GE
Intelligent Platforms

Programmable Control Products

PCI GENIUS Card OEM Windows Drivers*

User's Manual

GFK-2343 *April 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.

* indicates a trademark of GE Intelligent Platforms, Inc. and/or its affiliates. All other trademarks are the property of their respective owners.

©Copyright 2010 GE Intelligent Platforms, Inc.
All Rights Reserved

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)

Contents

Chapter 1 Introduction

This manual describes installation and usage of the PCI Genius OEM Windows Drivers. This manual has the following layout:

Chapter 1: Introduction

Chapter 2: Installation procedures

Chapter 3: Programming API

Chapter 4: Examples

Chapter 5: Troubleshooting

The PCI Genius card provides an intelligent interface between a PCI slot in a PC compatible computer and a single channel general purpose controller interface to the Genius serial bus. The primary function of this card is to provide a means for third parties to interface their CPU, PLC, or Genius I/O Blocks on the Genius bus.

The PCI Genius OEM Windows® Drivers serve as a pass-through service to allow a user-mode program to access the three Functional Registers and the Shared RAM region. On the PCI Genius card, they do not perform any higher-level functions. The interface uses the standard Windows API calls, such as CreateFile, CloseHandle, GetLastError, and DeviceIoControl, so it is not limited to any particular programming language or version.

The OEM Windows drivers support only the PCI Genius card (IC660ELB931). They do not support ISA version cards. They run on Microsoft Windows 2000, XP, and NT4.0. Windows 95, 98, and ME are not supported.

Other Genius Manuals

PCI Genius Card Quick Install Guide, GFK-2342	Describes the hardware of the IC660ELB931 PCI Genius card and the Functional Registers.
Genius I/O µGENI Board User's Manual, GFK-0845	Describes the organization and operation of the Genius Shared RAM interface.
Genius I/O System and Communications User's Guide, GEK-90486F-1	Describes the high-level concepts of Genius communications and bus design.

1

Genius PC Interface Products Background

In working with the PCI Genius card, it is helpful to understand some of the history of the Genius PC interfaces. The core of the Genius PC interface products is a shared memory module called a GENI or µGENI, which is described in GFK-0845A. This has a 16Kbyte memory mapped interface which allows a host processor to communicate on the Genius network. The other input and output signals to a µGENI module are the four Genius bus lines, two LED signals, and eight initialization signals that specify the card's SBA, network baud rate, and output enabled status.

Previous ISA PC Genius cards either implemented the µGENI circuitry as a single circuit board or had a carrier ISA circuit board that allowed one or two µGENI daughter modules to attach to the carrier ISA circuit board. The eight initialization signal values and the address information as to where the µGENI mapped into the ISA space were programmed into non-volatile memory on the carrier board. This configuration information resided at one range of I/O registers and control of the reset and watchdog lines going into the µGENI hardware were controlled at a second set of I/O registers.

The PCI Genius card continues to use this format. The µGENI module has been integrated onto the single circuit board and the setup signals and control of the reset and watchdog functions have been mapped to I/O registers. There is no non-volatile memory on the PCI card, so the card does not retain its SBA address, bus baud rate, or output enable state in non-volatile storage.

Upgrading from ISA PC Genius Cards

OEM users of the ISA PC Genius cards should note a few significant differences in the PCI version.

1. The SBA and network settings are not stored in the card. There are no DIP switches or non-volatile memory on the PCI Genius card to maintain the Genius settings of the card. These settings must be maintained by the application program and set into the PCI card Functional Registers on each startup.
2. The hardware location of the card in I/O and memory space are defined at startup by PCI enumeration. The concept of setting the card at I/O address 0x3E0 and host memory address 0xD0000 is not the ideal PCI implementation. When using these OEM drivers, Windows will place the card in a valid available address. The Functional Registers, similar to what was available at the I/O addresses of an ISA card, are available through the Read and Write Register commands of the driver. The base address, equivalent to the host memory address, is obtained through the Map memory call of the driver. For details, see “Map Shared RAM” in chapter 3.
3. The definition of the Shared memory interface is unchanged from ISA versions. If an application was programmed to be based off of a specific host memory address such as 0xC8000 or 0xD0000. All that is required is to obtain that new host memory address from the driver after the interface has been initialized in the application.
4. The definitions of Functional Registers (PCIM status and PCIM control on the ISA cards) have changed very slightly and a new Configuration register has been added which contains information that was present in non-volatile storage. ISA users must add code to setup the Configuration register and review the Status and Control registers in their application.
5. The TestGENI and dpcimcfg utilities are no longer needed. These utilities provided the ability to program the nonvolatile memory on ISA cards and to test their connectivity on the Genius network. There is no non-volatile memory on the PCI Genius card, and the control panel applets support enabling the card and verifying its general operation.
6. The physical orientation of the connector has changed. It is the same as newer Genius hardware such as the VersaMax Genius NIU.
7. The PCI Genius card is available only in a single channel version.

Chapter 2 Installation

This chapter provides instructions for installing the drivers. It is divided into two sections:

These drivers support only the PCI Genius card (IC660ELB931). They do not support other Genius ISA cards.

Section 1 – Windows NT 4.0

Discusses installation on Windows NT 4.0 machines using the provided batch files.

Section 2 – Windows 2000 and Windows XP

Discusses installation on Windows 2000 and XP machines.

Section 1 – Windows NT 4.0

Windows NT does not support Plug and Play (PnP) PCI, so the driver must be installed as a service.

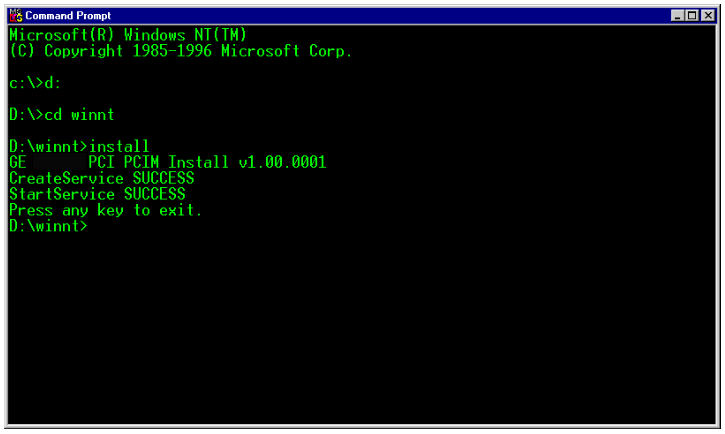
1. Install the PCI Genius card hardware (see the PCI Genius Card Quick Install Guide, GFK-2342).
2. Power up the computer and log on to Windows.

Note: The login user must have administrator privileges on the machine.

3. Open a command window (DOS box) and change the directory to the compact disk drive with the driver files. (For example, type e: if the compact disk drive is the e: drive.)
4. Change to the winnt directory of the compact disk. (For example, type cd e:\winnt.) (Note that this is not the Windows system directory.)
5. Run the install program by typing install.
6. The install program will copy the files gegenius.sys and gegenius.cpl into the Windows directory and start the driver service. It will display:

   CreateService SUCCESS
   StartService SUCCESS
   Press any key to exit.

7. Press any key to return to the command prompt. The driver is now installed.



```
Microsoft(R) Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.

c:\>d:

D:\>cd winnt

D:\winnt>install
GE       PCI PCIM Install v1.00.0001
CreateService SUCCESS
StartService SUCCESS
Press any key to exit.
D:\winnt>
```

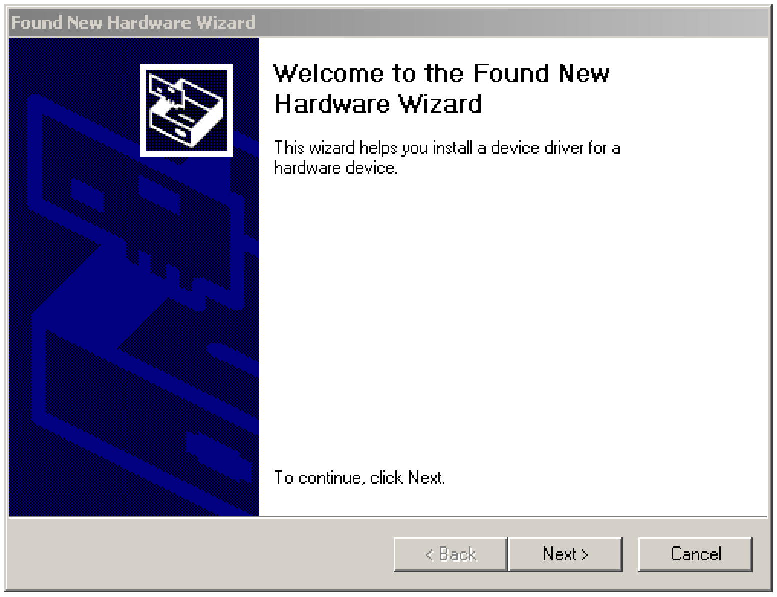
8. To verify the hardware is operational, open the Windows Control Panel and click the GE Genius control panel. The control panel displays the name of each PCI card (up to four). The Details button opens a dialog box that can turn on the card and show the LED status as the card is powered on.

Section 2 – Windows 2000 and Windows XP

1. Install the PCI Genius card hardware (see the PCI Genius® Card Quick Install Guide, GFK-2342).
2. Power up the computer and log in to Windows.

Note: The login user must have administrator privileges on the machine.

3. Windows should detect the new device and display the Found New Hardware Wizard. Click Next.

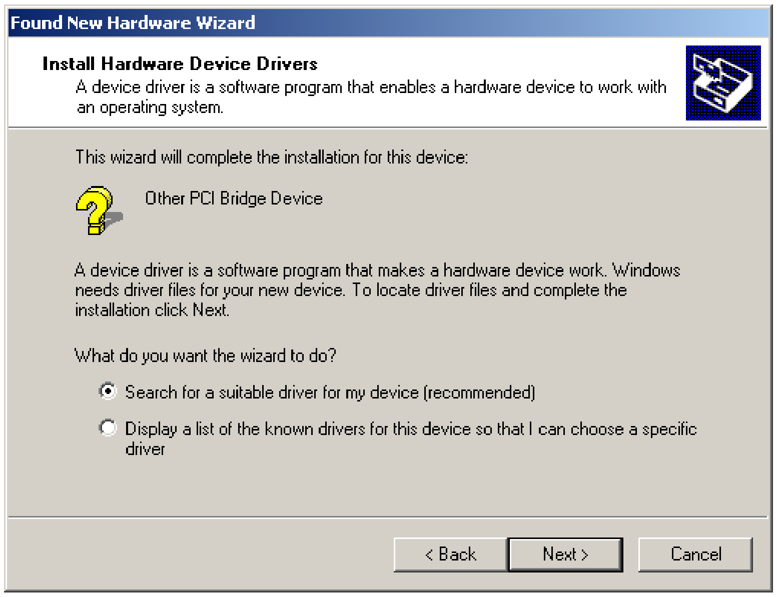


Welcome to the Found New Hardware Wizard

This wizard helps you install a device driver for a hardware device.

To continue, click Next.

4. Windows then displays the wizard to search for new drivers. Select "Search for a suitable driver for my device". Click Next.



Found New Hardware Wizard

Install Hardware Device Drivers
A device driver is a software program that enables a hardware device to work with an operating system.

This wizard will complete the installation for this device:

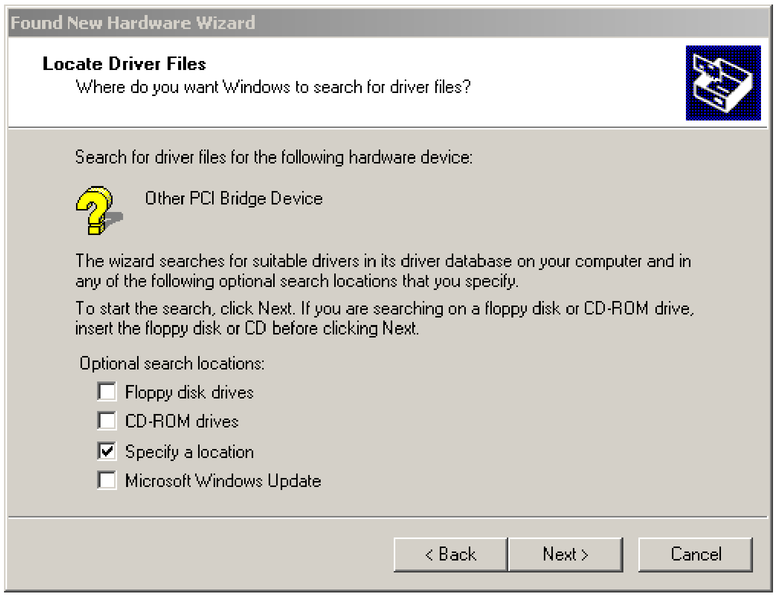
Other PCI Bridge Device

A device driver is a software program that makes a hardware device work. Windows needs driver files for your new device. To locate driver files and complete the installation click Next.

What do you want the wizard to do?

- [✓ Search for a suitable driver for my device (recommended)]
- Display a list of the known drivers for this device so that I can choose a specific driver

5. Windows then asks where to search. Select "Specify a location" and click Next.



Found New Hardware Wizard

Locate Driver Files
Where do you want Windows to search for driver files?

Search for driver files for the following hardware device:

Other PCI Bridge Device

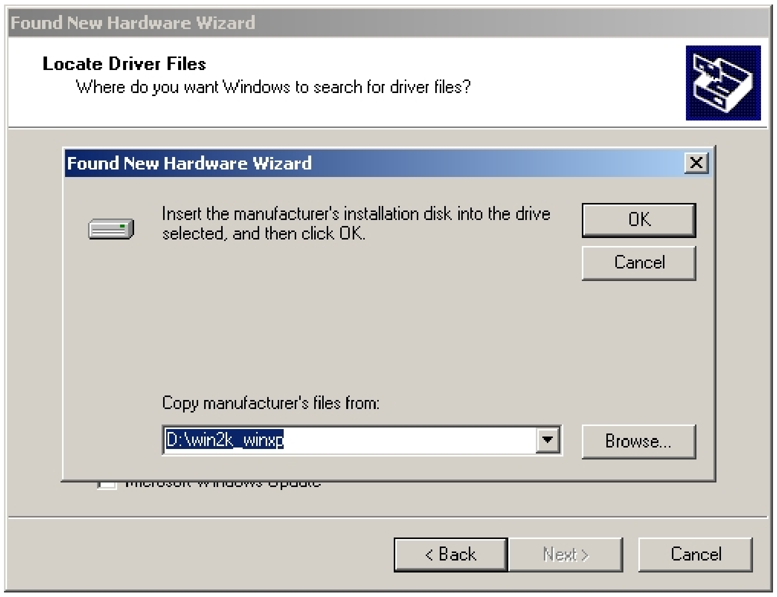
The wizard searches for suitable drivers in its driver database on your computer and in any of the following optional search locations that you specify.

To start the search, click Next. If you are searching on a floppy disk or CD-ROM drive, insert the floppy disk or CD before clicking Next.

Optional search locations:

- Floppy disk drives
- CD-ROM drives
- [✓ Specify a location]
- Microsoft Windows Update

6. Enter the location of the gegenius.inf file from the compact disk and click OK.



Found New Hardware Wizard

Locate Driver Files

Where do you want Windows to search for driver files?

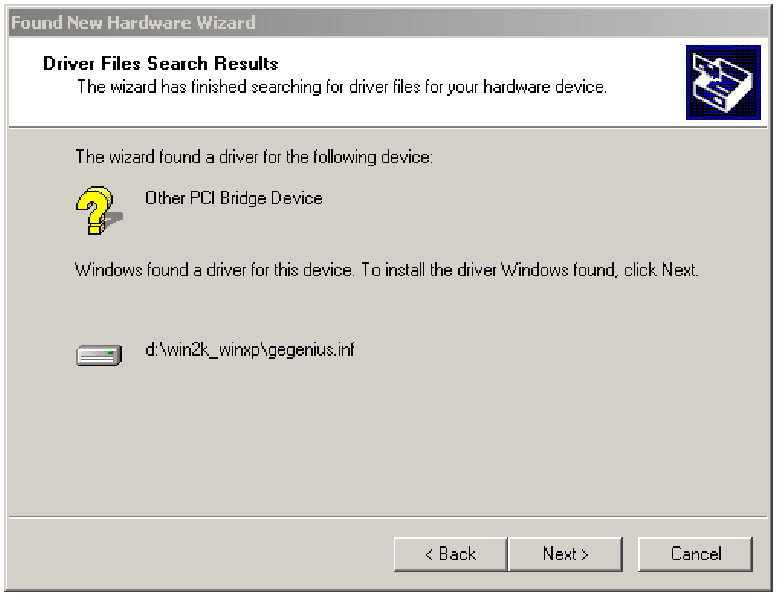
Found New Hardware Wizard

Insert the manufacturer's installation disk into the drive selected, and then click OK.

Copy manufacturer's files from:

D:\win2k_winxp

7. When Windows verifies the driver matches the hardware, it shows the results dialog. Click Next.



Found New Hardware Wizard

Driver Files Search Results

The wizard has finished searching for driver files for your hardware device.

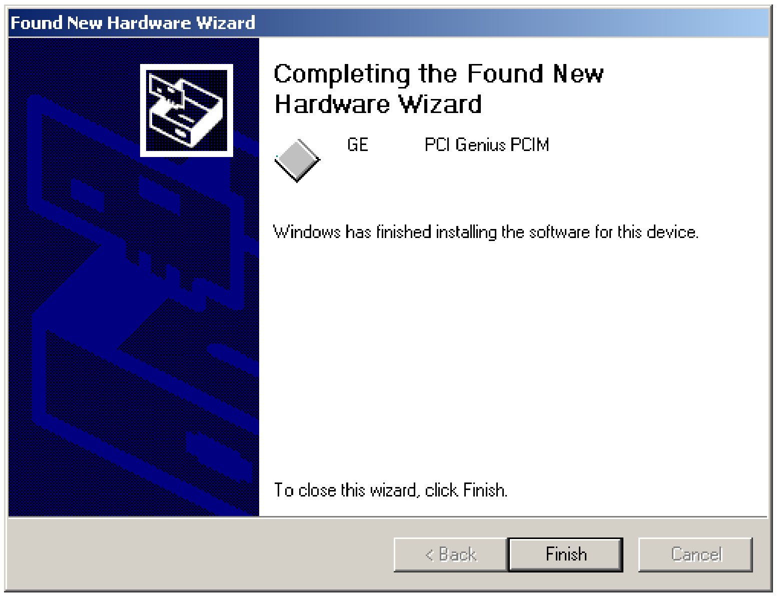
The wizard found a driver for the following device:

Other PCI Bridge Device

Windows found a driver for this device. To install the driver Windows found, click Next.

d:\win2k_winxp\gegenius.inf

8. Windows executes the install script and shows the following dialog when it is done. Click Finish.


Found New Hardware Wizard

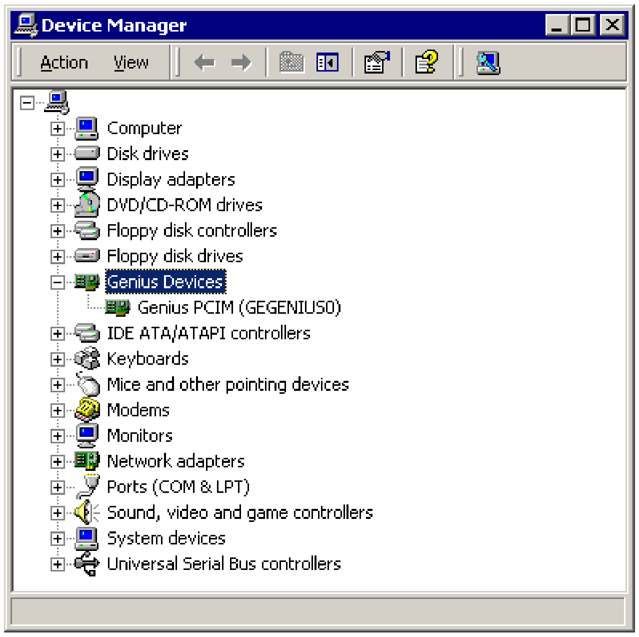
Completing the Found New Hardware Wizard

GE PCI Genius PCIM

Windows has finished installing the software for this device.

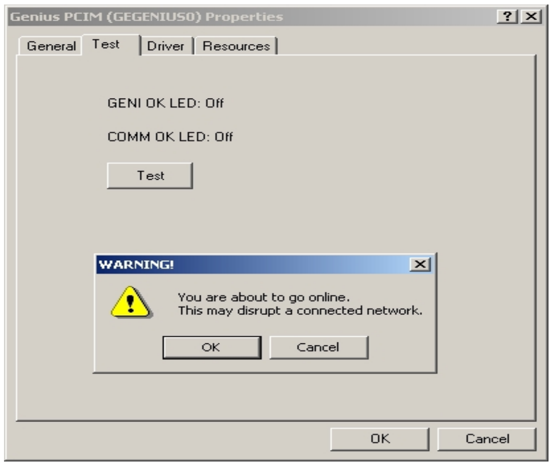
To close this wizard, click Finish.

9. The card will be in the Device Manager under Genius Devices.


Device Manager

- Computer
- Disk drives
- Display adapters
- DVD/CD-ROM drives
- Floppy disk controllers
- Floppy disk drives
- Genius Devices
  - Genius PCIM (GEGENIUS0)
- IDE ATA/ATAPI controllers
- Keyboards
- Mice and other pointing devices
- Modems
- Monitors
- Network adapters
- Ports (COM & LPT)
- Sound, video and game controllers
- System devices
- Universal Serial Bus controllers

10. The properties of the card in the Device Manager has four tabs. Three are standard Windows device tabs: General, Driver, and Resources. The fourth tab, Test, allows the user to turn on the card as SBA 0 at 153.6Kbaud Std to verify the card is working and is in the physical slot expected. The user is prompted before going online since a connected GENIUS bus may be disrupted by the fixed SBA and baud rate.


Genius PCIM (GEGENIUS0) Properties

GENI OK LED: Off

COMM OK LED: Off

WARNING!

You are about to go online.
This may disrupt a connected network.

Chapter 3 Programming

This chapter details the calls exported from the driver. It explains each call and its parameters and return values. Users are expected to be familiar with Win32 API programming and to have access to Win32 API documentation such as Microsoft Developer's Network (MSDN).

Note that the driver serves as a pass-through service to allow a user-mode program to gain access to the three Functional Registers and the Shared RAM region. It does not perform any higher-level functions.

This chapter describes:

- **Opening and Closing the device**

  Returns and frees a Window's handle to the card device.
- **Reading the card's physical location**

  Returns the bus and slot of the card.
- **Reading and writing Functional Registers**

  Allows access to the three Functional Registers.
- **Mapping and Unmapping Shared RAM**

  Allows User-mode access to the Shared RAM interface common to the GENIUS® PC interface cards.

Opening the Device

The driver exposes the PCI GENIUS card as a device using the name GEGENIUS*x* where *x* is the card number from zero to three indicating the specific card. Users can connect to the driver by calling the Windows API function CreateFile() as follows:

```
HANDLE hCardDevice;

hCardDevice = CreateFile("\\\\.\\GEGENIUS0", GENERIC_READ |
GENERIC_WRITE, 0, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
```

The characters "\\.\" identify the name as a device name. Double slashes are required for each slash to prevent interpretation as an escape sequence.

Return Values:

The return values from CreateFile() are specified in the Win32 documentation. In general, CreateFile() returns either a handle to the device or:

- INVALID_HANDLE_VALUE:
  GetLastError() will return a system error code specified by the Win32 SDK. The most likely error returned from GetLastError() would be ERROR_FILE_NOT_FOUND when the card is not present.

Closing the Device

Handles opened with CreateFile should be closed after any resources mapped (see Map Shared RAM below) have been released. This is done with the Win32 CloseHandle() function.

```
CloseHandle(hCardDevice);
```

Return Values:

CloseHandle does not return any values.

Device I/O Request Packets (IRPs)

The driver supports five I/O requests. Each of these requests is accomplished by calling the Windows API function DeviceIoControl() using the HANDLE opened with CreateFile().

Read Card Location

Returns the physical PCI bus and slot of the card represent by the current handle. This is useful to determine that a card has not been enumerated to a different logical address during PCI enumeration. The order of enumeration of cards in Windows is not guaranteed, therefore multiple cards could move in their Genius*x* assignment. In practice, cards are enumerated by the electrical organization of the PC. If you have multiple cards installed in a PC, this will return the physical location of the card assigned to the GENIUS*x* device used in the CreateFile() call. Users with a single PCI GENIUS card do not have a significant reason to ever need this function.

Definition:

```
#define IOCTL_GEGENIUS_READ_LOC \
CTL_CODE(FILE_DEVICE_UNKNOWN, 0x900, METHOD_BUFFERED, FILE_READ_ACCESS)
```

Or numerically:

```
#define IOCTL_GEGENIUS_READ_LOC 0x00226400
```

Input Parameter: None.

Output Parameter: 16-bit word.   Upper byte - bus number.
                                  Lower byte - slot number.

Return Value: DeviceIoControl returns zero if there is a failure. GetLastError() will then return a Win32 system error code. Most likely errors are:

- hCardDevice handle invalid. Either not opened or corrupted.
- Output parameter not valid or too small.
- The bytes_returned parameter not given. DeviceIoControl() requires a 32-bit variable address for the number of bytes it returns even if not used by the program code. NULL is not allowed.

Example:

```
USHORT address;
ULONG ulret;
DeviceIoControl(hCardDevice, IOCTL_GEGENIUS_READ_LOC, NULL, 0, &address,
sizeof(USHORT), &ulret, NULL);
```

Read Register

Reads the control, configuration, or status register.

Definition:

```
#define IOCTL_GEGENIUS_READ_PORT \
CTL_CODE(FILE_DEVICE_UNKNOWN, 0x910, METHOD_BUFFERED, FILE_READ_ACCESS)
```

Or Numerically:

```
#define IOCTL_GEGENIUS_READ_PORT 0x00226440
```

Input Parameter: The input is a 32-bit register number:

0: Configuration Register
1: Status Register
2: Control Register

Output Parameter: 8-bit value of the requested register.

Return Value: DeviceIoControl returns zero if there is a failure. GetLastError() will then return a Win32 system error code. Most likely errors are:

- hCardDevice handle invalid. Either not opened or corrupted.
- Input parameter not a 32-bit value.
- Input parameter not 0, 1, or 2.
- Output parameter invalid or less than one byte in size.
- Bytes_return parameter NULL.

Example:

```
ULONG ulret;
ULONG RegisterNumber;
UCHAR RegisterValue;

RegisterNumber = 0; // Configuration Register

DeviceIoControl(hCardDevice, IOCTL_GEGENIUS_READ_PORT, &RegisterNumber,
sizeof(RegisterNumber), &RegisterValue, sizeof(RegisterValue), &ulret,
NULL);
```

Write Register

Writes the control or configuration register.

Definition:

```
#define IOCTL_GEGENIUS_WRITE_PORT \
CTL_CODE(FILE_DEVICE_UNKNOWN, 0x911, METHOD_BUFFERED, FILE_WRITE_ACCESS)
```

Or Numerically:

```
#define IOCTL_GEGENIUS_WRITE_PORT 0x0022A444
```

Input Parameter: 32-bit register number followed by 8-bit value.

Register numbers:	0: Configuration Register,
	1: Status Register,
	2: Control Register

Output Parameter: None.

Return Value: DeviceIoControl returns zero if there is a failure. GetLastError() will then return a Win32 system error code. Most likely errors are:

- hCardDevice handle invalid. Either not opened or corrupted.
- Input parameter not at least 5 bytes.
- Input parameter not 0, 1, or 2.
- Bytes_return parameter NULL.

Example:

```
typedef struct _GENPORT_WRITE_INPUT
{
  ULONG   PortNumber;      // Port # to write to
  union
  {         // Data to be output to port
     ULONG LongData;
     USHORT ShortData;
     UCHAR CharData;
  };
} GENPORT_WRITE_INPUT;

ULONG ulret;
GENPORT_WRITE_INPUT sWriteData;

sWriteData.PortNumber = 0; // Configuration Register
sWriteData.CharData = 0x7F; // Set Baud Rate, SBA, and enable outputs

DeviceIoControl(hCardDevice, IOCTL_GEGENIUS_WRITE_PORT, &sWriteData,
sizeof(sWriteData), NULL, 0, &ulret, NULL);
```

Map Shared RAM

Requests a pointer to the start of Shared RAM on the card. This should only be called once per card for a process. Only one process should be accessing the Shared RAM of a card at any given time.

Definition:

```
#define IOCTL_GEGENIUS_MAP_MEMORY \
CTL_CODE(FILE_DEVICE_UNKNOWN, 0x912, METHOD_BUFFERED, FILE_ANY_ACCESS)
```

Or Numerically:

```
#define IOCTL_GEGENIUS_MAP_MEMORY 0x00222448
```

Input Parameter: None.

Output Parameter: 32-bit pointer to first address of Shared RAM mapped to calling process.

Return Value: DeviceIoControl returns zero if there is a failure. GetLastError() will then return a Win32 system error code. Most likely errors are:

- hCardDevice handle invalid. Either not opened or corrupted.
- Output Parameter not exactly 32-bits.
- Bytes_return parameter NULL.

Example:

```
ULONG dwret;
PVOID pSharedRAM;

DeviceIoControl(hCardDevice, IOCTL_GEGENIUS_MAP_MEMORY, NULL, 0,
&pSharedRAM, sizeof(pSharedRAM), &dwret, NULL);
```

UnMap Shared RAM

Frees memory mapping of Shared RAM.

<u>Definition:</u>

```
#define IOCTL_GEGENIUS_UNMAP_MEMORY \
CTL_CODE(FILE_DEVICE_UNKNOWN, 0x913, METHOD_BUFFERED, FILE_ANY_ACCESS)
```

Or Numerically:

```
#define IOCTL_GEGENIUS_UNMAP_MEMORY 0x0022244C
```

<u>Input Parameter:</u> 32-bit pointer to first address of Shared RAM mapped to calling process returned from Map Shared RAM call.

<u>Output Parameter:</u> None.

<u>Return Value:</u> DeviceIoControl returns zero if there is a failure. GetLastError() will then return a Win32 system error code. Most likely errors are:

- hCardDevice handle invalid. Either not opened or corrupted.
- Input Parameter not exactly 32-bits.
- Bytes_return parameter NULL.

<u>Example:</u>

```
ULONG ulret;
PVOID pSharedRAM;

{ IOCTL_GEGENIUS_MAP_MEMORY call }

DeviceIoControl(hCardDevice, IOCTL_GEGENIUS_UNMAP_MEMORY, &pSharedRAM,
sizeof(pSharedRAM), NULL, 0, &ulret, NULL);
```

Chapter 4 Examples

This section provides a few programming examples for using the PCI GENIUS® card drivers. These examples are console applications for simplicity and are coded for Microsoft Visual C++, but the concepts are valid for any Win32 API programming (which includes VisualBasic).

- Datagrams
- Global Data Read
- Global Data Write
- Reading Fault Information

Datagrams

Below is a simple example of how one might use Genius datagrams to read the number of configured data points in a VersaMax Genius NIU on the network at SBA5. For more documentation about datagrams see Chapter 10 of the *Genius I/O μGENI Board User's Manual,* GFK-0845 and for more details about the VersaMmax Genius NIU see GFK-1535.

```
#define CONFIG_REG  0
#define STATUS_REG  1
#define CONTROL_REG 2

#define IOCTL_GEGENIUS_READ_PORT \
CTL_CODE(FILE_DEVICE_UNKNOWN, 0x910, METHOD_BUFFERED, FILE_READ_ACCESS)
#define IOCTL_GEGENIUS_WRITE_PORT \
CTL_CODE(FILE_DEVICE_UNKNOWN, 0x911, METHOD_BUFFERED, FILE_WRITE_ACCESS)
#define IOCTL_GEGENIUS_MAP_MEMORY \
CTL_CODE(FILE_DEVICE_UNKNOWN, 0x912, METHOD_BUFFERED, FILE_ANY_ACCESS)
#define IOCTL_GEGENIUS_UNMAP_MEMORY \
CTL_CODE(FILE_DEVICE_UNKNOWN, 0x913, METHOD_BUFFERED, FILE_ANY_ACCESS)

typedef struct _GENPORT_WRITE_INPUT
{
  ULONG   PortNumber;             // Port # to write to
  union {                         // Data to be output to port
     ULONG LongData;
     USHORT ShortData;
     UCHAR CharData;};
} GENPORT_WRITE_INPUT;

HANDLE hWin32Device;
PVOID MemoryBaseAddress;
DWORD dwret;
GENPORT_WRITE_INPUT mydata;
UCHAR DeviceType;

/* Get Windows handle to Device */
hWin32Device = CreateFile("\\\\.\\GEGENIUS0", GENERIC_READ | GENERIC_WRITE,
           0, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);

/* Map Shared RAM into local process */
DeviceIoControl(hWin32Device, IOCTL_GEGENIUS_MAP_MEMORY, NULL, 0,
           &MemoryBaseAddress, sizeof(PVOID), &dwret, NULL);

/* Set Baud rate, Serial Bus Address and make output enable */
mydata.PortNumber = CONFIG_REG;
mydata.CharData = 0x7F; /* 153.6 STD, SBA 31, Outputs enabled */
DeviceIoControl(hWin32Device, IOCTL_GEGENIUS_WRITE_PORT, &mydata,
           sizeof(mydata), NULL, 0, &dwret, NULL);

/* Bring board out of reset and disable watchdog timer */
/* If watchdog were enabled here, need to do a read from Share RAM */
/* to reset watchdog before enabling it */
mydata.PortNumber = CONTROL_REG;
mydata.CharData = 0x60; /* Disable Watchdog, bring out of reset */
DeviceIoControl(hWin32Device, IOCTL_GEGENIUS_WRITE_PORT, &mydata,
           sizeof(mydata), NULL, 0, &dwret, NULL);

/* Wait for 4 seconds to get board up from reset state */
/* If the Watchdog is enabled, need to be reading from Shared RAM */
/* during a startup delay to service Watchdog */
Sleep(4000);

/* Do a Read Map Datagram to a Versamax GNIU at SBA 5 */
/* This is a Transmit Datagram with Reply */
/* Check that Command Block is free or complete */
if ((((volatile UCHAR *)MemoryBaseAddress)[0x8C2] == 0) ||
    (((volatile UCHAR *)MemoryBaseAddress)[0x8C2] == 4))
{
    /* Set Command Byte */
    ((UCHAR *)MemoryBaseAddress)[0x8C3] = 3; /* Transmit with Reply */
    /* Set Destination SBA */
    ((UCHAR *)MemoryBaseAddress)[0x8C4] = 5;
    /* Set Function Code */
    ((UCHAR *)MemoryBaseAddress)[0x8C5] = 0x20;
    /* Set SubFunction Code */
    ((UCHAR *)MemoryBaseAddress)[0x8C6] = 0x2A;
    /* Set Return SubFunction Code */
    ((UCHAR *)MemoryBaseAddress)[0x8C7] = 0x2B;
    /* Set Priority to Normal */
    ((UCHAR *)MemoryBaseAddress)[0x8C8] = 0;
    /* Set Transmit buffer size to zero */
    ((UCHAR *)MemoryBaseAddress)[0x8C9] = 0;

    /* Issue Transmit */
    ((volatile UCHAR *)MemoryBaseAddress)[0x8C2] = 1;

    /* Wait up to 10 seconds for the response */
    for (int i = 0; i < 20; i++)
    {
         Sleep(500);
         if (((volatile UCHAR *)MemoryBaseAddress)[0x8C2] == 4)
             break;
    }

    /* Dump results if status is complete */
    if (((volatile UCHAR *)MemoryBaseAddress)[0x8C2] == 4)
    {
         /* The expected length is 19 bytes */
         if (((UCHAR *)MemoryBaseAddress)[0x8CA] == 19)
         {
             UCHAR ReadDatagramBuffer;

             ReadDatagramBuffer = &((UCHAR *) MemoryBaseAddress)[0x9C2]);
             printf("SBA 5 has\r\n");
             printf("%d bytes of %I data\r\n", ReadDatagramBuffer[3]);
             printf("%d bytes of %AI data\r\n", ReadDatagramBuffer[6]);
             printf("%d bytes of %Q data\r\n", ReadDatagramBuffer[9]);
             printf("%d bytes of %AQ data\r\n", ReadDatagramBuffer[12];
         }
    }
    else
         printf("Command timed out\r\n");
}
else
    printf("Command Block not available.\r\n");

/* Turn off card */
mydata.PortNumber = CONTROL_REG;
mydata.CharData = 0;
DeviceIoControl(hWin32Device, IOCTL_GEGENIUS_WRITE_PORT, &mydata,
            sizeof(mydata), NULL, 0, &dwret, NULL);

/* Free Shared Memory range */
DeviceIoControl(hWin32Device, IOCTL_GEGENIUS_UNMAP_MEMORY,
            &MemoryBaseAddress, sizeof(PVOID), NULL, 0, &dwret, NULL);

/* Close windows handle */
CloseHandle(hWin32Device);
```

Global Data Read

Below is a simple example of how one might read data from a Versamax Genius NIU which is sent over the network as Global Data. It is necessary to lockout the data tables with which data is read in order to guarantee coherency when the input data is larger than one byte.

```
#define CONFIG_REG  0
#define STATUS_REG  1
#define CONTROL_REG 2

#define IOCTL_GEGENIUS_READ_PORT \
CTL_CODE(FILE_DEVICE_UNKNOWN, 0x910, METHOD_BUFFERED, FILE_READ_ACCESS)
#define IOCTL_GEGENIUS_WRITE_PORT \
CTL_CODE(FILE_DEVICE_UNKNOWN, 0x911, METHOD_BUFFERED, FILE_WRITE_ACCESS)
#define IOCTL_GEGENIUS_MAP_MEMORY \
CTL_CODE(FILE_DEVICE_UNKNOWN, 0x912, METHOD_BUFFERED, FILE_ANY_ACCESS)
#define IOCTL_GEGENIUS_UNMAP_MEMORY \
CTL_CODE(FILE_DEVICE_UNKNOWN, 0x913, METHOD_BUFFERED, FILE_ANY_ACCESS)

typedef struct _GENPORT_WRITE_INPUT
{
  ULONG   PortNumber;              // Port # to write to
  union {                          // Data to be output to port
     ULONG LongData;
     USHORT ShortData;
     UCHAR CharData;};
} GENPORT_WRITE_INPUT;

HANDLE hWin32Device;
PVOID MemoryBaseAddress;
DWORD dwret;
GENPORT_WRITE_INPUT mydata;
UCHAR DeviceType;

/* Get Windows handle to Device */
hWin32Device = CreateFile("\\\\.\\GEGENIUS0", GENERIC_READ | GENERIC_WRITE,
           0, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);

/* Map Shared RAM into local process */
DeviceIoControl(hWin32Device, IOCTL_GEGENIUS_MAP_MEMORY, NULL, 0,
           &MemoryBaseAddress, sizeof(PVOID), &dwret, NULL);

/* Set Baud rate, Serial Bus Address and make output enable */
mydata.PortNumber = CONFIG_REG;
mydata.CharData = 0x7F; /* 153.6 STD, SBA 31, Outputs enabled */
DeviceIoControl(hWin32Device, IOCTL_GEGENIUS_WRITE_PORT, &mydata,
           sizeof(mydata), NULL, 0, &dwret, NULL);

/* Bring board out of reset and disable watchdog timer */
/* If watchdog were enabled here, need to do a read from Share RAM */
/* to reset watchdog before enabling it */
mydata.PortNumber = CONTROL_REG;
mydata.CharData = 0x60; /* Disable Watchdog, bring out of reset */
DeviceIoControl(hWin32Device, IOCTL_GEGENIUS_WRITE_PORT, &mydata,
           sizeof(mydata), NULL, 0, &dwret, NULL);

/* Wait for 4 seconds to get board up from reset state */
/* If the Watchdog is enabled, need to be reading from Shared RAM */
/* during a startup delay to service Watchdog */
Sleep(4000);
```

```
/* Request Lockout */
/* Check if table already locked out */
if (((UCHAR *)MemoryBaseAddress)[0xA49] == 0)
{
     /* Check if request already outstanding */
     if (((UCHAR *)MemoryBaseAddress)[0xA48] == 0)
     {
           /* Request lockout */
           ((UCHAR *)MemoryBaseAddress)[0xA48] = 1;
     }
}

/* Wait for lockout to suceed. */
while (((volatile UCHAR *)MemoryBaseAddress)[0xA49] == 0)
     Sleep(500);

/* Read Data */
printf("Inputs from SBA5 %x %x\r\n",
            ((UCHAR *)MemoryBaseAddress)[0x2000 + 5*128],
            ((UCHAR *)MemoryBaseAddress)[0x2000 + 5*128 + 1]);

/* Remove Lockout */
((UCHAR *)MemoryBaseAddress)[0xA48] = 0;

/* Wait for lockout to release. */
while (((volatile UCHAR *)MemoryBaseAddress)[0xA49] == 1)
     Sleep(500);

/* Turn off card */
mydata.PortNumber = CONTROL_REG;
mydata.CharData = 0;
DeviceIoControl(hWin32Device, IOCTL_GEGENIUS_WRITE_PORT, &mydata,
            sizeof(mydata), NULL, 0, &dwret, NULL);

/* Free Shared Memory range */
DeviceIoControl(hWin32Device, IOCTL_GEGENIUS_UNMAP_MEMORY,
            &MemoryBaseAddress, sizeof(PVOID), NULL, 0, &dwret, NULL);

/* Close windows handle */
CloseHandle(hWin32Device);
```

Global Data Write

Below is a simple example of how one might write data to a Versamax Genius NIU which is sent over the network as Global Data. It is necessary to lockout the data tables with which data is written in order to guarantee coherency when the output data is larger than one byte.

```
#define CONFIG_REG  0
#define STATUS_REG  1
#define CONTROL_REG 2

#define IOCTL_GEGENIUS_READ_PORT \
CTL_CODE(FILE_DEVICE_UNKNOWN, 0x910, METHOD_BUFFERED, FILE_READ_ACCESS)
#define IOCTL_GEGENIUS_WRITE_PORT \
CTL_CODE(FILE_DEVICE_UNKNOWN, 0x911, METHOD_BUFFERED, FILE_WRITE_ACCESS)
#define IOCTL_GEGENIUS_MAP_MEMORY \
CTL_CODE(FILE_DEVICE_UNKNOWN, 0x912, METHOD_BUFFERED, FILE_ANY_ACCESS)
#define IOCTL_GEGENIUS_UNMAP_MEMORY \
CTL_CODE(FILE_DEVICE_UNKNOWN, 0x913, METHOD_BUFFERED, FILE_ANY_ACCESS)

typedef struct _GENPORT_WRITE_INPUT
{
  ULONG   PortNumber;             // Port # to write to
  union {                         // Data to be output to port
     ULONG LongData;
     USHORT ShortData;
     UCHAR CharData;};
} GENPORT_WRITE_INPUT;

HANDLE hWin32Device;
PVOID MemoryBaseAddress;
DWORD dwret;
GENPORT_WRITE_INPUT mydata;
UCHAR DeviceType;

/* Get Windows handle to Device */
hWin32Device = CreateFile("\\\\.\\GEGENIUS0", GENERIC_READ | GENERIC_WRITE,
           0, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);

/* Map Shared RAM into local process */
DeviceIoControl(hWin32Device, IOCTL_GEGENIUS_MAP_MEMORY, NULL, 0,
           &MemoryBaseAddress, sizeof(PVOID), &dwret, NULL);

/* Set Baud rate, Serial Bus Address and make output enable */
mydata.PortNumber = CONFIG_REG;
mydata.CharData = 0x7F; /* 153.6 STD, SBA 31, Outputs enabled */
DeviceIoControl(hWin32Device, IOCTL_GEGENIUS_WRITE_PORT, &mydata,
           sizeof(mydata), NULL, 0, &dwret, NULL);

/* Bring board out of reset and disable watchdog timer */
/* If watchdog were enabled here, need to do a read from Share RAM */
/* to reset watchdog before enabling it */
mydata.PortNumber = CONTROL_REG;
mydata.CharData = 0x60; /* Disable Watchdog, bring out of reset */
DeviceIoControl(hWin32Device, IOCTL_GEGENIUS_WRITE_PORT, &mydata,
           sizeof(mydata), NULL, 0, &dwret, NULL);

/* Wait for 4 seconds to get board up from reset state */
/* If the Watchdog is enabled, need to be reading from Shared RAM */
/* during a startup delay to service Watchdog */
Sleep(4000);
```

```
/* Request Lockout */
/* Check if table already locked out */
if (((UCHAR *)MemoryBaseAddress)[0xA49] == 0)
{
     /* Check if request already outstanding */
     if (((UCHAR *)MemoryBaseAddress)[0xA48] == 0)
     {
           /* Request lockout */
           ((UCHAR *)MemoryBaseAddress)[0xA48] = 1;
     }
}

/* Wait for lockout to suceed. */
while (((volatile UCHAR *)MemoryBaseAddress)[0xA49] == 0)
     Sleep(500);

/* Write Data */
((UCHAR *)MemoryBaseAddress)[0x3000 + 5*128] = 0x55;
((UCHAR *)MemoryBaseAddress)[0x3000 + 5*128 + 1] = 0xAA;

/* Remove Lockout */
((UCHAR *)MemoryBaseAddress)[0xA48] = 0;

/* Wait for lockout to release. */
while (((volatile UCHAR *)MemoryBaseAddress)[0xA49] == 1)
     Sleep(500);

/* Turn off card */
mydata.PortNumber = CONTROL_REG;
mydata.CharData = 0;
DeviceIoControl(hWin32Device, IOCTL_GEGENIUS_WRITE_PORT, &mydata,
            sizeof(mydata), NULL, 0, &dwret, NULL);

/* Free Shared Memory range */
DeviceIoControl(hWin32Device, IOCTL_GEGENIUS_UNMAP_MEMORY,
            &MemoryBaseAddress, sizeof(PVOID), NULL, 0, &dwret, NULL);

/* Close windows handle */
CloseHandle(hWin32Device);
```

Reading Fault Information

Below is a simple example of how one might poll to see faults come in from a Versamax Genius NIU.

```
#define CONFIG_REG  0
#define STATUS_REG  1
#define CONTROL_REG 2

#define IOCTL_GEGENIUS_READ_PORT \
CTL_CODE(FILE_DEVICE_UNKNOWN, 0x910, METHOD_BUFFERED, FILE_READ_ACCESS)
#define IOCTL_GEGENIUS_WRITE_PORT \
CTL_CODE(FILE_DEVICE_UNKNOWN, 0x911, METHOD_BUFFERED, FILE_WRITE_ACCESS)
#define IOCTL_GEGENIUS_MAP_MEMORY \
CTL_CODE(FILE_DEVICE_UNKNOWN, 0x912, METHOD_BUFFERED, FILE_ANY_ACCESS)
#define IOCTL_GEGENIUS_UNMAP_MEMORY \
CTL_CODE(FILE_DEVICE_UNKNOWN, 0x913, METHOD_BUFFERED, FILE_ANY_ACCESS)

typedef struct _GENPORT_WRITE_INPUT
{
  ULONG   PortNumber;              // Port # to write to
  union {                          // Data to be output to port
     ULONG LongData;
     USHORT ShortData;
     UCHAR CharData;};
} GENPORT_WRITE_INPUT;

HANDLE hWin32Device;
PVOID MemoryBaseAddress;
DWORD dwret;
GENPORT_WRITE_INPUT mydata;
UCHAR DeviceType;

/* Get Windows handle to Device */
hWin32Device = CreateFile("\\\\.\\GEGENIUS0", GENERIC_READ | GENERIC_WRITE,
           0, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);

/* Map Shared RAM into local process */
DeviceIoControl(hWin32Device, IOCTL_GEGENIUS_MAP_MEMORY, NULL, 0,
           &MemoryBaseAddress, sizeof(PVOID), &dwret, NULL);

/* Set Baud rate, Serial Bus Address and make output enable */
mydata.PortNumber = CONFIG_REG;
mydata.CharData = 0x7F; /* 153.6 STD, SBA 31, Outputs enabled */
DeviceIoControl(hWin32Device, IOCTL_GEGENIUS_WRITE_PORT, &mydata,
           sizeof(mydata), NULL, 0, &dwret, NULL);

/* Bring board out of reset and disable watchdog timer */
/* If watchdog were enabled here, need to do a read from Share RAM */
/* to reset watchdog before enabling it */
mydata.PortNumber = CONTROL_REG;
mydata.CharData = 0x60; /* Disable Watchdog, bring out of reset */
DeviceIoControl(hWin32Device, IOCTL_GEGENIUS_WRITE_PORT, &mydata,
           sizeof(mydata), NULL, 0, &dwret, NULL);

/* Wait for 4 seconds to get board up from reset state */
/* If the Watchdog is enabled, need to be reading from Shared RAM */
/* during a startup delay to service Watchdog */
Sleep(4000);


/* The fault data from the Versamax ENIU is sent as fault reporting and */
/* as mail. See if there are any mail messages of type subfunction 0x30 */
/* in the queue from SBA 5 */

DWORD tickCount = GetTickCount();
```

```
/* Try to read an unsolicited datagram from the card */
while(GetTickCount() - tickCount < 10000)
{
     /* Set Command to read datagram */
     ((UCHAR *)MemoryBaseAddress)[0x8C3] = 1;

     /* Set Status to incoming */
     ((UCHAR *)MemoryBaseAddress)[0x8C2] = 1;

     /* Wait while status is still incoming or in progress */
     while((((volatile UCHAR *)MemoryBaseAddress)[0x8C2] == 1) ||
            (((volatile UCHAR *)MemoryBaseAddress)[0x8C2] == 2))
           Sleep(500);

     /* If status is complete, go look at datagram */
     if (((volatile UCHAR *)MemoryBaseAddress)[0x8C2] == 4)
     {
           /* Look at datagram */
           printf("Datagram from SBA %d\r\n",
                  ((volatile UCHAR *)MemoryBaseAddress)[0x8C4]);
           printf("Datagram Subfunction %x\r\n",
                  ((volatile UCHAR *)MemoryBaseAddress)[0x8C6]);
           printf("Datagram Data Length %d\r\n",
                  ((volatile UCHAR *)MemoryBaseAddress)[0x8C8]);

           /* If subfunction is mail, decode a bit more */
           if (((volatile UCHAR *)MemoryBaseAddress)[0x8C6] == 0x30)
           {
                 /* Decode Mail Transport to get I/O or PLC Fault */
           }
           printf("\r\n");
     }
     else if (((volatile UCHAR *)MemoryBaseAddress)[0x8C2] == 8)
     {
           /* Read Queue empty */
     }
     else
     {
           printf("Read Queue error %x.\r\n",
                  ((volatile UCHAR *)MemoryBaseAddress)[0x8C2]);
     }
}

/* Turn off card */
mydata.PortNumber = CONTROL_REG;
mydata.CharData = 0;

DeviceIoControl(hWin32Device, IOCTL_GEGENIUS_WRITE_PORT, &mydata,
           sizeof(mydata), NULL, 0, &dwret, NULL);

/* Free Shared Memory range */
DeviceIoControl(hWin32Device, IOCTL_GEGENIUS_UNMAP_MEMORY,
           &MemoryBaseAddress, sizeof(PVOID), NULL, 0, &dwret, NULL);

/* Close windows handle */
CloseHandle(hWin32Device);
```

Chapter 5 Diagnostics

This chapter provides some suggestions to help resolve startup problems and common operational issues.

Checking Status and Operation using the Control Panel Applets

After the card is installed, a control panel applet called "GE Genius" is available on Windows NT 4.0. On Windows 2000/XP, the card should show up in the Device Manager of the System control panel under Genius Devices.

Using the test button or test tab, one can turn on each card and see the LEDs light. This will show that the card turns on and help determine which card is which when multiple cards are installed.

If the card will not power up with this utility, either the driver is not installed properly or there is a hardware problem. Driver errors should be reported on the General Tab of the Device Manager in Windows 2000/XP or in the Windows Event Log.

Warning: Turning on the PCI Genius card with the control panel applet will bring the card online at SBA 0, 153.6Kbaud (STD). This may disrupt a working network.

Checking the LEDs at Runtime

A healthy operating network should have both the GENI OK and COMM OK LEDs on solid. If the COMM OK LED is going off occasionally and there are delays in the network operations, review the following items:

- Cabling order throughout the entire network. Crossing X1 and X2 on a single node in the network can cause intermittent problems.
- Cable termination. The Genius network should be terminated on both ends. The value of the termination depends on the cable type. Some nodes, including the PCI Genius card may include optional termination resistance internal to the node based on DIP switch settings, jumpers, or configuration.

Closing all Resources

Be sure to call CloseHandle() on the device handle on all exits from the application. Also, unmap the Shared RAM pointer before closing, but after all accesses to the Shared RAM are complete.

Allowing a user-mode application direct access to the hardware is helpful to previous ISA card users, but runs the risk of memory access problems if the application is not closed correctly. Users may also find using try/catch handling on the Shared RAM helpful if their application is not stable.

Appendix A Glossary

API	Application programming interface. A set of routines used by an application program to direct the performance of procedures by the computer's operating system.
ISA	Industry Standard Architecture bus. The bus architecture used in the IBM PC/XT and PC/AT. The AT version of the bus is called the AT bus and became a de facto industry standard.
MSDN	The Microsoft Developer Network is a set of online and offline services designed to help developers write applications using Microsoft products and technologies. See http://msdn.microsoft.com/
PCI	Peripheral Component Interconnect, a local bus standard developed by Intel Corporation. Most modern PCs include a PCI bus in addition to a more general ISA expansion bus.
PCIM	PC Interface Module.
SBA	Serial Bus Address. A node identifier on the Genius network. Typically ranges from 0-31.
Win32	The Windows API for developing 32-bit applications. Win32 is built into Windows 95 and Windows NT so applications that rely on the API (*Win32 applications*) should run equally well in both environments.
WDM	Windows Driver Model. A driver technology developed by Microsoft to create drivers that are source-code compatible for Windows 98, 2000, Me and XP

Index