PX14400 Operators Manual

Revision 2.0 – 04/04/2016

Signatec Contact Information	
Toll Free:	1-800-567-4243
Tel:	1-514-633-7447
Fax:	1-514-633-0770
Sales Email:	sales@signatec.com
Support Email:	techsupport@signatec.com
Web:	http://www.signatec.com

Signatec is a product brand of
DynamicSignals LLC, an ISO 9001:2008 Certified Company

DynamicSignals LLC
900 North State Street
Lockport, Illinois 60441-2200
USA

Toll Free: 1-800-DATA-NOW
Tel: 1-815-838-005
Fax: 1-815-838-4424

http://www.dynamicsignals.com

Table of Contents

IMPORTANT NOTICE
ON
HARDWARE COMPATIBILITY

The PX14400 is a PCI Express (PCIe) product and is a PCI Local Bus compliant device that implements a PCIe Gen1 x8 bus connection. As such the PX14400 contains the configuration space register organization as defined by the PCI Local Bus Specification. Among the functions of the configuration registers is the storage of unique identification values for the PX14400 as well as storage of base address size requirements for PX14400 operation.

The host computer that the PX14400 is installed in is responsible for reading and writing to/from the PCI configuration registers to enable proper operation. This functionality is referred to as 'Plug and Play' (PnP). As such, the host computer PnP BIOS must be capable of automatically identifying a PCI compliant device, determining the system resources required by the device, and assigning the necessary resources to the device. Failure of the host computer to execute any of these operations will prohibit the use of the PX14400 in such a system.

It has been determined that systems that implement PnP BIOS, and contain only fully compliant PnP boards and drivers, operate properly. However, systems that do not have a PnP BIOS installed, or contain hardware or software drivers that are not PnP compatible, may not successfully execute PnP initialization. This can render the PX14400 inoperable. It is beyond the ability of Signatec hardware or software to force a non-PnP system to operate a PX14400 device.

1 | Before Using the PX14400

1.1 | Package Contents

The PX14400 package will normally contain (as a minimum) the following:

- PX14400 circuit board assembly
- PX14400 Software Disk
- Two BNC to SMA coaxial cables

1.2 | Unpacking and Handling

The PX14400, and any other electronic circuit board assembly included in its shipment, is shipped in an anti-static bag. These circuit boards are extremely sensitive to static electricity that can damage sensitive electronic parts. The human body can build up a damaging amount of electrical charge, especially in dry weather and in carpeted rooms. To avoid damaging the boards, rid yourself of any charge buildup by touching some large metal object which ideally is at earth potential.

1.3 | Checking for Damage

Carefully inspect the PX14400 for any sign of physical damage on the card. Any such damage must be reported within 15 days from the date of actual shipment in order to be covered by warranty. To report such damage, contact Signatec, explain the nature of the damage, and request a RMA number for returning the merchandise.

1.4 | Warranty

All Signatec manufactured products carry a full 2-year warranty. During the warranty period, DynamicSignals will repair or replace any defective product at no cost to the customer. This warranty does not cover physical damage not reported within 15 days of the time of shipment and does not cover customer misuse or abuse of the product. Contact Signatec for a RMA number before returning any merchandise.

1.5 | System Requirements

The PX14400 requires the following minimum hardware configuration:

- Availability of one open mechanical PCIe x8 or larger PCIe x16 slot for PX14400 card installation
- Intel Core Series or higher CPU
- Minimum of 4 GB system RAM
- Plug-n-Play system BIOS
- 32-bit or 64-bit Operating System: Windows 7, Windows 8, Windows 10, or Linux distribution

1.6 | Physical Layout

The PX14400 is shown in Figure 1. The PX14400 features a total of 5 SMA connectors on the bracket as shown with the identified function for each connector. Note that the PX14400 product bracket will have abbreviated labeling for the identified connections shown.

The actual PX14400 digitizer card onboard circuitry will vary from hardware configuration models: PX14400A, PX14400D, and PX14400D2; however, the identified SMA connectors and their functionality are the same across all PX14400 model configurations.

Figure 1: PX14400 Digitizer (model shown is PX14400D-SP95)

The PX14400 is a 4.3 inches (109.22 mm) full height and 7.5 inches (190.5 mm) length single slot PCIe Gen1 x8 based interface card. All power for the PX14400 digitizer is supplied through its PCIe interface to the host system.

The PX14400 can be installed in a PCIe x8 or x16 Gen1, Gen2, or Gen3 slot but will electrically operate at its maximum PCIe Gen1 x8 interface rate. It is also possible for the PX14400 to be installed in a mechanical PCIe x8 or x16 slot that electrically operates at lower x1 or x4 rates, however doing so will limit the data transfer rate of the PX14400 device to those rates at Gen1 speeds.

1.7 | Software

The PX14400 is supplied with product software for both Windows and Linux operating system environments.

1.7.1 | Windows Software Installation

To install the PX14400 Windows software and documentation just insert the Signatec Product Software CD into your optical drive. Browse to the appropriate 32-bit or 64-bit Windows software folder and proceed with installation by running the “setup.exe” Windows installer file. You must have Administrator level privileges to properly install the software. Note that the 32-bit and 64-bit version releases are functionally equivalent; the only difference is the target architecture on which they may run.

The 32-bit version includes both a 32-bit and 64-bit PX14400 hardware driver and can be installed on either a 32-bit or 64-bit Windows operating system. When the 32-bit version is installed in 64-bit Windows, the 64-bit based PX14400 hardware driver will be installed and will work with installed 32-bit based PX14400 software. Use

the 32-bit version for 32-bit based PX14400 software application development and/or for operating the PX14400 with other 32-bit based software applications.

The 64-bit version installs the 64-bit PX14400 hardware driver and 64-bit based PX14400 software and can only be installed in 64-bit Windows. Use the 64-bit version for 64-bit based PX14400 software application development and/or for operating the PX14400 with other 64-bit based software applications.

Note that on 64-bit Windows, it is possible to install both 32-bit and 64-bit versions of the PX14400 software that can coexist and run side-by-side on the same 64-bit operating system. This is mainly of interest to custom PX14400 software developers that may be developing software intended to run on multiple architectures. For these purposes, the 64-bit software installation includes all necessary libraries required for 32-bit code. They are installed into the "Lib (32-bit)" directory folder where the developer can access and put these 32-bit libraries in a location visible to the linker/loader.

The Windows installation will install all PX14400 software and documentation to the directory folder selected during installation that includes PX14400 drivers, libraries, documentation, components, applications, utilities, and programming examples.

The Windows installation will also install the PX14400 Scope Application that is a full featured virtual oscilloscope application that allows the operator to view or edit all PX14400 digitizer hardware settings as well as record and display acquisition data. This application is a great starting point to getting familiar with the PX14400. Please refer to the separate PX14400 Scope Application Manual for further details on this software application.

1.7.2 | Linux Software Installation

PX14400 Linux software is validated with Red Hat Enterprise Linux and Ubuntu Linux distributions. In general, user mode code can typically be ported to other Linux distributions as well.

The PX14400 software CD contains PX14400 Linux Software with a tar file format of px14400_X.Y.Z.tar.gz. Copy this tar file into a targeted directory of your Linux system and then proceed to extract the contents of this file by running 'tar –xzvf px14400_X.Y.Z.tar.gz' from a console in that directory. Extracting the files contained within this tar file, will create the following directory structure with the indicated contents:

- driver : source for the PX14400 driver
- doc : source for documentation regarding PX14400 development and usage
- examples : source for PX14400 example applications
- libsig_px14400 : source for PX14400 shared software function library
- util : source for example PX14400 utility applications and scripts
- load_scripts : source for utility PX14400 scripts

Note that the PX14400 Linux software does not include a full PX14400 Scope Application utility. The PX14400 Scope Application utility is only available with Windows software.

Please read any Release Notes or README documents that may be installed in the root PX14400 folder.

You must be running the kernel on which your PX14400 software will be running. If you are compiling against a stock kernel (that is, a kernel not built locally on the system), then you need to make sure that correct config file is in place. The kernel configuration file is named .config and is always at the top of the kernel source tree. (Note

that this is a hidden file and will not be displayed in a default directory listing. Use 'ls –a' to see hidden files in listing.)

To verify if you have a .config file, run 'stat /lib/modules/`uname –r`/build/.config' in a terminal. If the file is present you should see some information on the file. If it is not present, then stat will output 'stat: cannot stat…'. In this case you will need to supply a .config file.

To get a .config file:

1.) Open a terminal and go to arch in the kernel source directory:

    Run: 'cd /lib/modules/`uname r`/build/arch'

    Copy the default config file:

    Run: 'cp XXX/defconfig ../.config' where XXX is your particular architecture (e.g. i386 for 32-bit x86 machines, x86_64 for 64-bit Intel or AMD64 machines).

2.) Compile everything by running 'make' from the PX14400 directory. This will compile all driver modules and libraries. Since drivers and header files will be installed, you must be running as root.

3.) Install required files by running 'make install'. This will copy and/or register modules and shared libraries.

4.) Compile example and/or utility applications by running 'make example'. This will compile all example/utility applications. This step needs to be done after Step 3 because the utilities are dependent on the PX14400 shared library being installed.

5.) The PX14400 driver module (sig_px14400.ko) should be automatically loaded when a PX14400 device is detected by the system. A quick way to check if the driver is loaded is to run 'lsmod | grep sig_' in a terminal. If the kernel mode driver is loaded, then you should see output similar to:

    sig_px14400 34412 0 (the actual numbers will vary)

    To explicitly load or unload the PX14400 driver, there are scripts in the load_scripts directory.

ATTENTION RED HAT or FEDORA CORE USERS:

The PX14400 shared library (libsig_px14400) is installed to /usr/local/lib/. Some Red Hat distributions do not have this directory listed in the library loader's search path. This results in the library not being able to be found by the shared library loader. (Client PX14400 software will fail to build due to linker errors.) To fix this (you will need to be root):

1.) Add '/usr/local/lib' to the /etc/ld.so.conf file. (Just add it to the bottom of the file.)

2.) Run ldconfig. This can be done from any terminal and will not require any further input. (It just applies the change made in the previous step.)

3.) The PX14400 library should now be visible to the loader after a 'make install'.

2 | Functional Description

2.1 | Overview

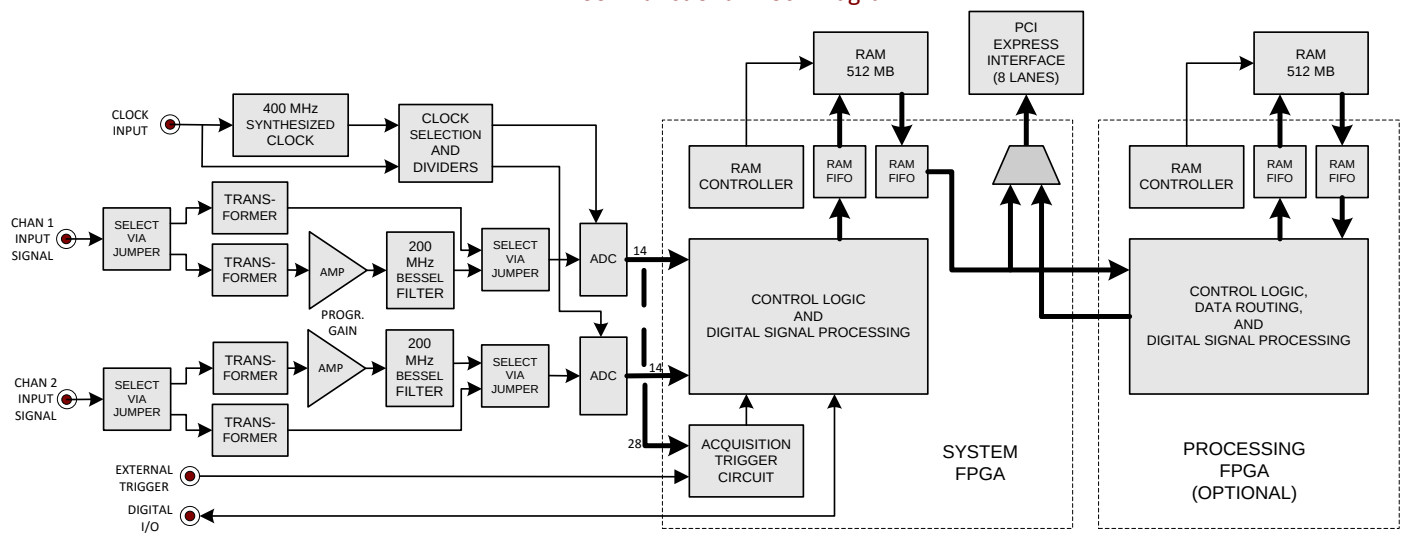
The PX14400 is a dual channel waveform capture board that can acquire up to 400 MS/s on each channel with 14-bit resolution and is provided in three different hardware model configurations: PX14400A, PX14400D, and PX14400D2. The primary differences between these three models revolve around their analog front ends:

Analog Input Front End Specifications	Model PX14400A	Model PX14400D	Model PX14400D2
Coupling:	AC	DC	DC
Impedance:	50Ω	50Ω	50Ω
Full Scale Voltage Ranges:	• Amplifier Hardware Configuration: 25 Software Selectable: 220mV$_{p-p}$ to 3.487V$_{p-p}$ in 1dB Steps • Transformer Hardware Configuration: Single Fixed 1.1V$_{p-p}$	2 Software Selectable: 400mV$_{p-p}$ and 1.2V$_{p-p}$	6 Software Selectable: 200mV$_{p-p}$, 333mV$_{p-p}$, 600mV$_{p-p}$, 1V$_{p-p}$, 1.6V$_{p-p}$, and 3V$_{p-p}$
Bandwidth:	• Amplifier Hardware Configuration: 100 kHz to 200 MHz (Bessel Filter) • Transformer Hardware Configuration: 500 kHz to 400 MHz	DC to 200 MHz (Bessel Filter)	DC to 106 MHz @ 200mV$_{p-p}$ DC to 182 MHz @ 333mV$_{p-p}$ DC to 239 MHz @ 600mV$_{p-p}$ DC to 130 MHz @ 1V$_{p-p}$ DC to 201 MHz @ 1.6V$_{p-p}$ DC to 248 MHz @ 3V$_{p-p}$ (All with Bessel Filter)

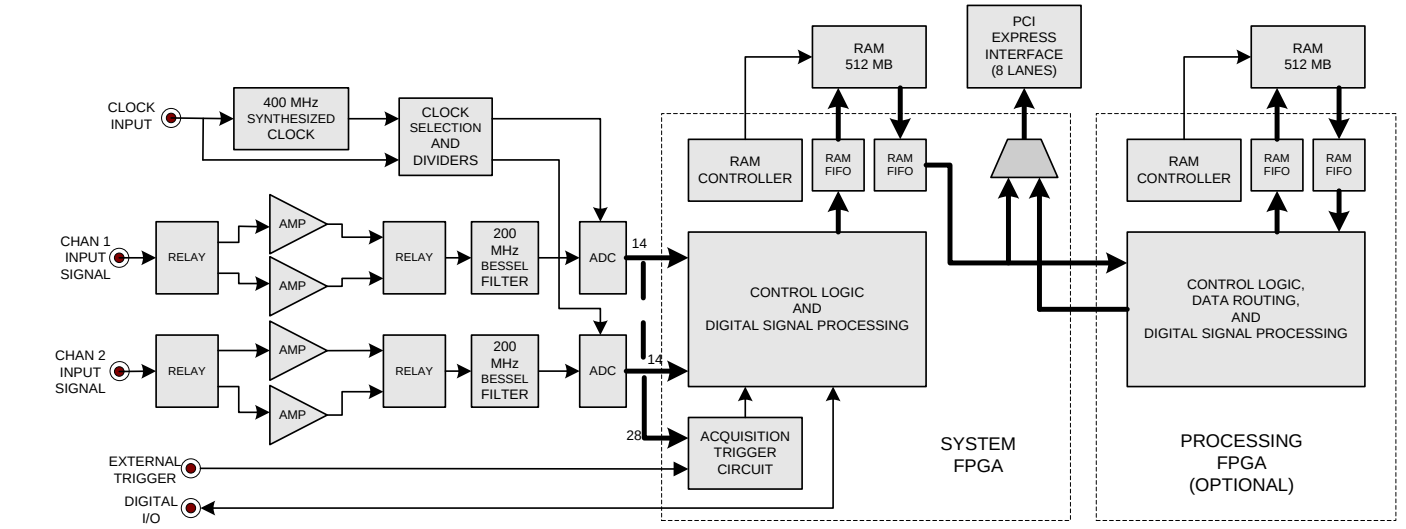
Note that all input voltage range specifications are specified in peak-to-peak voltage (V$_{p-p}$). For example, a full scale input voltage range of 1.1V$_{p-p}$ starts at -0.55mV$_{p-p}$ and extends to +0.55mV$_{p-p}$.

The block diagram figures on the following page depict the simplified functional representations for the PX14400A, PX14400D, and PX14400D2 models and will be referenced many times in the following descriptions of various board functions.

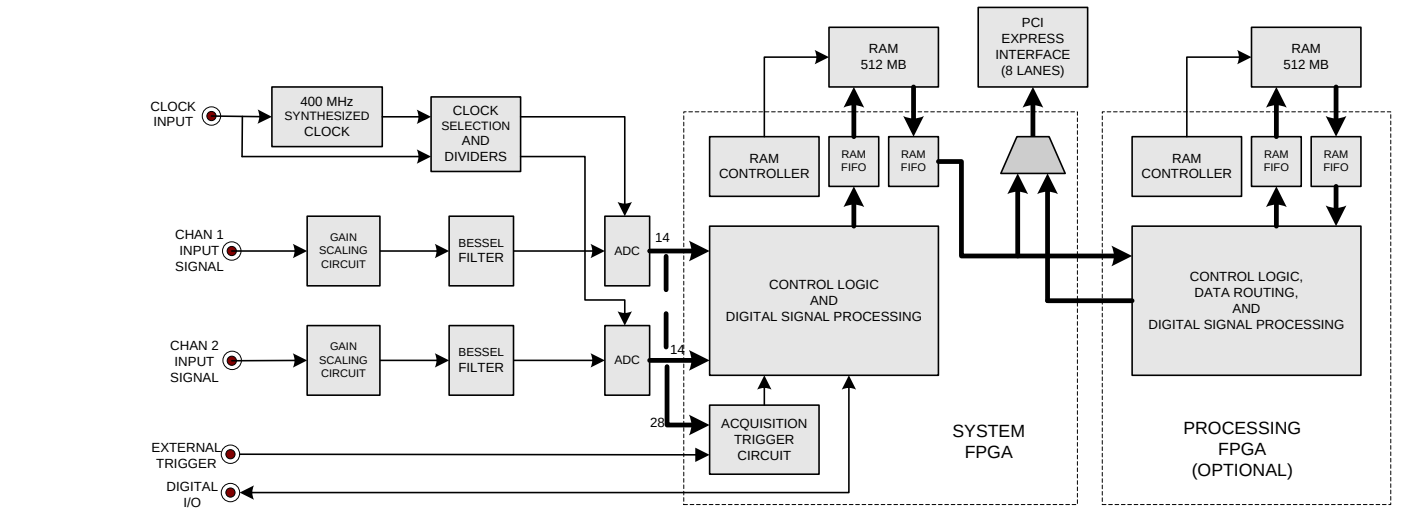
PX14400A Functional Block Diagram



PX14400D Functional Block Diagram



PX14400D2 Functional Block Diagram



2.2 | Analog Input Circuitry

2.2.1 | Active Channels

All PX14400 models feature two active input channels. ADC data can be captured in the following modes:

- Dual channel = Acquires ADC data for Channel 1 and Channel 2 simultaneously.
- Single channel – CH1 = Acquires ADC data for Channel 1 only.
- Single channel – CH2 = Acquires ADC data for Channel 2 only.

The onboard acquisition memory is not dedicated to a particular channel resource, so in single channel modes the entire 512 MB (256 MS) of signal acquisition memory can be used to capture data from channel 1 only or from channel 2 only. In dual channel mode the onboard acquisition memory is shared evenly between the two active channels, 256 MB (128 MS) per channel.

The active channel selection may only be changed while the PX14400 is in the Standby operating mode.

The SetActiveChannelsPX14 function is used to set the number of active channels.

2.2.2 | Front End for PX14400A

The PX14400A model incorporates two identical analog AC-coupled input channels with 50Ω impedance that are factory hardware configured to utilize either a direct transformer front end connection or an amplifier front end connection. The front end hardware configuration is specified at the time of order placement for the PX14400A and can only be modified at the factory. If needed, contact Signatec support to request product RMA service to change the front end hardware configuration of any existing PX14400A units.

The direct transformer front end connection allows for higher bandwidth frequency operation up to 400 MHz as well as slightly better performance in terms of SNR and SFDR, but this higher performance comes at the cost of a single fixed gain input voltage range of $1.1V_{p-p}$. For applications that provide relatively high voltage signals close to the full voltage range of the transformer channel and at a constant amplitude, these would be better served to use the transformer input to gain approximately 3 dB of better SNR performance.

For applications with very low level signals or varying amplitudes, the flexibility of the programmable amplifier is probably the better selection. The amplifier front end connection allows for 25 unique programmable settings of the full-scale input voltage range for each input channel from $220mV_{p-p}$ to $3.487V_{p-p}$ in 1 dB steps. The gain of the amplifier is set via a 5-bit digitally controlled attenuator to achieve the various input voltage ranges. The amplifier drives a 200 MHz 3-pole Bessel filter, which has the characteristic of a flat (constant) time delay response over the frequency range.

To programmatically determine if a PX14400 input channel is configured for direct transformer or an amplifier front end connection, check the return value of GetInputVoltRangeCh*X*PX14. If the return value is PX14VOLTRNG_STATIC_1_100_VPP (0) then the input channel is configured for a direct transformer connection. Any other return value will indicate that the input channel is configured for an amplifier front end connection.

The input voltage range selections can be changed while the PX14400 is in active acquisition modes.

The SetInputVoltRangeCh*X*PX14 function is used to select the input voltage range.

2.2.3 | Front End for PX14400D

The PX14400D model incorporates two identical analog DC-coupled input channels with 50Ω impedance. The PX14400D is identical to the PX14400A with the exception that the analog front end has been changed to accommodate DC-coupled operation and only supports an amplifier front end connection. The figure below shows the analog mechanization, where each channel is identical.

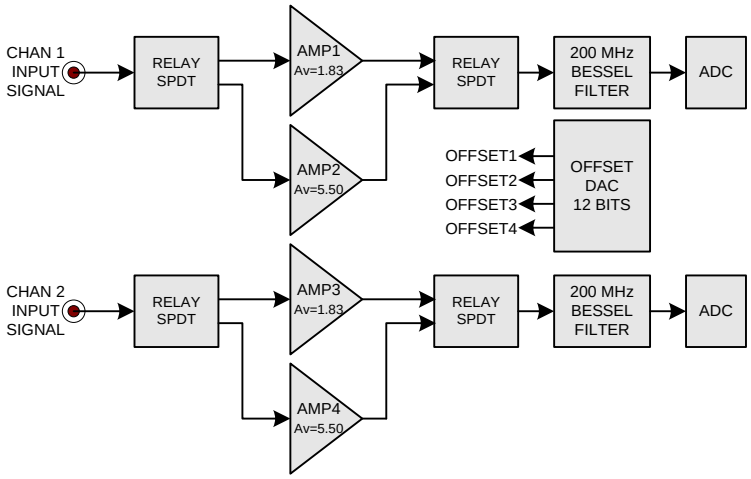


Figure 2: PX14400D Model DC-Coupled Input Channels

Two software selectable input voltage ranges are mechanized on-board. When the low gain amplifiers (Av=1.83) are selected, the full scale input voltage is $1.2V_{p-p}$. When the high gain amplifiers are selected, the full scale input voltage is $400mV_{p-p}$.

Each offset consists of a plus and a minus component. OFFSET1, for instance, consists of OFFSET1+ and OFFSET1-. Each of these components is connected to its own independent DAC and can be set to a value from 0 to 4095. When the + and – components are set to the same value, no net offset is injected into the circuit. The offset circuitry has been designed so that the input range can be set for uni-polar positive, uni-polar negative, or anything in-between.

The source input impedance and coupling affect both the amplitude and the DC offset of the captured signal. Note: If the source impedance is other than 50Ω or if the coupling is AC rather than DC then the signal amplitude and offset capability may be degraded.

The SetInputGainLevelDcPX14 function is used to select the input gain to effectively select the full scale input voltage of $1.2V_{p-p}$ or $400mV_{p-p}$ for each input channel. The input gain selection can be changed while the PX14400 is in active acquisition modes.

The DC offset setting can be changed while the PX14400 is in active acquisition modes.

The SetDcOffsetCh*X*PX14 function is used to set the DC offset for Channel 1 or Channel 2.

2.2.4 | Front End for PX14400D2

The PX14400D2 model incorporates two identical analog DC-coupled input channels with 50Ω impedance. The PX14400D2 is identical to the PX14400D with the exception that the analog front end has been changed to accommodate DC-coupled operation with up to six programmable input voltage ranges through the use of a gain scaling circuit.

On the PX14400D2, after input protection and coupling, the input signal is routed through a gain scaling circuit that provides six voltage ranges with full-scale peak-to-peak levels of $200mV_{p-p}$, $333mV_{p-p}$, $600mV_{p-p}$, $1V_{p-p}$, $1.6V_{p-p}$, and $3V_{p-p}$. An installed Bessel filter has a 3-pole Bessel characteristic to give a flat (constant) time delay response over the frequency range and sets the channel bandwidth for the following ranges:

Input Voltage Range	Input Bandwidth
$200mV_{p-p}$	DC to 106 MHz
$333mV_{p-p}$	DC to 182 MHz
$600mV_{p-p}$	DC to 239 MHz
$1V_{p-p}$	DC to 130 MHz
$1.6V_{p-p}$	DC to 201 MHz
$3V_{p-p}$	DC to 248 MHz

The input voltage range selections can be changed while the PX14400 is in active acquisition modes.

The SetInputVoltRangeCh*X*PX14 functions are used to select the input voltage range for channels 1 and 2.

DC Offset control is implemented via 12-bit DACs which inject an offset voltage into the amplifier inputs to effectively cancel any offsets present in the input signal.

The PX14400D2 adds an additional fine DC offset setting that differs from the normal DC offset in that 1 count of fine DC-offset is equivalent to 1/64 a count of normal DC offset. The fine DC offset is intended for DC offset adjustment when using the low amplitude input voltage ranges on the PX14400D2. For these lower amplitude ranges, the normal DC offset adjustment may be too coarse to provide adequate DC offset adjustment.

The fine DC offset setting can be changed while the PX14400 is in active acquisition modes.

The SetFineDcOffsetCh*X*PX14 functions are used to set the fine DC-offset for channels 1 and 2.

2.3 | ADC Clock

The ADC clock is the clock that is used to drive the onboard analog to digital converters (ADC). The PX14400 can be configured to use one of two ADC clock sources:

- An internal synthesized clock; this is the primary clock source for the PX14400.
- An externally provided clock supplied to the PX1440 external clock input connector.

The ADC clock source may only be changed while the PX14400 is in the Standby operating mode.

The SetAdcClockSourcePX14 function is used to select the ADC clock source to be used for data conversion.

2.3.1 | Internal Clock

An internal synthesized clock is the primary clock source for the ADCs, offering maximum flexibility for sampling rate selection. The synthesizer can generate any clock frequency from 20 MHz to 400 MHz, with the exception of an un-settable frequency range that cannot be reached with the internal clock that is as follows:

- 277.5 MHz to 308.3 MHz (Δ 30.8 MHz)

When the internal synthesized clock is selected, ADC clock jitter is extremely low at about 200 fS RMS. The jitter is independent of the clock divider setting. Clock jitter can reduce the SNR of the captured signal at high frequencies. Due to the low synthesizer clock jitter, degradation does not occur until the input signal frequency is typically greater than 200 MHz.

The internal synthesized clock frequency selection flexibility comes at no cost to the acquisition clock quality/performance when locked to the onboard 10 MHz reference clock that is accurate to better than ±5 PPM. This sets the ADC clock accuracy to also be within ±5 PPM and is used in conjunction with a Phase Lock Loop (PLL) to maintain the targeted internal clock rate, resulting in sampling clock performance that matches or beats most fixed crystal oscillator performance.

An external 10 MHz reference clock source can also be selected for use with the internal synthesized clock, with the external 10 MHz reference clock supplied via the external clock input connector on the PX14400. A supplied external 10 MHz reference clock must have an accuracy of ±50 PPM maximum or better to establish a proper lock.

The internal ADC clock rate and the reference clock source may only be changed while the PX14400 is in the Standby operating mode.

The SetInternalAdcClockRatePX14 function is used to select the desired synthesized clock frequency.

The SetInternalAdcClockReferencePX14 function is used to select the reference clock source for use with the internal clock.

2.3.2 | External Clock

The ADC clock can also be supplied from the external clock input connector on the PX14400. When using the external clock as the ADC clock, the PX14400 firmware needs to be aware of the actual external clock frequency so that it can properly synchronize other onboard Digital Clock Manager (DCM) circuits to it.

When using an external clock, once the PX14400 has entered a data acquisition mode the external clock must be stable for the duration of the data acquisition. If the external clock is removed (or stopped and restarted) while in acquisition mode, the PX14400 firmware will lose synchronization and data errors may occur.

If the external clock input is the ADC clock source, a clock divider may also be applied. An external clock divider #1 can be set to any integer value from 1 (no division) to 20.

The external clock rate and any external clock divider selections may only be changed while the PX14400 is in the Standby operating mode.

The SetExternalClockRatePX14 function is used to specify the external clock frequency that is being applied to the external clock input connector.

The SetExtClockDividersPX14 function is used to specify the external clock divider settings.

2.3.3 | Post ADC Emulated Clock Division

For all clock sources (internal or external) the effective digitization rate can be reduced further via a hardware emulation of clock division that works by discarding samples after conversion. This second post ADC emulated clock division can be set to effectively divide down the clock rate by 1 (no division), 2, 4, 8, 16 and 32.

The post ADC emulated clock division setting may only be changed while the PX14400 is in the Standby operating mode.

The SetPostAdcClockDividerPX14 function is used to specify the post ADC emulated clock division setting.

2.4 | Memory

2.4.1 | Active Memory Region for RAM Acquisitions

The total size of the PX14400 onboard RAM bank for signal acquisition memory is 512 MB, which equates to a maximum sample storage capacity of 256 MS for 14-bit data (using 2 bytes per sample) for RAM acquisition modes. The acquisition memory is not dedicated to a particular channel resource and therefore supports:

- Dual channel mode = Maximum sample capacity of 128 MS per channel.
- Single channel modes = Maximum sample capacity of 256 MS.

The active memory region defines the area of PX14400 onboard RAM that is used for all subsequent acquisitions and transfers for RAM acquisition modes. The active memory region is defined by two parameters:

- Starting Sample = Defines the offset into PX14400 RAM.
- Sample Count = Defines the total number of samples to acquire or transfer.

2.4.1.1 | Starting Sample

The starting sample setting defines the offset into PX14400 RAM at which all subsequent acquisitions and transfers will begin. For acquisition operations, the starting sample defines the 0-based sample index at which new acquisition data will be written. For transfer operations, the starting sample defines the 0-based sample index of the first sample to be transferred.

The starting sample must be an integer multiple of 4,096 samples. The first sample in PX14400 RAM is sample 0. The maximum allowed starting sample is 268,431,360 (this is the maximum sample count for single channel mode minus 4,096 samples).

Dual-channel data is stored in RAM in a channel-interleaved format (CH 1, CH 2, CH 1, CH 2 …). When dealing with dual channel data, the PX14400 RAM address of CH 1 sample *N* would be 2**N* and the RAM address of CH 2 sample *N* would be 2**N* + 1.

When the PX14400 is placed in Standby mode the address counter is automatically reset to zero. In most applications, data is acquired to the PX14400 RAM and then transferred to the PC, always starting at sample 0.

The reset feature removes the necessities of setting the starting sample counter to zero before the data is transferred.

The starting sample setting may only be changed while the PX14400 is in the Standby operating mode.

The SetStartSamplePX14 function is used to set the starting sample.

2.4.1.2 | Sample Count

The sample count setting defines the total number of samples to consider for all subsequent RAM acquisitions and transfers. For acquisition operations, the sample count defines the total number of samples to acquire. For transfer operations, the sample count defines the total number of samples to transfer.

In either case, the sample count is independent of channel count. For a given sample count, the number of samples acquired/transferred per-channel is the sample count divided by the channel count. So for a total sample count of *T*:

- A single channel acquisition/transfer will result in an acquisition/transfer of *T* samples.
- A dual-channel acquisition will result in an acquisition of *T* samples, or *T*/2 samples per channel.

The sample count must be an integer multiple of 32 samples. The maximum allowed sample count is:

- 134,217,728 = for dual channel mode.
- 268,435,456 = for single channel mode.

The sample count setting may only be changed while the PX14400 is in the Standby operating mode.

The SetSampleCountPX14 function is used to set the sample count.

2.4.2 | Memory FIFO Buffer for Streaming Acquisitions

The PX14400 also supports infinite-length free-run streaming acquisitions with PCIe Buffered Acquisition mode. In this mode, the onboard 512 MB RAM bank is operated as a large FIFO for acquiring and transferring data simultaneously by interleaving write and read data packets to the PCIe bus at high-speed sustained data rates.

Free-run streaming acquisitions will acquire an infinite number of samples (or an infinite number of segments if using segmented trigger mode) to the PCIe bus until the operation is ended by putting the PX14400 back into Standby operating mode.

The starting sample and sample count parameters are both set to a value of 0 when conducting free-run streaming acquisitions with PCIe Buffered Acquisition mode.

Streaming acquisitions are typically used to conduct real-time high-speed signal processing and signal recordings of the acquired data on the host PC.

See PCIe Buffered Acquisition Mode for further details.

2.4.3 | Memory for FPGA Signal Processing

All PX14400 FPGA signal processing models, –SP50 and –SP95, include a secondary onboard 512 MB RAM bank, that is utilized to facilitate real-time data transfers between the primary system FPGA and the processing FPGA.

This secondary 512 MB RAM bank cannot be used to extend the maximum sample storage capacity for raw data acquisition in RAM acquisition modes, and is not physically populated on non-processing PX14400 –DR models.

2.5 | Triggering

2.5.1 | Trigger Sources

The trigger source defines where trigger events originate. The PX14400 has two primary trigger sources, internal and external. It is also possible to issue a software generated trigger.

Primary source trigger events are only considered when the PX14400 is in one of the acquisition operating modes. Trigger events that occur prior to entering an acquisition mode are not recognized. An exception to this is the software generated trigger, which may be issued prior to entering an acquisition mode; in which case the acquisition will begin immediately upon entering acquisition mode.

The trigger source selection can be changed while the PX14400 is in active acquisition modes.

The SetTriggerSourcePX14 function is used to select the primary trigger source.

2.5.1.1 | Internal Trigger

The PX14400 can be configured to use one of two internal trigger sources: Internal Channel 1 or Internal Channel 2. Both internal sources operate the same way; the only difference between the two is the data channel that is monitored.

When using an internal trigger source, a trigger event occurs when the selected channel's ADC value crosses one of two programmable trigger levels in a given positive or negative direction. The two trigger levels, A and B, are independent of one another. If only one trigger level is required, the other trigger level can be disabled by setting its value to 0.

A trigger hysteresis setting defines the absolute gap that the signal has to cross to qualify a trigger, regardless if it’s rising or falling edge. This feature sets the trigger sensitivity to avoid triggering on noise.

In addition, a trigger selection setting allows the choice to use the trigger level B or hysteresis for the trigger process (in use with trigger level A) and to use the selected trigger in AND or OR modes.

The trigger level, trigger direction, trigger hysteresis, and trigger selections settings may only be changed while the PX14400 is in the Standby operating mode.

The SetTriggerLevelAPX14 and SetTriggerLevelBPX14 functions are used to configure the A and B trigger levels respectively.

The SetTriggerDirectionAPX14 and SetTriggerDirectionBPX14 functions are used to configure the A and B trigger direction respectively.

The SetTriggerHysteresisPX14 function is used to configure the hysteresis value.

The SetTriggerSelectionPX14 function is used to configure the trigger selection setting.

2.5.1.2 | External Trigger

When the external trigger is selected as the trigger source, a trigger event is defined by a LVCMOS level pulse on the PX14400 external trigger input connector. Triggering may be set to occur on either the positive rising or negative falling going edge of the pulse.

The SetTriggerDirectionExtPX14 function is used to configure the trigger direction setting for the external trigger source.

2.5.1.3 | Software Generated Trigger

The PX14400 requires a trigger signal to start data acquisition. In addition to the internal and external trigger sources, it is also possible to generate an acquisition trigger via software. The software trigger is functional whether internal or external triggering is selected as a current trigger source.

Issuing a software generated trigger is useful for forcing data acquisition to begin in situations where a proper trigger is not available. A software generated trigger may also be issued prior to entering an acquisition mode; in which case the acquisition will begin immediately upon entering acquisition mode.

If a software trigger is issued when segmented trigger mode is being used, the software generated trigger will trigger all subsequent segments.

The IssueSoftwareTriggerPX14 function is used to generate a software trigger.

2.5.2 | Trigger Modes

The trigger mode defines how much data will be acquired for each trigger event. The PX14400 supports two trigger modes: post trigger mode and segmented mode. In either trigger mode, data capture begins on detection of a trigger event.

The trigger mode setting may only be changed while the PX14400 is in the Standby operating mode.

The SetTriggerModePX14 function is used to select the trigger mode.

2.5.2.1 | Post Trigger Mode

Post trigger mode is used for acquiring continuous uninterrupted gap-free data after a single detected trigger event occurs.

In post trigger mode, a single detected trigger event is used to begin acquiring data. The PX14400 will continue acquiring data, once per acquisition clock cycle, for all active input channels until the acquisition is completed.

For RAM acquisition operating modes, acquiring data to onboard PX14400 RAM, acquisition will continue until the specified total number of samples over all active channels has been acquired; at which point the acquisition is considered complete. The specified total number of samples to acquire is defined by the sample count parameter for the active memory region.

For streaming acquisition operating modes, acquiring and transferring data simultaneously to the PCIe bus, the PX14400 can acquire an indefinite amount of continuous data until manually stopped, until a specified total number of samples over all active channels have been reached, or until a specified amount of time duration has been reached.

2.5.2.2 | Segmented Trigger Mode

Segmented mode is used to acquire discrete number of samples of continuous data for every detected trigger event that occurs.

In segmented mode, a detected trigger event is used to begin acquiring a fixed specified number of samples, called a segment for all active input channels. The length of a segment is defined by the segment size setting. After a segment has been acquired, the PX14400 will stop acquisition and then rearm itself and wait for another trigger event to occur. This process will continually repeat until the specified total number of samples to be acquired over all data segments has been reached; at which point the operation is considered complete.

The PX14400 segment rearm time is 150 nanoseconds; any triggers that may occur during the rearm time period will not be detected. Therefore, the timing between received triggers must be greater than 150 nanoseconds in order to ensure that no targeted data to acquire is missed during the rearm time.

For RAM acquisition operating modes, acquiring data to onboard PX14400 RAM, segmented acquisition will continue until the specified total number of samples to be acquired over all data segments for all active channels has been reached; at which point the acquisition is considered complete. The specified total number of samples to acquire is defined by the sample count parameter for the active memory region.

For streaming acquisition operating modes, acquiring and transferring data simultaneously to the PCIe bus, the PX14400 can continuously acquire an indefinite number of segments until manually stopped, until a specified total number of samples to be acquired over all data segments for all active channels have been reached, or until a specified amount of time duration has been reached.

The segment size is independent of channel count; so if *N* samples/segment per channel are desired, the segment size should be set to *N* * *channel count*. The segment size can be any integer multiple of 4 samples and the maximum allowed segment size is:

- 134,217,728 = for dual channel mode.
- 268,435,456 = for single channel mode.

The segment size setting may only be changed while the PX14400 is in the Standby operating mode.

The SetSegmentSizePX14 function is used to specify the segment size setting

2.5.3 | Trigger Timing Options

In addition to the trigger modes described above, two trigger options exist that can affect which samples are acquired relative to the trigger event: pre-trigger sample count and trigger delay sample count. These trigger options allow for starting the data capture slightly before or after the actual trigger event.

The pre-trigger sample count and trigger delay sample count settings are mutually exclusive. Using both at the same time will result in undefined behavior.

2.5.3.1 | Pre-trigger Samples

The pre-trigger samples setting allow the digitizer to keep a specified number of samples that occurred prior to the trigger event. This feature allows the operator to look at data that had occurred before the actual trigger event.

With this trigger option, data from the ADC is routed to a pre-trigger samples buffer of programmable length. The digitization and shifting of data through the pre-trigger samples buffer is always active (while in the acquisition mode) so that the programmable length pre-trigger samples buffer is full of pre-trigger sample data when a trigger occurs. Once the pre-trigger samples buffer is full, data will continuously be removed in a first in, first out (FIFO) fashion until a trigger is received. When the trigger occurs, the output from the pre-trigger samples buffer is written to RAM or the PCIe bus (depending on the current acquisition mode) before the data samples collected after the trigger event.

The data written to the RAM (or the PCIe bus) consists of the defined number of pre-trigger samples immediately preceding the trigger plus the balance of samples required to reach either the set segment size (if in segmented mode) or the total samples size. This mode is useful for seeing backward in time, i.e. seeing the waveform as it existed before the trigger signal occurred.

The SetPreTriggerSamplesPX14 function is used to specify the amount of pre-trigger samples.

2.5.3.2 | Trigger Delay Samples

The trigger delay samples setting allow the digitizer to ignore a specified number of samples after a trigger event. This moves the collected data forward in time after the actual trigger event. Trigger delay samples can be used in both post trigger and segmented triggering modes.

The SetTriggerDelaySamplesPX14 function is used to specify the amount of trigger delay samples.

2.5.4 | Minimizing Jitter - Synchronized Triggering

Under normal triggering from an asynchronous external source there will be a peak-to-peak jitter of 1 clock cycle between the trigger signal and the captured waveform. The use of a synchronized trigger can eliminate this jitter. There are three methods that can be used.

External Clock and Trigger

Supplying an external clock to the PX14400 as well as an external trigger that is synchronized to that clock will eliminate all trigger jitter. This is the most commonly used method.

Synchronizing the Trigger via Clock Out

Via the digital output connector, an output clock may be selected for use by external circuitry to synchronize an input trigger to the PX14400's internal digitizer clock. The clock output frequency will be equal to the digitizer clock frequency.

The SetDigitalIoEnablePX14 function is used to enable the digital IO connector.

The SetDigitalIoModePX14 function is used to set the digital IO mode for clock out operation.

Synchronizing the Trigger via Trigger Out

A synchronized output trigger can be obtained via the digital output connector. For this type of triggering an asynchronous trigger is applied to the trigger input and an output trigger, that is synchronous to the digitizer clock, is used externally to drive a pulse source. This type of operation is useful for many types of pulsed systems.

The SetDigitalIoEnablePX14 function is used to enable the digital IO connector.

The SetDigitalIoModePX14 function is used to set the digital IO mode for synchronized trigger operation.

2.6 | Digital IO

The PX14400 implements a digital IO interface via the digital IO SMA connector on the card bracket that can supply a selection of digital signals that may be used to synchronize external events or to monitor internal processes. The digital IO connector must be enabled before any digital input/output can be received/sent.

The following defined digital IO modes can be selected:

Mode Type	Mode Setting	Mode Description
Output	0V	Outputs a 0V signal.
Output	Synchronized Trigger	Outputs the trigger signal, synchronized to the acquisition clock.
Output	ADC Clock Divided by 2	Outputs the acquisition clock divided by two.
Output	3.3V	Outputs a 3.3V signal.
Input	Timestamp Request	When used with timestamp mode, an input pulse will result in the PX14400 generating a timestamp.

CAUTION: If the digital IO port is configured as an output, the operator must ensure that no other external device is also driving the digital IO connector. Multiple devices driving the common bus can damage the PX14400. When changing modes it is highly recommended to first disable the digital IO port and then select the desired digital IO mode before then re-enabling the digital IO port.

The SetDigitalIoEnablePX14 function is used to enable the digital IO connector.

The SetDigitalIoModePX14 function is used to set the digital IO mode setting for operation.

2.7 | Timestamps

The PX14400 can be configured to generate timestamps for certain events. A timestamp is a 64-bit unsigned value that represents a number of clock ticks. Upon transition from Standby operating mode to any acquisition operating mode, the timestamp counter will reset to 0 and increment once per acquisition clock cycle.

As timestamps are generated, they are inserted into the Timestamp FIFO. Software can read timestamps from this FIFO as the acquisition progresses or after the acquisition completes. The timestamp mode determines how the PX14400 generates timestamps:

Mode	Mode Description
Timestamp per Segment	Generates a timestamp at the start of each segment acquired while in segmented trigger mode.
Timestamp per External Trigger	Generates a timestamp for each pulse received on the external trigger input.
Timestamp per Digital IO Pulse	Generates a timestamp for each pulse received on the digital I/O connector. This mode is useful if timestamp pulse is independent of the trigger signal, and can be used with GPS cards to correlate absolute time and position to the acquired sample index with the timestamp file.

The SetTimestampModePX14 function is used to set the timestamp mode.

See the Timestamp Management section for more details.

2.8 | Operating Modes

The state of a PX14400 device is primarily driven by the current operating mode. The PX14400 has several operating modes, which can be categorized into one of three types: idle, acquisition, and transfer. Further, an active operating mode refers to any of the acquisition or transfer modes.

The PX14400 software wraps each of the active operating modes with high-level functions that manage the operating mode as well as the active memory region, which is closely related to the operating mode.

The SetOperatingModePX4 function is used to change the operating mode.

Each operating mode is detailed in the following subsections.

2.8.1 | Standby Mode

Standby mode is the primary idle operating mode. This is the operating mode that should be used to configure the PX14400 hardware settings prior to a data acquisition. This includes things like setting up the acquisition clock and trigger parameters.

Operating mode transitions should always be coming from or going to Standby mode. Direct transitions between any two active operating modes are not allowed.

Putting the PX14400 into Standby mode while a transfer or acquisition is armed or in progress will result in the transfer/acquisition being cancelled.

2.8.2 | RAM Acquisition Mode

RAM acquisition mode is one of the two main data acquisition operating modes. In this operating mode, data is acquired directly to the PX14400 RAM. This mode can support the maximum acquisition rate for both input channels.

Entering this operating mode will arm the card for acquisition. When a trigger event is detected, the card will start collecting data and writing to the PX14400 RAM area identified by the active memory region. When all samples have been acquired and written into RAM, the PX14400 will interrupt the system with a Samples

Complete interrupt. When the PX14400 driver receives this interrupt, it will put the PX14400 back into the Standby operating mode and notify user-mode software that the acquisition has completed.

Note that while the board is in RAM acquisition mode, data may not be transferred from the card. With this type of acquisition, all data is acquired and written into RAM and only after the acquisition has completed may it be transferred to the host system via the PCIe Transfer operating mode.

2.8.3 | PCIe Transfer Mode

This is the operating mode used for transferring data from the PX14400 RAM to the host system. This operating mode will transfer exiting data that is residing in PX14400 RAM; no new data is acquired. This is the primary method of obtaining data previously acquired with the RAM Acquisition mode.

Users should never directly enter this operating mode. The PX14400 software implements data transfer routines that should be used.

When this operating mode is entered the PX14400 will begin transferring the data defined by the Active Memory Region to the PCIe FIFO. Data is transferred from the PCIe FIFO to the host system via a DMA transfer. The underlying PX14400 software will setup and manage all aspects of the DMA transfer.

See the Data Transfer Routines section for more details on PX14400 data transfers.

2.8.4 | PCIe Buffered Acquisition Mode (Streaming)

This is the other primary data acquisition operating mode and is also referred to as streaming mode. In this operating mode, data is buffered through PX14400 RAM and acquired directly to the PCIe bus. Unlike RAM Acquisition, this mode supports the transfer of data as it is being actively acquired. This is the primary method used to conduct high-speed sustained data streaming to the host PC system for optional real-time signal processing and/or real-time signal recording of the acquired data.

Free-run streaming acquisitions will acquire an infinite number of samples (or an infinite number of segments if using segmented trigger mode) to the PCIe bus until the operation is ended by putting the PX14400 back into Standby operating mode.

In PCIe Buffered Acquisition mode, the onboard 512 MB RAM bank is operated as a FIFO for acquiring and transferring data simultaneously. Acquired data may be put into the onboard FIFO RAM at a maximum data streaming rate of 1.6 GB/s (2 Channels * 400 MS/s per Channel * 2 Bytes per 14-bit Sample = 1.6 GB/s) while also being extracted at this same rate by interleaving write and read data packets.

However the PX14400 card's PCIe interface is limited to maximum streaming data rate of 1.4 GB/s; this is the total maximum PCIe rate of the card that is shared by the active channels. Therefore, without any onboard optional FPGA processing data reduction methods, the maximum possible sampling rates for conducting raw continuous gap-free data streaming operations via the PCIe interface for a single PX14400 card is as follows:

- Single Channel Operation = 400 MS/s Maximum Sampling Rate
  (1 CH * 400 MS/s * 2 Bytes per 14-bit Sample = 800 MB/s Data Streaming Rate)

- Dual Channel Operation = 350 MS/s Maximum Sampling Rate
  (2 CH * 350 MS/s per Channel * 2 Bytes per 14-bit Sample = 1.4 GB/s Data Streaming Rate)

When conducting triggered or pulsed based data capture using segmented mode (to acquire discrete specified number of samples for every detected trigger event that occurs), it is possible to sample at the maximum 400 MS/s for dual channel operation as long as the applied trigger rate is slow enough such that the calculated total data transfer rate to be sustained through the PCIe interface is 1.4 GB/s or less.

Use the following formula to calculate the data rate for the triggered or pulsed based capture:

(*Pulse Repetition Frequency or Trigger Frequency*)
* (*Number of Samples per Segment*)
* (*Number of Channels*)
* (*Number of Bytes per Sample for A/D Resolution*)
= Data Rate

For example:

(30 kHz PRF)
* (5,000 Samples per Segment)
* (2 Channels)
* (2 Bytes per Sample for 14-bit Resolution Data)
= 600 MB/s Data Rate

For either type of acquisition method used, the acquired streaming data is transferred efficiently through the PX14400 PCIe interface to the host PC system RAM using DMA transfer mechanisms. On the host PC side, when conducting real-time signal recording, multiple buffers in system RAM are used as circular buffers. While a write to storage is taking place after a DMA transfer completes, another DMA transfer is happening to a separate buffer.

It is the combined use of the PCIe Buffered Acquisition mode, with PX14400 onboard RAM operating as a FIFO buffer, and efficient high-speed DMA transfers to multiple circular host system RAM buffers that make it possible to achieve high performance real-time sustained data streaming operations within a non-real-time operating system such as Windows.

The PX14400 will only DMA transfer data to the system if the PCIe interface tells it that it can receive data, so data overflow will NOT occur at the PCIe interface. If the host PCIe interface can't receive data at a particular time, due to some host system activity, new acquisition data is accumulated in the onboard RAM of the digitizer card that is operating as a FIFO buffer.

In the event that the onboard PX14400 FIFO RAM completely fills up, an acquisition FIFO overflow occurs that is caught and flagged as an error. The underlying PX14400 software monitors a FIFO Full flag during acquisitions using this mode and will return an error in the event of a FIFO overflow error. A FIFO overflow error condition is usually a result of either:

- Inappropriate application settings applied that effectively produce a total PCIe data streaming rate that exceeds the PX14400 maximum PCIe sustained data streaming rate of 1.4 GB/s.

- The performance capabilities of the targeted receiving host PC system, storage system, or custom application processing the data is not adequate enough (fast enough) to effectively support the targeted total PCIe data streaming rate from the PX14400.

The PX14400 software library implements a higher-level interface that fully encapsulates all aspects of a data recording with this operating mode, including the saving of acquisition data to persistent storage with multiple output formats.

See the Recording Session Management section for more details.

2.8.5 | SP PCIe Buffered Acquisition Mode (Streaming with FPGA Processing)

This operating mode is the same as the PCIe Buffered Acquisition mode, with the exception that the acquired data is first routed to the optional onboard processing FPGA for targeted signal processing routines to be conducted on the data. The resulting FPGA processed output data is then streamed to the PCIe bus.

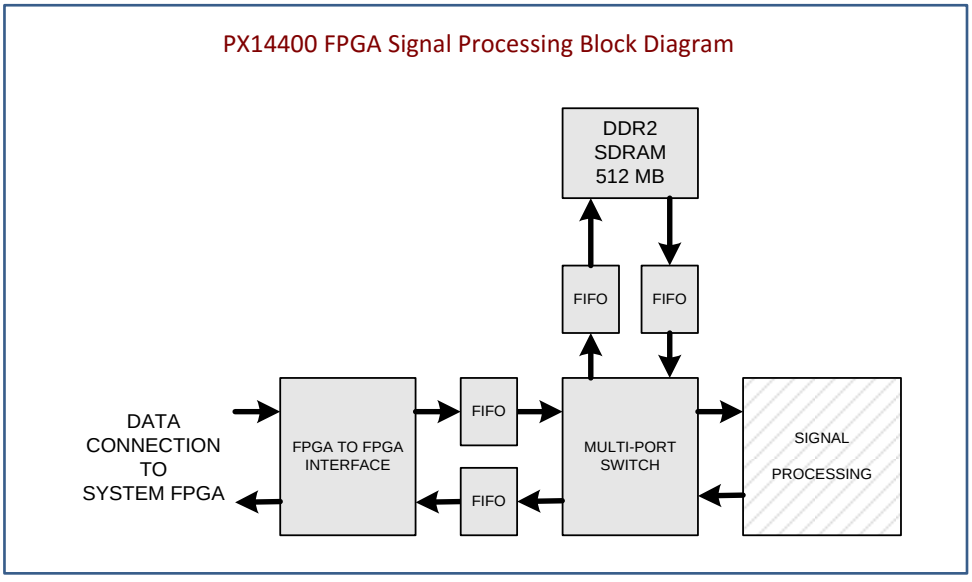
This operating mode is only applicable to PX14400 FPGA signal processing models (–SP50 and –SP95) and cannot be used with non-FPGA processing models (-DR) that do not have an onboard processing FPGA.

Acquired data can be transferred to the processing FPGA at the full maximum data streaming rate of 1.6 GB/s (2 Channels * 400 MS/s per Channel * 2 Bytes per 14-bit Sample = 1.6 GB/s), but the targeted FPGA processing routine must result in output data with an effective reduced data transfer rate of 1.4 GB/s or less in order to be sustained through the PX14400 PCIe interface.

2.9 | FPGA Signal Processing Options

PX14400 FPGA signal processing models feature an onboard secondary Xilinx Virtex-5 SX50T (–SP50) or Xilinx Virtex-5 SX95T (–SP95) FPGA with its own 512 MB RAM bank for dedicated embedded signal processing. The following block diagram shows the data flow within the optionally available signal processing FPGA:

PX14400 FPGA Signal Processing Block Diagram



When FPGA processing is enabled, acquired data is transferred from the System FPGA to the Signal Processing FPGA where the targeted signal processing routine is conducted. The resulting processed data output is then transferred back to the System FPGA, which can then stream the resulting data to the host PC system via the PCIe interface with SP PCIe Buffered Acquisition mode.

PX14400 FPGA signal processing models are supplied with the following signal processing routines:

FPGA SP Routine	Description
Programmable Decimation and Down Conversion (DDC)	Provides decimation processing for single channel mode (CH 1) only with decimation factors of 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, and 8192 for a specified NCO frequency from 0 to 25 MHz.
Fast Fourier Transform (FFT)	Performs programmable zero padding of the data to 1k or 2k samples, Blackman Harris and Rectangular windowing, 1k or 2k FFTs, optional magnitude square calculation in dB, interleaving of FFT block of data with the raw block of data, and also allows for a programmable window function for single channel mode (CH 1) only.
Finite Impulse Response (FIR) Filtering	Provides programmable FIR filtering for one or two channels with the ability to load filter coefficients from a file source for each channel.

The Programmable Decimation and Down Conversion (DDC) routine is the default loaded and enabled processing feature supplied on all PX14400 FPGA processing models.

There are no operational differences of these supplied signal processing routines between the PX14400 FPGA processing models –SP50 and –SP95, these routines will operate identically on either model version.

Only one FPGA signal processing routine feature can be loaded and enabled for use at a time on the PX14400. The desired targeted FPGA signal processing routine can be loaded onto the PX14400 with the PX14400 Scope Application software, which also detects the current loaded processing feature and displays the related interface for making settings that apply to the routine.

Logic firmware file packages for the FPGA signal processing routines are installed with the PX14400 software to the following default location:

C:\Program Files or Program Files (x86)\Signatec\PX14400\Firmware\

From within the Signatec PX14400 Scope Application software, the desired targeted processing firmware logic feature package can be loaded by selecting the "Hardware" menu, select the sub-menu item "Upload Firmware..." and proceed to browse and select the desired firmware logic file feature to be applied. Please note that the firmware update process will take several minutes to complete and should not be interrupted. When completed, it is necessary to shut down the PC system so that it is completely OFF and then re-power the PC system back ON for the firmware update changes to take effect.

CAUTION: Care must be taken to select the appropriate firmware file that matches the correct hardware model version of the PX14400 card. Selecting an improper firmware file may corrupt the PX14400, requiring the PX14400 to be returned to factory for re-initialization and re-programming.

See the separate PX14400 Fixed Logic Kit (FLK) User Guide for further details on these provided FPGA signal processing routines and for their dedicated library functions for custom software application development.

The SetBoardProcessingEnablePX14 function is used to enable or disable FPGA processing.

2.9.1 | Custom FPGA Signal Processing Development

The processing FPGA is fully end user programmable, allowing for customized embedded processing routines to be developed and utilized with the PX14400.

Signatec provides custom FPGA design services to meet specific application requirements for volume OEM customers. Contact Signatec to discuss specific project requirements, feasibility, and scope for customized solutions. Alternatively, the optional PX14400 FPGA Development Kit is available for customers who want to develop their own embedded processing routines.

The Signatec PX14400 FPGA Development Kit requires the end user to have the Xilinx ISE Design Suite software sold directly by Xilinx and provides the native VHDL source code projects of the existing PX14400 FPGA processing routines for DDC, FFT, and FIR Filtering. These VHDL source code projects serve to demonstrate how to write real-time embedded signal processing routines for the onboard Xilinx FPGA device with its defined interfaces for utilizing the various FIFO, RAM, processing elements, and bus interface resources.

There are two PX14400 FPGA signal processing models that provide the following raw FPGA resources:

	–SP50	–SP95
FPGA Device	Xilinx Virtex-5 SX50T	Xilinx Virtex-5 SX95T
# of Logic Cells	52,224	94,208
kbits Block RAM	4,752	8,784
# of DSP Slices	288	640

There are two data interfaces the custom programmable logic will have to manage: the acquisition data interface and the user register interface.

The acquisition data interface provides data from the PX14400 acquisition circuits to be processed by the user logic in the programmable FPGA. The register interface provides a way for user defined custom parameters to be dynamically set by the host PC system application.

The PX14400 Scope Application software includes a generic FPGA Processing interface for enabling FPGA processing and to read/write to specific registers for working with the custom user logic in the programmable FPGA. For custom application software development, standard library functions are provided for interfacing with the FPGA registers.

Developed custom user logic firmware is packaged and uploaded to the PX14400 through its PCIe host interface utilizing the PX14400 Scope Application software. Alternatively, the custom user logic firmware can be directly loaded through the PX14400 JTAG header connector utilizing a Xilinx JTAG programmer sold directly by Xilinx.

See the separate PX14400 FPGA Programming User Manual for further details on custom FPGA programming.

The SetBoardProcessingParamPX14 function is used to specify FPGA processing parameter data.

2.10 | Synchronized Operation of Multiple PX14400 Digitizers

Up to five PX14400 digitizers can be setup for synchronous acquisition operations for a total of 10 input channels by utilizing the separate Signatec SYNC1500 clock/trigger driver source card.

For this operation, an external trigger is supplied to the SYNC1500 card. The SYNC1500 provides a common clock to each PX14400 digitizer's external clock input along with a synchronized trigger to each PX14400 digitizer's external trigger input. In this way, the data acquisition will be synchronized across the active input channels for the multiple connected digitizers.

It is also possible for the SYNC1500 and PX14400 digitizers to operate together while physically installed in separate systems; assuming that all required cable connections can be made between devices and all required software can be run from each system.

See the separate SYNC1500 Operators Manual and SYNC1500 Quick Start Guide for further details on the SYNC1500 and configuration for use with PX14400 digitizers.

2.11 | PX14400 Acquisition Data Format

2.11.1 | Sample Format

The resolution of the onboard PX14400 analog-to-digital converter is 14-bits. To keep the acquisition data nicely aligned and easy to work with; the 2 least significant bits will always be zero, which keeps all data aligned to 16-bits (where the MSB is at bit 15 or the 16th bit).

PX14400 data samples are little-endian (the native binary format for the x86 platform), unsigned 16-bit integral values.

The PX14400 software library uses the px14_sample_t data type to represent a PX14400 data sample. This data type is an alias that will always equate to an unsigned 16-bit type (usually 'unsigned short' in C/C++ parlance) for the current platform.

Since samples are unsigned, this means that sample values can go from 0 to 65532 (or FFFC in hexadecimal):

- A sample value of 0 is equivalent to the minimum input voltage.
- A sample value of 32768 (decimal) is equivalent to approximately 0V.
- A sample value of 65532 (decimal) is equivalent to the maximum input voltage range.

A general formula for converting a sample value to its corresponding input voltage is:

$$V_s = -R/2 + ((S / 65532) * R)$$

Where:

- V_s is the voltage for a particular sample
- R is the input volt range selection in peak-to-peak voltage
- S is the sample value.

So for a sample value of 8192 @ 1V input voltage range:

$$V_s = -1/2 + ((8192 / 65535) * 1) = -0.375V \text{ or } -375mV$$

2.11.2 | Multichannel Data Format

When acquiring dual-channel data, the PX14400 stores and transfers data in a channel-interleaved format. This means that each channel is alternated in the resultant data:

CH 1, CH 2, CH 1, CH 2, CH 1, CH 2, …

The PX14400 software library implements routines to interleave (InterleaveDataPX14) or de-interleave (DeInterleaveDataPX14) dual-channel data.

Be advised that de-interleaving channel data 'on-the-fly' while conducting real-time streaming will place additional strain on the host CPU(s) and might not be sustainable for high-speed rate recording applications. For these circumstances, de-interleaving should be performed as a post process operation after the recording is completed.

2.11.3 | PX14400 Sample Data Files (*.rd16)

Signatec software, such as the PX14400 Scope Application, as well as certain PX14400 library routines have the ability to save PX14400 acquisition data to a file. The native file format used by these entities for saved PX14400 acquisition data is the RD16 file format.

The RD16 moniker is derived from "Raw Data 16-bit". RD16 files are identified by the '.rd16' file extension. Note for the Windows OS, you may need to change the view folder option settings to de-select the default "Hide extensions for known file types" in order to see the .rd16 file extension for Signatec PX14400 data files.

RD16 files are binary files that contain only raw acquisition sample data. There is no file header or additional information in the file. The first two bytes of the file are the first data sample. Samples in .rd16 files are 16-bits in size; however, for recorded 14-bit data, only the lower 14-bits are relevant (the upper two bits will always be zero) as is the case with the PX14400 digitizer.

This simple file format has two big advantages. First, it is very fast to write these files since data is written to the file exactly as it is received from the PX14400. The second advantage is that this file format is very generic which makes it easy for other software to utilize the data. This includes custom software, or other software application environments such as MATLAB.

As RD16 files contain only raw data, they can easily be imported into MATLAB. Note that the Signatec data samples are unsigned values. When reading the binary data into MATLAB, use "uint16" to import the data as unsigned 16-bit format. To center the data around 0 after the data is read in, subtract 32768 from each sample value.

2.11.4 | Signatec Recorded Data Context Files (*.srdc)

The main disadvantage of the RD16 file format is that no context information is stored in the file and the details of the data may not be apparent. Important details such as channel count, voltage range selection and sampling rate are unknown by looking at the raw data alone.

Therefore, Signatec software is also configured to generate a small SRDC auxiliary context file that can contain information such as channel count, input voltage range, sampling rate, source board, operator notes, or any other user-defined data.

The SRDC moniker is derived from "Signatec Recorded Data Context". SRDC files are identified by the '.srdc' file extension. Note for the Windows OS, you may need to change the view folder option settings to de-select the default "Hide extensions for known file types" in order to see the .srdc file extension for Signatec PX14400 SRDC files.

SRDC files are written in XML format and are easily read by any XML-aware software. By convention, the file name of the SRDC file is the full pathname of the associated RD16 data file, appended with a '.srdc' extension. For example, if acquisition data was being written to the file:

C:\Data\Recordings\MyData.rd16

Then the SRDC file would be written to:

C:\Data\Recordings\MyData.rd16.srdc

The PX14400 library has routines that can be used to read and write these files.

See the Signatec Recorded Data Context (SRDC) section for more details.

3 | Software Development Reference

3.1 | Contents

The main PX14400 software installation installs several software components on the host system:

- A kernel-mode driver which allows the operating system to communicate with the PX14400 hardware:
    - Windows 32-bit Driver: **PX14.sys**
    - Windows 64-bit Driver: **PX14_64.sys**
    - Linux Driver Source: **driver** folder
- A user-mode library that interfaces with the driver. All PX14400 software accesses the PX14400 hardware through this library that is fully detailed in this software development section. This library exports a number of functions that may be used by custom software to control the PX14400 hardware:
    - Windows 32-bit Library: **PX14.dll**
    - Windows 64-bit Library: **PX14_64.dll**
    - Linux Library Source: **libsig_px14400** folder
- C source code example applications that are fully commented and demonstrate how to write PX14400 client programs:
    - Windows: **Examples** folder in the software installation path
    - Linux Examples Source: **examples** folder
- PX14400 Scope Application C source code for the full featured Windows application for operating PX14400 devices including running data recordings, viewing acquisition data, uploading firmware, and more:
    - Windows: **Source\PX14400 Scope Application** folder in the software installation path
    - Linux: Not Applicable

3.2 | Developing PX14400 Software

PC applications interface with the PX14400 through the PX14400 library. This library exports a variety of functions that are used to control the PX14400. All user-mode code access the PX14400 devices through this library. This library, in turn, is the only entity that interacts with the underlying PX14400 device driver. The PX14400 device driver is the only software entity that directly interacts with the PX14400 hardware.

The following sections describe how to set up the library and program the PX14400 with complete details on the available C library functions for custom software application development.

3.2.1 | Setting up the Build Environment – Windows Platform

The PX14400 library is composed of three parts:

- The header file: **px14.h**
- The import library: **PX14.lib** (32-bit) / **PX14_64.lib** (64-bit)
- The dynamic link library: **PX14.dll** (32-bit) / **PX14_64.dll** (64-bit)

The header and import library are required to build PX14400 applications and the dynamic link library is required to run PX14400 applications.

The header file, px14.h, is the C interface to the library. Your PX14400 applications will "#include" this file to define PX14400 data structures, function prototypes, constants, and macros. This header file, px14.h, is installed to the **include** folder of the PX14400 software installation path. It is recommended that you add this path to your compiler's header file search path[1]. Adding the header file to the include file search path will allow you to keep a single copy of px14.h on your system and access it like a standard C header file. (That is, you can do a "#include <px14.h>" instead of an "#include "*Path-to-px14.h*".) The same header file is used for both 32- and 64-bit code.

The import library, PX14.lib (32-bit code) or PX14_64.lib (64-bit code), is the stub-library that the linker needs to resolve linkage issues when building your application. This stub-library is installed to the **lib** folder of the of the PX14400 software installation path. Like the library header file, it is recommended that you put this library in your build environment's library file search path. This file needs to be included by the build process or you will get linker errors when you build your application. If you are using a Microsoft Visual C/C++ compiler, the stub-library will automatically be added to the build process when you include the library header so you do not have to manually add this file to your project. (Automatic linking with the stub library can be overridden by "#defining" PX14PP_NO_LINK_LIBRARY before including px14.h.)

The dynamic link library, PX14.dll (32-bit code) or PX14_64.lib (64-bit code), is installed to the PX14400 software installation folder (C:\Program Files or Program Files (x86)\Signatec\PX14400 by default). During software installation, the system library search path is updated to include this folder so the PX14400 library can be located when needed.

The 64-bit PX14400 Windows software installation will install the 32-bit library, import library, and dependent third-party libraries to the "Lib (32-bit)" folder. These 32-bit libraries are included as a convenience for those writing custom PX14400 software and need to build for a 32-bit architecture.

3.2.2 | Setting up the Build Environment – Linux Platform

The PX14400 library is composed of two files: the header file (px14.h) and library binary (libsig_px14400.so). Both files are required to build PX14400 applications. Only the library binary is required to run PX14400 applications.

The header file, px14.h, is the C interface to the library. Your PX14400 applications will "#include" this file to define PX14400 data structures, function prototypes, constants, and macros. The main PX14400 software build

[1] Microsoft Visual C++ 6 users can do this by selecting *Options* from the *Tools* menu, and selecting the *Directories* tab. Microsoft Visual Studio .NET users can do this by selecting *Options* from the *Tools* menu, and selecting *VC++ Directories* under the *Projects* folder.

process will put the PX14400 library header in the "/usr/local/include" so that it is accessed like a standard C header file. (That is, you can do a "#include <px14.h>" instead of a "#include "*Path-to-px14.h*".)

PX14400 applications must be linked with the PX14400 library. This is done by adding the "-lsig_px14400" linker option to the gcc/g++ command line. For an example of this, refer to the Makefile for one of the PX14400 example applications.

Note that filenames are case sensitive on the Linux platform.

3.2.3 | Library Functions That Use Character Strings

For library functions that accept or generate character strings (e.g. GetErrorTextPX14), the PC platform library may actually implement two versions: one version that works with ASCII strings (char*) and another version that works with UNICODE strings (wchar_t*). The ASCII version function names are suffixed with an 'APX14' and the UNICODE version function names are suffixed with 'WPX14'.

Macros have been defined so that library users can write generic code that will work with either version. Consider GetErrorTextPX14 as an example. The library implements two versions of this function: GetErrorTextAPX14 and GetErrorTextWPX14. If the _UNICODE symbol is defined, then the GetErrorTextPX14 macro will expand to GetErrorTextWPX14, else it will default to GetErrorTextAPX14. In this manual library functions are documented using the character-size agnostic type TCHAR.

```
int GetErrorTextAPX14 (int res, char** bufpp, unsigned int flags, HPX14 hBrd);
int GetErrorTextWPX14 (int res, wchar_t** bufpp, unsigned int flags, HPX14 hBrd);

#ifdef _UNICODE
# define GetErrorTextPX14       GetErrorTextWPX14
#else
# define GetErrorTextPX14       GetErrorTextAPX14
#endif

// Effective library function prototype:
int GetErrorTextPX14 (int res, TCHAR** bufpp, unsigned int flags, HPX14 hBrd);
```

3.2.4 | Multithreading Considerations

PX14400 library functions are thread-safe, but with one qualification. The functions themselves are thread-safe; they do not modify any global memory or rely on internal static data. However, most of the library functions take a device handle (HPX14) and then act on the internal data associated with that handle. Access to this handle-specific data is not thread-safe.

For example, if you have two separate threads of execution and each thread is working with a different device then there is no concern. On the other hand, if you have two threads of execution using the same device handle (and no explicit synchronization measures) then there is a risk of corrupting the internal device state which can cause all kinds of problems.

There is no problem with interacting with the same underlying PX14400 device from multiple threads of execution. For example, one thread may be recording data to a file and another thread may be used to check device status, adjust DC-offset or input voltage range, or maybe stop the recording. In multithreaded scenarios like this, it is highly recommended to use different device handles (HPX14) for different threads. The library

function DuplicateHandlePX14 may be used to obtain a unique device handle that is connected to the same device as a given device handle.

Note: When using the Recording Session Management API, there is no need to use a duplicated handle when creating the recording session. The underlying code creates its own duplicate handle under the hood for the recording thread.

Microsoft Windows: On the Windows platform, using a different HPX14 device handle per thread is sometimes required. The reason for this is that if the underlying PX14400 device driver has a pending user-space request (such as a synchronous data transfer operation), then no other requests from the same HPX14 device handle will be received by the driver until the first one completes. In some scenarios, this can lead to a software deadlock situation. Note that this is a constraint of the operating system itself. To reiterate, this is only a problem if two threads are using the same HPX14 device handle. If both threads have unique HPX14 device handles then the driver can handle simultaneous requests just fine.

The underlying kernel-mode device driver is completely thread-safe. All hardware access is synchronized to prevent simultaneous user-space requests from corrupting data or hardware state.

3.3 | Library Utility

The functions in this section are generic library utility functions.

3.3.1 | FreeMemoryPX14

Form

```
int FreeMemoryPX14 (void* p);
```

Description

Free memory allocated by the PX14400 library.

Parameters

[in] *p*
The address of the PX14400 library-allocated data.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

Certain PX14400 library functions allocate memory on behalf of the caller and it is the caller's responsibility to free this memory when they are done. Use this function to free the memory. Do not use free or delete because the PX14400 library may have allocated from another heap and freeing could result in memory corruption or an application crash.

Do not free DMA buffers with this function; rather use the proper function FreeDmaBufferPX14 to free DMA buffers.

3.3.2 | GetErrorTextPX14

Form

```
int GetErrorTextPX14 (int res, TCHAR** bufpp, unsigned int flags, HPX14 hBrd = PX14_INVALID_HANDLE);
```

Description

Obtain a user-friendly string describing the given SIG_* error value.

Parameters

[in] *res*

The library error value for which to obtain information. This can be any (generic) SIG_*, or (PX14400-specific) SIG_PX14_* error value.

[out] *bufpp*

A pointer to a char pointer that will receive the address of the library allocated buffer containing the error text string. This buffer must be freed by caller by calling the FreeMemoryPX14 library function

[in] *flags*

A set of flags that dictate the function behavior.

Flag	Interpretation
PX14ETF_IGNORE_SYSERROR (0x00000001)	Ignore system specific error information (Windows: GetLastError, Linux: errno)
PX14ETF_NO_SYSERROR_TEXT (0x00000002)	Do not generate any text for system specific error
PX14ETF_FORCE_SYSERROR (0x00000004)	Force inclusion of system error information even if it might not be relevant

[in] *hBrd*

A handle to the PX14400 device for which the error occurred. For some types of errors, the PX14400 handle used when the error was generated may contain additional context information. This parameter may be PX14_INVALID_HANDLE.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

This function is used to get a user-friendly string describing the given library error value. For ambiguous (and certain known) error values, information on the current (thread-specific) system error state is also provided.

Related Functions

ConnectToVirtualDevicePX14, DisconnectFromDevicePX14

3.4 | Device Enumeration and Connection Management

The functions in this section involve managing PX14400 device connections and general PX14400 device handle management.

3.4.1 | ConnectToDevicePX14

Form

```
int ConnectToDevicePX14 (HPX14* hBrd, unsigned int brdNum = 1);
```

Description

This function is used to establish a connection to a PX14400 data acquisition device, represented by a PX14400 device handle of type HPX14.

Parameters

[in] *hBrd*

A pointer to a HPX14 variable that will receive the PX14400 device handle. This PX14400 device handle is only valid in the current process. This handle should be treated as an opaque object; interpretation of the handle is PX14400 library specific.

[in] *brdNum*

This parameter is used to select which PX14400 in the system to connect to. If this value is in the range [1, PX14_MAX_DEVICES (16)] then the number is assumed to be the ordinal number of the device in the system. A board's ordinal represents the system-specific enumeration of the device. This may or may not follow the order of the physical device slots. If this value is outside the aforementioned range then it is assumed to be the PX14400 serial number of the device in which to connect.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

This function does not interact with the underlying PX14400 hardware device in any way. The device driver's hardware register cache is consulted for software register values and the operating mode is left unchanged. This allows a thread to attach to a currently running PX14400 device in a non-intrusive manner.

Like other handle-based mechanisms, the actual value of a PX14400 device handle should be treated as an opaque value. The exception to this is a special value, PX14_INVALID_HANDLE, which is used to identify an invalid PX14400 device handle.

An application should close the connection to the PX14400 device by calling the DisconnectFromDevicePX14 library function. Failure to disconnect from the device can result in memory leaks in the calling process.

Related Functions

ConnectToVirtualDevicePX14, DisconnectFromDevicePX14

3.4.2 | ConnectToVirtualDevicePX14

Form

```
int ConnectToVirtualDevicePX14 (HPX14* hBrd, unsigned int serialNum, unsigned int brdNum);
```

Description

Establish a connection to a virtual (fake) PX14400 device.

Parameters

[in] *hBrd*
A pointer to a HPX14 variable that will receive the virtual PX14400 device handle.

[in] *serialNum*
The serial number to use for the virtual device. This can be any number; the PX14400 library ignores this value. This will be the number obtained via the GetSerialNumberPX14 function.

[in] *brdNum*
The board number to use for the virtual device. This can be any number; the PX14400 library ignores this value. This will be the number obtained by the GetOrdinalNumberPX14 function

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

This function is used to establish a connection to a virtual PX14400 data acquisition device. A virtual device is one that is not connected to any real PX14400 hardware. Virtual devices are mainly used for software development and debugging.

Related Functions

ConnectToDevicePX14, DisconnectFromDevicePX14

3.4.3 | DisconnectFromDevicePX14

Form

```
int DisconnectFromDevicePX14 (HPX14 hBrd);
```

Description

This function is used to release the handle for the PX14400 when it is no longer needed in the program. The function also frees any utility DMA buffers allocated by the library for the given PX14400 device.

Parameters

[in] *hBrd*

The PX14400 device handle to close. This handle ceases to be valid once this function returns successfully. The function will just return SIG_SUCCESS if this parameter is PX14_INVALID_HANDLE.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

In general, closing a PX14400 device handle will have no effect on the underlying hardware. This allows one to connect/disconnect to a PX14400 device in an unobtrusive manner. A slight exception to this is that the PX14400 driver may interact with the PX14400 hardware when all connections to a particular PX14400 (over all processes in the system) have been closed.

This function does not alter the current operating mode. You will typically want to put the PX14400 into Standby mode before your application exits. This function does not automatically do this because it is possible that the device may still be in use by another thread or process.

This function should also be used to disconnect from a virtual device.

Related Functions

ConnectToDevicePX14

3.4.4 | DuplicateHandlePX14

Form

```
int DuplicateHandlePX14 (HPX14 hBrd, HPX14* phNew);
```

Description

Duplicate an PX14400 device handle.

Parameters

[in] *hBrd*

The PX14400 device handle to duplicate. This handle must have been obtained from the current process.

[out] *phNew*

A pointer to a HPX14 variable that will receive the new, duplicated handle.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Related Functions

ConnectToDevicePX14

3.4.5 | GetDeviceCountPX14

Form

```
int GetDeviceCountPX14();
```

Description

Obtain a count of all PX14400 devices present in the local system.

Return Value

Returns the number of physical PX14400 devices present in the local system. If this function returns zero then either no PX14400 devices are present or the PX14400 driver has not been installed.

3.4.6 | IsDeviceVirtualPX14

Form

```
int IsDeviceVirtualPX14 (HPX14 hBrd);
```

Description

Determines if a given PX14400 device handle is for a virtual device.

Parameters

[in] *hBrd*

The PX14400 device handle to check. A PX14400 device handle is obtained by calling the ConnectToDevicePX14 or ConnectToVirtualDevicePX14 function.

Return Value

Returns a positive value if the given handle is connected to a virtual device, zero if the handle is not connected to a virtual device, or one of the library error codes (which are all negative) on error.

Remarks

Use this function to determine if the given handle is associated with a virtual PX14400 device. A virtual PX14400 is not connected to real PX14400 hardware and is mainly used for software development and debugging.

Related Functions

IsHandleValidPX14

3.4.7 | IsHandleValidPX14

Form

```
int IsHandleValidPX14 (HPX14 hBrd);
```

Description

Determines if the given PX14400 device handle is connected to a device.

Parameters

[in] *hBrd*

The PX14400 device handle to check. A PX14400 device handle is obtained by calling the ConnectToDevicePX14 or ConnectToVirtualDevicePX14 function.

Return Value

Returns a positive value if the given handle is connected to a valid device, zero if the handle is not connected to a valid device, or one of the library error codes (which are all negative) on error.

Remarks

A handle is valid if it is connected to a local or virtual PX14400 device.

Related Functions

IsDeviceVirtualPX14

3.5 | Device State and Configuration

The functions in this section are used to obtain state and configuration information on the PX14400.

3.5.1 | ClearAcqFifoFlagPX14

Form

```
int ClearAcqFifoFlagPX14 (HPX14 hBrd);
```

Description

Clear the acquisition FIFO flag.

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

This function call must be followed by ClearDcmLockPX14.

Related Functions

GetAcqFifoLatchedOverflowFlagPX14

3.5.2 | ClearDcmLockPX14

Form

```
int ClearDcmLockPX14 (HPX14 hBrd);
```

Description

Clear the DCM lock status.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Related Functions

GetAcqDcmLatchedLockStatusPX14

3.5.3 | GetAcqDcmLatchedLockStatusPX14

Form

```
int GetAcqDcmLatchedLockStatusPX14 (HPX14 hBrd);
```

Description

Get status of the latched acquisition DCM lock flag.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

Return Value

Returns 1 if the latched acquisition DCM is locked, 0 if it's not locked or one of the library error codes (which are all negative) on error.

Remarks

This function call must be followed by ClearDcmLockPX14.

Related Functions

ClearDcmLockPX14, GetAcqDcmLockStatusPX14

3.5.4 | GetAcqDcmLockStatusPX14

Form

```
int GetAcqDcmLockStatusPX14 (HPX14 hBrd);
```

Description

Get status of the acquisition DCM lock flag.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

Return Value

Returns 1 if the acquisition DCM is locked, 0 if it's not locked or one of the library error codes (which are all negative) on error.

Related Functions

GetAcqDcmLatchedLockStatusPX14

3.5.5 | GetAcqFifoLatchedOverflowFlagPX14

Form

```
int GetAcqFifoLatchedOverflowFlagPX14 (HPX14 hBrd);
```

Description

Get status of the acquisition FIFO latched overflow flag.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

Return Value

Returns 1 if the acquisition FIFO latched is currently in overflow, 0 if it's not or one of the library error codes (which are all negative) on error.

Remarks

This function call must be followed by ClearAcqFifoFlagPX14.

Related Functions

ClearAcqFifoFlagPX14

3.5.6 | GetActualAdcAcqRatePX14

Form

```
int GetActualAdcAcqRatePX14 (HPX14 hBrd, double* pRateMHz);
```

Description

Obtain the actual ADC acquisition rate in MHz.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *pRateMHz*
A pointer to a float variable that will receive the effective clock rate in MHz.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

This routine is used to obtain the actual ADC acquisition rate taking the current ADC clock source and any clock dividers into consideration.

This routine does not consider post-ADC clock division emulation. For the effective acquisition rate, which does take post-ADC division into consideration, use the GetEffectiveAcqRatePX14 function.

Related Functions

GetEffectiveAcqRatePX14

3.5.7 | GetEffectiveAcqRatePX14

Form

```
int GetEffectiveAcqRatePX14 (HPX14 hBrd, double* pRateMHz);
```

Description

Obtain effective acquisition rate considering post-ADC clock division emulation.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *pRateMHz*
A pointer to a float variable that will receive the effective clock rate in MHz.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Related Functions

GetActualAdcAcqRatePX14, SetPostAdcClockDividerPX14

3.5.8 | GetFifoFullFlagPX14

Form

```
int GetFifoFullFlagPX14 (HPX14 hBrd);
```

Description

Get status of the RAM FIFO full flag; use with RAM-buffered acquisitions.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

Return Value

Returns 1 if an internal FIFO has overflowed at some point during the acquisition, 0 if no FIFOs have overflowed or one of the library error codes (which are all negative) on error.

Remarks

The function GetFifoFullFlagPX14 checks the status of the FIFO Full Flag that indicates whether either of the two on-board first-in-first-out (FIFO) data buffers has become full during an acquisition to the PCIe bus or whether the RAM itself has overflowed in either of the buffered acquisition modes.

This flag is latched so that once an overflow condition is detected it will remain set until a transition into an acquisition mode from standby mode.

Explicitly calling this library function is not necessary; the PX14400 and driver will automatically consult the FIFO overflow flag when relevant and, if set, will return SIG_PX14_FIFO_OVERFLOW. This includes synchronous calls to GetPciAcquisitionDataFastPX14 or GetPciAcquisitionDataBufPX14 as well as WaitForTransferCompletePX14.

3.5.9 | GetFPGATemperaturePX14

Form

```
int GetFPGATemperaturePX14 (HPX14 hBrd, int bFromCache = 1);
```

Description

Get the current temperature status of the primary system FPGA reported in Celsius.

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given PX14400 handle, which will result in no hardware or driver access. If zero, the setting is obtained from the driver which may or may not result in an actual PX14400 device register read.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

On success, GetFPGATemperaturePX14 returns the temperature of the primary system FPGA in Celsius.

Remarks

This function is useful for thermal monitoring of the PX14400.

3.5.10 | GetHardwareRevisionPX14

Form

```
int GetHardwareRevisionPX14 (HPX14 hBrd, unsigned long long* verp);
```

Description

Obtains the revision of the PX14400 hardware.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[out] *verp*
A pointer to the variable that will receive the version.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

The PX14400 hardware revision is a 64-bit value broken up into four 16-bit fields:

Field	Mask
Major version	0xFFFF000000000000ULL
Minor version	0x0000FFFF00000000ULL
Sub-minor version	0x00000000FFFF0000ULL
Package number	0x000000000000FFFFULL

Related Functions

GetSabFirmwareVersionPX14, GetSysFirmwareVersionPX14

3.5.11 | GetOrdinalNumberPX14

Form

```
int GetOrdinalNumberPX14 (HPX14 hBrd, unsigned int* onp);
```

Description

Obtain the ordinal number of the PX14400 connected to the given handle.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[out] *onp*
A pointer to the variable that will receive the PX14400's ordinal number.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

A PX14400's ordinal number is the number of the PX14400 in the system, as in the first, second, third, etc. board. This order is determined by the system and may or may not be the same order as the physical PCIe slots.

Related Functions

GetSerialNumberPX14

3.5.12 | GetPllLockStatusPX14

Form

```
int GetPllLockStatusPX14 (HPX14 hBrd);
```

Description

Get lock status of the PLL.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

Return Value

Returns 1 if PLL is locked, 0 if unlocked, or one of the library error codes (which are all negative) on error.

3.5.13 | GetSabFirmwareVersionPX14

Form

```
int GetSabFirmwareVersionPX14 (HPX14 hBrd, unsigned long long* verp);
```

Description

Obtains the version of the PX14400 sab FPGA processing firmware.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[out] *verp*

A pointer to the variable that will receive the version.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

The PX14400 sab FPGA processing firmware version is a 64-bit value broken up into four 16-bit fields:

Field	Mask
Major version	0xFFFF000000000000ULL
Minor version	0x0000FFFF00000000ULL
Sub-minor version	0x00000000FFFF0000ULL
Package number	0x000000000000FFFFULL

Related Functions

GetHardwareRevisionPX14, GetSysFirmwareVersionPX14

3.5.14 | GetSamplesCompleteFlagPX14

Form

```
int GetSamplesCompleteFlagPX14 (HPX14 hBrd);
```

Description

Get status of the samples complete flag; set when a RAM acquisition completes.

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

Return Value

Returns 1 if the Samples Complete Flag is set (indicating acquisition complete), 0 if the flag is not set (indicating that acquisition has not completed), or one of the library error codes (which are all negative) on error.

Remarks

This function checks the status of the PX14400 Samples Complete flag (SCF) and returns the flag status. A value of 1 indicates that the samples count is complete and a value of 0 indicates that the transfer/acquisition is not yet complete. The Samples Complete flag is meaningful both for detecting the end of data acquisition and also for detecting the end of data transfer.

The WaitForAcquisitionCompletePX14 function is a more reliable way of waiting for acquisition complete.

3.5.15 | GetSerialNumberPX14

Form

```
int GetSerialNumberPX14 (HPX14 hBrd, unsigned int* snp);
```

Description

Obtain the PX14400 board's serial number.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[out] *snp*
A pointer to the variable that will receive the PX14400's serial number.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Related Functions

GetOrdinalNumberPX14

3.5.16 | GetSysFirmwareVersionPX14

Form

```
int GetSysFirmwareVersionPX14 (HPX14 hBrd, unsigned long long* verp);
```

Description

Obtain the version of the PX14400 system firmware.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[out] *verp*
A pointer to the variable that will receive the version.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

The PX14400 system firmware version is a 64-bit value broken up into four 16-bit fields:

Field	Mask
Major version	0xFFFF000000000000ULL
Minor version	0x0000FFFF00000000ULL
Sub-minor version	0x00000000FFFF0000ULL
Package number	0x000000000000FFFFULL

Related Functions

GetHardwareRevisionPX14, GetSabFirmwareVersionPX14

3.5.17 | GetSystemInternalClockRatePX14

Form

```
int GetSystemInternalClockRatePX14 (HPX14 hBrd);
```

Description

This function is used to measure the internal clock rate used inside the system FPGA. This measurement is done based from the system clock that is used as a reference clock. In all cases, the internal clock rate should be set to be always lower than 400 MHz to avoid the acquisition FIFO to become full and lose data.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

Return Value

Returns the internal clock rate or one of the library error codes (which are all negative) on error.

Related Functions

GetActualAdcAcqRatePX14, GetEffectiveAcqRatePX14

3.5.18 | ReadConfigEepromPX14

Form

```
int ReadConfigEepromPX14 (HPX14 hBrd, unsigned int eeprom_addr, unsigned short* eeprom_datap);
```

Description

Read an element from the PX14400 configuration EEPROM

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] eeprom_addr
The EEPROM address that to be read. Valid addresses that user applications may use are within the range [0x80, 0xFF]. All values below address 0x80 are reserved for internal use and are read and write protected by the library.

[out] *eeprom_datap*
A pointer to the variable that will receive the EEPROM data.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

Each PX14400 board has a small configuration EEPROM that is used to contain board-specific configuration data. A region of this EEPROM is set aside for application specific configuration data.

Related Functions

WriteConfigEepromPX14

3.5.19 | WriteConfigEepromPX14

Form

```
int WriteConfigEepromPX14 (HPX14 hBrd, unsigned int eeprom_addr, unsigned short eeprom_data);
```

Description

Write an element from the PX14400 configuration EEPROM.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *eeprom_addr*
The EEPROM address to be written. Valid addresses that user applications may use are within the range [0x80, 0xFF]. All values below address 0x80 are reserved for internal use and are read and write protected by the library.

[in] *eeprom_datal*
The value that will be written to the specified EEPROM address.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Related Functions

ReadConfigEepromPX14

3.6 | Hardware Settings

The functions in this section control the getting and setting of the various PX14400 hardware settings. These settings include the numerous data acquisition, clock, and trigger parameters.

3.6.1 | GetInputVoltageRangeVoltsCh*X*PX14

Form

int GetInputVoltageRangeVoltsCh1PX14 (HPX14 hBrd, double* voltsp, int bFromCache = 1);
int GetInputVoltageRangeVoltsCh2PX14 (HPX14 hBrd, double* voltsp, int bFromCache = 1);

Description

Get the input voltage range, in peak-to-peak voltage, for channel 1 or 2. (AC-Coupled devices only: PX14400A)

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *voltsp*
A pointer to the variable that will receive the current input voltage range in peak-to-peak voltage.

[in] *bFromCache*
If non-zero, the setting will be read from the local device register cache associated with the given PX14400 handle, which will result in no hardware or driver access. If zero, the setting is obtained from the driver which may or may not result in an actual PX14400 device register read.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

This function is only relevant for AC-Coupled device models: PX14400A.

This function obtains the actual peak-to-peak input voltage range; which differs from the GetInputVoltRangeChXPX14 function that obtains the input voltage range selection (PX14VOLTRNG_*) value.

The SetInputVoltRangeChXPX14 functions are used to select the input voltage range selection.

Related Functions

SelectInputVoltRangePX14, SetInputVoltRangeChXPX14

3.6.2 | GetInputVoltRangeFromSettingPX14

Form

```
int GetInputVoltRangeFromSettingPX14 (int vr_sel, double* vpp);
```

Description

Get the input voltage range, in peak-to-peak voltage, represented by an input voltage selection enumeration (PX14VOLTRNG_*).

Parameters

[in] *vr_sel*

The input voltage range selection value to obtain peak-to-peak input voltage for. See SetInputVoltRangeCh*X*PX14.

[in] *vpp*

A pointer to a double variable that will receive the peak-to-peak input voltage represented by the PX14VOLTRNG_* value specified in the *vr_sel* parameter.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Related Functions

GetInputVoltageRangeVoltsChXPX14

3.6.3 | IssueSoftwareTriggerPX14

Form

```
int IssueSoftwareTriggerPX14 (HPX14 hBrd);
```

Description

Issue a software-generated trigger event to a PX14400.

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

This function allows the user to issue a software trigger to the PX14400 board. After a software trigger has been issued, the next (or current) data acquisition will begin immediately and will not wait for a physical trigger event to occur.

Related Functions

SetTriggerLevelAPX14, SetTriggerModePX14, SetTriggerSourcePX14

3.6.4 | SelectInputVoltRangePX14

Form

```
int SelectInputVoltRangePX14 (HPX14 hBrd, double vr_pp, int bCh1 = PX14_TRUE, int bCh2 = PX14TRUE, int sel_how = PX14VRSEL_NOT_LT );
```

Description

Select input voltage ranges for channel 1 and/or channel 2 inputs. (AC-Coupled devices only: PX14400A)

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *vr_pp*
The desired input voltage range in peak-to-peak volts.

[in] *bCh1*
If non-zero, the function will apply the selected input voltage range to channel 1.

[in] *bCh2*
If non-zero, the function will apply the selected input voltage range to channel 2.

[in] *sel_how*
Selects how the function will constrain the selection of the input voltage range. Can be any of the following values:

PX14400 Library Constant	Interpretation
PX14VRSEL_NOT_LT (0)	Select voltage range not less than given value, *vr_pp*.
PX14VRSEL_NOT_GT (1)	Select voltage range not greater than given value, *vr_pp*.
PX14VRSEL_CLOSEST (2)	Select voltage range closest to given value, *vr_pp*.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

This function is only relevant for AC-Coupled (PX14400A), amplifier front end configuration (-AMP-) device models.

This function is used to programmatically select the input voltage range for one or both channels by specifying the desired input voltage range in peak-to-peak voltage.

This function sets the input voltage range given a desired peak-to-peak voltage; which differs from the SetInputVoltRangeCh*X*PX14 function that sets the input voltage range from an enumeration representing a particular input voltage range (PX14VOLTRNG_*).

The input voltage range selections can be changed while the PX14400 is in active acquisition modes.

Related Functions

SetInputVoltRangeCh*X*PX14

3.6.5 | SetActiveChannelsPX14 / GetActiveChannelsPX14

Form

int SetActiveChannelsPX14 (HPX14 hBrd, unsigned int val);
int GetActiveChannelsPX14 (HPX14 hBrd, int bFromCache = 1);

Description

Set or get the active channel selection; defines which channels are digitized.

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *val*

Selects which channels will be active for the next data acquisition. This parameter may be any of the following:

PX14400 Library Constant	Interpretation
PX14CHANNEL_DUAL (0)	Dual Channels (Channel 1 and Channel 2)
PX14CHANNEL_ONE (1)	Single Channel (Channel 1 only)
PX14CHANNEL_TWO (2)	Single Channel (Channel 2 only)

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given PX14400 handle, which will result in no hardware or driver access. If zero, the setting is obtained from the driver which may or may not result in an actual PX14400 device register read.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

On success, GetActiveChannelsPX14 will return the current active channel selection.

Remarks

The active channel selection may only be changed while the PX14400 is in the Standby operating mode.

3.6.6 | SetAdcClockSourcePX14 / GetAdcClockSourcePX14

Form

int SetAdcClockSourcePX14 (HPX14 hBrd, unsigned int val);
int GetAdcClockSourcePX14 (HPX14 hBrd, int bFromCache = 1);

Description

Set or get the source ADC clock setting.

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *val*

The identifier of the ADC clock source to set. Can be any of the following:

PX14400 Library Constant	Interpretation
PX14CLKSRC_INT_VCO (0)	Internal voltage-controlled oscillator (VCO)
PX14CLKSRC_EXTERNAL (1)	External clock supplied on the PX14400's external clock input

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given PX14400 handle, which will result in no hardware or driver access. If zero, the setting is obtained from the driver which may or may not result in an actual PX14400 device register read.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

On success, GetAdcClockSourcePX14 will return the current ADC clock source selection.

Remarks

Do not change the clock source or frequency while an acquisition is in progress. The ADC clock source may only be changed while the PX14400 is in the Standby operating mode.

Internal Voltage-Controlled Oscillator:

When this clock source is selected, the acquisition rate is defined by the SetInternalAdcClockRatePX14 function. When this clock source is selected, user-specified clock divider values are ignored. (The clock dividers are used internally to setup acquisition rate.)

External clock:

When this clock source is used, the SetExternalClockRatePX14 function should be used to specify the external clock rate so the software can properly track things like effective acquisition rate, calculate FFT statistics, etc. The underlying PX14400 firmware needs to know the actual clock rate for synchronization purposes.

Related Functions

SetExtClockDividersPX14, SetExternalClockRatePX14, SetInternalAdcClockRatePX14, SetPostAdcClockDividerPX14

3.6.7 | SetBoardProcessingEnablePX14 / GetBoardProcessingEnablePX14

Form

int SetBoardProcessingEnablePX14 (HPX14 hBrd, unsigned int bEnable);
int GetBoardProcessingEnablePX14 (HPX14 hBrd, int bFromCache = 1);

Description

Enable/disable PX14400 FPGA processing. (FPGA processing devices only: –SP50 and –SP95)

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *bEnable*

If this parameter is non-zero then PX14400 FPGA processing will be enabled. If this parameter is zero then PX14400 FPGA processing will be disabled. This setting only has effect if the PX14400 model supports custom FPGA processing.

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given PX14400 handle, which will result in no hardware or driver access. If zero, the setting is obtained from the driver which may or may not result in an actual PX14400 device register read.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

On success, GetBoardProcessingEnablePX14 returns the current board processing enable setting.

Remarks

This function is only relevant for PX14400 FPGA signal processing device models: –SP50 and –SP95.

This function is used only to enable or disable PX14400 FPGA processing.

When FPGA processing is enabled, acquired data is transferred from the System FPGA to the Signal Processing FPGA where the targeted signal processing routine is conducted.

When FPGA processing is disabled, acquired data is not transferred to the Signal Processing FPGA.

FPGA processing is a generic term applied to custom or user-defined logic running on the onboard dedicated signal processing FPGA. The actual signal processing routine that is executed is dependent on the underlying firmware loaded on the processing FPA, which may be user-defined. Depending on the underlying processing firmware, additional configuration may be required to satisfy the requirements of the processing routines. Consult specific FPGA processing firmware project documentation for additional details.

Related Functions

SetBoardProcessingParamPX14

3.6.8 | SetBoardProcessingParamPX14 / GetBoardProcessingParamPX14

Form

int SetBoardProcessingParamPX14 (HPX14 hBrd, unsigned short idx, unsigned short value);
int GetBoardProcessingParamPX14 (HPX14 hBrd, unsigned short idx, unsigned short* pVal);

Description

Set or get a PX14400 FPGA processing parameter. (FPGA processing devices only: –SP50 and –SP95)

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *idx*

The index of the PX14400 FPGA processing parameter to write. This can be any value in the range [0, 65535].

[in] *value*

The FPGA processing value to write. This can be any value in the range [0, 65535]. Interpretation of this value is dependent on the underlying FPGA signal processing firmware logic.

[out] *pVal*

A pointer to the variable that will receive the value read from the board processing parameter.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

On success, GetBoardProcessingParamPX14 returns the current board processing parameter value.

Remarks

This function is only relevant for PX14400 FPGA signal processing device models: –SP50 and –SP95.

This function is used to specify PX14400 FPGA processing parameter data.

This is a generic interface that allows the host PC to specify arbitrary arguments to custom PX14400 FPGA processing firmware. These parameters might include things like FFT sizes, FFT window data, or frequency specification for example.

Actual location and interpretation of the parameter data is dependent on the actual underlying FPGA processing firmware. This firmware will either be custom firmware generated by Signatec or by an end-user. Custom FPGA processing firmware created by Signatec will include documentation that defines the details of any processing parameters. End-users developing their own custom firmware will know the details of any parameters because they are the ones defining them.

Related Functions

SetBoardProcessingEnablePX14

3.6.9 | SetDcOffsetCh*X*PX14 / GetDcOffsetCh*X*PX14

Form

int SetDcOffsetCh1PX14 (HPX14 hBrd, unsigned int val);
int SetDcOffsetCh2PX14 (HPX14 hBrd, unsigned int val);
int GetDcOffsetCh1PX14 (HPX14 hBrd, int bFromCache = 1);
int GetDcOffsetCh2PX14 (HPX14 hBrd, int bFromCache = 1);

Description

Set or get the DC-offset for channel 1 or channel 2. (DC-Coupled devices only: PX14400D or PX14400D2)

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *val*

A value in the range [0, 4095] that defines the amount of DC offset added to the input signal. A value of 0 is equivalent to an offset equal to the minimum ADC input voltage range. A value of 4095 is equivalent to an offset equal to the maximum ADC input voltage range. A value of 2048 should produce no offset. The function will clip this value to the maximum valid value if necessary.

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given PX14400 handle, which will result in no hardware or driver access. If zero, the setting is obtained from the driver which may or may not result in an actual PX14400 device register read.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

On success, GetDcOffsetCh1PX14 returns the current channel 1 DC-offset.
On success, GetDcOffsetCh2PX14 returns the current channel 2 DC-offset.

Remarks

These functions are only relevant for DC-coupled PX14400 device models: PX14400D or PX14400D2. It is safe to call these functions for AC-coupled devices, but will have no effect.

The DC offset setting can be changed while the PX14400 is in active acquisition modes.

Related Functions

SetFineDcOffsetCh*X*PX14

3.6.10 | SetDigitalIoEnablePX14 / GetDigitalIoEnablePX14

Form

int SetDigitalIoEnablePX14 (HPX14 hBrd, unsigned int bEnable);
int GetDigitalIoEnablePX14 (HPX14 hBrd, int bFromCache = 1);

Description

Enable/disable the digital IO interface.

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *bEnable*

If this parameter is nonzero then the PX14400 digital IO interface will be enabled. If this parameter is zero then the digital IO interface will be disabled. The actual digital IO operation is determined by the SetDigitalIoModePX14 function.

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given PX14400 handle, which will result in no hardware or driver access. If zero, the setting is obtained from the driver which may or may not result in an actual PX14400 device register read.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

On success, GetDigitalIoEnablePX14 returns the current digital IO interface enable setting.

Related Functions

SetDigitalIoModePX14

3.6.11 | SetDigitalIoModePX14 / GetDigitalIoModePX14

Form

int SetDigitalIoModePX14 (HPX14 hBrd, unsigned int val);
int GetDigitalIoModePX14 (HPX14 hBrd, int bFromCache = 1);

Description

Set or get the digital IO interface mode; determines what is driven on PX14400 digital IO connector.

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *val*

Selects one of eight possible signals according to the following table:

PX14400 Library Constant	Digital Input/Output	Interpretation
PX14DIGIO_OUT_0V (0)	Output	0V
PX14DIGIO_OUT_SYNC_TRIG (1)	Output	Synchronized Trigger
PX14DIGIO_OUT_ADC_CLOCK_DIV_2 (2)	Output	ADC Clock divided by 2
PX14DIGIO_OUT_3_3V (3)	Output	3.3V
PX14DIGIO_IN_TS_GEN (4)	Input	Digital input pulse requests a timestamp
PX14DIGOUT_RESERVED_5 (5)	*Assumed Output*	*Reserved*
PX14DIGOUT_RESERVED_6 (6)	*Assumed Output*	*Reserved*
PX14DIGOUT_RESERVED_7 (7)	*Assumed Output*	*Reserved*

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given PX14400 handle, which will result in no hardware or driver access. If zero, the setting is obtained from the driver which may or may not result in an actual PX14400 device register read.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

On success, GetDigitalIoModePX14 returns the current digital output mode setting.

Remarks

This setting is used to define what gets driven on the PX14400 digital IO interface. This setting is only applicable when the digital IO interface is enabled, which is done by calling the SetDigitalIoEnablePX14 function.

CAUTION: If the digital IO port is configured as an output, the operator must ensure that no other external device is also driving the digital IO connector. Multiple devices driving the common bus can damage the PX14400. When changing modes it is highly recommended to first disable the digital IO interface and then select the desired digital IO mode before then re-enabling the digital IO interface.

Related Functions

SetDigitalIoEnablePX14

3.6.12 | SetExtClockDividersPX14 / GetExtClockDividersPX14

Form

int SetExtClockDividersPX14 (HPX14 hBrd, unsigned int val);
int GetExtClockDividersPX14 (HPX14 hBrd, int bFromCache = 1);

Description

Set or get the external clock divider value setting.

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *val*

The divider value to use for the external clock. This can be any value from 1 (no division) to 20.

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given PX14400 handle, which will result in no hardware or driver access. If zero, the setting is obtained from the driver which may or may not result in an actual PX14400 device register read.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

On success, GetExtClockDividersPX14 returns the external clock divider value.

Remarks

The external clock divider value is ignored when the internal clock is selected as the ADC clock source.

The PX14400 library will automatically apply the external clock divider value when the clock source is changed to external clock.

The external clock divider value may only be changed while the PX14400 is in the Standby operating mode.

Related Functions

SetAdcClockSourcePX14, SetExternalClockRatePX14

3.6.13 | SetExternalClockRatePX14 / GetExternalClockRatePX14

Form

int SetExternalClockRatePX14 (HPX14 hBrd, double dRateMHz);
int GetExternalClockRatePX14 (HPX14 hBrd, double* ratep, int bFromCache = 1);

Description

Set or get the assumed external clock rate in MHz.

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *dRateMHz*

The assumed external clock rate in MHz. This value must be in the range [20, 400]. An error will be returned if this is not a well-formed finite value.

[out] *ratep*

A pointer to the variable that will receive the current assumed external clock rate.

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given PX14400 handle, which will result in no hardware or driver access. If zero, the setting is obtained from the driver which may or may not result in an actual PX14400 device register read.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

On success, GetExternalClockRatePX14 returns the assumed external clock rate value.

Remarks

When using an applied external clock as the ADC clock, the PX14400 firmware needs to be aware of the actual external clock frequency so that it can properly synchronize other onboard digital clock manager (DCM) circuits to it.

When using an external clock source, once the PX14400 has entered a data acquisition mode the external clock must be stable for the duration of the data acquisition. If the external clock source is removed (or stopped and restarted) while in acquisition mode, the PX14400 firmware will lose synchronization and data errors may occur.

The external clock rate may only be changed while the PX14400 is in the Standby operating mode.

Related Functions

SetAdcClockSourcePX14, SetExtClockDividersPX14

3.6.14 | SetFineDcOffsetCh*X*PX14 / GetFineDcOffsetCh*X*PX14

Form

int SetFineDcOffsetCh1PX14 (HPX14 hBrd, unsigned int val);
int SetFineDcOffsetCh2PX14 (HPX14 hBrd, unsigned int val);
int GetFineDcOffsetCh1PX14 (HPX14 hBrd, int bFromCache = 1);
int GetFineDcOffsetCh2PX14 (HPX14 hBrd, int bFromCache = 1);

Description

Set or get the find DC-offset for channel 1 or channel 2. (DC-Coupled device model only: PX14400D2)

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *val*

A value in the range [0, 2047] that defines the amount of DC offset added to the input signal. A value of 0 is equivalent to the maximum negative swing. A value of 2047 is equivalent to the maximum positive swing. A value of 1024 is equivalent to no fine DC offset. 1 count of fine DC offset is equivalent to 1/64 of a count of the normal DC offset.

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given PX14400 handle, which will result in no hardware or driver access. If zero, the setting is obtained from the driver which may or may not result in an actual PX14400 device register read.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

On success, GetFineDcOffsetCh1PX14 returns the current channel 1 fine DC-offset.
On success, GetFineDcOffsetCh2PX14 returns the current channel 2 fine DC-offset.

Remarks

These functions are only relevant for the DC-coupled device model: PX14400D2. It is safe to call these functions for non-PX14400D2 device models, but will have no effect.

The fine DC offset differs from the normal DC offset in that 1 count of fine DC-offset is equivalent to 1/64 a count of normal DC offset. The fine DC offset is intended for DC offset adjustment when using the low amplitude input voltage ranges on the PX14400D2. For these lower amplitude ranges, the normal DC offset adjustment may be too coarse to provide adequate DC offset adjustment.

The fine DC offset setting can be changed while the PX14400 is in active acquisition modes.

Related Functions

SetDcOffsetChXPX14

3.6.15 | SetInputGainLevelDcPX14 / GetInputGainLevelDcPX14

Form

int SetInputGainLevelDcPX14 (HPX14 hBrd, unsigned int val);
int GetInputGainLevelDcPX14 (HPX14 hBrd, int bFromCache = 1);

Description

Set or get the input gain level. (DC-Coupled device model only: PX14400D)

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *val*

The input gain level to set. This can be any of the following values:

PX14400 Library Constant	Interpretation
PX14DGAIN_LOW (0)	Low gain (1.2Vp-p input; power-up default value)
PX14DGAIN_HIGH (1)	High gain (400mVp-p)

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given PX14400 handle, which will result in no hardware or driver access. If zero, the setting is obtained from the driver which may or may not result in an actual PX14400 device register read.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

On success, GetInputGainLevelDcPX14 returns the input gain level setting.

Remarks

These functions are only relevant for the DC-coupled device model: PX14400D.

The input gain selection can be changed while the PX14400 is in active acquisition modes.

Related Functions

SetInputVoltRangeCh*X*PX14 (for PX14400A or PX14400D2)

3.6.16 | SetInputVoltRangeCh*X*PX14 / GetInputVoltRangeCh*X*PX14

Form

int SetInputVoltRangeCh1PX14 (HPX14 hBrd, unsigned int val);
int SetInputVoltRangeCh2PX14 (HPX14 hBrd, unsigned int val);
int GetInputVoltRangeCh1PX14 (HPX14 hBrd, int bFromCache = 1);
int GetInputVoltRangeCh2PX14 (HPX14 hBrd, int bFromCache = 1);

Description

Set or get the input voltage range selection for channel 1 or 2 for PX14400 device models: PX14400A or PX14400D2.

NOTE: For the device model PX14400D, refer to the SetInputGainLevelDcPX14 function.

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given PX14400 handle, which will result in no hardware or driver access. If zero, the setting is obtained from the driver which may or may not result in an actual PX14400 device register read.

[in] *val*

The input voltage range selection to set. This can be any of the values in the following table detailed on the next page. See Remarks.

PX14400 Library Constant	Value	Interpretation
The following selections are valid only for PX14400A devices:		
PX14VOLTRNG_STATIC_1_100_VPP	0	Static 1.1 V_{p-p} (See Remarks)
PX14VOLTRNG_0_220_VPP	1	220 mV_{p-p}
PX14VOLTRNG_0_247_VPP	2	247 mV_{p-p}
PX14VOLTRNG_0_277_VPP	3	277 mV_{p-p}
PX14VOLTRNG_0_311_VPP	4	311 mV_{p-p}
PX14VOLTRNG_0_349_VPP	5	349 mV_{p-p}
PX14VOLTRNG_0_391_VPP	6	391 mV_{p-p}
PX14VOLTRNG_0_439_VPP	7	439 mV_{p-p}
PX14VOLTRNG_0_493_VPP	8	493 mV_{p-p}
PX14VOLTRNG_0_553_VPP	9	553 mV_{p-p}
PX14VOLTRNG_0_620_VPP	10	620 mV_{p-p}
PX14VOLTRNG_0_696_VPP	11	696 mV_{p-p}
PX14VOLTRNG_0_781_VPP	12	781 mV_{p-p}
PX14VOLTRNG_0_876_VPP	13	876 mV_{p-p}
PX14VOLTRNG_0_983_VPP	14	983 mV_{p-p}
PX14VOLTRNG_1_103_VPP	15	1.103 V_{p-p}
PX14VOLTRNG_1_237_VPP	16	1.237 V_{p-p}
PX14VOLTRNG_1_388_VPP	17	1.388 V_{p-p}
PX14VOLTRNG_1_557_VPP	18	1.557 V_{p-p}
PX14VOLTRNG_1_748_VPP	19	1.748 V_{p-p}
PX14VOLTRNG_1_961_VPP	20	1.961 V_{p-p}
PX14VOLTRNG_2_200_VPP	21	2.200 V_{p-p}
PX14VOLTRNG_2_468_VPP	22	2.468 V_{p-p}
PX14VOLTRNG_2_770_VPP	23	2.770 V_{p-p}
PX14VOLTRNG_3_108_VPP	24	3.108 V_{p-p}
PX14VOLTRNG_3_487_VPP	25	3.487 V_{p-p}
The following selections are valid only for PX14400D2 devices:		
PX14VOLTRNG_D2_3_00_VPP	26	3.00 V_{p-p}
PX14VOLTRNG_D2_1_60_VPP	27	1.60 V_{p-p}
PX14VOLTRNG_D2_1_00_VPP	28	1.00 V_{p-p}
PX14VOLTRNG_D2_0_600_VPP	29	600 mV_{p-p}
PX14VOLTRNG_D2_0_333_VPP	30	333 mV_{p-p}
PX14VOLTRNG_D2_0_200_VPP	31	200 mV_{p-p}

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

GetInputVoltageRangeChXPX14 will always return PX14VOLTRNG_STATIC_1_100_VPP for non-amplified front end configuration devices.

Remarks

This function is only relevant for PX14400A and PX14400D2 devices. Input voltage range selection for PX14400D devices is configured with the SetInputGainLevelDcPX14 function.

The SetInputVoltRangeCh*X*PX14 function sets the input voltage range from an enumeration representing a particular input voltage range (PX14VOLTRNG_*); which differs from the SelectInputVoltRangePX14 that sets the input voltage range given a desired peak-to-peak voltage range value.

The GetInputVoltageRangeCh*X*PX14 function gets the input voltage range from an enumeration representing a particular input voltage range (PX14VOLTRNG_*); which differs from the GetInputVoltageRangeVoltsCh*X*PX14 that gets the actual peak-to-peak input voltage range value.

PX14400A devices have input channels that can be configured with either an amplifier (-AMP-) or a non-amplified transformer (-XF-) front end. PX14400D and PX14400D2 devices do not support a non-amplified transformer based front end configuration and always have amplified based input channels.

Non-Amplified Devices

The PX14VOLTRNG_STATIC_1_100_VPP input voltage selection value is reserved for non-amplified PX14400 devices. For non-amplified devices, calling SetInputVoltRangeCh*X*PX14 with a value other than PX14VOLTRNG_STATIC_1_100_VPP will result in the SIG_PX14_INVALID_CHAN_IMP error being returned.

Amplified Devices

For PX14400A devices: if the *val* parameter is PX14VOLTRNG_STATIC_1_100_VPP and the PX14400 device is configured as amplified, then the PX14VOLTRNG_1_103_VPP (1.103 Vp-p) setting will be used since it is the closest range not smaller than 1.1 V_{p-p}.

The input voltage range selections can be changed while the PX14400 is in active acquisition modes.

Related Functions

SelectInputVoltRangePX14, GetInputVoltageRangeVoltsChXPX14, SetInputGainLevelDcPX14 (PX14400D devices)

3.6.17 | SetInternalAdcClockRatePX14 / GetInternalAdcClockRatePX14

Form

int SetInternalAdcClockRatePX14 (HPX14 hBrd, double dRateMHz);
int GetInternalAdcClockRatePX14 (HPX14 hBrd, double* ratep, int bFromCache = 1);

Description

Set or get the ADC clock rate; only applies to the internal clock source.

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *dRateMHz*

The clock rate (i.e. sampling rate), in MHz, to use when the internal clock is selected as the source clock. This value must be in the range of 20 MHz and 400 MHz, with the exception of an un-settable frequency range that cannot be reached with the internal clock: 277.5 MHz to 308.3 MHz (Δ 30.8 MHz).

[out] *ratep*

A pointer to the variable that will receive the current internal clock rate when the internal clock is selected as the ADC clock source.

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given PX14400 handle, which will result in no hardware or driver access. If zero, the setting is obtained from the driver which may or may not result in an actual PX14400 device register read.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

The internal ADC clock source is a voltage controlled oscillator (VCO). This means that by varying a voltage input the oscillator can change its frequency. This, in conjunction with onboard clock dividers and a phase lock loop (PLL), allow for a wide range of possible acquisition rates.

The internal ADC clock rate may only be changed while the PX14400 is in the Standby operating mode.

Related Functions

SetAdcClockSourcePX14

3.6.18 | SetInternalAdcClockReferencePX14 / GetInternalAdcClockReferencePX14

Form

int SetInternalAdcClockReferencePX14 (HPX14 hBrd, unsigned int val);
int GetInternalAdcClockReferencePX14 (HPX14 hBrd, int bFromCache = 1);

Description

Set or get the ADC internal clock reference.

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *val*

The internal ADC clock reference selection to set. This can be any of the values in the following table.

PX14400 Library Constant	Interpretation
PX14CLKREF_INT_10MHZ (0)	Internal 10MHz clock reference (Power-up default)
PX14CLKREF_EXT (1)	Externally supplied 10MHz reference clock

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given PX14400 handle, which will result in no hardware or driver access. If zero, the setting is obtained from the driver which may or may not result in an actual PX14400 device register read.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

On success, GetInternalAdcClockReferencePX14 returns the internal ADC clock reference selection.

The internal ADC clock reference source may only be changed while the PX14400 is in the Standby operating mode.

Related Functions

SetAdcClockSourcePX14

3.6.19 | SetOperatingModePX14 / GetOperatingModePX14

Form

int SetOperatingModePX14 (HPX14 hBrd, unsigned int val);
int GetOperatingModePX14 (HPX14 hBrd, int bFromCache = 1);

Description

Set or get the PX14400's operating mode.

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *val*

The operating mode to switch to. This can be any of the following values detailed in the table on the next page:

PX14400 Library Constant	Interpretation
PX14MODE_STANDBY (1)	Standby (ready) Mode; the primary idle operating mode. This is the operating mode that should be used to configure the PX14400 hardware settings prior to a data acquisition.
PX14MODE_ACQ_RAM (2)	RAM Acquisition Mode; data is acquired directly to the PX14400 onboard RAM. This mode can support the maximum acquisition rate for both input channels.
PX14MODE_ACQ_PCI_BUF (4)	PCIe Buffered Acquisition Mode; data is buffered through PX14400 RAM and acquired directly to the PCIe bus. Unlike RAM Acquisition, this mode supports the transfer of data as it is being actively acquired. This is the primary method used to conduct high-speed sustained data streaming to the host PC system for optional real-time signal processing and/or real-time signal recording of the acquired data. **Never set this mode directly; see Remarks.**
PX14MODE_RAM_READ_PCI (7)	PCIe Transfer Mode; used to transfer existing data that is residing in PX14400 RAM (no new data is acquired) to the host PC. This is the primary mode for obtaining data previously acquired with the RAM Acquisition mode. **Never set this mode directly; see Remarks.**

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given PX14400 handle, which will result in no hardware or driver access. If zero, the setting is obtained from the driver which may or may not result in an actual PX14400 device register read.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

On success, GetOperatingModePX14 returns the current operating mode setting.

Remarks

Operating mode transitions should always be coming from or going to Standby mode. Direct transitions between any two active operating modes are not allowed.

Putting the PX14400 into Standby Mode while a transfer or acquisition is armed or in progress will result in the transfer/acquisition being cancelled.

In RAM Acquisition Mode, data may not be transferred from the card. With this type of acquisition, all data is acquired and written into onboard RAM and only after the acquisition has completed may it be transferred to the host system via the PCIe Transfer operating mode.

In PCIe Buffered Acquisition Mode, free-run streaming acquisitions will acquire an infinite number of samples (or an infinite number of segments if using segmented trigger mode) to the PCIe bus until the operation is ended by putting the PX14400 back into Standby operating mode.

For the active operating modes the PX14400 library implements higher level data acquisition routines that automatically manage the operating mode as well as the active memory region, which is closely related to the

operating mode. These higher-level wrapper functions for the corresponding operating mode are identified in the following table.

Underlying Operating Mode	Wrapper Function(s)
PX14MODE_ACQ_RAM	AcquireToBoardRamPX14
PX14MODE_ACQ_PCI_BUF	BeginBufferedPciAcquisitionPX14
PX14MODE_RAM_READ_PCI	ReadSampleRamFastPX14 ReadSampleRamBufPX14

Related Functions

Refer to the Data Acquisition Routines section for further details.

3.6.20 | SetPostAdcClockDividerPX14 / GetPostAdcClockDividerPX14

Form

```
int SetPostAdcClockDividerPX14 (HPX14 hBrd, unsigned int div);
int GetPostAdcClockDividerPX14 (HPX14 hBrd, int bFromCache = 1);
```

Description

Set or get the post-ADC clock divider; effective clock division by dropping samples.

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *div*

The post-ADC clock divider value to set. Can be any of the following values:

PX14400 Library Constant	Effective Clock Divider
PX14POSTADCCLKDIV_01 (0)	1 (no division)
PX14POSTADCCLKDIV_02 (1)	2
PX14POSTADCCLKDIV_04 (2)	4
PX14POSTADCCLKDIV_08 (3)	8
PX14POSTADCCLKDIV_16 (4)	16
PX14POSTADCCLKDIV_32 (5)	32

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given PX14400 handle, which will result in no hardware or driver access. If zero, the setting is obtained from the driver which may or may not result in an actual PX14400 device register read.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

On success, GetPostAdcClockDividerPX14 returns the current post-ADC clock divider setting.

Remarks

For all clock sources (internal or external) the effective digitization rate can be reduced further via a hardware emulation of clock division that works by discarding samples after conversion with this function.

The post ADC emulated clock division setting may only be changed while the PX14400 is in the Standby operating mode.

Related Functions

SetAdcClockSourcePX14, SetInternalAdcClockRatePX14, SetExternalClockRatePX14

3.6.21 | SetPreTriggerSamplesPX14 / GetPreTriggerSamplesPX14

Form

int SetPreTriggerSamplesPX14 (HPX14 hBrd, unsigned int val);
int GetPreTriggerSamplesPX14 (HPX14 hBrd, int bFromCache = 1);

Description

Set or get pre-trigger sample count; count of samples to keep prior to trigger event.

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *val*

The number of pre-trigger samples to set. If this value is 0 then no pre-trigger samples are collected. If non-zero, this value can be any multiple of 4 in the range [0, 16360]. This value will be automatically aligned down or clipped to minimum/maximum as necessary.

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given PX14400 handle, which will result in no hardware or driver access. If zero, the setting is obtained from the driver which may or may not result in an actual PX14400 device register read.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

On success, GetPreTriggerSamplesPX14 returns the current pre-trigger sample count setting.

Remarks

Pre-trigger samples can be collected in both Post Trigger and Segmented triggering modes.

The pre-trigger sample count settings and trigger delay sample count settings are mutually exclusive. Using both at the same time will result in undefined behavior.

Related Functions

SetTriggerDelaySamplesPX14

3.6.22 | SetSampleCountPX14 / GetSampleCountPX14

Form

int SetSampleCountPX14 (HPX14 hBrd, unsigned int val);
int GetSampleCountPX14 (HPX14 hBrd, int bFromCache = 1);

Description

Set or get the total sample count; defines length of subsequent acquisitions/transfers.

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *val*

The total sample count. This can be any multiple of 32 in the range [32, 268435456] or can also be set to a special value of PX14_FREE_RUN (0) to specify a free-run (infinite) sample count. See Remarks.

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given PX14400 handle, which will result in no hardware or driver access. If zero, the setting is obtained from the driver which may or may not result in an actual PX14400 device register read.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

On success, GetSampleCountPX14 returns the current sample count setting.

Remarks

This function sets the total number of samples to be acquired or transferred. When this function is called, all subsequent acquisitions or data transfers will end when the specified number of samples is reached.

The sample count is independent of channel count. For a given sample count, the number of samples acquired/transferred per-channel is the sample count divided by the channel count. So for a total sample count of *T*:

- A single channel acquisition/transfer will result in an acquisition/transfer of *T* samples.
- A dual-channel acquisition will result in an acquisition of *T* samples, or *T*/2 samples per channel.

The sample count must be an integer multiple of 32 samples. The maximum allowed sample count is:

- 134,217,728 = for dual channel mode.
- 268,435,456 = for single channel mode.

Note in the case of using PCIe Buffered Acquisition mode, it is possible to acquire and infinite number of samples. This is done by specifying the special value PX14_FREE_RUN (0) for the sample count. To terminate a free-run acquisition, return the PX14400 to Standby mode.

The sample count setting may only be changed while the PX14400 is in the Standby operating mode.

Related Functions

SetStartSamplePX14

3.6.23 | SetSegmentSizePX14 / GetSegmentSizePX14

Form

int SetSegmentSizePX14 (HPX14 hBrd, unsigned int segSize);
int GetSegmentSizePX14 (HPX14 hBrd, int bFromCache = 1);

Description

Set or get the segment size for use with Segmented triggering mode.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] segSize
The segment size, in samples, to set. This can be any multiple of 4 in the range [4, 268435456].

[in] *bFromCache*
If non-zero, the setting will be read from the local device register cache associated with the given PX14400 handle, which will result in no hardware or driver access. If zero, the setting is obtained from the driver which may or may not result in an actual PX14400 device register read.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

On success, GetSegmentSizePX14 returns the current segment size setting.

Remarks

This setting has no effect when the PX14400 is not configured to use Segmented trigger mode. This function selects the amount of signal memory that will be assigned to each data segment when the board is acquiring in segmented trigger mode.

For dual channel acquisitions the segment size represents the total number of samples acquired for both channels combined. When set to single channel mode the segment size represents the total number of samples acquired for the single channel only.

As the segment size is independent of channel count; if *N* samples/segment per channel are desired, the segment size should be set to *N* * *channel count*. The segment size can be any integer multiple of 4 samples and the maximum allowed segment size that is:

- 134,217,728 = for dual channel mode.
- 268,435,456 = for single channel mode.

When the PX14400 board is set to the data acquisition mode and a trigger occurs, the amount of data specified by the *segSize* parameter will be acquired. Acquisition will then stop until another trigger is received, at which point another segment of *segSize* samples will be acquired. This series of events will continue until the PX14400 reaches the specified total samples count (via SetSampleCountPX14). To be meaningful, the segment size should always be set smaller than the active memory size.

The segment size setting may only be changed while the PX14400 is in the Standby operating mode.

Related Functions

SetTriggerModePX14

3.6.24 | SetStartSamplePX14 / GetStartSamplePX14

Form

int SetStartSamplePX14 (HPX14 hBrd, unsigned int val);
int GetStartSamplePX14 (HPX14 hBrd, int bFromCache = 1);

Description

Set or get the start sample; defines starting PX14400 RAM address of subsequent acquisitions/transfers.

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *val*

The new starting sample. This should be a multiple of 4096 samples. This value will be clipped to the largest valid starting sample, 268431360.

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given PX14400 handle, which will result in no hardware or driver access. If zero, the setting is obtained from the driver which may or may not result in an actual PX14400 device register read.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

On success, GetStartSamplePX14 returns the current starting sample setting.

Remarks

This function sets the PX14400 RAM address counter to the given sample number. This value determines the starting sample at which data acquisitions or data transfers begin when put into an active operating mode.

The starting sample used by most applications is zero. This means that data acquisitions and transfers will start at the beginning of the PX14400 RAM. If a value of zero is to be used, this function does not normally need to be called. The address counter is automatically set to zero when the PX14400 is placed in Standby mode.

In most applications, data is acquired to the PX14400 RAM and then transferred to the PC, always starting at sample 0. The reset feature removes the necessities of setting the starting sample counter to zero before the data is transferred.

The starting sample must be an integer multiple of 4,096 samples. The first sample in PX14400 RAM is sample 0. The maximum allowed starting sample is 268,431,360 (this is the maximum sample count for single channel mode minus 4,096 samples).

Dual-channel data is stored in RAM in a channel-interleaved format (CH 1, CH 2, CH 1, CH 2 …). When dealing with dual channel data, the PX14400 RAM address of CH 1 sample *N* would be 2**N* and the RAM address of CH 2 sample *N* would be 2**N* + 1.

The starting sample setting may only be changed while the PX14400 is in the Standby operating mode.

Related Functions

SetSampleCountPX14

3.6.25 | SetTimestampCounterModePX14 / GetTimestampCounterModePX14

Form

int SetTimestampCounterModePX14 (HPX14 hBrd, unsigned int val);
int GetTimestampCounterModePX14 (HPX14 hBrd, int bFromCache = 1);

Description

Set or get the timestamp counter mode; determines how timestamp counter increments.

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *val*

The timestamp counter mode to set. Currently defined timestamp modes are listed in the following table.

PX14400 Library Constant	Value	Interpretation
PX14TSCNTMODE_DEFAULT	0	Counter unconditionally increments during acquisition mode.
PX14TSCNTMODE_PAUSE_WHEN_ARMED	1	Counter only increments when digitizing; pauses when waiting for trigger.

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given PX14400 handle, which will result in no hardware or driver access. If zero, the setting is obtained from the driver which may or may not result in an actual PX14400 device register read.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

On success, GetTimestampCounterModePX14 returns the current timestamp counter mode.

Remarks

Upon transition from Standby operating mode to any acquisition operating mode, the timestamp counter will reset to 0 and increment once per acquisition clock cycle.

Related Functions

ResetTimestampCounterPX14, SetTimestampModePX14

3.6.26 | SetTimestampModePX14 / GetTimestampModePX14

Form

```
int SetTimestampModePX14 (HPX14 hBrd, unsigned int val);
int GetTimestampModePX14 (HPX14 hBrd, int bFromCache = 1);
```

Description

Set or get the timestamp mode; determines how timestamps are generated.

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *val*

The timestamp mode to set. Currently defined timestamp modes are listed in the following table on the next page.

PX14400 Library Constant	Value	Interpretation
PX14TSMODE_NO_TIMESTAMPS	0	No timestamps are generated. This is the default setting.
PX14TSMODE_SEGMENTS	1	A timestamp is generated at the start of each segment in a segmented acquisition. Intended to work with Segmented trigger mode.
PX14TSMODE_TS_ON_EXT_TRIGGER	2	A timestamp is generated for each trigger received on external trigger, regardless if that trigger started an acquisition or segment. This timestamp mode works with any trigger mode.
PX14TSMODE_TS_ON_DIGITAL_INPUT	3	A timestamp is generated for each rising edge of digital input. In order to use this mode, the digital IO mode must be PX14DIGIO_IN_TS_GEN and the digital IO port must be enabled. See SetDigitalIoModePX14 and SetDigitalIoEnablePX14.

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given PX14400 handle, which will result in no hardware or driver access. If zero, the setting is obtained from the driver which may or may not result in an actual PX14400 device register read.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

On success, GetTimestampModePX14 returns the current timestamp mode.

Remarks

The PX14400 can be configured to generate timestamps during data acquisitions.

Upon entering an acquisition operating mode, the PX14400 firmware will automatically reset the timestamp counter to zero. This counter will then be incremented by 1 with each tick of the PX14400 acquisition clock.

When the PX14400 determines that a timestamp should be generated (by the timestamp mode), the current timestamp counter value will be inserted into the PX14400 timestamp FIFO. The PX14400 timestamp FIFO can hold 2048 64-bit timestamps. The GetTimestampFifoDepthPX14 function can be used to programmatically determine the timestamp FIFO depth.

The ReadTimestampDataPX14 library function is used to read timestamp values from the timestamp FIFO. The library can also be configured to automatically read and save timestamp data in an external file during a recording operation or a ReadSampleRamFileFastPX14 / ReadSampleRamFileBufPX14 call. See PX14S_FILE_WRITE_PARAMS structure documentation for details.

Related Functions

ReadTimestampDataPX14, SetTimestampCounterModePX14

3.6.27 | SetTriggerDelaySamplesPX14 / GetTriggerDelaySamplesPX14

Form

int SetTriggerDelaySamplesPX14 (HPX14 hBrd, unsigned int val);
int GetTriggerDelaySamplesPX14 (HPX14 hBrd, int bFromCache = 1);

Description

Set or get trigger delay samples; count of samples to skip after trigger.

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *val*

The number of samples to ignore after a trigger. This value can be any multiple of 4 in the range [4, 2097148] or zero to not ignore any samples. This value will be automatically aligned down or clipped to minimum/maximum as necessary.

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given PX14400 handle, which will result in no hardware or driver access. If zero, the setting is obtained from the driver which may or may not result in an actual PX14400 device register read.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

On success, GetTriggerDelaySamplesPX14 returns the trigger delay count.

Remarks

When using trigger delay samples, the acquisition of data will be delayed from the trigger event. The length of the delay is the number of specified samples.

Trigger delay samples can be used in both Post Trigger and Segmented triggering modes.

The pre-trigger samples setting and delay trigger samples setting are mutually exclusive. Using both at the same time will result in undefined behavior.

Related Functions

SetPreTriggerSamplesPX14

3.6.28 | SetTriggerDirectionAPX14 / GetTriggerDirectionAPX14

Form

int SetTriggerDirectionAPX14 (HPX14 hBrd, unsigned int val);
int GetTriggerDirectionAPX14 (HPX14 hBrd, int bFromCache = 1);

Description

Set or get trigger A direction; defines the direction that defines a trigger.

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *val*

Specifies the trigger slope to use and may be one of the following:

PX14400 Library Constant	Interpretation
PX14TRIGDIR_POS (0)	Trigger occurs on positive going signal (Power-up default)
PX14TRIGDIR_NEG (1)	Trigger occurs on negative going signal

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given PX14400 handle, which will result in no hardware or driver access. If zero, the setting is obtained from the driver which may or may not result in an actual PX14400 device register read.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

On success, GetTriggerDirectionAPX14 returns the current trigger A direction setting.

Remarks

The PX14400 implements two independent triggers, A and B. Each trigger has its own level and direction. A trigger can be disabled by setting its trigger level to 0 and direction to negative.

Related Functions

SetTriggerLevelAPX14, SetTriggerSourcePX14

3.6.29 | SetTriggerDirectionBPX14 / GetTriggerDirectionBPX14

Form

int SetTriggerDirectionBPX14 (HPX14 hBrd, unsigned int val);
int GetTriggerDirectionBPX14 (HPX14 hBrd, int bFromCache = 1);

Description

Set or get trigger B direction; defines the direction that defines a trigger.

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *val*

Specifies the trigger slope to use and may be one of the following:

PX14400 Library Constant	Interpretation
PX14TRIGDIR_POS (0)	Trigger occurs on positive going signal (Power-up default)
PX14TRIGDIR_NEG (1)	Trigger occurs on negative going signal

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given PX14400 handle, which will result in no hardware or driver access. If zero, the setting is obtained from the driver which may or may not result in an actual PX14400 device register read.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

On success, GetTriggerDirectionBPX14 returns the current trigger B direction setting.

Remarks

The PX14400 implements two independent triggers, A and B. Each trigger has its own level and direction. A trigger can be disabled by setting its trigger level to 0 and direction to negative.

Related Functions

SetTriggerLevelBPX14, SetTriggerSourcePX14

3.6.30 | SetTriggerDirectionExtPX14 / GetTriggerDirectionExtPX14

Form

int SetTriggerDirectionExtPX14 (HPX14 hBrd, unsigned int val);
int GetTriggerDirectionExtPX14 (HPX14 hBrd, int bFromCache = 1);

Description

Set or get external trigger direction.

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *val*

Specifies the trigger slope to use for the external trigger and may be one of the following:

PX14400 Library Constant	Interpretation
PX14TRIGDIR_POS (0)	Trigger occurs on positive-going edge of TTL signal (Power-up default)
PX14TRIGDIR_NEG (1)	Trigger occurs on negative-going edge of TTL signal

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given PX14400 handle, which will result in no hardware or driver access. If zero, the setting is obtained from the driver which may or may not result in an actual PX14400 device register read.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

On success, GetTriggerDirectionExtPX14 returns the current trigger B direction setting.

Remarks

This setting is only relevant when the external trigger is selected as the trigger source.

Related Functions

SetTriggerSourcePX14

3.6.31 | SetTriggerHysteresisPX14 / GetTriggerHysteresisPX14

Form

int SetTriggerHysteresisPX14 (HPX14 hBrd, unsigned int val);
int GetTriggerHysteresisPX14 (HPX14 hBrd, int bFromCache = 1);

Description

Set or get trigger hysteresis; defines the absolute gap that the signal has to cross to qualify a trigger, regardless if it's rising or falling edge.

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *val*

Specifies the trigger hysteresis value in the range of [1, 31]:

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given PX14400 handle, which will result in no hardware or driver access. If zero, the setting is obtained from the driver which may or may not result in an actual PX14400 device register read.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

On success, GetTriggerHysteresisPX14 returns the current trigger hysteresis setting.

Remarks

This function sets the trigger sensitivity to avoid triggering on noise.

The trigger hysteresis setting may only be changed while the PX14400 is in the Standby operating mode.

Related Functions

SetTriggerSelectionPX14

3.6.32 | SetTriggerLevelAPX14 / GetTriggerLevelAPX14

Form

int SetTriggerLevelAPX14 (HPX14 hBrd, unsigned int val);
int GetTriggerLevelAPX14 (HPX14 hBrd, int bFromCache = 1);

Description

Set or get trigger A level; defines threshold for an internal trigger event.

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *val*

The digital ADC value that defines the trigger level threshold for trigger A. See Remarks.

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given PX14400 handle, which will result in no hardware or driver access. If zero, the setting is obtained from the driver which may or may not result in an actual PX14400 device register read.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

On success, GetTriggerLevelAPX14 returns the current trigger A level setting.

Remarks

The PX14400 implements two independent triggers, A and B. Each trigger has its own level and direction. A trigger can be disabled by setting its trigger level to 0 and direction to negative.

The trigger level defines the threshold at which a trigger event is detected. A trigger event is defined by the current ADC value crossing the value specified by this function in the direction specified by the Trigger A Direction (SetTriggerDirectionAPX14).

The trigger level selection settings may only be changed while the PX14400 is in the Standby operating mode.

Related Functions

SetTriggerDirectionAPX14, SetTriggerSourcePX14

3.6.33 | SetTriggerLevelBPX14 / GetTriggerLevelBPX14

Form

int SetTriggerLevelBPX14 (HPX14 hBrd, unsigned int val);
int GetTriggerLevelBPX14 (HPX14 hBrd, int bFromCache = 1);

Description

Set or get trigger B level; defines threshold for an internal trigger event.

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *val*

The digital ADC value that defines the trigger level threshold for trigger B. See Remarks.

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given PX14400 handle, which will result in no hardware or driver access. If zero, the setting is obtained from the driver which may or may not result in an actual PX14400 device register read.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

On success, GetTriggerLevelBPX14 returns the current trigger B level setting.

Remarks

The PX14400 implements two independent triggers, A and B. Each trigger has its own level and direction. A trigger can be disabled by setting its trigger level to 0 and direction to negative.

The trigger level defines the threshold at which a trigger event is detected. A trigger event is defined by the current ADC value crossing the value specified by this function in the direction specified by the Trigger B Direction (SetTriggerDirectionBPX14).

The trigger level selection settings may only be changed while the PX14400 is in the Standby operating mode.

Related Functions

SetTriggerDirectionBPX14, SetTriggerSourcePX14

3.6.34 | SetTriggerModePX14 / GetTriggerModePX14

Form

int SetTriggerModePX14 (HPX14 hBrd, unsigned int val);
int GetTriggerModePX14 (HPX14 hBrd, int bFromCache = 1);

Description

Set or get triggering mode; relates trigger events to how digitized data is saved.

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *val*

Specifies the trigger mode to set and may be one of the following:

PX14400 Library Constant	Interpretation
PX14TRIGMODE_POST_TRIGGER (0)	Trigger event starts a single data acquisition (Power-up default)
PX14TRIGMODE_SEGMENTED (1)	Each trigger event begins a new statically sized data acquisition

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given PX14400 handle, which will result in no hardware or driver access. If zero, the setting is obtained from the driver which may or may not result in an actual PX14400 device register read.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

On success, GetTriggerModePX14 returns the current trigger mode setting.

Remarks

Post trigger mode is used for acquiring continuous uninterrupted gap-free data after a single detected trigger event occurs. In post trigger mode, a single detected trigger event is used to begin acquiring data. The PX14400 will continue acquiring data, once per acquisition clock cycle, for all active input channels until the acquisition is completed.

Segmented mode is used to acquire discrete number of samples of continuous data for every detected trigger event that occurs. In segmented mode, a detected trigger event is used to begin acquiring a fixed specified number of samples, called a segment for all active input channels. The length of a segment is defined by the segment size setting. After a segment has been acquired, the PX14400 will stop acquisition and then rearm itself and wait for another trigger event to occur. This process will continually repeat until the specified total number of samples to be acquired over all data segments has been reached; at which point the operation is considered complete.

The trigger mode setting may only be changed while the PX14400 is in the Standby operating mode.

Related Functions

SetTriggerSourcePX14

3.6.35 | SetTriggerSelectionPX14 / GetTriggerSelectionPX14

Form

int SetTriggerSelectionPX14 (HPX14 hBrd, unsigned int val);
int GetTriggerSelectionPX14 (HPX14 hBrd, int bFromCache = 1);

Description

Set or get trigger selection; allows choice to use the trigger level B or hysteresis for the trigger process (in use with trigger level A). Also, allows to use the selected trigger in AND or OR modes.

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *val*

Specifies the trigger source to use and may be one of the following:

PX14400 Library Constant	Interpretation
PX14TRIGSEL_HYST_AND (0)	Use the hysteresis in AND mode
PX14TRIGSEL_LVLB_AND (1)	Use the trigger level B in AND mode
---	*Reserved*
PX14TRIGSEL_LVLB_OR (3)	Use the trigger level B in OR mode

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given PX14400 handle, which will result in no hardware or driver access. If zero, the setting is obtained from the driver which may or may not result in an actual PX14400 device register read.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

On success, GetTriggerSelectionPX14 returns the current trigger selection setting.

Remarks

The trigger selection setting may only be changed while the PX14400 is in the Standby operating mode.

Related Functions

SetTriggerHysteresisPX14

3.6.36 | SetTriggerSourcePX14 / GetTriggerSourcePX14

Form

int SetTriggerSourcePX14 (HPX14 hBrd, unsigned int val);
int GetTriggerSourcePX14 (HPX14 hBrd, int bFromCache = 1);

Description

Set or get trigger source; defines where trigger events originate.

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *val*

Specifies the trigger source to use and may be one of the following:

PX14400 Library Constant	Interpretation
PX14TRIGSRC_INT_CH1 (0)	Internal trigger, channel 1 (Power-up default)
PX14TRIGSRC_INT_CH2 (1)	Internal trigger, channel 2
PX14TRIGSRC_EXT (2)	External trigger

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given PX14400 handle, which will result in no hardware or driver access. If zero, the setting is obtained from the driver which may or may not result in an actual PX14400 device register read.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

On success, GetTriggerSourcePX14 returns the current trigger source setting.

Remarks

This function allows the user to select the PX14400 trigger source. If the trigger source is set to internal, the ADC input signal from either channel 1 or channel 2 becomes the trigger signal. If the trigger source is set to external, the signal from the external trigger input connector is used for the trigger source.

The trigger source selection can be changed while the PX14400 is in active acquisition modes.

Related Functions

SetTriggerDirectionAPX14, SetTriggerDirectionBPX14, SetTriggerLevelAPX14, SetTriggerLevelBPX14, SetTriggerModePX14

3.7 | Device Register State

The functions in this section involve manipulation of PX14400 device registers.

3.7.1 | CopyHardwareSettingsPX14

Form

```
int CopyHardwareSettingsPX14 (HPX14 hBrdDst, HPX14 hBrdSrc);
```

Description

Copy hardware settings from another PX14400 device.

Parameters

[in] *hBrdDst*
A handle to the PX14400 device to which the copied hardware settings will be applied.

[in] *hBrdSrc*
A handle to the PX14400 device from which the hardware settings are copied from.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

Calling this function will copy all hardware settings of the PX14400 associated with the source device to the PX14400 associated with the destination device. The function will use the hardware settings as they are defined in the handle-specific software register cache.

Operating mode is not copied and the destination PX14400 is placed into Standby operating mode before any settings are applied.

Related Functions

RewriteHardwareSettingsPX14, RefreshLocalRegisterCachePX14

3.7.2 | LoadSettingsFromBufferXmlPX14

Form

```
int LoadSettingsFromBufferXmlPX14 (HPX14 hBrd, unsigned int flags, const TCHAR* bufp);
```

Description

Load hardware settings from an XML buffer.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *flags*
Flags that affect how settings are loaded:

PX14400 Library Constant	Value	Interpretation
PX14XMLSET_NO_PRELOAD_DEFAULTS	0x00000004	Do not set default hardware settings prior to loading settings.

[in] *bufp*
A pointer to a NULL-terminated string containing the XML data containing the settings data. The SaveSettingsToBufferXmlPX14 function is used to generate this data.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

This function is used to apply hardware settings saved by a previous call to SaveSettingsToBufferXmlPX14.

Related Functions

SaveSettingsToBufferXmlPX14

3.7.3 | LoadSettingsFromFileXmlPX14

Form

```
int LoadSettingsFromFileXmlPX14 (HPX14 hBrd, unsigned int flags, const TCHAR* bufp);
```

Description

Load hardware settings from an XML file.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *flags*
Flags that affect how settings are loaded:

PX14400 Library Constant	Value	Interpretation
PX14XMLSET_NO_PRELOAD_DEFAULTS	0x00000004	Do not set default hardware settings prior to loading settings.

[in] *bufp*
A pointer to a NULL-terminated string containing the pathname of the XML file containing the settings data. The SaveSettingsToFileXmlPX14 function is used to generate this data.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

This function is used to apply hardware settings saved by a previous call to SaveSettingsToFileXmlPX14.

Related Functions

SaveSettingsToFileXmlPX14

3.7.4 | ReadAllDeviceRegistersPX14

Form

```
int ReadAllDeviceRegistersPX14 (HPX14 hBrd);
```

Description

Read all device registers from hardware; updates all register caches.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

This function is used to refresh all internal device register caches with current hardware register values.

Related Functions

RewriteHardwareSettingsPX14, RefreshLocalRegisterCachePX14

3.7.5 | RefreshLocalRegisterCachePX14

Form

```
int RefreshLocalRegisterCachePX14 (HPX14 hBrd, int bFromHardware = 0);
```

Description

Refresh local device register cache from driver's cache; no hardware read.

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *bFromHardware*

If this parameter is non-zero then register content will be updated from the PX14400 hardware. If zero, the driver's local register cache will be consulted. Since all device writes go through the driver, the driver's cache will contain an updated cache of hardware register content. (Status registers are the exception to this.)

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

Call this function to update the local PX14400 device register cache associated with the given PX14400 device handle. The register values are obtained from the kernel-level register cache maintained by the PX14400 device driver. Calling this function has no effect on any underlying PX14400 hardware.

This function is automatically invoked by the ConnectToDevicePX14 function as part of the device connection procedure.

Related Functions

RewriteHardwareSettingsPX14

3.7.6 | RewriteHardwareSettingsPX14

Form

```
int RewriteHardwareSettingsPX14 (HPX14 hBrd);
```

Description

Bring hardware settings up to date with current cache settings.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

Call this function to rewrite all hardware settings with the values contained in the local register cache that is associated with the given PX14400 handle.

Calling this function will place the PX14400 in the Standby operating mode regardless of the operating mode in the register cache.

Related Functions

RefreshLocalRegisterCachePX14

3.7.7 | SaveSettingsToBufferXmlPX14

Form

```
int SaveSettingsToBufferXmlPX14 (HPX14 hBrd, unsigned int flags, TCHAR* bufp, int* buflenp);
```

Description

Save board settings to XML format in a library allocated-buffer.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *flags*

Flags that affect how settings are saved:

PX14400 Library Constant	Value	Interpretation
PX14XMLSET_NODE_ONLY	0x00000001	Serialize to a node only; do not include XML header info.
PX14XMLSET_FORMAT_OUTPUT	0x00000002	Pretty-print XML output; add newlines and indentation. This will generate nicer, human-readable output.

[out] *bufp*

A pointer to a TCHAR* variable that will receive the address of a library-allocated buffer containing the XML data. It is the caller's responsibility to free this memory with FreeMemoryPX14 function.

[out] *buflenp*

A pointer to an integer variable that will receive the length of the library-allocated buffer, in characters.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

This function is used to save hardware settings to an XML buffer. These settings can later be reloaded by calling the LoadSettingsFromBufferXmlPX14 function.

Related Functions

LoadSettingsFromBufferXmlPX14, SaveSettingsToFileXmlPX14

3.7.8 | SaveSettingsToFileXmlPX14

Form

```
int SaveSettingsToFileXmlPX14 (HPX14 hBrd, unsigned int flags, TCHAR* pathnamep, TCHAR* encoding = "UTF-8");
```

Description

Save board settings to an XML file.

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *flags*

Flags that affect how settings are saved:

PX14400 Library Constant	Value	Interpretation
PX14XMLSET_NODE_ONLY	0x00000001	Serialize to a node only; do not include XML header info.
PX14XMLSET_FORMAT_OUTPUT	0x00000002	Pretty-print XML output; add newlines and indentation. This will generate nicer, human-readable output.

[in] *pathnamep*

A pointer to a NULL-terminated string containing the pathname of the XML file.

[in] *encodingp*

A pointer to a NULL-terminated string containing the encoding to use for the generated XML file. This can be any encoding supported by the iconv library: ASCII, UTF-8, UTF16, char, wchar_t, etc.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

This function is used to save hardware settings to an XML file. These settings can later be reloaded by calling the LoadSettingsFromFileXmlPX14 function.

Related Functions

LoadSettingsFromFileXmlPX14

3.7.9 | SetPowerupDefaultsPX14

Form

```
int SetPowerupDefaultsPX14 (HPX14 hBrd);
```

Description

Restores all PX14400 settings to power-up default values.

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

The PX14400 is put into Standby mode prior to applying default values.

Calling this function will reset all hardware settings to their default power up values. For most all PX14400 settings, these equate to the '0' values. A few exceptions to this:

- Trigger levels are set to midscale (PX14_TRIGGER_LEVEL_MIDSCALE).
- Internal and external acquisition rates are set to 400MHz.
- The PX14400 is kept in Standby mode.

3.8 | DMA Buffer Management

DMA buffers are specially allocated regions of memory that can be used in a DMA transfer. A DMA transfer is a very fast, hardware-controlled data transfer mechanism that is used to move data between the PX14400 and host system memory. Since DMA buffers are mapped into the calling process' address space there is no intermediate buffering of the data, the data is immediately available to the application program.

Once a DMA buffer has been allocated and mapped into a process' address space it is no different from normal memory from a user's point of view. Users can read and write to it just like they would to any array. They can use the memory in function calls, etc.

The PX14400 library supports three different methods of DMA buffer allocation:

DMA Buffer Method	Description
Dynamic	DMA buffers are allocated dynamically as needed.
Boot-Time	DMA buffers are allocated at boot-time when the driver is loaded. These buffers may then be checked out by applications.
Chained	A number of individual DMA buffers are dynamically allocated and aggregated into an array of buffer address/length descriptors.

Dynamic DMA Buffer Allocation

Dynamic DMA buffer allocation is the most flexible DMA buffer allocation methodology. The other allocation methodologies are effectively derivatives of this method. With this method, DMA buffers are allocated and freed as the application needs them. There are no explicit limits to the amount of DMA buffers that may be allocated. There is no explicit upper bound to the maximum DMA buffer size.

The main issue with this method is that the longer the system is up and running, the more fragmented physical host memory becomes and the harder it gets to allocate contiguous memory space for DMA buffers. This can become particularly problematic in systems that have relatively small amounts of physical memory installed; typically 4 GB or less. Most modern operating systems do nothing to mitigate physical memory fragmentation because it is not really an issue in most application circumstances.

The following functions are associated with dynamic DMA buffer allocation and are detailed further in the subsequent subsections:

Library Function	Description
AllocateDmaBufferPX14	Allocate a single DMA buffer of a given size.
FreeDmaBufferPX14	Free a DMA buffer allocated by AllocateDmaBufferPX14.
EnsureUtilityDmaBufferPX14	Allocate (if necessary) a “utility” DMA buffer associated with the given device handle.
GetUtilityDmaBufferPX14	Get the current address and/or size of a “utility” DMA buffer associated with the given device handle.
FreeUtilityDmaBufferPX14	Explicitly free a utility DMA buffer.

Boot-Time DMA Buffer Allocation

The use of boot buffers is particularly effective in systems that have relatively small amounts of physical memory installed; typically 4 GB or less. Boot-time DMA buffer allocation works by having the driver allocate a number of DMA buffers when it loads, which is effectively at boot time.

At boot time it is much easier for the system to allocate the physically contiguous memory required for DMA buffers. These buffers are referred to as boot-buffers. Boot-buffers are held for as long as the PX14400 driver is running. User applications may then “check out” the buffers for their own usage and then “check in” the buffers when they’re no longer needed. There is no real difference between a DMA buffer allocated at boot-time and a dynamically allocated DMA buffer (AllocateDmaBufferPX14) aside from when they are allocated.

The PX14400 software supports a maximum of 4 boot-time DMA buffers. Boot-buffer size configuration data is stored in the configuration EEPROM so the settings are persistent across power cycles, or if the card is moved to another system. Library routines exist for querying or modifying the boot-buffer configuration settings.

The following functions are associated with boot-time DMA buffer allocation and are detailed further in the subsequent subsections:

Library Function	Description
BootBufGetMaxCountPX14	Get maximum number of boot buffers available.
BootBufCheckOutPX14	Check out boot buffer for usage.
BootBufCheckInPX14	Check in boot buffer when no longer needed.
BootBufQueryPX14	Query boot buffer information.
BootBufCfgSetPX14	Set the requested boot buffer size.
BootBufCfgGetPX14	Get the requested boot buffer size.
BootBufReallocNowPX14	Ask driver to allocate (if necessary) boot buffers now.

DMA Buffer Chain Allocation

DMA buffer chains are a number of discrete, dynamically allocated DMA buffers that are aggregated into an array of DMA buffer address/length pairs. This allocation methodology allows the user to allocate a large amount of DMA memory in aggregate.

Very few, if any, PX14400 library transfer functions work directly with a DMA buffer chain. If using a DMA buffer chain in a custom PX14400 application, the application code will be responsible for walking the DMA buffer chain and transferring to the component buffer described by each node.

The following functions are associated with DMA buffer chain allocation and are detailed further in the subsequent subsections:

Library Function	Description
AllocateDmaBufferChainPX14	Allocate a DMA buffer chain.
FreeDmaBufferChainPX14	Free a DMA buffer chain.
GetUtilityDmaBufferChainPX14	Obtain address of the utility DMA buffer chain.

3.8.1 | AllocateDmaBufferPX14

Form

```
int AllocateDmaBufferPX14 (HPX14 hBrd, unsigned int samples, px14_sample_t** bufpp);
```

Description

Allocate a DMA buffer for use with DMA transfers.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *samples*
The number of samples to allocate for the buffer. There is no explicit upper bound to the size of a DMA buffer, it is entirely dependent on the available host memory resources and the host operating system.

[out] *bufpp*
A pointer to a DMA buffer pointer that will receive the virtual address of the DMA buffer. This buffer is fully mapped into the calling process' address space. That is, it can be treated just like normal memory.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

This function is used to allocate physically contiguous, non-paged space in host PC memory for DMA transfers. The allocated buffer (or any region therein) may then be used by functions that require a DMA buffer. Note that the only way to get data from the board directly to the host PC is via a DMA transfer. A DMA buffer must be freed via the FreeDmaBufferPX14 function when it is no longer needed.

The DMA buffer may only be used with the PX14400 device that it was allocated for. A DMA buffer can be used with another PX14400 device handle (HPX14) of the same process as long as both device handles refer to the same underlying PX14400 device.

All ‘Fast’ PX14400 library functions (e.g. ReadSampleRamFastPX14) require a DMA buffer as a parameter. These functions are faster than their counterparts because the driver can perform the DMA transfer directly to the DMA buffer which is mapped in the user process. The ‘normal’ functions work by using a smaller driver-allocated DMA buffer to perform the data transfers which is slightly less efficient.

The maximum size of a DMA buffer is system-dependent.

On most platforms, the actual amount of memory allocated will usually be rounded up to the system page boundary, typically 4096 bytes.

Microsoft Windows: The virtual address of any allocated DMA buffer will be on a 64KB boundary. This allows DMA buffers to be used for non-buffered file IO routines. (See FILE_FLAG_NO_BUFFERING documentation for the Win32 CreateFile function.)

The PX14400 driver will ensure that all un-freed DMA buffers for a given process will be freed when that process’ last handle is closed. That is to say, like normal, heap-allocated memory, the underlying PX14400 software will ensure that all DMA buffers are properly cleaned up when a process exits, gracefully or not.

Related Functions

FreeDmaBufferPX14, ReadSampleRamFastPX14

3.8.2 | AllocateDmaBufferChainPX14

Form

```
int AllocateDmaBufferChainPX14 (HPX14 hBrd, unsigned int total_samples, PX14S_BUFNODE** nodepp, int flags = 0, int* buf_countp = NULL);
```

Description

Allocate an aggregated chain of DMA buffers of a total size.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *total_samples*
The total number of samples to allocate over all buffers in the buffer chain.

[out] *nodepp*
A pointer to a PX14S_BUFNODE* variable that will receive the address of the DMA buffer chain. See remarks.

[in] *flags*

A set of flags that control function behavior. Defined flags are as follows:

PX14400 Library Constant	Value	Interpretation
PX14DMACHAINF_LESS_IS_OKAY	0x00000001	If the function fails to allocate the requested number of samples, it will return a DMA chain containing the buffers it was able to allocate. If this flag is set, it's up to the caller to check the buffer chain to see how large a chain was allocated.
PX14DMACHAINF_UTILITY_CHAIN	0x00010000	The function will allocate a utility DMA buffer chain. Each PX14400 device handle can have one utility DMA buffer chain associated with it. Multiple DMA buffer chains may be allocated, but only one can be designated as the handle's utility DMA buffer chain. The utility DMA buffer chain can be used by certain library operations such as recording sessions. Unlike normal DMA buffer chains, the utility DMA buffer chain is automatically freed by the library when the device handle is closed. If a utility DMA buffer chain has already been allocated and is not less than the requested size then no allocation will take place and the previously allocated DMA buffer chain will be returned. If a utility DMA buffer chain has already been allocated and is less than the requested size then it will be freed and a new buffer chain will be allocated. In this case, previous data is not copied. The current utility DMA buffer chain can be obtained by calling GetUtilityDmaBufferChainPX14.

[out] *buf_countp*

A pointer to an integer variable that will receive the total number of buffers allocated for the DMA buffer chain. This count can also be used to determine the size of the DMA buffer chain array. This parameter may be NULL. See remarks.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

This function is used to allocate a DMA buffer chain. A DMA buffer chain is an array of DMA buffer address/length pairs that can be used by PX14400 software when additional buffering of acquisition data is required, such as high data throughput recordings.

When a DMA buffer chain is allocated, the library allocates an extra buffer descriptor at the end to indicate end-of-array. This descriptor is identified by the bufp value being NULL. For this special case node, the buf_samples value is the sum total sample count of all buffers in the DMA buffer chain. This end-of-array

marker allows software to pass around a DMA buffer chain object reference without having to include an array length parameter.

The library uses the following allocation scheme:

1.) Initial DMA buffer allocation size is 2 MiS (2 * 1024 * 1024 samples)

2.) The library iteratively allocates DMA buffers until the requested amount of memory has been allocated.

3.) If a DMA buffer allocation fails, the DMA buffer allocation size is halved and allocation continues.

4.) The minimum single DMA buffer allocation size is 512 KiS (512 * 1024 samples).

5.) The last buffer in the DMA buffer chain may be smaller than the current DMA buffer allocation size so that the function does not allocate more than what was requested.

Very few, if any, PX14400 library transfer functions work directly with a DMA buffer chain. If using a DMA buffer chain in a custom PX14400 application, the application code will be responsible for walking the DMA buffer chain and transferring to the component buffer described by each node.

The PX14400 Recording Session can be configured to use the utility DMA buffer chain for data buffering by specifying the PX14RECSESF_DEEP_BUFFERING flag when creating the recording session.

Example

```
// For a complete example see the PciAcqBufChainPX14 example application that is installed in the Examples
// directory of the main PX14400 installation directory.

// Pseudo-example:
PX14S_BUFNODE *buf_chainp, *node_curp;
int res;

// Allocate DMA buffer chain
res = AllocateDmaBufferChainPX14(hBrd, 512 * 1048576, &buf_chainp, 0, NULL);
if (SIG_SUCCESS != res) { ERROR; }

// ...

// Use buffer chain
for (node_curp=buf_chainp; node_curp->bufp; node_curp++)
{
        res = GetPciAcquisitionDataFastPX14(hBrd, node_curp->buf_samples, node_curp->bufp, PX14_FALSE);
        if (SIG_SUCCESS != res) { ERROR; }
}

// ...

// Free DMA buffer chain when done
FreeDmaBufferChainPX14(hBrd, buf_chainp);
```

Related Functions

FreeDmaBufferChainPX14, AllocateDmaBufferPX14

3.8.3 | BootBufCfgSetPX14 / BootBufCfgGetPX14

Form

int BootBufCfgSetPX14 (HPX14 hBrd, unsigned short buf_idx, unsigned int buf_samples);
int BootBufCfgGetPX14 (HPX14 hBrd, unsigned short buf_idx, unsigned int* pbuf_samples);

Description

Set or get boot buffer configuration; determines requested size of boot buffers.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *buf_idx*
The 0-based index of the boot-buffer. The maximum number of boot-buffers can be programmatically obtained by calling BootBufGetMaxCountPX14.

[in] *nodepp*
A pointer to a PX14S_BUFNODE* variable that will receive the address of the DMA buffer chain. See remarks.

[in] *buf_samples*
The new requested size of the boot buffer in samples.

[out] *pbuf_samples*
A pointer to an unsigned integer variable that will receive the current requested boot-buffer size. This is not necessarily the size of the currently allocated boot-buffer.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

This information is stored in the given board's configuration EEPROM so the setting will stick until explicitly changed. The configuration EEPROM will only be written if the current configuration is different.

Note that changing this configuration will have no effect on currently allocated boot buffers. As part of the device driver's loading, it will consult the boot buffer configuration and allocate accordingly. Software can force the driver to reallocate (if necessary) boot buffers by calling BootBufReallocNowPX14.

Related Functions

BootBufReallocNowPX14

3.8.4 | BootBufCheckInPX14

Form

```
int BootBufCheckInPX14 (HPX14 hBrd, px14_sample_t* dma_bufp);
```

Description

Check in boot buffer when no longer needed.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *dma_bufp*
The address of the boot-time allocated DMA buffer to check in.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

This function will return SIG_PX14_BUSY if the specified buffer is currently in use for a DMA transfer operation. A boot buffer cannot be checked in while in use. The current DMA transfer can be cancelled by putting the PX14400 into Standby mode and then the buffer can be checked in.

Related Functions

BootBufCheckOutPX14

3.8.5 | BootBufCheckOutPX14

Form

```
int BootBufCheckOutPX14 (HPX14 hBrd, unsigned short buf_idx, px14_sample_t** dma_bufpp, unsigned int* pbuf_samples);
```

Description

Check out boot buffer with given index for usage.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *buf_idx*
The 0-based index of the buffer to check out.

[out] *dma_bufpp*
A pointer to a px14_sample_t* variable that will receive the address of the DMA buffer. This buffer can be used by any library function that takes a DMA buffer parameter.

[out] *pbuf_samples*
A pointer to an unsigned integer variable that will receive the size of the checked-out buffer in px14_sample_t elements. This parameter may be NULL.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

This function will check out the boot-buffer with the given index. If the buffer is already checked out by another handle then the function will return error code SIG_PX14_BUFFER_CHECKED_OUT (-596). If no buffer has been allocated for the given index then the function will return SIG_PX14_BUFFER_NOT_ALLOCATED (-597).

When a boot-buffer is checked out, it is checked out to a particular board handle (HPX14). When this handle closes, all boot-buffers checked out to it are automatically checked-in. This includes explicit disconnection from the handle (DisconnectFromDevicePX14) or implicit disconnection (process cleanup).

A checked out buffer is only valid in the process that checked the buffer out. A boot-buffer may not be checked out to more than one process at a time. It is okay for one process to check out some boot buffers (say indices 0 and 1) and another process to check out other boot buffers (say 2 and 3).

If there are multiple custom applications on the same system that use boot buffers, it is up them to arbitrate access to the boot buffers. The PX14400 library checks out buffers on a first-come first-served basis. Signatec software, such as the PX14400 Scope Application, will only use boot buffers if explicitly enabled by the user.

Related Functions

BootBufCheckInPX14, BootBufGetMaxCountPX14, BootBufQueryPX14

3.8.6 | BootBufGetMaxCountPX14

Form

```
int BootBufGetMaxCountPX14 (HPX14 hBrd);
```

Description

Obtain maximum boot buffer count for given device.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

Return Value

Returns the maximum number of boot-time DMA buffers that are supported for the given device. On error, one of the library error codes (which are all negative) will be returned.

Remarks

The maximum boot-buffer count will never be less than 4.

Related Functions

BootBufQueryPX14

3.8.7 | BootBufQueryPX14

Form

```
int BootBufQueryPX14 (HPX14 hBrd, unsigned short buf_idx, int* pchecked_out, unsigned int* pbuf_samples);
```

Description

Query boot buffer status.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *buf_idx*
The 0-based index of the buffer to check out.

[out] *pchecked_out*
A pointer to an integer that will receive the check-out status of the buffer with the given index. If the buffer is currently checked out then the check-out status will be a non-zero value. If the buffer is not checked out then the check-out status will be zero.

[out] *pbuf_samples*
A pointer to an unsigned integer variable that will receive the size of the buffer with the given index in px14_sample_t elements. Zero will be used if no buffer has been allocated.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

The maximum boot-buffer count will never be less than 4.

Related Functions

BootBufCheckOutPX14, BootBufCfgGetPX14

3.8.8 | BootBufReallocNowPX14

Form

```
int BootBufReallocNowPX14 (HPX14 hBrd);
```

Description

Request driver to reallocate boot buffers now.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

Calling this function will have the driver attempt to allocate boot buffers now. If a particular boot buffer is already large enough it will not be reallocated. If a boot buffer is allocated and the new size is 0, the boot buffer will be freed.

This function is intended as an attempt to get boot buffers after making a configuration change without having to reboot the system.

This function will return SIG_PX14_BUSY (-527) if any boot buffers are checked out. This function may only be called while all boot buffers are checked in.

If any of the boot buffers could not be allocated, then the function will return SIG_DMABUFALLOCFAIL (-6). BootBufQueryPX14 can be used to query sizes and check-out status of all buffers.

Related Functions

BootBufCfgSetPX14, BootBufQueryPX14

3.8.9 | EnsureUtilityDmaBufferPX14

Form

int EnsureUtilityDmaBufferPX14 (HPX14 hBrd, unsigned int sample_count);
int EnsureUtilityDmaBuffer2PX14 (HPX14 hBrd, unsigned int sample_count);

Description

Ensures that the application-specific utility DMA buffer #1 or #2 is of the given size.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *sample_count*
The desired minimum utility DMA buffer size in samples. See remarks.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

Each PX14400 device handle can have up to two application specific utility DMA buffers associated with the device handle. These utility DMA buffers are not used by the PX14400 software and are reserved for application specific use. When the PX14400 device handle is obtained, no utility DMA buffers are allocated. Utility DMA buffers are not copied for duplicated handles (DuplicateHandlePX14).

This function is used to ensure that the specified application-specific utility DMA buffer is at least as large as the given sample count. In the event that the current buffer is too small, it will be freed and a new buffer will be allocated. Previous data is not copied.

The address of the DMA buffer is obtained by calling GetUtilityDmaBufferPX14/GetUtilityDmaBuffer2PX14.

The buffer may be freed by calling FreeUtilityDmaBufferPX14/FreeUtilityDmaBuffer2PX14. Utility buffers are also automatically freed when the associated device handle is closed (DisconnectFromDevicePX14).

Warning: Do not free utility DMA buffers with the FreeDmaBufferPX4 function. Doing so will free the DMA buffer but not invalidate the internal reference to the buffer held by the library implementation which may result in an access violation when the device handle is closed.

Related Functions

GetUtilityDmaBufferPX14, FreeUtilityDmaBufferPX14

3.8.10 | FreeDmaBufferPX14

Form

```
int FreeDmaBufferPX14 (HPX14 hBrd, px14_sample_t* bufp);
```

Description

Free a DMA buffer previously allocated by the AllocateDmaBufferPX14 function.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *bufp*
The address of the DMA buffer to free.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

The PX14400 driver will automatically free any DMA buffers that have not been freed by a process when it exits.

Related Functions

AllocateDmaBufferPX14

3.8.11 | FreeDmaBufferChainPX14

Form

```
int FreeDmaBufferPX14 (HPX14 hBrd, PX14S_BUFNODE* nodep);
```

Description

Free a DMA buffer chain previously allocated by the AllocateDmaBufferChainPX14 function.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *nodep*
The address of the DMA buffer chain to free.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Related Functions

AllocateDmaBufferChainPX14

3.8.12 | FreeUtilityDmaBufferPX14

Form

int FreeUtilityDmaBufferPX14 (HPX14 hBrd);
int FreeUtilityDmaBuffer2PX14 (HPX14 hBrd);

Description

Free utility DMA buffer #1 or #2 previously allocated by the EnsureUtilityDmaBufferPX14 function.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

These functions are used to free utility DMA buffer #1 or #2. The library will also automatically free utility DMA buffers when the associated handle is closed.

Utility DMA buffers are allocated by calling EnsureUtilityDmaBufferPX14 or EnsureUtilityDmaBuffer2PX14.

Warning: Do not free utility DMA buffers with the FreeDmaBufferPX14 function. Doing so will free the DMA buffer but not invalidate the internal reference to the buffer held by the library implementation which may result in an access violation when the device handle is closed.

Related Functions

EnsureUtilityDmaBufferPX14, GetUtilityDmaBufferPX14

3.8.13 | GetUtilityDmaBufferPX14

Form

int GetUtilityDmaBufferPX14 (HPX14 hBrd, px14_sample_t** bufpp, unsigned int* buf_samplesp);
int GetUtilityDmaBuffer2PX14 (HPX14 hBrd, px14_sample_t** bufpp, unsigned int* buf_samplesp);

Description

Get address and/or size of utility DMA buffer #1 or #2.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[out] *bufpp*
A pointer to a px14_sample_t pointer that will receive the address of the specified utility DMA buffer, or NULL if no utility DMA buffer has been allocated. This parameter may be NULL.

[out] *buf_samplesp*
A pointer to an unsigned integer variable that will receive the length of the specified utility DMA buffer in samples, or zero if no utility DMA buffer has been allocated. This parameter may be NULL.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

Utility DMA buffers are allocated by calling EnsureUtilityDmaBufferPX14 or EnsureUtilityDmaBuffer2PX14.

Related Functions

EnsureUtilityDmaBufferPX14, FreeUtilityDmaBufferPX14

3.8.14 | GetUtilityDmaBufferChainPX14

Form

int GetUtilityDmaBufferChainPX14 (HPX14 hBrd, PX14S_BUFNODE** nodepp, unsigned int* total_samplesp);

Description

Get address and/or size of utility DMA buffer chain.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[out] *nodepp*

A pointer to a PX14S_BUFNODE pointer that will receive the address of the specified utility DMA buffer chain, or NULL if no utility DMA buffer chain has been allocated. This parameter may be NULL.

[out] *total_samplesp*

A pointer to an unsigned integer variable that will receive the aggregate length of the specified utility DMA buffer chain in samples, or zero if no utility DMA buffer has been allocated. This parameter may be NULL.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

Utility DMA buffer chains are allocated by calling AllocateDmaBufferChainPX14 with the PX14DMACHAINF_UTILITY_CHAIN flag set in the flags parameter.

Related Functions

AllocateDmaBufferChainPX14

3.9 | Data Acquisition Routines

The functions in this section are used to perform data acquisitions.

3.9.1 | AcquireToBoardRamPX14

Form

```
int AcquireToBoardRamPX14 (HPX14 hBrd, unsigned int samp_start, unsigned int samp_count, unsigned int timeout_ms = 0, int bAsynchronous = 0);
```

Description

Acquire data to PX14400 onboard RAM.

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *samp_start*

The PX14400 RAM sample at which to put the first acquired sample. This parameter has the same restrictions defined in the SetStartSamplePX14 function.

[in] *samp_count*

The total number of samples to acquire. This is independent of active channel count and segment size. This parameter has the same restrictions defined in the SetSampleCountPX14 function.

[in] *timeout_ms*

An optional timeout value in milliseconds. If this value is 0 then the function will not timeout. This parameter is ignored if parameter *bAsynchronous* is nonzero.

[in] *bAsynchronous*

If this parameter is nonzero then an asynchronous data transfer will be performed; see remarks.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

This function will return SIG_CANCELLED (-10) if the RAM acquisition is cancelled before it finishes.

Remarks

This function will perform a synchronous RAM acquisition using current data acquisition settings to the specified region of PX14400 onboard RAM. This function will not return until the acquisition has completed or the optional timeout has elapsed. The invoking thread will sleep while the acquisition takes place. The Samples Complete interrupt will be used for end of acquisition notification.

Once the data acquisition has completed, data may be transferred from the PX14400 sample RAM to the host PC by calling ReadSampleRamFastPX14 or ReadSampleRamBufPX14.

An acquisition may also be cancelled by putting the PX14400 into Standby operating mode from a secondary thread or process. A separate thread is required because the thread that invoked this function will be suspended waiting for the acquisition to finish (or timeout).

This function handles all necessary operating mode changes as well as setting up the active memory region.

This function can also start an asynchronous acquisition. By default, this function will not return until the data acquisition completes. If the *bAsynchronous* parameter is nonzero, then the function will start the data acquisition and return immediately without waiting for it to finish allowing the calling thread to continue working. The caller can use the IsAcquisitionInProgressPX14 function to determine if the acquisition is still in progress. To wait (sleep) for the acquisition to complete, call the WaitForAcquisitionCompletePX14 function.

Related Functions

IsAcquisitionInProgressPX14, WaitForAcquisitionCompletePX14

3.9.2 | BeginBufferedPciAcquisitionPX14

Form

```
int BeginBufferedPciAcquisitionPX14 (HPX14 hBrd , unsigned int samp_count = PX14_FREE_RUN);
```

Description

Begin a PX14400 RAM buffered PCIe acquisition; also referred to as streaming mode.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *samp_count*
The total number of samples to acquire. For most applications the default value will be used to indicate an infinite recording length. When the desired amount of data has been obtained the recording is then stopped.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

Use this function to begin a PX14400 RAM buffered PCIe acquisition.

In this operating mode, data is buffered through PX14400 RAM and acquired directly to the PCIe bus. Unlike RAM Acquisition, this mode supports the transfer of data as it is being actively acquired. This is the primary method used to conduct high-speed sustained data streaming to the host PC system for optional real-time signal processing and/or real-time signal recording of the acquired data.

In this mode, the onboard 512 MB RAM bank is operated as a FIFO to buffer the acquired data as it is consumed via DMA transfers over the PCIe bus to the host PC system.

Once the acquisition has been started, the GetPciAcquisitionDataFastPX14 function is repeatedly called to transfer acquisition data to the host PC. When the desired amount of data has been obtained, the acquisition is ended by calling the EndBufferedPciAcquisitionPX14 function.

This function takes care of setting up the active memory region and all mode changes.

Refer to the PCIe Buffered Acquisition Mode (Streaming) section for further details on maximum sustained data streaming rate operations.

Related Functions

EndBufferedPciAcquisitionPX14, GetPciAcquisitionDataFastPX14

3.9.3 | EndBufferedPciAcquisitionPX14

Form

```
int EndBufferedPciAcquisitionPX14 (HPX14 hBrd);
```

Description

End a PX14400 RAM buffered PCIe acquisition.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

This function is used to end the current PX14400 RAM buffered PCIe acquisition started by a previous call to BeginBufferedPciAcquisitionPX14.

Related Functions

BeginBufferedPciAcquisitionPX14, GetPciAcquisitionDataFastPX14

3.9.4 | IsAcquisitionInProgressPX14

Form

```
int IsAcquisitionInProgressPX14 (HPX14 hBrd);
```

Description

Determine if an asynchronous RAM acquisition is currently in progress.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

Return Value

Returns 1 if an acquisition is currently in progress, 0 if an acquisition is not in progress, or one of the library error codes (which are all negative) on error. See remarks.

Remarks

This function is used to determine a RAM acquisition (AcquireToBoardRamPX14) is currently in progress.

Data acquisition operations are considered in progress even if the board has not yet received a trigger event.

A return value 1 should be interpreted as: A RAM acquisition operation has been started, but we haven't been notified by the hardware that the operation has completed.

A return value 0 should be interpreted as: The PX14400 driver is not currently expecting a completion notification for a RAM acquisition. This may be because the operation has already finished or because it was never attempted.

This function will always return 0 for virtual devices.

This function is useful when doing asynchronous data acquisition operations.

This function is not used for PCIe RAM buffered based acquisitions. Instead, the IsTransferInProgressPX14 is used to determine if a DMA transfer is currently in progress for PCIe RAM buffered based acquisitions.

Related Functions

WaitForAcquisitionCompletePX14

3.9.5 | WaitForAcquisitionCompletePX14

Form

```
int WaitForAcquisitionCompletePX14 (HPX14 hBrd, unsigned int timeout_ms = 0);
```

Description

Wait for a RAM acquisition operation to complete with optional timeout.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *timeout_ms*
An optional timeout value in milliseconds. If this value is 0 then the function will not timeout

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error; see remarks.

This function will return SIG_PX14_TIMED_OUT (-541) if the timeout elapses before the acquisition operation completes.

This function will return SIG_CANCELLED (-10) if the operation is cancelled before it finishes. An operation may be cancelled by putting the PX14400 into the Standby operating mode.

Remarks

This function is used when doing asynchronous data acquisition operations.

This function can be used to wait for a RAM acquisition (via AcquireToBoardRamPX14) to complete.

Calling this function while any of the above operations are in progress (see IsAcquisitionInProgressPX14) will result in the current thread being blocked until the operation finishes, times out, or is cancelled by another thread putting the PX14400 into Standby mode.

This function will automatically put the PX14400 into Standby operating mode if the wait for acquisition complete is successful. If this function returns because of an error (such as a timeout) the operating mode will remain in RAM acquisition mode.

This function is not used for PCIe RAM buffered based acquisitions. Instead, the WaitForTransferCompletePX14 is used to wait for a DMA transfer to finish for PCIe RAM buffered based acquisitions.

Related Functions

IsAcquisitionInProgressPX14

3.10 | Data Transfer Routines

The functions in this section are used to perform data transfers.

The transfer of acquisition data between the PX14400 and local host PC memory are implemented using Direct Memory Access (DMA) transfers. These are high speed data transfers that utilize the host system DMA controller in which data is transferred independently of local system processors. The PX14400 library, driver, and hardware manage all aspects of DMA transfers.

Before a DMA transfer may be performed a DMA buffer needs to be allocated. A DMA buffer is a contiguous, non-paged region of host PC memory. This differs from traditional heap-allocated memory which is only virtually contiguous and can be paged out when not in use. Once a DMA buffer is allocated, it is mapped into the user address space and used just like normal memory.

The AllocateDmaBufferPX14 library function is used to allocate a DMA buffer and map it into the address space of the current process. The FreeDmaBufferPX14 is used to free the DMA buffer when it is no longer needed.

Once the DMA buffer has been allocated, the PX14400 hardware can then transfer data directly to that buffer without any intermediate buffering by the system.

The PX14400 library implements two general categories of data transfer functions: ‘fast’ and ‘buffered’.

Fast Data Transfers

These ‘fast’ routines should be used when moving data from the PX14400 is speed critical, such as in PCIe-buffered acquisition mode streaming operations.

Each of these functions takes the address of a previously allocated DMA buffer as a function parameter. Because the data transfers are non-buffered, there are some alignment requirements on transfer lengths and offsets. See respective function documentation for more information.

The PX14400 library implements the following ‘fast’ transfer routines:

Library Function	Description
ReadSampleRamFastPX14	Transfer data from PX14400 RAM to DMA buffer on host PC.
ReadSampleRamFileFastPX14	Transfer data from PX14400 RAM to one or more files on host PC.
GetPciAcquisitionDataFastPX14	Get fresh acquisition data in RAM-Buffered PCIe acquisition mode.

Buffered Data Transfers

These 'buffered' routines can be used when speed is not a critical issue, such as transferring data after a RAM based acquisition has been completed to save data to drive storage.

These data transfer routines do not require DMA buffers to be allocated by the caller. Under the hood, these transfers work just like the 'fast' functions above, but a common driver-allocated DMA buffer is used to buffer the data before it is copied to the user buffer.

This extra buffering layer will result in a degradation in transfer speed. However, the advantage of these driver-buffered transfers is that there are no size or alignment requirements on the transfer size, length, or offset and any type of arbitrary memory locations can be used for the data transfer.

The PX14400 library implements the following driver-buffered transfer routines:

Library Function	Description
ReadSampleRamBufPX14	Transfer data in PX14400 RAM to host PC; any boundary or alignment at the expense of speed.
ReadSampleRamDualChannelBufPX14	Transfer and de-interleave dual-channel data in PX14400 RAM to host PC; any boundary or alignment at the expense of speed.
ReadSampleRamFileBufPX14	Transfer data in PX14400 RAM to a file on the host PC; any boundary or alignment at the expense of speed.
GetPciAcquisitionDataBufPX14	Get fresh acquisition data during PCIe acquisition at the expense of speed.

Asynchronous and Synchronous Data Transfers

In addition to the 'fast' and 'buffered' transfer types above, most all transfer functions can be either synchronous or asynchronous.

By default, all transfer functions are synchronous. For the functions that support asynchronous data transfers, they will have a *bAsynchronous* parameter, which determines whether a transfer will be synchronous or asynchronous. See respective transfer functions for details.

In a synchronous transfer, when the transfer function is called, it will not return until the data transfer has completed. During the duration of the data transfer, the calling thread is put into a sleep state, using no CPU resources. When the transfer completes, the thread is put back into a ready state and the function will return.

In an asynchronous transfer, when the transfer function is called the data transfer is started and the function returns immediately without waiting for the transfer to complete. This allows the calling thread to continue on with any work it may need to do, while the data transfer occurs (in parallel). When using asynchronous data transfers, the WaitForTransferCompletePX14 function is used to wait for the transfer to complete, with an optional timeout.

Asynchronous data transfers are useful when doing high-speed data recordings; an asynchronous data transfer can be occurring to one buffer in parallel with the processing of data from a previous transfer. This is how recordings are implemented in the library's Recording Session API.

3.10.1 | GetPciAcquisitionDataBufPX14

Form

```
int GetPciAcquisitionDataBufPX14 (HPX14 hBrd, unsigned int samp_count, px14_sample_t* bufp, int bAsynchronous = 0);
```

Description

Obtain fresh acquisition data during a PCIe data acquisition using buffered transfers; see remarks.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *samp_count*
The number of data samples to transfer. This value must be a multiple of 1024 samples.

[out] *bufp*
A pointer to a buffer that will receive the acquisition data. This buffer must be at least *samp_count* samples. This buffer does not need to be a DMA buffer.

[in] *bAsynchronous*
If this parameter is nonzero then an asynchronous data transfer will be performed. See remarks.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Synchronous transfers: This function will return SIG_CANCELLED (-10) if the data transfer is cancelled before it finishes. A data transfer is cancelled by putting the PX14400 into Standby mode, usually from another thread (or process).

Remarks

This function is identical in usage to the GetPciAcquisitionDataFastPX14 function. The main difference between the two functions is the underlying transfer method. Most users will want to use the 'fast' transfer functions for PCIe buffered acquisitions to maximize throughput so the FIFOs do not overflow.

This function is primarily implemented for use on platforms in which native memory access is not available (e.g. Visual Basic .NET), and is used to obtain data from the acquisition started by a previous call to BeginBufferedPciAcquisitionPX14.

This function will not return until all of the requested samples have been transferred. It is safe to call this function before the desired number of samples has been acquired.

This function can also start an asynchronous transfer. By default, this function will not return until the data transfer completes. If the bAsynchronous parameter is nonzero, then the function will start the data transfer and return immediately without waiting for it to finish allowing the calling thread to continue working. The

caller can use the IsTransferInProgressPX14 function to determine if the transfer is still in progress. To wait (sleep) for the transfer to complete, call the WaitForTransferCompletePX14 function.

Related Functions

BeginBufferedPciAcquisitionPX14, EndBufferedPciAcquisitionPX14, IsTransferInProgressPX14, WaitForTransferCompletePX14

3.10.2 | GetPciAcquisitionDataFastPX14

Form

```
int GetPciAcquisitionDataFastPX14 (HPX14 hBrd, unsigned int samp_count, px14_sample_t* dma_bufp, int bAsynchronous = 0);
```

Description

Obtain fresh acquisition data during a PCIe data acquisition using fast, unbuffered transfers.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *samp_count*
The number of data samples to transfer. This value must be a multiple of 1024 samples.

[out] *dma_bufp*
A pointer to a DMA buffer that will be used for the DMA transfer performed to obtain the acquisition data. This buffer must be at least *samp_count* samples. If a non-DMA buffer is used the function will return an error.

[in] *bAsynchronous*
If this parameter is nonzero then an asynchronous data transfer will be performed. See remarks.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Synchronous transfers: This function will return SIG_CANCELLED (-10) if the data transfer is cancelled before it finishes. A data transfer is cancelled by putting the PX14400 into Standby mode, usually from another thread (or process).

Remarks

This function is used to obtain data from the acquisition started by a previous call to BeginBufferedPciAcquisitionPX14.

This function will not return until all of the requested samples have been transferred. It is safe to call this function before the desired number of samples has been acquired.

This function can also start an asynchronous transfer. By default, this function will not return until the data transfer completes. If the bAsynchronous parameter is nonzero, then the function will start the data transfer and return immediately without waiting for it to finish allowing the calling thread to continue working. The caller can use the IsTransferInProgressPX14 function to determine if the transfer is still in progress. To wait (sleep) for the transfer to complete, call the WaitForTransferCompletePX14 function.

Related Functions

BeginBufferedPciAcquisitionPX14, EndBufferedPciAcquisitionPX14, IsTransferInProgressPX14, WaitForTransferCompletePX14

3.10.3 | IsTransferInProgressPX14

Form

```
int IsTransferInProgressPX14 (HPX14 hBrd);
```

Description

Determine if an asynchronous data transfer is currently in progress.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

Return Value

Returns 1 if a transfer is currently in progress, 0 if a transfer is not in progress, or one of the library error codes (which are all negative) on error. See remarks.

Remarks

This function is used to determine if a data transfer to or from the PX14400 is currently in progress and is only relevant when doing asynchronous DMA transfer operations.

A return value 1 should be interpreted as: A DMA transfer operation has been started, but we haven't been notified by the hardware that the operation has completed.

A return value 0 should be interpreted as: The PX14400 driver is not currently expecting a DMA complete notification from the hardware. This may be because the operation has already finished or because it was never attempted.

This function will always return 0 for virtual devices.

Related Functions

WaitForTransferCompletePX14

3.10.4 | ReadSampleRamBufPX14

Form

```
int ReadSampleRamBufPX14 (HPX14 hBrd, unsigned int sample_start, unsigned int sample_count, px14_sample_t* bufp, int bAsynchronous = 0);
```

Description

Transfer data in PX14400 RAM to a buffer on the PC; any boundary or alignment at the expense of speed.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *sample_start*
The first sample in the PX14400 RAM in which you would like to copy. This can be any valid sample number and need not be on any particular boundary.

[in] *sample_count*
The number of samples to copy from the PX14400 RAM. This value need not be on any particular boundary.

[out] *bufp*
A pointer to the buffer that will receive the PX14400 data. This buffer does not have to be a DMA buffer. (If you do have a DMA buffer you should use ReadSampleRamFastPX14 since it's much faster due to not having to use intermediate buffering.)

[in] *bAsynchronous*
If this parameter is nonzero then an asynchronous data transfer will be performed; see Remarks.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

This function transfers the data from the given region of the given board to the given buffer. Note that this buffer need not be a DMA buffer. This function exists as a convenience in situations where getting data off of the board is not speed-critical. This function does not run as fast as the ReadSampleRamFastPX14 function but has a few advantages:

- No DMA buffer needs to be allocated by the caller.
- Starting sample and transfer length needn't be on any particular boundary.
- The transfer size is not bound by the minimum or maximum DMA transfer size.

The difference between this function and the ReadSampleRamFastPX14 is that the ReadSampleRamFastPX14 function requires a DMA buffer in order to do the data transfer. Also, start and length of transfer are also

restricted to aligned values when using ReadSampleRamFastPX14. The advantage of this is that ReadSampleRamFastPX14 can run faster since no intermediate buffering is required.

To transfer and automatically de-interleave dual-channel data use the ReadSampleRamDualChannelBufPX14 function.

This function can also start an asynchronous transfer. By default, this function will not return until the data transfer completes. If the *bAsynchronous* parameter is nonzero, then the function will start the data transfer and return immediately without waiting for it to finish allowing the calling thread to continue working. The caller can use the IsTransferInProgressPX14 function to determine if the transfer has completed. To wait (sleep) for the transfer to complete, call the WaitForTransferCompletePX14 function.

Related Functions

ReadSampleRamFastPX14, ReadSampleRamDualChannelBufPX14

3.10.5 | ReadSampleRamDualChannelBufPX14

Form

```
int ReadSampleRamDualChannelBufPX14 (HPX14 hBrd, unsigned int sample_start, unsigned int sample_count,
px14_sample_t* buf_ch1p, px14_sample_t* buf_ch2p, int bAsynchronous = 0);
```

Description

Transfer and de-interleave dual-channel data in board RAM to host PC; any boundary or alignment at the expense of speed.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *sample_start*
The first sample in the PX14400 RAM in which you would like to copy. This can be any valid sample number and need not be on any particular boundary.

[in] *sample_count*
The number of samples to copy from the PX14400 RAM. This value need not be on any particular boundary.

[out] *buf_ch1p*
A pointer to the buffer that will receive the channel 1 data. This buffer must be large enough to hold *sample_count* / 2 samples. This parameter may be NULL if channel 1 data is not needed.

[out] *buf_ch2p*
A pointer to the buffer that will receive the channel 2 data. This buffer must be large enough to hold *sample_count* / 2 samples. This parameter may be NULL if channel 2 data is not needed.

[in] *bAsynchronous*
If this parameter is nonzero then an asynchronous data transfer will be performed; see Remarks.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

This function transfers the data from the given region of the given board to the given buffer. Note that this buffer need not be a DMA buffer. This function exists as a convenience in situations where getting data off of the board is not speed-critical. Speed critical applications should use the ReadSampleRamFastPX14 function to obtain single- or dual-channel data.

This function will also automatically de-interleave the data into separate buffers. If you do not want to de-interleave data you should use the ReadSampleRamBufPX14 function.

This function can also start an asynchronous transfer. By default, this function will not return until the data transfer completes. If the *bAsynchronous* parameter is nonzero, then the function will start the data transfer and return immediately without waiting for it to finish allowing the calling thread to continue working. The caller can use the IsTransferInProgressPX14 function to determine if the transfer has completed. To wait (sleep) for the transfer to complete, call the WaitForTransferCompletePX14 function.

Related Functions

ReadSampleRamBufPX14

3.10.6 | ReadSampleRamFastPX14

Form

```
int ReadSampleRamFastPX14 (HPX14 hBrd, unsigned int sample_start, unsigned int sample_count, px14_sample_t* dma_bufp, int bAsynchronous = 0);
```

Description

Transfer data in PX14400 RAM to a DMA buffer on the host PC.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *sample_start*
The index of the first sample to copy. This parameter has the same restrictions defined in the SetStartSamplePX14 function.

[in] *sample_count*
The total number of samples to copy. This parameter has the same restrictions defined in the SetSampleCountPX14 function and must be an integer multiple of 1024 samples.

[out] *dma_bufp*

The address of the host PC buffer at which the sample data will be copied to. This buffer must be large enough to hold *sample_count* samples and must have been allocated for this device by the AllocateDmaBufferPX14 function.

[in] *bAsynchronous*

If this parameter is nonzero then an asynchronous data transfer will be performed; see Remarks.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

This function copies a section of PX14400 signal RAM to a buffer in the host PC memory using Direct Memory Access (DMA) transfers. Before this function may be used, a DMA buffer must be allocated for the target device using the AllocateDmaBufferPX14 function.

This function handles all necessary operating mode changes and active memory settings.

The difference between this function and the ReadSampleRamBufPX14 function is that ReadSampleRamBufPX14 does not require a DMA buffer for input. Rather, an internal driver-managed DMA buffer is used for transfer. This requires an extra buffering of the data that can hinder high-performance applications.

This function can also start an asynchronous transfer. By default, this function will not return until the data transfer completes. If the *bAsynchronous* parameter is nonzero, then the function will start the data transfer and return immediately without waiting for it to finish allowing the calling thread to continue working. The caller can use the IsTransferInProgressPX14 function to determine if the transfer has completed. To wait (sleep) for the transfer to complete, call the WaitForTransferCompletePX14 function.

Related Functions

AllocateDmaBufferPX14, IsTransferInProgressPX14, ReadSampleRamBufPX14, WaitForTransferCompletePX14

3.10.7 | ReadSampleRamFileBufPX14

Form

```
int ReadSampleRamFileBufPX14 (HPX14 hBrd, unsigned int sample_start, unsigned int sample_count, PX14S_FILE_WRITE_PARAMS* paramsp);
```

Description

Transfer data in PX14400 RAM to a file on the host PC; any boundary or alignment at the expense of speed.

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *sample_start*

The index of the first sample to copy. This can be any valid sample number and need not be on any particular boundary.

[in] *sample_count*

The total number of samples to copy. This can be any valid sample number and need not be on any particular boundary.

[in] *paramsp*

A pointer to a PX14S_FILE_WRITE_PARAMS structure that defines how and where data will be saved.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

This function handles all necessary operating mode changes and active memory settings.

Related Functions

ReadSampleRamBufPX14, ReadSampleRamFileFastPX14

3.10.8 | ReadSampleRamFileFastPX14

Form

```
int ReadSampleRamFileFastPX14 (HPX14 hBrd, unsigned int sample_start, unsigned int sample_count, px14_sample_t* dma_bufp, unsigned int dma_buf_samples, PX14S_FILE_WRITE_PARAMS* paramsp);
```

Description

Transfer data in PX14400 RAM to a file on the host PC.

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *sample_start*

The index of the first sample to copy. This parameter has the same restrictions defined in the SetStartSamplePX14 function.

[in] *sample_count*

The total number of samples to copy. This parameter has the same restrictions defined in the SetSampleCountPX14 function and must be an integer multiple of 1024 samples.

[in] *dma_bufp*

The address of a DMA buffer previously allocated by the AllocateDmaBufferPX14 function. This DMA buffer is used for the data transfer between the PX14400 and host PC. This buffer need not be as large as the total amount of data to transfer.

[in] *dma_buf_samples*
The size, in samples, of the DMA buffer pointed to by dma_bufp

[in] *paramsp*
A pointer to a PX14S_FILE_WRITE_PARAMS structure that defines how and where data will be saved.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

This function copies a section of PX14400 signal RAM to a file on the host PC using Direct Memory Access (DMA) transfers. Before this function may be used, a DMA buffer must be allocated for the target device using the AllocateDmaBufferPX14 function.

This function handles all necessary operating mode changes and active memory settings.

Related Functions

AllocateDmaBufferPX14, ReadSampleRamBufPX14

3.10.9 | WaitForTransferCompletePX14

Form

```
int WaitForTransferCompletePX14 (HPX14 hBrd, unsigned int timeout_ms = 0);
```

Description

Wait for a DMA transfer operation to complete with optional timeout.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *timeout_ms*
An optional timeout value in milliseconds. If this value is 0 then the function will not timeout.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error; see remarks.

This function will return SIG_PX14_TIMED_OUT (-541) if the timeout elapses before the acquisition operation completes.

This function will return SIG_CANCELLED (-10) if the operation is cancelled before it finishes. An operation may be cancelled by putting the PX14400 into the Standby operating mode.

Remarks

This function is used when doing asynchronous DMA transfer operations.

Calling this function while a DMA transfer is in progress (see IsTransferInProgressPX14) will result in the current thread being blocked until the operation finishes, times out, or is cancelled by another thread putting the board into Standby mode.

Related Functions

IsTransferInProgressPX14

3.11 | Data Manipulation Routines

The functions in this section are used to perform data manipulations.

3.11.1 | DeInterleaveDataPX14

Form

```
int DeInterleaveDataPX14 (const px14_sample_t* srcp, unsigned int samples_in, px14_sample_t* dst_ch1p, px14_sample_t* dst_ch2p);
```

Description

This function is used to de-interleave dual channel data into separate buffers.

Parameters

[in] *srcp*
A pointer to a buffer containing interleaved dual-channel data.

[in] *samples_in*
The total number of samples contained in the buffer pointed to by srcp.

[out] *dst_ch1p*
A pointer to a buffer that will receive channel 1 data. This parameter may be NULL if channel 1 data isn't needed.

[out] dst_ch2p
A pointer to a buffer that will receive channel 2 data. This parameter may be NULL if channel 2 data isn't needed.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

When dual channel data is acquired on the PX14400, sample data is interleaved: Channel 1 Sample 1, Channel 2 Sample 1, Channel 1 Sample 2, Channel 2 Sample 2, etc. Use this function to extract out one or both channels of data into separate buffers.

Related Functions

InterleaveDataPX14

3.11.2 | InterleaveDataPX14

Form

```
int InterleaveDataPX14 (const px14_sample_t* src_ch1p, const px14_sample_t* src_ch2p, unsigned int samps_per_chan, px14_sample_t* dstp);
```

Description

Interleave dual channel data into a single buffer.

Parameters

[in] *src_ch1p*
A pointer to a buffer that contains channel 1 data. If this parameter is NULL then no data is copied over the channel 1 data samples in the destination buffer.

[in] *src_ch2p*
A pointer to a buffer that contains channel 2 data. If this parameter is NULL then no data is copied over the channel 2 data samples in the destination buffer.

[in] *samps_per_chan*
The number of samples contained in each of the input channel data buffers.

[out] *dstp*
A pointer to the buffer that will receive the interleaved data. This buffer must be at least (2 * *samps_per_chan*) samples in size.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

This function may be used to (re)create interleaved dual channel data.

Related Functions

DeInterleaveDataPX14

3.12 | Timestamp Management

The functions in this section are used for timestamp management.

3.12.1 | GetTimestampAvailabilityPX14

Form

```
int GetTimestampAvailabilityPX14 (HPX14 hBrd);
```

Description

Determine if any timestamps are available in the PX14400 timestamp FIFO.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

Return Value

Returns 0 if the timestamp FIFO is currently empty (indicating that no timestamps are available to be read), a positive value if the timestamp FIFO is not empty, or one of the library error codes (which are all negative) on error.

Remarks

This function is used to determine if any timestamps are available to be read.

Related Functions

GetTimestampOverflowFlagPX14, ReadTimestampDataPX14

3.12.2 | GetTimestampFifoDepthPX14

Form

```
int GetTimestampFifoDepthPX14 (HPX14 hBrd, unsigned int* ts_elementsp);
```

Description

Obtain the size of the PX14400 timestamp FIFO, in timestamp elements.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[out] *ts_elementsp*
A pointer to the unsigned int variable that will receive the size of the PX14400 timestamp FIFO.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

The standard PX14400 timestamp FIFO size is 2048 timestamps. Each timestamp is an unsigned 64-bit integer.

3.12.3 | GetTimestampOverflowFlagPX14

Form

```
int GetTimestampOverflowFlagPX14 (HPX14 hBrd);
```

Description

Read Timestamp Overflow Flag from PX14400.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

Return Value

Returns 0 if the PX14400 timestamp FIFO is not full, a positive value if the timestamp FIFO is full, or one of the library error codes (which are all negative) on error.

Remarks

If the timestamp FIFO goes full, then one or more timestamps may have been lost.

Related Functions

GetTimestampAvailabilityPX14, ReadTimestampDataPX14

3.12.4 | ReadTimestampDataPX14

Form

```
int ReadTimestampDataPX14 (HPX14 hBrd, px14_timestamp_t* bufp, unsigned int ts_count, unsigned int* ts_readp, unsigned int flags = 0, unsigned int timeout_ms = 0, unsigned int* flags_outp = NULL);
```

Description

Read timestamp data from PX14400 timestamp FIFO.

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[out] *bufp*

A pointer to the buffer that will receive the timestamp values.

[in] *ts_count*

The size of the buffer pointed to by the bufp parameter, in timestamp elements.

[out] *ts_readp*

A pointer to an unsigned int variable that will receive the number of timestamps read from the timestamp FIFO. This parameter may not be NULL.

[in] *flags*

A set of flags (PX14TSREAD_*) that define function behavior. Currently defined flags are in the following table:

PX14400 Library Constant	Value	Interpretation
PX14TSREAD_READ_FROM_FULL_FIFO	0x00000001	Set to read from a known-full timestamp FIFO.

[in] *timout_ms*

This value is currently ignored, but may be used in a future to specify a timeout for a read operation.

[out] *flags_outp*

A pointer to an unsigned int variable that will receive output flags describing the state of the timestamp FIFO after the read operation. These flags can be used to determine if another timestamp read operation can take place immediately. Currently defined flags are in the following table:

PX14400 Library Constant	Value	Interpretation
PX14TSREAD_MORE_AVAILABLE	0x01000000	Timestamp FIFO is not empty after read; more timestamps are available.
PX14TSREAD_FIFO_OVERFLOW	0x02000000	Timestamp FIFO was full prior to the read operation.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

Calling this function may not be necessary. The PX14400 library can be setup to automatically download and save timestamps during a recording session, or when using ReadSampleRamFileFastPX14/ ReadSampleRamFileBufPX14. See documentation for PX14S_FILE_WRITE_PARAMS structure for details on this.

This function will read up to *ts_count* timestamps from the PX14400 timestamp FIFO. If the timestamp FIFO goes empty before reading *ts_count* timestamps, the function will return; it will not wait for more timestamps to accumulate.

It is safe to call this function while an acquisition or recording is in progress.

Timestamps will be generated based on the rules of the current timestamp mode, which is set by the SetTimestampModePX14 function.

If the PX14TSREAD_READ_FROM_FULL_FIFO flag is not specified and the timestamp FIFO is full, then this function will return success (0) and the variable pointed to by the ts_readp parameter will be set to zero. The PX14TSREAD_FIFO_OVERFLOW output flag will also be set. To read from a known full FIFO, set the PX14TSREAD_READ_FROM_FULL_FIFO flag.

Related Functions

SetTimestampModePX14

3.12.5 | ResetTimestampCounterPX14

Form

```
int ResetTimestampCounterPX14 (HPX14 hBrd);
```

Description

Reset the timestamp counter.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

After this operation the timestamp counter is reset back to zero. This operation will have no effect on the timestamp FIFO; any existing FIFO data will be unaffected.

Related Functions

ResetTimestampFifoPX14, SetTimestampCounterModePX14

3.12.6 | ResetTimestampFifoPX14

Form

```
int ResetTimestampFifoPX14 (HPX14 hBrd);
```

Description

Reset the timestamp FIFO.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

After this operation the timestamp FIFO will be empty. This operation will have no effect on the timestamp counter.

Related Functions

ResetTimestampCounterPX14

3.13 | Recording Session Management

The PX14400 library implements a high-level recording interface that simplifies coding for recording PX14400 acquisition data to permanent storage. Using this interface, all aspects of PX14400 hardware interaction and output file management is managed internally by the library.

The library currently supports two types of recordings: PCIe Buffered Acquisition (streaming) recordings and RAM Acquisition/Transfer recordings.

3.13.1 | AbortRecordingSessionPX14

Form

```
int AbortRecordingSessionPX14 (HPX14RECORDING hRec);
```

Description

Abort the current PX14400 recording session; stops all recording and closes all files.

Parameters

[in] *hRec*

A handle to a PX14400 recording session. This handle is obtained by calling the CreateRecordingSessionPX14 function.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Related Functions

CreateRecordingSessionPX14

3.13.2 | ArmRecordingSessionPX14

Form

```
int ArmRecordingSessionPX14 (HPX14RECORDING hRec);
```

Description

Arm PX14400 device for recording.

Parameters

[in] *hRec*

A handle to a PX14400 recording session. This handle is obtained by calling the CreateRecordingSessionPX14 function.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

This function is used to arm the PX14400 for recording. This function is only needed if the PX14RECSESF_DO_NOT_ARM recording flag is specified during creation of the recording session.

Related Functions

CreateRecordingSessionPX14

3.13.3 | CreateRecordingSessionPX14

Form

int CreateRecordingSessionPX14 (HPX14 hBrd, PX14S_REC_SESSION_PARAMS* rec_paramsp, HPX14RECORDING* handlep);

Description

Create a PX14400 acquisition recording session.

Parameters

[in] *hBrd*

A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[in] *rec_paramsp*

A pointer to a PX14S_REC_SESSION_PARAMS structure that defines the parameters of the data recording session. This structure and its members are detailed in the Structure PX14S_REC_SESSION_PARAMS section.

[out] *handlep*

A pointer to a HPX14RECORDING variable that will receive a handle that identifies the recording session.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

This function is used to create a PX14400 recording session. A recording session is a PX14400 library interface used to fully manage the recording of PX14400 acquisition data to permanent storage.

Related Functions

DeleteRecordingSessionPX14

3.13.4 | DeleteRecordingSessionPX14

Form

int DeleteRecordingSessionPX14 (HPX14RECORDING hRec);

Description

Delete a PX14400 recording session.

Parameters

[in] *hRec*

A handle to a PX14400 recording session. This handle is obtained by calling the CreateRecordingSessionPX14 function.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

If a recording is currently in progress when this function is called, it will automatically be aborted via a call to AbortRecordingSessionPX14.

Related Functions

CreateRecordingSessionPX14

3.13.5 | GetRecordingSessionOutFlagsPX14

Form

```
int GetRecordingSessionOutFlagsPX14 (HPX14RECORDING hRec, unsigned int* flagsp);
```

Description

Obtain output flags of the specified recording session. This function is to be used when the recording session is stopped, and before deleting the session.

Parameters

[in] *hRec*

A handle to a PX14400 recording session. This handle is obtained by calling the CreateRecordingSessionPX14 function.

[out] *flagsp*

A pointer to an unsigned int variable that will receive the recording session output flags. See Remarks.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

Recording output flags are status flags that are available after a recording has completed and before the session has been deleted. These flags are only relevant after a recording session has stopped. Currently defined flags are listed in the following table:

PX14400 Library Constant	Value	Interpretation
PX14FILWOUTF_TIMESTAMP_FIFO_OVERFLOW	0x00000001	Timestamp FIFO overflowed during write/record process.
PX14FILWOUTF_NO_TIMESTAMP_DATA	0x00000002	No timestamp data was available during the operation.

Related Functions

CreateRecordingSessionPX14

3.13.6 | GetRecordingSessionProgressPX14

Form

```
int GetRecordingSessionProgressPX14 (HPX14RECORDING hRec, PX14S_REC_SESSION_PROG* progp, unsigned int flags = 0);
```

Description

Obtain progress/status for current recording session.

Parameters

[in] *hRec*

A handle to a PX14400 recording session. This handle is obtained by calling the CreateRecordingSessionPX14 function.

[out] *progp*

A pointer to a PX14S_REC_SESSION_PROG structure that will receive the current progress/status of the PX14400 recording session. The caller should initialize the struct_size field of this structure before calling this function.

[in] *flags*

A set of flags (PX14RECPROGF_*) that control function behavior. Currently, only a single flag is defined: PX14RECPROGF_NO_ERROR_TEXT (1). If this flag is set then the function will not generate an error text string if an error has occurred during recording.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Related Functions

CreateRecordingSessionPX14, GetRecordingSnapshotPX14

3.13.7 | GetRecordingSnapshotPX14

Form

```
int GetRecordingSnapshotPX14 (HPX14RECORDING hRec, px14_sample_t* bufp, unsigned int samples, unsigned int* samples_gotp, unsigned int* ss_countp);
```

Description

Obtain data snapshot from current PX14400 recording.

Parameters

[in] *hRec*

A handle to a PX14400 recording session. This handle is obtained by calling the CreateRecordingSessionPX14 function.

[out] *bufp*

A pointer to a buffer that will receive the recording snapshot data.

[in] *samples*

The size, in samples, of the buffer pointed to by the bufp parameter.

[out] *samples_gotp*

The address of an unsigned int variable that will receive the number of samples copied into the snapshot buffer. Pass NULL if this information is not needed.

[out] *ss_countp*

The address of an unsigned int variable that will receive the snapshot counter value. Each data snapshot taken by the recording is given a unique counter value. This allows client software to differentiate between unique data snapshots. Pass NULL if this information is not needed.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

Related Functions

CreateRecordingSessionPX14, GetRecordingSessionProgressPX14

3.14 | Signatec Recorded Data Context (SRDC)

The functions in this section are used for Signatec Recorded Data Context (SRDC) files associated with recording sessions. See the Signatec Recorded Data Context Files (*.srdc) section for further details.

SDRC data is stored in XML format. The data consists of a number of named item-value pairs. Here is an example SDRC data set:

```
<?xml version="1.0" encoding="UTF-8"?>
<SignatecRecordedData>
<!--Source board information-->
 <SourceBoard>PX14400</SourceBoard>
 <SourceBoardSerialNum>100147</SourceBoardSerialNum>
<!--Data sample information-->
 <ChannelCount>2</ChannelCount>
 <ChannelId>0</ChannelId>
 <SampleSizeBytes>2</SampleSizeBytes>
 <SampleSizeBits>16</SampleSizeBits>
 <SampleFormat>Unsigned</SampleFormat>
<!--Data acquisition information-->
 <SamplingRateMHz>350</SamplingRateMHz>
 <PeakToPeakInputVoltRange>2.2</PeakToPeakInputVoltRange>
 <SegmentSize>0</SegmentSize>
 <PreTriggerSampleCount>0</PreTriggerSampleCount>
 <TriggerDelaySamples>0</TriggerDelaySamples >
<!--User-defined items-->
 <OperatorNotes>Sample operator notes....</OperatorNotes>
 <SomeUserDefinedItem>User defined data abc123</SomeUserDefinedItem>
<!--Other items-->
 <RecArmTimeSec>1390574820</RecArmTimeSec>
 <RecArmTimeStr>Fri Jan 24 06:47:00 2014</RecArmTimeStr>
 <RecEndTimeSec>1390574824</RecEndTimeSec>
 <RecEndTimeStr>Fri Jan 24 06:47:04 2014</RecEndTimeStr>
</SignatecRecordedData>
```

A SRDC context item is defined by an item name, represented by an element, and that item's value, represented by the element's data. So in the data above, we can see there is an item "ChannelCount" with a value of "2". There are a number of predefined items that are used to define the data that this SDRC data is associated with. Users may specify any number of user defined items for their own applications.

Some notes on the XML used to represent SRDC data:

- The root element must be named SignatecRecordedData.
- Only immediate child elements of the root node are considered as context items.
- The PX14400 library considers item names to be case sensitive.
- Item names should be unique; if duplicate item names are used, only the last one will be used.

Predefined SRDC Items

Item Name	Value Interpretation	Default Value
SourceBoard	Name of the board that generated the described data. For data generated by a PX14400, this value will be “PX14400”.	<None>
SourceBoardSerialNum	The serial number of the board that generated the described data.	<None>
ChannelCount	The number of channels of data in the described data. Multichannel data is assumed to sample interleaved.	1
ChannelId	The board specific channel number that generated the described data. This value will be 0 for multichannel data.	1
SampleSizeBytes	The size of a data sample in bytes. This will be 2 for all data generated by a PX14400.	2
SampleSizeBits	The size of a data sample in bits. This will be 16 for all data generated by a PX14400. only the lower 14-bits are relevant (the upper two bits will always be zero).	16
SampleFormat	The format of a data sample. May be either “Signed” or “Unsigned”. PX14400 boards currently only acquire unsigned data, but software options allow for conversion to signed format.	Unsigned
SamplingRateMHz	The sampling rate, in MHz, used when acquiring the described data.	0
PeakToPeakInputVoltRange	The peak-to-peak input voltage, in volts, used when acquiring the described data.	0
InputGain_dB	The amount of gain, in dB, applied to the input data.	0
SegmentSize	The size of a data segment used when acquiring with the Segmented triggering mode. If data is non-segmented this value should be 0.	0
PreTriggerSampleCount	The number of pre-trigger samples in the described data. This indicates the number of samples kept prior to the trigger event that started the acquisition.	0
TriggerDelaySamples	The number of trigger delay samples in the described data. This is the number of samples after a trigger event that are ignored.	0
FileFormat	The format of the data file containing the described data. May be either “Binary” or “Text”.	Binary
SampleRadix	The radix used to save data in text format. This value only has relevance when the FileFormat item is “Text’.	
HeaderBytes	The number of initial bytes set aside for an application defined header in the described data.	0
OperatorNotes	User-defined operator notes describing the described data set.	<None>

Continued:

Item Name	Value Interpretation	Default Value
RecArmTimeSec	Time when recording was armed, in seconds since midnight 1/1/1970. This item will only be present for files generated when using the PX14400 Recording Session API.	
RecArmTimeStr	A full string representation of date and time of when recording was armed (i.e. placed into an acquisition operating mode). This item will only be present for files generated when using the PX14400 Recording Session API.	
RecEndTimeSec	Time when recording was stopped, in seconds since midnight 1/1/1970. This item will only be present for files generated when using the PX14400 Recording Session API.	
RecEndTimeStr	A full string representation of date and time of when recording stopped This item will only be present for files generated when using the PX14400 Recording Session API.	

The following library functions are used to programmatically interact with SRDC data files.

3.14.1 | CloseSrdcFilePX14

Form

```
int CloseSrdcFilePX14 (HPX14SRDC hFile);
```

Description

Close given SRDC file without updating contents.

Parameters

[in] *hFile*
A handle to a SRDC file previously opened by calling the OpenSrdcFilePX14 function.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

Call this function to close the in-memory representation of the SRDC data. After calling this function, the given SRDC file handle is invalid.

Note that closing the SRDC file handle does not automatically flush modified content to drive storage. Call the SaveSrdcFilePX14 function to ensure all modifications are written to drive storage.

Related Functions

OpenSrdcFilePX14, SaveSrdcFilePX14

3.14.2 | EnumSrdcItemsPX14

Form

```
int EnumSrdcItemsAPX14 (HPX14SRDC hFile, TCHAR** itemspp, unsigned int flags);
```

Description

Obtain enumeration of all SRDC items with given constraints.

Parameters

[in] *hFile*
A handle to a SRDC file previously opened by calling the OpenSrdcFilePX14 function.

[out] *itemspp*
A pointer to a TCHAR pointer that will receive the address of a library allocated buffer. This buffer will contain a space-delimited list of all SRDC context items with constraints defined by the flags parameter.

[in] *flags*
A set of flags that determine which items to include in the enumeration

Flag	Value	Interpretation
PX14SRDCENUM_SKIP_STD	0x00000001	Skip standard SRDC items; use to obtain only user-defined items.
PX14SRDCENUM_SKIP_USER_DEFINED	0x00000002	Skip user-defined SRDC items; use to obtain only standard items.
PX14SRDCENUM_MODIFIED_ONLY	0x00000004	Only include modified items in enumeration.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

This library function will return an enumeration of currently defined SRDC items in the form of a space-delimited string in case-sensitive, alphabetical-ascending order.

3.14.3 | GetRecordedDataInfoPX14

Form

```
int GetRecordedDataInfoPX14 (const TCHAR* pathnamep, PX14S_RECORDED_DATA_INFO* infop, TCHAR** operator_notespp = NULL);
```

Description

Obtain basic SRDC information on acquisition data in given file.

Parameters

[in] *pathnamep*
A pointer to a NULL-terminated string containing the pathname of the acquisition data (*.rd16).

[in] *infop*
A pointer to a PX14S_RECORDED_DATA_INFO structure that will receive some most basic SRDC information for the acquired data.

[out] *operator_notespp*
The address of a TCHAR* that will receive the address of a library-allocated buffer containing the operator notes for the acquired data. If no operator notes have been specified, the library will write NULL. The caller should free the memory for this string when no longer needed by calling the FreeMemoryPX14 function. Pass NULL if operator notes are not required.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

Prior to calling this function, the caller must initialize the struct_size field of the input PX14S_RECORDED_DATA_INFO structure to the size of the structure in bytes.

The library will attempt to find the SRDC information for the given acquisition data file. It will first check to see if an external file is present (pathname + ".srdc"), then check for an alternate file stream (pathname + ":SRDC"), and lastly, check the given file itself.

Note this function is capable for obtaining recorded data information for data generated by other Signatec data acquisition devices. The underlying SRDC data format is generic across all Signatec data acquisition devices.

3.14.4 | GetSrdcItemPX14

Form

```
int GetSrdcItemPX14 (HPX14SRDC hFile, const TCHAR* namep, TCHAR** valuepp)
```

Description

Look up SRDC item with given name; name is case-sensitive.

Parameters

[in] *hFile*
A handle to a SRDC file previously opened by calling the OpenSrdcFilePX14 function.

[in] *namep*
A pointer to a NULL-terminated string containing the case-sensitive name of the context item to find the value for.

[out] *valuepp*

A pointer to a TCHAR pointer that will receive the address of the library allocated buffer containing the value for the given named SRDC item. If no value has been specified for this name the library will return a NULL address. It is the caller's responsibility to free the storage for the item value when no longer needed; use the FreeMemoryPX14 function. NULL may be passed for this parameter if the item value is not needed.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error. If the named item does not exist in the SRDC data, SIG_PX14_NAMED_ITEM_NOT_FOUND will be returned.

Related Functions

SetSrdcItemPX14

3.14.5 | IsSrdcFileModifiedPX14

Form

```
int IsSrdcFileModifiedPX14 (HPX14SRDC hFile)
```

Description

Determine if given SRDC file data has been modified.

Parameters

[in] *hFile*

A handle to a SRDC file previously opened by calling the OpenSrdcFilePX14 function.

Return Value

Returns > 0 if given SRDC data has been modified, 0 if SRDC data has not been modified, or one of the library error codes (which are all negative) on error.

Remarks

The SetSrdcItemPX14 function is used to modify the value of a specific SRDC data item. The RefreshSrdcParametersPX14 function can also change one or more standard SRDC data item values.

Saving SRDC data via the SaveSrdcFilePX14 function will reset the modified state of all SRDC items.

3.14.6 | OpenSrdcFilePX14

Form

```
int OpenSrdcFilePX14 (HPX14 hBrd, HPX14SRDC* handlep, const TCHAR* pathnamep, unsigned flags = 0);
```

Description

Open a new or existing .srdc file.

Parameters

[in] *hBrd*
A handle to the PX14400 board. This handle is obtained by calling the ConnectToDevicePX14 function.

[out] *handlep*
A pointer to a HPX14SRDC variable that will receive a handle that identifies the SRDC file.

[in] *pathnamep*
A pointer to a NULL-terminated string that contains the pathname of the SRDC context file. This can be a path to a .srdc file (e.g. C:\Path\To\RecordingData.rd16.srdc) or an alternate file stream containing SRDC data (e.g. C:\Path\To\RecordingData.rd16:SRDC).

[in] *flags*
A set of flags (PX14SRDOF_*) that define function behavior. Currently defined flags are:

Flag	Value	Interpretation
PX14SRDCOF_QUICK_SET	0x00000001	Opens/create file, refresh and write settings, then close file. See Remarks.
PX14SRDCOF_OPEN_EXISTING	0x00000002	Open existing SRDC file; function will fail if file does not exist.
PX14SDRCOF_CREATE_NEW	0x00000004	Create a new SDRC file; will ignore and overwrite previously existing data.
PX14SRDCOF_PATHNAME_IS_REC_DATA	0x00000008	Given pathname is recorded data file; have library search for SRDC data. Using this flag implies PX14SRDCOF_OPEN_EXISTING. This flag cannot be used with PX14SDRCOF_CREATE_NEW.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

This function is used to open or create a new Signatec Recorded Data Context file. Once the file has been opened, context item-value pairs can be read or written.

Flags PX14SRDCOF_OPEN_EXISTING and PX14SDRCOF_CREATE_NEW are mutually exclusive. The library will return an error (SIG_PX14_INVALID_ARG_4) if both are set. Also note that use of PX14SRDCOF_PATHNAME_IS_REC_DATA implies PX14SRDCOF_OPEN_EXISTING.

If the PX14SRDCOF_QUICK_SET flag is set, then the function will open the file (if it exists), refresh standard SRDC context items (by calling RefreshSrdcParametersPX14), save changes, and close the file. In this case, no SRDC file handle is returned.

Related Functions

CloseSrdcFilePX14, GetSrdcItemPX14, RefreshSrdcParametersPX14, SaveSrdcFilePX14, SetSrdcItemPX14

3.14.7 | RefreshSrdcParametersPX14

Form

```
int RefreshSrdcParametersPX14 (HPX14SRDC hFile, unsigned flags = 0);
```

Description

Refresh SRDC data with current board settings; not written to file.

Parameters

[in] *hFile*
A handle to a SRDC file, previously opened by calling the OpenSrdcFilePX14 function.

[in] *flags*
No flags have been defined yet; pass 0.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

Call this function to refresh current SRDC data with current hardware settings. This includes the following SDRC items: SourceBoard, SourceBoardSerialNum, ChannelCount, ChannelId, SampleSizeBytes, SampleSizeBits, SampleFormat, SamplingRateMHz, PeakToPeakInputVoltRange, InputGain_dB, SegmentSize, PreTriggerSampleCount, and TriggerDelaySamples.

Note that these changes are only written to the in-memory representation of the SDRC data. Changes will not be saved to disk until the SaveSrdcFilePX14 function is called.

Related Functions

OpenSrdcFilePX14, SaveSrdcFilePX14

3.14.8 | SaveSrdcFilePX14

Form

```
int SaveSrdcFilePX14 (HPX14SRDC hFile, const TCHAR* pathnamep);
```

Description

Write SRDC data to file.

Parameters

[in] *hFile*
A handle to a SRDC file, previously opened by calling the OpenSrdcFilePX14 function.

[in] *pathnamep*
A pointer to a NULL-terminated string containing the pathname of the output file. This can also be the pathname of an alternate file stream in which to save the SDRC data. This parameter may be NULL if the SRDC file was opened with an explicit pathname, or if the data has already been saved to a file with an earlier call to this function.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

Call this function to flush all SRDC data to a file (or alternate file stream). After calling this function, all 'modified' flags will be reset.

Related Functions

OpenSrdcFilePX14

3.14.9 | SetSrdcItemPX14

Form

```
int SetSrdcItemPX14 (HPX14SRDC hFile, const TCHAR* namep, const TCHAR* valuep);
```

Description

Add/modify SRDC item with given name; not written to file.

Parameters

[in] *hFile*
A handle to a SRDC file, previously opened by calling the OpenSrdcFilePX14 function.

[in] *namep*

A pointer to a NULL-terminated string containing the case sensitive name of the context item to modify.

[in] *valuep*

A pointer to a NULL-terminated string containing the value to be associated with the given name. If this parameter is NULL, then the given named item will be removed from the SRDC data.

Return Value

Returns SIG_SUCCESS (0) on success or one of the library error codes (which are all negative) on error.

Remarks

Calling this function only changes the in-memory representation of the SDRC data. Changes will not be written to drive storage until the SaveSrdcFilePX14 function is called.

Modifying the value of a SRDC item will result in that item's internal "modified" bit to be set.

Related Functions

GetSrdcItemPX14, OpenSrdcFilePX14, SaveSrdcFilePX14

3.15 | Library Data Types and Structures

The following sections define some of the data types and structures used by the PX14400 library.

3.15.1 | Data Type: HPX14

This data type is used to represent a handle to a PX14400 device.

A special library constant PX14_INVALID_HANDLE (NULL) represents an invalid PX14400 device handle. Any other value should be considered opaque; this value only has relevance to the PX14400 library implementation.

PX14400 handles are only valid in the process for which they are obtained.

The DisconnectFromDevicePX14 library function should be called to release a PX14400 device handle when it is no longer needed. This will free handle-specific resources allocated by the library.

3.15.2 | Data Type: HPX14RECORDING

This data type is used to represent a handle to a PX14400 Recording Session.

A special library constant INVALID_HPX14RECORDING_HANDLE (NULL) represents an invalid PX14400 Recording Session handle. Any other value should be considered opaque; this value only has relevance to the PX14400 library implementation.

PX14400 Recording Session Handles are only valid in the process for which they are obtained.

3.15.3 | Data Type: HPX14SRDC

This data type is used to represent a handle to a Signatec Recorded Data Context (SRDC) file.

A special library constant PX14_INVALID_HPX14SRDC (NULL) represents an invalid SRDC file handle. Any other value should be considered opaque; this value only has relevance to the PX14400 library implementation.

3.15.4 | Data Type: px14_sample_t

This data type is used to represent a PX14400 data sample. It is typically defined as an 'unsigned short' (unsigned 16-bit integral value; the 2 least significant bits will always be zero).

Currently, the PX14400 will only generate little-endian, unsigned data.

3.15.5 | Data Type: px14_timestamp_t

This data type is used to represent a PX14400 timestamp value. It is typically defined as an 'unsigned long long' (unsigned 64-bit integral value) and is little-endian.

Timestamp values are taken from a counter that is incremented at each tick of the acquisition clock.

3.15.6 | Structure: PX14S_BUFNODE

This structure is used to describe an element in a DMA buffer chain. A DMA buffer chain is an array of DMA buffers, allocated by calling AllocateDmaBufferChainPX14.

Structure definition

```
typedef struct _PX14S_BUFNODE_tag
{
        px14_sample_t*      bufp;           // Address of DMA buffer
        unsigned int        buf_samples;    // Size of buffer in samples

} PX14S_BUFNODE;
```

Fields

bufp

The address of the DMA buffer represented by this node. For the last node in the DMA buffer chain, this field will be NULL.

buf_samples

The size of the DMA buffer, in samples, of the DMA buffer represented by this node. For the last node in the DMA buffer chain (identified by the NULL bufp field) this field will be sum total sample count of all buffers in the DMA buffer chain.

Remarks

When a DMA buffer chain is allocated, the library allocates an extra buffer descriptor at the end to indicate end-of-array. This descriptor is identified by the bufp value being NULL. For this special case node, the

buf_samples value is the sum total sample count of all buffers in the DMA buffer chain. This end-of-array marker allows software to pass around a DMA buffer chain object reference without having to include an array length parameter.

3.15.7 | Structure: PX14S_FILE_WRITE_PARAMS

This structure is used by a few of the PX14400 library functions that save PX14400 board data (either from PX14400 RAM or from a PX14400 recording session) to one or more files on the host system.

Structure definition

```
typedef struct _PX14S_FILE_WRITE_PARAMS_tag
{
        unsigned int            struct_size;        // Size of structure in bytes

        const char*             pathname;           // Destination file pathname
        const char*             pathname2;          // For dual channel
        unsigned int            flags;              // PX14FILWF_*
        unsigned int            init_bytes_skip;    // Initial bytes to skip in file
        unsigned long long      max_file_seg;       // Max file size in samples; binary only

        PX14_FILEIO_CALLBACK        pfnCallback;    // Optional progress callback
        uintptr_t                   callbackCtx;    // User-defined callback context

        unsigned int*           flags_out;          // Output flags; PX14FILWOUTF_*
        const char*             operator_notes;     // User-defined notes
        const char*             ts_filenamep;       // Timestamp filename override

} PX14S_FILE_WRITE_PARAMS;
```

Fields

struct_size

This field must be initialized to the size of the structure, in bytes, prior to using the structure with any of the library functions.

pathname

A pointer to a NULL-terminated char string that defines the pathname to use for output data.

pathname2

A pointer to a NULL-terminated char string that defines the pathname to use for channel 2 data. This parameter may be NULL.

flags

A set of flags that define how data will be saved. Currently defined flags are:

Library constant	Value	Interpretation
PX14FILWF_APPEND	0x00000001	Append if file already exists; default is to overwrite existing file.
PX14FILWF_ALWAYS_SKIPBYTES	0x00000002	Use *bytes_skip* even if appending to a file. By default, if appending to an existing file, the *bytes_skip* field is ignored.
PX14FILWF_SAVE_AS_TEXT	0x00000004	Save data as text; sample values will be whitespace-delimited. This is not a fast operation, so it's not recommended for use with PCIe Buffered Acquisition (streaming) recording operations.
PX14FILWF_NO_UNBUFFERED_IO	0x00000008	Windows: Don't try to use fast, unbuffered IO. If this flag is not set then the library may try to use the FILE_FLAG_NO_BUFFERING flag when creating the file. Using this flag can significantly improve file IO performance but requires file access lengths/offsets to be aligned to the underlying volume's sector size. If the library cannot guarantee these requirements, then normal, buffered file IO is used. See Win32 CreateFile function for details on this.
PX14FILWF_ASSUME_DUAL_CHANNEL	0x00000010	Assume data is dual channel; applies to single-file text saves. If this flag is set and data is being saved to a single text file then channel 1 data will be listed in one column and channel 2 data will be in another column.
PX14FILWF_HEX_OUTPUT	0x00000020	Use hexadecimal output; applies to text saves.
PX14FILWF_DEINTERLEAVE	0x00000040	De-interleave data into separate files. If this flag is set then the library will split channel data into separate files. Channel 1 data will be written to the file specified by the pathname field. Channel 2 data will be written to the file specified by the pathname2 field. Either one of these parameters can be NULL if that channel is not needed.

Continued:

Library constant	Value	Interpretation
PX14FILWF_CONVERT_TO_SIGNED	0x00000080	Convert data to signed format before saving. When this flag is set, data will be converted to equivalent signed value before being written. Note that this conversion is done in the software, not the hardware so it may affect IO throughput. If this flag is not specified then native, unsigned data will be saved.
PX14FILWF_GENERATE_SRDC_FILE	0x00000100	Generate a Signatec Recorded Data Context (SRDC) file for each output file.
PX14FILWF_EMBED_SRDC_AS_AFS	0x00000200	Embed Signatec Recorded Data Context (SRDC) information in data file as NTFS alternate file stream (:SRDC).
PX14FILWF_SAVE_TIMESTAMPS	0x00000400	Save PX14400 timestamp data in external file.
PX14FILWF_TIMESTAMPS_AS_TEXT	0x00000800	Save timestamps as newline-delimited text; use with PX14FILWF_SAVE_TIMESTAMPS flag.
PX14FILWF_USE_TS_FIFO_OVFL_MARKER	0x00001000	Use timestamp FIFO overflow marker: F1F0F1F0F1F0F1F0 F1F0F1F0F1F0F1F0.
PX14FILWF_ABORT_OP_ON_TS_OVFL	0x00002000	Abort and fail operation if timestamp FIFO overflows; meant to be used with recording sessions.

init_byte_skip

Specifies the number of bytes to skip in the file, before writing the first chunk of data. This can be used to have the library leave an application-defined blank area of the file than can be used for application-specific header data. The data in the skipped region will be all zeroes. Pass zero if you do not want to skip any bytes in the output file(s).

max_file_seg

In some applications, it is preferable to write to multiple, statically-sized files rather than one large file. This member is used to specify this behavior. If this member is zero, then all data is written to a single file. (Or, in the case of dual channel data, two files.)

If this member is non-zero, then it is the maximum size, in samples, that the library will write before creating a new file. In this case, the pathname(s) that are specified are used as a base for the resultant files that are created by the library routine. The library uses the given pathname appended with an underscore ('_') and an increasing number, beginning with 0.

This parameter is only considered when writing binary files. When writing a text file, this parameter is ignored.

pfnCallback

If non-NULL, this member is the address of a user-defined callback function that the library will call prior to writing each block of data to file. This callback function is intended to give applications a chance to handle progress updates. This callback function is documented in the Callback Function: PX14_FILEIO_CALLBACK section.

callbackCtx

An application-defined value that is passed to user-defined callback function.

flags_out

The library will write output flags to this member to convey additional information about the operation. NOTE: If this structure is being used for a recording session, use the GetRecordingSessionOutFlagsPX14 library function to obtain this information after stopping the recording and before deleting the recording session. Currently defined flags are listed in the following table.

PX14400 Library Constant	Value	Interpretation
PX14FILWOUTF_TIMESTAMP_FIFO_OVERFLOW	0x00000001	Timestamp FIFO overflowed during write/record process.
PX14FILWOUTF_NO_TIMESTAMP_DATA	0x00000002	No timestamp data was available during the operation.

operator_notes

A pointer to a NULL-terminated string that defines a user-defined operator note that will be saved when generating Signatec Recorded Data Context (SRDC) for the acquisition data being saved. SRDC data is generated when either PX14FILWF_GENERATE_SRDC_FILE or PX14FILWF_EMBED_SRDC_AS_AFS is specified in the flags field.

ts_filenamep

An optional override for the timestamp data file. This member is only considered when the PX14FILWF_SAVE_TIMESTAMPS flag is set in the flags member. See remarks below for details on how the timestamp data file is named.

Remarks

PX14400 Timestamps

If the PX14FILWF_SAVE_TIMESTAMPS flag is set in the flags member then the library will automatically read and save timestamp data from the PX14400 timestamp FIFO during the data recording. Timestamp data will be inserted into the timestamp FIFO by the PX14400 based on the rules defined by the current timestamp mode, which is specified via the SetTimestampModePX14 library function.

The *ts_filenamep* member may be used to specify the timestamp data file pathname. Or, if *ts_filenamep* is NULL, the library will name the timestamp data file as follows:

- If the *pathname* member is not NULL, *pathname* will be used as the base pathname.
- If the *pathname* member is NULL and the *pathname2* member is not NULL, *pathname2* will be used as the base pathname.
- If both *pathname* and *pathname2* are NULL, an error will be returned.

- Once the base pathname has been determined, the library will be appended with a '.px14ts' extension and this pathname will be used for the timestamp data file.
- Further, if the PX14FILWF_TIMESTAMPS_AS_TEXT flag is specified then a '.txt' extension will be appended. (The net effect of this is base *pathname* + ".px14ts.txt")

During the recording session, the library will create and manage a secondary thread that will be responsible for pulling and saving timestamp data.

PX14400 Timestamp Data File Format

If the PX14FILWF_TIMESTAMPS_AS_TEXT flag is set then the library will write all timestamp data to a text file with a new-line ('\n') between each timestamp value. The timestamp values will be written in decimal format.

If the PX14FILWF_TIMESTAMPS_AS_TEXT flag is not set then the library will write all timestamp data to a binary file. There is no header or auxiliary data written to the file; the first 64-bit unsigned integer of the file is the first timestamp value.

PX14400 Timestamp FIFO Overflow

By default, if the Timestamp FIFO overflows at any point, the operation will continue. There are two flags that can be specified to alter this behavior.

If the PX14FILWF_ABORT_OP_ON_TS_OVFL flag is specified during a recording session and a Timestamp FIFO overflow condition occurs, the recording will abort and end with an error code of SIG_PX14_TIMESTAMP_FIFO_OVERFLOW (-573). Timestamps will continue being recorded in this case until stopped by the main internal recording thread.

If the PX14FILWF_USE_TS_FIFO_OVFL_MARKER flag is specified and a Timestamp FIFO overflow condition occurs then the library will do the following:

- Read and save all timestamp values in the FIFO.
- Write a "timestamp overflow marker" to the timestamp data file. The timestamp overflow marker is two sequential timestamp values with the following value: 0xF1F0F1F0F1F0F1F0 (or 17,433,700,174,704,734,704 in decimal).

By inserting a timestamp overflow marker, third-party software can programmatically detect and handle FIFO overflow conditions.

It should be noted that if the Timestamp FIFO goes full, this does not guarantee that any timestamp data has been lost. If any timestamps are generated while the FIFO is full, they are lost.

Related Functions

CreateRecordingSessionPX14, ReadSampleRamFileBufPX14, ReadSampleRamFileFastPX14

3.15.8 | Structure: PX14S_REC_SESSION_PARAMS

This structure is used by the CreateRecordingSessionPX14 function to specify the various parameters that define how a PX14400 recording session will run.

Structure definition

```
typedef struct _PX14S_REC_SESSION_PARAMS_tag
{
        unsigned int            struct_size;            // Size of structure in bytes

        int                     rec_flags;              // PX14RECSESF_*
        unsigned long long      rec_samples;            // # samples or 0 for infinite
        unsigned int            acq_samples;            // For RAM acq recordings
        unsigned int            xfer_samples;           // Transfer size or 0 for autoset

        PX14S_FILE_WRITE_PARAMS*        filwp;          // Defines output file(s)

        px14_sample_t*          dma_bufp;               // Optional DMA buffer address
        unsigned                dma_buf_samples;        // Size of DMA buffer in samps

        // Data snapshot parameters; valid when PX14RECSESF_DO_SNAPSHOTS set

        unsigned int            ss_len_samples;         // Data snapshot length in samples
        unsigned int            ss_period_xfer;         // Snapshot period in DMA xfers
        unsigned int            ss_period_ms;           //  or in milliseconds

        PX14_REC_CALLBACK       pfnCallback;            // Optional callback
        void*                   callbackCtx;            // Context data for callback

} PX14S_REC_SESSION_PARAMS;
```

Fields

struct_size

This field must be initialized to the size of the structure, in bytes, prior to using the structure with any of the library functions.

rec_flags

A set of flags that define PX14400 recording session behavior. Currently defined flags are:

Library Constant	Value	Interpretation
PX14RECSESF_REC_PCI_ACQ	0x00000001	Do an PX14400 RAM-buffered PCIe acquisition recording.
PX14RECSESF_REC_RAM_ACQ	0x00000002	Do an PX14400 RAM acquisition/transfer recording.
PX14RECSESF_REC__MASK	0x0000000F	Mask for session recording type.
PX14RECSESF_DO_NOT_ARM	0x00000010	Do not auto arm recording; will be done with ArmRecordingSessionPX14.
PX14RECSESF_DO_SNAPSHOTS	0x00000020	Periodically obtain snapshots of recording data.

Continued:

Library constant	Value	Interpretation
PX14RECSESF_USE_UTILITY_BUFFERS	0x00000040	The library will use utility DMA buffers for recording data transfers. If no such buffers exist then they will be allocated. The length of the allocated buffers will be derived from the *xfer_samples* member. The buffers will not be freed when the recording session is closed. This gives application code a convenient way to reuse DMA buffers for subsequent recording sessions. Utility DMA buffers are automatically freed when the PX14400 device handle is closed. See EnsureUtilityDmaBufferPX14. NOTE: This bit requires PX14400 library version 1.14.1 or higher to have effect.
PX14RECSESF_DEEP_BUFFERING	0x00000080	The library will use a utility DMA buffer chain to buffer recording data. Using a DMA buffer chain allows for a deeper buffering of the data that is not available using the traditional recording mechanization, which uses a double-buffer approach. When this flag is set, the aggregate total size of the DMA buffer chain to use is specified by the dma_buf_samples field of this structure. If zero is specified for the dma_buf_samples field then the library will use a default DMA buffer chain size. At the time of this writing the default DMA buffer chain size is 128 MiS (134,217,728 samples). NOTE: This bit requires PX14400 library version 1.17.1 or higher to have effect.
PX14RECSESF_BOOT_BUFFERS_OKAY	0x00000100	The library will use boot-time DMA buffers for data transfers if they are available. This flag is only valid for RAM-buffered PCIe acquisition recordings. This flag has no effect if the PX14RECSESF_DEEP_BUFFERING flag is specified. When interrogating the boot-time DMA buffers, the library will first try to find a single buffer that it can use for data transfers. Next it will try to find two boot-time DMA buffers. If adequate boot-time buffers cannot be found then the standard buffering rules apply. NOTE: This bit requires PX14400 library version 2.7.1 or higher to have effect. Further, device driver 1.21.1 or higher is required for boot-time DMA buffer support.

See remarks below for more details on these flags.

rec_samples

Defines the total amount of samples to record. For dual channel recordings, this count is the total samples over both channels. If zero is specified for this parameter then the recording will run indefinitely. In this case, the AbortRecordingSessionPX14 can be used to stop the recording.

acq_samples

For RAM acquisition/transfer recordings, this parameter defines the total RAM acquisition size in samples. For dual channel acquisitions, this count is the total samples to acquire over both channels. This parameter has the same constraints as the samp_count parameter of the AcquireToBoardRamPX14 function. This parameter is ignored for PCIe-Buffered acquisition based recordings.

xfer_samples

Defines the DMA transfer length that should be used for PCIe Buffered acquisition based recordings. Pass zero to have the library determine what the transfer size should be. It is recommended to let the library determine the appropriate transfer size.

filwp

A pointer to a PX14S_FILE_WRITE_PARAMS structure that defines how data will be saved to persistent storage. See the Structure PX14S_FILE_WRITE_PARAMS section for details on this structure. Note that the PX14400 library supports many different ways of saving data and some of these (saving as text for instance) can be quite slow. For PCIe Buffered acquisition based recordings, if data cannot be written fast enough, FIFOs will overflow and data will be lost.

dma_bufp

A pointer to a previously allocated DMA buffer to use for transfer of acquisition data. If NULL is passed for this parameter, the library will allocate a DMA buffer to be used for the duration of the recording session.

dma_buf_samples

The size, in samples, of the DMA buffer pointed to by the *dma_bufp* field.

ss_len_samples

Defines the length, in samples, of the recording data snapshot. This field is only considered when the PX14RECSESF_DO_SNAPSHOTS flag is specified in the *rec_flags* field.

ss_period_xfer

The recording data snapshot update period, in DMA transfers. For long, non-segmented PCIe Buffered acquisition based recordings, a snapshot period in DMA transfers can be used to obtain synchronized recording snapshots over multiple independently running recording sessions. This field is only considered when the PX14RECSESF_DO_SNAPSHOTS flag is specified in the *rec_flags* field.

ss_period_ms

The recording data snapshot update period, in milliseconds. If the ss_period_xfer field is zero, the PX14400 library will default back to a simple, approximate-timed approach for recording data snapshot updates. This field is only considered when the PX14RECSESF_DO_SNAPSHOTS flag is specified in the *rec_flags* field.

pfnCallback

A pointer to an optional callback function that the library will periodically call during the recording session. This callback is intended to provide recording progress indications.

callbackCtx

An application specific value that is passed to the callback function specified by the *pfnCallback* field.

Remarks

The PX14400 library currently supports two types of recording sessions: PCIe Buffered Acquisition (streaming) and RAM Acquisition/Transfer.

PCIe Buffered Acquisition Recording Sessions

In a PCIe Buffered Acquisition recording session, data is buffered through PX14400 onboard RAM operating as a FIFO and acquired directly to the PCIe bus. Unlike RAM Acquisition/Transfer, this mode supports the transfer of data as it is being actively acquired. This is the primary method used to conduct high-speed sustained data streaming to the host PC system for optional real-time signal processing and/or real-time signal recording of the acquired data.

A PCIe Buffered Acquisition recording session is specified by setting the PX14RECSESF_REC_PCI_ACQ flag in the *rec_flags* field.

RAM Acquisition/Transfer Recording Sessions

A RAM Acquisition/Transfer recording session uses the PX14400's RAM Acquisition operating mode to acquire data into PX14400 RAM. Once the desired number of samples has been acquired, the acquisition is stopped and the data is transferred to the host system via DMA transfer. This process is repeated until the total number of samples has been recorded.

A RAM Acquisition/Transfer recording session is specified by setting the PX14RECSESF_REC_RAM_ACQ flag in the *rec_flags* field.

It is important to note that with this type of recording, multiple, independent acquisitions are performed. If a continuous uninterrupted stream of data is needed, then the PCIe Buffered Acquisition recording session should be used.

3.15.9 | Structure: PX14S_REC_SESSION_PROG

This structure is used by the GetRecordingSessionProgressPX14 function to contain progress details on the current recording session.

Structure definition

```
typedef struct _PX14S_REC_SESSION_PROG_tag
{
        unsigned int            struct_size;        // Size of structure in bytes

        int                     status;             // PX14RECSTAT_*
        unsigned long           elapsed_time_ms;    // Recording time in ms
        unsigned long long      samps_recorded;     // Total # of samples recorded thus far
        unsigned long long      samps_to_record;    // Total # of samples to be recorded
        unsigned int            xfer_samples;       // Transfer size in samples
        unsigned int            xfer_count;         // Transfer counter
        unsigned int            snapshot_counter;   // Current snapshot counter

// Valid when status == PX14RECSTAT_ERROR

        int                     err_res;            // SIG_*
        char*                   err_textp;          // Caller free via FreMemoryPX14

} PX14S_REC_SESSION_PROG;
```

Fields

struct_size
This field must be initialized to the size of the structure, in bytes, prior to using the structure with any of the library functions.

status
The status of the current recording progress. This can be any of the following values:

- PX14RECSTAT_IDLE (0) : Idle; recording not yet started.
- PX14RECSTAT_IN_PROGRESS (1) : Operation is in progress or waiting for trigger event.
- PX14RECSTAT_COMPLETE (2) : Operation complete or stopped by user.
- PX14RECSTAT_ERROR (3) : Operation ended due to error.

elapsed_time_ms
Current elapsed time of recording session in milliseconds.

samps_recorded
The total number of samples, over all channels, that have been recorded so far.

samps_to_record
The total number of samples, over all channels, that the recording session is configured to record. If this value is 0 then recording will run indefinitely.

xfer_samples

The data transfer size, in samples, used by the recording session. This is for informational purposes only.

xfer_count

The current data transfer count. Each time the recording session does a transfer of new acquisition data the data transfer counter is incremented.

snapshot_counter

The current data snapshot counter value. Each time the recording session collects a new recording data snapshot the snapshot counter is incremented by one. This value is only relevant if the recording session was configured to collect snapshots. See PX14S_REC_SESSION_PARAMS::rec_flags for more details. Data snapshots are obtained by calling GetRecordingSnapshotPX14.

err_res

This parameter is only relevant if the status member is PX14RECSTAT_ERROR. This is the library error codes (SIG_*) of the error that caused the recording session to prematurely end.

err_textp

This parameter is only relevant if the status member is PX14RECSTAT_ERROR. This is a library-allocated string containing a description of the error that occurred during the recording session. The caller must free the memory for this string when it is no longer needed by passing the address of the string to FreeMemoryPX14. If the PX14RECPROGF_NO_ERROR_TEXT flag is passed to GetRecordingSessionProgressPX14 then no error text will be generated and this member will be set to NULL.

3.15.10 | Structure: PX14S_RECORDED_DATA_INFO

This structure is used by the GetRecordedDataInfoPX14 function to contain details on acquisition data that was previously saved to drive storage.

Structure definition

```
typedef struct _PX14S_RECORDED_DATA_INFO_tag
{
        unsigned int        struct_size;        // Size of structure in bytes

        char                boardName[16];      // Name of board: "PX14400"
        unsigned int        boardSerialNum;     // Serial number

        unsigned int        channelCount;       // Channel count
        unsigned int        channelNum;         // Channel ID; single chan data
        unsigned int        sampSizeBytes;      // Sample size in bytes
        unsigned int        sampSizeBits;       // Sample size in bits
        int                 bSignedSamples;     // Signed or unsigned

        double              sampleRateMHz;      // Sampling rate in MHz
        double              inputVoltRngPP;     // Peak-to-peak input volt range
        int                 inputGain_dB;       // Input gain in dB

        unsigned int        segment_size;       // Segment size or zero
        int                 trigger_delta;      // Trig pos relative to start

        unsigned int        header_bytes;       // Size of app-specific header
        int                 bTextData;          // Data is text (versus binary)?
        int                 textRadix;          // Radix of text data (10/16)

} PX14S_RECORDED_DATA_INFO;
```

Fields

struct_size

This field must be initialized to the size of the structure, in bytes, prior to using the structure with any of the library functions.

boardName

The name of the board that generated the acquisition data. For PX14400 generated data this field will contain “PX14400”.

boardSerialNum

The serial number of the board that generated the acquisition data.

channelCount

The channel count of the underlying data. For PX14400 data this can be 1 or 2 depending on the active channel selection at the time of the recording.

channelNum

For multichannel data this value will be -1 (0xFFFFFFFF). For single channel data, this will be the zero-based channel index that acquired the data. (0 = channel 1, 1 = channel 2, etc.)

sampSizeBytes

The size of a single data sample in bytes. For PX14400 data, this will always be 2.

sampSizeBits

The size of a single data sample in bits. Though the PX14400 only has 14 effective bits, it will always use 16 for this field. Only the lower 14-bits are relevant (the upper two bits will always be zero).

bSignedSamples

This field will be non-zero if the underlying data samples are signed (±N). By default, PX14400 data samples are unsigned (0..N).

sampleRateMHz

The sampling rate, in MHz, used to acquire the underlying data.

inputVoltRngPP

The peak-to-peak input voltage range used to acquire the underlying data.

inputGain_dB

The gain, in dB, applied to the input data prior to conversion.

segment_size

This field is used to identify data acquired using segmented trigger mode. If this value is non-zero, then it is the segment size in samples. If this value is zero, then the underlying data is not segmented.

trigger_delta

This field is used to identify the trigger position relative to the start of the acquisition data (or segment if data is segmented). Certain data acquisition devices can begin storing data before or a certain number of acquisition clock cycles after the trigger event. If the value of this field is zero, the first sample is the trigger point. If the value of this field is positive then the trigger point is this many samples into the record. If the value of this field is negative then the trigger point is this many samples before the start of data, meaning a certain number of samples were ignored after the trigger event.

header_bytes

The number of bytes set aside for a user-defined file header. If this value is non-zero then acquisition data begins at *header_bytes* into the file.

bTextData

The underlying acquisition data is saved as text, not binary. Signatec software, such as the PX14400 Scope Application will not read text data.

textRadix

If data is saved as text, this field defines the radix of the output data: 10 for decimal, 16 for hexadecimal.

3.15.11 | Callback Function: PX14_FILEIO_CALLBACK

This item defines the signature of a callback function that is used when saving PX14400 data to file.

Form

```
int SomeCallbackFunction (HPX14 hBrd,
        uintptr_t callbackCtx,
        ec14_sample_t* sample_datap,
        int sample_count,
        const char* pathname,
        unsigned long long samps_in_file,
        unsigned long long samps_moved,
        unsigned long long samps_to_move);
```

Arguments

hBrd

A handle to the PX14400 device for which data is being saved. This handle is obtained by calling the ConnectToDevicePX14 function.

callbackCtx

A user-defined value that is ignored by the library.

sample_datap

A pointer to the chunk of data that is about to be written to file.

sample_count

The number of samples in the buffer pointed to by the *sample_datap* argument.

pathname

The pathname of the file currently being written to.

samps_in_file

The total number of samples written to the current file so far.

samps_moved

The total number of samples written/recorded so far. In cases where only a single file is being written, this will be the same value as the *samps_in_file* argument. In cases where board data is written to multiple files, this argument contains the total amount of data written over all files up to this point.

samps_to_move

The total number of samples that are to be written over the entire operation. If this value is zero, then the library is writing an infinite (e.g. manually stopped) amount of data.

Return Value

This function should return SIG_SUCCESS on success. Any other return value will result in the invoking library function to stop the current save operation (cleaning up as necessary) and returning that error code to the top-level caller.

4 | Appendix A – PX14400 Library Error Codes

The table below lists all of the currently defined PX14400 library error codes.

The GetErrorTextPX14 function can be used to generate a user-friendly string for any of these error codes.

Symbolic Constant	Error Code	Interpretation
SIG_SUCCESS	0	Operation successful.
SIG_PX14_QUASI_SUCCESSFUL	512	Operation was quasi-successful; one or more items failed and one or more items succeeded.
SIG_ERROR	-1	Generic error; platform's system error may provide more info.
SIG_INVALIDARG	-2	An invalid argument was specified.
SIG_OUTOFBOUNDS	-3	An argument is out of valid bounds.
SIG_NODEV	-4	Invalid board device.
SIG_OUTOFMEMORY	-5	Error allocating memory.
SIG_DMABUFALLOCFAIL	-6	Error allocating a DMA buffer.
SIG_NOSUCHBOARD	-7	Board with given serial or ordinal number not found.
SIG_NT_ONLY	-8	This feature is only available on Windows NT platforms.
SIG_INVALID_MODE	-9	Invalid operation for current operating mode.
SIG_CANCELLED	-10	Operation was cancelled by user.
SIG_PX14__FIRST	-512	First PX14400-specific error code value.
SIG_PX14_NOT_IMPLEMENTED	-512	This operation is not currently implemented.
SIG_PX14_INVALID_HANDLE	-513	An invalid PX14400 device handle (HPX14) was specified.
SIG_PX14_BUFFER_TOO_SMALL	-514	A specified buffer is too small.
SIG_PX14_INVALID_ARG_1	-515	Argument 1 is invalid.
SIG_PX14_INVALID_ARG_2	-516	Argument 2 is invalid.
SIG_PX14_INVALID_ARG_3	-517	Argument 3 is invalid.
SIG_PX14_INVALID_ARG_4	-518	Argument 4 is invalid.
SIG_PX14_INVALID_ARG_5	-519	Argument 5 is invalid.
SIG_PX14_INVALID_ARG_6	-520	Argument 6 is invalid.
SIG_PX14_INVALID_ARG_7	-521	Argument 7 is invalid.
SIG_PX14_INVALID_ARG_8	-522	Argument 8 is invalid.
SIG_PX14_XFER_SIZE_TOO_SMALL	-523	Requested transfer size is too small.
SIG_PX14_XFER_SIZE_TOO_LARGE	-524	Requested transfer size is too large.
SIG_PX14_INVALID_DMA_ADDR	-525	Given address is not part of a DMA buffer.
SIG_PX14_WOULD_OVERRUN_BUFFER	-526	Operation would overrun given buffer.
SIG_PX14_BUSY	-527	Device is busy; try again later.
SIG_PX14_INVALID_CHAN_IMP	-528	Incorrect function for board's channel implementation.
SIG_PX14_XML_MALFORMED	-529	Invalid XML data was encountered.
SIG_PX14_XML_INVALID	-530	XML data was well formed, but not valid.
SIG_PX14_XML_GENERIC	-531	Generic XML related error.
SIG_PX14_RATE_TOO_FAST	-532	The specified rate is too fast.
SIG_PX14_RATE_TOO_SLOW	-533	The specified rate is too slow.
SIG_PX14_RATE_NOT_AVAILABLE	-534	The specified frequency is not available.
SIG_PX14_UNEXPECTED	-535	An unexpected error occurred; debug builds will have failed assertion.

Continued:

Symbolic constant	Error Code	Interpretation
SIG_PX14_SOCKET_ERROR	-536	A socket error occurred.
SIG_PX14_NETWORK_NOT_READY	-537	Network subsystem is not ready for network communication.
SIG_PX14_SOCKETS_TOO_MANY_TASKS	-538	Limit on number of tasks/processes using sockets has been reached.
SIG_PX14_SOCKETS_INIT_ERROR	-539	Generic sockets implementation start up failure.
SIG_PX14_NOT_REMOTE	-540	Not connected to a remote PX14400 device.
SIG_PX14_TIMED_OUT	-541	Operation timed out.
SIG_PX14_CONNECTION_REFUSED	-542	Connection refused by service; service may not be running.
SIG_PX14_INVALID_CLIENT_REQUEST	-543	Received an invalid client request.
SIG_PX14_INVALID_SERVER_RESPONSE	-544	Received an invalid server response.
SIG_PX14_REMOTE_CALL_RETURNED_ERROR	-545	Remote service call returned with an error.
SIG_PX14_UNKNOWN_REMOTE_METHOD	-546	Undefined method invoked on remote server.
SIG_PX14_SERVER_DISCONNECTED	-547	Server closed the connection.
SIG_PX14_REMOTE_CALL_NOT_AVAILABLE	-548	Remote call for this operation is not implemented or available.
SIG_PX14_UNKNOWN_FW_FILE	-549	Unknown firmware file type.
SIG_PX14_FIRMWARE_UPLOAD_FAILED	-550	Firmware upload failed.
SIG_PX14_INVALID_FW_FILE	-551	Invalid firmware upload file.
SIG_PX14_DEST_FILE_OPEN_FAILED	-552	Failed to open destination file.
SIG_PX14_SOURCE_FILE_OPEN_FAILED	-553	Failed to open source file.
SIG_PX14_FILE_IO_ERROR	-554	File IO error.
SIG_PX14_INCOMPATIBLE_FIRMWARE	-555	Firmware is incompatible with PX14400.
SIG_PX14_UNKNOWN_STRUCT_VER	-556	Unknown structure version specified to library function (X::struct_size).
SIG_PX14_INVALID_REGISTER	-557	An invalid hardware register read/write was attempted.
SIG_PX14_FIFO_OVERFLOW	-558	An internal FIFO overflowed during acquisition; couldn't keep up with data rate.
SIG_PX14_DCM_SYNC_FAILED	-559	PX14400 firmware could not synchronize to acquisition clock.
SIG_PX14_DISK_FULL	-560	Could not write all data; disk is full.
SIG_PX14_INVALID_OBJECT_HANDLE	-561	An invalid object handle was used.
SIG_PX14_THREAD_CREATE_FAILURE	-562	Failed to create a thread.
SIG_PX14_PLL_LOCK_FAILED	-563	Phase lock loop (PLL) failed to lock; clock may be bad.
SIG_PX14_THREAD_NOT_RESPONDING	-564	Recording thread is not responding.
SIG_PX14_REC_SESSION_ERROR	-565	A recording session error occurred.
SIG_PX14_REC_SESSION_CANNOT_ARM	-566	Cannot arm recording session; already armed or stopped.
SIG_PX14_SNAPSHOTS_NOT_ENABLED	-567	Snapshots not enabled for given recording session.
SIG_PX14_SNAPSHOT_NOT_AVAILABLE	-568	No data snapshot is available.
SIG_PX14_SRD_FILE_ERROR	-569	An error occurred while processing .SRD file or stream-embedded SRD data.

Continued:

Symbolic constant	Error Code	Interpretation
SIG_PX14_NAMED_ITEM_NOT_FOUND	-570	Named item could not be found.
SIG_PX14_CANNOT_FIND_SRDC_DATA	-571	Could not find Signatec Recorded Data Context info.
SIG_PX14_NOT_IMPLEMENTED_IN_FIRMWARE	-572	Feature is not implemented in current firmware version; upgrade firmware.
SIG_PX14_TIMESTAMP_FIFO_OVERFLOW	-573	Timestamp FIFO overflowed during recording.
SIG_PX14_CANNOT_DETERMINE_FW_REQ	-574	Cannot determine which firmware needs to be uploaded; update software.
SIG_PX14_REQUIRED_FW_NOT_FOUND	-575	Required firmware not found in firmware update file.
SIG_PX14_FIRMWARE_IS_UP_TO_DATE	-576	Loaded firmware is up to date with firmware update file.
SIG_PX14_NO_VIRTUAL_IMPLEMENTATION	-577	Operation not implemented for virtual devices.
SIG_PX14_DEVICE_REMOVED	-578	PX14400 device has been removed from system.
SIG_PX14_JTAG_IO_ERROR	-579	JTAG IO error; JTAG is the interface used to upload and verify device firmware.
SIG_PX14_ACCESS_DENIED	-580	Access denied.
SIG_PX14_PLL_LOCK_FAILED_UNSTABLE	-581	Phase lock loop (PLL) failed to lock; lock not stable.
SIG_PX14_UNREACHABLE	-582	Item could not be set with the given constraints.
SIG_PX14_INVALID_MODE_CHANGE	-583	Invalid operating mode change; all changes must go or come from Standby.
SIG_PX14_DEVICE_NOT_READY	-584	Underlying device is not yet ready; try again shortly.
SIG_PX14_ALIGNMENT_ERROR	-585	A parameter is not aligned properly.
SIG_PX14_INVALID_OP_FOR_BRD_CONFIG	-586	Invalid operation for current board configuration.
SIG_PX14_UNKNOWN_JTAG_CHAIN	-587	Unknown JTAG chain configuration; out-of-date software or hardware error.
SIG_PX14_SLAVE_FAILED_MASTER_SYNC	-588	Slave device failed to synchronize to master device.
SIG_PX14_NOT_IMPLEMENTED_IN_DRIVER	-589	Feature is not implemented in current driver version; upgrade software.
SIG_PX14_TEXT_CONV_ERROR	-590	An error occurred while converting text from one encoding to another.
SIG_PX14_INVALIDARG_NULL_POINTER	-591	An invalid pointer was specified.
SIG_PX14_INCORRECT_LOGIC_PACKAGE	-592	Operation not available in current logic package.
SIG_PX14_CFG_EEPROM_VALIDATE_ERROR	-593	Configuration EEPROM validation failed.
SIG_PX14_RESOURCE_ALLOC_FAILURE	-594	Failed to allocate or create a system object such as semaphore, mutex, etc.
SIG_PX14_OPERATION_FAILED	-595	Operation failed; additional context information available via GetErrorTextPX14.
SIG_PX14_BUFFER_CHECKED_OUT	-596	Requested buffer is already checked out.
SIG_PX14_BUFFER_NOT_ALLOCATED	-597	Requested buffer does not exist.

5 | Appendix B – PX14400A Specifications

External Signal Connections

Analog Input, Channel 1	: SMA
Analog Input, Channel 2	: SMA
Clock Input	: SMA
Trigger Input	: SMA
Digital Input/Output	: SMA

Analog Inputs – Amplifier Front End

Full-Scale Volt. Ranges	: 200mV to 3.5V peak-peak (in 1 dB increments)
Impedance	: 50 ohms
Bandwidth	: 100 kHz to 200 MHz (Bessel filter)
Coupling	: AC
SNR (1 – 200 MHz)	: 65 dB
SFDR (@ 100 MHz)	: 83 dB

Analog Inputs – Transformer Front End

Full-Scale Volt. Ranges	: 1.1V peak-peak
Impedance	: 50 ohms
Bandwidth	: 500 kHz to 400 MHz
Coupling	: AC
SNR (1 – 200 MHz)	: 68 dB
SFDR (@ 100 MHz)	: 80 dB

External Trigger

Signal Type	: digital, LVCMOS signal level
Impedance	: >10k ohms
Bandwidth	: 50 MHz

Internal Synthesized Clock

Frequency Range	: 20 MHz to 400 MHz
Un-settable Range	: 277 MHz to 308 MHz
Resolution	: better than ±10 PPM
Accuracy	: better than ±5 PPM

External Clock

Signal Type	: sine wave or square wave
Coupling	: AC
Impedance	: 50 ohms
Frequency	: 20 MHz to 400 MHz
Amplitude	: 100mV to 2.0V peak-peak
Clock Dividers	: 1 to 20

Post ADC Clock Divider

Divider Settings	: 1, 2, 4, 8, 16, 32

Reference Clock

Internal	: 10 MHz, ±5 PPM max.
External	: 10 MHz, ±50 PPM max. (required for lock)

Digital Input/Output

Type	: TTL logic level
Max. Frequency	: 200 MHz
Connection	: 50 ohms to FPGA I/O
Output Modes	: 0V, Synchronized Trigger, ADC Clock ÷ 2, 3.3V
Input Modes	: Digital pulse for timestamp request

Trigger Modes

Post Trigger	: single start trigger fills active memory
Segmented	: start trigger for each memory segment

Trigger Options

Pre-trigger Samples	: samples prior to trigger are stored; Single Channel: 8k max.; Dual Channel: 4k max. per channel
Trigger Delay Samples	: delay from trigger to data storage; Up to 64k digitizer clock cycles

Memory

Total Size for Acquisition	: 256 Megasamples (512 MB)
Segment Size	: Up to 128 Megasamples
Segment Re-Arm Time[1]	: 150 nanoseconds
Addressing	: DMA transfer from starting address

Power Requirements

+3.3V	: 3.3 Amps max.
+12V	: 1.0 Amps max.

Absolute Maximum Ratings

Analog Inputs	: ±4V
Trigger Input	: -0.2V to +4V DC
Clock Input	: 5V peak-peak
Operating Temperature	: +32°F to +122°F / 0°C to 50°C
Storage Temperature	: -4°F to +158°F / -20°C to +70°C
Operating Relative Humidity	: 10% to 90%, non-condensing
Operating Vibration	: 0.25 G, 5 Hz to 500 Hz
Operating Shock	: 2.5 G, 11 ms, ½ sine
Board Dimensions	: 7.5" L x 4.3" H x 0.75" W 190.5 mm L x 109.22 mm H x 19.05 mm W

Notes:

1. In segmented mode, time from the end of a segment until a trigger will be accepted to begin another segment acquisition.

Signatec is a product brand of
DynamicSignals LLC, an ISO 9001:2008 Certified Company

Data Sheet Revision 1.5 – 08/24/2015
Specifications are subject to change without notice.

6 | Appendix C – PX14400D Specifications

External Signal Connections

Analog Input, Channel 1	: SMA
Analog Input, Channel 2	: SMA
Clock Input	: SMA
Trigger Input	: SMA
Digital Input/Output	: SMA

Analog Inputs

Full-Scale Volt. Ranges	: 400mV and 1.2V peak-peak
Impedance	: 50 ohms
Bandwidth	: DC to 200 MHz (Bessel filter)
Coupling	: DC
SNR (1 – 200 MHz)	: 67 dB
SFDR (@ 25 MHz)	: 80 dB
SFDR (@ 100 MHz)	: 73 dB

External Trigger

Signal Type	: digital, LVCMOS signal level
Impedance	: >10k ohms
Bandwidth	: 50 MHz

Internal Synthesized Clock

Frequency Range	: 20 MHz to 400 MHz
Un-settable Range	: 277 MHz to 308 MHz
Resolution	: better than ±10 PPM
Accuracy	: better than ±5 PPM

External Clock

Signal Type	: sine wave or square wave
Coupling	: AC
Impedance	: 50 ohms
Frequency	: 20 MHz to 400 MHz
Amplitude	: 100mV to 2.0V peak-peak
Clock Dividers	: 1 to 20

Post ADC Clock Divider

Divider Settings	: 1, 2, 4, 8, 16, 32

Reference Clock

Internal	: 10 MHz, ±5 PPM max.
External	: 10 MHz, ±50 PPM max. (required for lock)

Digital Input/Output

Type	: TTL logic level
Max. Frequency	: 200 MHz
Connection	: 50 ohms to FPGA I/O
Output Modes	: 0V, Synchronized Trigger, ADC Clock ÷ 2, 3.3V
Input Modes	: Digital pulse for timestamp request

Trigger Modes

Post Trigger	: single start trigger fills active memory
Segmented	: start trigger for each memory segment

Trigger Options

Pre-trigger Samples	: samples prior to trigger are stored; Single Channel: 8k max.; Dual Channel: 4k max. per channel
Trigger Delay Samples	: delay from trigger to data storage; Up to 64k digitizer clock cycles

Memory

Total Size for Acquisition	: 256 Megasamples (512 MB)
Segment Size	: Up to 128 Megasamples
Segment Re-Arm Time[1]	: 150 nanoseconds
Addressing	: DMA transfer from starting address

Power Requirements

+3.3V	: 3.3 Amps max.
+12V	: 1.0 Amps max.

Absolute Maximum Ratings

Analog Inputs	: ±4V
Trigger Input	: -0.2V to +4V DC
Clock Input	: 5V peak-peak
Operating Temperature	: +32°F to +122°F / 0°C to 50°C
Storage Temperature	: -4°F to +158°F / -20°C to +70°C
Operating Relative Humidity	: 10% to 90%, non-condensing
Operating Vibration	: 0.25 G, 5 Hz to 500 Hz
Operating Shock	: 2.5 G, 11 ms, ½ sine
Board Dimensions	: 7.5” L x 4.3” H x 0.75” W 190.5 mm L x 109.22 mm H x 19.05 mm W

Notes:

1. In segmented mode, time from the end of a segment until a trigger will be accepted to begin another segment acquisition.

Signatec is a product brand of
DynamicSignals LLC, an ISO 9001:2008 Certified Company

Data Sheet Revision 1.3 – 08/24/2015
Specifications are subject to change without notice.

7 | Appendix D – PX14400D2 Specifications

External Signal Connections

Analog Input, Channel 1	: SMA
Analog Input, Channel 2	: SMA
Clock Input	: SMA
Trigger Input	: SMA
Digital Input/Output	: SMA

Analog Inputs

Full-Scale Volt. Ranges	: 200mV, 333mV, 600mV, 1V, 1.6V, 3V peak-peak
Impedance	: 50 ohms
Coupling	: DC
Bandwidth	
@ 220mV	: DC to 106 MHz (Bessel filter)
@ 33mV	: DC to 182 MHz (Bessel filter)
@ 600mV	: DC to 239 MHz (Bessel filter)
@ 1V	: DC to 130 MHz (Bessel filter)
@ 1.6V	: DC to 201 MHz (Bessel filter)
@ 3V	: DC to 248 MHz (Bessel filter)

External Trigger

Signal Type	: digital, LVCMOS signal level
Impedance	: >10k ohms
Bandwidth	: 50 MHz

Internal Synthesized Clock

Frequency Range	: 20 MHz to 400 MHz
Un-settable Range	: 277 MHz to 308 MHz
Resolution	: better than ±10 PPM
Accuracy	: better than ±5 PPM

External Clock

Signal Type	: sine wave or square wave
Coupling	: AC
Impedance	: 50 ohms
Frequency	: 20 MHz to 400 MHz
Amplitude	: 100mV to 2.0V peak-peak
Clock Dividers	: 1 to 20

Post ADC Clock Divider

Divider Settings	: 1, 2, 4, 8, 16, 32

Reference Clock

Internal	: 10 MHz, ±5 PPM max.
External	: 10 MHz, ±50 PPM max. (required for lock)

Digital Input/Output

Type	: TTL logic level
Max. Frequency	: 200 MHz
Connection	: 50 ohms to FPGA I/O
Output Modes	: 0V, Synchronized Trigger, ADC Clock ÷ 2, 3.3V
Input Modes	: Digital pulse for timestamp request

Trigger Modes

Post Trigger	: single start trigger fills active memory
Segmented	: start trigger for each memory segment

Trigger Options

Pre-trigger Samples	: samples prior to trigger are stored; Single Channel: 8k max.; Dual Channel: 4k max. per channel
Trigger Delay Samples	: delay from trigger to data storage; Up to 64k digitizer clock cycles

Memory

Total Size for Acquisition	: 256 Megasamples (512 MB)
Segment Size	: Up to 128 Megasamples
Segment Re-Arm Time[1]	: 150 nanoseconds
Addressing	: DMA transfer from starting address

Performance (dB)

Dual Channel Operation		Channel 1	Channel 2
SNR	:	64.070	63.881
SINAD	:	62.337	62.364
SFDR	:	69.312	70.221
2nd Harm	:	-69.312	-70.349
3rd Harm	:	-71.459	-71.130
THD	:	-67.213	-67.678

Definition of Terms

SNR: Signal to Noise Ratio: The ratio of the fundamental sinusoidal signal power to the noise power. For this data sheet noise is considered to be the power from all spectral components except for the fundamental signal, the first harmonic, and the second harmonic.

SINAD: Signal to Noise and Distortion: The ratio of the fundamental sinusoidal signal power to the total noise and distortion component power. In other words this is the ratio of the fundamental signal power to the measured power from the remainder of the detectable spectrum.

SFDR: Spurious Free Dynamic Range: The ratio of the fundamental sinusoidal power to the power of the next highest spurious signal. Normally the highest spurious signal is the second or third harmonic.

2nd Harm: Second Harmonic Distortion: The ratio of the power at twice the fundamental frequency to the power of the fundamental sinusoid.

3rd Harm: Third Harmonic Distortion: The ratio of the power at three times the fundamental frequency to the power of the fundamental sinusoid.

THD: Total Harmonic Distortion: The ratio of the total power of the second and third harmonics to the fundamental sinusoidal power.

Test Method

A filtered 12.0 MHz sine wave signal is applied to the channel 1 and channel 2 inputs. The digitizer clock setting is 400 MHz. The voltage range is 1.6V. Signal amplitude is set for 95% of full scale. Conduct averaging of a 4096 size FFT for 50 acquisition cycles. Performance measurements are made using a 4096 point FFT with a Blackman-Harris window. Signatec uses the first 10 bins to represent the DC term, 34 bins centered around the peak for the fundamental signal power, 9 bins centered at twice the fundamental for the second harmonic and 9 bins centered at three times the fundamental for the third harmonic. All other bins are considered to be noise.

Power Requirements

+3.3V	: 3.3 Amps max.
+12V	: 1.0 Amps max.

Absolute Maximum Ratings

Analog Inputs	: ±4V
Trigger Input	: -0.2V to +4V DC
Clock Input	: 5V peak-peak
Operating Temperature	: +32°F to +122°F / 0°C to 50°C
Storage Temperature	: -4°F to +158°F / -20°C to +70°C
Operating Relative Humidity	: 10% to 90%, non-condensing
Operating Vibration	: 0.25 G, 5 Hz to 500 Hz
Operating Shock	: 2.5 G, 11 ms, ½ sine
Board Dimensions	: 7.5" L x 4.3" H x 0.75" W 190.5 mm L x 109.22 mm H x 19.05 mm W

Notes:

1. In segmented mode, time from the end of a segment until a trigger will be accepted to begin another segment acquisition.

Signatec is a product brand of
DynamicSignals LLC, an ISO 9001:2008 Certified Company
Data Sheet Revision 1.12 – 08/24/2015
Specifications are subject to change without notice.