

GTX2200

GC2200

**Universal Time Interval Counter
PXI and PCI Boards
GXCNT Software**

User's Guide

Last updated March 19, 2014

Safety and Handling

Each product shipped by Marvin Test Solutions is carefully inspected and tested prior to shipping. The shipping box provides protection during shipment, and can be used for storage of both the hardware and the software when they are not in use.

The circuit boards are extremely delicate and require care in handling and installation. Do not remove the boards from their protective plastic coverings or from the shipping box until you are ready to install the boards into your computer.

If a board is removed from the computer for any reason, be sure to store it in its original shipping box. Do not store boards on top of workbenches or other areas where they might be susceptible to damage or exposure to strong electromagnetic or electrostatic fields. Store circuit boards in protective anti-electrostatic wrapping and away from electromagnetic fields.

Be sure to make a single copy of the software diskette for installation. Store the original diskette in a safe place away from electromagnetic or electrostatic fields. Return compact disks (CD) to their protective case or sleeve and store in the original shipping box or other suitable location.

Warranty

Marvin Test Solutions products are warranted against defects in materials and workmanship for a period of 12 months. Software products and accessories are warranted for 3 months. Unless covered by software support or maintenance agreement. Marvin Test Solutions shall repair or replace (at its discretion) any defective product during the stated warranty period. The software warranty includes any revisions or new versions released during the warranty period. Revisions and new versions may be covered by a software support agreement. If you need to return a board, please contact Marvin Test Solutions Customer Technical Services department via <http://www.marvintest.com/magic> the Marvin Test Solutions on-line support system.

If You Need Help

Visit our web site at <http://www.marvintest.com> for more information about Marvin Test Solutions products, services and support options. Our web site contains sections describing support options and application notes, as well as a download area for downloading patches, example, patches and new or revised instrument drivers. To submit a support issue including suggestion, bug report or question please use the following link: <http://www.marvintest.com/magic>.

You can also use Marvin Test Solutions technical support phone line (949) 263-2222. This service is available between 7:30 AM and 5:30 PM Pacific Standard Time.

Disclaimer

In no event shall Marvin Test Solutions or any of its representatives be liable for any consequential damages whatsoever (including unlimited damages for loss of business profits, business interruption, loss of business information, or any other losses) arising out of the use of or inability to use this product, even if Marvin Test Solutions has been advised of the possibility for such damages.

Copyright

Copyright © 2003-2014 by Marvin Test Solutions, Inc. All rights reserved. No part of this document can be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Marvin Test Solutions.

Trademarks

| | |
|---|---|
| ATEasy®, CalEasy, DIOEasy®, DtifEasy, WaveEasy® | Marvin Test Solutions (prior name is Geotest - Marvin Test Systems Inc.) |
| C++ Builder, Borland C++, Pascal and Delphi | Embarcadero Technologies Inc. |
| LabView, LabWindows™/CVI | National Instruments |
| Microsoft Developer Studio, Microsoft Visual C++, Microsoft Visual Basic, .NET, Windows 95, 98, NT, ME, 2000, XP and VISTA and , Windows 7 or 8 | Microsoft Corporation |

All other trademarks are the property of their respective owners.

Table of Contents

| | |
|---|------------|
| Safety and Handling..... | i |
| Warranty | i |
| If You Need Help..... | i |
| Disclaimer | i |
| Copyright | i |
| Trademarks | ii |
| Table of Contents..... | iii |
| Chapter 1 - Introduction | 1 |
| Manual Scope and Organization..... | 1 |
| Manual Scope..... | 1 |
| Manual Organization..... | 1 |
| Conventions Used in this Manual | 1 |
| Chapter 2 - Overview | 3 |
| Introduction..... | 3 |
| Features..... | 3 |
| Applications..... | 5 |
| Board Description | 5 |
| Architecture | 7 |
| Block Diagram | 7 |
| Measurement Functions | 7 |
| Frequency..... | 7 |
| Input Signal Pre-Scaling | 8 |
| Fast Frequency | 9 |
| Time Interval..... | 9 |
| Time Interval with Delay | 10 |
| Period | 10 |
| Single Period..... | 11 |
| Ratio & AutoRatio | 11 |
| Totalize, Gated Totalize, & Accumulate..... | 11 |
| Pulse Width..... | 12 |
| Auto Trigger and Hold..... | 12 |
| Arming | 13 |
| Paced Measurements..... | 14 |
| Triggered Pacing | 14 |
| Single Measurement..... | 14 |

| | |
|---|-----------|
| Input Prescaling..... | 15 |
| External Clock Input | 15 |
| PXI 10MHz Clock Source | 15 |
| Counter Reference Clock for PXI Reference Clock Source..... | 15 |
| External Arm Input | 15 |
| Gate Output..... | 15 |
| Coupling DC/AC..... | 16 |
| Filtering Input Signal | 16 |
| Common Input Mode | 16 |
| Specifications..... | 17 |
| Virtual Panel Description..... | 23 |
| Virtual Panel Initialize Dialog | 24 |
| Virtual Panel Setup Page..... | 25 |
| Condition (Group Box) | 25 |
| Acquisition Settings | 26 |
| Virtual Panel Channel A \ Channel B Page..... | 27 |
| Virtual Panel Advanced Page..... | 28 |
| Virtual Panel About Page..... | 30 |
| Chapter 3 - Installation and Connections | 31 |
| Getting Started | 31 |
| Packing List | 31 |
| Unpacking and Inspection..... | 31 |
| System Requirements..... | 31 |
| Installation of the GXCNT Software | 32 |
| Overview of the GXCNT Software | 32 |
| Configuring Your PXI System using the PXI/PCI Explorer..... | 33 |
| Board Installation..... | 34 |
| Before you Begin | 34 |
| Electric Static Discharge (ESD) Precautions | 34 |
| Installing a Board..... | 34 |
| Plug & Play Driver Installation..... | 35 |
| Removing a Board | 36 |
| Connectors | 36 |
| Connectors and Accessories | 37 |
| Installation Folders | 38 |
| GXCNT Driver Files Description..... | 38 |
| Driver File and Virtual Panel | 38 |

| | |
|---|-----------|
| Interface Files..... | 38 |
| On-line Help and Manual..... | 39 |
| ReadMe File..... | 39 |
| Example Programs | 39 |
| Setup Maintenance Program | 40 |
| Chapter 4 - Programming the Board | 41 |
| The GXCNT Driver..... | 41 |
| Programming Using C/C++ Tools..... | 41 |
| Programming Using Visual Basic..... | 41 |
| Programming Using Pascal/Delphi..... | 42 |
| Programming Using ATEasy®..... | 42 |
| Programming Using LabView and LabView/Real Time..... | 42 |
| Using and Programming under Linux..... | 42 |
| Using the GXCNT driver functions | 43 |
| Initialization, HW Slot Numbers and VISA Resource..... | 43 |
| Board Handle | 44 |
| Reset..... | 44 |
| Error Handling | 44 |
| Driver Version..... | 44 |
| Panel..... | 44 |
| Distributing the Driver | 45 |
| Sample Programs | 45 |
| Sample Program Listing | 46 |
| Chapter 5 - In-System Calibration | 51 |
| Introduction..... | 51 |
| Required instrument..... | 51 |
| Calibration Interval | 51 |
| In-System Calibration License Setup..... | 52 |
| Running In-System calibration from the virtual panel..... | 53 |
| In-System Calibration Example program..... | 58 |
| In-System Calibration Sample Program Listing | 58 |
| Chapter 6 - Functions Reference..... | 63 |
| Introduction..... | 63 |
| GC2200/GTX2200 Functions..... | 64 |
| GxCntChannelAutoSet | 68 |
| GxCntClear | 69 |
| GxCntGetAcquisitionMode | 70 |

| | |
|--|-----|
| GxCntGetAcquisitionTimeInterval..... | 71 |
| GxCntGetArmSlope..... | 72 |
| GxCntGetArmSource..... | 73 |
| GxCntGetBoardSummary..... | 74 |
| GxCntGetBoardType..... | 75 |
| GxCntGetCalibrationInfo..... | 76 |
| GxCntGetCalibrationMode..... | 78 |
| GxCntGetChannelAFrequencyRange..... | 79 |
| GxCntGetChannelCouplingMode..... | 80 |
| GxCntGetChannelFilterMode..... | 81 |
| GxCntGetChannelFilterValue..... | 82 |
| GxCntGetChannelImpedance..... | 83 |
| GxCntGetChannelSlope..... | 84 |
| GxCntGetChannelTriggerLevel..... | 85 |
| GxCntGetChannelTriggerLevelMode..... | 86 |
| GxCntGetClockSource..... | 88 |
| GxCntGetCommonInput..... | 89 |
| GxCntGetCounterRefClockToPxiRefClockState..... | 90 |
| GxCntGetDriverSummary..... | 91 |
| GxCntGetErrorString..... | 92 |
| GxCntGetExtendedSerialNumber..... | 95 |
| GxCntGetFunction..... | 96 |
| GxCntGetGateTime..... | 98 |
| GxCntGetMeasurementNumberOfDigits..... | 99 |
| GxCntGetMeasurementTimeout..... | 100 |
| GxCntGetPrescalerMode..... | 101 |
| GxCntGetTimeIntervalDelay..... | 102 |
| GxCntGetTotalizeGateMode..... | 103 |
| GxCntGetTriggerSlope..... | 104 |
| GxCntGetTriggerSource..... | 105 |
| GxCntInitialize..... | 106 |
| GxCntInitializeVisa..... | 107 |
| GxCntInSystemCalDevice..... | 108 |
| GxCntInSystemCalGetStatus..... | 109 |
| GxCntInSystemCalRestore..... | 110 |
| GxCntInSystemCalSave..... | 111 |
| GxCntInSystemCalStart..... | 112 |

| | |
|---|-----|
| GxCntIsMeasurementReady | 113 |
| GxCntPanel | 114 |
| GxCntReadMeasurement | 115 |
| GxCntReadMeasurementArray | 116 |
| GxCntReadMeasurementString | 118 |
| GxCntReadStatusRegister | 119 |
| GxCntReset | 121 |
| GxCntSelfTest | 122 |
| GxCntSetAcquisitionMode | 123 |
| GxCntSetAcquisitionTimeInterval | 124 |
| GxCntSetArmSlope | 125 |
| GxCntSetArmSource | 126 |
| GxCntSetCalibrationMode | 127 |
| GxCntSetChannelAFrequencyRange | 128 |
| GxCntSetChannelCouplingMode | 129 |
| GxCntSetChannelFilterMode | 130 |
| GxCntSetChannelFilterValue | 131 |
| GxCntSetChannelImpedance | 132 |
| GxCntSetChannelSlope | 133 |
| GxCntSetChannelTriggerLevel | 134 |
| GxCntSetChannelTriggerLevelMode | 135 |
| GxCntSetClockSource | 137 |
| GxCntSetCommonInput | 138 |
| GxCntSetCounterRefClockToPxiRefClockState | 139 |
| GxCntSetFunctionAccumulate | 140 |
| GxCntSetFunctionAutoRatio | 141 |
| GxCntSetFunctionFastFrequency | 142 |
| GxCntSetFunctionFrequency | 143 |
| GxCntSetFunctionPeriod | 144 |
| GxCntSetFunctionWidth | 145 |
| GxCntSetFunctionRatio | 146 |
| GxCntSetFunctionSinglePeriod | 147 |
| GxCntSetFunctionTestInternalClock | 148 |
| GxCntSetFunctionTimeInterval | 149 |
| GxCntSetFunctionTimeIntervalDelay | 150 |
| GxCntSetFunctionTotalize | 151 |
| GxCntSetFunctionTotalizeGated | 152 |

| | |
|--|------------|
| GxCntSetFunctionTotalizeGatedOnce..... | 153 |
| GxCntSetGateTime..... | 154 |
| GxCntSetMeasurementNumberOfDigits..... | 155 |
| GxCntSetMeasurementTimeout..... | 156 |
| GxCntSetPrescalerMode..... | 157 |
| GxCntSetTimeIntervalDelay..... | 158 |
| GxCntSetTotalizeGateMode..... | 159 |
| GxCntSetTriggerSlope..... | 160 |
| GxCntSetTriggerSource..... | 161 |
| GxCntTrig..... | 162 |
| GxCntUpgradeFirmware..... | 163 |
| GxCntUpgradeFirmwareStatus..... | 164 |
| Index..... | 165 |

Chapter 1 - Introduction

Manual Scope and Organization

Manual Scope

This manual provides all the information necessary for installation, operation, and maintenance of the GTX2200 (GTX2230, GTX2220, GTX2210) and GC2200 (GC2230, GC2220, GC2210) families of PXI Universal Time Interval Counters. This manual assumes the reader has a general knowledge of PC based computers, Windows operating systems, and some with using frequency and time interval counters, as well as test equipment in general.





This manual also provides programming information about using the GTX2200/ GC2200 driver (referred in this manual **GXCNT**). Therefore, this manual assumes a thorough understanding of Windows application development tools and languages.

Manual Organization

The GTX2200/GC2200 manual is organized in the following manner:

| Chapter | Content |
|--|--|
| Chapter 1 – Introduction | Introduces the GTX2200/GC2200 manual. Lists all the supported boards and shows warning conventions used in the manual. |
| Chapter 2 – Overview | Provides the GTX2200/GC2200 list of features, description of the individual boards, architecture, specifications and the virtual panel description and operation. |
| Chapter 3 – Installation and Connections | Provides instructions on how to install a GTX2200/ GC2200 board and the GXCNT software. |
| Chapter 4 – Programming the Board | Provides a list of the GXCNT software driver files, general purpose and generic driver functions, and programming methods. Discusses supported application development tools and programming examples. |
| Chapter 5 – In-System Calibration | Describes the procedure used to perform the in-system calibration. |
| Chapter 6 – Functions Reference | Provides a list of the GXCNT driver functions. Each function description provides syntax, parameters, and any special programming comments. |

Conventions Used in this Manual

| Symbol Convention | Meaning |
|---|---|
|  | Static Sensitive Electronic Devices. Handle Carefully. |
|  | Warnings that may pose a personal danger to your health. For example, shock hazard. |
|  | Cautions where computer components may be damaged if not handled carefully. |
|  | Tips that aid you in your work. |

| Formatting Convention | Meaning |
|------------------------------|--|
| Monospaced Text | Examples of field syntax and programming samples. |
| Bold type | Words or characters you type as the manual instructs. For example: function or panel names. |
| <i>Italic type</i> | Specialized terms. Titles of other references and information sources. Placeholders for items you must supply, such as function parameters |

Chapter 2 - Overview

Introduction

The Universal Time Interval Counter has two versions, a PXI and PCI version. Both versions are identical in performance and specifications. The GTX2200 is a Universal Time Interval Counter on a 3U PXI Board while the GC2200 is a Universal Time Interval Counter PCI Board.

The PXI/PCI counters measures frequencies from DC to 2GHz (GTX2230/GC2230). Channel A of the GTX2220, GTX2230, GC2220 and GC2230 has a high frequency option that extends its measurement range to 1.3GHz/2.0GHz respectively. The GTX2200/GC2230 provides a maximum of 10 digits per second with 100pS resolution. Each input channel can be programmed for high (1 MOhm) or low (50 Ohm) input impedance.

The GTX2200/GC2230 includes fourteen pre defined measurement functions. The GTX2200/GC2230 has functions to measure Frequency, Period, Pulse Width and Time Interval to name a few. Every GTX2200/GC2230 has an internal clock source and supports an external reference source. A programmable ARM source provides internal arming or external arming with positive or negative slope. The counter also provides a TTL Gate output that signals when the measurement gate is open or closed.

The GTX2200/GC2230 uses reciprocal counting and dual interpolation techniques to achieve high-resolution measurements on lower frequency signals without sacrificing measurement time. Reciprocal counting provides a fixed number of digits of resolution for all frequencies rather than a fixed resolution in Hz for the same gate time. For example, the GTX22X0/GC22X0 provides 10 digits of resolution for frequencies from 1Hz to 100MHz in one second.

Features

The Universal Time Interval Counter has the following features:

- Two Independent Input Channels
- Three models available: GC2210/GTX2210 input range of DC to 225MHz, GC2220/GTX2220 input range of DC to 1.3GHz and the GC2230/GTX2230 with input range of DC to 2GHz.
- High measurement resolution, GC2220, GC2230, GTX2220 and GTX2230 has 100pS of resolution without averaging while the GC2200/GTX2210 has 10nS of resolution without averaging.
- Fourteen measurement functions are supported: Accumulate, Auto Ratio, Fast Frequency (GC2220/30, GC2220/GC2230/GTX2220/GTX2230 only), Frequency, Period, Ratio, Single Period, Test Clock, Time Interval, Time Interval Delay, Totalize, Totalize Gated, Totalize Gated Once, and Width.
- All functions permit swapping channel reference or order. For example, Time Interval can be measured from A → B or B → A. This is a function of the driver software. This feature simplifies cable assignment and reduces the need for signal switching in test systems.
- Software selectable input impedance (1 MOhm or 50 Ohm) and coupling (AC or DC coupling) for each input channel.
- Programmable, high resolution, wide range Gate Time and Delay Time control. Both parameters can be set with a resolution of 0.75μS, over the range of 250μS to 3200.
- Auto Trigger function automatically sets the input trigger threshold levels just before EACH AND EVERY measurement. The trigger level can then be set to HOLD this level for additional measurements.
- Manual Trigger Level provides a programmable, high-resolution trigger threshold level. The manual trigger level is programmable from -5V to +5V with 1mV resolution (previous PCB had 40mV resolution). This level setting is cached or stored and is not affected if the operator selects Auto Trigger Level and returns to the FIXED Manual Trigger.

- Programmable Input Prescale mode can be set to Continuous, Off or Auto mode. In Auto mode the counter determines the need for prescaling automatically by making a quick frequency check prior to the actual measurement.
- External Gate output signals when the measurement gate is open or closed. The Gate output signal is an active high TTL level during the gate interval and low at other times. It does not require external pull-up or pull-down.
- Paced Measurements with intervals ranging from less than 1mS to 3200 seconds. The pace time is digitally generated and is accurate to within 200 μ S (errors do not accumulate).
- Programmable measurement External Arming mechanism that create a “time window” explicitly selecting specific events. The arming window is defined by events (signal transitions) applied to the External Arm input. The External Arm Slope is programmable (negative or positive).
- Programmable clock source (external or internal).
- Programmable Built-in calibration circuitry. The counter automatically calibrates the interpolators and the channel-to-channel timing skews and compensates for the differential non-linearity inherent in analog time-measurement circuitry. This calibration provides verification of the board operation. Calibration mode can be set to either Continuous, Once or Off.
- Programmable Acquisition mode: continuous, single or paced. Acquisition mode specifies how frequently unarmed measurements should be made. A single measurement or continuous measurements are possible. The Paced measurements define a precise interval between measurements.
- Each of the input channels can be programmed to filter the input signal using a programmable window in time (one shot). The filter permits accurate and stable measurements of signals with frequencies less than 20KHz that have noise or glitches.
- The GTX220/30 (only) boards, when plugged into the star-controller slot 2 of a PXI chassis, the board under software control can replace the 10 MHz PXI clock with its highly stable and accurate oscillator.
- When plugged into one of the non-star controller slots (slots 3 and higher), the GTX22x0 boards can be programmed so that its clock source will use the PXI backplane clock.
- User invoked In-System automatic calibration of the Time base and channels A and B trigger level (PCB version C and above). The In-System Calibration requires purchasing a license for this feature.
- The Universal Time Interval Counter software driver supports both the Virtual Instrument Software Architecture (VISA) standard and Marvin Test Solutions' HW Instrument driver for versions 2.50 and above.

Applications

- PC-Based PCI, cPCI or PXI Automatic Test Equipment Systems
- Event timing and counting measurements.
- Frequency and frequency hopping measurements using the Fast Frequency capability.
- Frequency stability and comparison measurements.

Board Description

The GTX2200 is a Dual Channel Universal Time Interval Counter in a 3U PXI form factor, while the GC2200 is a Dual Channel Universal Time Interval Counter in a PCI form factor. The card has three BNC connectors (input channels A and B and External Clock) and a DIN-6 connector providing connection for External ARM input and TTL Gate output signals.

The input impedance for each channel is independently selectable for High impedance (1 MOhm) or Low Impedance (50 Ohm). Both input channels are capable of measuring frequencies from DC to 225MHz, while Channel A is capable of measuring frequencies from DC to 1.3GHz for the GC2220/GTX2220 and DC to 2GHz for the GC2230/GTX2230. The GC2220/GC2230/GTX2220/GTX2230 models reports 10 digits per second of gate time with 100pS resolution while the GC2210/GTX2210 has 8 digits per second of gate time with 10nS resolution.

All GC22x0/GTX22x0 models support an auto trigger level that automatically searches for the best trigger level before each measurement. Trigger level can also be set manually from -5.12V to $+5.12\text{V}$ with resolution of 0.04V .

The GC2200/GTX2200 uses unique reciprocal measurement techniques and achieves high-resolution measurements on low frequency signals. A reciprocal counting technique results in high-resolution measurements on lower frequency signals without sacrificing measurement time. Reciprocal counting provides a fixed number of digits of resolution for all frequencies rather than a fixed resolution in Hz for the same gate time. For example, the GC2220/GC2230/GTX2220/GTX2230 provides 10 digits of resolution for frequencies from 1Hz to 100MHz in one second of gate time.

Measurement arming is a programmable mechanism that creates a “time window” for explicitly selecting specific events. The arming window is defined by events (signal transitions) applied to the External Arm input.

The counter contains a Programmable Built-in calibration circuitry that does the following:

- Automatically calibrating the interpolators.
- Automatically calibrating the Channel-to-Channel timing skews.
- Compensate for the differential non-linearity inherent in analog time-measurement circuitry
- Provides verification of the board operation.

Calibration mode can be set to either Continuous, Once or Off.

Figure 2-1 shows the GTX2200 with its front panel connectors.

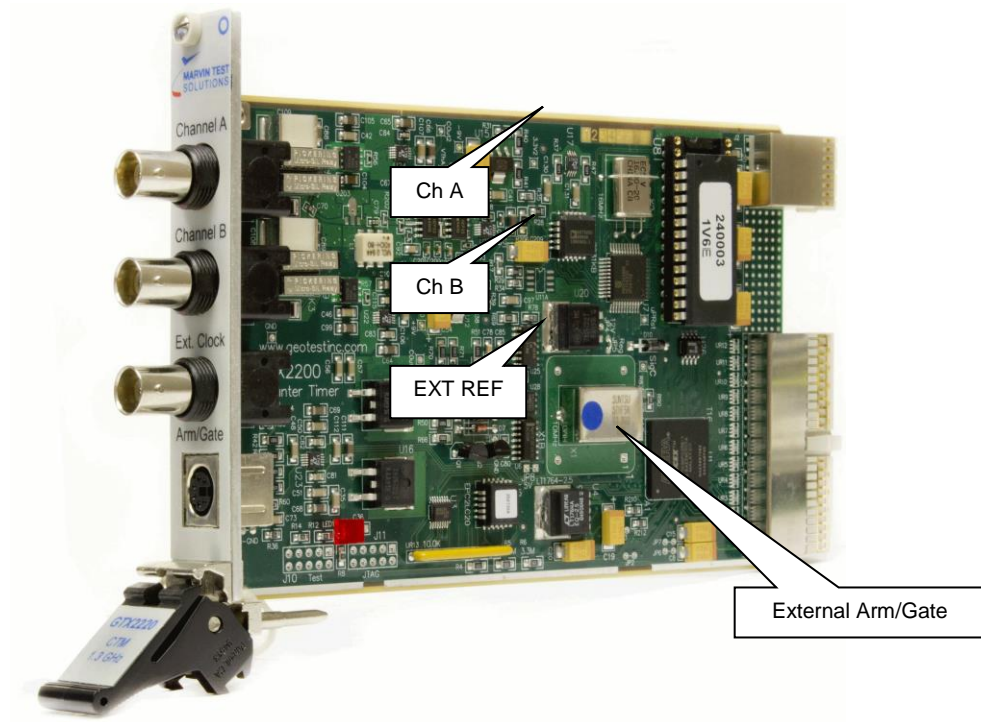


Figure 2-1: GTX2200 Board Side View

Figure 2-2 shows the GC2200 with its front panel connectors.

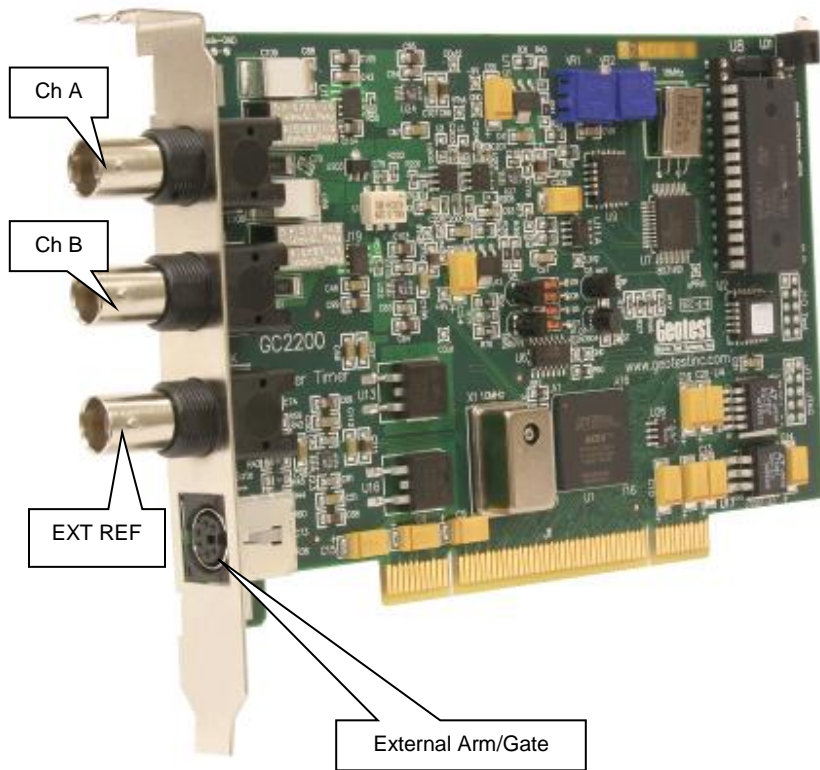


Figure 2-2: GC2200 Board Side View

Architecture

Block Diagram

The block diagram below, Figure 2-3 illustrates the GC2200/GTX2200 architecture. The board communicates with the host computer using the PCI interface. The software support package and application software sets the measurement operation mode, channel A and B settings as well as Pace mode and all other settings. The board has three BNC connectors on the front bezel for input channels A and B, and External Clock.

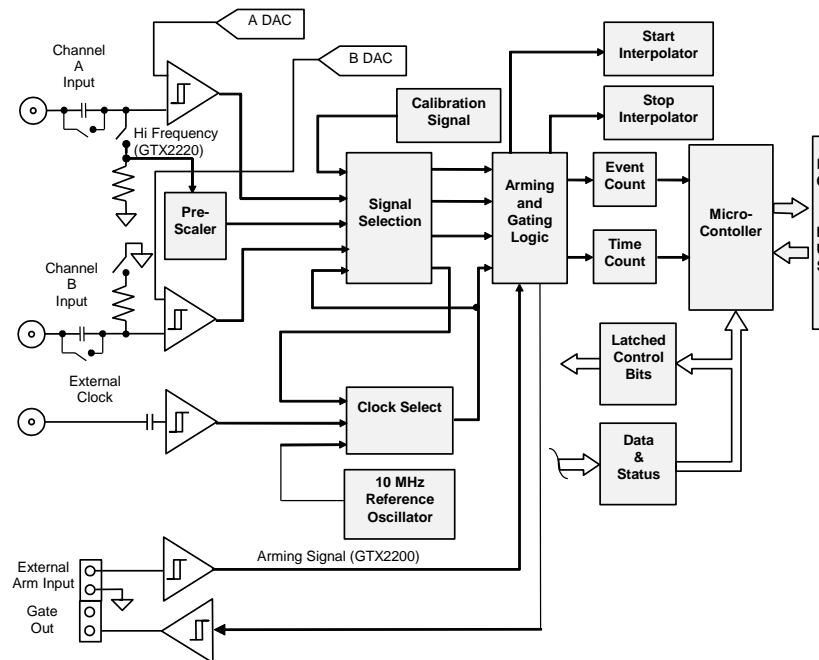


Figure 2-3: GC2200/GTX2200 Block Diagram

Measurement Functions

Frequency

The GC2200/GTX2200 measures the frequency of input signals using reciprocal counting and time interpolation. The two primary benefits for using these methods are improved accuracy and reduced measurement time. Fast measurements with high accuracy permit greater knowledge of the stability of a signal. For example, given a test frequency of 10 kHz, a basic “direct count” counter will resolve 1 Hz using a one-second measurement time. In contrast, the GC2210/GTX2210 can resolve 0.1 Hz using a 1-millisecond measurement time, and the GC2220/GC2230/GTX2220/GTX2230 will resolve 0.001 Hz in 1 millisecond. The GC2200/GTX2200 computes the drift rate, mean, and peak-to-peak jitter of the signal in the same time a conventional counter measures “Frequency” alone

Frequency can be measured on either of the two signal inputs, referred to as Channel A and Channel B. The software supports channel swapping, which allows two signals to be monitored without moving cables or using a signal multiplexer. GC2210/GTX2210 measures signal frequencies ranging from DC to 225 MHz. The GC2220/GTX2220 Channel A has a wider bandwidth and measures frequencies from DC to 1.3GHz and GC2230/GTX2230 DC to 2GHz.

The user must specify the period over which the signal frequency is to be measured. This interval is traditionally referred to as the “gate time”. Gate time provides the user a way to control the measurement parameters. First, longer measurement times increase the number of significant digits in the result and increase the potential accuracy. Second, the gate time defines the averaging time of the measurement. A long gate time is useful if the AVERAGE frequency is of interest, while a series of measurements taken with short gate times can indicate the short-term stability or jitter of the source. The GC2200/GTX220 allows the gate time to be set over a range of 250 micro sec to 3200 seconds.

Because of the measurement technique used, very low frequency signals will modify the actual measurement interval. In frequency mode, the minimum measurement time is one signal period: for example, it takes at least 100 milliseconds to measure a 10 Hz signal, no matter how short the selected gate time. Furthermore, the measurement interval is converted to an integral number of signal periods. For example, with a 10 Hz signal and a gate time specification of 0.1 seconds, the actual measurement time could be either 0.1 or 0.2 seconds. The uncertainty can be eliminated simply by selecting a gate time where no ambiguity could exist: e.g. 0.01 seconds would ensure a one period measurement and 0.15 seconds would guarantee a 2 period measurement. Finally, there may be a significant dead time (up to 1 period) between initiation by the gate and the start of a measurement since that measurement must begin at a period boundary. These effects are mostly of concern to designers of automatic test systems, where unexpected variation in measurement time can cause confusion or malfunction to the test program.

Frequency, Fast Frequency and Period modes can be armed for precise control over the measurement interval starting point. This is especially useful if the signal to be measured is a burst signal or it is frequency modulated. Although Armed Frequency allows the beginning of the gate interval to be defined, the end of the measurement interval is defined by the specified gate time; it cannot be set by the arming signal. For more information see the “Arming” section.

Frequency allows the GC2220/GC2230/GTX2220/GTX2230 to acquire up to 1400/sec theoretical measurements and around real 1100/sec measurements or better. The maximum number of measurements depends on the CPU speed, system configuration and number application running at the background. See the **GxCntReadMeasurementArray** function on how to achieve high number of measurements. For best results the counter settings should as follows:

- The Gate time needs to be set to the minimum (250usec).
- The channel Trigger Level Mode should be set to fixed.
- The channel Trigger Level should be set to fixed value, e.g. 0.5V.
- The channel Prescale Mode should be turned off.

Input Signal Pre-Scaling

The GC2200/GTX2200 will pre-scale a signal over 1MHz. Because the GC2200/GTX2200 utilizes reciprocal counting measurement techniques, pre-scaling does NOT cause a reduction in resolution. The counter determines the need for pre-scaling just before each frequency or period measurement. There are some situations, however, where the user should explicitly turn pre-scaling “OFF”. For example, during high-speed data acquisition, the extra time required to automatically test the signal can reduce the maximum sample rate. Pre-scaling can be disabled if the signal will be less than 10 MHz. Enable the prescale function for any signal above 10MHz. It is also possible to fool the automatic selection process, if the signal frequency varies dramatically over the gate time. Manual selection, based on the maximum expected frequency, ensures correct operation. Note that the only “problem” with pre-scaling is that a minimum of four signal periods must be measured. When the input signal is below about 10 kHz, pre-scaling makes the measurement time significantly longer than specified by the gate time.

If an application requires that frequency resolution and accuracy be maximized, care should be used in “conditioning” the input signals. Any noise (amplitude, frequency, or phase) associated with the signal will affect the measured stability. In fact, large amplitude “spikes” can produce very erroneous results, providing they exceed the hysteresis of the input trigger circuits. If application problems are encountered, check the input signal carefully with a scope. A low pass filter can often eliminate false triggering due to impulse noise and reduce jitter caused by high frequency interference or wideband noise. The GC2200/GTX2200 has a filter that can eliminate erroneous results caused by large amplitude “spikes” for input frequencies less than 20KHz. See the section titled Filtering for more information about using the Filter function.

“Ground” noise between the source and the counter can also contribute to increased measurement jitter and the solution is usually better grounding or some method of isolating the different signal grounds.

Some applications require that frequency measurements begin at very precisely defined times. This control can be provided by measurement arming. See the section titled Arming for more information about using the Armed Frequency function.

Fast Frequency

A special “Fast Frequency” mode of the GC2200/GTX2200 provides frequency measurements using a very short measurement interval (minimum gate time in the standard mode is 250 μ s). The shorter measurement interval permits substantially faster data acquisition. In Fast Frequency, gate time is not a fixed interval, but a fixed number of signal periods; four periods. Fast Frequency measurement mode ignores the gate time setting. Measurement resolution for Fast Frequency mode is best with low frequency signals since the resolution is proportional to the gate time.

Note: While the Fast Frequency mode exists in the GC2210/GTX2210, it has little value because of the lower resolution of the GC2210/GTX2210.

Fast Frequency allows the GC2220/GC2230/GTX2220/GTX2230 to acquire up to 2300/sec theoretical measurements and around real 1400/sec measurements or better. The maximum number of measurements depends on the CPU speed, system configuration and number application running at the background. See the **GxCntReadMeasurementArray** function on how to achieve high number of measurements. For best results the counter settings should as follows:

- The Gate time needs to be set to the minimum (250usec).
- The channel Trigger Level Mode should be set to Fixed.
- The channel Trigger Level should be set to fixed value, e.g. 0.5V.

Of even greater importance, Fast Frequency allows the counter to used for characterization of very rapid frequency transitions, as found in voltage-controlled oscillators or phase locked loops.

Frequency, Fast Frequency and Period modes can be armed for precise control over the measurement interval starting point. This is especially useful if the signal to be measured is a burst signal or it is frequency modulated. Although Armed Frequency allows the beginning of the gate interval to be defined, the end of the measurement interval is defined by the specified gate time; it cannot be set by the arming signal. For more information, see the “Arming” section.

Time Interval

Time Interval is defined as the elapsed time between events on the “start” input channel and events on the “stop” input channel. Time Interval has additional settings to determine whether A or B is the start. An “event” can be defined through the channel slope and trigger level settings. An event edge may be either a positive or a negative transition of the input signal. The trigger level determines where along the transition the event is valid. Figure 2-4 illustrates the Time Interval concept:

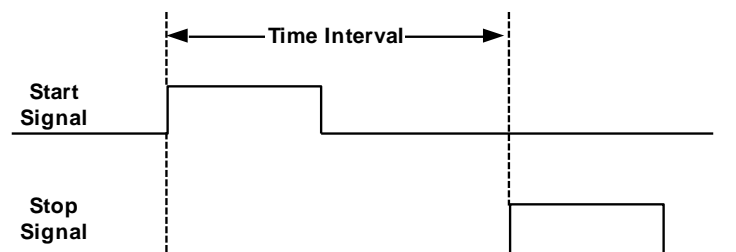


Figure 2-4: Time Interval

Signal swapping allows the user to define either input channel as the start or stop. This function minimizes setup changes in bench top applications, and reduces the need for signal multiplexers in test systems.

All timing functions utilize the time interpolation measurement technique, resulting in a single shot resolution of 10 nanoseconds for the GC2210/GTX2210, and 100 picoseconds for the GC2220/GC2230/GTX2220/GTX2230: this is equivalent to a conventional counter with a 100 MHz or 10 GHz clock rate respectively. The term “single shot” means that only one measurement needs to be made: no averaging is necessary to achieve the stated resolution. Averaging can be used to reduce jitter caused by variations in the input interval and jitter caused by the measurement circuits.

In the Time Interval measurements can be both “start” armed and “stop” armed for maximum flexibility. See the section on Arming below.

Time Interval with Delay

Sometime measurement situations are complicated by the presence of multiple “stop” events. For example, contact “bounce” can prevent an ill-equipped counter from measuring the polling time of a relay, since the interval will always end on the stop event generated by the first bounce. The GTX2200 can utilize a special mode to disable or “hold off” measurement completion to ease such difficult measurement setups.

Time Interval with Delay mode provides this capability through an internal, digitally generated, “hold off”. The user through a dialog box controls the duration of the hold off. A time interval with delay measurement begins like a standard measurement except that the stop event is disabled. As soon as the start event is recognized, the programmed delay begins. When the delay interval elapses, completion is enabled and the next stop event terminates the measurement.

The hold off interval can be specified from 20 μ s to 3200 seconds. Digital generation of the delay interval means that long delays will be very accurate; it also means, however, that the actual hold off period can vary from one measurement to another by a few microseconds, due to clock synchronization uncertainty.

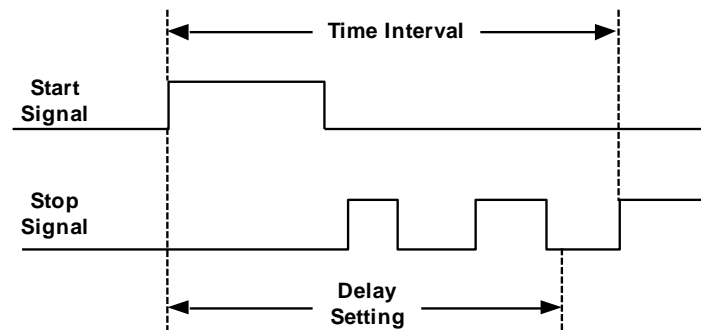


Figure 2-5: Time Interval with Delay

Period

The signal period is measured in exactly the same way as the signal frequency, described above. The period is simply the reciprocal of the frequency, since $\text{Period} = 1/\text{Frequency}$.

Frequency, Fast Frequency and Period modes can be armed for precise control over the measurement interval starting point. This is especially useful if the signal to be measured is a burst signal or it is frequency modulated. Although Armed Frequency allows the beginning of the gate interval to be defined, the end of the measurement interval is defined by the specified gate time; it cannot be set by the arming signal. For more information, see the “Arming” section.

Period allows the GC2220/GC2230/GTX2220/GTX2230 to acquire up to 1400/sec theoretical measurements and around real 1100/sec measurements or better. The maximum number of measurements depends on the CPU speed, system configuration and number application running at the background. See the **GxCntReadMeasurementArray** function on how to achieve high number of measurements. For best results the counter settings should as follows:

- The Gate time needs to be set to the minimum (250usec).
- The channel Trigger Level Mode should be set to fixed.

- The channel Trigger Level should be set to fixed value, e.g. 0.5V.
- The channel Prescale Mode should be turned off.

Single Period

Some applications require that the signal period be measured over exactly one Pace. In Single Period mode, the GC2200/GTX2200 configures the internal measurement logic to perform a time interval. As in time interval mode, resolution is fixed at 10 nanoseconds for GC2210/GTX2210, and at 100 picoseconds for GC2220/GC2230/GTX2220/GTX2230. Note that the specifications require a signal period greater than 25 nanoseconds for this mode.

Although the auto calibration process of the GC2200/GTX2200 minimizes errors, single period will produce results that are less accurate than the conventional period mode. Period mode is superior since it averages errors over multiple signal Paces. For example, if a 1 MHz signal (1 μ s period) is measured over a 1 second interval using Period mode, approximately 10 nanoseconds (GC2210/GTX2210), or 100 picoseconds (GC2220/GC2230/GTX2220/GTX2230) error is distributed over 1 million events, for an average error of 10 femtoseconds (10^{-15} second) for the GC2210/GTX2210, or 100 attoseconds (10^{-18} second) for the GC2220/GC2230/GTX2220/GTX2230. However, in single period, the same total error must be distributed over a single event.

Ratio & AutoRatio

Ratio mode determines the ratio between the signal frequency on one input channel and the signal frequency on the second input channel. As in frequency mode, the measurement is performed during a user definable interval or gate time: longer gate times produce higher resolution results. Signal swapping allows the user to select either A/B or B/A, depending on preference, without swapping cables and reconfiguring the input controls.

AutoRatio, a function unique to the GC2200/GTX2200, removes the setup constraints and uncertainty commonly associated with this function, as implemented on other counters. The GC2200/GTX2200 tests the input signals and automatically selects the internal connections that maximize resolution. If necessary, the measurement results are automatically converted to the format requested by the operator (e.g. A/B or B/A). (Other counters provide optimum performance only when the result is a ratio > 1.000)

Note that only one of the input signals may exceed 25 MHz in Ratio or AutoRatio modes. Furthermore, AutoRatio should not be used when signal frequencies drop below 400 Hz.

Totalize, Gated Totalize, & Accumulate

The Totalize function allows events on one input channel to be counted for a period of time determined by manual keyboard inputs, program statements, or events on the second input channel. In contrast to most other universal counters, the data is accurately reported on EVERY read command. On other counters, the ± 1 count accuracy specification applies only to data read out following gate closure, or at periods when no events are being accumulated.

For maximum flexibility, the GC2200/GTX2200 provides three variations of the Totalize mode. In the “manual” mode the host computer application will explicitly open the count gate, close the count gate, or reset the count to zero. The manual mode provides the most convenient control of counting, but does not allow precise timing.

Gated Totalize is a “gated” mode that allows a second input signal to open and close the gate on user specified signal transitions. After the gate closes, the measurement data is held until explicitly reset by the host computer application, such as the software front panel.

The input slopes control the gating interval for Gated Totalize mode. Selecting same slopes for the start and stop (e.g. both positive) sets the gating interval to one period of the gating signal. Setting opposite slopes for start and stop changes the gating interval to a pulse width of the gating signal. Figure 2-6 illustrates how different slope setting controls the Gated Totalize sample period or interval.

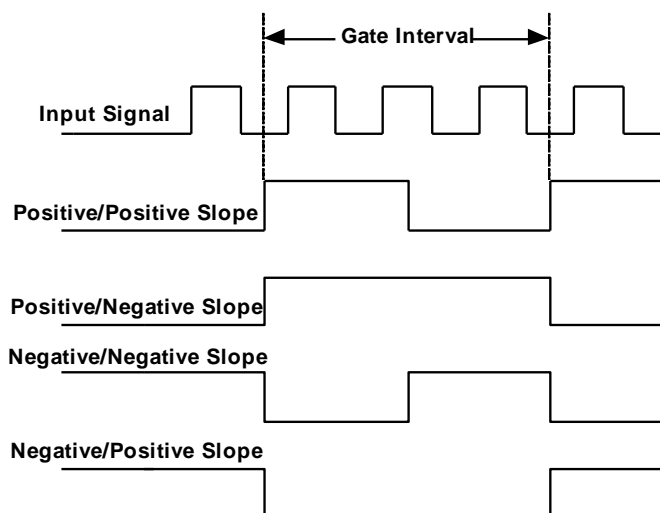


Figure 2-6: Gate Interval controlled by slope settings

In some applications, it is desirable to keep a running total count over multiple gate intervals. The Accumulate mode enables this capability. Accumulate mode, like Gated Totalize, allows an external signal to determine the gate interval. The count gate, however, can be opened and closed an indefinite number of times. Less than 0.5 microsecond of “dead time” is required between the closing of one gate interval and the beginning of the next. Data can be accurately read at any time.

Pulse Width

The Pulse Width function provides a convenient way to automatically setup the GC2200/GTX2200 for this common measurement. The pulse polarity (i.e. positive or negative) is defined by the slope selection. For example, to measure the width of a negative going pulse on Channel B, set the measurement function to “Pulse Width” and set Channel B slope to Negative. Measurement resolution, as in other time interval modes, is 10 nanoseconds for GC2210/GTX2210, and 100 picoseconds for GC2220/GC2230/GTX2220/GTX2230. The threshold levels can be set by the auto trigger function or manually.

Auto Trigger and Hold

The Auto Trigger function automatically sets the input comparator trigger levels just before EACH measurement. The GC2200/GTX2200 first measures the positive and negative peak levels applied to the active input channels. It then sets the trigger threshold levels approximately midway between the measured peaks. This process takes approximately 50 to 250 milliseconds, depending on signal amplitude and frequency. Although Auto Trigger is a very convenient feature, there are some situations where its use is inappropriate. The device driver allows Auto Trigger to be disabled and the levels to be set explicitly. For example, Auto Trigger limits the maximum measurement rate to about 20 measurements per second. When faster rates are needed, Auto Trigger must be disabled (see the discussion of the Hold function, below). Second, Auto Trigger will not operate reliably when the input signal repetition rate falls below 100 Hz. Although the basic algorithm could be extended to lower frequencies, the time required for ALL Auto Trigger operations would increase. Finally, it is inappropriate to use Auto Trigger in any measurement where signal amplitude can vary periodically, as in pulsed RF.

The Hold function captures the benefit of Auto Trigger for an application where it is continuous Auto Trigger is unsuitable, such as in Totalize mode and making high-speed measurements. Trigger level “Hold” retains the last auto trigger setting until the selecting another Trigger Mode.

The steps below demonstrate the Auto Trigger algorithm:

1. Measure signal positive and negative peak levels.
2. Trigger level set half-way between positive and negative peaks.
3. Wait 0.5 second for measurement to start, if the measurement does not start, reset and return to step 1.
4. If the signal is lost after the start of a measurement (frequency is less than 2 Hz), reset and return to step 1.
5. Complete measurement and return to step 1.

Steps 3 and 4 prevent the instrument from “hanging up” if the signal changes amplitude or disappears. Note that Auto Trigger skips step 4 in Time Interval mode, since the actual measurement time is unknown. All test programs using ANY timer/counter should have “time-out” facilities to prevent the possibility of an infinite wait states.

Arming

Some Time Interval measurements, such as circuit propagation delay, are easy to set up. Other measurements can be more challenging, especially if the signal is not perfectly repetitive and if every event is unique. Measurement arming is a significant aid, since it creates a “time window”, explicitly selecting specific events. Arming is especially valuable in test systems, since it ensures repeatable as well as accurate results on every measurement. The arming window is defined by events (signal transitions) applied to the **External Arm input**.

Two modes of armed operation are possible, Start Arm, and Stop Arm:

In the Start Arm mode, the arm input event opens the time window and the next start event initiates the measurement. The FIRST stop event following the start will terminate the measurement. Sometimes, it is desirable to “skip” stop events; for example when the delay from the first to the fifth pulse must be measured. Start Plus Stop Arm allows the stop event to be selected as well as the start event. See Figure 2-6 for an illustration of how Start Arm and Start Plus Stop Arm operates.

In many cases an appropriate arming signal can be found in the circuit under test: if this is not the case, either external equipment or dedicated circuitry must be used. Some useful external tools for generating the arming signals are pulse generators and oscilloscopes.

Pulse generators can provide appropriately delayed signals of variable width to facilitate both arming modes. Some oscilloscopes can provide an easily positioned arm signal from their delayed gate output. The input signal is routed to both the scope and the GC2200/GTX2200 and the delayed gate output is connected to the GC2200/GTX2200 External Arm input; the scope delay controls are then adjusted to “highlight” the appropriate portion of the signal. This technique is especially beneficial since the arm window is visible and can be precisely adjusted to select the desired event.

Frequency, Fast Frequency and Period modes can be armed for precise control over the measurement interval starting point. This is especially useful if the signal to be measured is a burst signal or it is frequency modulated. Although Armed Frequency allows the beginning of the gate interval to be defined, the end of the measurement interval is defined by the specified gate time; it cannot be set by the arming signal.

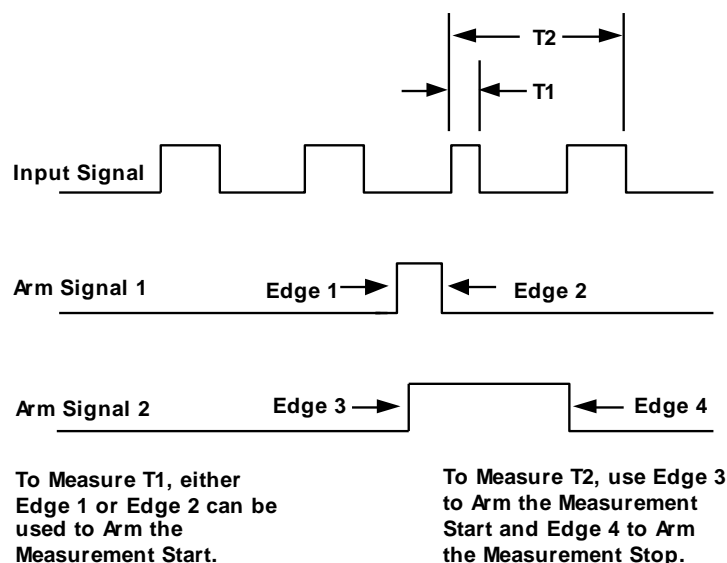


Figure 2-7: How the Arm settings affect Time Interval measurements.

Paced Measurements

The GC2200/GTX2200 features a Pace mode which ensures the acquisition of data at accurately spaced time intervals. In Paced operation, the user specifies the time that should elapse BETWEEN measurements initiations. The pace times are digitally generated on the GC2200/GTX2200 board and are accurate to within 200 μ s (the errors do not accumulate). The intervals can be set from 1 mSec to 3200 seconds depending on measurement mode. The minimum pace interval is 0.8 ms for Time Interval and 1.0 mSec for Frequency and Period. See the Frequency section for timing ambiguities that can occur if the signal has a low repetition rate. The shorter intervals allow the GC2200/GTX2200 to operate much like a frequency domain, digital sampling oscilloscope, while the longer intervals allow unattended acquisition of slowly changing data.

Some care must be used when selecting pacing intervals to ensure consistent timing between measurements. The Acquisition interval must be at least 1.5 ms longer than the actual gate time in Frequency and Period modes. The Acquisition interval should be at least 1.5 ms longer than the actual measurement interval in timing modes, such as Pulse Width. If these recommendations are not followed, the counter will simply “skip” measurements, and the data will be taken at uneven intervals.

Triggered Pacing

In many situations it is advantageous to synchronize the start of data acquisition to an external signal. Signal applied to the External Arm input to initiate a block of paced measurements. For convenience, either a positive or negative slope can be selected as the trigger event. When triggered mode is disabled, pacing is initiated immediately after a setup or reset. For example, the GC2200/GTX2200 can be used to determine the frequency versus time performance of a VCO (voltage-controlled oscillator). Triggered Acquisition allows the counter to wait until the arrival of the stimulus pulse before beginning acquisition.

Single Measurement

The Single mode halts display update after a single measurement is made. This feature is useful when results must be recorded manually, or when a high-resolution result must be checked carefully.

Input Prescaling

Prescaling is necessary when the input frequency exceeds 10 MHz in Frequency, Period, Ratio, and AutoRatio modes (see specifications). Prescaling does not affect resolution or accuracy in any way. Prescaling can be enabled or disabled by the user, or it can be set to Auto mode. In Auto mode, the counter determines the need for prescaling automatically by making a quick frequency check prior to the each and every actual measurement. The process takes about 20 μ S.

If the input frequency exceeds 10 MHz and arming is enabled, the prescaler must be explicitly turned on. This is because the counter disables the prescaler whenever arming is enabled to reduce potential measurement start point ambiguity. Armed measurements may require up to 4 Paces of the input signal before the counter starts.

External Clock Input

The third signal input on the GC2200/GTX2200 is used for an External Clock Input. The External Clock Input employs high-speed comparators, with hysteresis, to provide external clock source signal conditioning. The input is AC coupled with input impedance of 2 KOhm. The input requires at least 150 mV rms of signal amplitude (420 mV peak to peak). Note that the external clock frequency must be within 5% of 10 MHz, or the internal circuits may malfunction, reducing resolution.

PXI 10MHz Clock Source

The PXI chassis has a 10 MHz clock line (PXI_CLK10) on its backplane that allows devices plugged into the non-star controller slots (slots 3 and higher) to phase-lock their oscillators to the backplane clock. When plugged into one of the non-star controller slots (slots 3 and higher), the GTX22x0 board can be programmed to use the PXI backplane clock.

Counter Reference Clock for PXI Reference Clock Source

The PXI chassis' 10 MHz clock line (PXI_CLK10) allows devices plugged into the non-star controller slots (slots 3 and higher) to phase-lock their oscillators to the backplane clock. The GTX22x0 board, when plugged into the star-controller slot (slot 2) can replace the PXI backplane clock with its much more stable oscillator and function as the PXI_CLK10 source.

Note: Valid only if the GTX22x0 board is plugged into the star-controller slot (slot 2) of the PXI chassis.

External Arm Input

The External Arm Input (DIN-6 pin 1) is used for the function of an external arm signal input. The input stage employs high-speed comparators, with hysteresis, to provide signal conditioning. The arm function is DC coupled with input impedance of 2 KOhm with TTL level signals: i.e. the threshold voltage is approximately 1.4V.

Gate Output

In some situations, it is advantageous to know exactly when the GC2200/GTX2200 measurement gate opens and closes. Applications would include debugging a setup in which arming is used or to verify input comparator trigger points. The "gate out" signal is available on the GC2200/GTX2200 board from the Gate Output (DIN-6, pin 3) connector. The output is TTL level, with a 100-ohm series resistor. This combination can drive a 50-ohm cable and 50-ohm termination and provide about 0.8 Volts peak to peak to the load.

The Gate Out signal is a TTL "high" during the gate interval, and low at other times. Several of the automatic functions available on the GC2200/GTX2200 can cause spurious "gate out" signals just before the expected gate. If the gate events generated by auto trigger, auto calibration and auto prescale cause a problem, they can be explicitly disabled. For example, spurious gate events may cause a scope to trigger erratically, or may prematurely trigger another instrument.

Note: An example application for the "gate out" signal would be to use it to arm a second GC2200/GTX2200. Two counters could be used to acquire sequential measurements on a complex signal. The first GC2200/GTX2200 would acquire a data point and arm the second: the second counter could then immediately capture a related signal transition. The counters could be interconnected with a simple cable.

Coupling DC/AC

Each of the input channels can be programmed for DC coupling or AC coupling which is the default setting after a reset.

Filtering Input Signal

Each of the input channels can be programmed to filter the input signal. The filter is a window in time (one shot), which starts every time the input signal crosses the trigger threshold level. The length of the window is programmable in the range of 5uS to 6400 uSec. Transitions in the input signal during this window will be ignored. Filtering is effective for input frequencies less than 20 KHz.

Note: The delay needs to be less than half of the expected frequency signal duty cycle. Filter value can be manually set when filter mode is set to FIXED. The filter value should be set according to the following equation:

$$\text{Filter Value (uS)} \leq \frac{1.0E + 6}{\text{ExpectedFrequency} * 4}$$

Maximum Filter value in uSec can be 6400.

E.g.: if the expected value is about 200Hz then filter value should be around 1250uSec.

The filter permits accurate and stable measurements of signals that have noise or glitches. Typical applications would include encoders and sensors. The filter provides a way to prevent false triggering after the initial trigger.

Common Input Mode

Programmable mode which allows the user to make measurements on the same input signal without the need to connect both inputs (when using a common input the trigger slopes for channel A and channel B cannot be the same).

Note: Valid only for GTX2210/20/30 board with firmware versions 0xCXXX and above.

Specifications

The following table outlines the specifications of the GC2200/GTX2200.

| Input Characteristics Channels A and B | |
|---|---|
| Impedance | 1M Ω or 50 Ω software programmable |
| Coupling | DC or AC (per channel) software programmable |
| Maximum Signal Input | 1 M Ω : 15V rms (DC to 1MHz), 5V rms (above 1MHz) |
| | 50 Ω : 5V rms |
| Connectors | Front panel BNC s for Channels A and B |
| Frequency Range | |
| GTX2230 | Channel A: Programmable DC to 225 MHz or 100 MHz to 2GHz |
| | Channel B: DC to 225MHz |
| GTX2220 | Channel A: Programmable DC to 225 MHz or 100 MHz to 1.3GHz |
| | Channel B: DC to 225MHz |
| GC2210/GTX2210 | Channels A and B: DC to 225MHz |
| Signal Range | +5 V to -5 V |
| Sensitivity | |
| Sine | 25 mV rms DC – 20MHz |
| | 50 mV rms 20MHz – 50MHz |
| | 200 mV rms 50MHz – 100MHz |
| | 500 mV rms 100MHz – 1.3GHz (GTX2220) |
| | 500 mV rms 100MHz – 2GHz (GTX2230) |
| Pulse | 500mV pk-pk at 5nS pulse width |
| Trigger (Threshold) Level | |
| Range | PCB revision C and above: $\pm 5.00V$ in 1mV steps PCB revision A and B: $\pm 5.00V$ in 40mV steps |
| Accuracy | $\pm 3\%$ of setting $\pm 0.001V$ |
| Auto Trigger | Automatic selection of optimum trigger level |
| | Signal repetition rate: 100Hz to 75MHz |

| Auxiliary Inputs and Outputs | |
|---------------------------------------|--|
| External Reference Clock Input | |
| Impedance | 2 K Ω in series with 47 nF |
| Maximum Input Voltage | 15V rms |
| Coupling | AC |
| Sensitivity | 150 mV RMS sine, 450 mV pk-pk pulse |
| Duty Ratio | 40% to 60% |
| Frequency | 10 MHz |
| Connector | Front panel BNC |
| External Arm Input | |
| Input Signal Characteristics | DC Coupled, TTL compatible (1.4 V threshold) |
| Minimum Pulse Width | 15 ns |
| Impedance | 2 K Ω |
| Connector | Front panel DIN |
| External Gate Output | |
| Output Signal Characteristics | DC Coupled, TTL compatible (1.4 V threshold) |
| Connector | Front panel DIN |
| Measurement Functions | |
| Frequency A or B | |
| Range | DC to 225MHz |
| Gate time | GC2200/GTX2200: 250 μ S to 3200S (plus up to one signal period) 0.75 usec of resolution |
| Number of significant digits | GC2220/GC2230/GTX2220/GTX2230: 10 per second of gate time, e.g., 7 digits in 1 ms |
| | GC2210/GTX2210: 8 per second of gate time, e.g., 5 digits in 1 ms |
| Least Significant Digit (LSD) | GTX2220/30: $Freq \times \frac{100 \text{ pS}}{GateTime}$ |
| | GTX2210: $Freq \times \frac{10 \text{ nS}}{GateTime}$ |

| | |
|---|--|
| Resolution (in Hertz) | $\pm LSD \pm \frac{Freq \times (300 \text{ pS rms} + 1.4 \times TriggerError)}{GateTime}$ |
| Accuracy (in Hertz) | $\pm Resolution \pm TimeBaseError$ |
| Fast Frequency A or B (GC2220/GC2230/GTX2220/GTX2230 Only) | |
| Range | DC to 225MHz |
| Gate time | 4 signal periods, fixed |
| Accuracy (in Hertz) | $\pm \frac{Freq \times (500 \text{ pS} + 300 \text{ pS rms} + 1.4 \times TriggerError)}{GateTime} \pm TimeBaseError$ |
| Time Interval A to B or Time Interval B to A | |
| Range | -1 ns to 100,000 seconds (> 25 hr) |
| Least Significant Digit (LSD): | GC2220/GC2230/GTX2220/GTX2230: 100pS |
| | GC2210/GTX2210: 10nS |
| Resolution | $\pm LSD \pm 300 \text{ pS rms} \pm StartTriggerError \pm StopTriggerError$ |
| Accuracy | $\pm Resolution \pm TimeBaseError \pm TriggerLevelTimingError \pm 2 \text{ nS}$ |
| Min. pulse width | 8 ns |
| Delay | Recognition of stop events is inhibited for a set time |
| | GC2200/GTX2200 range: 20µS to 3200 seconds |
| Totalize and Gated Totalize, A or B | |
| Control | Count gate can be controlled by software, or by events on the second input channel (Gated Totalize). In Gated Totalize, start and stop event slopes are selectable |
| Count rate | DC to 50 MHz, 10 ns min. pulse width |
| Modes | Software gate: Gate and count reset are controlled by software |
| | Hardware gate: Count is reset before every gate |
| | Accumulative: Count is totalized over multiple gates |
| Range | 0 to 2.8×10^{14} counts |
| Accuracy | ± 1 count, reading allowed while counting |
| Period A or B (Single Period) | |
| Range | 25nS to 100,000 seconds. See "Time Interval" for resolution and accuracy |

| Period A or B (Multiple Period Average) | |
|--|--|
| Range | See "Frequency" |
| Gate time | See "Frequency" |
| Least Significant Digit (LSD) | GTX2220/30: $Period \times \frac{100 \text{ pS}}{GateTime}$ |
| | GTX2210: $Period \times \frac{10 \text{ nS}}{GateTime}$ |
| Resolution | $\pm LSD \pm \frac{Period \times (300 \text{ pS rms} + 1.4 \times TriggerError)}{GateTime}$ |
| Accuracy | $\pm Resolution \pm TimeBaseError$ |
| Ratio A/B or B/A | |
| Gate time | GC2200/GTX2200: 250µS to 3200 seconds with 0.75 µSec resolution (plus up to one signal period). |
| Range | DC – 225 MHz on either input. DC - 25MHz on second input |
| Least Significant Digit (LSD): | $\frac{Ratio}{FREQ_{hi} \times GateTime}$ FREQ _{hi} = higher frequency input |
| Resolution and accuracy | $\pm LSD \pm \frac{Ratio \times FREQ_{low} TriggerError}{GateTime}$ FREQ _{low} = lower frequency input |
| Width A or B | |
| Accuracy | Same as Time Interval, plus 3 ns |
| Paced Measurement Function | |
| Time controlled | Interval between measurements can programmed from 0.8mS to 3200 seconds for time interval measurements & 1 ms to 3200 seconds for frequency and period measurements. Accuracy: 200 us |
| Maximum Measurement Rate (GC2220/GC2230/GTX2220/GTX2230) | |
| Up to 2300/sec theoretical measurement rate and 1400/sec measurement rate typical or better for Fast Frequency and all time modes. Maximum number of measurements depends on the CPU speed, system configuration, number of applications running simultaneously and instrument settings. | |
| Up to 1400/sec theoretical measurement rate and 1100/sec measurement, rate typical or better for Frequency and Period modes. Maximum number of measurements depends on the CPU speed, system configuration, number of applications running simultaneously, and instrument settings. | |

| | |
|--|--|
| Maximum Measurement Rate (GC2210/GTX2210) | |
| 200/Sec for all modes | |
| Signal Slope | |
| Programmable on Channel A, Channel B, Arm Start, and Arm Stop | |
| Arming | |
| Available on all modes, except Totalize | |
| Source: Internal (alternate channel) or External Input (DIN connector) | |
| Arm setup time: Min. 40nS before selected event | |
| Time Base (PCB revision B) | |
| Standard | 10 MHz crystal oscillator |
| GC2220/GC2230 GTX2220/GTX2230 | Accuracy: ± 5 ppm, 0°C to 50°C |
| | Aging: < 2 ppm/ year |
| | Supply voltage: < 1×10^{-8} for 10% change |
| | Short term: < 5×10^{-10} rms with a 1 second averaging time |
| GC2210/GTX2210 | Accuracy: ± 100 ppm (0.01%), 0°C to 50°C, including the effects of time, temperature and supply voltage |
| Time Base (PCB revision C and above) | |
| Standard | 10 MHz crystal oscillator |
| GC2220/GC2220/GC2230 GTX2210/GTX2220/GTX2230 | Accuracy: ± 1 ppm, 0°C to 50°C |
| | Aging: < 1 ppm/year |
| | Supply voltage: < 3×10^{-8} for $\pm 1\%$ change |
| | Short term: < 5×10^{-10} rms with a 1 second averaging time |
| Optional Time Base | 10MHz Oven Controlled Oscillator |
| GTX2200-OCXO GTX2300-OCXO | Initial tolerance: +/- 100 ppb to final frequency at 1 hour, at 25 C Frequency stability: +/- 100 ppb, 0°C to + 50°C |
| | Aging (1 day): after 72 hour of operation, +/- 30 ppb, max Aging (1 year): after 72 hours of operation, +/- 500 ppb, max Aging (10 years): after 72 hours of operation, +/- 3.0 ppm, max |
| | |
| Reference Source | Internal standard External Reference Input (front panel BNC) PXI 10 MHz from backplane |
| Time Base Output | When installed in the PXI Star Trigger Controller slot, (slot 2), the module's 10 MHz time base can source the PXI 10 MHz backplane clock |

| Supplemental Definitions | |
|---|--|
| Trigger Error | <p>Error due to noise superimposed on the input signal from both internal and external sources</p> $TriggerError = \frac{\sqrt{500\mu V^2 + E_n^2}}{\text{(input signal slew rate)}} \text{ secondsrms}$ <p>E_n = rms noise of input signal (100 MHz bandwidth)</p> |
| Trigger Level Timing Error | <p>Time error due to threshold uncertainty</p> $TriggerLevelTimingError = \frac{< 250mV}{\text{(input signal slew rate)}}$ |
| Time Base Error | Fractional frequency error of time-base reference, times the measurement result |
| Power and Environmental Specifications | |
| Power Requirements | +5 V @ 0.3 Amp (Typical) |
| | +3.3 V @ 0.1 Amp (Typical) |
| | +12 V @ 0.2 Amp (Typical) |
| | -12 V @ 0.05 Amp (Typical) |
| Storage Temperature | -20 to+70°C |
| Operating Temperature | 0 °C to+50°C |
| Size | 3U PXI / PCI |
| Weight | 12 oz. |

Virtual Panel Description

The GXCNT includes a virtual panel program, which provides full access to the various configuration settings and operating modes. To understand the front panel operation, it is best to become familiar with the functionality of the board.

To open the virtual panel application, select **GC2200/GTX2200 Panel** from the **Marvin Test Solutions, GXCNT** menu under the **Start** menu. The GC2200/GTX2200 virtual panel opens as shown here:

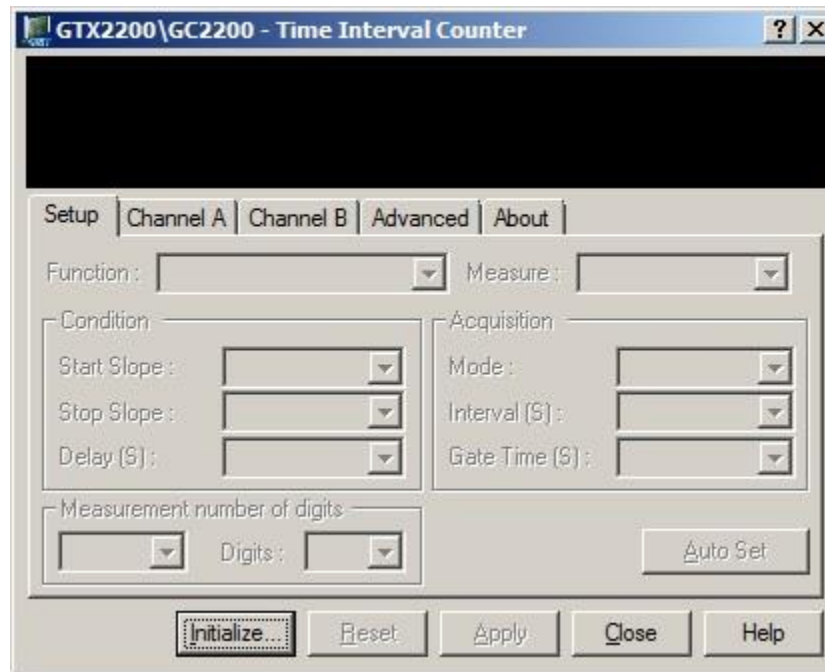


Figure 2-8: GC2200/GTX2200 Virtual Panel (not Initialized)

The function of the panel button controls are shown below:

Initialize - Opens the Initialize Dialog (see Initialize Dialog paragraph) in order to initialize the board driver. The current settings of the selected counter **will not change after calling initialize**. The panel will reflect the current settings of the counter after the Initialize dialog closes.

Reset - resets the PXI board settings to their default state and clears the reading.

Apply – applies changed settings to the board

Close - closes the panel. Closing the panel **does not affect** the counter settings.

Help - opens the on-line help window. In addition to the help menu, the caption shows a **What's This Help** button (?) button. This button can be used to obtain help on any control that is displayed in the panel window. To display the What's This Help information click on the (?) button and then click on the control – a small window will display the information regarding this control.

Virtual Panel Initialize Dialog

The Initialize dialog initializes the driver for the selected counter board. The counter settings **will not change** after initialize is called. Once initialize, the panel will reflect the current settings of the counter.

The Initialize dialog supports two different device drivers that can be used to access and control the board:

Use Marvin Test Solutions's HW – this is the device driver installed by the setup program and is the default driver. When selected, the **Slot Number** list displays the available counter boards installed in the system and their slots. The chassis, slots, devices and their resources are also displayed by the HW resource manager, **PXI/PCI Explorer** applet that can be opened from the Windows Control Panel. The PXI/PCI Explorer can be used to configure the system chassis, controllers, slots and devices. The configuration is saved to PXISYS.INI and PXIeSYS.INI located in the Windows folder. These configuration files are also used by VISA. The following figure shows the slot number 0x105 (chassis 1 Slot 5). This is the slot number argument (*nSlot*) passed by the panel when calling the driver **GxCntInitialize** function used to initialize driver with the specified board.

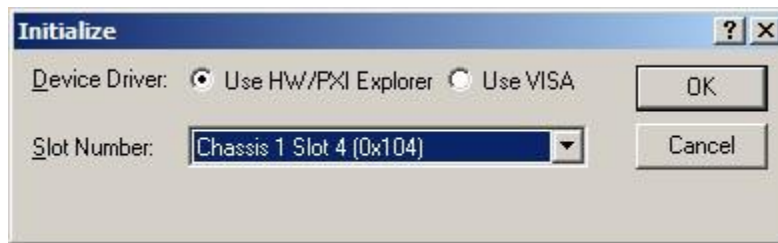


Figure 2-9: Initialize Dialog Box using Marvin Test Solutions' HW driver

Use VISA – this is a third party device driver usually provided by National Instrument (NI-VISA). When selected, the Resource list displays the available boards installed in the system and their VISA resource address. The chassis, slots, devices and their resources are also displayed by the VISA resource manager, Measurement & Automation (NI-MAX) and in Marvin Test Solutions PXI/PCI Explorer. The following figure shows PXI9::13::INSTR as the VISA resource (PCI bus 9 and Device 13). This is VISA resource string argument (*szVisaResource*) passed by the panel when calling the driver **GxCntInitializeVisa** function to initialize the driver with the specified board.



Figure 2-10: Initialize Dialog Box using VISA resources

Virtual Panel Setup Page

After the board is initialized the panel is enabled and will display the current setting of the board. The panel caption will show the board address (0x105 in this example). The following figure shows the **Setup** page settings:



Figure 2-11: GC2200/GTX2200 Virtual Panel (Initialized)

The following controls are shown in the Setup page:

Function Settings

Function: Sets/displays the current counter Function mode.

Measure: Sets/displays the Measure mode. Measure mode updates according to the function mode. Measure mode settings are as follow:

- “Channel A” or “Channel B”: Select the measured channel when Function mode is **Fast Frequency, Frequency, Period, Pulse Width, Single Period** or **Totalize**.
- “A Gated By B” or “B Gated By A”: When Function mode is **Totalize Gated, Totalize Gated Once** or **Accumulate**.
- “A Divided by B” or “B Divided by A”: When Function mode is **Auto Ratio** or **Ratio**.
- “A to B” or “B to A”: When Function mode is **Time Interval** or **Time Interval Delay**.

Condition (Group Box)

Start / Stop Slope: Sets/displays the Start or Stop Slope Polarity. The Slope Polarity can be Positive or Negative. This is enabled only when the Function mode is Accumulate, Totalize Gated or Totalize Gated Once.

Delay (S): Sets/displays the Delay time. The control is enabled when Function mode is Time Interval Delay.

Acquisition Settings

Acquisition Mode: Sets/displays the Acquisition mode. Selections specify how often unarmed measurements should be made.

- **Continuous:** instrument continuously measures.
- **Once:** instrument makes a single measurement and displays the result. **Trig** button clears the last measurement from the display and initiates a new measurement.
- **Paced:** Enables a pacing interval between measurements. (Pacing provides precise control of the spacing between measurements). The current Acquisition interval is shown in the **Interval** dropdown edit box.

Interval (S): Sets/displays the Acquisition Interval time. The Interval setting is either a drop down choice of several intervals or direct value entry.

Gate Time: Sets/displays the gate time. The **Gate Time** sets the minimum measurement time window.

Auto Set: Automatic adjustment of the board setting in order to obtain a stable reading.

Measurement number of digits (Group Box)

Mode (dropdown list):

- **Auto:** Sets/displays the counter automatically sets the number of digits according to the gate time.
- **Fixed:** Sets/displays the measurement's number of digits. Allows the user to override the counter's automatic measure number of digits settings, all measurement will have the same number of digits as was set by the nMaxDigits parameter.

Digits (dropdown list): Number of digits can be 5 to 14.

Virtual Panel Channel A \ Channel B Page

Clicking on **Channel A** / **Channel B** page will show the setting for the specified channel shown in Figure 2-12:

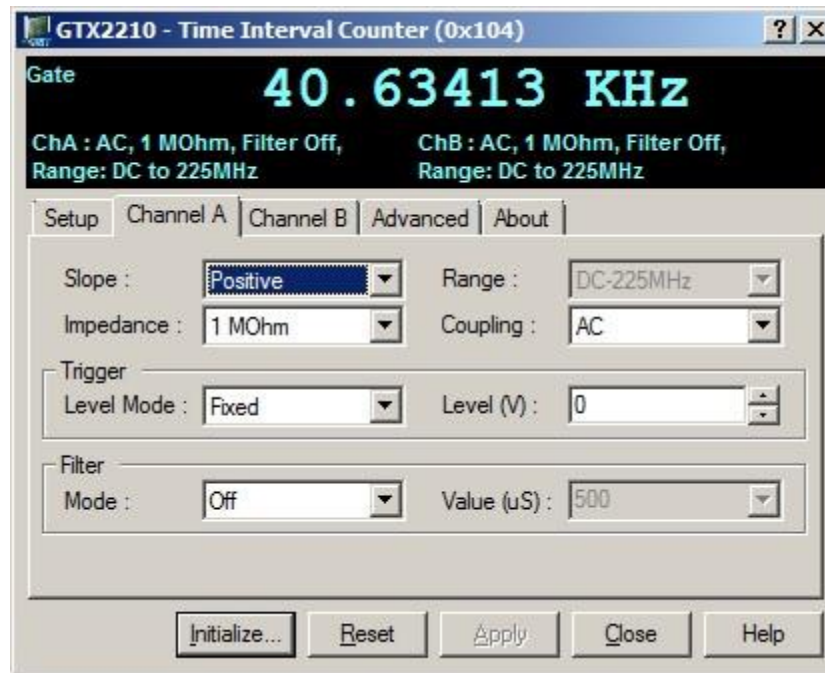


Figure 2-12: GC2200/GTX2200 Virtual Panel (Channel A\Channel B Page)

The following controls are shown in the Channel A/B page:

Slope: Sets/displays the Trigger Slope for the input channel. The input channel slope determines when new measurement starts.

Impedance: Sets/displays the channel input impedance. The two settings are High (1 MOhms) or Low (50 Ohms).

Coupling: Sets/displays the channel coupling. Coupling may be set to DC or AC.

Frequency Range: Sets/displays the channel range. Channel A and Channel B may be set to 0 to 225MHz. Channel A may be set to 0 to 1.3GHz/2GHz range and the panel enables the high frequency option only for GC2220/GC2230/GTX2220/GTX2230 model counters. High Frequency automatically sets the Channel A input impedance to 50 ohms.

Trigger Level Mode: sets or displays the Trigger Level Mode, modes are **Fixed**, **Auto** or **Hold Last**. When the **Trigger Level Mode** is set to **Fixed** the user may set the trigger level manually in the **Trigger Level** edit box.

Trigger Level (V): Sets/displays the input channel trigger level voltage. When **Trigger Level Mode** is set to **Fixed**, the operator can set the trigger level manually (–5.12V to +5.12V). Input voltage will be rounded to the trigger level resolution of 0.04V. Trigger levels are read back from the board after applying new value. **Auto** or **Hold Last** modes disable editing the trigger level but continue to display the current trigger level setting..

Virtual Panel Advanced Page

Clicking on the **Advanced** tab will show the **Advanced** page as shown in Figure 2-13:

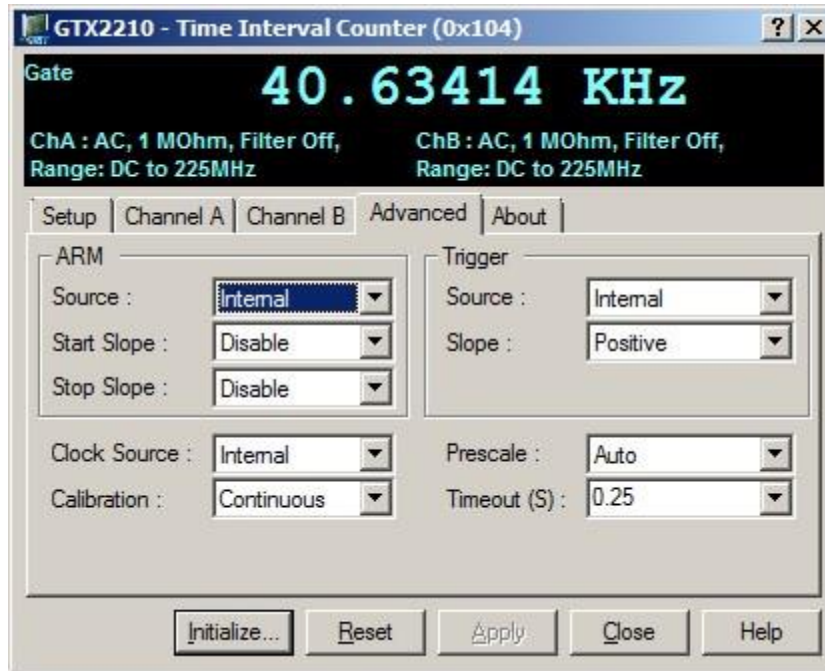


Figure 2-13: GC2200/GTX2200 Virtual Panel – Advanced Page

The following controls are shown in the Advanced page:

Arm Settings:

Source: Sets/displays the Arm source as follow:

- Disable: Measurements are done according to Acquisition mode.
- External: The External Arm Input connector
- Alternate: The Alternate input channel (only for single channel measurements such as **Frequency** or **Period**).

Start/Stop Slope: Sets/displays the Arm Start or Stop Slope as follow:

- Positive Slope: Positive signal transition.
- Negative Slope: Negative signal transition.
- Disable: Disables start and stop arming.

Trigger Settings:

Source: Sets/displays the Trigger source. A trigger source initiates the Acquisition operation of the counter. The available modes are:

- Internal: Acquisition begins immediately after selection or after **Reset**.
- External: Synchronize measurements to external events as defined by the Trigger Slope settings. Selecting this mode enables the Trigger Slope dropdown box.

Slope: Sets/displays the External Trigger Slope settings applied to the ARM input as follow:

- Positive: Acquisition is initiated by the first positive edge.
- Negative: Acquisition is initiated by the first negative edge.

Clock Source: Sets/displays the Clock Source.

- Internal: Selects the internal clock installed on the board.
- External: External clock input as the clock.
- Alternate: When measuring only one channel, the second (or alternate) channel can provide the reference frequency input. Available only with **Frequency, Fast Frequency, Period, Single Period** and **Width** functions.

Calibration Mode: Sets/displays the Calibration Mode.

- Off: The current calibration factors are retained. Stable calibration factors preclude a possible shift in results during a period of data collection.
- Continuous: Calibration is performed continuously.
- Once: Selecting “Once” causes an immediate execution of the calibration procedure, and then disables self-calibration. This mode ensures that data is taken with recent calibration factors, and helps to provide repeatable results.

Prescale Mode: Sets/displays the Prescale Mode.

- Off: Prescaling is disabled.
- Continuous: The counter continuously prescales the actual measurement.
- Auto: The counter determines the need for prescaling automatically by making a quick frequency check prior to the actual measurement. The process takes about 20 μ S.

Prescale Mode: Sets/displays the Prescale Mode.

- Off: Prescaling is disabled.
- Continuous: The counter continuously prescales the actual measurement.
- Auto: The counter determines the need for prescaling automatically by making a quick frequency check prior to the actual measurement. The process takes about 20 μ S.

Timeout (S): Sets/displays the measurement timeout. Setting is either a drop down choice of several values or direct value entry.

Counter Ref Clock To PXI Ref Clock: Sets/displays the connection state of the Counter Reference Clock to the PXI Reference Clock. The PXI chassis has a 10 MHz clock line (PXI_CLK10) in its backplane that allows devices plugged into the non-star controller slots (slots 3 and higher) to phase-lock their oscillators to the backplane clock. A single backplane clock can control this way the drift in the oscillators on all the devices in the non-star controller slots. The GTX22x0 board, when plugged into the star-controller slot (slot 2) can replace the PXI backplane clock with its much more stable oscillator.

Note: Visible only if the GTX22x0 board is plugged into the star-controller slot (slot 2) of the PXI chassis.

Virtual Panel About Page

Clicking on the **About** tab will show the **About** page as shown in Figure 2-14:



Figure 2-14: GC2200/GTX2200 Virtual Panel – About Page

The following controls are shown in the About page:

The top part of the **About** page displays version and copyright of the GXCNT driver. The bottom part displays the board summary. The board summary lists the Boards Type, e.g. GTX2210, PROM version, FPGA Version, Serial number, Oscillator type, Standard or Oven controlled and calibration time for the time base and channels A and B trigger level.

Calibration License... button opens the **License Setup** dialog, see “Chapter 5: In-System Calibration “for details.

In-System Calibration... button opens the in-system calibration dialog. See “Chapter 5: In-System Calibration “for more information.

The **About** page also contains a button **Upgrade Firmware...** used to upgrade the board FPGA. This button maybe used only when the board requires upgrade as directed by Marvin Test Solutions support. The upgrade requires a firmware file (.jam) that is written to the board FPGA. After the upgrade is complete you must shut down the computer to recycle power to the board.

Chapter 3 - Installation and Connections

Getting Started

This section includes general hardware installation procedures for the GC2200/GTX2200 board and installation instructions for the GC2200/GTX2200 - GXCNT software. Before proceeding, please refer to the appropriate chapter to become familiar with the board being installed.

| To Find Information on... | Refer to... |
|-----------------------------|--------------|
| GXCNT Software Installation | This Chapter |
| Hardware/Board Installation | This Chapter |
| Programming | Chapter 4 |
| In-system Calibration | Chapter 5 |
| Software Function Reference | Chapter 6 |

Packing List

All GC2200/GTX2200 boards have the same basic packing list, which includes:

1. GC2200/GTX2200 Board
2. Disk with GXCNT Software

Unpacking and Inspection

After removing the board from the shipping carton:



Caution - Static sensitive devices are present. Ground yourself to discharge static.

1. Remove the board from the static bag by handling only the metal portions.
2. Be sure to check the contents of the shipping carton to verify that all of the items found in it match the packing list.
3. Inspect the board for possible damage. If there is any sign of damage, return the board immediately. Please refer to the warranty information at the beginning of the manual.

System Requirements

The GTX2200 instrument boards are designed for use with a 3U or 6U cPCI or PXI compatible chassis. The software is compatible with any computer system running Windows (32-bit or 64 bit) that was released prior to the release date of this software.

Each GTX2200 board requires one unoccupied 3U PXI bus slot. GC2200 requires one PCI slot.

Installation of the GXCNT Software

Before installing the board, it is recommended to install the software as described in this section:

1. Insert the Marvin Test Solutions CD-ROM and locate the **GXCNT.EXE** setup program. If your computer's Auto Run is configured, when inserting the CD a browser will show several options, select the Marvin Test Solutions Files option, then locate the setup file. If Auto Run is not configured you can open the Windows explorer and locate the setup files (usually located under \Files\Setup folder). You can also download the file from Marvin Test Solutions web site (www.MarvinTest.com).
2. Run the setup and follow the instruction on the Setup screen to install the software.

Note: You may be required to restart the setup after logging-in as a user with Administrator privileges. This is required in-order to upgrade your system with newer Windows components and to install the HW kernel-mode device drivers that are required by the GXCNT driver to access resources on your board.

3. The first setup screen to appear is the Welcome screen. Click **Next** to continue.
4. Enter the folder where the software is to be installed. Either click **Browse** to set up a new folder, or click **Next** to accept the default entry of C:\Program Files\Marvin Test Solutions\GXCNT.
5. Select the type of Setup you wish and click **Next**. You can choose between **Typical**, **Run-Time** and **Custom** setups. **Typical** setup type installs all files. **Run-Time** setup type will install only the files required for controlling the board either from its driver or from its virtual panel. **Custom** setup type lets you select from the available components.

The program will now start its installation. During the installation, Setup may upgrade some of the Windows shared components and files. The Setup may ask you to reboot after it complete if some of the components it replaced were used by another application during the installation – do so before attempting to use the software.

You can now continue with the installation to install the board. After the board installation is complete you can test your installation by starting a panel program that let you control the board interactively. The panel program can be started by selecting it from the Start, Programs, GXCNT menu located in the Windows Taskbar.

Overview of the GXCNT Software

Once the software installed, the following tools and software components are available:

- **PXI/PCI Explorer applet** – use to configure the PXI chassis, controllers and devices. This is required for accurate identification of your PXI instruments later on when installed in your system. The applet configuration is saved to PXISYS.ini and PXIE SYS.ini that are used by Marvin Test Solutions instruments, the VISA provider and VISA based instruments drivers. In addition, the applet can be used to assign chassis numbers, Legacy Slot numbers and instruments alias names.

VISA is a standard maintained by the VXI Plug & Play System Alliance and the PXI Systems Alliance organizations (<http://www.vxipnp.org/>, <http://www.pxisa.org/>). VISA provides a standard way for instrument manufacturers and users to write and use instruments drivers. The VISA resource managers such as National Instruments **Measurement & Automation (NI-MAX)** can display and configure instruments and their address (similar to Marvin Test Solutions's PXI/PCI Explorer).

- **GXCNT Panel** – use to configure, control and display the board settings and counter reading.
- **GXCNT driver** - A DLL based function library (GXCNT.DLL or GXCNT64.DLL, located in the Windows System folder) used to program and control the board. The driver uses Marvin Test Solutions's HW driver or VISA supplied by third party vendor to access and control the GXCNT board.
- **Programming files and examples** – interface files and libraries for various programming tools, see later in this chapter for a complete list of files, programming languages and development tools supported by the driver.
- **Documentation** – On-Line help and User's Guide.

Configuring Your PXI System using the PXI/PCI Explorer

To configure your PXI/PCI system using the **PXI/PCI Explorer** applet follow these steps:

1. **Start the PXI/PCI Explorer applet.** The applet can be start from the Windows Control Panel or from the Windows Start Menu, **Marvin Test Solutions, HW, PXI/PCI Explorer**.
2. **Identify Chassis and Controllers.** After the PXI/PCI Explorer started it will scan your system for changes and will display the current configuration. The PXI/PCI Explorer automatically detects systems that have Marvin Test Solutions controllers and chassis. In addition, the applet detects PXI-MXI-3/4 extenders in your system (manufactured by National Instruments). If your chassis is not shown in the explorer main window, use the Identify Chassis/Controller commands to identify your system. Chassis and Controller manufacturers should provide INI and driver files for their chassis and controllers to be used by these commands.
3. **Change chassis numbers, PXI devices Legacy Slot numbering and PXI devices Alias names.** These are optional steps to be performed if you would like your chassis to have different numbers. Legacy slots numbers are used by older Marvin Test Solutions or VISA drivers. Alias names can provide a way to address a PXI device using your logical name (e.g. "DMM1"). For more information regarding these numbers see the **GxCntInitialize** and **GxCntInitializeVisa** functions.
4. **Save you work.** PXI Explorer saves the configuration to the following files located in the Windows folder: PXISYS.ini, PXIeSYS.ini and GxPxiSys.ini. Click on the **Save** button to save you changes. The PXI/Explorer prompt you to save the changes if changes were made or detected (an asterisk sign ‘ * ‘ in the caption indicated changes).

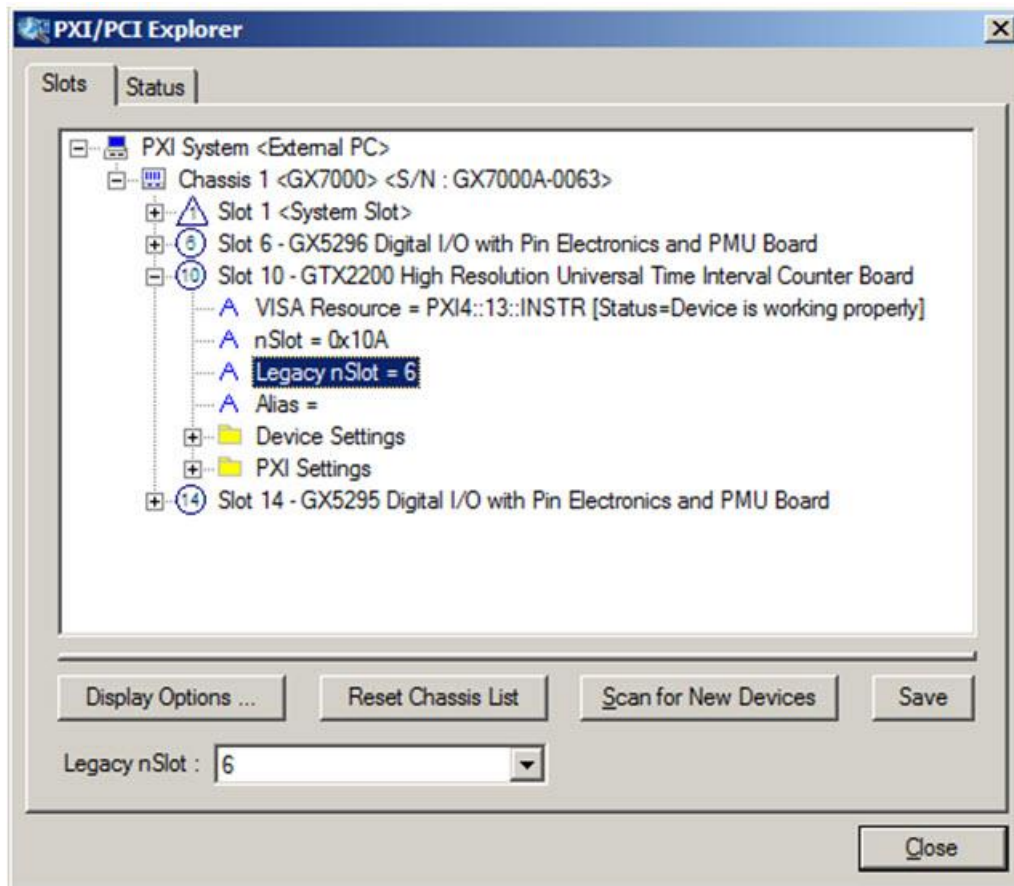


Figure 3-1: PXI/PCI Explorer

Board Installation

Before you Begin

- Install the software driver as described in the prior section.
- Configure your PXI/PC system using **PXI/PCI Explorer** as described in the prior section.
- Verify that all the components listed in the packing list (see previous paragraph) are present.

Electric Static Discharge (ESD) Precautions

To reduce the risk of damage to the board, the following precautions should be observed:

- Leave the board in the anti-static bags until installation requires removal. The anti-static bag protects the board from harmful static electricity.
- Save the anti-static bag in case the board is removed from the computer in the future.
- Carefully unpack and install the board. Do not drop or handle the board roughly.
- Handle the board by the edges. Avoid contact with any components on the circuit board.



Caution - Do not insert or remove any board while the computer is on. Turn off the power from the PXI chassis before installation.

Installing a Board

Install the board as follows:

1. Turn off the PXI chassis and unplug the power cord.
2. Locate a PXI empty slot on the PXI chassis.
3. Place the module edges into the PXI chassis rails (top and bottom).
4. Carefully slide the PXI board to the rear of the chassis, make sure that the ejector handles are pushed **out** (as shown in Figure 3-2).

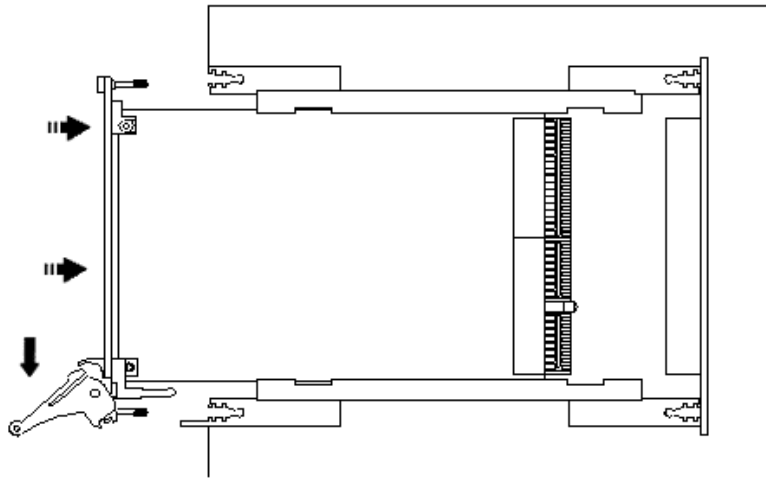


Figure 3-2: Ejector handles position during module insertion

5. After you feel resistance, push in the ejector handles as shown in Figure 3-3 to secure the module into the frame.

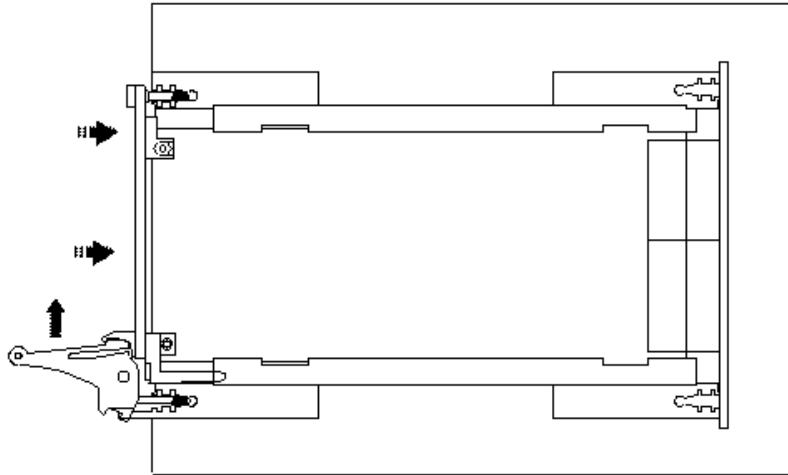


Figure 3-3: Ejector handles position after module insertion

6. Tighten the module's front panel to the chassis to secure the module in.
7. Connect any necessary cables to the board.
8. Plug the power cord in and turn on the PXI chassis.

Plug & Play Driver Installation

Plug & Play operating systems such as Windows notifies the user that a new board was found using the **New Hardware Found** wizard after restarting the system with the new board.

If another Marvin Test Solutions board software package was already installed, Windows will suggest using the driver information file: HW.INF. The file is located in your Program Files\Marvin Test Solutions\HW folder. Click **Next** to confirm and follow the instructions on the screen to complete the driver installation.

If the operating system was unable to find the driver (since the driver was not installed prior to the board installation), you may install the software as described in the prior section, then click on the **Have Disk** button and browse to select the HW.INF file located in C:\Program File\Marvin Test Solutions\HW.

If you are unable to locate the driver click **Cancel** to the found New Hardware wizard and exit the New Hardware Found Wizard, install the GXCNT driver, reboot your computer and repeat this procedure.

The Windows Device Manager (open from the System applet from the Windows Control Panel) must display the proper board name before continuing to use the board software (no Yellow warning icon shown next to device). If the device is displayed with an error you can select it and press delete and then press F5 to rescan the system again and to start the New Hardware Found wizard.

Removing a Board

Remove the board as follows:

1. Turn off the PXI chassis and unplug the power cord.
2. Locate a PXI slot on the PXI chassis.
3. Disconnect and remove any cables/connectors connected to the board.
4. Un-tighten the module's front panel screws to the chassis.
5. Push out the ejector handles and slide the PXI board away from the chassis.
6. Optionally - uninstall the software by running the setup again (or from the Windows Control Pane, Programs and Features or Add Remove Programs applet) and selecting Remove/Uninstall.

Connectors

Figure 3-4 shows the available GTX2200 board connector with description:



Figure 3-4: GTX2200 Front Connectors



Figure 3-5: GC2200 Front Connectors

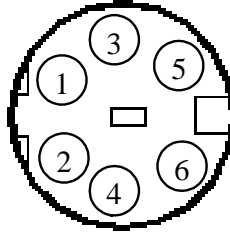


Figure 3-6: GC2200/GTX2200 Front DIN-6 Connector Pin numbers

The following are the connector signal for the GTX2200 board:

| Pin# | Signal |
|-------------|------------------------------------|
| BNC 1 | Input Channel A. |
| BNC 2 | Input Channel B. |
| BNC 3 | External Reference Clock. |
| DIN-6 pin 1 | Arm/Gate Signal Ground. |
| DIN-6 pin 2 | Input Arm Start / Stop TTL signal. |
| DIN-6 pin 3 | Arm/Gate Signal Ground. |
| DIN-6 pin 4 | Output Gate TTL signal. |

Table 3-1: Output Connectors

Connectors and Accessories

The following accessories are available from Marvin Test Solutions for your switching board.

| Part / Model Number | Description |
|---------------------|---|
| GX93005 | DIN Mating Connector for GTX22xx \$20 |
| GX93006 | 3' Harness for GTX22xx DIN connector (DIN to Header) \$65 |
| GX92012 | Cable, BNC Male to BNC Male, 50 Ohm, 2' \$30 |
| GX92015 | Cable, BNC Male to BNC Male, 50 Ohm, 5' |

Table 3-2: Spare Connectors and Accessories

Installation Folders

The GXCNT driver files are installed in the default directory C:\Program Files\Marvin Test Solutions\GXCNT. You can change the default GXCNT directory to one of your choosing at the time of installation.

During the installation, GXCNT Setup creates and copies files to the following directories:

| Name | Purpose / Contents |
|---------------------------------|--|
| ...\Marvin Test Solutions\GXCNT | The GXCNT directory. Contains panel programs, programming libraries, interface files and examples, on-line help files and other documentation. |
| ...\Marvin Test Solutions\HW | HW device driver. Provide access to your board hardware resources such as memory, IO ports and PCI board configuration. See the README.TXT located in this directory for more information. |
| ...\ATEasy\Drivers | ATEasy drivers directory. GXCNT Driver and example are copied to this directory only if <i>ATEasy</i> is installed to your machine. |
| Windows System Folders | Windows System directory. Contains the GXCNT.DLL or GXCNT64.DLL driver HW driver shared files and some upgraded system components, such as the HTML help viewer, etc. |

GXCNT Driver Files Description

The Setup program copies the GXCNT driver, a panel executable, the GXCNT help file, the README.TXT file, and driver samples. The following is a brief description of each installation file:

Driver File and Virtual Panel

- GXCNT.DLL and GXCNT64.DLL - 32/64 bit Windows DLLs for 32/64 bit applications running under Windows.
- GXCNTPANEL.EXE and GXCNTPANEL64.EXE – An instrument front panel program for all GXCNT supported boards.

Interface Files

The following GXCNT interface files are used to support the various development tools:

- GXCNT.H - header file for accessing the DLL functions using the C/C++ programming language. The header file compatible with the following 32 bit development tools:
 - Microsoft Visual C++, Microsoft Visual C++ .NET
 - Borland C++
- GXCNT.LIB and GXCNT64.LIB - Import library for GXCNT.DLL and GXCNT64.DLL (used when linking C/C++ application).
- GXCNTBC.LIB - Import library for GXCNT.DLL (used when linking Borland C/C++ application that uses GXCNT.DLL).
- GXCNT.PAS - interface file to support Borland Pascal Borland Delphi.
- GXCNT.BAS - Supports Microsoft Visual Basic 4.0, 5.0 and 6.0.
- GXCNT.VB - Supports Microsoft Visual Basic .NET.

- GTX2200.drv - ATEasy driver File for GTX2200 and GXCNT Virtual Panel Program.
- GC2200.drv - ATEasy driver File for GC2200 and GXCNT Virtual Panel Program.
- GxCnt.llb – LabView library.

On-line Help and Manual

GXCNT.CHM – On-line version of the GXCNT User’s Guide. The help file is provided in a Windows Compiled HTML help file (.CHM). The file contains information about the GC2200/GTX2200 board, programming reference and panel operation.

GXCNT.PDF – On line, printable version of the GXCNT User’s Guide in Adobe Acrobat format. To view or print the file you must have the reader installed. If not, you can download the Adobe Acrobat reader (free) from <http://www.adobe.com>.

ReadMe File

README.TXT – Contains important last minute information not available when the manual was printed. This text file covers topics such as a list of files required for installation, additional technical notes, and corrections to the GXCNT manuals. You can view and/or print this file using the Windows NOTEPAD.EXE or any other text file editors.

Example Programs

The sample program includes a C/C++ sample compiled with various development tools, Visual Basic example and an ATEasy sample. Other examples may be available for other programming tools.

Microsoft Visual C++ .NET example files:

- GxCntExampleC.cpp - Source file
- GxCntExampleC.ico - Icon file
- GxCntExampleC.rc - Resource file
- GxCntExampleC.vcproj - VC++ .NET project file
- GxCntExampleC.exe - 32 bit example executable
- GxCntExampleC64.exe - 64 bit example executable

Microsoft Visual C++ 6.0 example files:

- GxCntExampleC.cpp - Source file
- GxCntExampleC.ico - Icon file
- GxCntExampleC.rc - Resource file
- GxCntExampleC.dsp - VC++ project file
- GxCntExampleC.exe - Example executable

Borland C++ example files:

- GxCntExampleC.cpp - Source file
- GxCntExampleC.ico - Icon file
- GxCntExampleC.rc - Resource file
- GxCntExampleC.bpr - Borland project file
- GxCntExampleC.exe - Example executable

Microsoft Visual Basic .NET example files:

- GxCntExampleVB.vb - Example form.

- GxCntExampleVB.resx - Example form resource.
- GxCntExampleVBapp.config - Example application configuration file.
- GxCntExampleVBAssemblyInfo.vb - Example application assembly file
- GxCntExampleVB.vbproj - Project file
- GxCntExampleVB.exe - Example executable

Microsoft Visual C# example files:

- GxCntExampleCS.cs - Example source
- GxCntExampleCS.csproj - Project file
- GxCntExampleCS.exe - Example executable

Microsoft Visual Basic 6.0 example files:

- GxCntExampleVB6.frm - Example form
- GxCntExampleVB6.frx - Example form binary file
- GxCntExampleVB6.vbp - Project file
- GxCntExampleVB6.exe - Example executable.

ATEasy driver and examples files (ATEasy Drivers directory):

- GC2200.prj - example project
- GC2200.sys - example system
- GC2200.prg - example program
- GTX2200.prj - example project
- GTX2200.sys - example system
- GTX2200.prg - example program

Setup Maintenance Program

You can run the Setup again after GXCNT has been installed from the original disk or from the Windows Control Panel – Add Remove Programs applet. Setup will be in the Maintenance mode when running for the second time. The Maintenance window show below allows you to modify the current GXCNT installation. The following options are available in Maintenance mode:

- **Modify.** When you want to add or remove GXCNT components.
- **Repair.** When you have corrupted files and need to reinstall.
- **Remove.** When you want to completely remove GXCNT.

Select one of the options and click **Next**.

Follow the instruction on the screen until Setup is complete.

Chapter 4 - Programming the Board

This chapter contains information about how to program the PXI Universal Time Interval Counters using the GXCNT driver. The driver contains functions to initialize, reset, and control the PXI counters. This chapter includes a brief description of the functions, as well as how and when to use them, with code examples. Chapter 5 contains a complete and detailed description of the available programming functions.

The driver supports many development tools. This chapter describes using these tools with the driver. The GXCNT installation directory contains examples written for these development tools. Chapter 3 lists the available examples.

An example using the DLL driver with Microsoft Visual C++ is at the end of this chapter. The driver functions and parameters are identical for all operating systems and development tools, and this example serves as a guide for use other programming languages.

The GXCNT Driver

The driver is a 32-bit Windows DLL file: GXCNT.DLL and 64-bit DLL: GXCNT64.DLL. The DLL is used with 32-bit or a 64-bit applications running under Windows. The HW device driver is installed by the setup program and is shared by other Marvin Test Solutions products (ATEasy, GXPIO, GXSW, etc). The DLLs can also use VISA (provided by a third party) to access the board hardware instead of the provided HW driver.

The DLL can be used with various development tools such as Microsoft Visual C++, Borland C++ Builder, Microsoft Visual Basic, Borland Pascal or Delphi, ATEasy and more. The following paragraphs describe how to create an application that uses the driver with various development tools. Refer to the paragraph describing the specific development tool for more information.

Programming Using C/C++ Tools

The following steps are required to use the GXCNT driver with C/C++ development tools:

- Include the GXCNT.H header file in the C/C++ source file that uses the GC2200/GTX2200 function. This header file is used for all driver functions. The file contains function prototypes and constant declarations to be used by the compiler for the application.
- Add the required .LIB file to the projects. This can be the import library GXCNT.LIB for Microsoft Visual C++ for 32-bit applications, GXCNT64.LIB for 64-bit applications and GXCNTBC.LIB for Borland C++. Windows- based applications that explicitly load the DLL by calling the Windows **LoadLibrary** API should not include the .LIB file in the project.
- Add code to call the GXCNT library functions as required by the application.
- Build the project.
- Run, test, and debug the application.

Programming Using Visual Basic

To use the driver with Visual Basic 4.0, 5.0 or 6.0 (for 32-bit applications), the user must include the GXCNT.BAS to the project. Visual Basic .NET developers must use the GXCNT.VB.

The file can be loaded within the integrated development environment (IDE) using *Add File* from the Visual Basic IDE *File menu*. The GXCNT.BAS/.VB contains function declarations for the DLL driver.

Programming Using Pascal/Delphi

To use the driver with Borland Pascal or Delphi, the user must include the GXCNT.PAS to the project. The GXCNT.PAS file contains a **unit** with function prototypes for the DLL functions. Include the GXCNT unit in the **uses** statement before making calls to the GXCNT functions.

Programming Using ATEasy®

The GXCNT package supplies a separate ATEasy driver for the PXI counters. The ATEasy driver uses the same GXCNT.DLL to program the board for all development environments. In addition, there is an example project that contains a program and a system file pre-configured with the ATEasy driver. Use the property dialog to change the driver shortcut property in the System Drivers sub-module and change the PCI slot number to reflect your current installation before running the example.

The ATEasy driver plain-languages commands are easier to use than using the DLL functions directly. The driver commands will generate an exception if the function call fails. That allows the ATEasy application to trap errors without directly checking the status code returned by the DLL function. The driver commands check the status parameter after each function call.

The ATEasy driver contains commands that are similar to the DLL functions in name and parameters, with the following exceptions:

- The driver handles the *nHandle* parameter automatically. ATEasy handles board instances as driver logical names i.e. CNT1, CNT2 for GC2200/GTX2200. Direct access to the parameter is possible using the property operator (the “dot” operator) once the parameter is set public in the ATEasy driver.
- The driver handles the *nStatus* parameter automatically. The Get Status command in the ATEasy Driver provides access to the board's status. After calling a DLL function, the ATEasy driver will check the returned status and will call the CheckError procedure. An error status generates an exception that can be easily trapped by the application using the **OnError** module event or using the **try-catch** statement. ATEasy will notify the user or developer with an error dialog if the condition is not handled by other ATEasy code. Direct access to the parameter is possible using the property operator (the “dot” operator) once the parameter is set public in the ATEasy driver.

Some ATEasy drivers contain additional commands to permit easier access to the board features. For example, parameters for a function may be omitted by using a command item instead of typing the parameter value. The plain-language commands are self-documenting. Their syntax is similar to an English language statement. In addition, you can use the commands from the code editor context menu or by using the ATEasy's code completion feature instead of typing them directly.

Programming Using LabView and LabView/Real Time

To use the driver with LabView use the provided lab view library GXCNT.llb. The library is located in the GXCNT folder. An example for LabView is also provided in the Examples folder. A DLL located in the LabViewRT folder can be used for deployment with LabView/Real-Time.

Using and Programming under Linux

Marvin Test Solutions provides a separate software package with a Linux driver (Marvin Test Solutions Drivers Pack for Linux). The software package can be download from the Marvin Test Solutions website. See the ReadMe.txt in that package for more information regarding using and programming the driver under Linux.

Using the GXCNT driver functions

The GXCNT driver contains a set of functions for the GC2200/GX2200 family of counters. Functions names that starts with the **GxCnt** prefix applies to all GXCNT boards (i.e. **GxCntReset**). The GXCNT functions are designed with consistent set of arguments and functionality. All boards have a function that initializes the GXCNT driver for a specific board, reset the board, and display the virtual panel. All the functions use handles to identify and reference a specific board and all functions return status and share the same functions to handle error codes.

Initialization, HW Slot Numbers and VISA Resource

The GXCNT driver supports two device drivers HW and VISA, which are used to initialize, identify and control the board. The user can use the **GxCntInitialize** to initialize the board 's driver using HW and **GxCntInitializeVisa** to initialize using VISA. The following describes the two different methods used:

1. **Marvin Test Solutions's HW** - the default device driver that is installed by the GXCNT driver. To initialize and control the board using the HW use the **GxCntInitialize**(*nSlot*, *pnHandle*, *pnStatus*) function. The function initializes the driver for the board at the specified PXI slot number (*nSlot*) and returns a board handle. The **PXI/PCI Explorer** applet in the Windows Control Panel displays the PXI slot assignments. You can specify the *nSlot* parameter in the following way:
 - A combination of chassis number (chassis # x 256) with the chassis slot number, e.g. 0x105 for chassis 1 and slot 5. Chassis number can be set by the **PXI/PCI Explorer** applet.
 - Legacy nSlot as used by earlier versions of HW/VISA. The slot number contains no chassis number and can be changed using the **PXI/PCI Explorer** applet: 23 in this example.

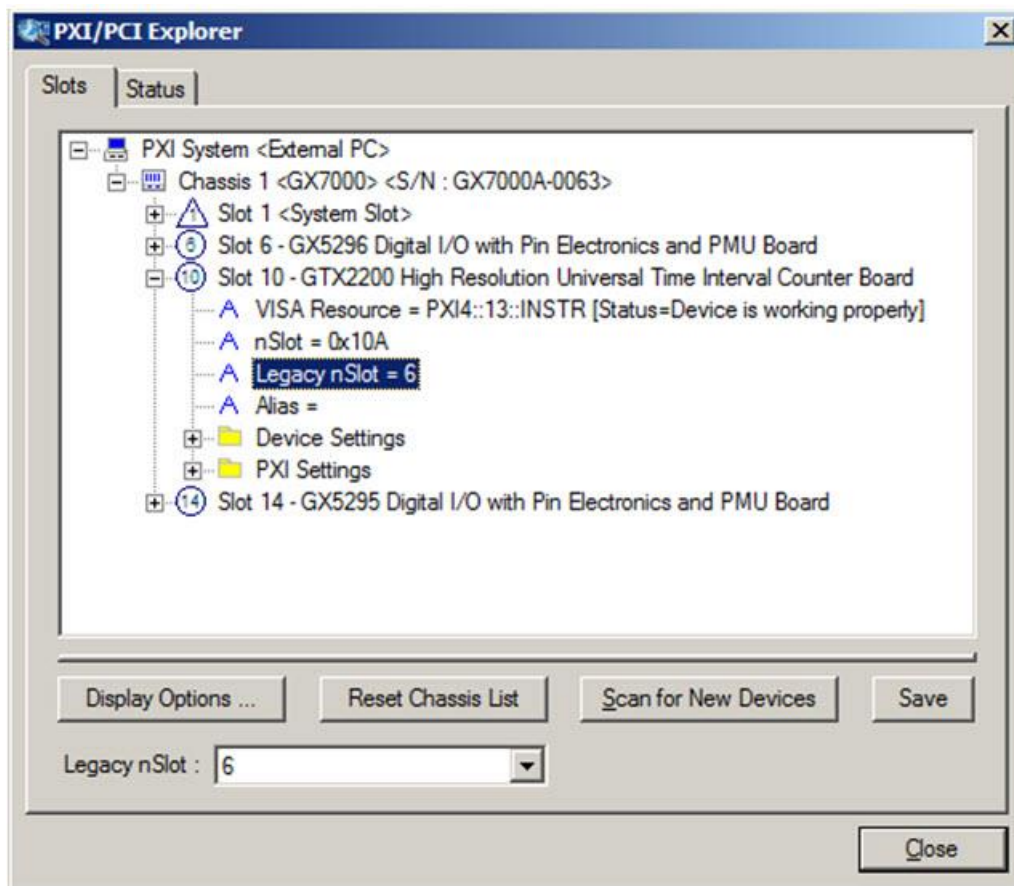


Figure 4-1: PXI/PCI Explorer

2. **VISA** – this is a third party library usually by National Instruments (NI-VISA). You must ensure that the VISA installed supports PXI and PCI devices (not all VISA providers supports PXI/PCI). GXCNT setup installs a VISA compatible driver for the GXCNT board in-order to be recognized by the VISA provider. Use the GXCNT function **GxCntInitializeVisa** (*szVisaResource*, *pnHandle*, *pnStatus*) to initialize the driver board using VISA. The first argument *szVisaResource* is a string that is displayed by the VISA resource manager such as **NI Measurement and Automation (NI_MAX)**. It is also displayed by Marvin Test Solutions **PXI/PCI Explorer** as shown in the prior figure. The VISA resource string can be specified in several ways as the following examples:

- Using chassis, slot: “PXI0::CHASSIS1::SLOT5”
- Using the PCI Bus/Device combination: “PXI9::13::INSTR” (bus 9, device 9).
- Using alias: “COUNTER1”. Use the PXI/PCI Explorer to set the device alias.

Information about VISA is available at <http://www.pxisa.org>.

The **GxCntInitialize** function returns a handle that is required with other driver functions to program the board. This handle is usually saved in the program in a global variable for later use when calling other functions. The initialize function does not change the state of the board or its settings.

Board Handle

The board handle argument, *nHandle*, passed (by reference) to the parameter *pnHandle* of the **GxCntInitialize** or the **GxCntInitializeVisa** functions is a short integer (16 bits) number. It is used by the GXCNT driver functions to identify the board being accessed by the application. Since the driver supports many boards at the same time, the *nHandle* argument is required to uniquely identify which board is being programmed.

The *nHandle* is created when the application calls the **GxCntInitialize** function. There is no need to destroy the handle. Calling **GxCntInitialize** with the same slot number will return the same handle.

Once the board is initialized the handle can be used with other functions to program the board.

Reset

The Reset function causes the driver to change all settings to their default state. The application software issue a Reset after the initializing the Counter, but a Reset can be issued any time. All counter boards have the **GxCntReset**(*nHandle*, *nStatus*) function. See the Function Reference for more information regarding the specific board.

Error Handling

All GXCNT functions pass a fail or success status - *pnStatus* - in the last parameter. A successful function call passes zero in the status parameter upon return. If the status is non-zero, then the function call fails. This parameter can be later used for error handling. When the status is error, the program can call the **GxCntGetErrorString** function to return a string representing the error. The **GxCntGetErrorString** reference contains possible error numbers and their associated error strings.

Driver Version

The **GxCntGetDriverSummary** function can be used to return the current GXCNT driver version. It can be used to differentiate between the driver versions. See the Function Reference for more information.

Panel

Calling the **GxCntPanel** will display the instrument's front panel dialog window. The panel can be used to initialize and control the board interactively. The panel function may be used by the application to allow the user to directly interact with the board.

The **GxCntPanel** function is also used by the GXCNTPANEL.EXE. or GXCNTPANEL64.EXE. panel program that is supplied with this package and provides a stand-alone Windows application that displays the instrument panel.

Distributing the Driver

Once the application is developed, the driver files (GXCNT.DLL or GXCNT64.DLL) and the HW device driver files located in the HW folder) can be shipped with the application. Typically, the DLLs should be copied to the Windows System directory. The HW device driver files should be installed using a special setup program HWSETUP.EXE that is provided with GXCNT driver files. Alternatively, you can provide the GXCNT disk to be installed along with the board.

Sample Programs

The following example demonstrates how to program the board using the C programming language under Windows. The example shows how to initialize the counter, set it up for measurement or trigger settings and get the reading.

To run, enter the following command line:

```
GxCntExample <Slot> <Channel|Measure> <Operation> <Mode|Level>
```

Where:

| | |
|-------------------|--|
| <Slot> | PXI Explorer slot number where the board reside. |
| <Channel Measure> | Channel setting or measurement: SCHA= set channel A function, trigger level or trigger mode SCHB= set channel B function, trigger level or trigger mode MEASURE=Read single measurement |
| <Operation> | Operation code: FREQ=Sets measurement function to frequency PERIOD=Sets measurement function to period TRIG_MODE=Sets the trigger level mode TRIG_LEVEL=Sets the trigger level |
| <Mode Level> | Trigger mode or level depends on the operation: Mode: 0=trigger level set by user 1=trigger level set automatically 2=trigger level uses the last value Level: voltage level (-5.12v to +5.12v) |

Sample Program Listing

```

/*****
FILE           : GxCntExampleC.cpp
PURPOSE        : WIN32/LINUX example program for GX2200 boards
                  using the GXCNT driver.
CREATED        : Mar 2002-2011
COPYRIGHT      : Copyright 2002-2014 MARVIN TEST SOLUTIONS.
COMMENTS       :
To compile the example:

1. Microsoft VC++
Load GxCntExampleC.dsp, .vcproj or .mak, depends on
the VC++ version from the Project\File/Open... menu
Select Project/Rebuild all from the menu

2. Borland C++ Builder
Load GxCntExampleC.bpr from the Project/Open
Project... menu
Select Project/Build all from the menu

3. Linux (GCC for CPP and Make must be available)
make -fGxCntExampleC.mk [CFG=Release[64] | Debug[64]] [rebuild | clean]
*****/

#ifndef __GNUC__
#include "windows.h"
#endif
#include "GxCnt.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#ifdef __BORLANDC__
#pragma hdrstop
#include <condefs.h>
USELIB("GxCntBC.lib");
USERC("GxCntExampleC.rc");
#endif

/*****
//
//           DisplayMsg
//
void DisplayMsg(PCSTR lpszMsg)
{
#ifndef __GNUC__
    MessageBeep(0);
    MessageBox(0, lpszMsg, "GxCnt example program", MB_OK);
#else
    printf("\r\nGxCnt example program: %s\r\n", lpszMsg);
#endif
    return;
}

*****/

```

```

//__strupr
//*****
char * __strupr(char * sz)
{
    int i;

    for (i=0; sz[i]; i++)
        sz[i] = toupper(sz[i]);
    return sz;
}

//*****
//          DisplayUsage
//*****
void DisplayUsage(void)
{
    DisplayMsg(
        "This example shows how to use the GX2200:\r\n"

        "Usage:\r\n"
        "GxCntExampleC <slot|address> <channel|measure|summary>
        <operation> <mode|level>"

        "\r\n\r\nWhere : "
        "<slot> - Under Windows: PCI/PXI slot number as shown by the PXI
        explorer\r\n"
        "<address> - Under Linux: Bus/device as displayed with lspci
        utility\r\n"

        "<channel|measure|summary>:\r\n"
        "\tSCHA=set channel A function, trigger level or trigger
        mode\r\n"
        "\tSCHB=set channel B function, trigger level or trigger
        mode\r\n"
        "\tMEASURE=Read single measurment\r\n"
        "\tSUM=Print board summary\r\n"

        "<operation> - one of the followings:\r\n"
        "\tFREQ=Sets measurment function to frequency\r\n"
        "\tPERIOD=Sets measurment function to period\r\n"
        "\tTRIG_MODE=Sets the trigger level mode\r\n"
        "\tTRIG_LEVEL=Sets the trigger level\r\n"

        "<mode|level>- trigger mode or level:\r\n"
        "\t<mode>:\t0=trigger level set by user\r\n"
        "\t\t1=trigger level set automatically\r\n"
        "\t\t2=trigger level uses the last value\r\n"
        "\t<level>:\tvoltage level (-5.12v to +5.12v)\r\n"

        "\r\nThe example should be run from the Windows command
        prompt/Linux terminal.\r\n"
        "\tExamples:\r\n"
        "\t./GxCntExampleC 7 SCHA FREQ 1\r\n"
        "\t./GxCntExampleC 7 MEASURE (under Windows, slot 7)\r\n"
        "\t./GxCntExampleC 0x60c MEASURE (under Linux, bus 6 device 12)\r\n"
    );
    exit(1);
}

```

```

}

//*****
//          CheckStatus
//*****
void CheckStatus(SHORT nStatus)
{
    char    sz[512];

    if (!nStatus) return;
    GxCntGetErrorString(nStatus, sz, sizeof sz, &nStatus);
    DisplayMsg(sz);
    DisplayMsg("Aborting the program...");
    exit(nStatus);
}

//*****
//          MAIN
//
// This main function receives the following parameters
//
// GX2200 GXCNT board slot number (e.g. 1)
// <channel|measure>:
//     SCHA=set channel A function, trigger level or mode
//     SCHB=set channel B function, trigger level or mode
//     MEASURE=Read single measurment
//     SUM=print board summary
//
// <operation> - one of the followings:
//     FREQ=Sets measurement function to frequency
//     PERIOD=Sets measurement function to period
//     TRIG_MODE=Sets the trigger level mode
//     TRIG_LEVEL=Sets the trigger level
//
// <mode|level>- trigger mode or level:\r\n"
//     <mode>:      0=trigger level set by user\r\n"
//                 1=trigger level set automatically\r\n"
//                 2=trigger level uses the last value\r\n"
//     <level>: voltage level (-5.12v to +5.12v)\r\n"
//
//*****
int main(int argc, char ** argv)
{
    short nSlotNum;           // Board slot number
    char * szChannelOrMeasure; // Channel or Measure operation
    char * szOperation;      // Board Operation
    short nHandle;           // Board handle
    short nStatus;           // Returned status
    short nChannel;          // Channel number
    short nTriggerMode;      // Trigger Mode
    short nReadTriggerMode;  // Read Trigger Mode
    double dTriggerLevel;    // Trigger Level Voltage
    double dMeasure;         // Measurement
    char    sz[512];         // Board Summary

    // Check minimum number of arguments recived
    if (argc<3) DisplayUsage();

```

```

// Parse command line parameters
nSlotNum=(SHORT)strtol(*(++argv), NULL, 0);
if (nSlotNum<0) DisplayUsage();
GxCntInitialize(nSlotNum, &nHandle, &nStatus);
CheckStatus(nStatus);

szChannelOrMeasure=__strupr(*(++argv));
if (!strcmp(szChannelOrMeasure, "SCHA") || !strcmp(szChannelOrMeasure,
"SCHB"))
{
    // Check number of required arguments recived
    if (argc<4)
        DisplayUsage();
    // Get Channel number
    nChannel=strcmp(szChannelOrMeasure, "SCHA")==0 ? GXCNT_CHANNEL_A :
        GXCNT_CHANNEL_B;
    szOperation=__strupr(*(++argv));
    if (!strcmp(szOperation, "FREQ"))
    {
        GxCntSetFunctionFrequency(nHandle, nChannel, &nStatus);
        CheckStatus(nStatus);
        printf("Set channel %c function to frequeuncy\n", nChannel+65);
    }
    else if (!strcmp(szOperation, "PERIOD"))
    {
        GxCntSetFunctionPeriod(nHandle, nChannel, &nStatus);
        CheckStatus(nStatus);
        printf("Set channel %c function to period\n", nChannel+65);
    }
    else if (!strcmp(szOperation, "TRIG_MODE"))
    {
        if (argc<5) DisplayUsage();
        nTriggerMode=(SHORT)strtol(*(++argv), NULL, 0);
        if (nTriggerMode<0 || nTriggerMode>2)
            DisplayUsage();
        GxCntSetChannelTriggerLevelMode(nHandle, nChannel, nTriggerMode,
            &nStatus);
        CheckStatus(nStatus);
        // Verify settings
        GxCntGetChannelTriggerLevelMode(nHandle, nChannel,
            &nReadTriggerMode, &nStatus);
        CheckStatus(nStatus);
        printf("successfully set channel %c Trigger Level Mode\n",
            nChannel+65);
    }
    else if (!strcmp(szOperation, "TRIG_LEVEL"))
    {
        dTriggerLevel=strtod(*(++argv), NULL);
        GxCntSetChannelTriggerLevel(nHandle, nChannel, dTriggerLevel,
            &nStatus);
        CheckStatus(nStatus);
        // verify settings
        GxCntGetChannelTriggerLevel(nHandle, nChannel, &dTriggerLevel,
            &nStatus);
        CheckStatus(nStatus);
        printf("Channel %c Trigger Level voltage is %f\n", nChannel+65,
            dTriggerLevel);
    }
}

```

```
        else
            DisplayUsage();
    }

    else if (!strcmp(szChannelOrMeasure, "MEASURE"))
    { // read set channel rail
        GxCntReadMeasurement(nHandle, &dMeasure, &nStatus);
        CheckStatus(nStatus);
        printf("Measurement: %f\n", dMeasure);
    }

    else if (!strcmp(szChannelOrMeasure, "SUM"))
    { // print board summary
        GxCntGetBoardSummary(nHandle, sz, sizeof sz, &nStatus);
        CheckStatus(nStatus);
        printf("Board Summary: %s.\n", sz);
    }

    else
        DisplayUsage();

    return 0;
}

//*****
//          End Of File
//*****
```

Chapter 5 - In-System Calibration

Introduction

Marvin Test Solutions offers Automatic In-System Calibration software for counter boards with PCB revision C and above. The In-System Calibration requires purchasing a license setup from Marvin Test Solutions. The Automatic In-System Calibration can be performed through the virtual panel or through the driver function calls. The support for the Automatic In-System Calibration needs to be activated by running the **GXNT-CAL** license setup in order to be used by the virtual panel and the driver functions.

The calibration program provides an easy-to-use Automatic In-System Calibration procedure, which uses a calibration front panel or Driver function calls. This allows for calibration without the need to remove the counter board from a rack or system. The calibration virtual front panel provides an interactive step-by-step process that ensures proper and accurate calibration.



Note – It is recommended to calibrate the counted board every 12 months.

The following provides information regarding:

- Required instrument for the calibration.
- Installation of the **GxCntCalibration** license setup.
- Running calibration from the virtual panel.
- Calibration Example program (running calibration using API calls).

Required instrument

In order to calibrate the counter board a reference signal of 10MHz \pm 0.01PPM sine wave with amplitude of 1 ± 0.3 Vrms into 50 ohms and less than -35dB total harmonics distortions.

Calibration Interval

The instrument should be calibrated on a regular interval determined by the measurement accuracy requirements of your application. A 1-year interval is adequate for most applications. Accuracy specifications are warranted only if adjustment is made at regular calibration intervals. Accuracy specifications are not warranted beyond the 1-year calibration interval. Marvin Test Solutions does not recommend extending calibration intervals beyond 2 years for any application.

In-System Calibration License Setup

The **GxCnt Calibration** license string needs to be installed in order to activate and enable calibration option through the virtual front panel or through the driver function calls. The following procedure describes how to setup the calibration license:

1. Open the Panel application, **About** page:

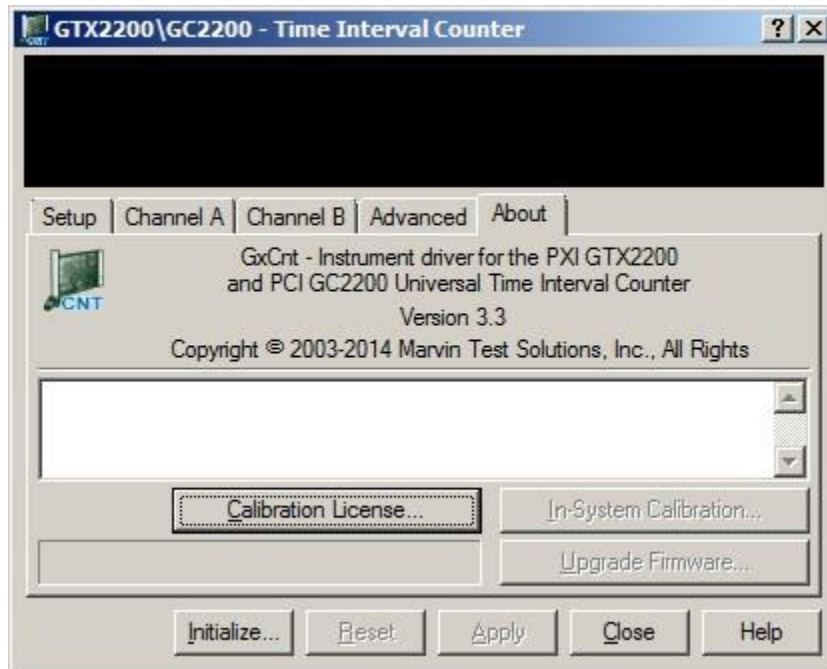


Figure 5-1: GC2200/GTX2200 Virtual Panel About Page

2. Click on the **Calibration License...** button to open the License Setup dialog:



Figure 5-2: GXCNT-CAL License Setup Dialog

3. Request a license from Marvin Test Solutions. You may need to create or log into your magic account at <http://www.MarvinTest.com/magic>. Then create an incident requesting a license specifying your product serial number (as it appears on your purchase order or packing list) and the Computer Id shown in the License Setup dialog (in this example "C338 ECEE 05").

- Once you receive the license string (by email), enter it to the edit box in the **License Setup** dialog along with your name and organization and click **OK**.

You can now test your installation by starting a panel program that let you control the board interactively. The panel program can be started by selecting it from the **Start, Programs, Marvin Test Solutions and GXCNT** menu located in the Windows Taskbar. In the virtual panel click the “Initialize” button, then click on the **About** tab. The **Calibration By User...** button should be enabled.

Running In-System calibration from the virtual panel

The GXCNT includes a virtual panel program, which provides full access to the various configuration settings and operating modes. To open the virtual panel application, select **GC2200/GTX2200 Panel** from the **Marvin Test Solutions, GXCNT** menu under the **Start** menu. The GC2200/GTX2200 virtual panel opens as shown here:

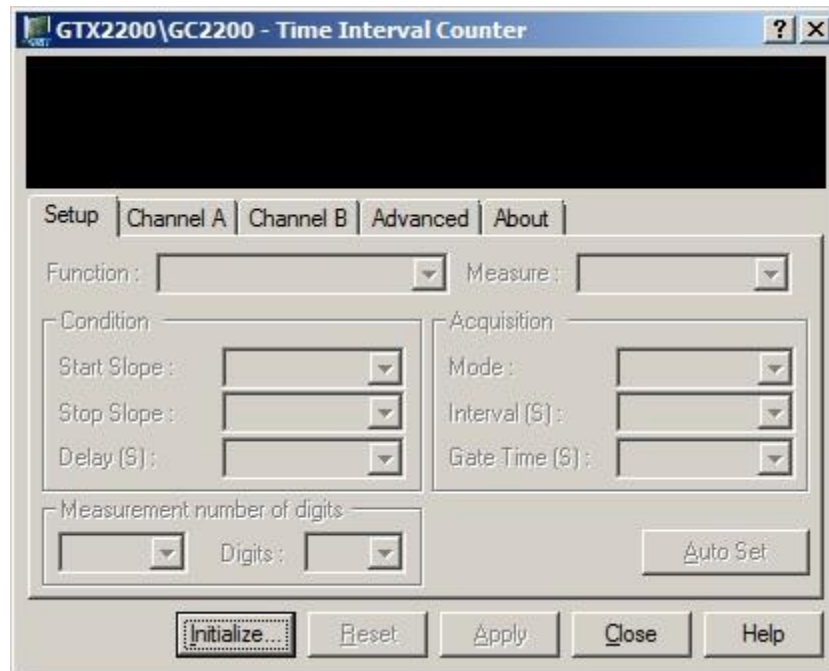


Figure 5-3: GC2200/GTX2200 Virtual Panel (not Initialized)

The function of the panel button controls are shown below:

Initialize allows the user to associate the front panel with a specific PXI counter. The settings of the chosen counter **will not change**. Then panel will reflect the current settings of the counter after the Initialize dialog closes.

The Initialize button opens an Initialize dialog window. The user may select the slot where the PXI slot location for a PXI counter. Slot 0x105 has a PXI counter installed and is available for selection in the Initialize dialog box as shown in Figure 5-4:

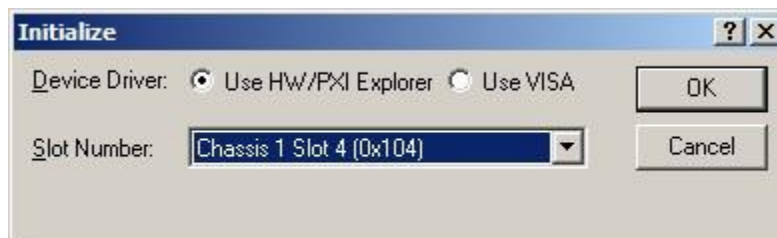


Figure 5-4: Initialize Dialog Box

The **Slot Number**: in the Initialize dialog box refers to the PXI slot in which the GC2200/GTX2200 board is installed. Select the slot from the drop down list. The list displays the slots where GC2200/GTX2200 boards are installed. The specified slot number can also be reviewed or set by using the **PXI/PCI Explorer** applet located in the Windows Control Panel. Select the board slot number and click **OK** to initialize the driver for the specified board.

After the board is initialized the panel is enabled and will display the current setting of the board, click on the **About** tab. The **Calibration By User...** button should be enabled.



Figure 5-5: GC2200/GTX2200 Virtual Panel – About Tab

Clicking on the “Calibration By User...” button will open the User Calibration dialog as follows:

Time Base Calibration

Click the Time Base (10MHz) button. Read the instruction regarding calibration the Time Base and when ready click the Calibrate button. The calibration process takes about a minute to complete.

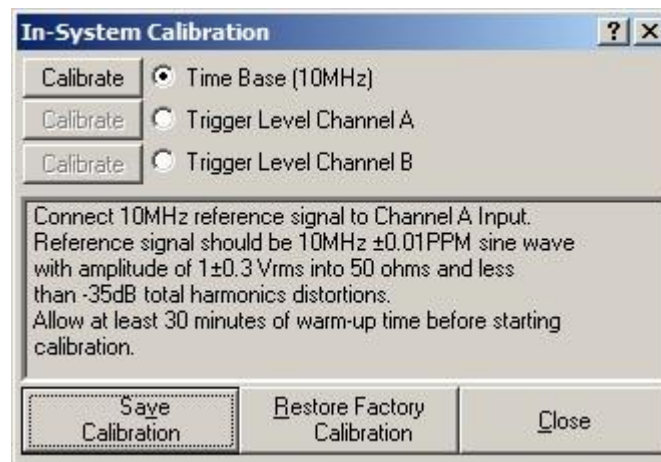


Figure 5-6: User Calibration – Time base

When done the dialog updates the calibration status accordingly.

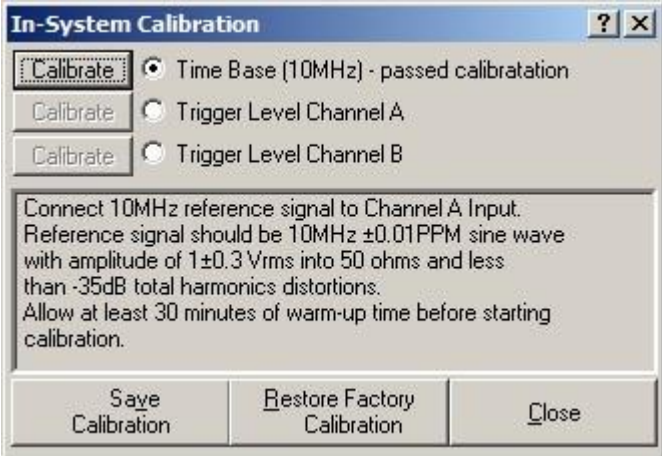


Figure 5-7: User Calibration – Time base passed calibration

Trigger Level Channel A Calibration

Click the Trigger Level Channel A button. Read the instruction regarding calibration the Trigger Level Channel A and when ready click the Calibrate button. The calibration process takes few seconds to complete.

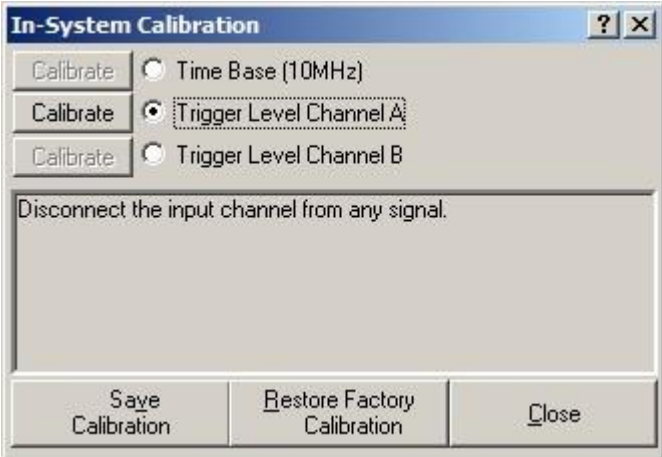


Figure 5-8: User Calibration – Trigger Level Channel A

When done the dialog updates the calibration status accordingly.

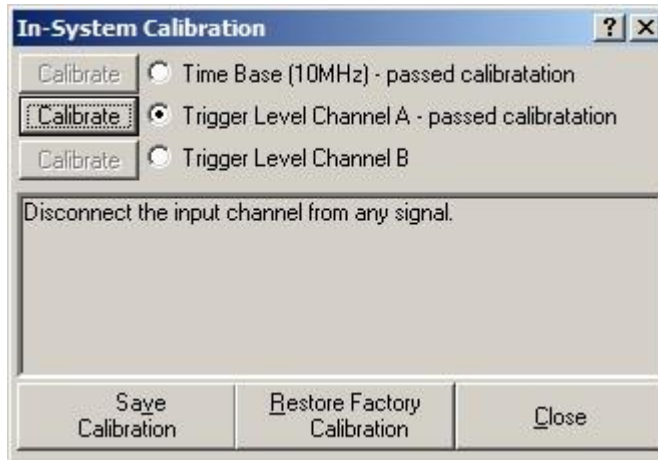


Figure 5-9: User Calibration – Trigger Level Channel A passed calibration

Trigger Level Channel B Calibration

Click the Trigger Level Channel B button. Read the instruction regarding calibration the Trigger Level Channel B and when ready click the Calibrate button. The calibration process takes few seconds to complete.

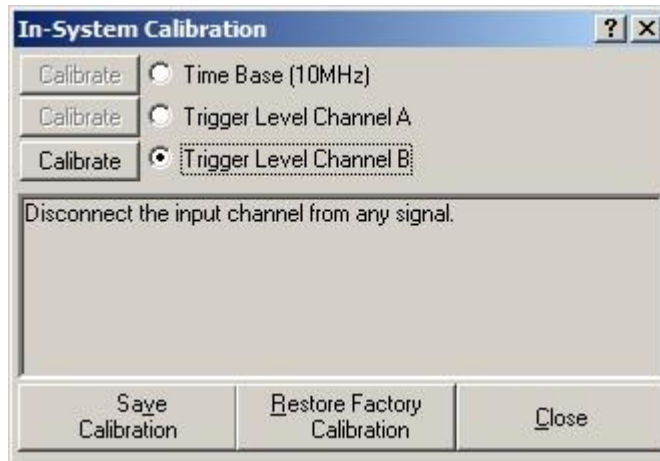


Figure 5-10: User Calibration – Trigger Level Channel B

When done the dialog updates the calibration status accordingly.

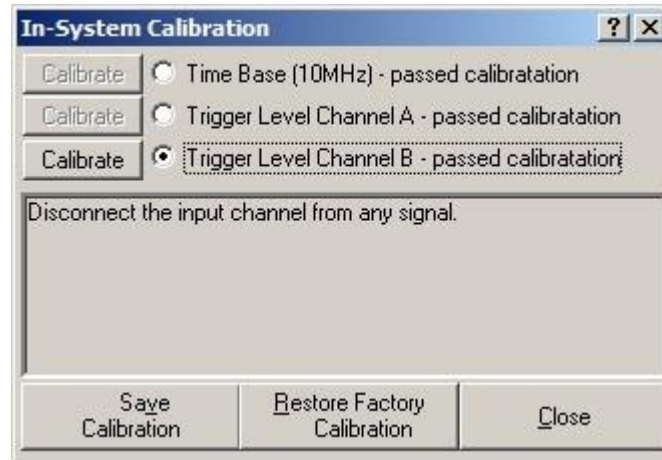


Figure 5-11: User Calibration – Trigger Level Channel B passed calibration

Save Calibration

Click the Save Calibration button to save the calibration data or Click k cancel to abort calibration without saving.

Restore Calibration

Click the Restore Factory Calibration in order to resort all devices back to the factory calibration

In-System Calibration Example program

The following example demonstrates how to Calibrating the GTX22X0 and GC22X0 boards using the GXCNT drive using C programming language under Windows. To run, enter the following command line:

GxCntExample <Slot> <Operation>

Where:

| | | |
|-------------|--|---|
| <Slot> | PXI Explorer slot number where the board reside. | |
| <Operation> | Operation code: | |
| | CAL_ALL_DEVICES | Calibrate Time base and Channels A and B Trigger Levels. |
| | RESTORE_ALL_DEVICES_CAL | Restore calibration of all devices to Factory calibration. |
| | RESTORE_TIME_BASE_CAL | Restore Time Base calibration to Factory calibration. |
| | RESTORE_CHA_TRG_LVL_CAL | Restore Channel A Trigger Level calibration to Factory calibration. |
| | RESTORE_CHB_TRG_LVL_CAL | Restore Channel B Trigger Level calibration to Factory calibration. |

In-System Calibration Sample Program Listing

```
/******
```

```
FILE           : GxCntExampleUserCal.cpp
PURPOSE        : WIN32 example program for Calibrating the GTX22X0 and GC22X0 boards using the
                 GXCNT driver.
CREATED        : July 2006
COPYRIGHT      : Copyright 2006 MARVIN TEST SOLUTIONS.
COMMENTS      :
```

To compile the WIN32 example:

1. Microsoft VC++

```
Load GxCntExampleUserCal.dsp, .vcproj or .mak, depends on
the VC++ version from the Project/File/Open... menu
Select Project/Rebuild all from the menu
```

2. Borland C++ Builder

```
Load GxCntExampleUserCal.bpr from the Project/Open
Project... menu
Select Project/Build all from the menu
```

```
*****/
```

```
#include "windows.h"
#include "GxCnt.h"
#include "stdio.h"
```

```

// Borland C++ Builder compat. block
#if defined(__BORLANDC__)
#pragma hdrstop
#include <condefs.h>
USELIB("GxCntBC.lib");
USERC("GxCntExampleUserCal.rc");
#endif // defined(__BORLANDC__)

//*****
//          DisplayMsg
//*****

void DisplayMsg(PSTR lpszMsg)
{
    MessageBeep(0);
    MessageBox(0, lpszMsg, "GxCnt User Calibrating example program", MB_OK);
    return;
}

//*****
//          DisplayUsage
//*****

void DisplayUsage(void)
{
    DisplayMsg(
        "Example program for Calibrating the GTX22X0 and GC22X0 boards using the GXCNT driver.\r\n\r\n"
        "Usage: GxCntExampleUserCal <slot> <operation>\r\n\r\n"
        "<slot> - PCI/PXI slot number as shown by the PXI explorer\r\n\r\n\r\n"
        "<operation> - one of the followings:\r\n"
        " CAL_ALL_DEVICES=Calibrate Time base and Channels A and B Trigger Levels\r\n"
        " RESTORE_ALL_DEVICES_CAL=Restore calibration of all devices to Factory calibration.\r\n"
        " RESTORE_TIME_BASE_CAL=Restore Time Base calibration to Factory calibration.\r\n"
        " RESTORE_CHA_TRG_LVL_CAL=Restore Channel A Trigger Level calibration to “
        “Factory calibration.\r\n"
        " RESTORE_CHB_TRG_LVL_CAL=Restore Channel B Trigger Level calibration to “
        “Factory calibration.\r\n"
    );
    exit(1);
}

//*****
//          CheckStatus

```

```

/*****
void CheckStatus(SHORT nStatus)
{
    CHAR  sz[256];
    if (!nStatus) return;
    GxCntGetErrorString(nStatus, sz, sizeof sz, &nStatus);
    DisplayMsg(sz);
    DisplayMsg("Aborting the program...");
    exit(nStatus);
}
/*****

// MAIN
// The main function receives the following parameters:
// <slot> - GTX22X0/GC22X0 counter board slot number (e.g. 1)
//<operation> - one of the followings:
//CAL_ALL_DEVICES = Calibrate Calibrate Time base and Channels A and B Trigger Levels.
//RESTORE_ALL_DEVICES_CAL = Restore calibration of all devices to Factory calibration.
//RESTORE_TIME_BASE_CAL = Restore Time Base calibration to Factory calibration.
//RESTORE_CHA_TRG_LVL_CAL= Restore Channel A Trigger Level calibration to Factory calibration.
//RESTORE_CHB_TRG_LVL_CAL= Restore Channel B Trigger Level calibration to Factory calibration.
/*****

int main(int argc, char **argv)
{
    short  nSlotNum;           // Board slot number
    char*  sOperation;        // Board Operation
    short  nHandle;           // Board handle
    short  nStatus;           // Returned status
    short  nCalStatus;        // Calibration status

    // Check minimum number of arguments received
    if (argc<3) DisplayUsage();
    // Parse command line parameters
    nSlotNum=(SHORT)strtol(++argv, NULL, 0);
    if (nSlotNum<0) DisplayUsage();
    GxCntInitialize(nSlotNum, &nHandle, &nStatus);
    CheckStatus(nStatus);
    sOperation = strupr(++argv);

```



```

// Calibrate all devices
if(!strcmp(sOperation, "CAL_ALL_DEVICES"))
{
    // Start the Calibration process
    GxCntInSystemCalStart(nHandle, &nStatus);
    CheckStatus(nStatus);
    // Time Base Calibration
    // Display Time Base calibration instructions
    DisplayMsg("Connect 10MHz reference signal to Channel A Input. Reference\r\n"
        "signal should be 10MHz  $\pm$ 0.01PPM sine wave with amplitude of\r\n"
        " $1\pm 0.3$  Vrms into 50 ohms and less than -35dB total harmonics\r\n"
        "distortions.\r\n"
        "Allow at least 30 minutes of warm-up time before starting\r\n"
        "calibration.\r\n");
    GxCntInSystemCalDevice(nHandle, GX2200_CAL_TIME_BASE, &nStatus);
    CheckStatus(nStatus);
    // Check calibration status
    GxCntInSystemCalGetStatus(nHandle, GX2200_CAL_TIME_BASE, &nCalStatus, &nStatus);
    if (nCalStatus!=GX2200_DEVICE_PASSED_CALIBRATION)
        DisplayMsg("Time Base Calibration - Failed");
    // Channel A Trigger Level
    // Display Trigger Level calibration instructions
    DisplayMsg("Disconnect the input channel from any signal.\r\n");
    GxCntInSystemCalDevice(nHandle, GX2200_CAL_TRIG_LEVEL_CH_A, &nStatus);
    CheckStatus(nStatus);
    // Check calibration status
    GxCntInSystemCalGetStatus(nHandle, GX2200_CAL_TRIG_LEVEL_CH_A, &nCalStatus,
        &nStatus);
    if (nCalStatus!=GX2200_DEVICE_PASSED_CALIBRATION)
        DisplayMsg("Channel A Trigger Level Calibration - Failed");
    // Channel B Trigger Level
    // Display Trigger Level calibration instructions
    DisplayMsg("Disconnect the input channel from any signal.\r\n");
    GxCntInSystemCalDevice(nHandle, GX2200_CAL_TRIG_LEVEL_CH_B, &nStatus);
    CheckStatus(nStatus);
    // Check calibration status
    GxCntInSystemCalGetStatus(nHandle, GX2200_CAL_TRIG_LEVEL_CH_B, &nCalStatus,
        &nStatus);
}

```

```

        if (nCalStatus!=GX2200_DEVICE_PASSED_CALIBRATION)
            DisplayMsg("Channel B Trigger Level Calibration - Failed");
        // Save calibration data
        GxCntInSystemCalSave(nHandle, &nStatus);
        CheckStatus(nStatus);
    }
    // Restore calibration of all devices to Factory calibration
    else if(!strcmp(sOperation, "RESTORE_ALL_DEVICES_CAL"))
    {
        GxCntInSystemCalRestore(nHandle, GX2200_CAL_TIME_BASE, &nStatus);
        CheckStatus(nStatus);
        GxCntInSystemCalRestore(nHandle, GX2200_CAL_TRIG_LEVEL_CH_A, &nStatus);
        CheckStatus(nStatus);
        GxCntInSystemCalRestore(nHandle, GX2200_CAL_TRIG_LEVEL_CH_B, &nStatus);
        CheckStatus(nStatus);
    }
    // Restore Time Base calibration to Factory calibration
    else if(!strcmp(sOperation, "RESTORE_TIME_BASE_CAL"))
    {
        GxCntInSystemCalRestore(nHandle, GX2200_CAL_TIME_BASE, &nStatus);
        CheckStatus(nStatus);
    }
    // Restore Channel A Trigger Level calibration to Factory calibration
    else if(!strcmp(sOperation, "RESTORE_CHA_TRG_LVL_CAL"))
    {
        GxCntInSystemCalRestore(nHandle, GX2200_CAL_TRIG_LEVEL_CH_A, &nStatus);
        CheckStatus(nStatus);
    }
    // Restore Channel B Trigger Level calibration to Factory calibration
    else if(!strcmp(sOperation, "RESTORE_CHB_TRG_LVL_CAL"))
    {
        GxCntInSystemCalRestore(nHandle, GX2200_CAL_TRIG_LEVEL_CH_B, &nStatus);
        CheckStatus(nStatus);
    }
    }else
        DisplayUsage();
    return 0;
}
//*****
//
//           End Of File
//*****

```

Chapter 6 - Functions Reference

Introduction

The functions reference chapter organizes the list of GC2200/GTX2200 driver functions in an alphabetical order. Each function description contains the function name; purpose, syntax, parameters description and type followed by Comments, an Example (written in C), and a See Also sections.

All function and parameter syntax follow the same rules:

- Strings are ASCIIZ (null or zero character terminated).
- The first parameter of most functions is *nHandle* (16-bit integer). This parameter is required for operating the board and is returned by the **GxCntInitialize** function. The *nHandle* is used to identify the board when calling a function for programming and controlling the operation of that board.
- All functions return a status with the last parameter named *pnStatus*. The *pnStatus* is zero if the function was successful, or non-zero on error. The description of the error is available using the **GxCntGetErrorString** function or by using a predefined constant, defined in the driver interface files: GXCNT.H, GXCNT.BAS, GXCNT.VB, GXCNT.PAS or GC2200/GTX2200.DRV.
- Parameter name are prefixed as follows:

| Prefix | Type | Example |
|-------------|---|---------------------------------|
| <i>a</i> | Array - prefix this before the simple type. | <i>anArray</i> (Array of Short) |
| <i>b</i> | BOOL – Boolean, 0 for FALSE; <>0 for TRUE | <i>bUpdate</i> |
| <i>d</i> | DOUBLE - 8 bytes floating point | <i>dReading</i> |
| <i>dw</i> | DWORD - double word (unsigned 32-bit) | <i>dwTimeout</i> |
| <i>hwnd</i> | Window handle (32-bit integer). | <i>hwndPanel</i> |
| <i>l</i> | LONG - (signed 32-bit) | <i>lBits</i> |
| <i>n</i> | SHORT - (signed 16-bit) | <i>nMode</i> |
| <i>p</i> | Pointer - Usually used to return a value. Prefix this before the simple type. | <i>pnStatus</i> |
| <i>sz</i> | Null - (zero value character) terminated string | <i>szMsg</i> |
| <i>uc</i> | BYTE - (8 bits) unsigned. | <i>ucValue</i> |
| <i>w</i> | WORD - Unsigned short (unsigned 16-bit) | <i>wParam</i> |

Table 6-1: Parameter Name Prefixes

GC2200/GTX2200 Functions

The following list is a summary of functions available for the GC2200/GTX2200:

| Driver Functions | Description |
|--|--|
| General | |
| GxCntInitialize | Initializes the driver for the specified slot using the HW device driver. |
| GxCntInitializeVisa | Initializes the driver for the specified slot using VISA. |
| GxCntPanel | Opens a virtual panel used to interactively control the GC2200/GTX2200. |
| GxCntReset | Resets the GC2200/GTX2200 board to its default settings. |
| GxCntGetDriverSummary | Returns the driver name and version. |
| GxCntGetErrorString | Returns the error string associated with the specified error number. |
| Functions | |
| GxCntChannelAutoSet | Automatic adjustment of the board setting in order to obtain a stable reading. |
| GxCntClear | Clear the counter buffers and starts a new measurement. |
| GxCntGetAcquisitionMode | Returns the board Acquisition mode. |
| GxCntGetAcquisitionTimeInterval | Returns the board Acquisition time interval. |
| GxCntGetArmSlope | Returns the Arm start and stop slope. |
| GxCntGetArmSource | Returns the Arm source. |
| GxCntGetBoardSummary | Returns the board information. |
| GxCntGetBoardType | Returns the board type GC2210/GTX2210, GTX2220 or GTX2230. |
| GxCntGetCalibrationInfo | Returns the calibration information. |
| GxCntGetCalibrationMode | Returns the board Calibration mode. |
| GxCntGetChannelAFrequencyRange | Returns the channel A frequency mode. |
| GxCntGetChannelCouplingMode | Returns the specified channel-coupling mode. |
| GxCntGetChannelFilterMode | Returns the specified channel filter mode. |
| GxCntGetChannelFilterValue | Returns the specified channel filter value. |
| GxCntGetChannelImpedance | Returns the specified channel input impedance. |
| GxCntGetChannelSlope | Returns the specified channel input slope. |
| GxCntGetChannelTrigerLevel | Returns the specified channel input trigger level. |
| GxCntGetChannelTrigerLevelMode | Returns the specified channel input trigger level mode. |
| GxCntGetClockSource | Returns the board clock source. |
| GxCntGetCommonInput | Returns the common input mode state and active input channel. |

| Driver Functions | Description |
|---|--|
| GxCntGetCounterRefClockToPxiRefClock State | Returns the connection state of the Counter Reference Clock to the PXI Reference Clock. |
| GxCntGetExtendedSerialNumber | Returns the Extended board's Serial Number information. |
| GxCntGetFunction | Returns the function and function settings. |
| GxCntGetGateTime | Returns the gate time. |
| GxCntGetPrescaleMode | Returns the pre-scale mode for both channels. |
| GxCntGetMeasurementNumberOfDigits | Returns the measurement number of digits and mode. |
| GxCntGetMeasurementTimeout | Returns the measurement timeout value. |
| GxCntGetTimeIntervalDelay | Returns the delay time for the GxCntSetFunctionTimeIntervalDelay function. |
| GxCntGetTotalizeGateMode | Returns the counting Gate to open or close when in Totalize mode. |
| GxCntGetTriggerSlope | Returns the trigger slope. |
| GxCntGetTriggerSource | Returns the trigger source. |
| GxCntMeasurementReady | Returns true if a new measurement is ready to read. |
| GxCntReadMeasurement | Read a single measurement. |
| GxCntReadMeasurementArray | Returns specified number of measurements into an array while specifying the amount of time allowed for filling the array before returning. |
| GxCntReadMeasurementString | Read a single measurement formatted as string.. |
| GxCntReadStatusRegister | Returns the status register. |
| GxCntSelfTest | Run a self-test. |
| GxCntSetAcquisitionMode | Sets the board Acquisition mode. |
| GxCntSetAcquisitionTimeInterval | Sets the board Acquisition time interval. |
| GxCntSetArmSlope | Sets the Arm start and stop slope. |
| GxCntSetArmSource | Sets the Arm source. |
| GxCntSetCalibrationMode | Sets the board Calibration mode. |
| GxCntSetChannelAFrequencyRange | Sets the channel A frequency mode. |
| GxCntSetChannelCouplingMode | Sets the specified channel-coupling mode. |
| GxCntSetChannelFilterMode | Sets the specified channel filter mode. |
| GxCntSetChannelFilterValue | Sets the specified channel filter value. |
| GxCntSetChannelImpedance | Sets the specified channel input impedance. |
| GxCntSetChannelSlope | Sets the specified channel input slope. |
| GxCntSetChannelTrigerLevel | Sets the specified channel input trigger level. |
| GxCntSetChannelTrigerLevelMode | Sets the specified channel input trigger level mode. |
| GxCntSetClockSource | Sets the board clock source. |

| Driver Functions | Description |
|---|--|
| GxCntSetCommonInput | Sets the common input mode state and active input channel. |
| GxCntSetCounterRefClockToPxiRefClock State | Sets the connection state of the Counter Reference Clock to the PXI Reference Clock. |
| GxCntSetFunctionAccumulate | Sets the board function to Accumulate |
| GxCntSetFunctionAutoRatio | Sets the function mode to Auto Ratio. |
| GxCntSetFunctionFastFrequency | Sets the function mode to Fast Frequency. |
| GxCntSetFunctionFrequency | Sets the function mode to Frequency. |
| GxCntSetFunctionPeriod | Sets the function mode to Period |
| GxCntSetFunctionPulseWidth | Sets the function mode to Pulse Width. |
| GxCntSetFunctionRatio | Sets the function mode to Ratio. |
| GxCntSetFunctionSinglePeriod | Sets the function mode to Single Period. |
| GxCntSetFunctionTestInternalClock | Sets the function mode to test the internal clock. |
| GxCntSetFunctionTimeInterval | Sets the function mode to Time Interval. |
| GxCntSetFunctionTimeIntervalDelay | Sets the function mode to Time Interval with Delay. |
| GxCntSetFunctionTotalize | Sets the function mode to Totalize. |
| GxCntSetFunctionTotalizeGated | Sets the function mode to Totalize Gated. |
| GxCntSetFunctionTotalizeGatedOnce | Sets the function mode to Totalize Gated Once. |
| GxCntSetGateTime | Sets the gate time for frequency-type measurements (Frequency, Period and Check). |
| GxCntSetMeasurementNumberOfDigits | Sets the measurement number of digits and mode. |
| GxCntSetMeasurementTimeout | Sets the measurement timeout value. |
| GxCntSetPrescaleMode | Sets the pre-scale mode for both channels. |
| GxCntSetTimeIntervalDelay | Sets the delay time for the GxCntSetFunctionTimeIntervalDelay function. |
| GxCntSetTotalizeGateMode | Sets the counting Gate to open or closed when in Totalize mode. |
| GxCntSetTriggerSlope | Sets trigger slope. |
| GxCntSetTriggerSource | Sets trigger source. |
| GxCntTrig | The counter will initiate a new measurement after calling this function. |

| In-System Calibration Functions | |
|--|--|
| GxCntInSystemCalDevice | Calibrate the specified device. |
| GxCntInSystemCalGetStatus | Returns the specified device calibration status. |
| GxCntInSystemCalRestore | Restore the specified device calibration to the factory calibration. |
| GxCntInSystemCalSave | Save calibrated devices data to the on-board EEPROM. |
| GxCntInSystemCalStart | Start the In-System calibration. |
| Upgrade firmware functions | |
| GxCntUpgradeFirmware | Upgrades the board's firmware. |
| GxCntUpgradeFirmwareStatus | Monitor the firmware upgrade process. |

GxCntChannelAutoSet

Purpose

Automatic adjustment of the board setting in order to obtain a stable reading.

Syntax

GxCntChannelAutoSet (*nHandle*, *nChannel*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|---|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nChannel</i> | SHORT | Specified Channel number: <ul style="list-style-type: none"> • 0 = GXCNT_CHANNEL_A: Channel A • 1 = GXCNT_CHANNEL_B: Channel B. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

The Autoset function significantly eases instrument operation through Automatic adjustment of the board setting for almost any signal. Manual adjustments might be needed for special cases (e.g. complex signals). The Autoset function will try to obtain a stable measurement with the best settings. It automatically adjusts the trigger level mode, trigger level, AC/DC coupling, filter mode and filter value.

Calling this function will reset the board and apply settings to the specified channel. If the function is unable to find suitable settings upon return the function will reset the board.

Further adjustments to the board settings might be needed in order to increase accuracy. This function is mostly useful for low frequencies waveforms (less than 20KHz) with a slow slew rate or noisy waveform were the default setting would have difficulties to adjust.

Example

The following example Auto sets channel A:

```
SHORT nHandle, nStatus;
GxCntChannelAutoSet(nHandle, GXCNT_CHANNEL_A, &nStatus)
```

See Also

GxCntGetChannelCouplingMode, **GxCntGetChannelFilterMode**, **GxCntGetChannelFilterValue**, **GxCntGetChannelAFrequencyRange**, **GxCntGetChannelImpedance**, **GxCntGetChannelSlope**, **GxCntGetErrorString**

GxCntClear

Purpose

Clear the counter buffers and start new measurement.

Syntax

GxCntClear (*nHandle*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

Clears the output data buffer, the count in **Totalize** and **Accumulate** modes, and clears the errors and starts a new measurement. The counter's setup does not is not change.

Example

The following example clears the counter:

```
SHORT nHandle, nStatus;  
GxCntClear (nHandle, &nStatus);
```

See Also

GxCntSetFunctionAccumulate, **GxCntSetFunctionTotalize**, **GxCntReset**, **GxCntGetErrorString**

GxCntGetAcquisitionMode

Purpose

Returns the board Acquisition mode.

Syntax

GxCntGetAcquisitionMode (*nHandle*, *pnMode*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|---|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>pnMode</i> | PSHORT | Returned Acquisition mode: <ul style="list-style-type: none"> • 0 = GXCNT_ACQUISITION_CONTINUOUS: The counter continuously makes measurements. • 1 = GXCNT_ACQUISITION_SINGLE: Instrument makes a single measurement. Each Call to GxCntTrig initiates a new measurement. • 2 = GXCNT_ACQUISITION_PACED: The Pace mode ensures acquisition of data at accurately spaced time intervals. In Paced operation, the user specifies the time which should elapse BETWEEN initiations of measurements. See GxCntSetAcquisitionTimeInterval function for more details on the timing settings. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Example

The following example returns the Acquisition mode:

```
SHORT nHandle, nMode, nStatus;
GxCntGetAcquisitionMode (nHandle, &nMode, &nStatus)
```

See Also

GxCntSetAcquisitionMode, **GxCntGetErrorString**

GxCntGetAcquisitionTimeInterval

Purpose

Returns the board Acquisition Time Interval.

Syntax

GxCntGetAcquisitionTimeInterval (*nHandle*, *pdSeconds*, *pnStatus*)

Parameters

| Name | Type | Comments |
|------------------|---------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>pdSeconds</i> | PDOUBLE | Returned acquisition time interval in seconds, interval range. Intervals can be set from 0.8 ms to 3200 seconds. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

When setting the acquisition mode to Paced mode (see **GxCntSetAcquisitionMode** function) the GTX22x0 ensures the acquisition of data at accurately spaced time intervals. In Paced operation, the user specifies the time which should elapse BETWEEN initiations of measurements. The pace times are digitally generated on the GTX22x0 and are accurate to within 200 μ s (the errors do not accumulate).

Some care must be used when selecting pacing intervals to ensure consistent timing between measurements. The Acquisition interval must be at least 600 μ s, longer than the actual gate time in Frequency and Period modes and 600 μ s longer than the actual measurement interval in timing modes (e.g: Pulse Width). If these recommendations are not followed, the counter will simply “skip” measurements, and the data will be taken at uneven intervals. If skipping occurs in Acquire mode the function returns an error.

Example

The following example returns the Acquisition time interval:

```
SHORT nHandle, nStatus;
DOUBLE dSeconds
GxCntGetAcquisitionTimeInterval (nHandle, &dSeconds, &nStatus)
```

See Also

GxCntSetAcquisitionTimeInterval, **GxCntSetAcquisitionMode**, **GxCntGetAcquisitionMode**, **GxCntGetErrorString**

GxCntGetArmSlope

Purpose

Returns the Arm Start or Stop slope.

Syntax

GxCntGetArmSlope (*nHandle*, *nStartStop*, *pnSlope*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-------------------|--------|---|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nStartStop</i> | SHORT | Specify Arm Start or Stop slope: <ul style="list-style-type: none"> • 0 = GXCNT_ARM_START_SLOPE: Arm Start slope • 1 = GXCNT_ARM_STOP_SLOPE: Arm Stop slope |
| <i>pnSlope</i> | PSHORT | Returned slope: <ul style="list-style-type: none"> • 0 = GXCNT_ARM_POSITIVE_SLOPE: Positive signal transition • 1 = GXCNT_ARM_NEGATIVE_SLOPE: Negative signal transition • 2 = GXCNT_ARM_DISABLE_SLOPE: Disables start/stop arming |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

The Arm start/stop slope controls the initiation and termination of the measurement. This setting is active only when the Arm source is set to external using the **GxCntSetArmSource** command.

Example

The following example returns the Arm starts slope settings:

```
SHORT nHandle, nSlope, nStatus;
GxCntGetArmSlope (nHandle, GXCNT_ARM_START_SLOPE, &nSlope, &nStatus)
```

See Also

GxCntSetArmSlope, **GxCntSetArmSource**, **GxCntGetArmSource**, **GxCntGetErrorString**

GxCntGetArmSource

Purpose

Returns Arm source.

Syntax

GxCntGetArmSource(*nHandle*, *pnSource*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|---|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>pnSource</i> | PSHORT | Returned Arm source: <ul style="list-style-type: none"> • 0 = GXCNT_ARM_INTERNAL: Arming measurements are done internally according to Acquisition mode. • 1 = GXCNT_ARM_EXTERNAL: The External Arm Input connector • 2 = GXCNT_ARM_ALTERNATE: The Alternate input channel (only for single channel measurements such as Frequency or Period). • 3 = GXCNT_ARM_OFF: Disable external Arm input, Measurements are armed internal to the counter board. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

The Arm source setting does not affect the **Totalize**, **Totalize Gated** and **GxCntSetFunctionTestInternalClock** functions.

Example

The following example returns the Arm source:

```
SHORT nHandle, nSource, nStatus;
GxCntGetArmSource (nHandle, &nSource, &nStatus)
```

See Also

GxCntSetArmSlope, **GxCntSetArmSlope**, **GxCntGetArmSlope**, **GxCntGetErrorString**

GxCntGetBoardSummary

Purpose

Returns the board information.

Syntax

GxCntGetBoardSummary(*nHandle*, *pszSummary*, *nSumMaxLen*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-------------------|--------|---|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>pszSummary</i> | PSTR | Buffer to contain the returned board info (null terminated) string. |
| <i>nMaxLen</i> | SHORT | Size of the buffer to contain the string. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

The counter summary string provides the following data from the board in the order shown:

- Instrument Name (e.g., GX2220)
- Hardware Revision (e.g., 1.00)
- Firmware Revision (e.g., 'A')

For example, the returned string for for PCB revision B looks like the following:

```
"GTX2220, Hardware Revision 1.00, Firmware Revision A"
```

For PCB revision C and above the returned string looks like the following:

```
GTX2230, PROM-Version: E 1.60, FPGA-Version: 0xC001, S/N: GTX22300154-CG-CB-0, Oscillator type:
Standard, Time Base Calibration time: Mon Jan 14 14:43:25 2008, Ch. A Trigger Level Calibration time: Mon Jan
14 14:38:21 2008, Ch. B Trigger Level Calibration time: Mon Jan 14 14:39:02 2008
```

Example

The following example returns the board summary:

```
SHORT nHandle, nStatus;
CHAR szSummary [256];

GxCntGetBoardSummary(nHandle, szSummary, sizeof(szSummary), &nStatus);
```

See Also

GxCntGetDriverSummary, **GxCntInitialize**, **GxCntGetErrorString**

GxCntGetBoardType

Purpose

Returns the board type.

Syntax

GxCntGetBoardType (*nHandle*, *pnBoardType*, *pnStatus*)

Parameters

| Name | Type | Comments |
|--------------------|--------|---|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>pnBoardType</i> | PSHORT | Returned board type: <ul style="list-style-type: none"> • 1 = GTX2220 • 2 = GTX2210 • 3 = GTX2230 • 11 = GC2220 • 12 = GC2210 • 13 = GC2230 |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

Example

The following example returns the board type:

```
SHORT nHandle, nBoardType, nStatus;
GxCntGetBoardType (nHandle, &nBoardType, &nStatus)
```

See Also

GxCntInitialize, **GxCntGetErrorString**

GxCntGetCalibrationInfo

Purpose

Returns the calibration information.

Syntax

GxCntGetCalibrationInfo (*nHandle*, *pszCalibrationInfo*, *nInfoMaxLen*, *pnDaysUntilExpire*, *pnStatus*)

Parameters

| Name | Type | Comments |
|--------------------------|--------|---|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>pszSummary</i> | PSTR | Buffer to contain the returned board's calibration information (null terminated) string. |
| <i>nSumMaxLen</i> | SHORT | Size of the buffer to contain the error string. |
| <i>pnDaysUntilExpire</i> | PSHORT | Returns the number of days until or from expiration, if number is > 0 then calibration is current otherwise past due. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

The returned board's calibration information has the following fields:

Model: model number, e.g. "GTX2230"

Serial Number: serial number, e.g. 216

Control Number: Marvin Test Solutions control number, e.g. "*-CH-CB-0"

Production Calibration Date: Wed Oct 24 12:30:25 2010

Calibration Date: Wed Oct 24 12:31:58 2010

Recommended Interval: 1 year

Next Calibration Date: Fri Oct 24 12:31:58 2011

Status: calibration status can be either "Expired" followed by the number of days past expiration or "Current" followed by number of days until expire.

Calibration License: can be either "Installed" with the calibration license number or "Not Installed".

Example

The following example returns the board's calibration information string:

```
SHORT  nStatus;  
char   szCalibrationInfo[1024];  
BOOL   bExpired;  
GxCntGetCalibrationInfo(nHandle, szCalibrationInfo, sizeof szCalibrationInfo,  
&bExpired, &nStatus);
```

szCalibrationInfo string printout:

```
Model: GTX2230  
Serial Number: 216  
Control Number: *-CH-CB-0  
Production Calibration Date: Wed Oct 24 12:30:25 2007  
Calibration Date: Wed Oct 24 12:31:58 2007  
Recommended Interval: 1 year  
Next Calibration Date: Fri Oct 24 12:31:58 2008  
Status: Expired (891 days past expiration)  
Calibration License: Installed license number 3
```

See Also

GxCntInitialize, **GxCntGetBoardSummary**, **GxCntGetErrorString**

GxCntGetCalibrationMode

Purpose

Returns the Calibration mode

Syntax

GxCntGetCalibrationMode (*nHandle*, *pnMode*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|---|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>pnMode</i> | PSHORT | Returned Calibration mode: <ul style="list-style-type: none"> • 0 = GXCNT_CAL_OFF: The current calibration factors are retained. Stable calibration factors preclude a possible shift in results during a period of data collection. • 1 = GXCNT_CAL_CONTINUOUS: Calibration is performed continuously approximately once every 60 seconds. Default settings. • 2 = GXCNT_CAL_ONCE: Selecting “Once” causes an immediate execution of the calibration procedure, and then disables future calibration. This mode ensures that data is taken with recent calibration factors, and prevents re-calibration from causing a systematic shift in results. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

The calibration mode controls how often the calibration routine is invoked.

Example

The following example returns the Calibration mode:

```
SHORT nHandle, nMode, nStatus;
GxCntGetCalibrationMode (nHandle, &nMode, &nStatus)
```

See Also

GxCntSetCalibrationMode, **GxCntGetErrorString**

GxCntGetChannelAFrequencyRange

Purpose

Returns channel A frequency range (GC2220/GC2230/GTX2220/GTX2230 only).

Syntax

GxCntGetChannelAFrequencyRange (*nHandle*, *pnRange*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|---|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>pnRange</i> | PSHORT | Channel A frequency range: <ul style="list-style-type: none"> • 0 = GXCNT_FREQUENCY_RANGE_NORMAL: Channel A frequency range is DC to 225MHz. • 1 = GXCNT_FREQUENCY_RANGE_HIGH: Channel A frequency range is 100MHz to 1.3GHz for the GC2220/GTX2220 or 2GHz for the GC2230/GTX2230. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

When in high frequency range the following channel's settings are automatically set:

- Impedance is set to **1MOhms**
- Coupling is set to **AC**.
- Filter mode is set to **OFF**

Example

The following example returns the channel A frequency mode:

```
SHORT nHandle, nRange, nStatus;
```

```
GxCntGetChannelAFrequencyRange (nHandle, &nRange, &nStatus)
```

See Also

GxCntSetChannelAFrequencyRange, **GxCntGetErrorString**

GxCntGetChannelCouplingMode

Purpose

Returns the specified coupling mode.

Syntax

GxCntGetChannelCouplingMode (*nHandle*, *nChannel*, *pnMode*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nChannel</i> | SHORT | Specified Channel number: <ul style="list-style-type: none"> • 0 = GXCNT_CHANNEL_A: Channel A • 1 = GXCNT_CHANNEL_B: Channel B. |
| <i>pnMode</i> | PSHORT | Returned channel's input coupling mode: <ul style="list-style-type: none"> • 0 = GXCNT_COUPLING_AC: AC Coupling mode (default). • 1 = GXCNT_COUPLING_DC: DC Coupling mode. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Example

The following example returns channel A coupling mode:

```
SHORT nHandle, nMode, nStatus;
GxCntGetChannelImpedance (nHandle, GXCNT_CHANNEL_A, &nMode, &nStatus)
```

See Also

GxCntSetChannelCouplingMode, **GxCntGetErrorString**

GxCntGetChannelFilterMode

Purpose

Returns the specified channel filter mode.

Syntax

GxCntGetChannelFilterMode (*nHandle*, *nChannel*, *pnMode*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|---|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nChannel</i> | SHORT | Specified Channel number: <ul style="list-style-type: none"> • 0 = GXCNT_CHANNEL_A: Channel A • 1 = GXCNT_CHANNEL_B: Channel B. |
| <i>pnMode</i> | PSHORT | Returned channel's filter mode: <ul style="list-style-type: none"> • 0 = GXCNT_FILTER_OFF: The filter is disabled. • 1 = GXCNT_FILTER_AUTO: In this mode the filter is enabled and the driver analyzes the measured signal and determines the best filter value each time GxCntReadMeasurement or GxCntReadMeasurementString is called. • 2 = GXCNT_FILTER_VALUE_FIXED: In this mode the filter is enabled and the user sets the filter value see GxCntSetChannelFilterValue for details. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

The Filter should be enabled when measuring low frequencies (less than 20KHz) superimposed with a high-frequency noise. When enabled the Filter adds a delay in microseconds after first trigger is detected. The counter will then disregard any additional trigger events for the duration of the delay.

Example

The following example returns channel A filter mode:

```
SHORT nHandle, nMode, nStatus;
GxCntGetChannelFilterMode (nHandle, GXCNT_CHANNEL_A, &nMode, &nStatus)
```

See Also

GxCntSetChannelFilterMode, **GxCntGetErrorString**

GxCntGetChannelFilterValue

Purpose

Returns the specified channel filter value.

Syntax

GxCntGetChannelFilterValue (*nHandle*, *nChannel*, *ndValue*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nChannel</i> | SHORT | Specified Channel number: <ul style="list-style-type: none"> • 0 = GXCNT_CHANNEL_A: Channel A • 1 = GXCNT_CHANNEL_B: Channel B. |
| <i>pnValue</i> | PSHORT | Return the specified channel filter value. The filter value is an integer representing filter value in uSec, e.g. value of 50 represents 50uSec. Filter value range from 5 to 6400, (5uSec to 6400uSec). |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

The Filter should be used when measuring low frequencies (less than 20KHz) superimposed with a high-frequency noise. The Filter value adds a delay equal to *pdValue* in microseconds. After first trigger is detected the counter will disregard any additional trigger events for *pdValue* microseconds.

Note: The delay needs to be less than half of the expected frequency signal duty cycle. Filter value can be set when the filter mode is set to GXCNT_FILTER_VALUE_FIXED.

The filter value should be set according to the following equation:

$$\text{Filter Value (uS)} \leq \frac{1.0E + 6}{\text{ExpectedFrequency} * 4}$$

E.g.: if the expected value is about 10KHz then filter value should be around 25uSec.

Example

The following example returns channel A filter value:

```
SHORT nHandle, nStatus;
SHORT nValue
GxCntGetChannelFilterValue (nHandle, GXCNT_CHANNEL_A, &nValue, &nStatus)
```

See Also

GxCntSetChannelFilterValue, **GxCntGetErrorString**

GxCntGetChannelImpedance

Purpose

Returns the specified channel input impedance.

Syntax

GxCntGetChannelImpedance (*nHandle*, *nChannel*, *pnMode*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|---|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nChannel</i> | SHORT | Specified Channel number: <ul style="list-style-type: none"> • 0 = GXCNT_CHANNEL_A: Channel A • 1 = GXCNT_CHANNEL_B: Channel B. |
| <i>pnMode</i> | PSHORT | Returned channel's input impedance mode: <ul style="list-style-type: none"> • 0 = GXCNT_IMPEDANCE_1MOHMS: Selects 1MOhm impedance (default). • 1 = GXCNT_IMPEDANCE_50OHMS: Selects 50 Ohms impedance. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Example

The following example returns channel A impedance mode:

```
SHORT nHandle, nMode, nStatus;
GxCntGetChannelImpedance (nHandle, GXCNT_CHANNEL_A, &nMode, &nStatus)
```

See Also

GxCntSetChannelImpedance, **GxCntGetErrorString**

GxCntGetChannelSlope

Purpose

Returns the specified channel input slope.

Syntax

GxCntGetChannelSlope (*nHandle*, *nChannel*, *pnSlope*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|---|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nChannel</i> | SHORT | Specified Channel number: <ul style="list-style-type: none"> • 0 = GXCNT_CHANNEL_A: Channel A • 1 = GXCNT_CHANNEL_B: Channel B. |
| <i>pnSlope</i> | PSHORT | Returned channel's slope: <ul style="list-style-type: none"> • 0 = GXCNT_POSITIVE_SLOPE: Rising (Positive) edge. • 1 = GXCNT_NEGATIVE_SLOPE: Falling (Negative) edge. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

The channel slope sets the input signal edge that utilized for the measurement. The slope can specify either rising (**Positive**) edge or falling (**Negative**) edge.

Example

The following example returns channel A input slope:

```
SHORT nHandle, nMode, nStatus;
GxCntGetChannelSlope (nHandle, GXCNT_CHANNEL_A, &nMode, &nStatus)
```

See Also

GxCntSetChannelSlope, **GxCntGetErrorString**

GxCntGetChannelTriggerLevel

Purpose

Returns the specified channel input trigger level.

Syntax

GxCntGetChannelTriggerLevel (*nHandle*, *nChannel*, *pdVoltage*, *pnStatus*)

Parameters

| Name | Type | Comments |
|------------------|---------|---|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nChannel</i> | SHORT | Specified Channel number: <ul style="list-style-type: none"> • 0 = GXCNT_CHANNEL_A: Channel A • 1 = GXCNT_CHANNEL_B: Channel B. |
| <i>pdVoltage</i> | PDOUBLE | Returned channel's input trigger level voltage. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

The Trigger Level setting defines the input signal voltage that creates a logical event from the input comparator circuit (trigger level is also commonly known as “threshold” voltage). The trigger level can be set anywhere between +5.12V and -5.12V with 1mV resolution for PCB revision C and above (40mV for previous PCB). User entered values are rounded to the nearest least significant bit.

Trigger Level can be set automatically by the counter using or manually, see **GxCntSetChannelTriggerLevelMode** function for more details.

Example

The following example returns channel A Trigger Level setting:

```
SHORT nHandle, nMode, nStatus;
DOUBLE dVoltage;
GxCntGetChannelTriggerLevel (nHandle, GXCNT_CHANNEL_A, &dVoltage, &nStatus)
```

See Also

GxCntSetChannelTriggerLevel, **GxCntSetChannelTriggerLevelMode**, **GxCntGetChannelTriggerLevelMode**, **GxCntGetErrorString**

GxCntGetChannelTriggerLevelMode

Purpose

Returns the specified channel input trigger level mode.

Syntax

GxCntGetChannelTriggerLevelMode (*nHandle*, *nChannel*, *pnMode*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|---|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nChannel</i> | SHORT | Specified Channel number: <ul style="list-style-type: none"> • 0 = GXCNT_CHANNEL_A: Channel A • 1 = GXCNT_CHANNEL_B: Channel B. |
| <i>pnMode</i> | PSHORT | Returned channel's input trigger level mode: <ul style="list-style-type: none"> • 0 = GXCNT_TRIGGER_LEVEL_FIXED • 1 = GXCNT_TRIGGER_LEVEL_AUTO • 2 = GXCNT_TRIGGER_LEVEL_HOLD_LAST See comments for details. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

The Trigger Level setting defines the input signal voltage that creates a logical event from the input comparator circuit (trigger level is also commonly known as “threshold” voltage). The trigger level can be set anywhere between +5.12V and -5.12V with 1mV resolution for PCB revision C and above (40mV for previous PCB). User entered values are rounded to the nearest least significant bit. See **GxCntSetChannelTriggerLevelMode** function for more details.

The following shows channel A and B values:

0. GXCNT_TRIGGER_LEVEL_FIXED: Sets the trigger level to the value displayed in the Set Level controls.
1. GXCNT_TRIGGER_LEVEL_AUTO: The board evaluates the peak input signal levels just before each measurement. The trigger levels are then automatically set to the approximate signal midpoint. Auto trigger may add up to 250 ms per measurement and is specified only for signal frequencies above 100 Hz. Auto trigger and Hold can be used together to provide most of the benefits of auto trigger without excessive measurement time.
2. GXCNT_TRIGGER_LEVEL_HOLD_LAST: The trigger level setting that was active when it was invoked. The primary application for “Hold” is to maintain an auto trigger set threshold over a number of measurements, without incurring the auto trigger time penalty on each measurement.

Example

The following example returns channel A Trigger Level setting:

```
SHORT nHandle, nMode, nStatus;
DOUBLE dVoltage;
GxCntGetChannelTriggerLevelMode (nHandle, GXCNT_CHANNEL_A, &dVoltage, &nStatus)
```

See Also

GxCntSetChannelTriggerLevelMode, GxCntSetChannelTriggerLevel, GxCntGetChannelTriggerLevel, GxCntGetErrorString

GxCntGetClockSource

Purpose

Returns the board clock source.

Syntax

GxCntGetClockSource (*nHandle*, *pnSource*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|---|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>pnSource</i> | PSHORT | Returned clock source: <ul style="list-style-type: none"> • 0 = GXCNT_CLOCK_INTERNAL: Selects the internal clock installed on the board. • 1 = GXCNT_CLOCK_EXTERNAL: External clock input as the clock. • 2 = GXCNT_CLOCK_ALTERNATE: When measuring only one channel the second (or alternate) channel can be defined as the reference frequency input. The main input channels provide better signal conditioning than the External Clock input. This mode affects only Frequency, FastFrequency, Period, Single Period and Width functions. • 3 = GXCNT_CLOCK_PXI_10MHZ_CLOCK: The PXI chassis backplane 10MHz clock line (PXI_CLK10) is the clock. Applied only to GTX2220/30 models. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

Settings the clock source to GXCNT_CLOCK_PXI_10MHZ_CLOCK (GTX2220/30 only):

The PXI chassis has a 10 MHz clock line (PXI_CLK10) in its backplane that allows devices plugged into the non-star controller slots (slots 3 and higher) to phase-lock their oscillators to the backplane clock. A single backplane clock can control this way the drift in the oscillators on all the devices in the non-star controller slots. The GTX2220/30board, when plugged into the star-controller slot (slot 2) can replace the PXI backplane clock with its much more stable oscillator. Alternatively, when plugged into one of the non-star controller slots (slots 3 and higher), the GTX2220/30board can be programmed that its clock source will use the PXI backplane clock.

Caution: The reference frequency must be within 5% of 10 MHz, or the resolution may be significantly reduced.

Example

The following example returns the clock source:

```
SHORT nHandle, nSource, nStatus;
GxCntGetClockSource (nHandle, &nSource, &nStatus)
```

See Also

GxCntSetClockSource, **GxCntSetCounterRefClockToPxiRefClockState**, **GxCntGetCounterRefClockToPxiRefClockState**, **GxCntGetErrorString**

GxCntGetCommonInput

Applies To

GTX2210, GTX2220, GTX2230

Purpose

Returns the common input mode state and active input channel.

Syntax

GxCntGetCommonInput (*nHandle*, *pbEnable*, *pnActiveInputChannel*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------------------|--------|---|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>pbEnable</i> | PBOOL | Returns the common input mode: <ul style="list-style-type: none"> • 0 = FALSE: Common input mode is disabled. • 1 = TRUE: Common input mode is enabled. |
| <i>pnActiveInputChannel</i> | PSHORT | Specified active channel for the common input: <ul style="list-style-type: none"> • 0 = GXCNT_CHANNEL_A: Channel A • 1 = GXCNT_CHANNEL_B: Channel B. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

In the common input mode, *nActiveInputChannel* input is connected internally to both channels.

Enabling this mode allows the user to make measurements. E.g., measure time interval between two events on the same input signal without the need to connect both inputs.

NOTE: There is a restriction when using **GxCntSetCommonInput**. When using a common input, the trigger slopes for channel A and channel B cannot be the same.

Example

The following example returns the common input mode state and active input channel:

```
SHORT nActiveInputChannel;
BOOL bEnable;
GxCntGetCommonInput(nHandle, &bEnable, &nActiveInputChannel, &nStatus);
```

See Also

GxCntSetCommonInput, **GxCntGetErrorString**

GxCntGetCounterRefClockToPxiRefClockState

Applies To

GTX2220, GTX2230

Purpose

Returns the connection state of the Counter Reference Clock to the PXI Reference Clock.

Syntax

GxCntGetCounterRefClockToPxiRefClockState (*nHandle*, *pnState*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|---|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>pnState</i> | PSHORT | Returned the Counter Reference Clock to PXI Reference Clock state: <ul style="list-style-type: none"> • 0 = GXCNT_CLOCK_TO_PXI_REF_CLOCK_OFF: Disconnect the Counter Reference Clock from the PXI Reference Clock. • 1 = GXCNT_CLOCK_TO_PXI_REF_CLOCK_ON: Connect the Counter Reference Clock from the PXI Reference Clock. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

The PXI chassis has a 10 MHz clock line (PXI_CLK10) in its backplane that allows devices plugged into the non-star controller slots (slots 3 and higher) to phase-lock their oscillators to the backplane clock. A single backplane clock can control this way the drift in the oscillators on all the devices in the non-star controller slots. The GTX2220/30 board when plugged into the star-controller slot (slot 2) can replace the PXI backplane clock with its much more stable oscillator. Alternatively, when plugged into one of the non-star controller slots (slots 3 and higher), the GTX2220/30 board can be programmed that its clock source will use the PXI backplane clock see the GxCntSetClockSource function for details.

This call is valid only if the GTX2220/30 board is plugged into the star-controller slot (slot 2) of the PXI chassis.

Example

The following example returns the connection state of the Counter Reference Clock to the PXI Reference Clock:

```
SHORT nState;
GxCntGetCounterRefClockToPxiRefClockState (nHandle, &nState, &nStatus);
```

See Also

GxCntSetCounterRefClockToPxiRefClockState, **GxCntSetClockSource**, **GxCntGetClockSource**, **GxCntGetErrorString**

GxCntGetDriverSummary

Purpose

Returns the driver description string and version number.

Syntax

GxCntGetDriverSummary (*pszSummary*, *nSummaryMaxLen*, *pdwVersion*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------------|---------|--|
| <i>pszSummary</i> | LPSTR | Buffer to receive the summary string. |
| <i>nSummaryMaxLen</i> | SHORT | Buffer size passed by pszSummary. |
| <i>pdwVersion</i> | LPDWORD | Driver version |
| <i>pnStatus</i> | LPSHORT | Returned status: 0 on success, negative number on failure. |

Example

The following example returns the driver summary:

```
SHORT nHandle, nStatus;
DWORD dwVersion;
CHAR szSummary[128];
GxCntGetDriverSummary(szSummary, 128, &dwVersion, &nStatus);
After the function call the parameter SzSummary will be set to:
"GXCNT Driver for GC2210, GC2220, GC2230, GX2210, GX2220 and GX2230, Version 2.10, Copyright(c)
2006 Marvin Test Solutions."
```

See Also

GxCntGetErrorString

GxCntGetErrorString

Purpose

Returns the error string associated with the specified error number.

Syntax

GxCntGetErrorString (*nError*, *pszMsg*, *nErrorMaxLen*, *pnStatus*)

Parameters

| Name | Type | Comments |
|---------------------|--------|--|
| <i>nError</i> | SHORT | Error number. |
| <i>pszMsg</i> | PSTR | Buffer to the returned error string. |
| <i>nErrorMaxLen</i> | SHORT | The size of the error string buffer. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

The function returns the error string associated with the *nError* as returned from other driver functions by the *pnStatus* parameter.

The following table displays the possible error values; not all errors apply to this board type:

Warnings

- 12 Measurement Value is out of range
- 11 Unable to find suitable filter value AutoFilter mode
- 10 Measurement timeout, no signal was detected. Timeout value needs to be greater than Gate Time.

Resource Errors

- 0 No error has occurred
- 1 Unable to open the HW driver. Check if HW is properly installed
- 2 Board does not exist in this slot/base address
- 3 Different board exist in the specified PCI slot/base address
- 4 PCI slot not configured properly. You may configure using the PciExplorer from the Windows Control Panel
- 5 Unable to register the PCI device
- 6 Unable to allocate system resource for the device
- 7 Unable to allocate memory
- 8 Unable to create panel
- 9 Unable to create Windows timer
- 10 Bad or Wrong board EEPROM
- 11 Not in calibration mode
- 12 Board is not calibrated
- 13 Function is not supported by the specified board

General Parameter Errors

- 20 Invalid or unknown error number
- 21 Invalid parameter
- 22 Illegal slot number
- 23 Illegal board handle
- 24 Illegal string length
- 25 Illegal operation mode

VISA Errors

- 30 Unable to Load VISA32/64.DLL, make sure VISA library is installed
- 31 Unable to open default VISA resource manager, make sure VISA is properly installed
- 32 Unable to open the specified VISA resource
- 33 VISA viGetAttribute error
- 34 VISA viInXX error
- 35 VISA ViMapAddress error

Misc Errors

- 38 Unable to lock a board resource, resource is used by another process or thread

Parameter Errors

- 40 Invalid channel number (0 or 1)
- 41 Invalid operation mode
- 42 Invalid value, value is out of range
- 43 Invalid trigger level voltage, voltage should be between -5.12V and +5.12V
- 44 The specified function call is not applicable to Counter Function settings
- 45 Timeout occur while waiting for measurement

Board/execution errors

- 50 Parameter range error
- 51 Error send command handshake timeout
- 52 Datacom error with board
- 53 Recv51 was aborted
- 54 Measurement timeout, no signal was detected
- 55 Timeout on stats byte
- 56 Error Last byte recv.
- 57 Cannot reinitialize, stuck in infinite loop
- 58 Timeout occur while waiting for measurement

Calibration errors

- 70 Calibration did not started
- 71 Invalid calibration device number, device number are between 0 to 2
- 72 Time base calibration timeout, unable to detect input signal

- 73 Error Channel A Trigger Level calibration
- 74 Error Channel B Trigger Level calibration
- 75 Error Time base calibration frequency
- 76 Error Channel B Trigger Level calibration
- 75 Error Time Base input Frequency
- 76 Error: unable to calibrate Time Base, the Time Base oscillator is out of range

Example

The following example initializes the board. If the initialization failed, the following error string is printed:

```
CHAR    sz[256];
SHORT   nStatus, nHandle;
..
GxCntInitialize (3, &Handle, &Status);
if (nStatus<0)
{   GxCntGetErrorString(nStatus, sz, sizeof sz, &nStatus);
    printf(sz); // prints the error string returns
}
```

GxCntGetExtendedSerialNumber

Purpose

Returns the Extended board's Serial Number information.

Syntax

GxCntGetExtendedSerialNumber (*nHandle*, *pszSerialNum*, *nSerialNumMaxLen*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-------------------------|--------|---|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>pszSerialNum</i> | PSTR | Buffer to contain the returned Extended board's Serial Number string. |
| <i>nSerialNumMaxLen</i> | SHORT | Size of the buffer to contain the error string. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

The returned Extended board's Serial Number string provides information regarding hardware changes and modifications of the specified board. The information can be used by Marvin Test Solutions's Customer Support department and by the user in order to determine what functionalities are supported by the board.

For example, the returned string could look like the following: "GTX22100154-CC-CA-10".

Example

The following example returns the board Extended board's Serial Number information:

```
SHORT nHandle, nStatus;
CHAR szSummary [64];

GxCntGetExtendedSerialNumber (nHandle, szSummary, sizeof(szSummary), &nStatus);
```

See Also

GxCntGetBoardSummary, **GxCntGetDriverSummary**, **GxCntInitialize**, **GxCntGetErrorString**

GxCntGetFunction

Purpose

Returns the function and function settings.

Syntax

GxCntGetFunction (*nHandle*, *pnFunction*, *pnChannelMode*, *pnStartSlope*, *pnStopSlope*, *pnStatus*)

Parameters

| Name | Type | Comments |
|----------------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>pnFunction</i> | PSHORT | Returned function number see comments for details. |
| <i>pnChannelMode</i> | PSHORT | Returned channels or Mode depends on the returned function value. See comments for details. |
| <i>pnStartSlope</i> | PSHORT | Returned measurement start slope depends on the returned function value. See comments for details. |
| <i>pnStopSlope</i> | PSHORT | Returned measurement stop slope depends on the returned function value. See comments for details. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. See comments for details. |

Comments

Returned function is as follow:

- 0 GXCNT_FUNCTION_ACCUMULATE:
- 1 GXCNT_FUNCTION_AUTO_RATIO:
- 2 GXCNT_FUNCTION_FAST_FREQUENCY:
- 3 GXCNT_FUNCTION_FREQUENCY:
- 4 GXCNT_FUNCTION_PERIOD:
- 5 GXCNT_FUNCTION_PULSE_WIDTH
- 6 GXCNT_FUNCTION_RATIO:
- 7 GXCNT_FUNCTION_SINGLE_PERIOD:
- 8 GXCNT_FUNCTION_TEST_CLOCK:
- 9 GXCNT_FUNCTION_TIME_INTERVAL:
- 10 GXCNT_FUNCTION_TIME_INTERVAL_DELAY:
- 11 GXCNT_FUNCTION_TOTALIZE:
- 12 GXCNT_FUNCTION_TOTALIZE_GATED:
- 13 GXCNT_FUNCTION_TOTALIZE_GATED_ONCE:

Returned measurement start slope is as follow:

- 0 GXCNT_POSITIVE_SLOPE: Rising (**Positive**) edge.
- 1 GXCNT_NEGATIVE_SLOPE: Falling (**Negative**) edge.

Returned measurement stop slope is as follow:

- 0 GXCNT_POSITIVE_SLOPE: Rising (**Positive**) edge.
- 1 GXCNT_NEGATIVE_SLOPE: Falling (**Negative**) edge.

Example

The following example returns the function and function settings:

```
SHORT nHandle, nStatus;  
SHORT nFunction, nChannelMode, nStartSlope, nStopSlope  
GxCntGetFunction (nHandle, &nFunction, &nChannelMode, &nStartSlope, &nStopSlope, &nStatus);
```

See Also

GxCntSetAcquisitionTimeInterval, GxCntSetAcquisitionMode, GxCntGetAcquisitionMode, GxCntGetErrorString

GxCntGetGateTime

Purpose

Returns the gate time.

Syntax

GxCntGetGateTime (*nHandle*, *pdSeconds*, *pnStatus*)

Parameters

| Name | Type | Comments |
|------------------|---------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>pdSeconds</i> | PDOUBLE | Returned gate time in seconds. Gate time range is 0.250 μ S to 3200 seconds with a resolution of 0.75 μ S. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

For **Frequency**, **Period**, **Ratio** and **GxCntSetFunctionTestInternalClock** functions, the selected **Gate** time sets the minimum measurement interval. It is ignored by all other functions.

Example

The following example returns the gate time:

```
SHORT nHandle, nStatus;
DOUBLE dSeconds
GxCntGetGateTime (nHandle, &dSeconds, &nStatus)
```

See Also

GxCntSetGateTime, **GxCntSetAcquisitionTimeInterval**, **GxCntGetAcquisitionTimeInterval**, **GxCntSetAcquisitionMode**, **GxCntGetAcquisitionMode**, **GxCntGetErrorString**

GxCntGetMeasurementNumberOfDigits

Purpose

Returns the measurement number of digits and mode.

Syntax

GxCntGetMeasurementNumberOfDigits (*nHandle*, *pnMode*, *pnMaxDigits*, *pnStatus*)

Parameters

| Name | Type | Comments |
|--------------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>pnMode</i> | PSHORT | Measurement number of digits modes are: <ol style="list-style-type: none"> 0. GXCNT_MEASURE_NUM_OF_DIGITS_AUTO: the counter automatically sets the number of digits according to the gate time. 1. GXCNT_MEASURE_NUM_OF_DIGITS_FIXED: user overrides the counter's automatic measure number of digits settings, all measurement will have the same number of digits as was set by the <i>nMaxDigits</i> parameter. |
| <i>pnMaxDigits</i> | PSHORT | If Measurement number of digits mode was set to GXCNT_MEASURE_NUM_OF_DIGITS_AUTO: returns the last completed measurements number of digits as was set internally by the counter. If Measurement number of digits mode was set to GXCNT_MEASURE_NUM_OF_DIGITS_FIXED: returns the users specified number of digits. Number of digits can be between 5 to 14 |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

The function allows the use to either format the returned measurement value to a specified number of digits or increase the total number of digits when taking measurements with long gate time in order to get more than maximum 11 digits.

Note: Gx2210 and GC2210 will not support number of digits greater than 9.

Example

The following example returns measurement number of digits and mode settings:

```
SHORT nMode, nMaxDigits;
GxCntGetMeasurementNumberOfDigits(nHandle, &nMode, &nMaxDigits, &nStatus)
```

See Also

GxCntSetMeasurementNumberOfDigits, **GxCntSetMeasurementTimeout**, **GxCntGetFunction**, **GxCntGetErrorString**

GxCntGetMeasurementTimeout

Purpose

Returns the time allowable for a measurement read operation.

Syntax

GxCntGetMeasurementTimeout (*nHandle*, *pdSeconds*, *pnStatus*)

Parameters

| Name | Type | Comments |
|------------------|---------|---|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>pdSeconds</i> | PDOUBLE | Returns timeout value in seconds, timeout range is 1mS to 3200 seconds. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

In most measurement functions, the counter will wait indefinitely for a measurement to end if the input signal is removed or changes amplitude. If an attempt is made to read data from the board under this condition, the host computer will wait until the data is available. This lockup situation can be avoided by programming the timeout to an interval that is longer than any expected measurement, but will still allow the program to regain control if signal is absent or changes significantly from the expected amplitude.

In cases of measurements with long gate times or low frequency input signals and there is a potential problem, the **GxCntIsMeasurementReady** or **GxCntReadStatusRegister** functions can be used to see if data is ready and help to avoid unnecessary time-outs.

Example

The following example returns the timeout settings:

```
SHORT nHandle, nStatus;
DOUBLE dSeconds
GxCntGetMeasurementTimeout (nHandle, &dSeconds, &nStatus)
```

See Also

GxCntSetMeasurementTimeout, **GxCntGetFunction**, **GxCntGetErrorString**

GxCntGetPrescalerMode

Purpose

Returns the pre-scale mode for both channels.

Syntax

GxCntGetPrescalerMode (*nHandle*, *pnMode*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|---|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>pnMode</i> | PSHORT | Returns the pre-scale mode: <ul style="list-style-type: none"> • 0 = GXCNT_PRESCALE_OFF: Prescaling set to off. • 1 = GXCNT_PRESCALE_ON: Prescaling set to on, the counter continuously prescales the actual measurement. • 2 = GXCNT_PRESCALE_AUTO: The counter determines the need for prescaling automatically by making a quick frequency check prior to the actual measurement. The process takes about 20 μS. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

Prescaling is necessary when the input frequency exceeds 10MHz in **Frequency**, **Period**, **Ratio**, and **AutoRatio** modes (see specifications). Prescaling does not affect resolution or accuracy in any way.

The by the user, or it can be set to auto mode.

If the input frequency exceeds 10MHz and arming is enabled, the prescaler must be explicitly turned on. The prescaler reverts to the off state whenever arming is enabled, to reduce potential measurement start point ambiguity (up to 4 Acquisitions of the input signal may pass before the counter starts).

Example

The following example returns the prescale mode:

```
SHORT nHandle, nMode, nStatus;
GxCntGetPrescalerMode (nHandle, &nMode, &nStatus)
```

See Also

GxCntSetGateTime, **GxCntSetAcquisitionTimeInterval**, **GxCntGetAcquisitionTimeInterval**, **GxCntSetAcquisitionMode**, **GxCntGetAcquisitionMode**, **GxCntGetErrorString**

GxCntGetTimeIntervalDelay

Purpose

Returns the delay time for the **GxCntSetFunctionTimeIntervalDelay** function.

Syntax

GxCntGetTimeIntervalDelay (*nHandle*, *pdSeconds*, *pnStatus*)

Parameters

| Name | Type | Comments |
|------------------|---------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>pdSeconds</i> | PDOUBLE | Returns delay time in seconds, delay time range is 20 μ S to 3200 seconds. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

The delay time is used as a special mode to disable or “hold off” measurement completion to ease such difficult measurement setups. The delay time is only used by the **GxCntSetFunctionTimeIntervalDelay** function.

Example

The following example returns the prescale mode:

```
SHORT nHandle, nStatus;
DOUBLE dSeconds
GxCntGetTimeIntervalDelay (nHandle, &dSeconds, &nStatus)
```

See Also

GxCntSetTimeIntervalDelay, **GxCntGetFunction**, **GxCntGetErrorString**

GxCntGetTotalizeGateMode

Purpose

Returns the Totalize Gate mode.

Syntax

GxCntGetTotalizeGateMode (*nHandle*, *pnMode*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|---|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>pnMode</i> | PSHORT | Returns Gate mode: <ul style="list-style-type: none"> • 0 = GXCNT_TOTALIZE_GATE_OPEN: open the count Gate. • 1 = GXCNT_TOTALIZE_GATE_CLOSE: close the count Gate. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

This function enables or disables the count Gate when in **Totalize** mode.

Example

The following example returns the Totalize Gate mode:

```
SHORT nHandle, nMode, nStatus;
GxCntGetTotalizeGateMode (nHandle, &nMode, &nStatus)
```

See Also

GxCntSetTotalizeGateMode, **GxCntGetFunction**, **GxCntGetErrorString**

GxCntGetTriggerSlope

Purpose

Returns the trigger slope.

Syntax

GxCntGetTriggerSlope (*nHandle*, *pnSlope*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>pnSlope</i> | PSHORT | Returns trigger slope: <ul style="list-style-type: none"> • 0 = GXCNT_POSITIVE_SLOPE: Rising (Positive) edge. • 1 = GXCNT_NEGATIVE_SLOPE: Falling (Negative) edge. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

The trigger slope setting is active only when the trigger source is set to external, see **GxCntSetTriggerSource** function for details.

Example

The following example returns the trigger slope:

```
SHORT nHandle, nSlope, nStatus;
GxCntGetTriggerSlope (nHandle, &nSlope, &nStatus)
```

See Also

GxCntSetTriggerSlope, **GxCntSetTriggerSource**, **GxCntGetTriggerSource**, **GxCntGetFunction**, **GxCntGetErrorString**

GxCntGetTriggerSource

Purpose

Returns the trigger source.

Syntax

GxCntGetTriggerSource (*nHandle*, *pnSource*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>pnSource</i> | PSHORT | Returns trigger source: <ul style="list-style-type: none"> • 0 = GXCNT_TRIGGER_INTERNAL: Software triggered, initiates the first measurement immediately. • 1 = GXCNT_TRIGGER_EXTERNAL: The first measurement is delayed until the signal edge specified in GxCntSetTriggerSlope is received. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Example

The following example returns the trigger source:

```
SHORT nHandle, nSource, nStatus;
GxCntGetTriggerSource (nHandle, &nSource, &nStatus)
```

See Also

GxCntGetTriggerSource, **GxCntSetTriggerSlope**, **GxCntGetTriggerSlope**, **GxCntGetFunction**, **GxCntGetErrorString**

GxCntInitialize

Purpose

Initializes the driver for the specified PXI slot using the HW device driver.

Syntax

GxCntInitialize (*nSlot*, *pnHandle*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|--|
| <i>nSlot</i> | SHORT | Counter board slot number. See Comments. |
| <i>pnHandle</i> | PSHORT | Returned Handle for a Counter board. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, 1 on failure. |

Comments

The Marvin Test Solutions HW device driver is installed with the driver and is the default device driver. The function returns a handle that for use with other Counter functions to program the board. The function does not change any of the board settings.

The specified PXI slot number is displayed by the **PXI/PCI Explorer** applet that can be opened from the Windows **Control Panel**. You may also use the label on the chassis below the PXI slot where the board is installed. The function accepts two types of slot numbers:

- A combination of chassis number (chassis # x 256) with the chassis slot number. For example 0x105 (chassis 1 slot 5).
- Legacy nSlot as used by earlier versions of HW/VISA. The slot number contains no chassis number and can be changed using the **PXI/PCI Explorer** applet (1-255).

Example

The following example initializes a Counter board at PXI chassis 2 slot 7.

```
SHORT nHandle, nStatus;
GxCntInitialize (0x207, &nHandle, &nStatus);
if (nHandle==0)
{
    printf("Unable to Initialize the board")
    return;
}
```

See Also

GxCntInitalizeVisa, **GxCntGetErrorString**, **GxCntReset**

GxCntInitializeVisa

Purpose

Initializes the driver for the specified PXI slot using the default VISA provider.

Syntax

GxCntInitializeVisa (*szVisaResource*, *pnHandle*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------------|---------|---|
| <i>szVisaResource</i> | LPCTSTR | String identifying the location of the specified board in order to establish a session. |
| <i>pnHandle</i> | PSHORT | Returned Handle (session identifier) that can be used to call any other operations of that resource |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, 1 on failure. |

Comments

The **GxCntInitializeVisa** opens a VISA session to the specified resource. The function uses the default VISA provider configured in your system to access the board. You must ensure that the default VISA provider support PXI/PCI devices and that the board is visible in the VISA resource manager before calling this function.

The first argument *szVisaResource* is a string that is displayed by the VISA resource manager such as NI Measurement and Automation (NI_MAX). It is also displayed by Marvin Test Solutions PXI/PCI Explorer as shown in the prior figure. The VISA resource string can be specified in several ways as follows:

- Using chassis, slot: "PXI0::CHASSIS1::SLOT5"
- Using the PCI Bus/Device combination: "PXI9::13::INSTR" (bus 9, device 9).
- Using alias: "COUNTER1". Use the PXI/PCI Explorer to set the device alias.

The function returns a board handle (session identifier) that can be used to call any other operations of that resource. The session is opened with `VI_TMO_IMMEDIATE` and `VI_NO_LOCK` VISA attributes. On terminating the application the driver automatically invokes `viClose()` terminating the session.

Example

The following example initializes a Counter boards at PXI bus 5 and device 11.

```
SHORT nHandle, nStatus;
GxCntInitializeVisa ("PXI5::11::INSTR", &nHandle, &nStatus);
if (nHandle==0)
{
    printf("Unable to Initialize the board")
    return;
}
```

See Also

GxCntInitialize, **GxCntGetErrorString**, **GxCntReset**

GxCntInSystemCalDevice

Purpose

Calibrate the specified device.

Syntax

GxCntInSystemCalDevice (*nHandle*, *nDevice*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nDevice</i> | SHORT | Specified device number: <ul style="list-style-type: none"> • 0 = GX2200_CAL_TIME_BASE: Time base • 1 = GX2200_CAL_TRIG_LEVEL_CH_A: Trigger level Channel A. • 2 = GX2200_CAL_TRIG_LEVEL_CH_B: Trigger level Channel B. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

Calling this function calibrate the specified device. The function is available only if the Calibration option was purchased and installed. Before calling this function the following settings should be taken:

Time base calibration:

Prior to calling this function with the time base device number connects an input signal with the following characteristics:

- Connect 10MHz reference signal to Channel A Input.
- Reference signal should be 10MHz \pm 0.01PPM sine wave with amplitude of 1 ± 0.3 Vrms into 50 ohms and less than -35dB total harmonics distortions.
- Allow at least 30 minutes of warm-up time before starting calibration.

Trigger level Channel A

Prior to calling this function with the Trigger level Channel A device number disconnect the input channel

Trigger level Channel B:

Prior to calling this function with the Trigger level Channel B device number disconnect the input channel

The **GxCntInSystemCalStart** needs to be called prior calling this function in order to start the calibration process.

Calling **GxCntInSystemCalGetStatus** returns the specified device calibration status.

See **Running In-System calibration from the virtual panel** and **In-System Calibration Example program** sections for details.

Example

The following example calibrates the Time Base:

```
SHORT nHandle, nStatus;
GxCntInSystemCalDevice (nHandle, GX2200_CAL_TIME_BASE, &nStatus)
```

See Also

GxCntInSystemCalGetStatus, **GxCntInSystemCalRestore**, **GxCntInSystemCalSaveData**, **GxCntInSystemCalStart**, **GxCntGetErrorString**

GxCntInSystemCalGetStatus

Purpose

Returns the specified device calibration status.

Syntax

GxCntInSystemCalGetStatus (*nHandle*, *nDevice*, *pnCalStatus*, *pnStatus*)

Parameters

| Name | Type | Comments |
|--------------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nDevice</i> | SHORT | Specified device number: <ul style="list-style-type: none"> • 0 = GX2200_CAL_TIME_BASE: Time base • 1 = GX2200_CAL_TRIG_LEVEL_CH_A: Trigger level Channel A. • 2 = GX2200_CAL_TRIG_LEVEL_CH_B: Trigger level Channel B. |
| <i>pnCalStatus</i> | PSHORT | Device Calibration status can be as follows: <ul style="list-style-type: none"> • 0 = GX2200_DEVICE_NOT_CALIBRATED: The specified device was not calibrated yet. • 1 = GX2200_DEVICE_PASSED_CALIBRATION: The specified device passed calibration successfully. • 2 = GX2200_DEVICE_FAILED_CALIBRATION: The specified device failed calibration. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

Calling **GxCntInSystemCalGetStatus** returns the specified device calibration status. The function is available only if the Calibration option was purchased and installed.

See **Running In-System calibration from the virtual panel** and **In-System Calibration Example program** sections for details.

Example

The following example Auto sets channel A:

```
SHORT nHandle, nStatus;
GxCntChannelAutoSet(nHandle, GXCNT_CHANNEL_A, &nStatus)
```

See Also

GxCntInSystemCalDevice, **GxCntInSystemCalRestore**, **GxCntInSystemCalSaveData**, **GxCntInSystemCalStart**, **GxCntGetErrorString**

GxCntInSystemCalRestore

Purpose

Restore the specified device calibration to the factory calibration.

Syntax

GxCntInSystemCalRestore (*nHandle*, *nDevice*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nDevice</i> | SHORT | Specified device number: <ul style="list-style-type: none"> • 0 = GX2200_CAL_TIME_BASE: Time base • 1 = GX2200_CAL_TRIG_LEVEL_CH_A: Trigger level Channel A. • 2 = GX2200_CAL_TRIG_LEVEL_CH_B: Trigger level Channel B. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

The function overwrites the current device calibration data with the manufacture's device calibration data. The function is available only if the Calibration option was purchased and installed.

Calling **GxCntInSystemCalGetStatus** returns the specified device calibration status.

See **Running In-System calibration from the virtual panel** and **In-System Calibration Example program** sections for details.

Example

The following example restores the Time Base calibration data back to the manufacture data.

```
SHORT nHandle, nStatus;
GxCntInSystemCalRestore (nHandle, GX2200_CAL_TIME_BASE, &nStatus)
```

See Also

GxCntInSystemCalDevice, **GxCntInSystemCalGetStatus**, **GxCntInSystemCalSaveData**, **GxCntInSystemCalStart**, **GxCntGetErrorString**

GxCntInSystemCalSave

Purpose

Save calibrated devices data to the on-board EEPROM.

Syntax

GxCntInSystemCalSave (*nHandle*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

Calling this function will only save calibrated devices data to the on board EEPROM and will end the calibration process. Devices calibration data that were not calibrated, by calling **GxCntInSystemCalDevice** function, will not change. The function is available only if the Calibration option was purchased and installed.

The **GxCntInSystemCalStart** needs to be called prior calling this function in order to start the calibration process.

Call **GxCntInSystemCalGetStatus** to get a specified device calibration status.

See **Running In-System calibration from the virtual panel** and **In-System Calibration Example program** sections for details.

Example

The following example Stores the calibration data to the on-board EPROM.

```
SHORT nHandle, nStatus;
GxCntInSystemCalSave (nHandle, &nStatus)
```

See Also

GxCntInSystemCalDevice, **GxCntInSystemCalGetStatus**, **GxCntInSystemCalRestore**, **GxCntInSystemCalStart**, **GxCntGetErrorString**

GxCntInSystemCalStart

Purpose

Start the In-System calibration.

Syntax

GxCntInSystemCalStart (*nHandle*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

The **GxCntInSystemCalStart** start the calibration process. This function needs to be called prior to calibrating any of the devices. The function is available only if the Calibration option was purchased and installed.

Call **GxCntInSystemCalGetStatus** to get a specified device calibration status.

See **Running In-System calibration from the virtual panel** and **In-System Calibration Example program** sections for details.

Example

The following example starts the calibration by user:

```
SHORT nHandle, nStatus;
GxCntInSystemCalStart (nHandle, &nStatus)
```

See Also

GxCntInSystemCalDevice, **GxCntInSystemCalGetStatus**, **GxCntInSystemCalRestore**, **GxCntInSystemCalSaveData**, **GxCntGetErrorString**

GxCntIsMeasurementReady

Purpose

Returns true if a new measurement is ready to read.

Syntax

GxCntIsMeasurementReady (*nHandle*, *pbReady*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>pbReady</i> | PBOOL | Returns status if a new measurement is ready to be read: <ul style="list-style-type: none"> • 0 = FALSE: no new measurement. • 1 = TRUE: new measurement is ready to read. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

Consecutive calls to this function will return FALSE after reading the measurement once if the counter did not acquire new measurement.

Example

The following example returns if a new measurement is ready to read:

```
SHORT nHandle, nStatus;
BOOL bReady
GxCntIsMeasurementReady (nHandle, &bReady, &nStatus);
```

See Also

GxCntReadMeasurement, **GxCntReadMeasurementString**, **GxCntReadStatusRegister**, **GxCntGetFunction**, **GxCntGetErrorString**

GxCntPanel

Purpose

Opens a virtual panel used to interactively control the counter.

Syntax

GxCntPanel (*pnHandle*, *hwndParent*, *nMode*, *phwndPanel*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-------------------|--------|---|
| <i>pnHandle</i> | PSHORT | Handle to a counter board. |
| <i>hwndParent</i> | HWND | Panel parent window handle. A value of 0 sets the desktop as the parent window. |
| <i>nMode</i> | SHORT | The mode in which the panel main window is created. 0 for modeless window and 1 for modal window. |
| <i>phwndPanel</i> | HWND | Returned window handle for the panel. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

The function is used to create the panel window. The panel window may be open as a modal or a modeless window depending on the *nMode* parameters.

If the mode is set to modal dialog (*nMode*=1), the panel will disable the parent window (*hwndParent*) and the function will return only after the window closes. In that case, the *pnHandle* may return the handle created by the user using the panel Initialize dialog. This handle may be used when calling other counter functions.

If a modeless dialog was created (*nMode*=0), the function returns immediately after creating the panel window returning the handle to the panel - *phwndPanel*. It is the responsibility of calling program to dispatch windows messages to this window so that the window can respond to messages.

Example

The following example opens the panel in modal mode:

```
DWORD dwPanel;
SHORT nHandle=0, nStatus;

GxCntPanel(&nHandle, 0, 1, &dwPanel, &nStatus);
```

See Also

GxCntInitialize, **GxCntGetErrorString**

GxCntReadMeasurement

Purpose

Read a single measurement.

Syntax

GxCntReadMeasurement (*nHandle*, *pdMeasurement*, *pnStatus*)

Parameters

| Name | Type | Comments |
|----------------------|---------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>pdMeasurement</i> | PDOUBLE | Returned measurement. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

This function is for reading measurement results out of the board. If a result is available it is returned immediately, otherwise the function will wait for a measurement to become available. Use **GxCntIsMeasurementReady** to prevent the software from “hanging up” if the signal is lost or for slow signals in order to ensure that measurement data is ready before attempting to read.

NOTE: A measurement value that is equal to -1 indicates that the measurement value is out of the board’s measuring range, e.g. measuring a value of 1.4GHz using a GC2220/GTX2220.

Example

The following example read a single measurement:

```
SHORT nHandle, nStatus;
DOUBLE dMeasurement;
GxCntReadMeasurement (nHandle, &dMeasurement, &nStatus);
```

See Also

GxCntReadMeasurementArray, **GxCntIsMeasurementReady**, **GxCntReadMeasurementString**, **GxCntReadStatusRegister**, **GxCntGetFunction**, **GxCntGetErrorString**

GxCntReadMeasurementArray

Purpose

Returns specified number of measurements into an array while specifying the amount of time allowed for filling the array before retuning.

Syntax

GxCntReadMeasurementArray (*nHandle*, *pdMeasurementBuff*, *pdwCount*, *dTimeout*, *pnStatus*)

Parameters

| Name | Type | Comments |
|---------------------------|---------|--|
| <i>NHandle</i> | SHORT | Handle to a counter board. |
| <i>psdMeasurementBuff</i> | PDOUBLE | Array to hold the returned measurements. |
| <i>pdwCount</i> | PDWORD | Number of measurements to read. Returning the actual number of measurements read. |
| <i>dTimeout</i> | DOUBLE | Specified the amount of time in mSec allowed for filling the buffer before retuning. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

This function is for reading an array of measurements utilizing the maximum throughput of the counter. The function can read measurement when in Fast Frequency, Frequency or Period mode. The GC2220/GC2230/GTX2220/GTX2230 boards can acquire up to 1400/sec theoretical measurements and around 1100/sec real measurements or better. When measurements are made using the Fast Frequency mode the GC2220/GC2230/GTX2220/GTX2230 boards can acquire up to 2300/sec theoretical measurements and around 1400/sec real measurements or better. The maximum number of measurements depends on the CPU speed, system configuration and number application running at the background.

The function returns upon completing reading the requested number of measurements or if the specified amount of time allowed for filling the buffer is up (whichever come first). The function then returns the actual number of reading that was taken.

The function returns upon completing reading the requested number of measurements or if the specified the amount of time allowed for filling the buffer is up (whichever come first).

Note: A measurement value that is equal to -1 indicates that the measurement value is out of the board's measuring range, e.g. measuring a value of 1.4GHz using a GC2220/GTX2220.

For best results, the counter settings should as follows:

- The Gate time needs to be set to the minimum (250usec).
- The channel Trigger Level Mode should be set to Fixed.
- The channel Trigger Level should be set to fixed value, e.g. 0.5V.
- The channel Prescale Mode should be turned off.

Example

The following example tries to read 100 measurements in 1Sec:

```
SHORT nHandle, nStatus;  
DWORD dwCount;  
DOUBLE dMeasurementBuff[100];  
  
dwCount=100;  
GxCntReadMeasurementArray (nHandle, adMeasurementBuff, &dwCount, 1000, &nStatus);  
printf ("Number of actual measurements=%d", dwCount);
```

See Also

**GxCntReadMeasurement, GxCntIsMeasurementReady, GxCntReadMeasurementString,
GxCntReadStatusRegister, GxCntGetFunction, GxCntGetErrorString**

GxCntReadMeasurementString

Purpose

Read a single measurement formatted as string.

Syntax

GxCntReadMeasurementString (*nHandle*, *szMeasurement*, *nMeasMaxLen*, *pnStatus*)

Parameters

| Name | Type | Comments |
|----------------------|---------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>szMeasurement</i> | PDOUBLE | Buffer to contain the Returned (null terminated) string. |
| <i>nMeasMaxLen</i> | SHORT | Size of the buffer to contain the error string. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

This function is used for reading measurement results out of the board. It returns the result both as a double precision floating point number, and as a formatted string. If a result is available it is returned immediately, otherwise the function will wait for a measurement to become available. Use **GxCntIsMeasurementReady** to prevent the software from “hanging up” if the signal is lost or for slow signals in order to ensure that measurement data is ready before attempting to read. See also **GxCntReadMeasurement**.

Example

The following example read a single measurement: and return the measurement formatted into a string:

```
SHORT nHandle, nStatus;
CHAR szMeasurement [256];

GxCntReadMeasurementString (nHandle, szMeasurement, sizeof(szMeasurement), &nStatus);
```

See Also

GxCntIsMeasurementReady, **GxCntReadMeasurement**, **GxCntReadStatusRegister**, **GxCntGetFunction**, **GxCntGetErrorString**

GxCntReadStatusRegister

Purpose

Read the counter status register.

Syntax

GxCntReadStatusRegister (*nHandle*, *pnData*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>pnData</i> | PSHORT | Returns status register value. See Comments. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

The status register is a 16-bit register (bits 0 – 15) configured as follows:

| | | | | | | | | | |
|------|-----|-----|-----|-----|-----|-----|-----|-----|------|
| 15-9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TBD | ODR | ERR | TBD | POV | TGD | TLB | TLA | ARM | GATE |

Where:

| Bit # | Bit name | Description |
|-------|----------|---|
| 0 | GATE | Gate is open no signal was detected. |
| 1 | ARM | Waiting for arm signal. High when arming is used, and the counter is waiting for an arming signal to start or to end the measurement. It is also high during the delay in the Time Interval with Delay mode. |
| 2 | TLA | Trigger Level A is settled. This is useful for making sure that the auto trigger already determines the trigger level before using the Hold option or reading out the trigger levels. |
| 3 | TLB | Trigger Level B is settled. See the description for the TLA above. |
| 4 | TGD | Gated Totalize measurement is Done. |
| 5 | POV | Pace Overrun Warning. A Acquisition Overrun occurs when a paced measurement does not complete soon enough to allow the next scheduled measurement to start on time. Skipping of measurements will occur, which may be of concern if it is necessary to know exactly when each data point was taken. |
| 6 | TBD | Reserved |
| 7 | ERR | An error has occurred. You can read the error code string with the <code>GTI_CTR_rd_err()</code> function. The error bit is cleared with the GxCntReset function. |
| 8 | ODR | Output Data Ready. The output buffer has measurement data or other data that you requested ready to be read by your program. You can check this bit before reading measurements to ensure that your program will never wait for data. |
| 9-15 | TBD | Reserved |

To determine if a bit is set you need to “mask” it by using the AND operator in the specific language used. Reading the status register does not disturb the measurement in progress or the measurement data.

Example

The following example counter status register:

```
SHORT nHandle, nData, nStatus;  
GxCntReadStatusRegister (nHandle, &nData, &nStatus);
```

See Also

**GxCntIsMeasurementReady, GxCntReadMeasurement, GxCntReadMeasurementString,
GxCntGetFunction, GxCntGetErrorString**

GxCntReset

Purpose

Resets the counter board to its default settings.

Syntax

GxCntReset (*nHandle*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|--|
| <i>nHandle</i> | SHORT | Handle for a counter board. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

Default settings are as follow:

- Arm Source: Off
- Arm Start: Off
- Arm Stop: Off
- Calibration: On
- Channel A frequency Mode: 200MHz
- Channels Slopes: Positive
- Clock source: Internal
- Acquisition Mode: Continuous
- Acquisition Time Interval: 1 S
- Delay: 1 mS
- Function: Frequency Channel A
- Gate Time: 300 mS
- Impedance: High
- Pace Start mode: Internal (GC2220/GC2230/GTX2220/GTX2230 Only)
- Prescaler: Auto
- Timeout: Off
- Trigger Levels Mode: Fixed
- Trigger Levels: 0V

Example

The following example initializes and resets the counter board:

```
GxCntInitialize (1, &nHandle, &nStatus);
GxCntReset (nHandle, &nStatus);
```

See Also

GxCntInitialize, **GxCntGetErrorString**

GxCntSelfTest

Purpose

Runs the self-test procedure on the board.

Syntax

GxCntSelfTest (*nHandle*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

The counter runs the following:

- On-board EPROM.
- Internal RAM in the microprocessor.
- Other checks.

The self-test takes about 60 ms for the GC2220/GC2230/GTX2220/GTX2230, and 100 ms for the GC2210/GTX2210.

Every time the power is turned on an extensive self-test is performed automatically.

Example

The following example run the board self test:

```
SHORT nHandle, nStatus;
GxCntSelfTest (nHandle, &nStatus);
```

See Also

GxCntReadStatusRegister, **GxCntGetFunction**, **GxCntGetErrorString**

GxCntSetAcquisitionMode

Purpose

Sets the board Acquisition mode.

Syntax

GxCntSetAcquisitionMode (*nHandle*, *nMode*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|---|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nMode</i> | SHORT | Acquisition mode: <ul style="list-style-type: none"> • 0 = GXCNT_ACQUISITION_CONTINUOUSLY: The counter continuously makes measurements. • 1 = GXCNT_ACQUISITION_SINGLE: Instrument makes a single measurement. Each Call to GxCntTrig initiates a new measurement. • 2 = GXCNT_ACQUISITION_PACE: The Pace mode ensures acquisition of data at accurately spaced time intervals. In Paced operation, the user specifies the time which should elapse BETWEEN initiations of measurements. See GxCntSetAcquisitionTimeInterval function for more details on the timing settings. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Example

The following example sets Acquisition to single mode:

```
SHORT nHandle, nStatus;
GxCntSetAcquisitionMode (nHandle, GXCNT_ACQUISITION_SINGLE, &nStatus)
```

See Also

GxCntGetAcquisitionMode, **GxCntGetErrorString**

GxCntSetAcquisitionTimeInterval

Purpose

Sets the board Acquisition time interval.

Syntax

GxCntSetAcquisitionTimeInterval (*nHandle*, *dSeconds*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|---|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>dSeconds</i> | DOUBLE | Acquisition time interval in seconds. Time interval range can be set from 0.8 ms to 3200 seconds. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

When setting the Acquisition mode to Paced mode (see **GxCntSetAcquisitionMode** function) ensures the acquisition of data at accurately spaced time intervals. In Paced operation, the user specifies the time which should elapse BETWEEN initiations of measurements. Acquisition time are digitally generated are accurate to within 200 μ s (the errors do not accumulate).

Some care must be used when selecting pacing intervals to ensure consistent timing between measurements. The pacing interval must be at least 1.5 ms longer than the actual gate time in Frequency and Period modes, 1.5 ms longer than the actual measurement interval in timing modes, such as Pulse Width. If these recommendations are not followed, the counter will simply "skip" measurements, and the data will be taken at uneven intervals. If skipping occurs in Acquire mode the function returns an error.

Example

The following example set the Acquisition time interval to 34mSec:

```
SHORT nHandle, nStatus;
GxCntSetAcquisitionTimeInterval (nHandle, 0.034, &nStatus)
```

See Also

GxCntGetAcquisitionTimeInterval, **GxCntSetAcquisitionMode**, **GxCntGetAcquisitionMode**,
GxCntGetErrorString

GxCntSetArmSlope

Purpose

Sets the Arm Start or Stop slope.

Syntax

GxCntSetArmSlope (*nHandle*, *nStartStop*, *nSlope*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-------------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nStartStop</i> | SHORT | Specify the Arm Start or Stop slope: <ul style="list-style-type: none"> • 0 = GXCNT_ARM_START_SLOPE: Arm Start slope • 1 = GXCNT_ARM_STOP_SLOPE: Arm Stop slope |
| <i>nSlope</i> | SHORT | Arm slope: <ul style="list-style-type: none"> • 0 = GXCNT_ARM_POSITIVE_SLOPE: Positive signal transition • 1 = GXCNT_ARM_NEGATIVE_SLOPE: Negative signal transition • 2 = GXCNT_ARM_DISABLE_SLOPE: Disables start/stop arming |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

The Arm start/stop slope controls the initiation and termination of the measurement. This setting is active only when the Arm source is set to external using the **GxCntSetArmSource** command.

Example

The following example sets the Arm starts slope to positive:

```
SHORT nHandle, nStatus;
GxCntSetArmSlope (nHandle, GXCNT_ARM_START_SLOPE, GXCNT_ARM_POSITIVE_SLOPE, &nStatus)
```

See Also

GxCntGetArmSlope, **GxCntSetArmSource**, **GxCntGetArmSource** , **GxCntGetErrorString**

GxCntSetArmSource

Purpose

Sets the Arm source

Syntax

GxCntSetArmSource (*nHandle*, *nSource*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nSource</i> | SHORT | Arm source: <ul style="list-style-type: none"> • 0 = GXCNT_ARM_INTERNAL: Arming measurements are done internally according to Acquisition mode. • 1 = GXCNT_ARM_EXTERNAL: The External Arm Input connector • 2 = GXCNT_ARM_ALTERNATE: The Alternate input channel (only for single channel measurements such as Frequency or Period). • 3 = GXCNT_ARM_OFF: Disable external Arm input, Measurements are armed internal to the counter board. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

The Arm source setting does not affect the **Totalize**, **Totalize Gated** and **GxCntSetFunctionTestInternalClock** functions.

Example

The following example sets the Arm source to the external input connector:

```
SHORT nHandle, nStatus;
GxCntSetArmSource (nHandle, GXCNT_ARM_EXTERNAL, &nStatus)
```

See Also

GxCntGetArmSlope, **GxCntSetArmSlope**, **GxCntGetArmSlope**, **GxCntGetErrorString**

GxCntSetCalibrationMode

Purpose

Sets the Calibration mode.

Syntax

GxCntSetCalibrationMode (*nHandle*, *nMode*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nMode</i> | SHORT | Calibration mode: <ul style="list-style-type: none"> • 0 = GXCNT_CAL_OFF: The current calibration factors are retained. Stable calibration factors preclude a possible shift in results during a period of data collection. • 1 = GXCNT_CAL_CONTINUOUS: Calibration is performed continuously approximately once every 60 seconds. Default settings. • 2 = GXCNT_CAL_ONCE: Selecting “Once” causes an immediate execution of the calibration procedure, and then disables future calibration. This mode ensures that data is taken with recent calibration factors, and prevents re-calibration from causing a systematic shift in results. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

The calibration mode controls how often the calibration routine is invoked.

Example

The following example sets the Calibration mode to on:

```
SHORT nHandle, nStatus;
GxCntSetCalibrationMode (nHandle, GXCNT_CAL_ON, &nStatus)
```

See Also

GxCntGetCalibrationMode, **GxCntGetErrorString**

GxCntSetChannelAFrequencyRange

Purpose

Sets channel A frequency range (GC2220/GC2230/GTX2220/GTX2230 only).

Syntax

GxCntSetChannelAFrequencyRange (*nHandle*, *nRange*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|---|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nRange</i> | SHORT | Channel A frequency range: <ul style="list-style-type: none"> • 0 = GXCNT_FREQUENCY_RANGE_NORMAL: Channel A frequency range is DC to 225MHz. • 1 = GXCNT_FREQUENCY_RANGE_HIGH: Channel A frequency range is 100MHz to 1.3GHz for GTX2220 or 2GHz for the GTX2230. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

When in high frequency range the following channel's settings are automatically set:

- Impedance is set to **1 MOhms**
- Coupling is set to **AC**.
- Filter mode is set to **OFF**

Example

The following example sets channel A frequency range to high frequency:

```
SHORT nHandle, nStatus;
GxCntSetChannelAFrequencyRange (nHandle, GXCNT_FREQUENCY_RANGE_HIGH, &nStatus)
```

See Also

GxCntGetChannelAFrequencyRange, **GxCntGetErrorString**

GxCntSetChannelCouplingMode

Purpose

Sets the specified coupling mode.

Syntax

GxCntSetChannelCouplingMode (*nHandle*, *nChannel*, *nMode*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nChannel</i> | SHORT | Specified Channel number: <ul style="list-style-type: none"> • 0 = GXCNT_CHANNEL_A: Channel A • 1 = GXCNT_CHANNEL_B: Channel B. |
| <i>nMode</i> | SHORT | Returned channel's input coupling mode: <ul style="list-style-type: none"> • 0 = GXCNT_COUPLING_AC: AC Coupling mode (default). • 1 = GXCNT_COUPLING_DC: DC Coupling mode. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Example

The following example returns channel A impedance mode:

```
SHORT nHandle, nMode, nStatus;
GxCntGetChannelImpedance (nHandle, GXCNT_CHANNEL_A, &nMode, &nStatus)
```

See Also

GxCntGetChannelCouplingMode, **GxCntGetErrorString**

GxCntSetChannelFilterMode

Purpose

Sets the specified channel filter mode.

Syntax

GxCntSetChannelFilterMode (*nHandle*, *nChannel*, *nMode*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|---|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nChannel</i> | SHORT | Specified Channel number: <ul style="list-style-type: none"> • 0 = GXCNT_CHANNEL_A: Channel A • 1 = GXCNT_CHANNEL_B: Channel B. |
| <i>nMode</i> | SHORT | Sets channel's filter mode: <ul style="list-style-type: none"> • 0 = GXCNT_FILTER_OFF: The filter is disabled. • 1 = GXCNT_FILTER_AUTO: In this mode the filter is enabled and the driver analyze the measured signal and determine the best filter value each time GxCntReadMeasurement or GxCntReadMeasurementString is called. • 2 = GXCNT_FILTER_VALUE_FIXED: In this mode the filter is enabled and the user sets the filter value see GxCntSetChannelFilterValue for details. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

The Filter should be enabled when measuring low frequencies (less than 20KHz) superimposed with a high-frequency noise. When enabled the Filter adds a delay in microseconds after first trigger is detected. The counter will then disregard any additional trigger events for the duration of the delay.

Example

The following example sets channel A filter mode to auto:

```
SHORT nHandle, nStatus;
GxCntGetChannelFilterMode (nHandle, GXCNT_CHANNEL_A, GXCNT_FILTER_AUTO, &nStatus)
```

See Also

GxCntGetChannelFilterMode, **GxCntGetErrorString**

GxCntSetChannelFilterValue

Purpose

Sets the specified channel filter value.

Syntax

GxCntSetChannelFilterValue (*nHandle*, *nChannel*, *nValue*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nChannel</i> | SHORT | Specified Channel number: <ul style="list-style-type: none"> • 0 = GXCNT_CHANNEL_A: Channel A • 1 = GXCNT_CHANNEL_B: Channel B. |
| <i>nValue</i> | SHORT | Sets the specified channel filter value. The filter value is an integer representing filter value in uSec, e.g. value of 50 represents 50uSec. Filter value range from 5 to 6400, (5uSec to 6400uSec). |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

The Filter should be used when measuring low frequencies (less than 20KHz) superimposed with a high-frequency noise. The Filter value adds a delay equal to *pdValue* in microseconds. After first trigger is detected the counter will disregard any additional trigger events for *pdValue* microseconds.

Note: The delay needs to be less than half of the expected frequency signal duty cycle. Filter value can be set when the filter mode is set to GXCNT_FILTER_VALUE_FIXED.

The filter value should be set according to the following equation:

$$\text{Filter Value (uS)} \leq \frac{1.0E + 6}{\text{ExpectedFrequency} * 4}$$

E.g.: if the expected value is about 10KHz than filter value should be around 25uSec.

Example

The following example sets channel A filter value to 55uSec :

```
SHORT nHandle, nStatus;
GxCntGetChannelFilterValue (nHandle, GXCNT_CHANNEL_A, 55, &nStatus)
```

See Also

GxCntGetChannelFilterValue, **GxCntGetErrorString**

GxCntSetChannelImpedance

Purpose

Sets the specified channel input impedance.

Syntax

GxCntSetChannelImpedance (*nHandle*, *nChannel*, *nMode*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nChannel</i> | SHORT | Specified Channel number: <ul style="list-style-type: none"> • 0 = GXCNT_CHANNEL_A: Channel A • 1 = GXCNT_CHANNEL_B: Channel B. |
| <i>nMode</i> | SHORT | Channel's input impedance mode: <ul style="list-style-type: none"> • 0 = GXCNT_IMPEDANCE_1MOHMS: Selects 1MOhm impedance (default). • 1 = GXCNT_IMPEDANCE_50OHMS: Selects 50 Ohms impedance. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Example

The following example sets channel A impedance to 1MOhm:

```
SHORT nHandle, nMode, nStatus;
GxCntSetChannelImpedance (nHandle, GXCNT_CHANNEL_A, GXCNT_IMPEDANCE_1MOhmS, &nStatus)
```

See Also

GxCntGetChannelImpedance, **GxCntGetErrorString**

GxCntSetChannelSlope

Purpose

Sets the specified channel input slope.

Syntax

GxCntSetChannelSlope (*nHandle*, *nChannel*, *nSlope*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nChannel</i> | SHORT | Specified Channel number: <ul style="list-style-type: none"> • 0 = GXCNT_CHANNEL_A: Channel A • 1 = GXCNT_CHANNEL_B: Channel B. |
| <i>nSlope</i> | SHORT | Channel's slope: <ul style="list-style-type: none"> • 0 = GXCNT_POSITIVE_SLOPE: Rising (Positive) edge. • 1 = GXCNT_NEGATIVE_SLOPE: Falling (Negative) edge. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

The channel slope sets the input signal edge that utilized for the measurement. The slope can specify either rising (**Positive**) edge or falling (**Negative**) edge.

Example

The following example sets channel A input slope to negative:

```
SHORT nHandle, nStatus;
GxCntGetChannelSlope (nHandle, GXCNT_CHANNEL_A, GXCNT_NEGATIVE_SLOPE, &nStatus)
```

See Also

GxCntGetChannelSlope, **GxCntGetErrorString**

GxCntSetChannelTriggerLevel

Purpose

Sets the specified channel input trigger level.

Syntax

GxCntSetChannelTriggerLevel (*nHandle*, *nChannel*, *dVoltage*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|---|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nChannel</i> | SHORT | Specified Channel number: <ul style="list-style-type: none"> • 0 = GXCNT_CHANNEL_A: Channel A • 1 = GXCNT_CHANNEL_B: Channel B. |
| <i>dVoltage</i> | DOUBLE | Channel's input trigger level voltage. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

The Trigger Level setting defines the input signal voltage that creates a logical event from the input comparator circuit (trigger level is also commonly known as “threshold” voltage). The trigger level can be set anywhere between +5.12V and -5.12V with 1mV resolution for PCB revision C and above (40mV for previous PCB). User entered values are rounded to the nearest least significant bit.

Trigger Level can be set automatically by the counter using or manually, see **GxCntSetChannelTriggerLevelMode** function for more details.

Example

The following example sets channel A Trigger Level to 1.24 volts:

```
SHORT nHandle, nStatus;
GxCntGetChannelTriggerLevel (nHandle, GXCNT_CHANNEL_A, 1.24, &nStatus)
```

See Also

GxCntGetChannelTriggerLevel, **GxCntSetChannelTriggerLevelMode**,
GxCntGetChannelTriggerLevelMode, **GxCntGetErrorString**

GxCntSetChannelTriggerLevelMode

Purpose

Sets the specified channel input trigger level mode.

Syntax

GxCntSetChannelTriggerLevelMode (*nHandle*, *nChannel*, *nMode*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nChannel</i> | SHORT | Specified Channel number: <ul style="list-style-type: none"> • 0 = GXCNT_CHANNEL_A: Channel A • 1 = GXCNT_CHANNEL_B: Channel B. |
| <i>nMode</i> | SHORT | Channel's input trigger level mode: <ul style="list-style-type: none"> • 0 = GXCNT_TRIGGER_LEVEL_FIXED • 1 = GXCNT_TRIGGER_LEVEL_AUTO • 2 = GXCNT_TRIGGER_LEVEL_HOLD_LAST See comments for details. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

The Trigger Level setting defines the input signal voltage that creates a logical event from the input comparator circuit (trigger level is also commonly known as “threshold” voltage). The trigger level can be set anywhere between +5.12V and -5.12V with 1mV resolution for PCB revision C and above (40mV for previous PCB). User entered values are rounded to the nearest least significant bit. See **GxCntSetChannelTriggerLevelMode** function for more details.

The following shows channel A and B values:

0. GXCNT_TRIGGER_LEVEL_FIXED: Sets the trigger level to the value displayed in the Set Level controls.
1. GXCNT_TRIGGER_LEVEL_AUTO: The board evaluates the peak input signal levels just before each measurement. The trigger levels are then automatically set to the approximate signal midpoint. Auto trigger may add up to 250 ms per measurement and is specified only for signal frequencies above 100 Hz. Auto trigger and Hold can be used together to provide most of the benefits of auto trigger without excessive measurement time.
2. GXCNT_TRIGGER_LEVEL_HOLD_LAST: The trigger level setting that was active when it was invoked. The primary application for “Hold” is to maintain an auto trigger set threshold over a number of measurements, without incurring the auto trigger time penalty on each measurement.

Example

The following example sets channel A Trigger Level mode to auto:

```
SHORT nHandle, nMode, nStatus;
DOUBLE dVoltage;
GxCntGetChannelTriggerLevelMode (nHandle, GXCNT_TRIGGER_LEVEL_AUTO, &dVoltage, &nStatus)
```

See Also

GxCntGetChannelTriggerLevelMode, GxCntSetChannelTriggerLevel, GxCntGetChannelTriggerLevel, GxCntGetErrorString

GxCntSetClockSource

Purpose

Sets the board clock source.

Syntax

GxCntSetClockSource (*nHandle*, *nSource*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nSource</i> | SHORT | Clock source: <ul style="list-style-type: none"> • 0 = GXCNT_CLOCK_INTERNAL: Selects the internal clock installed on the board. • 1 = GXCNT_CLOCK_EXTERNAL: External clock input as the clock. • 2 = GXCNT_CLOCK_ALTERNATE: When measuring only one channel the second (or alternate) channel can be defined as the reference frequency input. The main input channels provide better signal conditioning than the External Clock input. This mode affects only Frequency, FastFrequency, Period, Single Period and Width functions. • 3 = GXCNT_CLOCK_PXI_10MHZ_CLOCK: The PXI chassis backplane 10MHz clock line (PXI_CLK10) is the clock. Applied only to GTX2220/30 models. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

Settings the clock source to GXCNT_CLOCK_PXI_10MHZ_CLOCK (GTX2220/30 only):

The PXI chassis has a 10 MHz clock line (PXI_CLK10) in its backplane that allows devices plugged into the non-star controller slots (slots 3 and higher) to phase-lock their oscillators to the backplane clock. A single backplane clock can control this way the drift in the oscillators on all the devices in the non-star controller slots. The GTX2220/30 board, when plugged into the star-controller slot (slot 2) can replace the PXI backplane clock with its much more stable oscillator. Alternatively, when plugged into one of the non-star controller slots (slots 3 and higher), the GTX2220/30 board can be programmed that its clock source will use the PXI backplane clock.

Caution: the reference frequency must be within 5% of 10 MHz, or the resolution may be significantly reduced.

Example

The following example sets clock source to alternate:

```
SHORT nHandle, nStatus;
GxCntSetClockSource (nHandle, GXCNT_CLOCK_ALTERNATE, &nStatus)
```

See Also

GxCntGetClockSource, **GxCntGetErrorString**

GxCntSetCommonInput

Applies To

GTX2210, GTX2220, GTX2230

Purpose

Sets the common input mode state and active input channel.

Syntax

GxCntSetCommonInput (*nHandle*, *bEnable*, *nActiveInputChannel*, *pnStatus*)

Parameters

| Name | Type | Comments |
|----------------------------|--------|---|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>bEnable</i> | BOOL | Sets the common input mode: <ul style="list-style-type: none"> • 0 = FALSE: Common input mode is disabled. • 1 = TRUE: Common input mode is enabled. |
| <i>nActiveInputChannel</i> | SHORT | Specified active channel from the common input: <ul style="list-style-type: none"> • 0 = GXCNT_CHANNEL_A: Channel A • 1 = GXCNT_CHANNEL_B: Channel B. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

In the common input mode, *nActiveInputChannel* input is connected internally to both channels.

Enabling this mode allows the user to make measurements. E.g., measure time interval between two events on the same input signal without the need to connect both inputs.

Note: There is a restriction when using `GxCntSetCommonInput`. When using a common input, the trigger slopes for channel A and channel B cannot be the same.

Example

The following example enables the common input mode and set channel B as the active input channel:

```
GxCntSetCommonInput(nHandle, TRUE, GXCNT_CHANNEL_B: Channel B, &nStatus);
```

See Also

`GxCntGetCommonInput`, `GxCntGetErrorString`

GxCntSetCounterRefClockToPxiRefClockState

Applies To

GTX2220, GTX2230

Purpose

Sets the connection state of the Counter Reference Clock to the PXI Reference Clock.

Syntax

GxCntSetCounterRefClockToPxiRefClockState (*nHandle*, *nState*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nState</i> | SHORT | Counter Reference Clock to PXI Reference Clock state: <ul style="list-style-type: none"> • 0 = GXCNT_CLOCK_TO_PXI_REF_CLOCK_OFF: Disconnect the Counter Reference Clock from the PXI Reference Clock. • 1 = GXCNT_CLOCK_TO_PXI_REF_CLOCK_ON: Connect the Counter Reference Clock from the PXI Reference Clock. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

The PXI chassis has a 10 MHz clock line (PXI_CLK10) in its backplane that allows devices plugged into the non-star controller slots (slots 3 and higher) to phase-lock their oscillators to the backplane clock. A single backplane clock can control this way the drift in the oscillators on all the devices in the non-star controller slots. The GTX2220/30 board, when plugged into the star-controller slot (slot 2) can replace the PXI backplane clock with its much more stable oscillator. Alternatively, when plugged into one of the non-star controller slots (slots 3 and higher), the GTX2220/30 board can be programmed that its clock source will use the PXI backplane clock see the **GxCntSetClockSource** function for details.

This call is valid only if the GTX2220/30 board is plugged into the star-controller slot (slot 2) of the PXI chassis.

Example

The following example enables the connection from the Counter Reference clock to the PXI Reference clock:

```
GxCntSetCounterRefClockToPxiRefClockState (nHandle, GXCNT_CLOCK_TO_PXI_REF_CLOCK_ON, &nStatus);
```

See Also

GxCntGetCounterRefClockToPxiRefClockState, **GxCntSetClockSource**, **GxCntGetClockSource**, **GxCntGetErrorString**

GxCntSetFunctionAccumulate

Purpose

Sets the function mode to Accumulate.

Syntax

GxCntSetFunctionAccumulate (*nHandle*, *nMode*, *nStartSlope*, *nStopSlope*, *pnStatus*)

Parameters

| Name | Type | Comments |
|--------------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nMode</i> | SHORT | Measurement mode: <ul style="list-style-type: none"> • 0 = GXCNT_MEASURE_A_GATED_BY_B • 1 = GXCNT_MEASURE_B_GATED_BY_A |
| <i>nStartSlope</i> | SHORT | Measurement Start Slope: <ul style="list-style-type: none"> • 0 = GXCNT_POSITIVE_SLOPE: Rising (Positive) edge. • 1 = GXCNT_NEGATIVE_SLOPE: Falling (Negative) edge. |
| <i>nStopSlope</i> | SHORT | Measurement Stop Slope: <ul style="list-style-type: none"> • 0 = GXCNT_POSITIVE_SLOPE: Rising (Positive) edge. • 1 = GXCNT_NEGATIVE_SLOPE: Falling (Negative) edge. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

The Accumulate mode provides total count over multiple gate intervals. External signal determines the gate interval while the count gate can be opened and closed indefinite number of times. Less than 0.5 microsecond of “dead time” is required between the closing of one gate interval and the beginning of the next. Data can be accurately read at any time. The signal slopes that open and close the count gate specified by the *nStartSlope* and *nStopSlope*.

Note: Call **GxCntClear** function to clear the count and close the gate. The next gating signal edge will restart counting. If the counter is already running when this command is received, counting stops and the count is cleared. At least 0.5 μ s and one Acquisition of the counted signal must occur between the closing and reopening of the gate.

Example

The following example sets the function to Accumulate with channel A gated by channel B, negative start slope and positive stop slope:

```
SHORT nHandle, nStatus;
GxCntSetFunctionAccumulate (nHandle, GXCNT_MEASURE_A_GATED_BY_B, GXCNT_NEGATIVE_SLOPE,
GXCNT_POSITIVE_SLOPE, &nStatus);
```

See Also

GxCntSetFunctionTotalize, **GxCntSetFunctionTotalizeGated**, **GxCntSetFunctionTotalizeGatedOnce**, **GxCntGetErrorString**

GxCntSetFunctionAutoRatio

Purpose

Sets the function mode to Auto Ratio.

Syntax

GxCntSetFunctionAutoRatio (*nHandle*, *nMode*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nMode</i> | SHORT | Measurement Mode: <ul style="list-style-type: none"> • 0 = GXCNT_MEASURE_A_DIVIDED_BY_B: measure channel A divided by channel B. • 1 = GXCNT_MEASURE_B_DIVIDED_BY_A: measure channel B divided by channel A. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

This mode determines the ratio between the signal frequency on one input channel and the signal frequency on the second input channel. The measurement is performed during a user definable interval or gate time: (long gate time produce higher resolution results).

The counter tests the input signals and automatically selects the internal connections that maximize resolution. The measurement results are automatically converted to the specified mode (e.g. A/B or B/A).

Note: Only one of the input signals may exceed 25MHz and should not be used when signal frequencies drop below 400 Hz. If arming slope is specified the counter reverts to **GxCntSetFunctionRatio**.

Example

The following example sets the function to Auto Ratio and measuring channel A divided by channel B:

```
SHORT nHandle, nStatus;
GxCntSetFunctionAutoRatio (nHandle, GXCNT_MEASURE_A_DIVIDED_BY_B, &nStatus);
```

See Also

GxCntSetFunctionRatio, **GxCntSetGateTime**, **GxCntSetAcquisitionTimeInterval**, **GxCntGetErrorString**

GxCntSetFunctionFastFrequency

Purpose

Sets the function mode to Fast Frequency.

Syntax

GxCntSetFunctionFastFrequency (*nHandle*, *nChannel*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nChannel</i> | SHORT | Specified Channel number: <ul style="list-style-type: none"> • 0 = GXCNT_MEASURE_CHANNEL_A: measure channel A. • 1 = GXCNT_MEASURE_CHANNEL_B: measure channel B. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

Sets the mode to measure **FastFrequency** on the specified channel. This mode allows frequency measurement using short measurement interval enabling fast data acquisition. Since measurement resolution is proportional to the gate time low frequency signals are best suitable for this mode.

Note: Start arming can be used with **FastFrequency**, although Stop arming is not available.

Example

The following example sets the function to Fast Frequency measuring channel B:

```
SHORT nHandle, nStatus;
GxCntSetFunctionFastFrequency (nHandle, GXCNT_MEASURE_CHANNEL_B, &nStatus);
```

See Also

GxCntSetFunctionFrequency, **GxCntSetArmSource**, **GxCntSetArmSlope**, **GxCntGetErrorString**

GxCntSetFunctionFrequency

Purpose

Sets the function mode to Frequency.

Syntax

GxCntSetFunctionFrequency (*nHandle*, *nChannel*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nChannel</i> | SHORT | Specified Channel number: <ul style="list-style-type: none"> • 0 = GXCNT_MEASURE_CHANNEL_A: measure channel A. • 1 = GXCNT_MEASURE_CHANNEL_B: measure channel B. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

Sets the mode to measure Frequency on the specified channel. In this mode the measurement resolution in this mode is superior to that in **GxCntSetFunctionFastFrequency**. Signal frequencies can range from near DC to 100 MHz (see specification).

The user must specify the gate time (**GxCntSetGateTime**) over which the signal frequency is to be measured (long measurement times increase the number of significant digits).

Example

The following example sets the gate time to 1mS:

```
SHORT nHandle, nStatus;
GxCntSetFunctionFrequency (nHandle, 0.001, &nStatus)
```

See Also

GxCntSetFunctionFastFrequency, **GxCntSetGateTime**, **GxCntSetAcquisitionTimeInterval**, **GxCntSetAcquisitionMode**, **GxCntGetErrorString**

GxCntSetFunctionPeriod

Purpose

Sets the function mode to Period.

Syntax

GxCntSetFunctionPeriod (*nHandle*, *nChannel*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nChannel</i> | SHORT | Specified Channel number: <ul style="list-style-type: none"> • 0 = GXCNT_MEASURE_CHANNEL_A: measure channel A. • 1 = GXCNT_MEASURE_CHANNEL_B: measure channel B. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

Sets the mode to **Period** on the specified channel.

The signal period is measured in exactly the same way as the signal frequency, described above. The period is simply the reciprocal of the frequency, since $\text{Period} = 1/\text{Frequency}$.

Example

The following example sets the gate time to 1mS:

```
SHORT nHandle, nStatus;
GxCntSetFunctionPeriod (nHandle, GXCNT_MEASURE_CHANNEL_B, &nStatus);
```

See Also

GxCntSetFunctionFrequency, **GxCntSetGateTime**, **GxCntSetAcquisitionTimeInterval**, **GxCntSetAcquisitionMode**, **GxCntGetErrorString**

GxCntSetFunctionWidth

Purpose

Sets the function mode to Pulse Width.

Syntax

GxCntSetFunctionPulseWidth (*nHandle*, *nChannel*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nChannel</i> | SHORT | Specified Channel number: <ul style="list-style-type: none"> • 0 = GXCNT_MEASURE_CHANNEL_A: measure channel A. • 1 = GXCNT_MEASURE_CHANNEL_B: measure channel B. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

In this mode the counter measures pulse width on the specified channel. The pulse polarity (i.e. positive or negative) is set by **GxCntSetChannelSlope** function. Measurement resolution is 10nS for GC2210/GTX2210, and 100pS for GC2220/GC2230/GTX2220/GTX2230. Threshold levels can be set either to auto trigger **GxCntSetChannelTriggerLevelMode** or explicitly **GxCntSetChannelTriggerLevel**.

Example

The following example sets the function to measure pulse width on channel B:

```
SHORT nHandle, nStatus;
GxCntSetFunctionPulseWidth (nHandle, GXCNT_MEASURE_CHANNEL_B, &nStatus);
```

See Also

GxCntGetGateTime, **GxCntSetChannelSlope**, **GxCntSetTriggerLevelMode**, **GxCntSetTriggerLevel**, **GxCntGetErrorString**

GxCntSetFunctionRatio

Purpose

Sets the function mode to Ratio.

Syntax

GxCntSetFunctionRatio (*nHandle*, *nMode*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nMode</i> | SHORT | Measurement Mode: <ul style="list-style-type: none"> • 0 = GXCNT_MEASURE_A_DIVIDED_BY_B: measure channel A divided by channel B. • 1 = GXCNT_MEASURE_B_DIVIDED_BY_A: measure channel B divided by channel A. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

This mode determines the ratio between the signal frequency on one input channel and the signal frequency on the second input channel. The measurement is performed during a user definable interval or gate time: (long gate time produce higher resolution results).

The counter tests the input signals and automatically selects the internal connections that maximize resolution. The measurement results are automatically converted to the specified mode (e.g. A/B or B/A).

Note: Only one of the input signals may exceed 25MHz and should not be used when signal frequencies drop below 400 Hz.

Example

The following example sets the function mode to Ratio B/A:

```
SHORT nHandle, nStatus;
GxCntSetFunctionRatio (nHandle, GXCNT_MEASURE_B_DIVIDED_BY_A, &nStatus);
```

See Also

GxCntSetFunctionAutoRatio, **GxCntSetGateTime**, **GxCntSetAcquisitionTimeInterval**, **GxCntSetAcquisitionMode**, **GxCntGetAcquisitionMode**, **GxCntGetErrorString**

GxCntSetFunctionSinglePeriod

Purpose

Sets the function mode to Single Period.

Syntax

GxCntSetFunctionSinglePeriod (*nHandle*, *nChannel*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nChannel</i> | SHORT | Specified Channel number: <ul style="list-style-type: none"> • 0 = GXCNT_MEASURE_CHANNEL_A: measure channel A. • 1 = GXCNT_MEASURE_CHANNEL_B: measure channel B. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

Sets the mode to **Single Period**, in this mode a signal period can be measured over exactly one Acquisition by performing a time interval. Single period measurements may produce results that are less accurate than the conventional period mode.

Example

The following example function mode to Single Period measure channel A:

```
SHORT nHandle, nStatus;
GxCntSetFunctionSinglePeriod (nHandle, GXCNT_MEASURE_CHANNEL_A, &nStatus);
```

See Also

GxCntSetFunctionFrequency, **GxCntSetGateTime**, **GxCntSetAcquisitionTimeInterval**, **GxCntSetAcquisitionMode**, **GxCntGetErrorString**

GxCntSetFunctionTestInternalClock

Purpose

Sets the function mode to Test Internal Clock.

Syntax

GxCntSetFunctionTestInternalClock (*nHandle*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

In this mode the counter measures the internal clock frequency. This mode is useful as a quick verification of instrument operation. The function measures only the internal clock frequency even if the clock source is set to **Alternate** or **External**.

Example

The following example sets the function mode to test internal clock:

```
SHORT nHandle, nStatus;  
GxCntSetFunctionTestInternalClock (nHandle, &nStatus);
```

See Also

GxCntSetClockSource, **GxCntGetErrorString**

GxCntSetFunctionTimeInterval

Purpose

Sets the function mode to Time Interval.

Syntax

GxCntSetFunctionTimeInterval (*nHandle*, *nMode*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nMode</i> | SHORT | Measurement Mode: <ul style="list-style-type: none"> • 0 = GXCNT_MEASURE_A_TO_B: measure time interval from channel A to Channel B. • 1 = GXCNT_MEASURE_B_TO_A: measure time interval from channel B to Channel A. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

Time Interval is defined as the elapsed time between event on the start input channel and an event on the stop input channel. Event is defined either a positive or negative transition of the input signal (using **GxCntSetChannelSlope**).

The GC2200/GTX2200 allows Time Interval measurements to be both start armed and stop armed for maximum flexibility.

Example

The following example sets the function mode to Test Interval measure channel A to channel B:

```
SHORT nHandle, nStatus;
GxCntSetFunctionTimeInterval (nHandle, GXCNT_MEASURE_A_TO_B, &nStatus)
```

See Also

GxCntSetFunctionTimeIntervalDelay, **GxCntSetArmSource**, **GxCntSetArmSlope**, **GxCntGetErrorString**

GxCntSetFunctionTimeIntervalDelay

Purpose

Sets the function mode to Time Interval with Delay

Syntax

GxCntSetFunctionTimeIntervalDelay (*nHandle*, *nMode*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nMode</i> | SHORT | Measurement Mode: <ul style="list-style-type: none"> • 0 = GXCNT_MEASURE_A_TO_B: measure time interval from channel A to Channel B. • 1 = GXCNT_MEASURE_B_TO_A: measure time interval from channel B to Channel A. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

Time Interval is defined as the elapsed time between an event on the start input channel and an event on the stop input channel. Event is defined either a positive or negative transition of the input signal (using **GxCntSetChannelSlope**). A time interval with delay measurement begins like a standard measurement except that the stop event is disabled. As soon as the start event is recognized, the programmed delay begins. When the delay interval elapses, completion is enabled and the next stop event terminates the measurement. The delay interval can be specified using **GxCntSetTimeIntervalDelay** function.

The GC2200/GTX2200 allows Time Interval measurements to be start armed for maximum flexibility.

Note: Using the Time Interval with Delay the stop arm event is not available.

Example

The following example sets the function mode to time interval with delay measure channel A to channel B:

```
SHORT nHandle, nStatus;
GxCntSetFunctionTimeIntervalDelay (nHandle, GXCNT_MEASURE_A_TO_B, &nStatus);
```

See Also

GxCntSetTimeIntervalDelay, **GxCntSetFunctionTimeInterval**, **GxCntSetArmSource**, **GxCntSetArmSlope**, **GxCntGetErrorString**

GxCntSetFunctionTotalize

Purpose

Sets function to Totalize.

Syntax

GxCntSetFunctionTotalize (*nHandle*, *nChannel*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nChannel</i> | SHORT | Specified Channel number: <ul style="list-style-type: none"> • 0 = GXCNT_MEASURE_CHANNEL_A: measure channel A. • 1 = GXCNT_MEASURE_CHANNEL_B: measure channel B. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

In **Totalize** mode counting is controlled explicitly by software commands.

Note: **GxCntGetTotalizeGateMode** function opens and closes the count gate.

GxCntClear function clears the count.

If the counter is already in the **Totalize** mode, counting stops and the count is cleared.

The Totalize function allows events on one input channel to be counted for a period of time determined by either manual keyboard inputs, program statements

In the “manual” mode, a mouse click or program statement will explicitly open the count gate, close the count gate, or reset the count to zero. The manual mode provides the most convenient control of counting, but does not allow precise timing.

The “gated” mode allows a second input signal to open and close the gate on user specified signal transitions. After the gate closes, the measurement data is held until explicitly reset by the user or by a program statement.

The gating interval for gated Totalize mode is defined by the “Gated Tot Slopes” entries of the Inputs menu. Selecting like slopes for the start and stop (e.g. both positive) sets the gating interval to one period of the gating signal. Setting opposite slopes for start and stop changes the gating interval to a pulse width of the gating signal.

Example

The following example sets the gate time to 1mS:

```
SHORT nHandle, nStatus;
GxCntSetFunctionTotalize (nHandle, 0.001, &nStatus)
```

See Also

GxCntGetGateTime, **GxCntSetAcquisitionTimeInterval**, **GxCntGetAcquisitionTimeInterval**, **GxCntSetAcquisitionMode**, **GxCntGetAcquisitionMode**, **GxCntGetErrorString**

GxCntSetFunctionTotalizeGated

Purpose

Sets function to Totalize Gated.

Syntax

GxCntSetFunctionTotalizeGated (*nHandle*, *nMode*, *nStartSlope*, *nStopSlope*, *pnStatus*)

Parameters

| Name | Type | Comments |
|--------------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nMode</i> | SHORT | Measurement mode: <ul style="list-style-type: none"> • 0 = GXCNT_MEASURE_A_GATED_BY_B • 1 = GXCNT_MEASURE_B_GATED_BY_A |
| <i>nStartSlope</i> | SHORT | Measurement Start Slope: <ul style="list-style-type: none"> • 0 = GXCNT_POSITIVE_SLOPE: Rising (Positive) edge. • 1 = GXCNT_NEGATIVE_SLOPE: Falling (Negative) edge. |
| <i>nStopSlope</i> | SHORT | Measurement Stop Slope: <ul style="list-style-type: none"> • 0 = GXCNT_POSITIVE_SLOPE: Rising (Positive) edge. • 1 = GXCNT_NEGATIVE_SLOPE: Falling (Negative) edge. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

The Totalize Gated function allows events on one input channel to be counted for a period of time determined by events on the second input channel. The second input signal open and close the gate as specified in *nStartSlope* and *nStopSlope*.

Only one measurement, over a single gate interval, is performed. A new measurement will be started when the counter receives most commands. The **GxCntClear** or **GxCntTrig** functions are the preferred way to initiate a new measurement.

After the gate closes, the measurement data is held until explicitly cleared (**GxCntClear**). In order to identify the final count the **GxCntIsMeasurementReady** function must be used to check that the gate has opened and closed.

Note: If the counter is already making a measurement when this mode is set counting stops and the count is cleared.

Example

The following example sets the gate time to 1mS:

```
SHORT nHandle, nStatus;
GxCntSetFunctionTotalizeGated (nHandle, 0.001, &nStatus)
```

See Also

GxCntGetGateTime, **GxCntSetAcquisitionTimeInterval**, **GxCntGetAcquisitionTimeInterval**, **GxCntSetAcquisitionMode**, **GxCntGetAcquisitionMode**, **GxCntGetErrorString**

GxCntSetFunctionTotalizeGatedOnce

Purpose

Sets function to Totalize Gated Once.

Syntax

GxCntSetFunctionTotalizeGatedOnce (*nHandle*, *nMode*, *nStartSlope*, *nStopSlope*, *pnStatus*)

Parameters

| Name | Type | Comments |
|--------------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nMode</i> | SHORT | Measurement mode: <ul style="list-style-type: none"> • 0 = GXCNT_MEASURE_A_GATED_BY_B: • 1 = GXCNT_MEASURE_B_GATED_BY_A: |
| <i>nStartSlope</i> | SHORT | Measurement Start Slope: <ul style="list-style-type: none"> • 0 = GXCNT_POSITIVE_SLOPE: Rising (Positive) edge. • 1 = GXCNT_NEGATIVE_SLOPE: Falling (Negative) edge. |
| <i>nStopSlope</i> | SHORT | Measurement Stop Slope: <ul style="list-style-type: none"> • 0 = GXCNT_POSITIVE_SLOPE: Rising (Positive) edge. • 1 = GXCNT_NEGATIVE_SLOPE: Falling (Negative) edge. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

The Totalize Gated function allows events on one input channel to be counted for a period of time determined by events on the second input channel. The second input signal open and close the gate as specified in *nStartSlope* and *nStopSlope*.

Only one measurement, over a single gate interval, is performed. A new measurement will be started when the counter receives most commands. The **GxCntClear** or **GxCntTrig** functions are the preferred way to initiate a new measurement.

In this mode the counter won't output any data until the measurement is done (i.e. the gate is closed). It also ensures that intermediate data will not be read back to the control program, just the final result. The program will wait during a read operation until the gate closes.

Note: If the counter is already making a measurement when this mode is set counting stops and the count is cleared.

Example

The following example sets the gate time to 1mS:

```
SHORT nHandle, nStatus;
GxCntSetFunctionTotalizeGatedOnce (nHandle, 0.001, &nStatus)
```

See Also

GxCntSetFunctionAccumulate, **GxCntSetFunctionTotalizeGated**, **GxCntClear**, **GxCntTrig**, **GxCntGetGateTime**, **GxCntSetAcquisitionTimeInterval**, **GxCntSetAcquisitionMode**, **GxCntGetErrorString**

GxCntSetGateTime

Purpose

Sets the Gate time.

Syntax

GxCntSetGateTime (*nHandle*, *dSeconds*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>dSeconds</i> | DOUBLE | Gate time in seconds, Gate time range is 250 μ S to 3200 seconds with a resolution of 0.75 μ S |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

For **Frequency**, **Period**, **Ratio** and **GxCntSetFunctionTestInternalClock** functions, the selected **Gate** time sets the minimum measurement interval. It is ignored by all other functions.

Example

The following example sets the gate time to 1mS:

```
SHORT nHandle, nStatus;
GxCntSetGateTime (nHandle, 0.001, &nStatus)
```

See Also

GxCntGetGateTime, **GxCntSetAcquisitionTimeInterval**, **GxCntGetAcquisitionTimeInterval**, **GxCntSetAcquisitionMode**, **GxCntGetAcquisitionMode**, **GxCntGetErrorString**

GxCntSetMeasurementNumberOfDigits

Purpose

Sets the measurement number of digits and mode.

Syntax

GxCntSetMeasurementNumberOfDigits (*nHandle*, *nMode*, *nMaxDigits*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-------------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nMode</i> | SHORT | Measurement number of digits modes are: <ol style="list-style-type: none"> 0. GXCNT_MEASURE_NUM_OF_DIGITS_AUTO: the counter automatically sets the number of digits according to the gate time. 1. GXCNT_MEASURE_NUM_OF_DIGITS_FIXED: user overrides the counter's automatic measure number of digits settings, All measurement will have the same number of digits as was set by the <i>nMaxDigits</i> parameter. |
| <i>nMaxDigits</i> | SHORT | Number of digits can be between 5 to 14 |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

The function allows the use to either format the returned measurement value to a specified number of digits or increase the total number of digits when taking measurements with long gate time in order to get more than maximum 11 digits.

Note: Gx2210 and GC2210 will not support number of digits greater than 9.

Example

The following example sets the measurement number of digits to 13 and sets the mode:

```
GxCntSetMeasurementNumberOfDigits(nHandle, GXCNT_MEASURE_NUM_OF_DIGITS_FIXED, 13, &nStatus)
```

See Also

GxCntGetMeasurementNumberOfDigits, **GxCntSetMeasurementTimeout**, **GxCntGetFunction**, **GxCntGetErrorString**

GxCntSetMeasurementTimeout

Purpose

Sets the timeout allowable for a measurement read operation.

Syntax

GxCntSetMeasurementTimeout (*nHandle*, *dSeconds*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|---|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>dSeconds</i> | DOUBLE | Timeout range in seconds, Timeout range is 1mS to 3200 seconds. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

In most measurement functions the counter will wait indefinitely for a measurement to end if the input signal is removed or changes amplitude. If an attempt is made to read data from the board under this condition, the host computer will wait until the data is available. This lockup situation can be avoided by programming the timeout to an interval that is longer than any expected measurement, but will still allow the program to regain control if there is a signal fault.

In cases of measurements with long gate times or low frequency input signals and there is a potential problem, the **GxCntIsMeasurementReady** or **GxCntReadStatusRegister** functions can be used to see if data is ready and avoid unnecessary time-outs.

Example

The following example sets the timeout to 20mS:

```
SHORT nHandle, nStatus;
GxCntGetMeasurementTimeout (nHandle, 0.020, &nStatus)
```

See Also

GxCntGetMeasurementTimeout, **GxCntGetFunction**, **GxCntGetErrorString**

GxCntSetPrescalerMode

Purpose

Sets the pre-scale mode for both channels.

Syntax

GxCntSetPrescalerMode (*nHandle*, *nMode*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|---|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nMode</i> | SHORT | Pre-scale mode: <ul style="list-style-type: none"> • 0 = GXCNT_PRESCALE_OFF: Prescaling set to off. • 1= GXCNT_PRESCALE_ON: Prescaling set to on, the counter continuously prescales the actual measurement. • 2= GXCNT_PRESCALE_AUTO: The counter determines the need for prescaling automatically by making a quick frequency check prior to the actual measurement. The process takes about 20 μS. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

Prescaling is necessary when the input frequency exceeds 10MHz in **Frequency**, **Period**, **Ratio**, and **AutoRatio** modes (see specifications). Prescaling does not affect resolution or accuracy in any way.

The by the user, or it can be set to auto mode.

If the input frequency exceeds 10MHz and arming is enabled, the prescaler must be explicitly turned on. The prescaler reverts to the off state whenever arming is enabled, to reduce potential measurement start point ambiguity (up to 4 Acquisitions of the input signal may pass before the counter starts).

Example

The following example sets the Pre-scale mode to Auto:

```
SHORT nHandle, nStatus;
GxCntGetPrescalerMode (nHandle, GXCNT_PRESCALE_AUTO, &nStatus)
```

See Also

GxCntGetPrescalerMode, **GxCntGetFunction**, **GxCntSetAcquisitionMode**, **GxCntGetAcquisitionMode**, **GxCntGetErrorString**

GxCntSetTimeIntervalDelay

Purpose

Sets the delay time for the **GxCntSetFunctionTimeIntervalDelay** function.

Syntax

GxCntSetTimeIntervalDelay (*nHandle*, *nMode*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|---|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nMode</i> | SHORT | Delay time in seconds, Delay time range 20 μ S to 3200 seconds. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

The delay time is used as a special mode to disable or “hold off” measurement completion to ease such difficult measurement setups. The delay time is only used by the **GxCntSetFunctionTimeIntervalDelay** function.

Example

The following example sets the delay time to 1 mS:

```
SHORT nHandle, nStatus;  
GxCntSetTimeIntervalDelay (nHandle, 0.001, &nStatus)
```

See Also

GxCntGetTimeIntervalDelay, **GxCntGetFunction**, **GxCntGetErrorString**

GxCntSetTotalizeGateMode

Purpose

Sets the Totalize Gate mode.

Syntax

GxCntSetTotalizeGateMode (*nHandle*, *nMode*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|---|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nMode</i> | SHORT | Gate mode: <ul style="list-style-type: none"> • 0 = GXCNT_TOTALIZE_GATE_OPEN: open the count Gate. • 1 = GXCNT_TOTALIZE_GATE_CLOSE: close the count Gate. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

This function enables or disables the count Gate when in **Totalize** mode.

Example

The following example sets the Totalize Gate mode to start:

```
SHORT nHandle, nStatus;
GxCntSetTotalizeGateMode (nHandle, GXCNT_TOTALIZE_GATE_OPEN, &nStatus)
```

See Also

GxCntGetTotalizeGateMode, **GxCntGetFunction**, **GxCntGetErrorString**

GxCntSetTriggerSlope

Purpose

Sets the trigger slope.

Syntax

GxCntGetTriggerSlope (*nHandle*, *nSlope*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nSlope</i> | SHORT | Trigger slope: <ul style="list-style-type: none"> • 0 = GXCNT_POSITIVE_SLOPE: Rising (Positive) edge. • 1 = GXCNT_NEGATIVE_SLOPE: Falling (Negative) edge. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

The trigger slope setting is active only when the trigger source is set to external, see **GxCntSetTriggerSource** function for details.

Example

The following example sets the trigger slope to negative:

```
SHORT nHandle, nStatus;
GxCntSetTriggerSlope (nHandle, GXCNT_NEGATIVE_SLOPE, &nStatus)
```

See Also

GxCntSetTriggerSlope, **GxCntSetTriggerSource**, **GxCntGetTriggerSource**, **GxCntGetFunction**, **GxCntGetErrorString**

GxCntSetTriggerSource

Purpose

Sets the trigger source.

Syntax

GxCntSetTriggerSource (*nHandle*, *nSource*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>nSource</i> | SHORT | Trigger source: <ul style="list-style-type: none"> • 0 = GXCNT_TRIGGER_INTERNAL: Software triggered, initiates the first measurement immediately. • 1 = GXCNT_TRIGGER_EXTERNAL: The first measurement is delayed until the signal edge specified in GxCntSetTriggerSlope is received. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Example

The following example sets the trigger source to external:

```
SHORT nHandle, nStatus;
GxCntSetTriggerSource (nHandle, GXCNT_TRIGGER_EXTERNAL, &nStatus)
```

See Also

GxCntSetTriggerSource, **GxCntSetTriggerSlope**, **GxCntGetTriggerSlope**, **GxCntGetFunction**, **GxCntGetErrorString**

GxCntTrig

Purpose

Triggers the counter to make new measurement.

Syntax

GxCntTrig (*nHandle*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|--|
| <i>nHandle</i> | SHORT | Handle to a counter board. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Example

The following example sets the trigger the counter to initiate a new measurement:

```
SHORT nHandle, nStatus;  
GxCntTrig (nHandle, &nStatus)
```

See Also

GxCntSetTriggerSource, **GxCntSetTriggerSlope**, **GxCntGetTriggerSlope**, **GxCntGetFunction**, **GxCntGetErrorString**

GxCntUpgradeFirmware

Purpose

Upgrades the board's firmware.

Syntax

GxCntUpgradeFirmware (*nHandle*, *szFile*, *nMode*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-----------------|--------|--|
| <i>nHandle</i> | SHORT | Handle for a counter board. |
| <i>szFile</i> | PCSTR | Path and file name of the firmware file. The firmware file extension is JAM. |
| <i>nMode</i> | SHORT | The upgrading firmware mode can be as follows: <ol style="list-style-type: none"> 0. GT_FIRMWARE_UPGRADE_MODE_SYNC: the function returns when upgrading firmware is done or in case of an error. 1. GT_FIRMWARE_UPGRADE_MODE_ASYNC: the function returns immediately. The user can monitor the progress of upgrading firmware using the GxCntUpgradeFirmwareStatus API. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

This function used in order to upgrade the board's firmware. The firmware file can only be obtained by request from Marvin Test Solutions.

Note: Loading an incorrect firmware file to the board can permanently damage the board.

Example

The following example loads Upgrades the board's firmware using synchronous mode:

```
GxCntUpgradeFirmware (nHandle, "C:\\GxCntFw.jam", GT_LOAD_MODE_SYNC, &nStatus);
```

See Also

GxCntUpgradeFirmwareStatus, **GxCntGetErrorString**

GxCntUpgradeFirmwareStatus

Purpose

Monitor the firmware upgrade process.

Syntax

GxCntUpgradeFirmwareStatus (*nHandle*, *pszMsg*, *nMsgMaxLen*, *pnProgress*, *pbIsDone*, *pnStatus*)

Parameters

| Name | Type | Comments |
|-------------------|--------|--|
| <i>nHandle</i> | SHORT | Handle for a counter board. |
| <i>pszMsg</i> | PSTR | Buffer to contain the message from the firmware upgrade process. |
| <i>nMsgMaxLen</i> | SHORT | <i>pszMsg</i> buffer size . |
| <i>pnProgress</i> | PSHORT | Returns the firmware upgrades progress. |
| <i>pbIsDone</i> | PBOOL | Returned TRUE if the firmware upgrades is done. |
| <i>pnStatus</i> | PSHORT | Returned status: 0 on success, negative number on failure. |

Comments

This function is used in order to monitor the firmware upgrade process whenever the user called **GxCntUpgradeFirmware** API with GT_ FIRMWARE_UPGRADE_MODE_ASYNC mode.

Note: In order to prevent CPU over load if the function is called form within a loop, a delay of about 500mSec will be activated if the time differences between consecutive calls are less than 500mSec.

Example

The following example loads Upgrades the board's firmware using asynchronous mode, and ten monitors the firmware upgrade process:

```
CHAR    sz[256];
CHAR    szMsg[256];
BOOL    bIsDone=FALSE;
GxCntUpgradeFirmware (nHandle, "C:\\ GxCntFw.jam", GT_UPGRADE_FIRMWARE_MODE_ASYNC, &nStatus);
if (nStatus<0)
{
    GxCntGetErrorString(nStatus, sz, sizeof sz, &nStatus);
    printf(sz); // prints the error string returns
}
While (bIsDone==FALSE || nStatus<0)
{
    GxCntUpgradeFirmwareStatus (nHandle, szMsg, sizeof szMsg, &nProgress, &bIsDone, &nStatus);
    printf("Upgrade Progress %i", nProgress);
    sleep(1000);
}
if (nStatus<0)
{
    GxCntGetErrorString(nStatus, sz, sizeof sz, &nStatus);
    printf(sz); // prints the error string returns
}
```

See Also

GxCntUpgradeFirmware, **GxCntGetErrorString**

Index

- .
- .NET ii
- 3**
- 3U PXI.....22
- A**
- Accessories37
- Accumulate.....11
- Applications.....5
- Architecture1, 7
- Arming.....13
- ATEasyii, 38, 39, 40, 41, 42
- Auto Trigger12
- AutoRatio11
- B**
- Block Diagram.....7
- Board Description.....1, 5
- Board Handle.....44
- Board Installation34
- Borlandii, 38, 41, 42
- Borland-Delphi42
- C**
- C/C++38, 39, 41
- C++41
- Calibration Interval.....51
- Common Input Mode.....16
- Configuring.....33
- Connector36
- Connectors36, 37
- Connectors and Accessories37
- Corrupt files.....40
- Counter Reference Clock To PXI Reference Clock 15
- Coupling16
- Coupling DC/AC16
- D**
- Delphiii, 38, 41, 42
- DIN-6 Connector37
- Distributing.....45
- Driver
 - Directory38
 - Files38
- Driver Version44
- E**
- Error Handling.....44
- Example.....39
- External Arm Input.....15
- External Clock Input.....15
- F**
- Fast Frequency.....9
- Features.....3
- Filtering Input Signal.....16
- Frequency7
- Functions Reference63
- G**
- Gate Output15
- Gated Totalize11
- GC2200 Front Connectors36
- GC2200/GTX2200 Functions List64
- Getting Started.....31
- GTX2200 Front Connectors36
- GXCNT1, 32
 - Driver-Description.....41
 - Header-file41
 - Help-File-Description39
 - Panel-File-Description.....38
- GXCNT.BAS.....38, 41
- GXCNT.DLL.....32, 38, 41, 42
- GXCNT.EXE.....32
- GXCNT.H38, 41
- GXCNT.LIB.....38, 41
- GXCNT.lib42

| | | | |
|--|------------|--|-----------------------------|
| GXCNT.PAS | 38, 42 | GxCntInitialize . | 24, 43, 44, 60, 63, 94, 106 |
| GXCNT.VB | 38 | GxCntInitializeVisa | 24, 43, 44, 107 |
| GXCNT64.DLL | 32, 38, 41 | GxCntInSystemCalDevice | 108 |
| GXCNT64.LIB | 38, 41 | GxCntInSystemCalGetStatus | 109 |
| GXCNTBC.LIB | 38 | GxCntInSystemCalRestore | 110 |
| GxCntClear | 69 | GxCntInSystemCalSave | 111 |
| GxCntGetAcquisitionMode | 70 | GxCntInSystemCalStart | 112 |
| GxCntGetAcquisitionTimeInterval | 71 | GxCntIsMeasurementReady | 113 |
| GxCntGetArmSlope | 72 | GXCNTPANE64L.EXE | 38 |
| GxCntGetArmSource | 73 | GxCntPanel | 114 |
| GxCntGetBoardSummary | 74 | GXCNTPANEL.DLL | 38 |
| GxCntGetBoardType | 75 | GXCNTPANEL.EXE | 44 |
| GxCntGetCalibrationInfo | 76 | GXCNTPANEL64.EXE | 44 |
| GxCntGetCalibrationMode | 78 | GxCntReadMeasurement | 115 |
| GxCntGetChannelAFrequencyRange | 79 | GxCntReadMeasurementArray | 116 |
| GxCntGetChannelCouplingMode .. | 80, 129 | GxCntReadMeasurementString | 118 |
| GxCntGetChannelFilterMode | 81 | GxCntReadStatusRegister | 119 |
| GxCntGetChannelFilterValue | 82 | GxCntReset | 44, 121 |
| GxCntGetChannelImpedance | 83 | GxCntSelfTest | 122 |
| GxCntGetChannelSlope | 84 | GxCntSetAcquisitionMode | 123 |
| GxCntGetChannelTriggerLevel | 85 | GxCntSetAcquisitionTimeInterval | 124 |
| GxCntGetChannelTriggerLevelMode .. | 86 | GxCntSetArmSlope | 125 |
| GxCntGetClockSource | 88 | GxCntSetArmSource | 126 |
| GxCntGetCommonInput | 89 | GxCntSetCalibrationMode | 127 |
| GxCntGetCounterRefClockToPxiRefClockState .. | 90 | GxCntSetChannelAFrequencyRange .. | 128 |
| GxCntGetDriverSummary | 43, 44, 91 | GxCntSetChannelAutoSet | 68 |
| GxCntGetErrorString | 44, 63, 92 | GxCntSetChannelFilterMode | 130 |
| GxCntGetExtendedSerialNumber | 95 | GxCntSetChannelFilterValue | 131 |
| GxCntGetFunction | 96 | GxCntSetChannelImpedance | 132 |
| GxCntGetGateTime | 98 | GxCntSetChannelSlope | 133 |
| GxCntGetMeasurementNumberOfDigits | 99 | GxCntSetChannelTriggerLevel | 134 |
| GxCntGetMeasurementTimeout | 100 | GxCntSetChannelTriggerLevelMode .. | 135 |
| GxCntGetPrescalerMode | 101 | GxCntSetClockSource | 137 |
| GxCntGetTimeIntervalDelay | 102 | GxCntSetCommonInput | 138 |
| GxCntGetTotalizeGateMode | 103 | GxCntSetCounterRefClockToPxiRefClockState .. | 139 |
| GxCntGetTriggerSlope | 104 | GxCntSetFunctionAccumulate | 140 |
| GxCntGetTriggerSource | 105 | GxCntSetFunctionAutoRatio | 141 |

| | |
|---|-------------------------------------|
| GxCntSetFunctionFastFrequency | 142 |
| GxCntSetFunctionFrequency | 143 |
| GxCntSetFunctionPeriod | 144 |
| GxCntSetFunctionPulseWidth..... | 145 |
| GxCntSetFunctionRatio..... | 146 |
| GxCntSetFunctionSinglePeriod..... | 147 |
| GxCntSetFunctionTestInternalClock.... | 148 |
| GxCntSetFunctionTimeInterval | 149 |
| GxCntSetFunctionTimeIntervalDelay .. | 150 |
| GxCntSetFunctionTotalize | 151 |
| GxCntSetFunctionTotalizeGated..... | 152 |
| GxCntSetFunctionTotalizeGatedOnce . | 153 |
| GxCntSetGateTime | 154 |
| GxCntSetMeasurementNumberOfDigits | 155 |
| GxCntSetMeasurementTimeout | 156 |
| GxCntSetPrescalerMode | 157 |
| GxCntSetTimeIntervalDelay | 158 |
| GxCntSetTotalizeGateMode..... | 159 |
| GxCntSetTriggerSlope | 160 |
| GxCntSetTriggerSource | 161 |
| GxCntTrig..... | 162 |
| GxCntUpgradeFirmware | 163 |
| GxCntUpgradeFirmwareStatus..... | 164 |
| H | |
| Handle | 34, 35, 43, 44 |
| HW | 35, 38, 41, 45 |
| I | |
| Input Prescaling | 14, 15 |
| Input Signal Pre-Scaling | 8 |
| Installation Folders | 38 |
| Installation: | 34, 36 |
| In-System Calibration..... | 51 |
| In-System Calibration Example program | 58 |
| In-System Calibration license setup | 52 |
| In-System Calibration Program Listing .. | 58 |
| In-System Calibration Sample | 58 |
| Introduction | 3, 63 |
| L | |
| LabView | 42 |
| LabView/Real Time | 42 |
| Linux | 42 |
| Listing | 46 |
| M | |
| Measurement Functions..... | 7 |
| N | |
| <i>nHandle</i> | 42, 44 |
| O | |
| OnError..... | 42 |
| P | |
| Paced Measurements | 14 |
| Panel | 23, 28, 39, 40, 43, 44, 53, 54, 114 |
| Pascal..... | 38, 41, 42 |
| PCI..... | 3, 38 |
| Period..... | 10 |
| Plug & Play..... | 35 |
| <i>pnStatus</i> | 44, 63 |
| Prescaling | 14, 15 |
| Program-File-Descriptions | 38 |
| Programming | |
| Borland-Delphi | 42 |
| Error-Handling..... | 44 |
| Panel-Program | 44 |
| Programming Using Visual Basic | 41 |
| Pulse Width | 12 |
| PXI..... | 3, 31, 33, 34, 35, 36, 43, 54 |
| PXI 10MHz Clock Source | 15 |
| PXI/PCI Explorer..... | 24, 33, 43, 44, 54, 106, 107 |
| PXIeSYS.INI | 24 |
| PXISYS.INI..... | 24 |
| R | |
| Ratio | 11 |
| README.TXT | 38, 39 |
| Readme-File | 39 |
| Removing a Board | 36 |

| | | | |
|---|--------------------------------|--|-----------------------------|
| Required instrument | 51 | Time Interval with Delay | 10 |
| Reset | 44 | Totalize | 11 |
| Restore Calibration | 57 | Totalize, Gated Totalize, & Accumulate | 11 |
| Running calibration using API calls | 58 | Trigger | 12 |
| Running In-System calibration | 53 | Trigger Level Channel A Calibration | 55 |
| S | | Trigger Level Channel B Calibration | 56 |
| Sample | 45, 46 | Triggered Pacing..... | 14 |
| Sample Programs | 45 | U | |
| Save Calibration | 57 | Using the GXCNT driver functions..... | 43 |
| Setup | 32, 38, 40 | V | |
| Setup Maintenance | 40 | Virtual Panel23, 24, 27, 28, 32, 39, 43, 53, 114 | |
| Setup-and-Installation..... | 31 | About Page | 30 |
| Single Measurement | 14 | Channel A \ Channel B Page | 27 |
| Single Period | 11 | Channel B Page | 27 |
| Slot..... | 24, 34, 36, 43, 45, 53, 54, 58 | Initialize Dialog | 24 |
| Specifications | 1, 17 | Virtual Panel Description | 23 |
| Supported-Development-Tools..... | 41 | Virtual Panel Description | 1 |
| System | | Virtual Panel Setup Page | 25 |
| Directory..... | 38 | VISA..... | 4, 24, 43, 44, 64, 106, 107 |
| System-Requirements | 31 | Visual Basic..... | ii, 41 |
| T | | Visual C++ | ii, 38, 41 |
| The GC2200/GTX2200 Driver..... | 41 | W | |
| Time Base Calibration | 54 | Width | 12 |
| Time Interval | 9 | | |