GuideTech

GT668
Time Interval Analyzer

Operating Manual

GuideTech, LLC. ♦ 1300 Memorex Drive ♦ Santa Clara, CA 95050 ♦ USA
Tel: 1-408-988-9998
www.guidetech.com

Trademarks

Windows is a trademark of Microsoft Corporation. LabView is a trademark of National Instruments Corporation.

Copyright Notice

Disclaimer

GuideTech, LLC. makes no representations or warranties with respect to this manual. Further, GuideTech, LLC. reserves the right to make changes in the specifications of the products described within this manual at any time without notice and without obligation of GuideTech, LLC. to notify any person of such revision or changes. All Statements, technical information, and recommendations in this document and in any manuals or related documents are believed reliable, but the accuracy or completeness thereof is not guaranteed.

Table of Contents

Table of Figures

1. INSTALLATION

SYSTEM REQUIREMENTS

To install the GT668 Time Interval Analyzer board and software you must have the following equipment and software:

- PC computer
- A slot appropriate for the board model
- Windows 7 or newer (Windows XP and Windows Vista available upon request - please contact GuideTech), or Linux with kernel version up to 4.3.3 (it was tested with OpenSUSE 13.1 and with Ubuntu 15.10).
- For USB based instrument the computer must use a USB3.0 socket (USB is supported only under Windows).

Note to users USB instrument:

When using a USB3.0 instrument please connect the USB cable to the computer before powering on the instrument. If this is not done the instrument may not be identified (disconnecting and reconnecting the cable will fix this).

INSTALLATION INSTRUCTIONS

The installation instructions in this section provide a detailed description of the installation procedure. Please read the instructions in this section completely before attempting to install your Universal Counter.

Software Installation under Windows

The GT668 software for Windows is distributed as a downloadable self-extracting executable file named *GT668Setup.exe* (for 32-bit Windows) and *GT668Setup64.exe* (for 64-bit Windows). Download instructions to get the file are provided with the purchase of the instrument.
To install the software simply run the provided GT668Setup.exe or GT668Setup64.exe program and follow the instructions.

Note to users of Windows 7 64-bit:

Some 64-bit Windows 7 (Service Pack 1) systems will generate an error when the board drivers are installed. The error complains about inability to verify the signature of the published. To fix (or avoid) this problem please download and install the Microsoft Security Update KB3033929 (available from the following link):

KB3033929 Security Update (or copy and paste the following link to your browser https://www.microsoft.com/en-us/download/details.aspx?id=46148).

Note to users of Windows 8, Windows 8.1 and Windows 10:

Before running the installation you have to do the following steps:

1. Restart the computer (using any power menu) while holding the “Shift” key down.
2. After the computer restarts click on “Troubleshoot”.
3. Click on the “Advanced Options”.

4. Click on “Startup Settings”.
5. Click on the “Restart” button.
6. After the computer restarts again select option 7 “Disable driver signature enforcement”.
7. After the computer restarts run the GT668 installation file.

Note to users of Windows Vista, Windows 7, Windows 8, and Windows 8.1:

During installation of the software you may see the following message pop twice:

Please click on the "Install this driver anyway".

Software Installation under Linux

The GT668 software for Linux is distributed as a downloadable archive file named GT668Setup.tar.gz (32-bit Linux for x86 architecture) and GT668Setup64.tar.gz (64-bit Linux for x86_64 architecture). Download instructions to get the file are provided with the purchase of the instrument.

To install the software in directory *<parent_directory>/gt668* (where *<parent_directory>* is some absolute directory path) run the following commands (assuming the distribution file is in *<setup_file_dir>*):

```
cd <parent_directory>
tar xvzf <setup_file_dir>/GT668Setup.tar.gz
      or (for 64-bit)
tar xvzf <setup_file_dir>/GT668Setup64.tar.gz
cd gt668/Setup
./setup
```

To make the kernel mode driver load after every boot add the following lines to your local boot script file (a script file such as /etc/rc.d/boot.local – system dependent):

```
<parent_directory>/gt668/Setup/start
```

The installation will install:

- Include files, libraries, some simple sample programs, and installation files.
- A kernel mode driver GTDriver.ko will be added into your system, including a device "/dev/GTDriver".

- A kernel “plugin module” kpgt668.ko will be added into your system to work with the GTDriver driver..

- Symbolic links to the driver libraries installed in *<parent_directory>/gt668/Lib* will be added into the /usr/lib directory (for 32-bit systems) or into /usr/lib64 (for 64-bit systems) directory.

Hardware Installation

> **WARNING!**
>
> To install the board you have to remove the cover from your computer. For protection against electrical shock and to prevent the possibility of damage to your computer, *never* remove the cover without first ensuring that your computer is turned off *and* unplugged from the wall socket.

Please check the power requirements in the specifications section and make sure that your computer is able to provide the necessary power on all supplies.

Use the following procedure to install the GT668 Universal Counter board in your computer.

1. Remove the board from its packaging. Before handling the board, touch a grounded metal object such as the metal chassis of your computer to remove any static electricity which could damage the board. To ensure reliable operation, do not touch the gold contact fingers on the bottom edge of the board.

3. Turn off the power to your computer and unplug the power cord from the wall socket.

4. Refer to the owner's manual for the host computer to obtain information on how to remove the computer cover.

5. Select a card slot for the GT668. Preference should be given to slots which are far from existing high power dissipation cards, such as high-end graphic cards.

6. Press the board firmly into the socket: check for proper alignment of the card edge connector if significant resistance is felt. Ensure that the SMA connectors are properly centered in the panel cut out.

7. Make certain that you reinstall the screw which secures the GT668 to the computer frame. This screw ensures proper electrical grounding of the card.

8. Slide the cover back onto the computer and reinstall the retaining screws.

9. Plug the power cord into the wall socket. Turn on the power.

Software Installed:

- GT668 Exerciser – a GUI based program that allow configuration of the GT668 and making some simple measurements.

- GT668 Manual – this document.

- Include directory – contains include files for C/C++ program interface. The main include file is gt668drv.h.

- Lib directory – contains the file GT668.lib to be linked to C/C++ programs.

- GT668.dll and GTPCI.dll – Dynamic Link Libraries used by the GT668 software, installed also into the Windows\System32 directory.

- Present – a command line program (ported from older GT65x instruments) that detects the signal amplitude on all four SMA inputs, and measures the frequency of the A and B input signals. Source code is provided in the Samples\Present directory.

- Prescale – a command line program (ported from older GT65x instruments) that takes measurements with user specified prescaling to the standard output or to a file. Source code is provided in the Samples\Prescale directory.

- ShowData – a command line program (ported from older GT65x instruments) that demonstrates taking measurements on channels A and B and display the first few time tags on channel A. Source code is provided in the Samples\ShowData directory.

- SlowSig – a command line program (ported from older GT65x instruments) that demonstrates taking measurements on channel A continuously (when the number of raw data tags is more than can fit in memory). Source code is provided in the Samples\SlowSig directory.

2. OVERVIEW

What is a TIA?

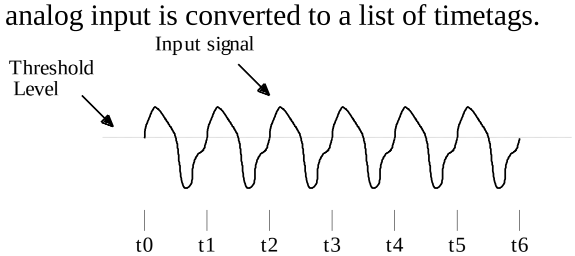
Time Interval Analyzers (TIA's) are super high speed time and frequency measurement instruments that can make millions of measurements per second. Compared to traditional time interval counters, the fast measurement rate provides new capability to analyze *dynamic changes* in frequency or time intervals.

TIAs give you new insight into your signals. The table below shows the type of measurements you can make with oscilloscopes, spectrum analyzers and time interval analyzers.

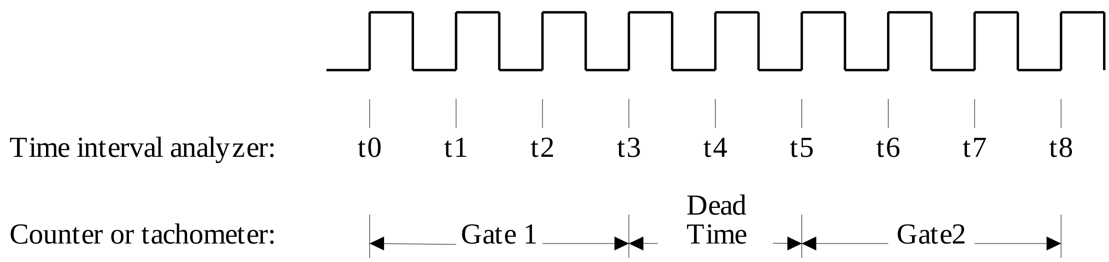
Instrument	Display	Measurement Domain
Oscilloscope	Amplitude (voltage) vs. time	Time domain
Spectrum analyzer	Amplitude vs. frequency	Frequency domain
TIA	Frequency vs. time Time-interval vs. time Phase vs. time	Modulation domain

The difference between TIA's and time interval counters (or frequency counters) is analogous to the difference between voltmeters and digital scopes. Voltmeters make individual measurements of the voltage of the signal, and they are usually intended for slow measurements of slowly varying signals. Digital scopes, on the other hand, measure the voltage of the signal at a much higher rate and provide sophisticated and precise control of the timing of measurements. In addition, they display the measurements on a graphical display ***as a function of time***.

In the most basic form, TIAs log the time of occurrence of events at the inputs. Events are defined as a signal voltage crossing a specified threshold in the positive or negative direction. These "time-tags" of the positive or negative edges are logged into memory along with the total count of edges received. The figure below shows how the analog input is converted to a list of timetags.



To illustrate how measurements are derived from the timetag list, consider the following waveform, which has a period of 1 ms (frequency is 1 kHz):



The waveform has a positive edge every ms. The TIA would end up with a block of time tags that might look like this:

t0	1234.1234567891 Seconds
t1	1234.1244567891 Seconds
t2	1234.1254567891 Seconds
t3	1234.1264567891 Seconds
t4	1234.1274567891 Seconds
t5	1234.1284567891 Seconds
t6	1234.1294567891 Seconds
t7	1234.1304567891 Seconds
t8	1234.1314567891 Seconds

Note that the starting time, t0, is arbitrary. The important thing is that the time difference between each of the time tags is 1 ms. From this block of time tags it is possible to determine many things about this waveform. For example,

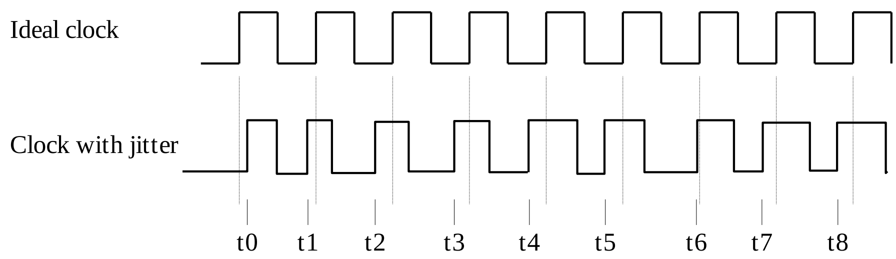
$$Frequency = \frac{Events}{Time} = \frac{1}{t_1 - t_0} \quad \text{or} \quad \frac{2}{t_2 - t_0} \quad \text{or} \quad \frac{6}{t_8 - t_2}$$

In other words, the block of time tags allows the software to go back and calculate the instantaneous frequency over any time "window". Note that we can measure the frequency of every period (the measurements are "back-to-back"). This is also called "zero dead time" measurement.

As a comparison, a traditional frequency counter would make independent measurements (shown above as "Gate 1" and "Gate 2"). These measurements would give the average frequency of the signal over the gate times, but the time at which the measurements are taken is not kept. That is, you simply get a set of individual frequency measurements.

Since all measurements with a TIA can be referenced to the same starting point t0, we can determine the absolute time deviation of any edge from its expected time. This is very useful in various applications including clock signal characterization, mechanical position and jitter measurements, and in data communications.

The waveform diagram below illustrates the use of TIA's for jitter measurements. The software can compute the deviation of each of the edges t0 to t8 from an ideal clock. There is no need to measure against an actual ideal clock, since the expected times can be computed.

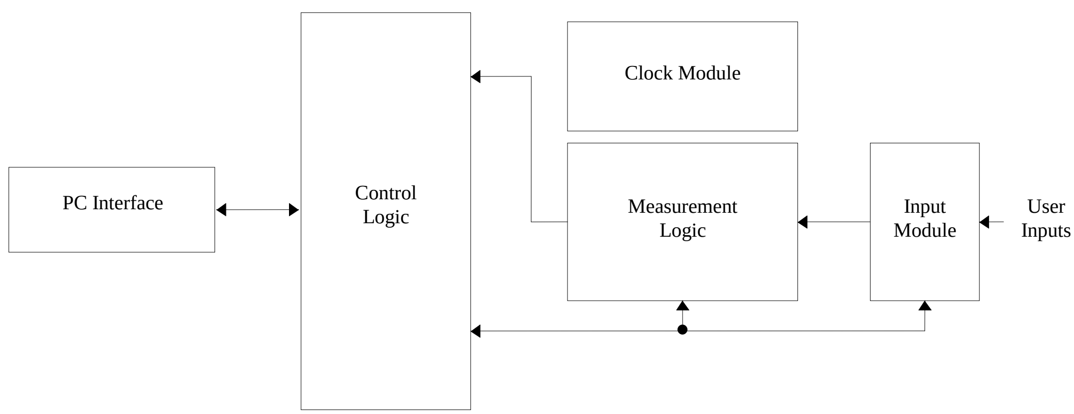


Another important aspect of TIA's is the sophisticated arming they provide. Simply viewed, arming gives you control of *when* measurements (time tags) are taken.

Hardware Architecture

The block diagram below shows the relationship between the various systems on the board. An important aspect of the architecture is the fact that it stores the timetags directly to the memory in the computer. This feature greatly enhances the speed of the board.

NOTE: There is no direct writing to memory on USB connected TIA.



Hardware Architecture of the GT668

The Clock Module

The clock module provides the 50 MHz time base clock for the board (100 MHz for newer boards). This frequency is generated by multiplying a 10 MHz TCXO (temperature compensated crystal oscillator) using a PLL (phase-locked loop) circuit. The board can also accept an external 10 MHz or 5MHz reference signal instead of the local TCXO. The PLL ensures smooth switching from the internal to the external clock and has a ±50 ppm range. To guarantee lock, the external reference should be kept at 10 MHz ±200 Hz or 5 MHz ±100 Hz. If the external reference is outside that range, the PLL will not lock to it and will “peg” at one of the limits of its range.

The Input Module

The block diagram below shows in more detail the inputs of the GT668. The comparators are used to "square" the input signal. The comparator's output is logic high whenever the positive input voltage is greater than the negative input and a logic low otherwise. The analog input signals are applied to the positive inputs, and a programmable voltage is applied to the negative inputs. This programmable voltage dictates the voltage point on the signal in which an event will be captured. Each of the two main channels (A and B), the ARM input, and the EXT CLK input, have separate threshold setting DAC's (digital to analog converters). That is, each of these inputs can be set to a different threshold voltage.

Each of the two main channels (A and B) contains a programmable prescaler that is used to reduce the measurement rate to be not higher than 4 million per second.

Following the inputs there are two multiplexers that allow selecting the measured signal for each of the two measurement channels. The measured signal to each channel may come from either input and with optional inversion (to measure the negative edge).

NOTE: The prescaler always works on the positive edges of the input signal, because of this selecting the negative edge as source to the multiplexers when the prescaler value is higher than 1 will NOT measure the negative edge of the signal.

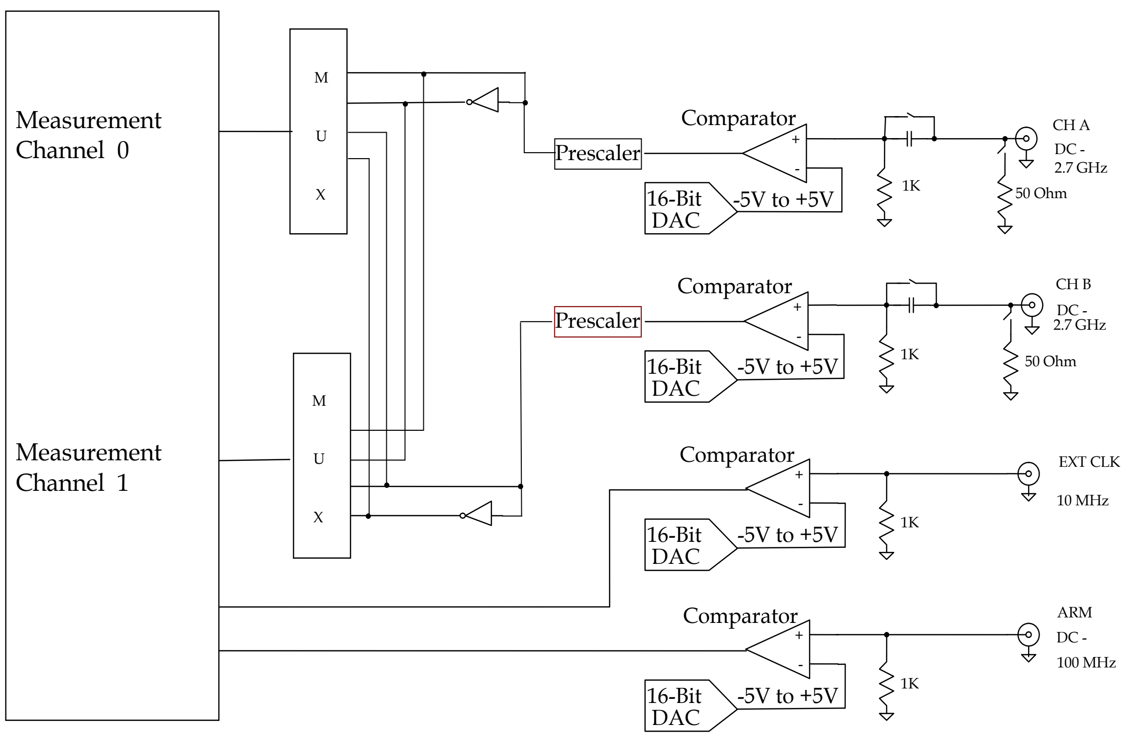


Figure 1 Inputs Block Diagram

Block Arming

The GT668 has block arming capability. Block arming enables the TIA to start a block of measurements. Block arming source can be configured to be one of the following: permanently armed (default), software generated, external signal, or generated by another TIA card. After the TIA operation is started, it will wait for block arming before generating any time tag.

Timetag Arming

The GT668 has time tag arming capability. Timetag arming enables the TIA to generate one timetag. Timetag arming source can be configured to be one of the following: permanently armed (default), software generated, other channel signal, external signal, or generated by another TIA card. Each arming event will enable generation of one timetag.

Real Time Clock

The GT668 (in Rev. 8 and later boards) has a real time clock that can be synchronized to UTC (or any other time standard) by a software command or by a 1PPS signal. The clock runs continuously independently of any measurements as long the board power is not interrupted. Changing the reference clock (e.g. switching from internal reference to external reference) can cause change in accuracy. Running self-calibration switches temporarily the reference setting which may also introduce a small error. The long term accuracy of the real time clock depends on the accuracy of the reference used.

Output Module

The GT668 (in Rev. 8 and later boards) has an output module with two SMA outputs. The outputs can drive 4V pulses into 50Ω load. The pulses start time can be configured by software on one of the following conditions:

1. At a specified real time value (in older boards up to 64 pulse commands can be pending, newer ones can support 512 or even 1024 commands pending)
2. Start by an external signal edge connected to the ARM input.
3. Start when block arm happened during measurements.
4. Start on a start measurements command.

The pulse width is programmable, and a programmable delay can be added to options 2, 3, and 4. All these features are programmed with a 10 nsec resolution.

Please look at the following functions in Function Reference for details: *GT668SetOutput(), GT668GetRealTimeOutputStatus(), GT668GetRealTimeOutputMaxPend(),GT668RealTimeOutput().*

Differences of the GT668 and GT658

Both instruments are similar except for the following:

- GT668 has front end prescaler for 1, 2, 4, 8, 16, 32, 64,128, 256, 512, and 1024. The GT658 had only prescaler values of 1, 2, 4, 16, 32, 64, and 256.
- GT668 can measure up to 4 million measurements per seconds vs. 3.5 million on the GT658.
- GT668 can measure higher frequencies and has a much better accuracy.
- The GT668 uses a block of kernel memory inside the PC driver while the GT658 use on board memory.
- The GT668 memory can contain up to 8 million raw time tags vs. 1 million in the GT658.

- The GT668 has much faster access to the raw data than the GT658 allowing continuous measurements at a much higher rate - up to the full measurement speed (while the GT658 could not run continuously at more than about 200,000 measurements per second).

- GT668 uses a 10MHz TCXO to generate a 50 MHz time base signal (100MHz in newer boards). GT658 uses 50 MHz OCXO.

- Real time clock and outputs are new to GT668 from Rev. 8.

3. APPLICATIONS

Cables and Interconnects

Proper and reliable operation of any instrument requires good quality signals at its inputs. This is not so easy to achieve with today's high speed signals. An important thing to note is that it is the ***frequency content*** in the signal that matters rather than just the frequency. The highest frequency components of a square wave depend on the rise time of the edges and not on the pulse rate.

Common reasons for poor quality signals include the "bounce" and overshoot caused by improper termination, and the misuse of drivers. The discussion below is a practical tutorial on cable termination and is useful for all your interconnect problems.

Electrical signals travel through cables at speeds approaching the speed of light (1 ft. per ns). A typical 6 ft. coaxial cable delays a signal by 9 ns as it travels from one end of the cable to the other. During this time, the impedance that the driver "sees" is only that of the cable. For a coax, it is usually 50 Ω, while for a flat ribbon cable it is usually 110 Ω. If the impedance that is attached at the end of the cable (termination) is not equal to the impedance of the cable, some of the signal will be reflected back toward the driver. This reflection causes the signal to bounce and overshoot (ringing). After several reflections everything settles and the driver "sees" the termination at the end of the cable as its constant load.

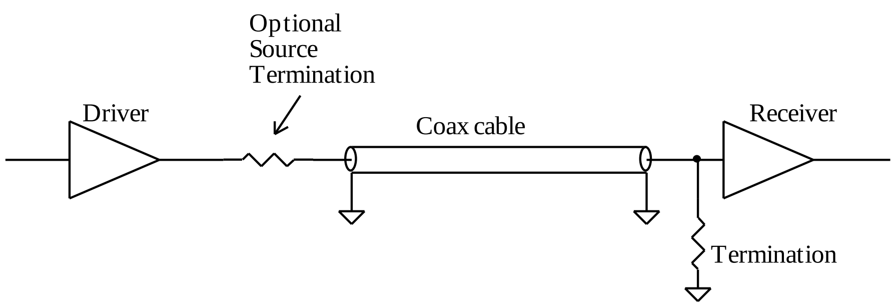


Figure 2 A Typical Transmission Line Using a Coax

The simplest way to solve the problem is to use a termination resistor which is the same value as the impedance of the cable. Unfortunately, many sources (such as TTL gates) cannot drive a 50 or 110 Ohm load. We therefore present here several alternatives. The choice between them depends on whether you have control over the kind of driver (source) you are using, speed requirements, etc. First we will look at the termination choices on the GT650 Series TIA's.

Input Characteristics of the GT668 TIAs

The GT668 Series instruments give you a lot of choices for setting up the inputs.

- The main channels of the GT668 have built-in software selectable 1kΩ or 50Ω input impedances to ground. The 1 kΩ impedance is for connecting scope probes and other high impedance require-ments.

- The EXT CLK and ARM inputs of all models have a 1kΩ input impedance to ground. If 50Ω termination is needed it should be installed on the connector – either as feed-thru 50Ω termination or as a SMA T adapter with a 50Ω plug on one end.

Checking the Quality of Your Signal

It is a good idea to confirm the quality of the signal you are feeding into the instrument. The best way is to probe right at the input to the instrument. With the SMA inputs, the best way is to use an SMA "T" and a scope probe. Simply touch the scope probe to the center conductor in the "T". Since the ground connection must be as short as possible, try to touch the ground of the probe to the outer shell of the "T". Some scope probes have special attachments for this. Do not use another coax cable to the scope!

Using Multiple GT668 Boards in One System

A system can contain multiple GT668 boards working in parallel to measure multiple signals. If the results of the boards need to be correlated the same reference signal must be used. This can be done in one of the following ways:

1. Connect the reference signal through a splitter to each board CLK input and select GT_REF_EXTERNAL as the boards’ reference.

2. For PXI boards in a PXI chassis - connect the reference signal to the chassis and select GT_REF_PXI as the boards’ reference.

3. Output whichever reference signal was selected on one board to the AUX connector (at the front bracket of PXI boards, at the top edge of PCI boards) and select GT_REF_AUX on the other boards.

The AUX connector can also pass between boards one arming signal – the Block Arm, or one of the Tag Arms.

4. EXERCISER

Introduction

The GT668 Time Interval Analyzers comes with a software program named “Exerciser” - a graphic user interface that allows control over all the features of the instrument and calculates some results (such as Frequency, TIE, etc).

Note: Only the Windows versions of the software include the Exerciser.

The installation of the GT668 software installs a shortcut (“icon”) for the Exerciser on the computer desktop and also in the start menu. To run it double-click the icon or click “Start | All Programs | GT668 Time Interval Analyzer | GT668 Exerciser”.

The values of controls and the position and size of open windows are saved (into an internal file) when the Exerciser is closed, and restored when the Exerciser is restarted.

Main Window

When the Exerciser is started the Main Window appears (see Figure 3). The “Device” selection, “Board #”, and “Remap” button will appear only if more than one GT668 instrument is detected in the computer. Initially all the controls except for the “Initialize” button and the device selection (if visible) will be grayed out.

Remap Button

The “Remap” button will open a window that allows mapping of physical board numbers to a logical device number (by default the two numbers are the same) (see Remap Window below).

Initialize Button

The “Initialize” button will initialize the selected instrument and enable all the other controls on the Main Window.

Configure Button

The “Configure” button opens the Configuration Window which allows control over the features of the GT668 instrument (see Remap Window below).

Calibrate Button

The “Calibrate” button runs internal calibration of the board’s time measurements to compensate for effects of changes in the ambient temperature.

Info Button

The “Info” button opens a dialog box displaying information about the instrument and driver revisions and the date of the last factory calibration.

Measurements Textbox

The “Measurements” textbox defines how many measurements to run. It is limited only by the amount of memory available in the computer.

Auto Save Checkbox

The “Auto Save” checkbox enables automatic saving of all measured results (at the cost of slowing down the maximum rate of continuous measurements). If this control is checked during closing (or crash) of the Exerciser when it is restarted the results will show in the window (and can be viewed in details with the Graph and Results windows).

Frequency Option

The “Frequency” option button selects calculating frequency from the timetags generated.

Period Option

The “Period” option button selects calculating period from the timetags generated.

Continuous Time Option

The “Continuous Time” option button selects calculating time interval between consecutive timetags generated on each channel.

TIE Option

The “TIE” option button selects calculating Time Interval Error from the timetags generated. If the “Auto” option is selected the TIE will calculate the frequency from the first and last timetags generated, otherwise it will use the frequency specified by the user.

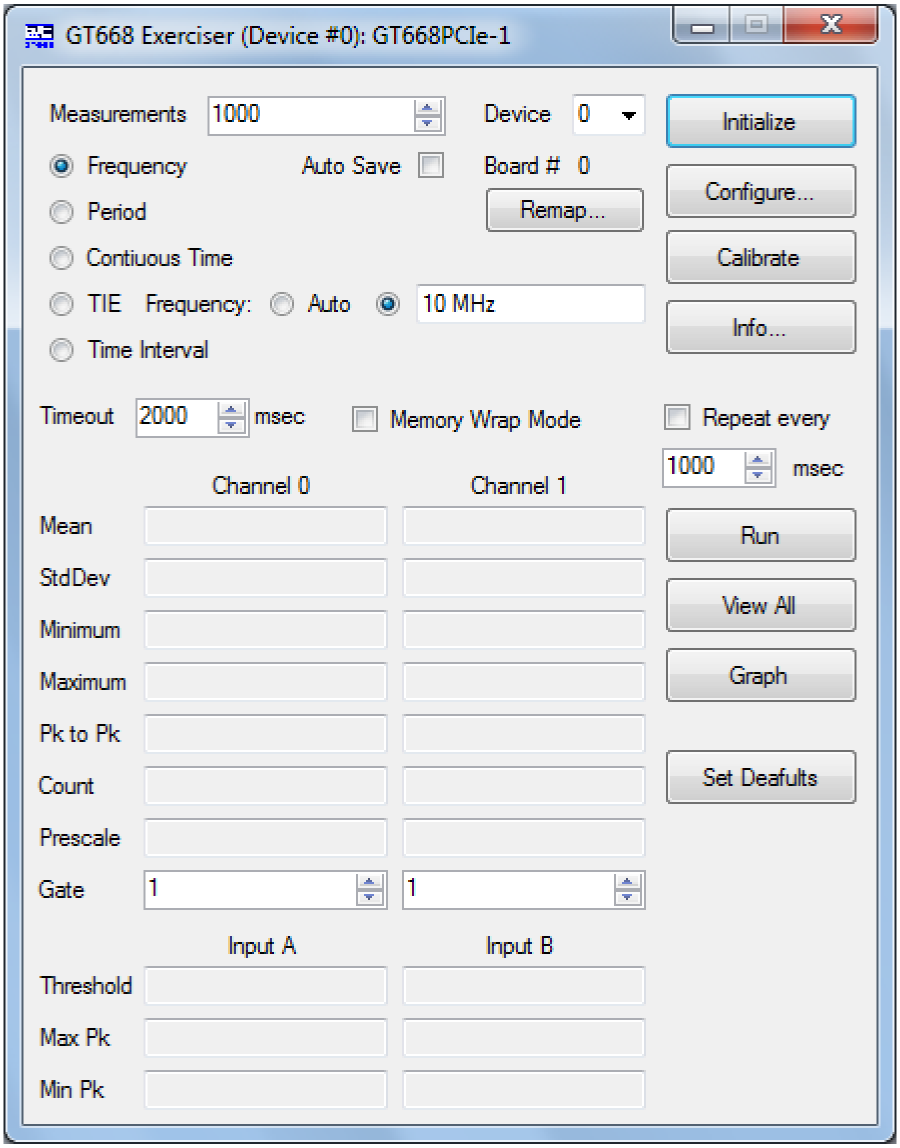


Figure 3 Exerciser - Main Window

Time Interval Option

The "Time Interval" option button selects calculating time interval between the channels (the corresponding timetags generated on each channel). This function calculates the signal's period from first and last timetag and subtracts all full periods from the result (this requires at least two measurements).

Timeout Textbox

The "Timeout" textbox configure a timeout value in milliseconds. If no timetag is found within the timeout value the measurement will stop.

Memory Wrap Mode Checkbox

The "Memory Wrap Mode" checkbox enables using the instrument's memory in "wrap" mode (see Configuring Measurements Memory in Chapter 5 below).

NOTE: USB instrument have no such memory so the "wrap" option does not exist.

Repeat Checkbox
The “Repeat every” checkbox will cause the run to repeat itself at the specified interval (milliseconds).

Run Button
The “Run” button will start running the requested measurements. While measurements execute the button caption will change to “Abort”, allowing the user to abort the measurements by clicking again on it.

View All Button
The “View All” button opens the Results Window that displays all the measured results. This window also allows saving the results into a user specified file. See Figure 6.

Graph Button
The “Graph” button opens the Graph Window that displays all the measured results in graph form. See Figure 8.

Set Defaults Button
The “Set Defaults” button initializes the instrument and Exerciser program to default setting.

Gate
The “Gate” field under the results summary allows configuring an averaging gate for each measurement channel. This option is enabled and used only for Frequency, Period, and TIE measurements. When disabled or set to 1 (the default) no averaging is done. When it is set to a greater value, measurements will be averaged over each gate (there is no dead time between gates). For example if gate is 1000, every 1000 measurements will be averaged into one result.

Results Summary
The rest of the Main Window displays summary of the results.

For each measurement channel this area displays:

- Mean – statistical mean.
- StdDev – statistical standard deviation.
- Minimum – smallest result.
- Maximum – biggest result.
- Peak to Peak – difference between maximum and minimum.
- Count – number of measurements.
- Prescale – the total prescaling used = (input prescale) x (number of skipped tags plus one).

For each input signal:

- Threshold – the threshold level at which timetags are generated.
- Max Pk – Maximum peak voltage of the signal.
- Min Pk – Minimum peak voltage of the signal.

Remap Window

This window allows the user to map board numbers (that are determined by the order in which the BIOS and Windows detect the boards – which is determined by the computer hardware).

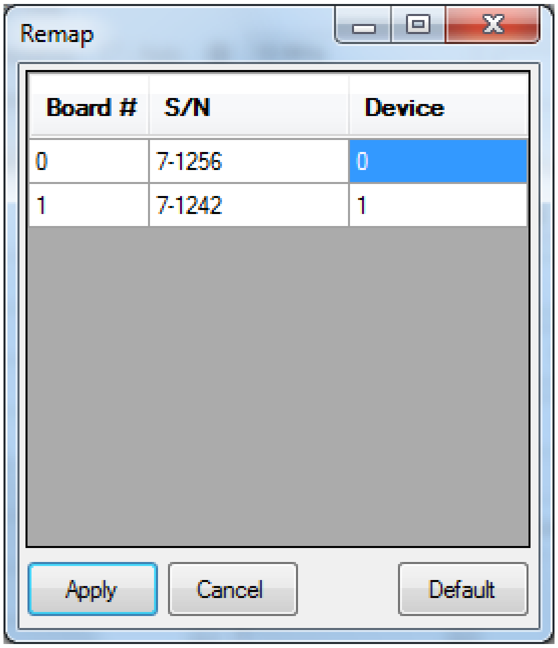


Figure 4 Remap Devices

The first column contains the physical board number, the second column identifies the serial number of the board, and the third column contains the logical device number. The device number can be modified to any number between 0 and 31. If that number is already in use on another board – this other board device number will be modified to the first unused device number.

Apply Button

The "Apply" button will accept the current content of the mapping table, assign the device numbers to the corresponding board numbers and close the window.

Cancel Button

The "Cancel" button will discard the current values in the table and close the window leaving the mapping the same as it was before.

Default Button

The "Default" button will set all the device numbers to default values, i.e. to be the same as the board numbers.

Configuration Window

This window allows the user to control all the features of the GT668 instrument.

Input A/B

These frames configure the input signals.

- Impedance – select input impedance as 50Ω or 1kΩ.

- Coupling – select between DC or AC input coupling.
- Threshold – select signal threshold level at which timetags will be generated. The threshold can be set as percentage of the peak-to-peak voltage (when “Auto” is selected) or as an absolute value in Volts (when “Man” is selected).
- Prescale – select input prescaling that will ensure at least 250 nsec between timetags (prescale the signal to a value of 4MHz or less). If “Auto” is selected the software will perform a quick frequency measurement and select the smallest prescale value that satisfies this condition.

Channel 0/1

These frames configure the measurement channels.

- Enable – enable or disable the measurement channel.
- Source – selects the signal source and polarity to be measured.
- Skip – configure how many timetags to skip after each timetag measured (allows reducing the measurement rate to very low values - such as 1PPS or less - when needed).

Channel 0/1 Arm

These frames configure the timetag arming of the measurement channels.

- Source – select the source of the arming as one of:
    - Auto – arm automatically as soon as measurement circuit is ready.
    - By Command – arm on user command (click on the button to the right).
    - External – use the external arming signal.
    - Aux – use arming signal from another GT668 passed through the AUX cable.
    - Other Channel – use signal selected as source of the other channel.

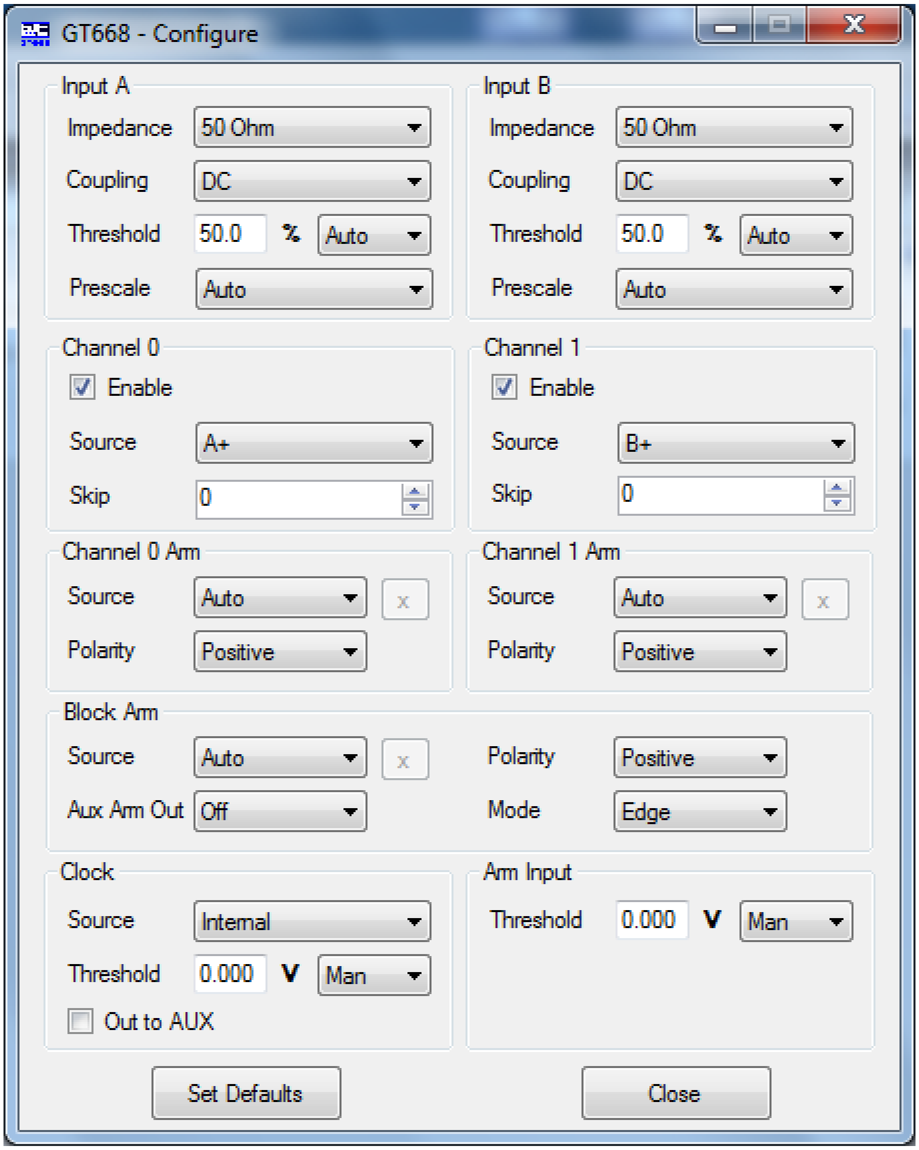


Figure 5 Configuration Window

- Polarity – select the polarity that will cause arming.

Block Arm

This frame configures the block arming (which enables both channels).

- Source – select the source of the arming as one of:
  - Auto – arm automatically as soon as measurement circuit is ready.
  - By Command – arm on user command (click on the button to the right).
  - External – use the external arming signal.
  - Aux – use arming signal from another GT668 passed through the AUX cable.
  - Channel 0 – use signal selected as source of channel 0.

    - Channel 1 – use signal selected as source of channel 1.

- Polarity – select the polarity that will cause arming.
- Aux Arm Out – select which arming signal will be output to the AUX cable.
    - Off – no output.
    - Block Arm – output the block arm.
    - Chan 0 Arm – output channel 0 arm.
    - Chan 1 Arm – output channel 1 arm.
- Mode – select block arming mode:
    - Edge – block arming will happen on the edge of the signal of the selected polarity.
    - Level – block arming will happen when the signal level has the selected polarity.

Clock

This frame configures the reference for the Timebase clock of the instrument.

- Source – select the source of the reference as one of:
    - Internal – use the internal 10MHz TCXO as reference.
    - External 10MHz – use an external 10MHz signal connected to the CLK input.
    - External 5MHz – use an external 5MHz signal connected to the CLK input.
    - AUX – use a 10MHz signal coming through the AUX cable from another GT668.
- Threshold – select signal threshold level for the external clock. The threshold can be set as percentage of the peak-to-peak voltage (when “Auto” is selected) or as an absolute value in Volts (when “Man” is selected).
- Out to AUX – if this checkbox is checked the clock from this board will be output to the AUX cable. Only one instrument on the same cable should have this option enabled.

Arm Input

This frame configures the ARM input signal.

- Threshold – select signal threshold level for the external arm. The threshold can be set as percentage of the peak-to-peak voltage (when “Auto” is selected) or as an absolute value in Volts (when “Man” is selected).

Set Defaults

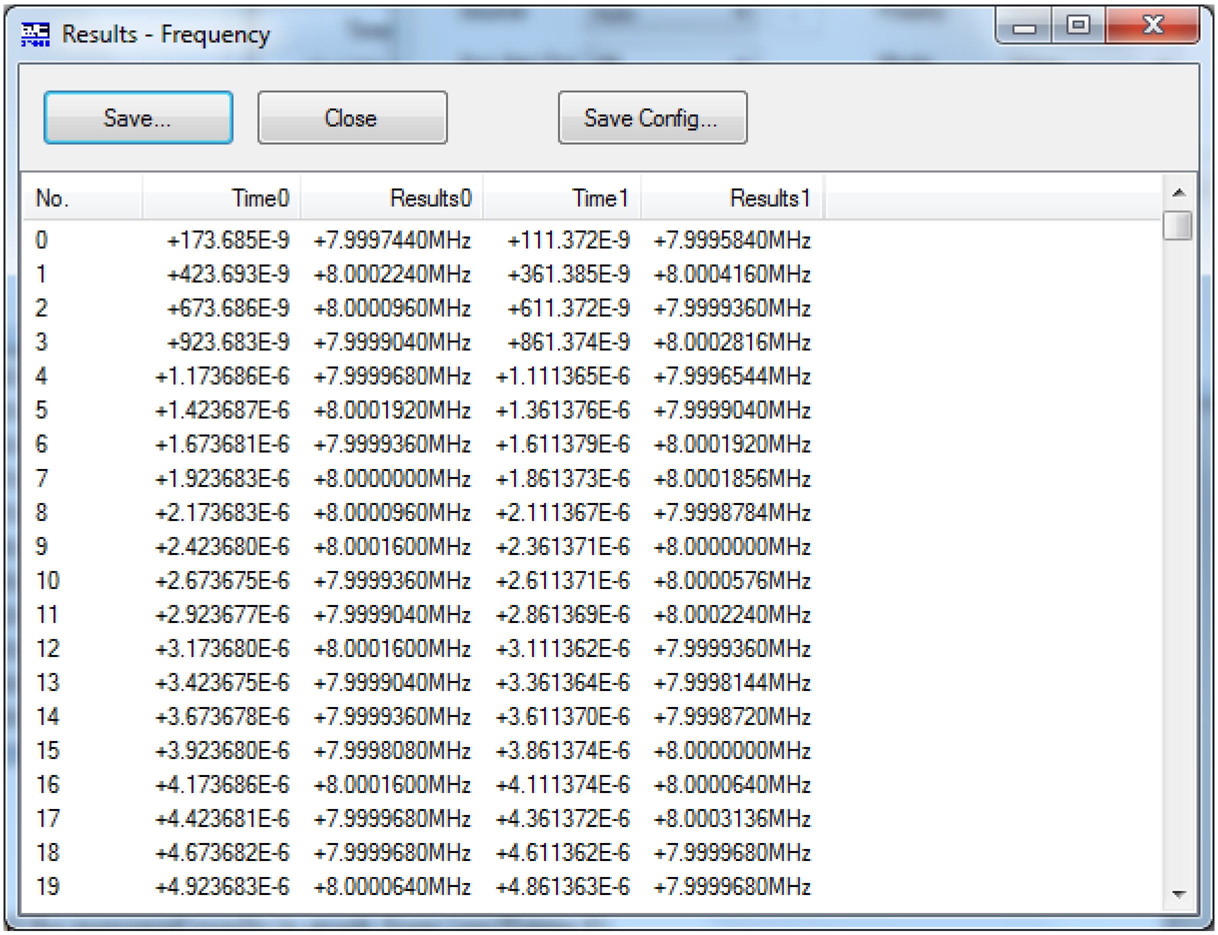
Clicking this button will set the GT668 configuration to default values.

Close

Clicking this button will close the Configuration Window.

Results Window

This window displays all the measured results and allows saving them to a file – either as a simple table or in Excel compatible ".cvs" format (see Figure 6).



No.	Time0	Results0	Time1	Results1
0	+173.685E-9	+7.9997440MHz	+111.372E-9	+7.9995840MHz
1	+423.693E-9	+8.0002240MHz	+361.385E-9	+8.0004160MHz
2	+673.686E-9	+8.0000960MHz	+611.372E-9	+7.9999360MHz
3	+923.683E-9	+7.9999040MHz	+861.374E-9	+8.0002816MHz
4	+1.173686E-6	+7.9999680MHz	+1.111365E-6	+7.9996544MHz
5	+1.423687E-6	+8.0001920MHz	+1.361376E-6	+7.9999040MHz
6	+1.673681E-6	+7.9999360MHz	+1.611379E-6	+8.0001920MHz
7	+1.923683E-6	+8.0000000MHz	+1.861373E-6	+8.0001856MHz
8	+2.173683E-6	+8.0000960MHz	+2.111367E-6	+7.9998784MHz
9	+2.423680E-6	+8.0001600MHz	+2.361371E-6	+8.0000000MHz
10	+2.673675E-6	+7.9999360MHz	+2.611371E-6	+8.0000576MHz
11	+2.923677E-6	+7.9999040MHz	+2.861369E-6	+8.0002240MHz
12	+3.173680E-6	+8.0001600MHz	+3.111362E-6	+7.9999360MHz
13	+3.423675E-6	+7.9999040MHz	+3.361364E-6	+7.9998144MHz
14	+3.673678E-6	+7.9999360MHz	+3.611370E-6	+7.9998720MHz
15	+3.923680E-6	+7.9998080MHz	+3.861374E-6	+8.0000000MHz
16	+4.173686E-6	+8.0001600MHz	+4.111374E-6	+8.0000640MHz
17	+4.423681E-6	+7.9999680MHz	+4.361372E-6	+8.0003136MHz
18	+4.673682E-6	+7.9999680MHz	+4.611362E-6	+7.9999680MHz
19	+4.923683E-6	+8.0000640MHz	+4.861363E-6	+7.9999680MHz

Figure 6 Results Window

Save
Click this button to save the results to a file.

Close
Click this button to close this window.

Save Config
Click this button to open a dialog box to configure some of the save file parameters (see Figure 7 below).

Save Configuration Window

This window allows configuring of headers for the results save files. It contains two tabs, one to configure text files and one to configure csv files. If the text string "[Meas]" appears in any header, it will be replaced in the file with the name of the measurement (Frequency, Period, etc.). Each tab has the same options:

No Headers
If this option is checked the file will have no headers and the rest of the options will be disabled.

Title
Title line at the top of the file.

Chan 0 Time
Header for the column of the measurement times on measurement channel 0.

Chan 0 Result
Header for the column of measurements from measurement channel 0.

Chan 1 Time
Header for the column of the measurement times on measurement channel 1.

Chan 1 Result
Header for the column of measurements from measurement channel 1.

Add ordinal number column
Add a column on the left with the ordinal number of the measurements (starting from 0).

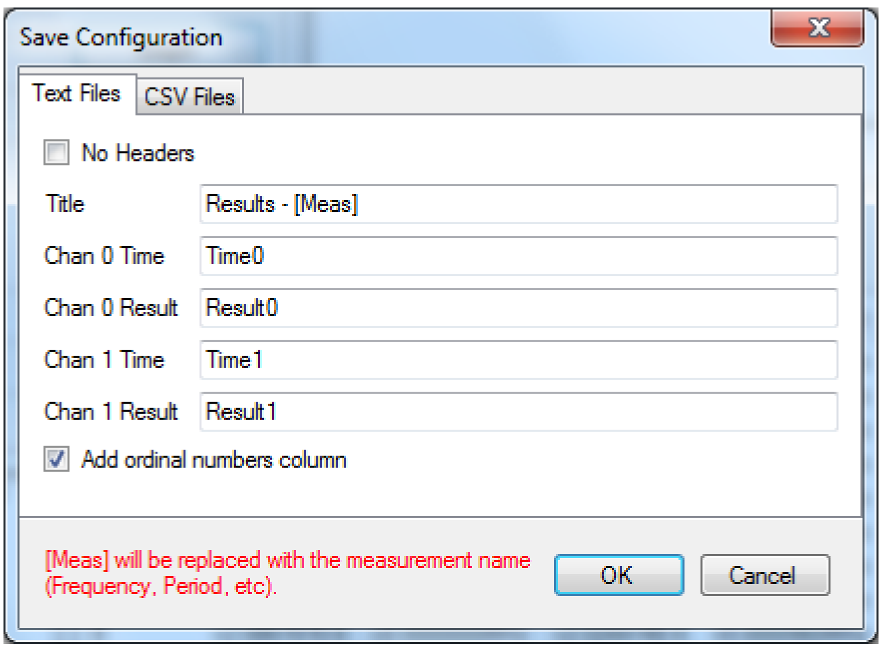


Figure 7 Save Configuration

Graph Window

This window displays all the measured results in graph form (see Figure 8).

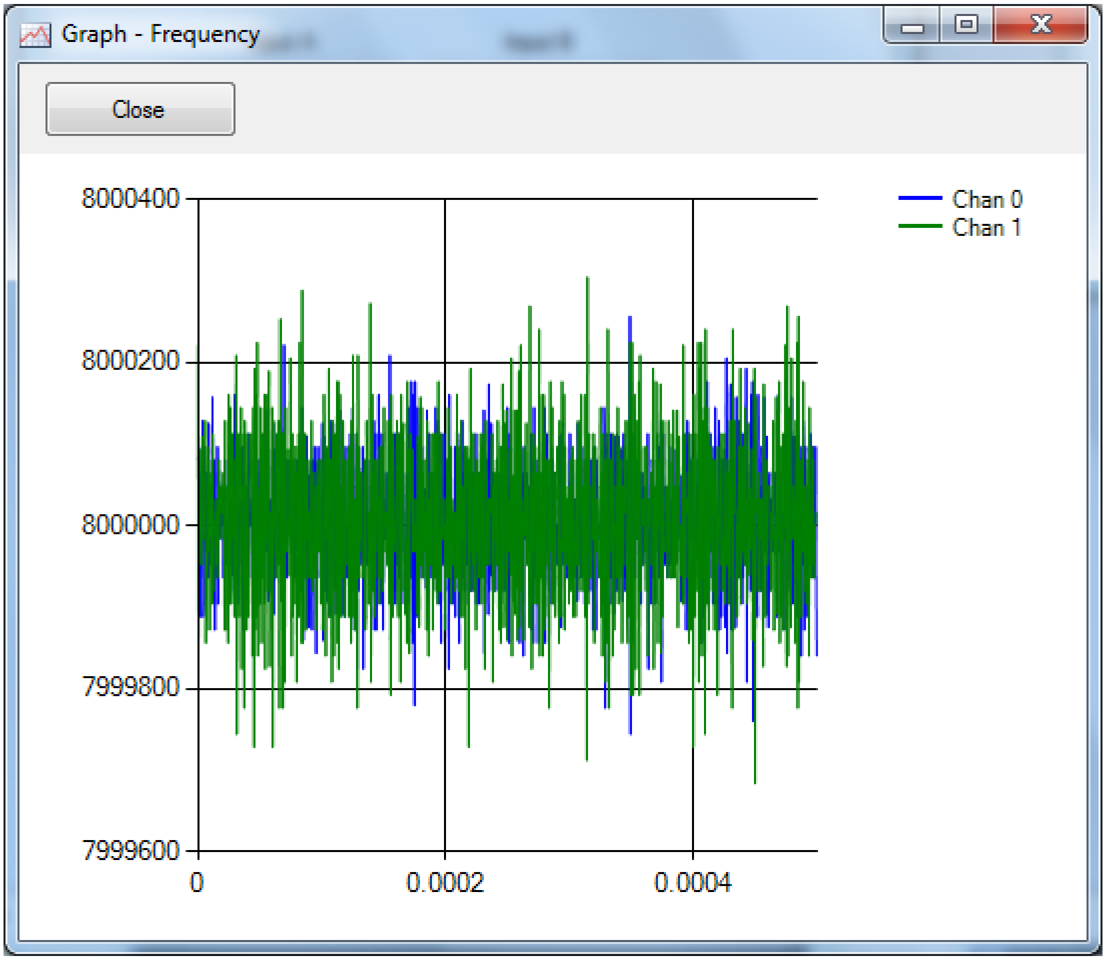


Figure 8 Graph Window

5. Software Driver

Introduction

The GT668 Time Interval Analyzers are high-throughput, system-oriented instruments. The software supplied with the instruments is therefore optimized for system use, providing the fastest possible control for the boards. The driver software is written in the C language, and was tested with Microsoft® Visual Studio under Windows, and with GCC under Linux.

Using Multiple Boards

When multiple GT668 boards, maybe in combination of other GuideTech boards such as the GT210, are installed in the same computer, all these boards receive consecutive physical board numbers starting at zero in the order in which they were scanned by the driver, the BIOS and/or the operating system. To allow more user-friendly addressing the boards are accessed by logical device number which can be mapped by the user to any physical board number. By default the logical device number is the same as the physical board number.

NOTE: The order of the physical board numbers is usually not the same as the order of the physical slots into which they are inserted.

Most of the driver functions are written to control one GT668 device – defined as the current device. The following function are used to locate physical boards, map them to logical device numbers, initialize them, and switch between initialized devices,:

- *GT668EnumerateBoards()* – find and enumerate all the GT668 boards in the system in physical board number order.
- *GT668BoardNumber()* – map a logical device number to a physical board number.
- *GT668Initialize()* – initialize one logical device number and make it the current device.
- *GT668Select()* – switch to a different initialized device.

Compiling Your Program

C/C++ Programs

The “Include” directory contains a C header file named GT668DRV.H that needs to be included in any C/C++ program that uses the driver. This file contains all the needed function prototypes, structures, and constants definitions.

Visual Basic Programs

The “Support\VB.NET” directory contains a file named “GT668Def.vb” with all the declarations and definitions required to call the driver from any Visual Basic .Net program. To use the driver from a Visual Basic program include this file in your project.

Windows

The software for the GT668 board contains two DLL libraries named GT668.DLL and GTPCI.DLL located in the installation directory (during the installation a copy of both libraries was copied into the “Windows\System32” directory). An import library named GT668.LIB is located in the same “Lib” directory. To link a program to the driver, link the import library to your program.

Linux

The software for the GT668 board contains two shared object libraries named libgt668.so and libgtpci.so located in the “Lib” directory. To link a program to the driver, link the libgt668.so and libgtpci.so libraries to your program.

Sample Programs

The software contains sample programs that illustrate the use of the driver. We suggest that you use these programs as a starting point for your own programs. The sample programs are in the sub-directories of the “Samples” directory..

Error Codes and Return Values

Most of the functions return a ‘*false’* value in case of error. A call the function *GT668GetError()* will return an non-zero error code. To convert the error code to an error message call the function *GT668GetErrorMessage()*.

Configuring Measurements Memory

NOTE: This section does not apply to USB instrument. When a TIA is connected through USB there is only a small buffer of about 6000 timetags on the board and the application must read the timetags fast enough to prevent overflow. In case of such overflow the measurements stop.

The memory in the GT668 board’s driver can be read while measurements are taking place. The driver's functions for reading the data out will read only the measurements that are already available. There are two modes of operation for the memory — the **No-Wrap** mode and the **Wrap** mode. The term refers to the fact that in the wrap mode the instrument will "wrap around" to the beginning of the memory after filling it. The choice between these modes depends on the measurement rate, the number of consecutive measurements that you need, the rate that your PC can read and process the data, and the percentage of time that the program in the PC can allocate to the TIA.

The measurement rate is not necessarily the input frequency. This is because you may have prescaling, or some arming control which causes "skipping" of some of the input pulses. What is of concern in our discussion here is the rate at which timetags are stored in memory. In addition, if the input frequency is too low, the measurement circuitry stores "dummy" timetags in memory to guarantee a minimum sampling rate. These timetags are taken care of by the driver functions for reading the board so that an application program does not see them. Their only effect is in forcing a minimum reading rate in the continuous (wrap) mode, or using up part of the memory in the No-Wrap mode. The rate of these dummy timetags is about 6104 Hz. If the rate of real (actual) timetags is just above this frequency, there will be no dummy timetags generated. However, if the rate of real timetags is just below this frequency, there will be dummy timetags generated at that rate. Therefore, in the worst case the total rate of data to the memory will be 12,208 per second of which half are true timetags. For the purposes of this discussion, the rate of real timetags is the sum of all the channels. The same "dummy" timetags are used by all the channels.

No-Wrap Mode

In the *No Wrap* mode the instrument stores timetags in its memory until the memory is full. Since new timetags are not acquired once memory is full, the instrument can be used to store data during or after a test, thus allowing the CPU to perform other tasks while the instrument is measuring. As noted before, you do not have to wait until the memory is full since you can read measurements while the instrument is running. You *must* use the No-Wrap mode when the instrument is running at an *average measurement rate* which is higher than the rate at which you can read and process measurements from the PC.

You *should* use the No-Wrap mode when the number of continuous measurements you need will fit in the memory. The only complication to this are the dummy timetags mentioned above, which reduce the amount of memory available for true timetags. This occurs only when the rate of true timetags is too low.

Wrap Mode

In the Wrap Mode the instrument will continuously write to the memory. The memory in this mode works like a FIFO (first in first out), and is analogous to a funnel. Samples enter from one end while the PC extracts the samples from the other end.

Unlike the *No Wrap* Mode, the FIFO or funnel must be unloaded periodically to avoid overflowing it, causing the loss of data. The percentage of CPU time that must be allocated to unloading the FIFO depends on the timetag rate and the rate at which the PC can extract data from the instrument.

Summary of Memory Modes

As a summary for the above discussion and to help you choose the best approach for your application, we would like to emphasize the following key points:

1. If the *average* measurement rate (rate of real timetags) is higher than the rate that the PC can read the data, you *must* use the No-Wrap mode for the memory. In this case, your maximum number of consecutive measurements is limited by the size of the driver memory. In addition, when the measurement rate drops below 6104 Hz for the 2-channel model, the capacity of the memory is further reduced by "dummy" timetags which are stored in memory by the measurement circuitry in order to guarantee a minimum measurement rate.

2. If the *average* measurement rate (rate of real timetags) is lower than the rate that the PC can read the data, you can use either the Wrap mode or the No-Wrap mode for the memory. If you use the Wrap mode, your program must read the data at the average measurement rate on a continuous basis in order to avoid overflowing the memory. When the measurement rate drops below 6104 Hz, the instrument will store "dummy" timetags in the memory, which forces your program to read timetags at that minimum rate, even if the rate of actual timetags is very low.

6. Function Reference

List of Functions

Below is a list of the driver functions for easy selection. After selecting the function you need, you may look it up in detail in the reference section below which is in alphabetical order.

Initialization and Calibration Functions

GT668EnumerateBoards()	Find all GT668 boards and identify their board number.
GT668BoardNumber()	Map a board number to a logical device number.
GT668Initialize()	Initialize a logical device.
GT668SystemInitialize()	Initialize all the devices in the system.
GT668IsInitialized()	Check is device is initialized
GT668Close()	Close the current device.
GT668SystemClose()	Close all the devices in the system.
GT668Select()	Select a device.
GT668SelfCal()	Calibrated the current device.

Instrument Setup Functions

GT668InitDefault()	Initialize device to default setup.
GT668GetInputCoupling()	Retrieve the input coupling setup.
GT668GetInputImpedance()	Retrieve the input impedance setup.
GT668GetInputPrescale()	Retrieve the input prescaling setup.
GT668GetInputThreshold()	Retrieve the input threshold setup.
GT668GetArmAuxOut()	Retrieve the AUX connector arming output setup.
GT668GetBlockArm()	Retrieve the block arming setup.
GT668GetMeasInput()	Retrieve the input selection for a measurement channel.
GT668GetMeasSkip()	Retrieve the measurement channel skip rate setup.
GT668GetMeasTagArm()	Retrieve the tag arming setup for a measurement channel.
GT668GetMeasEnable()	Retrieve the measurement channel enabling setup.
GT668GetMeasGate()	Retrieve the measurement channel averaging gate count setup.
GT668GetMemoryWrapMode()	Retrieve the memory configuration setup.

GT668GetReferenceClock()	Retrieve the reference clock setup.
GT668GetT0Mode()	Retrieve the T0 configuration setup.
GT668SetArmAuxOut()	Select the AUX connector arming output.
GT668SetBlockArm()	Setup the block arming mode.
GT668SetInputCoupling()	Set the input coupling mode.
GT668SetInputImpedance()	Set the input impedance mode.
GT668SetInputPrescale()	Set the input prescaling value.
GT668SetInputThreshold()	Setup the input threshold.
GT668SetMeasEnable()	Enable or disable a measurement channel.
GT668GetMeasGate()	Set the measurement channel averaging gate count setup.
GT668SetMeasInput()	Select the input for a measurement channel.
GT668SetMeasSkip()	Setup the measurement channel skip rate.
GT668SetMeasTagArm()	Setup the tag arming for a measurement channel.
GT668SetMemoryWrapMode()	Setup the memory configuration (Wrap or No-Wrap).
GT668SetReferenceClock()	Select the reference clock source.
GT668SetT0Mode()	Configure the T0 mode.

Measurement Functions

GT668AutoPrescale()	Measure the minimum prescaling needed for a signal.
GT668GetTotalPrescale()	Retrieve the total prescaling used for measurement channel.
GT668MeasureAmplitude()	Measure signal amplitude.
GT668StartMeasurements()	Start measurements on enabled channel(s).
GT668StopMeasurements()	Stop measurements on enabled channel(s).
GT668BlockArmCommand()	Generate block arm if software arming is configured.
GT668TagArmCommand()	Generate tag arm if software arming is configured.
GT668GetT0()	Get T0 value for current measurements.
GT668GetT0Ex()	Get T0 value for current measurements as seconds and fraction.
GT668ReadTimetags()	Read timetags on one or both channels.
GT668ReadTimetagsEx()	Read timetags on one or both channels in GtiRealTime format.
GT668ReadTTUnpacked()	Read timetags on one or both channels in unpacked seconds/fraction format.

GT668ReadRaw()	Read raw timetags data.
GT668ConvertRawToTimetags()	Convert raw data to timetags.
GT668ConvertRawToTimetagsEx()	Convert raw data to timetags in GtiRealTime format.
GT668ConvertRawToTTUnpacked()	Convert raw data to timetags in unpacked seconds/fraction format.

Inter-board Skew Functions

GT668CheckAux()	Check that the flat ribbon cable is connected between boards.
GT668BoardsSkewSelfCal()	Calibrate inter-board skew.
GT668GetBoardsSkewCal ()	Retrieve skew correction values.

Real Time Clock Functions

GT668SetRealTime()	Synchronizes real time clock to a given time.
GT668GetRealTime()	Read current real time.
GT668IsRealTimeSet()	Check if real time clock was set since power on.

Output Functions

GT668SetOutput()	Configure board outputs.
GT668RealTimeOutput()	Send a real time output command.
GT668GetRealTimeOutputStatus()	Retrieve number of pending output commands.
GT668GetRealTimeOutputMaxPend()	Retrieve maximum possible number of pending commands.

Miscellaneous Functions

GT668GetBaseSeconds()	Retrieve current time seconds offset.
GT668SetBaseSeconds()	Set time seconds offset.
GT668GetError()	Retrieve current error code.
GT668GetErrorMessage()	Convert error code to an error message.
GT668ClearError()	Clear current error.
GT668GetSerialNumber()	Retrieve the board serial number.
GT668GetBoardModel()	Retrieve the board model name.
GT668GetBoardRevision()	Retrieve the board Revision.
GT668GetMemorySize()	Retrieve the size of the memory available for timetags.

Functions Reference

The boolean type '**bool**' was replaced with **GT_Bool**, and the Boolean values '*true*' and '*false*' were replaced with *GT_True* and *GT_False*. The values are the same but this allowed for better portability between different compilers.

GT_Bool GT668AutoPrescale(int ch, GtiPrescale *presc)

This function measures the minimum input prescaling needed for the input signal.

ch	0 – Input channel A.
	1 – Input channel B.
presc	Pointer to a place to store the prescaling setting measured.

Return value: Returns *GT_False* if measurement failed, *GT_True* if measurement was successful.

---

int GT668BlockArmCommand(GT_Bool arm)

This function sets the level of the software arm signal for block arming.

arm	true – set arm signal high.
	GT_False – set arm signal low.

Return value: Returns *GT_False* if setting failed, *GT_True* if setting was successful.

---

int GT668BoardNumber(int dev, int board)

This function maps a physical board number (determined by the order in which the operating system scans all computer slots) into a logical device number. By default device number is the same as the board number.
NOTE: This function should be called before calling *GT668Initialize()* for the same board number or the same device number.

dev	logical device number.
board	physical board number.

Return value: Returns *GT_False* if setting failed, *GT_True* if setting was successful.

GT_Bool GT668BoardsSkewSelfCal(unsigned int dev_mask)

Calibrate inter-board skew for all the devices specified by dev_mask. Skews are calibrated relative to the currently selected board.

NOTE: The physical order of the board is determined during factory calibration of a set of boards and should not be changed. Changing the order will cause error in this function.

NOTE: This function requires GT668 board h/w revision 6 or newer.

dev_mask bits mask where bit ‘i’ is set if device #i should be calibrated.

Return value: Returns *GT_False* if measurement has failed (on at least one board), *GT_True* if measurement was successful.

GT_Bool GT668CheckAUX(unsigned int dev_mask)

Check if a ribbon is attached to the AUX connector of all the devices specified by dev_mask.

dev_mask bits mask where bit ‘i’ is set if device #i should be checked.

Return value: Returns *GT_False* if check has failed (at least one board is not connected to the cable), *GT_True* if check was successful.

void GT668ClearError(void)

This function clears the current error.

Return value: None.

void GT668Close(void)

This function closes the current board.

Return value: None.

GT_Bool GT668ConvertRawToTimetags(void *pRawBuf, unsigned int tags_cnt, unsigned int *tags_used, double *pTags0, unsigned int NumTags0, unsigned int *ActNumTags0, double *pTags1, unsigned int NumTags1, unsigned int *ActNumTags1)

This function converts raw time tags data (read using the *GT668ReadRaw()* function) into user provided buffers. The function will read up-to the requested number of time tags, but will stop sooner if the number of raw time tags available is smaller.

pRawBuf Buffer of raw data.

tags_cnt The number (in 32-bit words) of raw data words available to convert.

tags_used Pointer to a place to store the number (in 32-bit words) of raw data words actually converted. Can be NULL.

pTags0 Buffer for time tags from measurements channel 0. Can be NULL if **NumTags0** is 0.

NumTags0 The number of time tags from channel 0 to read.

ActNumTags0 Pointer to a place to store the number of time tags actually read from channel 0. Can be NULL if **NumTags0** is 0.

pTags1 Buffer for time tags from measurements channel 1. Can be NULL if **NumTags1** is 0.

NumTags1 The number of time tags from channel 1 to read.

ActNumTags1 Pointer to a place to store the number of time tags actually read from channel 1. Can be NULL if **NumTags1** is 0.

Return value: Returns *GT_False* if conversion has failed, *GT_True* if conversion was successful.

GT_Bool GT668ConvertRawToTimetagsEx(void *pRawBuf, unsigned int tags_cnt, unsigned int *tags_used, GtiRealTime *pTags0, unsigned int NumTags0, unsigned int *ActNumTags0, GtiRealTime *pTags1, unsigned int NumTags1, unsigned int *ActNumTags1)

This function converts raw time tags data (read using the *GT668ReadRaw()* function) into user provided buffers. The function will read up-to the requested number of time tags, but will stop sooner if the number of raw time tags available is smaller. The time tags are in GtiRealTime format which is made of an unsigned integer for the seconds and a double precision floating point for the fraction. This format does not lose precision when the time values become big.

pRawBuf	Buffer of raw data.
tags_cnt	The number (in 32-bit words) of raw data words available to convert.
tags_used	Pointer to a place to store the number (in 32-bit words) of raw data words actually converted. Can be NULL.
pTags0	Buffer for time tags from measurements channel 0. Can be NULL if **NumTags0** is 0.
NumTags0	The number of time tags from channel 0 to read.
ActNumTags0	Pointer to a place to store the number of time tags actually read from channel 0. Can be NULL if **NumTags0** is 0.
pTags1	Buffer for time tags from measurements channel 1. Can be NULL if **NumTags1** is 0.
NumTags1	The number of time tags from channel 1 to read.
ActNumTags1	Pointer to a place to store the number of time tags actually read from channel 1. Can be NULL if **NumTags1** is 0.

Return value: Returns *GT_False* if conversion has failed, *GT_True* if conversion was successful.

GT_Bool GT668ConvertRawToTTUnpacked (void *pRawBuf, unsigned int tags_cnt, unsigned int *tags_used, unsigned int *pSec0, double *pFrac0, unsigned int NumTags0, unsigned int *ActNumTags0, unsigned int *pSec1, double *pFrac1, unsigned int NumTags1, unsigned int *ActNumTags1)

This function converts raw time tags data (read using the *GT668ReadRaw()* function) into user provided buffers. The function will read up-to the requested number of time tags, but will stop sooner if the number of raw time tags available is smaller. The time tags are in GtiRealTime format which is made of an unsigned integer for the seconds and a double precision floating point for the fraction. This format does not lose precision when the time values become big.

pRawBuf	Buffer of raw data.
tags_cnt	The number (in 32-bit words) of raw data words available to convert.
tags_used	Pointer to a place to store the number (in 32-bit words) of raw data words actually converted. Can be NULL.
pSec0	Buffer for the time tags seconds from measurements channel 0. Can be NULL if **NumTags0** is 0.
pFrac0	Buffer for time tags fractions from measurements channel 0. Can be NULL if **NumTags0** is 0.
NumTags0	The number of time tags from channel 0 to read.
ActNumTags0	Pointer to a place to store the number of time tags actually read from channel 0. Can be NULL if **NumTags0** is 0.
pSec1	Buffer for the time tags seconds from measurements channel 1. Can be NULL if **NumTags1** is 0.
pFrac1	Buffer for time tags fractions from measurements channel 1. Can be NULL if **NumTags1** is 0.
NumTags1	The number of time tags from channel 1 to read.
ActNumTags1	Pointer to a place to store the number of time tags actually read from channel 1. Can be NULL if **NumTags1** is 0.

Return value: Returns *GT_False* if conversion has failed, *GT_True* if conversion was successful.

int GT668EnumerateBoards(GT_Bool first)

This function enumerates all GT668 boards in the computer. This function can be called at any time, even before any call to *GT668Initialize().*

first	Should be set to '*GT_True*' on the first call and to '*GT_False*' on all subsequent calls.

Return value: When **first** is set to *GT_True* the function will return the first board number found (lowest number), when **first** is set to *GT_False* it will return the next board number found, if no more boards are found the function will return -1.

GT_Bool GT668GetArmAuxOut(GtiArmAuxOut *aux_out)

This function retrieves the current selection of the AUX bus arm output. (See *GT668SetArmAuxOut()*).

ch	Measurement channel (0 or 1).
aux_out	Place to store the retrieved setting.

Return value: Returns *GT_False* if retrieve failed, *GT_True* if retrieve was successful.

GT_Bool GT668GetActualInputThreshold(GtiSignal sig, double *thr)

This function retrieves the current actual threshold used for the selected signal.

sig	GT_SIG_A – Input channel A.
	GT_SIG_B – Input channel B.
	GT_SIG_CLK – Clock Input.
	GT_SIG_ARM – Arm Input.
thr	Place to store the retrieved threshold.

Return value: Returns *GT_False* if retrieve failed, *GT_True* if retrieve was successful.

unsigned int GT668GetBaseSeconds(void)

This function retrieves the current setting of the base seconds. (See *GT668SetBaseSeconds()*).

Return value: Base seconds value.

GT_Bool GT668GetBlockArm(GtiBlkArmSrc *src, GtiPolarity *pol, GT_Bool *level)

This function retrieves the current setting of the block arm. (See *GT668SetBlockArm()*).

ch	Measurement channel (0 or 1).
src	Place to store the retrieved setting.
pol	Place to store the retrieved setting.
level	Place to store the retrieved setting.

Return value: Returns *GT_False* if retrieve failed, *GT_True* if retrieve was successful.

int GT668GetBoardModel(char *buf, unsigned int buf_size)

This function retrieves the board model name of the current device.

buf	Buffer to store the retrieved model name.
buf_size	Size of **buf**.

Return value: Returns the actual length of the retrieved string. If the function fails it returns zero. If the provided buffer is too small it returns the negative of the required buffer length.

GT_Bool GT668GetBoardRevision(unsigned int *revision)

This function retrieves the revision of the board of the current device.

revision	Place to store the board revision.

Return value: Returns *GT_False* if retrieve failed, *GT_True* if retrieve was successful.

GT_Bool GT668GetBoardsSkewCal(int ref_dev, unsigned int dev_mask, GT_Bool aux_ref, double *skew)

This function retrieves the inter-board skew correction factors to be subtracted from measured skews. Only boards that are sold as a system are calibrated for inter-board skews, and only relative to one reference board.

ref_dev	Device that is the reference for the skews.
dev_mask	Bit mask where bit #i is set to 1 if the skew correction for device #i is to be retrieved (if the bit corresponding to *ref_dev* is set it will be ignored).
aux_ref	Not used – left for backwards compatibility.
skew	Array to store retrieved correction factors with room for one factor per each bit set in *dev_mask*.

Return value: Returns *GT_False* if retrieve failed, *GT_True* if retrieve was successful.

int GT668GetError(void)

This function retrieves the last driver error encountered.

Return value: Returns the error code (0 means no error). Use *GT668GetErrorMessage* to translate it to an error message.

GT_Bool GT668GetErrorMessage(int err, char *buf, unsigned int buf_size)

This function translates an error code (returned from *GT668GetError*) in to an error message.

err	Error code.
buf	Buffer in which to store the error message.
buf_size	Size of buffer.

Return value: Returns *GT_False* if failed, *GT_True* if successful.

GT_Bool GT668GetInputCoupling(GtiSignal sig, GtiCoupling *cpl)

This function retrieves the current settings of the input coupling of an input signal (GT_CPL_DC or GT_CPL_AC). Clock and Arm signal will always return GT_CPL_DC. (See *GT668SetInputCoupling()*).

sig	GT_SIG_A – Input channel A.
	GT_SIG_B – Input channel B.
	GT_SIG_CLK – Clock Input.
	GT_SIG_ARM – Arm Input.
cpl	Place to store the retrieved setting.

Return value: Returns *GT_False* if retrieve failed, *GT_True* if retrieve was successful.

GT_Bool GT668GetInputImpedance(GtiSignal sig, GtiImpedance *imp)

This function retrieves the current settings of the input impedance of an input signal (GT_IMP_LO or GT_IMP_HI). Clock and Arm signal will always return GT_IMP_HI. (See *GT668SetInputImpedance()*).

sig	GT_SIG_A – Input channel A.
	GT_SIG_B – Input channel B.
	GT_SIG_CLK – Clock Input.
	GT_SIG_ARM – Arm Input.
imp	Place to store the retrieved setting.

Return value: Returns *GT_False* if retrieve failed, *GT_True* if retrieve was successful.

GT_Bool GT668GetInputPrescale(GtiSignal sig, GtiPrescale *presc)

This function retrieves the current settings of the input prescaling of an input signal (GT_DIV_AUTO, or GT_DIV_1 through GT_DIV1024). Clock and Arm signal will always return GT_DIV_1. (See *GT668SetInputPrescale()*).

sig	GT_SIG_A – Input channel A.
	GT_SIG_B – Input channel B.
presc	Place to store the retrieved setting.

Return value: Returns *GT_False* if retrieve failed, *GT_True* if retrieve was successful.

GT_Bool GT668GetInputThreshold(GtiSignal sig, GtiThrMode *thr_mode, double *thr_val)

This function retrieves the current settings of the input threshold (mode and value) of an input signal. (See *GT668SetInputThreshold()*).

sig	GT_SIG_A – Input channel A.
	GT_SIG_B – Input channel B.
	GT_SIG_CLK – Clock Input.
	GT_SIG_ARM – Arm Input.
thr_mode	Place to store the retrieved setting.
thr_val	Place to store the retrieved setting.

Return value: Returns *GT_False* if retrieve failed, *GT_True* if retrieve was successful.

GT_Bool GT668GetMeasEnable(int ch, GT_Bool *enb)

This function retrieves the current enable state of a measurement channel. (See *GT668SetMeasEnable()*).

ch	Measurement channel (0 or 1).
enb	Place to store the retrieved setting.

Return value: Returns *GT_False* if retrieve failed, *GT_True* if retrieve was successful.

GT_Bool GT668GetMeasGate(int ch, unsigned int *count)

This function retrieves the current gate count of a measurement channel. (See *GT668SetMeasEnable()*).

ch	Measurement channel (0 or 1).
count	Place to store the retrieved count.

Return value: Returns *GT_False* if retrieve failed, *GT_True* if retrieve was successful.

GT_Bool GT668GetMeasInput(int ch, GtiInputSel *sel)

This function retrieves the current input signal selection of a measurement channel. (See *GT668SetMeasInput()*).

ch	Measurement channel (0 or 1).
sel	Place to store the retrieved setting.

Return value: Returns *GT_False* if retrieve failed, *GT_True* if retrieve was successful.

GT_Bool GT668GetMeasSkip(int ch, unsigned int *presc)

This function retrieves the current setting of the measurements skip rate of a measurement channel. (See *GT668SetMeasSkip()*).

ch	Measurement channel (0 or 1).
rate	Place to store the retrieved setting.

Return value: Returns *GT_False* if retrieve failed, *GT_True* if retrieve was successful.

GT_Bool GT668GetMeasTagArm(int ch, GtiTagArmSrc *src, GtiPolarity *pol)

This function retrieves the current setting of the tag arm of a measurement channel. (See *GT668SetMeasTagArm()*).

ch	Measurement channel (0 or 1).
src	Place to store the retrieved setting.
pol	Place to store the retrieved setting.

Return value: Returns *GT_False* if retrieve failed, *GT_True* if retrieve was successful.

unsigned int GT668GetMemorySize(void)

This function retrieves the total memory size used to store timetags.

Return value: Returns the memory size in bytes.

GT_Bool GT668GetMemoryWrapMode(GT_Bool *wrap)

This function retrieves the current memory “wrap” mode. (See *GT668SetMemoryWrapMode()*).
NOTE: USB instrument have no wrap mode. Calling this function will be ignored.

wrap	Place to store the retrieved setting.

Return value: Returns *GT_False* if retrieve failed, *GT_True* if retrieve was successful.

GT_Bool GT668GetRealTime(unsigned int *sec)

This function retrieves the current real time from the real time clock.

Note: this function cannot execute while measurements are running – it will return GT_False without setting any error code.

sec	Place to store the retrieved time.

Return value: Returns *GT_False* if retrieve failed, *GT_True* if retrieve was successful.

GT_Bool GT668GetRealTimeOutputStatus(unsigned int *pending0, unsigned int *pending1)

This function retrieves how many pending RealTime output commands are in the FIFO buffer of output signals OUT0 and OUT1 (See *GT668SetOutput()* and *GT668RealTimeOutput()*).

Note: this function is supported only on boards of rev 8 or above with FPGA date code of 8/17/2017 or later.

pending0	Place to store the retrieved number of pending output commands for OUT0.
Pending1	Place to store the retrieved number of pending output commands for OUT1.

Return value: Returns *GT_False* if retrieve failed, *GT_True* if retrieve was successful.

GT_Bool GT668GetRealTimeOutputMaxPend (unsigned int *max_pending)

This function retrieves the maximum number of pending RealTime output commands that can be in each FIFO buffer (See *GT668SetOutput()* and *GT668RealTimeOutput()*).

Note: this function is supported only on boards of rev 8 or above with FPGA date code of 8/17/2017 or later.

max_pending	Place to store the retrieved maximum number of pending output commands.

Return value: Returns *GT_False* if retrieve failed, *GT_True* if retrieve was successful.

GT_Bool GT668GetReferenceClock(GtiRefClkSrc *src, GT_Bool *ref_5MHz, GT_Bool *aux_out)

This function retrieves the current setting of the reference clock. (See *GT668SetReferenceClock()*).

src	Place to store the retrieved setting.
ref_5MHz	Place to store the retrieved setting.
aux_out	Place to store the retrieved setting.

Return value: Returns *GT_False* if retrieve failed, *GT_True* if retrieve was successful.

unsigned int GT668GetSerialNumber(void)

This function retrieves the serial number of the current device.

Return value: Returns the serial number.

GT_Bool GT668GetT0(double *t0)

This function retrieves the reference time (t0) for the current set of measurements.
NOTE: If real time is set on the board, or if the instrument was powered for a long time, the resolution of t0 will drop. A double precision floating point number can show about 15 digits of precision – e.g. if 4 digits are needed for the seconds, the fraction will have only 11 digits left which may mean 10ps resolution. If accurate t0 is needed please use the *GT668GetT0Ex()* function.

t0	Place to store the retrieved value.

Return value: Returns *GT_False* if retrieve failed, *GT_True* if retrieve was successful.

GT_Bool GT668GetT0Ex(unsigned int *t0sec, double *t0frac)

This function retrieves the reference time (t0) for the current set of measurements as seconds and fraction, allowing usage of real time (epoch) values without loss of resolution.

t0sec	Place to store the retrieved seconds.
t0frac	Place to store the retrieved fraction.

Return value: Returns *GT_False* if retrieve failed, *GT_True* if retrieve was successful.

GT_Bool GT668GetT0Mode(GT_Bool *arm, GT_Bool *rel)

This function retrieves the setting for the t0 mode. (See *GT668SetT0Mode()*).

arm	Place to store the retrieved setting.
rel	Place to store the retrieved setting.

Return value: Returns *GT_False* if setting failed, *GT_True* if setting was successful.

GT_Bool GT668GetTotalPrescale(int ch, unsigned int *presc)

This function retrieves the total current prescaling of a measurement channel. The total prescaling is calculated by multiplying the prescaling of the input signal selected for this measurement channel and the measurements prescaling. If the input prescaling is set to GT_DIV_AUTO the function will use the prescaling value selected during the last call to *GT668StartMeasurements()*.

ch	Measurement channel (0 or 1).
presc	Place to store the retrieved prescaling value.

Return value: Returns *GT_False* if retrieve failed, *GT_True* if retrieve was successful.

GT_Bool GT668InitDefault(GT_Bool keep_ref)

This function initializes the current board to default setup, with the possible exception of the reference clock selection (to avoid settling time penalty).

keep_ref	If **keep_ref** is *GT_True* the current clock reference selection will be kept unchanged, if it's *GT_False* the clock reference will be set to default value (internal clock).

Return value: Returns *GT_False* if initialization failed, *GT_True* if initialization was successful.

GT_Bool GT668Initialize(int dev)

This function initializes a board and selects it as the current one.

dev	Board number to initialize.

Return value: Returns *GT_False* if initialization failed, *GT_True* if initialization was successful.

GT_Bool GT668IsInitialized(int dev)

This function checks if a board is initialized.

dev	Board number to check.

Return value: Returns *GT_True* if initialized, *GT_False* if not.

GT_Bool GT668IsRealTimeSet(void)

This function checks if the real time clock was set since the board was powered on.

Return value: Returns *GT_True* if real time clock was set, *GT_False* if not.

GT_Bool GT668MeasureAmplitude(int ch, double *posv, double *negv, double minFreq)

This function measures the peak voltages of an input signal. Because the measurement is done by a threshold search the measurement can be slow if the frequency of the signal is low. The measurement takes about 60 msec + 36 * (signal maximum period).

ch	0 – Input channel A.
	1 – Input channel B.
posv	Pointer to a place to store the positive peak voltage.
negv	Pointer to a place to store the negative peak voltage.
minFreq	Minimum frequency of the measured signal. Using a value lower than necessary may slow down the measurement

Return value: Returns *GT_False* if measurement failed, *GT_True* if measurement was successful.

GT_Bool GT668ReadRaw(void *Buf, unsigned int offset, unsigned int ToRead, unsigned int *ActRead)

This function reads raw time tags data into a user provided buffer. The function will read up-to the requested number of words, but will stop sooner if the number of words available is smaller. The data can be converted to double precision time tags by calling the *GT668ConvertRawToTimetags()* function.

Note: In "wrap" mode the **offset** parameter is ignored. Each call to this function will read from the point where the last read ended.

Buf	Buffer for raw data (32-bit for each time tag).
offset	The index (in 32-bit words) of the first raw data word to read. It is ignored in memory "wrap" mode (see note above).
ToRead	The number (in 32-bit words) of the raw data words to read.
ActRead	Pointer to a place to store the number of words actually read.

Return value: Returns *GT_False* if reading has failed, *GT_True* if reading was successful.

GT_Bool GT668ReadTimetags(double *pTags0, unsigned int NumTags0, unsigned int *ActNumTags0, double *pTags1, unsigned int NumTags1, unsigned int *ActNumTags1)

This function reads time tags data into user provided buffers. The function will read up-to the requested number of time tags, but will stop sooner if the number of time tags available is smaller.

pTags0	Buffer for time tags from measurements channel 0. Can be NULL if **NumTags0** is 0.
NumTags0	The number of time tags from channel 0 to read.
ActNumTags0	Pointer to a place to store the number of time tags actually read from channel 0.
pTags1	Buffer for time tags from measurements channel 1. Can be NULL if **NumTags1** is 0.
NumTags1	The number of time tags from channel 1 to read.
ActNumTags1	Pointer to a place to store the number of time tags actually read from channel 1.

Return value: Returns *GT_False* if reading has failed, *GT_True* if reading was successful.

GT_Bool GT668ReadTimetagsEx(GtiRealTime *pTags0, unsigned int NumTags0, unsigned int *ActNumTags0, GtiRealTime *pTags1, unsigned int NumTags1, unsigned int *ActNumTags1)

This function reads time tags data into user provided buffers. The function will read up-to the requested number of time tags, but will stop sooner if the number of time tags available is smaller. The time tags are in GtiRealTime format which is made of an unsigned integer for the seconds and a double precision floating point for the fraction. This format does not lose precision when the time values become big.

pTags0	Buffer for time tags from measurements channel 0. Can be NULL if **NumTags0** is 0.
NumTags0	The number of time tags from channel 0 to read.
ActNumTags0	Pointer to a place to store the number of time tags actually read from channel 0.
pTags1	Buffer for time tags from measurements channel 1. Can be NULL if **NumTags1** is 0.
NumTags1	The number of time tags from channel 1 to read.
ActNumTags1	Pointer to a place to store the number of time tags actually read from channel 1.

Return value: Returns *GT_False* if reading has failed, *GT_True* if reading was successful.

GT_Bool GT668ReadTTUnpacked (unsigned int *pSec0, double *pFrac0, unsigned int NumTags0, unsigned int *ActNumTags0, unsigned int *pSec1,

double *pFrac1, unsigned int NumTags1, unsigned int *ActNumTags1)

This function reads time tags data into user provided buffers. The function will read up-to the requested number of time tags, but will stop sooner if the number of time tags available is smaller. The time tags are in two parallel buffers, one of unsigned integers for the seconds and one of double precision floating pointa for the fraction. This format does not lose precision when the time values become big.

pSec0	Buffer for the time tags seconds from measurements channel 0. Can be NULL if **NumTags0** is 0.
pFrac0	Buffer for time tags fractions from measurements channel 0. Can be NULL if **NumTags0** is 0.
NumTags0	The number of time tags from channel 0 to read.
ActNumTags0	Pointer to a place to store the number of time tags actually read from channel 0.
pSec1	Buffer for the time tags seconds from measurements channel 1. Can be NULL if **NumTags1** is 0.
pFrac1	Buffer for time tags fractions from measurements channel 1. Can be NULL if **NumTags1** is 0.
NumTags1	The number of time tags from channel 1 to read.
ActNumTags1	Pointer to a place to store the number of time tags actually read from channel 1.

Return value: Returns *GT_False* if reading has failed, *GT_True* if reading was successful.

GT_Bool GT668RealTimeOutput(int out, unsigned int sec, double frac)

This function sends to the board a real time value at which to generate an output pulse. The fraction will be converted to the nearest 10 nsec interval.

Notes:
1. Only future times are valid.
2. Calls should be in advancing time order for each output.
3. If two time values are in the same 10 nsec interval the second one is ignored.
4. Up to 64 time values per output can be pending at any time.
5. The function will fail if the output source is not set to GT_OUT_REALTIME (see function *GT668SetOutput()*).

Notes for Revision 8 and above boards with FPGA code ≥ 8/17/2017:
1. Number of pending commands was increased to 512.
2. If attempting to output a command while 512 commands are still pending – the command will be rejected and an error will be returned.
3. If you have a Revision 10 board with older FPGA and you need these features contact GuideTech to get your FPGA field upgraded.
4. A function *GT668GetRealTimeOutputStatus()* was added to check the current number of pending output commands.

out	Output number to select (0 or 1).
sec	Real time seconds value.
frac	Real time fraction (0 to less than 1).

Return value: Returns *GT_False* if failed, *GT_True* if successful.

GT_Bool GT668Select(int dev)

This function selects a board. All subsequent operations will apply to this board.

dev	Board number to select.

Return value: Returns *GT_False* if selection failed (probably board was not initialized), *GT_True* if selection was successful.

GT_Bool GT668SelfCal(void)

This function runs self-calibration.

Return value: Returns *GT_False* if self-calibration failed, *GT_True* if self-calibration was successful.

GT_Bool GT668SetArmAuxOut(GtiArmAuxOut aux_out)

This function selects the arming output to the AUX bus.

aux_out	GT_AUX_OUT_OFF – No arming on the AUX bus.
	GT_AUX_OUT_BA – Output the block arm to AUX.
	GT_AUX_OUT_TA0 – Output the tag arm from channel 0 to AUX.
	GT_AUX_OUT_TA1 – Output the tag arm from channel 1 to AUX.

Return value: Returns *GT_False* if setting failed, *GT_True* if setting was successful.

GT_Bool GT668SetBlockArm(GtiBlkArmSrc src, GtiPolarity pol, GT_Bool level)

This function selects the block arming source.

src	GT_BA_IMM – Arm immediately when the channel is ready.
	GT_BA_SW – Arm by software call to *GT668TagArmCommand()*.
	GT_BA_EXT – Arm from edge of external ARM signal.
	GT_BA_AUX – Arm from arming signal on AUX bus.
	GT_BA_CH0 – Arm from measurement channel 0.
	GT_BA_CH1 – Arm from measurement channel 1.
	GT_BA_OFF – Do not arming (same as disabling the channel).
pol	GT_POL_POS – Use positive edge of external Arm.
	GT_POL_NEG – Use negative edge of external Arm.
level	GT_False – Arming is edge triggered. Measurements start on arming edge and continue until done.
	GT_True – Arming is level triggered. Measurements happen as long as the arming is active. Measurements can be stopped and continued.

Return value: Returns *GT_False* if setting failed, *GT_True* if setting was successful.

GT_Bool GT668SetBaseSeconds(unsigned int sec)

This function sets a base in whole seconds for the following time tags (future time tags value will be relative to this value).
The base seconds is used to eliminate the problem of lost resolution over long time intervals (1000 seconds and more). Because the time tags are represented as double precision floating point numbers when the time value gets above 1000 seconds the least significant bit becomes 1 ps and from there on resolution will be lost. Setting the base seconds of the time tags to 1000 (for example) with a call to *GT668SecBaseSeconds(1000)* will reduce the time tag value to a value close to 0 with least significant bit of 1 fs.
Default value of base seconds is 0.

sec	Base seconds for the following time tags. The value must be less than or equal to the seconds' part of the last time tag.

Return value: Returns *GT_False* if setting has failed (because **sec** is too big), *GT_True* if setting was successful.

GT_Bool GT668SetInputCoupling(GtiSignal sig, GtiCoupling cpl)

This function sets the input coupling of a measured signal to DC or to AC.
NOTE: Clock and Arm signals support only DC coupling.

sig	GT_SIG_A – Input channel A.
	GT_SIG_B – Input channel B.
	GT_SIG_CLK – Clock Input.
	GT_SIG_ARM – Arm Input.
cpl	GT_IMP_DC – DC coupling.
	GT_IMP_AC – AC coupling.

Return value: Returns *GT_False* if setting failed, *GT_True* if setting was successful.

GT_Bool GT668SetInputImpedance(GtiSignal sig, GtiImpedance imp)

This function sets the input impedance of a measured signal to low impedance (50Ω) or to high impedance (1KΩ).
NOTE: Clock and Arm signals support only high impedance.

sig	GT_SIG_A – Input channel A.
	GT_SIG_B – Input channel B.
	GT_SIG_CLK – Clock Input.
	GT_SIG_ARM – Arm Input.
imp	GT_IMP_LO – Low impedance.
	GT_IMP_HI – High impedance.

Return value: Returns *GT_False* if setting failed, *GT_True* if setting was successful.

GT_Bool GT668SetInputPrescale(GtiSignal sig, GtiPrescale presc)

This function sets the hardware input prescaling of a measured signal to divide the input by a power of 2 value between 1 and 1024. The prescaling must be high enough to reduce the measured signal to a frequency under 4MHz.
NOTE: Clock and Arm signals do not support prescaling (same as GT_DIV_1).

sig	GT_SIG_A – Input channel A.
	GT_SIG_B – Input channel B.
	GT_SIG_CLK – Clock Input.
	GT_SIG_ARM – Arm Input.
presc	GT_DIV_AUTO – Select automatically the minimal prescaling value needed to bring the signal under 4MHz.
	GT_DIV_1 – divide by 1 (no prescaling).
	GT_DIV_2 – divide by 2.
	GT_DIV_4 – divide by 4.
	GT_DIV_8 – divide by 8.
	GT_DIV_16 – divide by 16.
	GT_DIV_32 – divide by 32.
	GT_DIV_64 – divide by 64.
	GT_DIV_128 – divide by 128.
	GT_DIV_256 – divide by 256.
	GT_DIV_512 – divide by 512.
	GT_DIV_1024 – divide by 1024.

Return value: Returns *GT_False* if setting failed, *GT_True* if setting was successful.

GT_Bool GT668SetInputThreshold(GtiSignal sig, GtiThrMode thr_mode, double thr_val)

This function sets the input threshold either to a specific voltage value or to a percentage from a measured peak to peak voltage.

sig	GT_SIG_A – Input channel A.
	GT_SIG_B – Input channel B.
	GT_SIG_CLK – Clock Input.
	GT_SIG_ARM – Arm Input.
thr_mode	GT_THR_PERCENTS – Set threshold by percentage of peak-to-peak.
	GT_THR_VOLTS – Set threshold to absolute voltage.
thr_val	Threshold to be used, either in percentage or in Volts according to the value of **thr_mode**.

Return value: Returns *GT_False* if setting failed, *GT_True* if setting was successful.

GT_Bool GT668SetMeasEnable(int ch, GT_Bool enb)

This function enables or disables a measurement channel.

ch	Measurement channel (0 or 1).
enb	GT_True – enabled.
	GT_False - disabled.

Return value: Returns *GT_False* if setting failed, *GT_True* if setting was successful.

GT_Bool GT668SetMeasGate(int ch, unsigned int count)

This function sets the gate for averaging on a measurement channel. By default the count is 1 – i.e. there is no averaging done. If the count is set to N every N measurements will be averaged using a best fit algorithm and one result will be generated. This is useful for measurements such as Frequency, Period Average, and TIE and will reduce their error.

ch	Measurement channel (0 or 1).
count	Averaging gate count (1 to 2000000000).
	GT_False - disabled.

Return value: Returns *GT_False* if setting failed, *GT_True* if setting was successful.

GT_Bool GT668SetMeasInput(int ch, GtiInputSel sel)

This function selects the input source for a measurement channel.

ch	Measurement channel (0 or 1).
sel	GT_CHA_POS – Measure positive edges on input channel A.
	GT_CHA_NEG – Measure negative edges on input channel A.
	GT_CHB_POS – Measure positive edges on input channel B.
	GT_CHB_NEG – Measure negative edges on input channel B.
	GT_ARM_POS – Measure positive edges on ARM input[1].
	GT_ARM_NEG – Measure negative edges on ARM input[1].

Return value: Returns *GT_False* if setting failed, *GT_True* if setting was successful.

GT_Bool GT668SetMeasSkip(int ch, unsigned int rate)

This function sets the measurements skip rate for a measurement channel. This will cause the measurement channel to drop **rate** out of every **(rate + 1)** time tags. If **rate** value is 0 – no prescaling will be done.

ch	Measurement channel (0 or 1).
rate	Prescaling value (0 to 1999999999).

Return value: Returns *GT_False* if setting failed, *GT_True* if setting was successful.

[1] **Note:** The ARM signal has no prescaler so if its frequency is higher than 4MHz, attempting to measure it will miss some of the edges.

GT_Bool GT668SetMeasTagArm(int ch, GtiTagArmSrc src, GtiPolarity pol)

This function selects the time tag arming source for a measurement channel.

ch	Measurement channel (0 or 1).
src	GT_TA_IMM – Arm immediately when the channel is ready.
	GT_TA_SW – Arm by software call to *GT668TagArmCommand()*.
	GT_TA_EXT – Arm from edge of external ARM signal.
	GT_TA_AUX – Arm from arming signal on AUX bus.
	GT_TA_OTHER – Arm from other channel.
	GT_TA_OFF – Do not arming (same as disabling the channel).
pol	GT_POL_POS – Use positive edge of external Arm.
	GT_POL_NEG – Use negative edge of external Arm.

Return value: Returns *GT_False* if setting failed, *GT_True* if setting was successful.

GT_Bool GT668SetMemoryWrapMode(GT_Bool wrap)

This function enables or disabled the “wrap” mode of the instrument’s memory. If “wrap” mode is enable the instrument will work continuously (as long as the user reads the time tags fast enough). If “wrap” mode is disabled the instrument will fill the memory and stop.
NOTE: USB instrument have no wrap mode. Calling this function will be ignored.

wrap	GT_True – “Wrap” mode is enabled.
	GT_False – “Wrap” mode is disabled.

Return value: Returns *GT_False* if setting failed, *GT_True* if setting was successful.

GT_Bool GT668SetOutput(int out, GtiOutSrc src, GtiPolarity pol, double width, double delay)

This function configures one of the outputs to generate pulses.

Note: The source option GT_OUT_OTHER can be applied only to one output. If the other output is already set to GT_OUT_OTHER it will be changed to GT_OUT_REALTIME.

out	Select output to configure (0 or 1).
src	Source of the output pulse.
	GT_OUT_REALTIME – at real time specified by call to *GT668RealTimeOutput()*.
	GT_OUT_START – after start measurements command.
	GT_OUT_ARM_POS – after positive edge on ARM input.
	GT_OUT_ARM_NEG – after negative edge on ARM input.
	GT_OUT_BA – after block arm.
	GT_OUT_OTHER – after pulse on other input.
pol	Set output pulse polarity.
width	Set output pulse width in seconds (range 10 ns to 20 seconds).
delay	Set output pulse delay for sources other than real time in seconds (range 0 to 20 seconds).

Return value: Returns *GT_False* if setting failed, *GT_True* if setting was successful.

GT_Bool GT668SetRealTime(unsigned int sec, GT_Bool sync)

This function is used to set the real time clock to UTC or any other 32-bit time value in seconds (the time fraction will be set to zero). The **sync** option allows synchronization to a hardware pulse on the ARM input – usually a 1PPS signal from the same source as the seconds value.
Note: This function waits until time is set or until its timeout.

sec	The time value in seconds.
sync	GT_True – synchronize real time clock with next pulse on ARM input.
	GT_False – start real time clock immediately.

Return value: Returns *GT_False* if setting failed, *GT_True* if setting was successful.

GT_Bool GT668SetRealTimeEnd(void)

This function is used to end setting the real time clock (See function *GT668SetRealTimeStart*()).

Return value: Returns *GT_False* if setting failed, *GT_True* if setting was successful.

GT_Bool GT668SetRealTimeIsDone(void)

This function is used to check if the setting the real time clock is finished (See function *GT668SetRealTimeStart*()).

Return value: Returns *GT_False* if setting is not done, *GT_True* if setting is done.

GT_Bool GT668SetRealTimeStart(unsigned int sec, GT_Bool sync)

This function is used to start setting the real time clock to UTC or any other 32-bit time value in seconds (the time fraction will be set to zero). The **sync** option allows synchronization to a hardware pulse on the ARM input – usually a 1PPS signal from the same source as the seconds value. Use the *GT668SetRealTimeIsDone()* function to find when the setting is done, and then call the *GT668SetReadlTimeEnd()* function.

sec	The time value in seconds.
sync	GT_True – synchronize real time clock with next pulse on ARM input.
	GT_False – start real time clock immediately.

Return value: Returns *GT_False* if setting failed, *GT_True* if setting was successful.

GT_Bool GT668SetReferenceClock(GtiRefClkSrc src, GT_Bool ref_5MHz, GT_Bool aux_out)

This function selects the source of the instrument's clock.

src	GT_REF_INTERNAL – Use the internal clock on-board.
	GT_REF_EXTERNAL – Lock to the 10MHz external clock input.
	GT_REF_PXI – Use the clock from the PXI chassis (valid only for GT668PXI boards).
	GT_REF_AUX – Use the clock transmitted on the AUX bus.
ref_5MHz	GT_True – Reference clock is 5MHz (valid only with GT_REF_EXTERNAL option).
	GT_False – Reference clock is 10MHz.
aux_out	GT_True – Output the selected clock to the AUX bus (invalid with GT_REF_AUX option).
	GT_False – Do not output clock to the AUX bus.

Return value: Returns *GT_False* if setting failed, *GT_True* if setting was successful.

GT_Bool GT668SetT0Mode(GT_Bool arm, GT_Bool rel)

This function configure the T0 mode. The **arm** value selects if the reference time (t0) will be generated from the block arming tag (with 20ns resolution) or from the first time tag (full channel resolution). The **rel** value selects if the T0 value will be subtracted from all time tags.

arm	GT_True – t0 generated from block arm.
	GT_False – t0 generated from first time tag.
rel	GT_True – t0 will be subtracted from all time tags.
	GT_False – t0 will not be subtracted from all time tags.

Return value: Returns *GT_False* if setting failed, *GT_True* if setting was successful.

GT_Bool GT668StartMeasurements(void)

This function loads the user specified configuration into the instrument and starts a set of measurements.

Return value: Returns *GT_False* if failed, *GT_True* if successful.

GT_Bool GT668StopMeasurements(void)

This function stops the measurements.

Return value: Returns *GT_False* if failed, *GT_True* if successful.

int GT668TagArmCommand(GT_Bool ch0, GT_Bool ch1)

This function sets the level of the software arm signals for tag arming on both measurement channels.

ch0	GT_True – set arm signal for channel 0 high.
	GT_False – set arm signal for channel 0 low.
ch1	GT_True – set arm signal for channel 1 high.
	GT_False – set arm signal for channel 1 low.

Return value: Returns *GT_False* if setting failed, *GT_True* if setting was successful.

7. LabView® Support

The distributed software contains support for National Instrument's LabView system. The support package was created with LabView 2012, but it also provides libraries that are compatible with LabView versions 8.2 or later (for later versions the LabView program may need to convert them to newer format).

Note: At this time LabView is supported only under Windows platforms.

The support includes:

- GT668.llb – GT668 VI Library - a collection of VI's (Virtual Instruments) to call the GT668 functions. This library is intended for using a single GT668 board.
- GT668M.llb – GT668 VI Library for multiple boards. Same VI's as in GT668.llb but each one of them has one more connector for the board number, and all of them are configured to use the same thread (the LabView user interface thread) to guarantee that the calls are serialized.

The examples include:

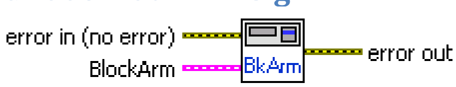
- Example.vi – A simple example of using the LabView GT668.llb library.
- Example_2bd.vi – A simple example of using the LabView GT668M.llb library with two boards.
- LVExr.vi – A full exerciser example of using the GT668.llb LabView library. Use the LVExerciser VI to start it.

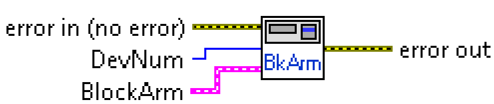
GT668 VI Library

All the VI's have "error in" and "error out" connectors of the standard LabView error cluster type. The configuration parameters are also usually combined into one cluster for each VI. All the VI's have standard "error in" and "error out" clusters.

For each VI in the following list the first image is the way it's defined in GT668.llb and the second image is the way it's defined in GT668M.llb. If only one image shows it means both libraries have the same VI.

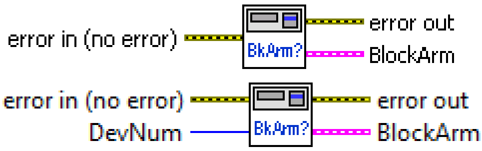
GT668BlockArmCfg





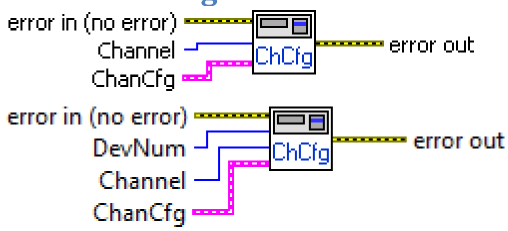
GT668BlockArmCfg VI configures the block arm options. See the GT668SetInputThreshold(), GT668SetBlockArm(), and GT668SetArmAuxOut() functions.

GT668BlockArmGet



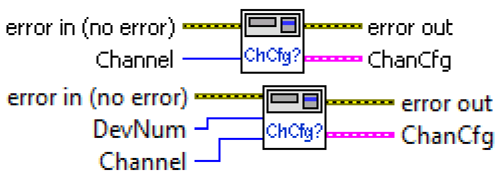
GT668BlockArmGet VI retrieves the configuration of the block arm options. See the GT668GetInputThreshold(), GT668GetBlockArm(), and GT668GetArmAuxOut() functions.

GT668ChanCfg



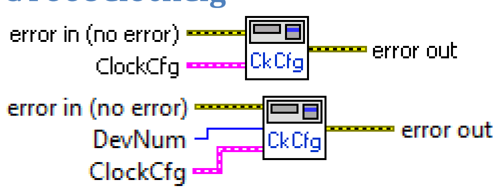
GT668ChanCfg VI configures one measurement channel options. See the GT668SetMeasEnable(), GT668SetMeasSkip(), GT668SetMeasInput() and GT668SetMeasTagArm() functions.

GT668ChanGet



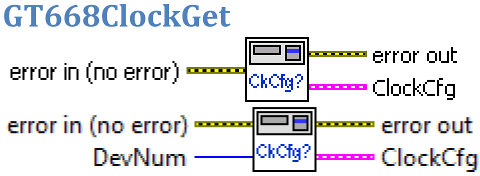
GT668ChanGet VI retrieves the configuration of one measurement channel options. See the GT668GetMeasEnable(), GT668GetMeasSkip(), GT668GetMeasInput() and GT668GetMeasTagArm() functions.

GT668ClockCfg



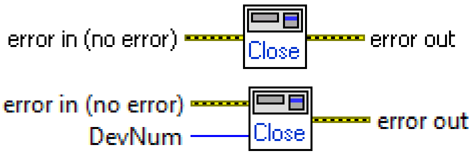
GT668ClockCfg VI configures the reference clock options. See the GT668SetInputThreshold() and GT668SetReferenceClock(), functions.

GT668ClockGet



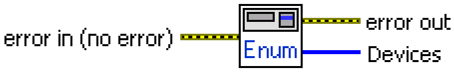
GT668ClockGet VI retrieves the configuration of the reference clock options. See the GT668GetInputThreshold() and GT668GetReferenceClock(), functions.

GT668Close



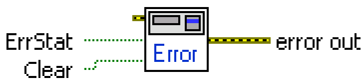
GT668Close VI closes the current device. See the GT668Close() function.

GT668Enumerate



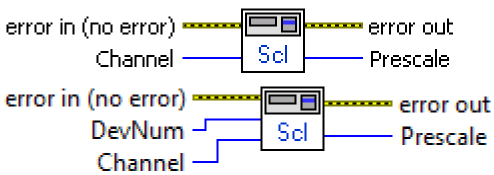
GT668Enumerate VI finds all the GT668 devices in the machine and returns an integer array with the board numbers. See the GT668EnumerateBoards() function.

GT668GetError



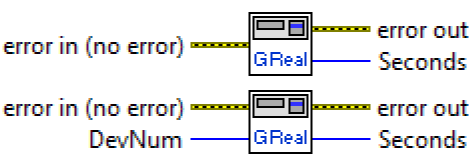
GT668GetError VI checks for error (if the *ErrStat* input is *true*) and generates an error out cluster with the error information. If the *Clear* input is true the error is cleared, otherwise it is retained. See the GT668GetError(), GT668ClearError(), and GT668GetErrorMessage() functions.

GT668GetPrescale



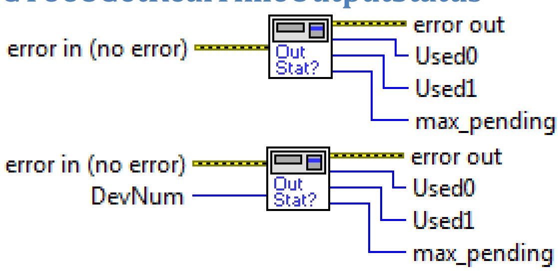
GT668GetPrescale VI retrieves the current total prescaling of the measurement channel specified by the *Channel* input. See the GT668GetTotalPrescale() function.

GT668GetRealTime



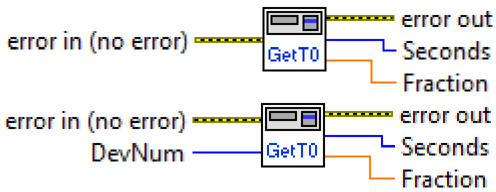
GT668GetRealTime VI retrieves the real time seconds clock from the current board. See the GT668GetRealTime() function.

GT668GetRealTimeOutputStatus



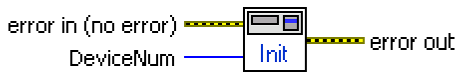
GT668GetRealTimeOutputStatus VI retrieves the number of real time output commands pending for each of the two outputs (issued by GT668RealTimeOutput VI) and the maximum possible number of such pending. See GT668GetRealTimeOutputStatus() and GT668GetRealTimeOutputMaxPend() functions.

GT668GetT0Ex



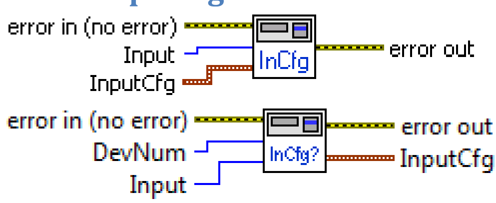
GT668GetT0Ex VI retrieves the T0 of the last measurements set as seconds and fraction. See the GT668GetT0Ex() function.

GT668Initialize



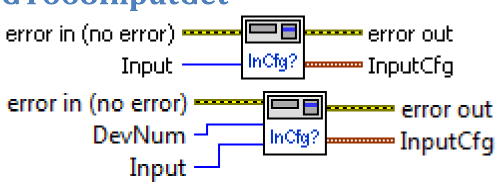
GT668Initialize VI initializes device specified on the DeviceNum input. See the GT668Initialize() function.

GT668InputCfg



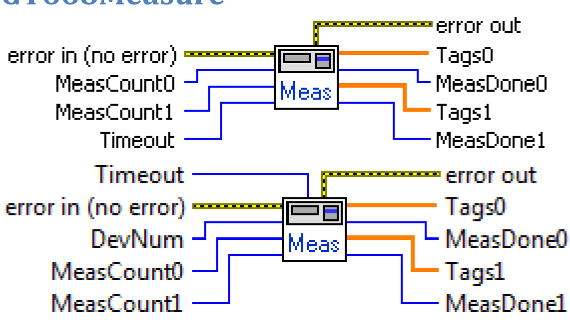
GT668InputCfg VI configures input A or B options. See the GT668SetInputImpedance(), GT668SetInputCoupling(), GT668SetInputThreshold(), and GT668SetInputPrescale() functions.

GT668InputGet



GT668InputGet VI retrieves the configuration of input A or B options. See the GT668GetInputImpedance(), GT668GetInputCoupling(), GT668GetInputThreshold(), and GT668GetInputPrescale() functions.

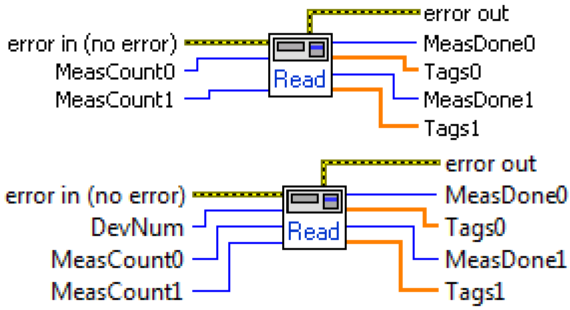
GT668Measure



GT668Measure VI starts a set of measurements on one or both channels (assuming that the inputs and channels were configured), reads the number of timetags requested for each channel (0 means none), and stops when the requested number of timetags for both channels was reached or the time specified by the Timeout input in milliseconds has passed. See the GT668StartMeasurements(), GT668ReadTimetags(), and GT668StopMeasurements() functions.

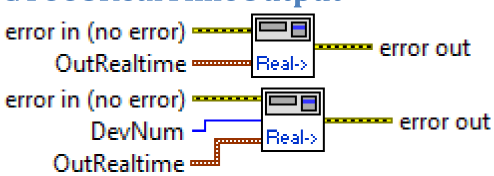
NOTE: Calling this vi will block execution until all measurements are done or until a timeout happens. To implement a non-blocking read use the VIs GT668Start, GT668Read and GT668Stop

GT668Read



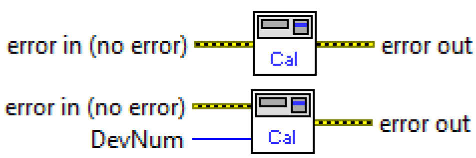
GT668Read VI reads up to the number of timetags requested for each channel. Unlike the GT668Measure VI it does not start or stop the measurements (the GT668Start and GT668Stop Vis need to be used), and does not wait but returns immediately with the available time tags. See the GT668ReadTimetags() function.

GT668RealTimeOutput



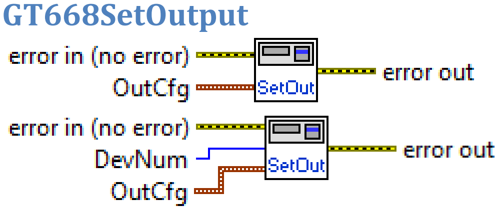
GT668RealTimeoutput VI sends to the board a real time value at which to generate an output pulse. The fraction will be converted to the nearest 10 nsec interval. See the GT668RealTimeOutput() function.

GT668SelfCalibration



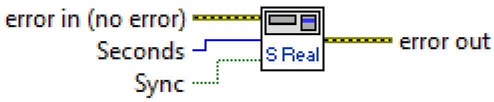
GT668SelfCalibration VI runs the GT668 self-calibration calibrating the interpolators timing measurement circuits. See the GT668SelfCal() function.

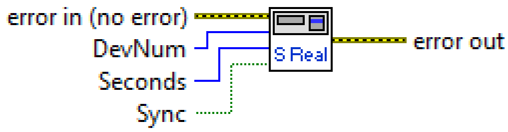
GT668SetOutput



GT668SetOutput VI configures the GT668 outputs. See GT668SetOutput() function.

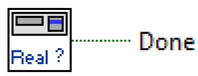
GT668SetRealTime

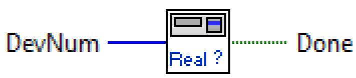




GT668SetRealTime VI sets the real time clock to a real time value ‘Seconds’. If ‘Sync’ is true the real time clock will wait for a pulse on the ARM input (usually from a 1PPS source) and synchronize itself to it. The VI will wait until real time is set or an error is detected.

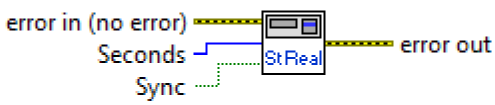
GT668SetRealTimeIsDone

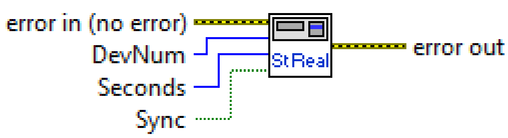




GT668SetRealTimeIsDone VI checks if the setting of real time started with GT668SetRealTimeStart VI is done.

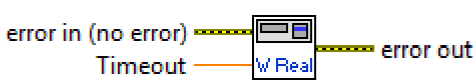
GT668SetRealTimeStart

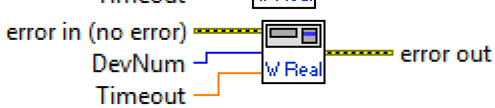




GT668SetRealTimeStart VI starts setting the real time clock to a real time value ‘Seconds’. If ‘Sync’ is true the real time clock will wait for a pulse on the ARM input (usually from a 1PPS source) and synchronize itself to it. The VI does not wait but returns immediately. Use the GT668SetRealTimeWaitDone to wait for it to be done.

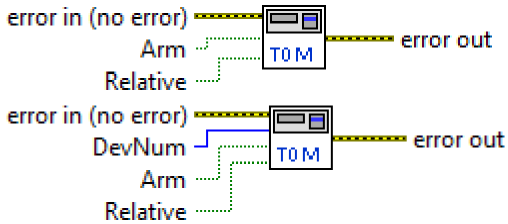
GT668SetRealTimeWaitDone





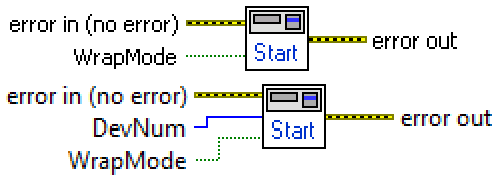
GT668SetRealTimeWaitDone VI wait for the real time setting started with GT668SetRealTimeStart VI to be done. ‘Timeout’ is the time in seconds to wait before aborting with timeout.

GT668SetT0Mode



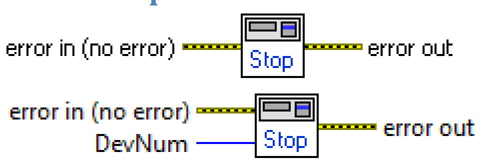
GT668SetT0Mode VI configures the T0 generation at the beginning of a set of measurements. See GT668SetT0Mode() function.

GT668Start



GT668Start VI configures the memory “Wrap” mode and starts a set of measurements. See the GT668SetMemoryWrapMode() and GT668StartMeasurements() functions.

GT668Stop



GT668Stop VI stops a set of measurements. See the GT668StopMeasurements() function.

8. Java Support

Support for Java is included in directory < install_dir >\Support\Java. It includes the file GT668.jar with the support java code, an interface library (JNIGT668.dll for Windows, libjnigt668.so for Linux), and a directory “Documentation” with the documentation of the support in HTML format.

The interface library is also installed into the system in a place that programs will be able to find it (Under Windows in directory “Windows\System32”, and under Linux in directory “/usr/lib” or “/usr/lib64”).

To view the Java Support documentation use your Internet Browser program to open the file “<install_dir>\Support\Java\Documentation\index.html”. Under Windows you can access the documentation by clicking on the menu entry “Java for GT668 Help”.

The Java sample programs come organized as a Maven project under Eclipse, to use them in this way import the project from directory “<install_dir>\Samples\com.guidetech.gt668.samples”, and configure your Eclipse to find the jar file “<install_dir>\Support\Java\gt668.jar”.

9. Python Support

Support for Python is included in directory < install_dir >\Support\Python. It includes the file GT668Driver.py with the support python code, an interface library (PyGT668.dll for Windows, libpygt668.so for Linux), and a directory “Documentation” with the documentation of the support in HTML format.

The interface library is also installed into the system in a place that programs will be able to find it (Under Windows in directory “Windows\System32”, and under Linux in directory “/usr/lib” or “/usr/lib64”).

To view the Python Support documentation use your Internet Browser program to open the file “<install_dir>\Support\Python\Documentation\index.html”. Under Windows you can access the documentation by clicking on the menu entry “Python for GT668 Help”.

The “<install_dir>\Support\Python” directory was added during installation to your PYTHONPATH environment variable so that python can find the GT668Driver.py driver.

Python Support for GT9000 System

The GT9000 System contains Time Distribution module that can supply reference clock, block arming, and optionally a GPS receiver module. Support for using this from Python code is provided by GTSYSDriver.py located in directory < install_dir >\Support\Python with an interface library (PyGTSYS.dll for Windows, libpygtsys.so for Linux).

To view the Python Support documentation for the GT9000 system use your Internet Browser program to open the file “<install_dir>\Support\Python\Documentation\gt9000sys\index.html”. Under Windows you can access the documentation by clicking on the menu entry “Python for GT9000 System Help”.

10. Compatibility with GT658

Overview

The GT668 uses the same general architecture as the GT658. The main differences between the two are:

- Improved resolution and noise floor.
- Using PC memory instead of on board memory to store timetags, which increases significantly the maximum number of timetags and the access speed to them.
- Improved prescaling.
- Support for PXI bus.
- Simplified driver API.

As the functionality is mostly the same, porting a GT658 application to the GT668 should be straight forward.

Compatibility Library

The Windows release of the GT668 contains a library name GT65XPCI.dll, the same name as the driver library of the GT658, which implement all the functions documented in the GT658 manual on the GT668 hardware. This allows running Windows applications developed and compiled for the GT658 on the GT668. This includes all the utilities, sample programs, and LabView library that come with the GT658 except for the ADDRTEST, READSPD, and TESTMEM utilities which are hardware dependent.

For users of master-slave board pairs of the GT658 instrument the release contains also a library named GT65XSUP.dll, the same name as the supplemental library for the GT658 master-slave boards. Only 32-bit applications are supported by this library.

Windows 32-bit

The GT65XPCI.dll and GT65XSUP.dll files are located in the installation directory and the import libraries for linking to it are in the LIB directory.

Windows 64-bit

In Windows 64-bit installation two versions of the library are provided, a 64-bit version in the installation directory and a 32-bit version in the BIN32 subdirectory. Both versions use the same file name GT65XPCI.dll. The import library for the 64-bit library is in directory LIB, and for the 32-bit library in LIB32. The BIN32 directory also contains the GT65XSUP.dll library and the LIB32 directory contains the GT65XSUP.lib import library for it.

11. Specifications

Measurements Memory

- Unlike the GT650 family the GT668 uses memory in the kernel mode driver on the computer. The memory available to each board in the driver is 32MB, enough for 8 million timetags per card.

Software

- Drivers are available for Windows 7, Windows 8, and Linux. (Windows XP and Windows Vista available on request).
- Drivers are callable from C/C++ programs or any program capable of calling a DLL (in Windows) or a Shared Object (in Linux).
- Drivers for National Instruments' LabView®
- A graphic user interface (GUI) exerciser program is available.

Timebase accuracy

- Standard 10 MHz temperature compensated crystal oscillator:
  - Temperature: ±1 ppm, 0°C to 50°C
  - Aging: <1 ppm / Year
  - Short term: $<2 \times 10^{-10}$ (Allan variance @ 1 sec averaging)

External Connections

- Main channels: 2, SMA
- External clock: 1, SMA
- External Arm: 1, SMA

Resolution, Accuracy, and Measurement Rate

- Time Resolution:

Model	-1	-2	-15	-40
Resolution	1 ps	2 ps	15 ps	40 ps

- Noise Floor:

Model	-1	-2	-15	-40
Resolution	3.5 ps	5.5 ps	25 ps	50 ps

- Frequency Resolution – 10-12 digits/second
- Time Interval Accuracy – TBD
- Max. Measurement Rate – 4 million/second per channel.

Main Input Channels

- No. of channels: 2
- Frequency range: DC - 2700 MHz
- Min. pulse width: 150 ps
- Sensitivity: TBD:

- Input impedance: 1 kΩ / 20 pF, or 50Ω, software programmable
- Coupling: AC or DC, software programmable
- Threshold setting (each channel):
    - Range: -5 V to +5 V
    - Resolution: 0.16 mV
    - Absolute accuracy: 1 mV or 0.1% of setting
    - Automatic threshold setting is included

External Clock and External Arm Inputs

- Frequency range:
    - External Clock: 10 MHz ±10 ppm (±100 Hz) or 5 MHz ±10 ppm (±50 Hz). Sine wave or square wave.
    - External Arm: DC - 100 MHz
- Min. pulse width: 4 ns
- Sensitivity: TBD
- Input impedance: 1 kΩ
- Coupling: DC
- Threshold setting:
    - Range: -5 V to +5 V
    - Resolution: 0.16 mV
    - Absolute accuracy: 1 mV or 0.1% of setting
    - Automatic threshold setting is included

Power Requirements

- 3.3V – max. 1.25A
- 5V – max. 4A
- 12V – max. 0.25A

Appendix A: Third Party Licenses

The Java support package contains the XStream library (http://xstream.codehaus.org) under the following license conditions:

XStream