16AI64SSA

16-Bit, 64 Channel
Simultaneous Sampling Analog Input

Windows 98\2K\XP Driver User Manual

Manual Revision: May 20, 2008

General Standards Corporation
8302A Whitesburg Drive
Huntsville, AL 35802
Phone: (256) 880-8787
Fax: (256) 880-8788
URL: http://www.generalstandards.com
E-mail: sales@generalstandards.com
E-mail: support@generalstandards.com

Preface

Additional copies of this manual or other literature may be obtained from:

General Standards Corporation
8302A Whitesburg Dr.
Huntsville, Alabama 35802
Phone: (256) 880-8787
FAX: (256) 880-8788
URL: http://www.generalstandards.com
E-mail: sales@generalstandards.com

General Standards Corporation makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Although extensive editing and reviews are performed before release to ECO control, **General Standards Corporation** assumes no responsibility for any errors that may exist in this document. No commitment is made to update or keep current the information contained in this document.

General Standards Corporation does not assume any liability arising out of the application or use of any product or circuit described herein, nor is any license conveyed under any patent rights or any rights of others.

General Standards Corporation assumes no responsibility for any consequences resulting from omissions or errors in this manual or from the use of information contained herein.

General Standards Corporation reserves the right to make any changes, without notice, to this product to improve reliability, performance, function, or design.

The Purchaser of this software may use or modify in source form the subject software, but not to re-market or distribute it to outside agencies or separate internal company divisions. The software, however, may be embedded in the Purchaser's distributed software. In the event the Purchaser's customers require the software source code, then they would have to purchase their own copy of the software.

General Standards Corporation makes no warranty of any kind with regard to this software, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose and makes this software available solely on an "as-is" basis. **General Standards Corporation** reserves the right to make changes in this software without reservation and without notification to its users.

GSC is a trademark of **General Standards Corporation**.

PLX and PLX Technology are trademarks of PLX Technology, Inc.

Table of Contents

1. Scope

The Purpose of this document is to describe how to interface with the 16AI64SSA Windows Driver API developed by General Standards Corporation (GSC). This software provides the interface between the “Application Software” and the 16AI64SSA board.

The 16AI64SSA Driver API Software executes under control of the Windows Operating System. The 16AI64SSA is implemented as a standard Windows driver API written in “C” programming language. The 16AI64SSA Driver API Software is designed to operate on CPU boards containing x86 processors.

The 16AI64SSA Driver consists of a Windows driver with an interface layer (GSC API) to simplify the interface to the PLX Driver. While an application may interface directly to the PLX driver, interfacing to the GSC API layer, will simplify the application software development.

2. Hardware Overview

The PMC66-16AI64SSA board is a single-width PCI mezzanine card (PMC) that provides high-speed simultaneous 16-bit analog input capability for PMC applications. 64 analog input lines can be digitized simultaneously at rates up to 200,000 conversions per second per channel, with software-controlled voltage ranges of ±2.5V, ±5V or ±10V. The board is functionally compatible with the IEEE PCI local bus specification Revision 2.2, and is mechanically and electrically compatible with the IEEE compact mezzanine card (CMC) specification IEEE 1386. A PCI interface adapter supports the "plug-n-play" initialization concept.

Selftest switching networks permit the converters to be calibrated automatically to an internal voltage reference. Offset and gain trimming of the converters is performed by calibration DAC's that are loaded with channel correction values during initialization. The correction values are determined during auto calibration, and are stored in nonvolatile EEprom for subsequent transfer to the calibration DAC's. Either auto calibration or initialization can be invoked at any time by asserting a single control bit in the board control register.

The board is designed for minimum off-line maintenance, and includes internal monitoring features that eliminate the need for disconnecting or removing the module from the system for calibration. All analog input and output system connections are made through a single 80-pin, dual-ribbon front-access I/O connector. Power requirements consist of +5 VDC, in compliance with the PCI specification, and operation over the specified temperature range is achieved with conventional convection cooling.

3. Referenced Documents

The following documents provide reference material for the 16AI64SSA board:

- PMC66-16AI64SSA User's Manual – GSC
- PLX Technology, Inc. PCI 9056 PCI Bus Master Interface Chip data sheet.

4. General Standards API

This section describes the interface to the 16AI64SSA GSC API. The 16AI64SSA GSC API isolates the user from operating system specific requirements, allowing the API to be used with most Windows operating systems (98\W2K\XP).

The 16AI64SSA Win Driver provides an interface to a 16AI64SSA card and a Windows application, which run on a x86 target processor. The driver is installed and devices are created when the driver is started during boot up. The functions of the driver can then be used to access the board. Devices are created with the name “board x” where “x” is the device number. Device numbers start at 1 and for each board found the device number will increment.

Included in the board driver software is a menu driven board application program. This program is delivered undocumented and unsupported but may be used to exercise the card and the device driver. It can also be used as an example for programming the 16AI64SS device.

The user interfaces to the GSC API at the basic level with the following functions:

- Find Boards() - Detects all PLX Devices connected via the PCI Bus.
- Get Handle() - Opens a driver interface to one 16AI64SS card.
- Readlocal32() - Reads local registers from one 16AI64SS card.
- Writelocal32() - Writes to local Registers of one 16AI64SS card.
- Close Handle() - Closes a driver interface to one 16AI64SS card.

The user MUST call Find Boards to determine what PLX devices are installed in the system, and get the associated board number. The user then calls the Get Handle function with each board number to be used. This function obtains a handle to the device and initializes the device parameters within the API / driver. The user is then free (assuming no errors) to write / read the registers as desired. The user should always call Close Handle when done to free resources prior to exiting.

The function definitions and parameters are defined in the following paragraphs of this section.

4.1 AI64SSA_FindBoards()

Detects all PLX Devices connected via the PCI Bus.

Prototype:

```
U32  AI64SSA_FindBoards (char      *pDeviceInfo,
                         U32       *ulError);
```

Returns – Total number of PLX boards found in the system or –1L if error or no boards found.

Where:

pDeviceInfo – Contains “Board #: Bus: Slot: Type: Ser#” info for PLX boards found.

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

4.2 AI64SSA_Get_Handle

Initializes Handle for the passed board number IN THE DRIVER.

Prototype:

```
U32  AI64SSA_Get_Handle(U32      *ulError,
                        U32      BoardNumber);
```

Returns – Error code if invalid board number passed (0, >10), else # boards.

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

4.3 AI64SSA_Read_Local32

Read a value from the board local register.

Prototype:

```
U32  AI64SSA_Read_Local32   (U32     BoardNumber,
                             U32     *ulError,
                             U16     iRegister);
```

Returns – Value read from the register.

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

iRegister – Register to read. Values defined in 66-AI64SSAintface.h

```
BCR                 Board Control Register
ICR                 Interrupt Control Register
IN_DATA_BUFF        Input FIFO Register
IN_DATA_CNTRL       Input Buffer Control Register
RATE_A              Rate A Generator Register
RATE_B              Rate B Generator Register
BUFF_SIZE           Number of Samples Register
BURST_SIZE          # of Sample Clocks in a triggered Burst
SCAN_CNTRL          Scan and Sync Control Register
ACTIVE_CHANS        Number of Active Channels Register
FW_REV              Firmware Register (Undocumented)
AUTOCAL             Autocal Register (Undocumented)
UVAL                Aux R/W Register (Undocumented)
AUX SYNC            Auxiliary Sync I/O Register
SCAN_MARKER_U       Packed Data scan marker (D[31..16])
SCAN_MARKER_L       Packed Data scan marker (D[15..00])
```

4.4 AI64SSA_Write_Local32

Write a value to the board local register.

Prototype:

```
void  AI64SSA_Write_Local32(U32     BoardNumber,
                            U32     *ulError,
                            U16     iRegister
                            U32     ulValue);
```

Returns – N/A

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

iRegister – Register to write. Values defined in AI64SSintface.h

```
BCR                 Board Control Register
ICR                 Interrupt Control Register
IN_DATA_BUFF        Input FIFO Register
IN_DATA_CNTRL       Input Buffer Control Register
RATE_A              Rate A Generator Register
RATE_B              Rate B Generator Register
BUFF_SIZE           Number of Samples Register
BURST_SIZE          # of Sample Clocks in a triggered Burst
SCAN_CNTRL          Scan and Sync Control Register
ACTIVE_CHANS        Number of Active Channels Register
FW_REV              Firmware Register (Undocumented)
AUTOCAL             Autocal Register (Undocumented)
UVAL                Aux R/W Register (Undocumented)
AUX SYNC            Auxiliary Sync I/O Register
SCAN_MARKER_U       Packed Data scan marker (D[31..16])
SCAN_MARKER_L       Packed Data scan marker (D[15..00])
```

ulValue – Value to write to the selected register.
Refer to the 16AI64SSA user manual for all register / bit definitions.

4.5 AI64SSA_Close_Handle

Closes the device handle and frees the resources.

Prototype:

```
void  AI64SSA_Close_Handle(U32     BoardNumber,
                           U32     *ulError);
```

Returns – N/A

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

4.6 Interface Functions

These functions allow the user to perform certain operations on the board, without having to keep track of individual register values and bit definitions.

4.6.1 AI64SSA_Initialize

Perform a reset on the board. All register values are set to defaults. This Function does NOT wait for Initialization to complete, such that multiple boards can be initialized without waiting for each to finish.

Prototype:

```
void AI64SSA_Initialize (U32 BoardNumber,
                         U32 *ulError);
```

Returns – N/A

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

4.6.2 AI64SSA_Autocal

Perform an auto calibration on the board. This operation generates new calibration correction values which are stored in nonvolatile EEprom. This function utilizes the autocal interrupt to determine when autocal is completed.

Prototype:

```
U32  AI64SSA_Autocal   (U32      BoardNumber,
                        U32      *ulError);
```

Returns – Autocal status (0 = Failed, 1 = Passed), 0x55 if insufficient resources, 0xAA if interrupt not requested.

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

4.6.3 AI64SSA_Set_Processing_Mode

Sets the data processing mode of the board: Single-Ended, Psuedo Differential or Full Differential.

Prototype:

```
void  AI64SSA_Set_Processing_Mode    (U32    BoardNumber,
                                      U32    *ulError
                                      U32    ulMode);
```

Returns – N/A

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

ulInputMode – Valid values: 0 – 2
0 = Single-Ended
1 = Psuedo Differential
2 = Full Differential

4.6.4 AI64SSA_Set_Input_Mode

Sets the input mode of the board: Single-Ended or Selftest (zero or Vref).

Prototype:

```
void  AI64SSA_Set_Input_Mode    (U32    BoardNumber,
                                 U32    *ulError
                                 U32    ulInputMode);
```

Returns – N/A

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

ulInputMode – Valid values: 0 – 3
0 = Single-Ended
1 = Reserved
2 = Zero Selftest
3 = +Vref Selftest

4.6.5 AI64SSA_Set_Input_Type

Sets the input type of the board: Bipolar Unipolar.

Prototype:

```
void  AI64SSA_Set_Input_Type    (U32    BoardNumber,
                                 U32    *ulError
                                 U32    ulInputType);
```

Returns – N/A

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

ulInputMode – Valid values: 0 – 1
0 = Bipolar
1 = Unipolar

4.6.6 AI64SSA_Clear_Input_Buffer

Clears all data from the active input buffer.

Prototype:

```
void  AI64SSA_Clear_Input_Buffer   (U32     BoardNumber,
                                    U32     *ulError);
```

Returns – N/A

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

4.6.7 AI64SSA_Enable_Input_Buffer

Enables data collection in the input buffer.

Prototype:

```
void  AI64SSA_Enable_Input_Buffer  (U32     BoardNumber,
                                    U32     *ulError);
```

Returns – N/A

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

4.6.8 AI64SSA_Disable_Input_Buffer

Disables data collection into the input buffer.

Prototype:

```
void  AI64SSA_Disable_Input_Buffer (U32     BoardNumber,
                                    U32     *ulError);
```

Returns – N/A

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

4.6.9 AI64SSA_EnableInterrupt

Enables the desired interrupt in the local register, and/or for the PCI bus. See 16AI64SSA User manual for interrupt sources.

Prototype:

```
U32  AI64SSA_EnableInterrupt (U32     BoardNumber,
                              U32     ulValue,
                              U32     ulType,
                              U32     *ulError);
```

Returns – Interrupt value set for Local, else ulValue for DMA Channel.

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

ulValue – The desired interrupt value to set, valid for 0 – 0x77 local. 0/1 for DMA

ulType – The desired type to set, 0 = Local ; 1 = DMA

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

4.6.10 AI64SSA_DisableInterrupt

Disables all interrupts in the local register, and for the PCI bus.

Prototype:

```
void  AI64SSA_DisableInterrupt    (U32     BoardNumber,
                                   U32     ulValue,
                                   U32     ulType,
                                   U32     *ulError);
```

Returns – N/A.

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

ulValue – The desired interrupt value to clear, valid for 0 – 0x77 local. 0/1 for DMA

ulType – The desired type to clear, 0 = Local ; 1 = DMA

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

4.6.11 AI64SSA_Open_DMA_Channel

Opens the desired DMA channel for transferring data from the buffer.

Prototype:

```
void  AI64SSA_Open_DMA_Channel(U32     BoardNumber,
                               U32     ulChannel,
                               U32     *ulError);
```

Returns – N/A.

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

uiValue – The desired channel to open, 0 or 1 .

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

4.6.12 AI64SSA_DMA_FROM_Buffer

Transfers the desired number of WORDS from the board input buffer. The user should set the threshold value to DMA words –1 to satisfy the hardware DemandMode requirements.

Prototype:

```
U32  AI64SSA_BlockDMA_FROM_Buffer   (U32     BoardNumber,
                                     U32     ulChannel,
                                     U32     ulWords,
                                     U32*    uData,
                                     U32     *ulError);
```

Returns – WORDS transferred if no error.

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

ulChannel – The DMA channel previously opened, 0 or 1.

ulWords – Number of WORDS to transfer. (BYTES = ulWords*4).

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

4.6.13 AI64SSA_Close_DMA_Channel

Closes the desired DMA channel.

Prototype:

```
void  AI64SSA_Close_DMA_Channel (U32  BoardNumber,
                                 U32  ulChannel,
                                 U32  *ulError);
```

Returns – N/A.

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

uiValue – The desired channel to close, 0 or 1.

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

4.6.14 AI64SSA_Setup_DmaCmdChaining

Sets up the descriptor blocks for SGL DMA transfer. A maximum of four (4) blocks can be setup per board. The descriptors are setup to be executed in turn by the hardware DMA controller based on the number of blocks specified in the function call. The first descriptor is always block1, and the blocks should be used in numerical order, i.e to specify three blocks, use block1, block2 and block3; such that specifying block1, block2 and block4 would be unacceptable.

NOTE: All descriptor blocks for a board are executed on the same DMA Channel.
NOTE: This function will disable clocking if internal clocking is enabled in the BCR.
NOTE: This function will CLEAR the input buffer.

Prototype:

```
U32  AI64SSA_Setup_DmaCmdChaining   (U32                    BoardNumber,
                                     GS_DMA_DESCRIPTOR      *DmaSetup,
                                     U32                    *ulError);
```

Returns – 0 if no errors, 1 for invalid board number, 2 for invalid data passed in DmaSetup, 3 for insufficient resources (physical memory to store descriptors), 4 is Reserved, 5 for error writing registers.

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

DmaSetup– GS_DMA_DESCRIPTOR with the following members: See 66-AI64SSAintface.h

BytesDesc_1	# of BYTES to transfer for Block1. Valid 2 – 8388607.
BytesDesc_2	# of BYTES to transfer for Block2. Valid 2 – 8388607.
BytesDesc_3	# of BYTES to transfer for Block3. Valid 2 – 8388607.
BytesDesc_4	# of BYTES to transfer for Block3. Valid 2 – 8388607.
PhyAddrDesc_1	Valid PHYSICAL address for contiguous memory block
PhyAddrDesc_2	Valid PHYSICAL address for contiguous memory block
PhyAddrDesc_3	Valid PHYSICAL address for contiguous memory block
PhyAddrDesc_4	Valid PHYSICAL address for contiguous memory block
NumDescriptors	Number of Descriptor blocks to use. Valid 1-4.
LocalToPciDesc_1	Direction of Transfer. 1= Local->Pci. Only Valid value is 1.
LocalToPciDesc_2	Direction of Transfer. 1= Local->Pci. Only Valid value is 1.
LocalToPciDesc_3	Direction of Transfer. 1= Local->Pci. Only Valid value is 1.
LocalToPciDesc_4	Direction of Transfer. 1= Local->Pci. Only Valid value is 1.
InterruptDesc_1	Generate interrupt at Descriptor completion IF ENABLED
InterruptDesc_2	Generate interrupt at Descriptor completion IF ENABLED
InterruptDesc_3	Generate interrupt at Descriptor completion IF ENABLED
InterruptDesc_4	Generate interrupt at Descriptor completion IF ENABLED
DmaChannel	Valid 0 or 1.

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

4.6.15 AI64SSA_Start_DmaCmdChaining

Sets the DMA start bit on the DMA Controller. The only error checking performed is to ensure the Enable DMA bit is set.

NOTE: The user should call AI64SSA_Setup_DmaCmdChaining prior to calling this function.

Prototype:

```
U32  AI64SSA_Start_DmaCmdChaining    (U32     BoardNumber,
                                      U32     ulChannel,
                                      U32     *ulError);
```

Returns – 0 if no error, 1 for invalid board # or error, 2 for DMA Channel not ready.

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

ulChannel – The DMA channel previously opened, 0 or 1.

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

4.6.16 AI64SSA_Close_DmaCmdChaining

Clears the DMA Controller start and enable bits to stop DMA transfers.

Prototype:

```
void  AI64SSA_Close_DmaCmdChaining   (U32  BoardNumber,
                                      U32  ulChannel,
                                      U32  *ulError);
```

Returns – N/A.

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

uiValue – The desired channel to close, 0 or 1.

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

4.6.17 AI64SSA_Reset_Device

Performs a software reset on the device.

Prototype:

```
void  AI64SSA_Reset_Device(U32     BoardNumber,
                           U32     *ulError);
```

Returns – N/A.

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

4.6.18 AI64SSA_Get_Physical_Memory

Obtains a contiguous block of physical memory from the OS for use with DMA, and maps it to user virtual memory space. The user requests a block size by setting the .Size member and the function returns the actual size in the .Size member if successful.

Prototype:

```
U32  AI64SSA_Get_Physical_Memory  (U32              BoardNumber,
                                  GS_PHYSICAL_MEM  *memPtr,
                                  BOOLEAN          bSmallerOk,
                                  U32              *ulError);
```

Returns – 0 if error occurred, else the actual memory size obtained.

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

memPtr – A GS_PHYSICAL_MEM structure with the following members:

UserAddr	The virtual memory address of the mapped memory block
PhysicalAddr	The actual physical address of the block to pass for DMA use
Size	Requested size passed, actual size returned if no errors

bSmallerOk — If a smaller block than requested is acceptable, set to 1
If set to FALSE (0), an error will be returned if memory unavailable

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

4.6.19 AI64SSA_Free_Physical_Memory

Unmap's and free's a physical memory block previously obtained by AI64SSA_Get_Physical_Memory().

Prototype:

```
VOID  AI64SSA_Free_Physical_Memory  (U32              BoardNumber,
                                     GS_PHYSICAL_MEM *memPtr,
                                     U32             *ulError);
```

Returns – N/A.

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

memPtr – A GS_PHYSICAL_MEM structure that has been initialized.

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

4.6.20 AI64SSA_Register_Interrupt_Notify

Registers a user event for interrupt notification.
NOTE: This function DOES NOT enable the interrupts the notification is based on.

Prototype:

```
void  AI64SSA_Register_Interrupt_Notify   (U32                BoardNumber,
                                          GS_NOTIFY_OBJECT    *event,
                                          U32                 ulIntr,
                                          U32                 ulType,
                                          U32                 *ulError);
```

Returns – N/A.

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

event – User creates an event and stores the handle in a GS_NOTIFY_OBJECT structure member hEvent. The user should not set or change the other members of the structure.

ulIntr – Used when ulType equals one (1). Valid for 0 (DMA0) or 1 (DMA1).

ulType – Determines interrupt type to attach to. Valid for 0 (Local) or 1 (DMA).

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

4.6.21 AI64SSA_Cancel_Interrupt_Notify

Cancels interrupt notification for a user event.

Prototype:

```
void  AI64SSA_Cancel_Interrupt_Notify     (U32                BoardNumber,
                                          GS_NOTIFY_OBJECT    *event,
                                          U32                 *ulError);
```

Returns – N/A.

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

event – A valid GS_NOTIFY_OBJECT structure which has been initialized by a call to AI64SSA_Register_Interrupt_Notify.

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

4.6.22 AI64SSA_Dma_DataMode

Enables / disables data packing **FOR DMA TRANSFERS**. When enabled, 2 channels 16 bit data is packed into each 32 bit lword transferred.

Prototype:

```
void  AI64SSA_Dma_DataMode (U32 BoardNumber,
                            U32 ulPack);
```

Returns – N/A.

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

ulPack – Valid for zero [0] (Normal) or nonzero [>0] (Packed).

5. Driver Installation

This section details driver installation on the target system. Any current driver previously installed for the 16AI64SSA must be uninstalled prior to this installation to avoid interference.

To install the driver, API, and associated example files, insert the CD ROM into the drive and close the bay. The installation should commence automatically and display user prompts. Follow the onscreen instructions to complete the installation.

Should the installation fail to automatically start, Select ***Start → Run → Browse*** on the Windows toolbar/popup and browse to find ***Setup.exe*** on the CD ROM. Click on ***OK*** to commence the installation.

The following files are installed on the target system:

OS dependent\…\GS66AI64SSA.sys
OS dependent\…\GSeApi.dll
Program Files\General Standards\66_16AI64SSA\eExample.exe
Program Files\General Standards\66_16AI64SS\AI64SSAe Driver C.dll
Program Files\General Standards\66_16AI64SS\AI64SSAe Driver C.lib
Program Files\General Standards\66_16AI64SS\66-AI64SSAintface.h
Program Files\General Standards\66_16AI64SS\66-AI64SSA_Example.c
Program Files\General Standards\66_16AI64SS\Tools.c
Program Files\General Standards\66_16AI64SS\Tools.h
Program Files\General Standards\66_16AI64SS\CioColor.h
Program Files\General Standards\66_16AI64SS\66-AI64SSAedriver.inf

6. Example Program

This section describes the example program, and the files required to develop an application.

The complied example program allows the user to exercise the installed device, while observing the outputs. To execute, double click on 'eExample.exe'. Refer to the Driver Installation section for file location.

The source is provided to educate the user with the GSC API function calls and provide a working example to aid the user with application development. To build the example program using MS Visual C++, create a project and add the following files:

Source Files	→ ***66-AI64SSA_Example.c***
	→ ***Tools.c***
Header Files	→ ***66-AI64SSAintface.h***
	→ ***CioColor.h***
	→ ***Tools.h***
Resource Files	→ ***AI64SSAe Driver C.lib***

Select ***Build → [ProjectName].exe*** on the toolbar.
NOTE: AI64SSAe Driver C.dll must be in the project directory to run the example.

Contact GSC for example programs (drivers) for other development environments (i.e LabVIEW™, LabWindows/CVI™, etc.)