

Computer, compatible types, 1-2

Configuration, example, 3-13

Configuration choices, 3-10

Configuration software
  overview, 1-6
  running, 3-8

Connectors, 1-3

Control bits, 2-2 , 2-4

D

Datagrams, 6-3 , 7-3

Daughterboard, 1-2 , 1-3

Device Configuration Table, 2-5

Device configuration, reading, 4-2 , 4-24 , 5-2 , 5-22

Device I/O Table, 2-5

Device number, 3-10

DIP switch settings, 3-2

Directed Control Input Table, 2-5

DisableOut, 4-2 , 4-26 , 5-2 , 5-24

DOS interrupt, 3-10

DOS requirements, 1-6 , 3-1

Drive capability, 2-1

E

Electrical characteristics, 2-1

Electrical specifications, 1-5

Environmental specifications, 1-5

Error status, 5-10

F

File linkage for C program, 4-1

Functions, software, 4-2 , 5-2

G

GetBusConfig, 4-2 , 4-22 , 5-2 , 5-20

GetBusIn, 4-2 , 4-28 , 5-2 , 5-26

GetCir, 4-2 , 4-38 , 5-2 , 5-36

GetDevConfig, 4-2 , 4-24 , 5-2 , 5-22

GetDevIn, 4-2 , 4-32 , 5-2 , 5-30

GetIMIn, 4-36 , 5-34

GetIMState, 4-2 , 4-20 , 5-2 , 5-18

GetINTR, 4-3 , 4-54 , 5-3 , 5-52

GetMsg, 4-3 , 4-52 , 5-3 , 5-50

GetWord, 4-2 , 4-42 , 5-2 , 5-40

Global Data, 6-1 , 7-3
  sending, 4-37 , 5-35

H

Hand-held Monitor, 1-4

Hand-held Monitor connector, 3-5

Heartbeat Enable, 2-5

Heartbeat Timeout Multiplier, 2-5

HHM Test bit, 2-4

Host Clear, 2-5

I

I/O blocks, 1-4

I/OTable Lockout, 2-5

Initialization function, 4-2 , 4-15 , 5-2 , 5-13

InitIM, 4-15 , 5-13

Inputs, read, 4-2 , 4-28 , 4-32 , 4-38 , 4-42 , 5-2 , 5-26 , 5-30 , 5-36 , 5-40

Installation, 3-2

Interrupt Disable Table, 2-5

Interrupt request, 2-3

Interrupt Status Table, 2-5

Interrupts, 3-11 , 4-3 , 4-54 , 5-3 , 5-52
  disable, 4-56 , 5-54

L

LEDs, 1-3 , 1-5 , 7-2

Low voltage detection, 2-3

M

Mechanical specifications, 1-5

Memory requirements, 1-5

Message, reading, 4-52 , 5-50

Message status, 4-50 , 5-48

Message, reading, 4-3 , 5-3

Motherboard, 1-3

MS DOS requirements, 3-1

O

Operating system, 1-6

Outputs
  disable, 3-10 , 4-2 , 4-26 , 5-2 , 5-24
  writing, 4-2 , 4-30 , 4-34 , 4-40 , 4-44 , 5-2 , 5-28 , 5-32 , 5-38 , 5-42

P

Parameters, BASIC, 5-4

Parameters, C, 4-4

Parity errors, 7-3

PCIM
  catalog numbers, 1-5
  control data, 2-2 , 2-4
  daughterboard, 1-2 , 1-3
  electrical characteristics, 2-1
  host memory required, 1-5
  motherboard, 1-3
  size and appearance, 1-1
  software driver, 1-1 , 1-6
  specifications, 1-5
  status data, 2-2 , 2-3

PCIM OK status, 2-3

PCIM Reset command bit, 2-4

PCIM software driver, 4-1

PCIM.H file, 4-4

Power supply requirements, 2-1

PutBusOut, 4-2 , 4-30 , 5-2 , 5-28

PutCir, 4-2 , 4-40 , 5-2 , 5-38

PutDevOut, 4-2 , 4-34 , 5-2 , 5-32

PutIMOut, 4-37 , 5-35

PutINTR, 4-3 , 4-56 , 5-3 , 5-54

PutWord, 4-2 , 4-44 , 5-2 , 5-42

R

Read Datagram Buffer, 2-5

Reading configuration and status data, 4-2 , 4-20 , 5-2 , 5-18

Reading inputs, 4-2 , 4-28 , 4-32 , 4-38 , 4-42 , 5-2 , 5-26 , 5-30 , 5-36 , 5-40

Reading the bus configuration, 4-2 , 4-22 , 5-2 , 5-20

Reset detection, 2-3

Reset Request, clear, 2-4

S

Sending message on the bus, 4-3 , 4-46 , 4-48 , 5-3 , 5-44 , 5-46

SendMsg, 4-3 , 4-46 , 5-3 , 5-44

SendMsgReply, 4-3 , 4-48 , 5-3 , 5-46

Serial bus address, 3-10

Service, telephone number, 7-1

Setup Table, 2-5

Shared RAM, 2-5

Signal conditioning, 2-1

Software driver, 1-6 , 4-1
- files, 3-1
- function calls, 4-2 , 5-2
- installation, 5-1
- overview, 1-1

Startup, 3-7

Status bits, 2-2 , 2-3

Status Table, 2-5

System overview, 1-4

T

Transmit Datagram Buffer, 2-5

W

Watchdog timer
- enable, 3-12
- pulse, 2-4
- status, 2-3

Wiring specifications, 1-5

Writing outputs, 4-2 , 4-34 , 4-40 , 4-44 , 5-2 , 5-32 , 5-38 , 5-42