24DSI

24-Bit, 20 Channel Delta-Sigma, Analog Input

Windows 2K\XP\Vista\W7 Driver User Manual

Manual Revision: April 18, 2011

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

Copyright © 2002, **General Standards Corporation**

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.

ALL RIGHTS RESERVED.

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.

The information in this document is subject to change without notice. This document may be copied or reproduced provided it is in support of products from **General Standards Corporation**. For any other use, no part of this document may be copied or reproduced in any form or by any means without prior written consent of **General Standards Corporation.**

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 24DSI Windows Driver API developed by General Standards Corporation (GSC). This software provides the interface between the “Application Software” and the 24DSI board.

The 24DSI Driver API Software executes under control of the Windows Operating System. The 24DSI is implemented as a standard Windows driver API written in “C” programming language. The 24DSI Driver API Software is designed to operate on CPU boards containing x86 processors.

The 24DSI 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 24DSI board is a 6U cPCI board that provides 24-bit analog input. In addition to providing twenty analog input channels, the board supports multi-board clocking and synchronization. The board is functionally and mechanically compatible with the IEEE PCI local bus specification Revision 2.3, and supports the "plug-n-play" initialization concept. Power requirements consist of +5 VDC in accordance with the PCI specification, and operation over the specified temperature range is achieved with minimal (150 LFPM) air-cooling. Specific details pertaining to physical characteristics and performance are contained in the cPCI6U64-24DSI20 product specification. 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 system input and output system connections are made at the front panel through Dsub connectors.

** Number of Channels available are dependant on the ordering options provided by the customer.

3. Referenced Documents

The following documents provide reference material for the 24DSI board:

- CPCI6U64-24DSI20 User's Manual – GSC
- PLX Technology, Inc. PCI 9656 PCI Bus Master Interface Chip data sheet.

4. General Standards API

This section describes the interface to the 24DSI GSC API. The 24DSI GSC API isolates the user from operating system specific requirements, allowing the API to be used with all Windows operating systems (W2K\XP\Vista\Win7).

The 24DSI Win Driver provides an interface to a 24DSI 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 24DSI 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 24DSI card.
- Readlocal32() - Reads local registers from one 24DSI card.
- Writelocal32() - Writes to local Registers of one 24DSI card.
- Close Handle() - Closes a driver interface to one 24DSI 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 DSI20_FindBoards()

Detects all PLX Devices connected via the PCI Bus.

Prototype:

```
U32  DSI20_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 DSI20_Get_Handle

Initializes Handle for the passed board number IN THE DRIVER.

Prototype:

```
U32  DSI20_Get_Handle  (U32      *ulError,
                        U32      BoardNumber);
```

Returns – Error code if invalid board number passed (0, >31), 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 DSI20_Read_Local32

Read a value from the board local register.

Prototype:

```
U32  DSI20_Read_Local32     (U32     BoardNumber,
                             U32     *ulError,
                             U32     ulRegister);
```

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.

ulRegister – Register to read. Values defined in DSI20eIntface.h

```
BCR                 0x00
RATE_CONTROL_A      0x04
DIG_IO              0x08
CLK_SOURCE          0x0C
RATE_DIV            0x10
Reserved0           0x14
Reserved1           0x18
BURST_BLOCK_SIZE    0x1C
BUFFER_CONTROL      0x20
BOARD_CONFIG        0x24
BUFF_SIZE           0x28
AUTOCAL             0x2C
INPUT_DATA_BUFFER   0x30
Reserved2           0x34
Reserved3           0x38
BURST_TRG_TIMER     0x3C
```

4.4 DSI20_Write_Local32

Write a value to the board local register.

Prototype:

```
void  DSI20_Write_Local32     (U32      BoardNumber,
                               U32      *ulError,
                               U32      ulRegister
                               U32      uiValue);
```

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.

ulRegister – Register to write. Values defined in DSI20eIntface.h

BCR	0x00
RATE_CONTROL_A	0x04
DIG_IO	0x08
CLK_SOURCE	0x0C
RATE_DIV	0x10
Reserved0	0x14
Reserved1	0x18
BURST_BLOCK_SIZE	0x1C
BUFFER_CONTROL	0x20
BOARD_CONFIG	0x24
BUFF_SIZE	0x28
AUTOCAL	0x2C
INPUT_DATA_BUFFER	0x30
Reserved2	0x34
Reserved3	0x38
BURST_TRG_TIMER	0x3C

uiValue – Value to write to the selected register.
Refer to the 24DSI20 user manual for all register / bit definitions.

4.5 DSI20_Close_Handle

Closes the device handle and frees the resources.

Prototype:

```
void  DSI20_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 DSI20_Initialize

Perform a reset on the board. All register values are set to defaults.

Prototype:

```
void  DSI20_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 DSI20_Autocal

Perform an auto calibration on the board. This operation generates new calibration correction values which are stored in nonvolatile EEprom.

Prototype:

```
void  DSI20_Autocal     (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.

4.6.3 DSI20_Set_Input_Mode

Sets the input mode of the board: Differential or SelfTest (Zero or +Vref).

Prototype:

```
void  DSI20_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: Differential (0), Zero SelfTest (2), VREF SelfTest (3).

4.6.4 DSI20_Clear_Buffer

Clears all data from the input buffer.

Prototype:

```
void  DSI20_Clear_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.

4.6.5 DSI20_Buffer_Control

Enables / Disables the input data buffer.

Prototype:

```
void  DSI20_Buffer_Control    (U32      BoardNumber,
                               U32      ulValue,
                               U32      *ulError);
```

Returns – N/A

Where:

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

ulValue – Valid Values: 0 = Disable , 1 = Enable input buffer.

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

4.6.6 DSI20_Set_Data_Width

Sets the desired data width: 16, 18, 20, or 24 bit.

Prototype:

```
void  DSI20_Set_Width   (U32      BoardNumber,
                         U32      ulWidth,
                         U32      *ulError);
```

Returns – N/A

Where:

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

ulWidth – Valid Values: 0 = 16bit , 1 = 18bit, 2 = 20bit, 3 = 24bit.

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

4.6.7 DSI20_EnableInterrupt

Enables the desired interrupt in the local register, and for the PCI bus. See 24DSI User manual for interrupt sources.

Prototype:

```
U32  DSI20_EnableInterrupt    (U32    BoardNumber,
                               U32    ulValue,
                               U32    ulType,
                               U32    *ulError);
```

Returns – Interrupt value set.

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 – 5.

ulType – The desired type to set, valid for LOCAL (0) or DMA (1).

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

4.6.8 DSI20_DisableInterrupt

Disables the interrupt in the local register, and for the PCI bus.

Prototype:

```
void  DSI20_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 disable, valid for 0 – 5.

ulType – The desired type to disable, valid for LOCAL (0) or DMA (1).

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

4.6.9 DSI20_Open_DMA_Channel

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

Prototype:

```
void  DSI20_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.

ulChannel – The desired channel to open, valid for channel 0 or 1 .

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

4.6.10 DSI20_DMA_ToVirtualMem

Transfers the desired number of WORDS from the board input buffer to user defined memory.

Prototype:

```
U32  DSI20_DMA_ToVirtualMem     (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, valid for channel 0 or 1.

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

uData – User defined (Virtual) memory.

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

4.6.11 DSI20_Close_DMA_Channel

Closes the desired DMA channel.

Prototype:

```
void  DSI20_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.

ulChannel – The desired channel to close, valid for channel 0 or 1.

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

4.6.12 DSI20_Register_Interrupt_Notify

Attaches a user supplied handle to an interrupt which can be used in WaitForSingleObject for notification when the interrupt occurs. A sample use is provided in the Autocal function of the example program.

```
 DSI20_Register_Interrupt_Notify(ulBdNum, &event, 0x01, LOCAL, &ulErr);
…
… Setup and code to cause interrupt to happen
…

 EventStatus = WaitForSingleObject(myHandle,10 * 1000);

…
 switch(EventStatus)
 {
    case WAIT_OBJECT_0:
        … code to perform desired action

        break;
    default:
        cprintf("Interrupt was NOT requested...");
        break;
 }
```

Prototype:

```
void  DSI20_Register_Interrupt_Notify      (U32              BoardNumber,
                                           GS_NOTIFY_OBJECT  event,
                                           U32               ulInterrupt,
                                           U32               ulType,
                                           U32               *ulError);
```

Returns – N/A.

Where:

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

event – User supplied handle is stored in the event.

ulInterrupt – The desired interrupt to attach to.

ulType – The desired type to attach to LOCAL (0) or DMA (1).

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

4.6.13 DSI20_Cancel_Interrupt_Notify

Releases a user supplied handle from interrupt notification.

Prototype:

```
void  DSI20_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 – Event where the user supplied handle was stored.

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

5. Driver Installation

This section details driver installation on the target system. Any current driver previously installed for the 24DSI must be uninstalled prior to this installation to avoid interference. A restore (OS dependent) may be required to completely remove the previous installation components.

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\…\GS66DSI20.sys
OS dependent\…\GSebApi.dll
Program Files\General Standards\24DSI\DSI20 eDriver C.dll
Program Files\General Standards\24DSI\DSI20 eDriver C.lib
Program Files\General Standards\24DSI\DSI20 eExample.c
Program Files\General Standards\24DSI\Tools.c
Program Files\General Standards\24DSI\Tools.h
Program Files\General Standards\24DSI\CioColor.h
Program Files\General Standards\24DSI\DSI20eIntface.h
Program Files\General Standards\24DSI\66-24DSI20.inf
Program Files\General Standards\24DSI\eExample.exe

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 inputs or outputs. To execute, double click on 'Example PLL.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	→ ***DSI20 eExample.c***
	→ ***Tools.c***
Header Files	→ ***DSI20eIntface.h***
	→ ***CioColor.h***
	→ ***Tools.h***
Resource Files	→ ***DSI20 eDriver C.lib***

Select ***Build → [ProjectName].exe*** on the toolbar.

NOTE: DSI20 eDriver C.dll must be in the project directory or [Windows Dir]\system32 to run the example.

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