



EC14150 Operators Manual

Revision 1.4 – 05/12/2014

Copyright © 2014

All Rights Reserved

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
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

1 BEFORE USING THE EC14150.....	1
1.1 Package Contents	1
1.2 Unpacking and Handling.....	1
1.3 Checking for Damage.....	1
1.4 Warranty.....	1
1.5 System Requirements.....	1
1.6 Software	2
1.7 Physical Layout.....	2
2 FUNCTIONAL DESCRIPTION.....	3
2.1 Overview.....	3
2.2 Operating Modes	5
2.2.1 Off Mode	5
2.2.2 Standby Mode.....	5
2.2.3 RAM Acquisition Mode.....	6
2.2.4 PCI Transfer Mode.....	6
2.2.5 Continuous Record Mode.....	6
2.3 Analog Input Circuitry.....	7
2.3.1 Active Channels.....	7
2.3.2 Input Voltage Range	7
2.3.3 Pre-Trigger Samples FIFO.....	7
2.4 ADC Clocking	8
2.5 Active Memory Region	8
2.5.1 Start Sample.....	8
2.5.2 Sample Count.....	9
2.6 Triggering the EC14150.....	9
2.6.1 Trigger Modes	9
2.6.1.1 Post-Trigger Mode.....	9
2.6.1.2 Segmented Trigger Mode	9
2.6.2 Trigger Sources.....	9
2.6.2.1 External Trigger.....	10
2.6.2.2 Internal Trigger	10
2.6.2.3 Software Generated Trigger	10
2.6.3 Trigger Timing Options	10
2.6.3.1 Pre-trigger Samples.....	10
2.6.4 Minimizing Jitter - Synchronized Triggering.....	10
2.6.4.1 External Clock and Trigger	11
2.6.4.2 Synchronizing the Trigger via Clock Out.....	11
2.6.4.3 Synchronizing the Trigger via Trigger Out.....	11
2.7 EC14150 Data Transfers.....	11
2.7.1 'Fast' Data Transfers.....	11
2.7.2 'Buffered' Data Transfers.....	12
2.7.3 Asynchronous and Synchronous Data Transfers	12
2.8 EC14150 Acquisition Data Format.....	13
2.8.1 Sample Format.....	13
2.8.2 Multichannel Data Format.....	13
2.8.3 EC14150 Sample Data Files (*.rd16).....	14
2.8.4 Signatec Recorded Data Context Files (*.srdc).....	14
2.9 Digital IO	14
2.10 Power Management.....	15
3 EC14150 SOFTWARE DEVELOPMENT REFERENCE.....	16

3.1 Windows Software Installation	16
3.2 Linux Software Installation	16
3.3 Developing EC14150 Software	18
3.3.1 Setting up the Build Environment – Windows Platform	18
3.3.2 Setting up the Build Environment – Linux Platform	19
3.3.3 Library Functions That Use Character Strings	19
3.4 Library Utility Functions.....	20
3.4.1 GetErrorTextEC14.....	20
3.5 EC14150 Device Enumeration and Connection Management	22
3.5.1 ConnectToDeviceEC14.....	22
3.5.2 ConnectToVirtualDeviceEC14	23
3.5.3 DuplicateHandleEC14	24
3.5.4 DisconnectFromDeviceEC14	24
3.5.5 GetDeviceCountEC14	25
3.5.6 IsDevicePresentEC14	25
3.5.7 IsDeviceVirtualEC14.....	26
3.5.8 IsHandleValidEC14	27
3.6 EC14150 Device State and Configuration	28
3.6.1 GetAcqInProgressFlagEC14.....	28
3.6.2 GetActualAdcAcqRateEC14.....	28
3.6.3 GetBoardNameEC14.....	29
3.6.4 GetBoardRevisionEC14	30
3.6.5 GetEffectiveAcqRateEC14.....	31
3.6.6 GetFifoFullFlagEC14.....	31
3.6.7 GetFirmwareVersionEC14.....	32
3.6.8 GetHardwareRevisionEC14	33
3.6.9 GetOrdinalNumberEC14	33
3.6.10 GetPllLockStatusEC14.....	34
3.6.11 GetSampleRamSizeEC14.....	34
3.6.12 GetSamplesCompleteFlagEC14	35
3.6.13 GetSerialNumberEC14.....	35
3.6.14 GetVersionTextEC14.....	36
3.6.15 InAcquisitionModeEC14	37
3.6.16 InIdleModeEC14.....	37
3.6.17 ReadConfigEepromEC14.....	38
3.6.18 SetUserDataEC14 / GetUserDataEC14.....	38
3.6.19 ValidateConfigEepromEC14	39
3.6.20 WriteConfigEepromEC14	39
3.7 EC14150 Hardware Settings.....	41
3.7.1 GetInputVoltRangeFromSettingEC14	41
3.7.2 GetInputVoltageRangeVoltsChXEC14.....	41
3.7.3 IssueSoftwareTriggerEC14	42
3.7.4 SelectInputVoltRangeEC14	43
3.7.5 SetActiveChannelsEC14 / GetActiveChannelsEC14	44
3.7.6 SetAdcClockSourceEC14 / GetAdcClockSourceEC14	45
3.7.7 SetDcOffsetChXEC14 / GetDcOffsetChXEC14	46
3.7.8 SetDigitalIoEnableEC14 / GetDigitalIoEnableEC14.....	47
3.7.9 SetDigitalIoModeEC14 / GetDigitalIoModeEC14	47
3.7.10 SetExtClockDividerEC14 / GetExtClockDividerEC14	49
3.7.11 SetExternalClockRateEC14 / GetExternalClockRateEC14.....	50
3.7.12 SetInputVoltageRangeChXEC14 / GetInputVoltageRangeChXEC14	51
3.7.13 SetInternalClockRateEC14 / GetInternalClockRateEC14.....	53
3.7.14 SetOperatingModeEC14 / GetOperatingModeEC14	54
3.7.15 SetPostAdcClockDividerEC14 / GetPostAdcClockDividerEC14	55
3.7.16 SetPowerDownOverrideEC14 / GetPowerDownOverrideEC14.....	56

3.7.17	SetPreTriggerSamplesEC14 / GetPreTriggerSamplesEC14	57
3.7.18	SetSegmentSizeEC14 / GetSegmentSizeEC14	58
3.7.19	SetSampleCountEC14 / GetSampleCountEC14	59
3.7.20	SetStartSampleEC14 / GetStartSampleEC14	60
3.7.21	SetTriggerDirectionAEC14 / GetTriggerDirectionAEC14	61
3.7.22	SetTriggerDirectionBEC14 / GetTriggerDirectionBEC14	62
3.7.23	SetTriggerDirectionExtEC14 / GetTriggerDirectionExtEC14	63
3.7.24	SetTriggerLevelAEC14 / GetTriggerLevelAEC14	64
3.7.25	SetTriggerLevelBEC14 / GetTriggerLevelBEC14	65
3.7.26	SetTriggerModeEC14 / GetTriggerModeEC14	66
3.7.27	SetTriggerSourceEC14 / GetTriggerSourceEC14	67
3.8	Device Register State Functions	68
3.8.1	CopyHardwareSettingsEC14	68
3.8.2	ReadAllDeviceRegistersEC14	68
3.8.3	RefreshLocalRegisterCacheEC14	69
3.8.4	RewriteHardwareSettingsEC14	70
3.8.5	SetPowerupDefaultsEC14	70
3.9	Memory/DMA Buffer Allocation Routines	72
3.9.1	AllocateDmaBufferEC14	72
3.9.2	FreeDmaBufferEC14	73
3.9.3	FreeMemoryEC14	74
3.10	Data Acquisition Routines	75
3.10.1	AcquireToBoardRamEC14	75
3.10.2	BeginBufferedPciAcquisitionEC14	76
3.10.3	EndBufferedPciAcquisitionEC14	77
3.10.4	IsAcquisitionInProgressEC14	77
3.10.5	WaitForAcquisitionCompleteEC14	78
3.11	Data Transfer Routines	80
3.11.1	GetPciAcquisitionDataBufEC14	80
3.11.2	GetPciAcquisitionDataFastEC14	81
3.11.3	IsTransferInProgressEC14	82
3.11.4	ReadSampleRamFastEC14	83
3.11.5	ReadSampleRamBufEC14	84
3.11.6	ReadSampleRamDualChannelBufEC14	85
3.11.7	ReadSampleRamFileFastEC14	87
3.11.8	ReadSampleRamFileBufEC14	88
3.11.9	WaitForTransferCompleteEC14	88
3.12	Data Manipulation Routines	90
3.12.1	DeInterleaveDataEC14	90
3.12.2	InterleaveDataEC14	91
3.13	EC14150 Recording Session Management Routines	92
3.13.1	AbortRecordingSessionEC14	92
3.13.2	ArmRecordingSessionEC14	92
3.13.3	CreateRecordingSessionEC14	93
3.13.4	DeleteRecordingSessionEC14	94
3.13.5	GetRecordingSessionOutFlagsEC14	94
3.13.6	GetRecordingSessionProgressEC14	95
3.13.7	GetRecordingSnapshotEC14	96
3.14	Signatec Recorded Data Context (SRDC) Information	97
3.14.1	CloseSrdcFileEC14	99
3.14.2	EnumSrdcItemsEC14	100
3.14.3	GetRecordedDataInfoEC14	100
3.14.4	GetSrdcItemEC14	101
3.14.5	IsSrdcFileModifiedEC14	102
3.14.6	OpenSrdcFileEC14	103

3.14.7 RefreshSrdcParametersEC14	104
3.14.8 SaveSrdcFileEC14	105
3.14.9 SetSrdcItemEC14	105
3.15 EC14150 Library Data Types.....	107
3.15.1 Data Type: HEC14	107
3.15.2 Data Type: HEC14RECORDING	107
3.15.3 Data Type: HEC14SRDC.....	107
3.15.4 Data Type: ec14_sample_t.....	107
3.15.5 Structure: EC14S_FILE_WRITE_PARAMS	108
3.15.6 Structure: EC14S_REC_SESSION_PARAMS.....	112
3.15.7 Structure: EC14S_RECORDED_DATA_INFO.....	115
3.15.8 Callback Function: EC14_FILEIO_CALLBACK.....	117
4 APPENDIX A – EC14150A SPECIFICATIONS	118
5 APPENDIX B – EC14150D SPECIFICATIONS	119
6 APPENDIX C – EC14150 LIBRARY ERROR CODES	120
7 APPENDIX D – REVISION HISTORY	123

**IMPORTANT NOTICE
ON
HARDWARE COMPATIBILITY**

The EC14150 is an ExpressCard 54 mm product and is a PCI Local Bus compliant device that implements ExpressCard via a PCI Express (PCIe) x1 bus connection. As such the EC14150 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 EC14150 as well as storage of base address size requirements for EC14150 operation.

The host computer (most likely a laptop) that the EC14150 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 EC14150 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 EC14150 inoperable. It is beyond the ability of Signatec hardware or software to force a non-PnP system to operate an EC14150 board.

In addition to PnP requirements, the EC14150 must be operated in an ExpressCard compliant socket that can accommodate 54 mm sized ExpressCard devices. The EC14150 is not compatible with PCMCIA.

1 | Before Using the EC14150

1.1 | Package Contents

The EC14150 package will normally contain (as a minimum) the following:

- EC14150 circuit board assembly
- EC14150 Software Disk
- Two BNC to MMCX coaxial cables

1.2 | Unpacking and Handling

The EC14150, 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 EC14150's frame for any sign of physical damage such as dents in the frame during shipment. 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 EC14150 requires the following minimum hardware configuration:

For Windows XP/Vista (32- or 64-bit)/7 (32- or 64-bit):

- Intel Core Duo 2.0 GHz or higher CPU
- Minimum of 512 MB system RAM
- Plug-n-Play system BIOS
- One open ExpressCard 54 mm slot

DOS and Windows NT (3.x/4.x)/95/98/ME/2000 are **NOT** supported by the EC14150 software.

1.6 | Software

The main EC14150 software installation installs several software components on the host system:

- A kernel-mode driver which allows the operating system to communicate with the EC14150 hardware.
- A user-mode library that interfaces with the driver. All EC14150 software accesses the EC14150 hardware through this library. This library exports a number of functions that may be used by custom software to control the EC14150 hardware. This library is documented in detail in the [EC14150 Software Development Reference](#) section.
- Several example EC14150 applications that demonstrate how to write EC14150 client programs. These example programs are located in the Examples directory of the main EC14150 software installation folder.
- The EC14150 Scope Application. This is a full featured Windows application that operates EC14150 devices. It can be used to run data recordings, view acquisition data, upload firmware, and more. This program has its own manual, which is located in the Documentation directory.

1.7 | Physical Layout

The EC14150 is shown in Figure 1. Signals are input via the MMCX connectors on the bracket as shown. The upper connector is channel 1 signal input, the second connector is for channel 2 signal input, the third connector is the external trigger input, the fourth connector is the clock input (for external clock), and the fifth connector is the digital I/O. All of these connectors are labeled on the EC14150 back panel.

The EC14150 is a 54 mm sized ExpressCard product that implements the ExpressCard bus using PCIe x1. The EC14150 can transfer data to PC RAM continuously at rates up to 170 MB/s (depending on laptop performance).



Figure 1: EC14150 ExpressCard Digitizer

2 | Functional Description

2.1 | Overview

The EC14150 is a dual channel waveform capture board that provides a remarkable combination of high speed and high resolution sampling along with a very large memory capacity all in a very compact and low power consuming ExpressCard form factor. At 4.5 W maximum, the EC14150 represents one of the lowest power consuming digitizer cards ever created for its class. The EC14150 was designed to maximize the quality of the captured signal in terms of signal-to-noise ratio and spurious-free dynamic range over a very wide frequency range.

The EC14150A features two AC-Coupled analog signal input channels. Signal frequencies up to 200 MHz can be accurately captured either in baseband or in higher order Nyquist zones using under-sampling techniques.

The EC14150D features two DC-Coupled analog signal input channels. Signal frequencies from DC to 75 MHz can be accurately captured.

Warning: Though the EC14150 is a relatively low-power consuming product, it is also a very small form-factor with very little innate heat-sink capability. Leaving all circuits powered on for extended periods of time will cause the channel 1 and channel 2 inputs to become very hot (up to about 70 C in room temperature environments when no cables are attached). Though recessed inside the frame, be careful to not touch these MMCX connectors since they will be very hot to the touch. Connecting the provided MMCX-to-BNC cables to these connectors will help to substantially dissipate this heat, which we recommend. Also, when not using the EC14150, it is advised to utilize the provided power-down features of the product.

The EC14150 is an ExpressCard 54 mm sized compliant board equipped with standard 'Plug and Play' features common in PCI systems. The entire 512 MB memory may be used as an exceptionally large FIFO for acquiring data directly to the ExpressCard bus continuously nonstop (referred to as "Continuous Record Mode") or in the simpler 2-step block acquisition to RAM and transfer to PC modes. In either the Continuous Record Mode (where the 512 RAM FIFO is used) or Data Transfer Mode; the EC14150 card is capable of sustaining 170 megabyte/sec transfers over the ExpressCard bus interface. Significant test data has shown that recordings with this large 512 MB FIFO buffering the recording process can be continuous at up to 85 MSPS even when operating in traditional non real-time environments such as the Windows operating system.

The following figures are simplified functional representations for the EC14150 and will be referenced many times in the following descriptions of various board functions.

The primary differences between the EC14150A (AC-Coupling) and the EC14150D (DC-Coupling) block diagrams revolve around their analog front ends.

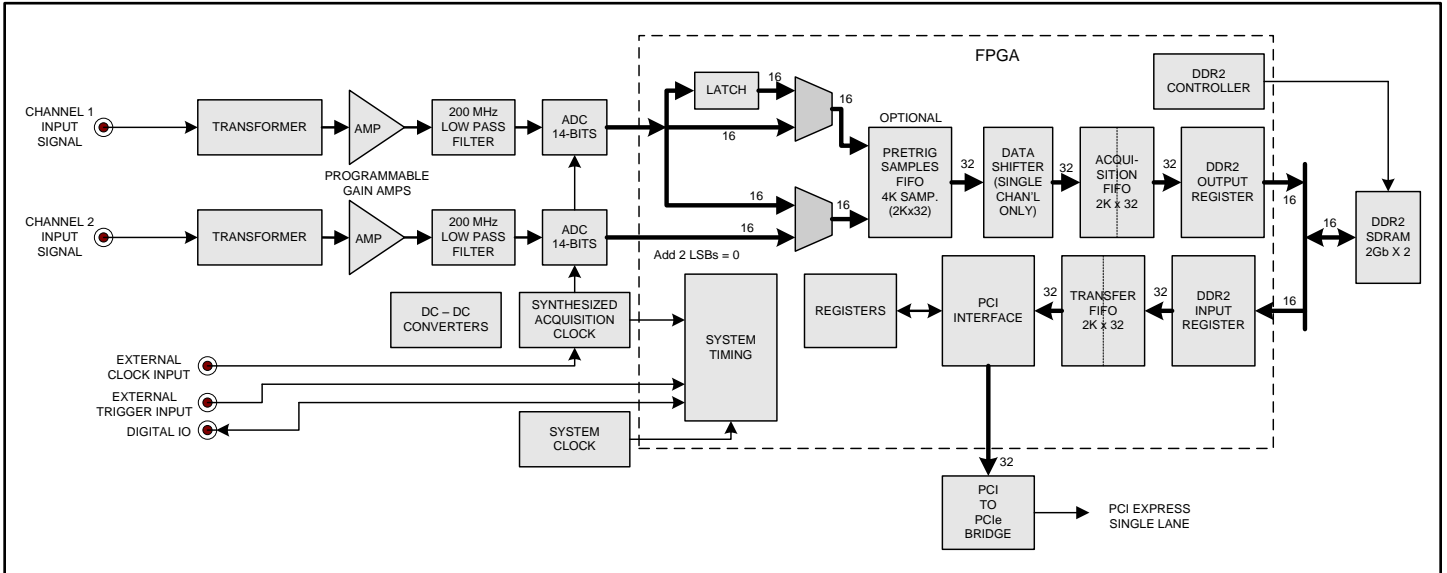


Figure 2: EC14150A (AC-Coupled) Functional Block Diagram

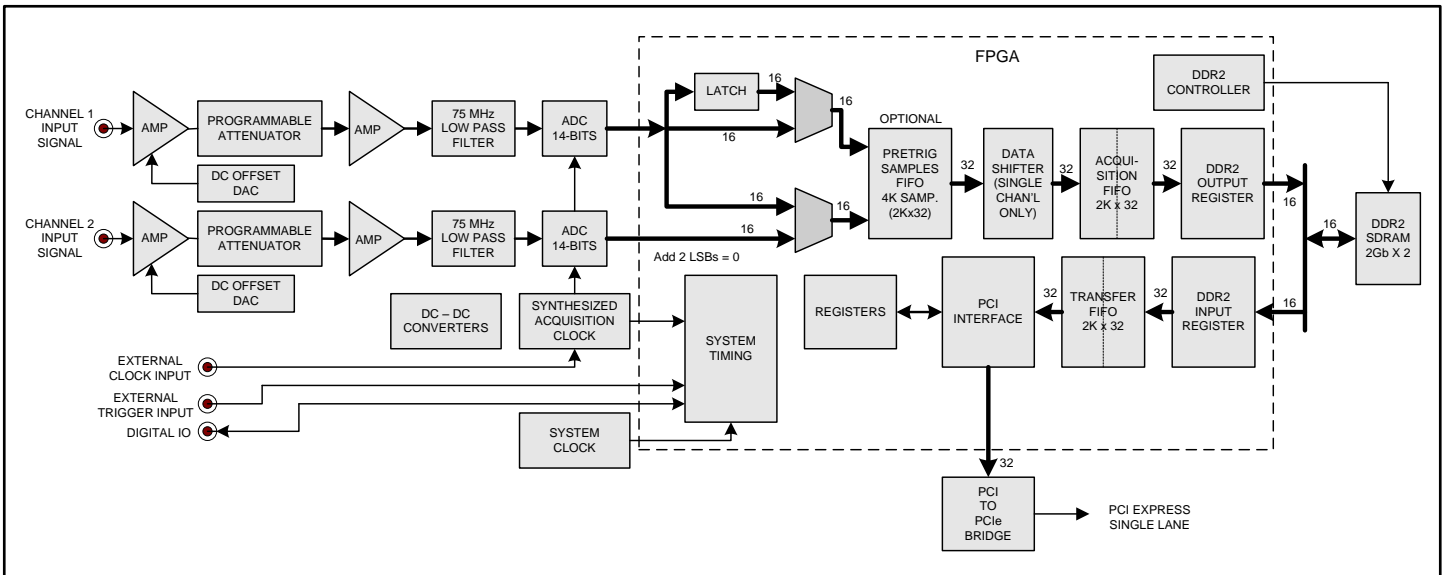


Figure 3: EC14150D (DC-Coupled) Functional Block Diagram

2.2 | Operating Modes

The EC14150 has several operating modes. The [SetOperatingModeEC14](#) and [GetOperatingModeEC14](#) functions may be used to set or get the current operating mode respectively. The EC14150 library wraps the functionality of many of these operating modes in higher-level functions that include automatic management of mode changes and other details. See [SetOperatingModeEC14](#) function for details.

The EC14150 operating modes are as follows:

Mode Number	Library Symbolic Constant	Operating Mode
0	EC14MODE_OFF	Off; device power-down
1	EC14MODE_STANDBY	Standby
2	EC14MODE_ACQ_RAM	RAM Acquisition
3	EC14MODE_RAM_READ_PCI	PCI Transfer Mode
4	EC14MODE_ACQ_PCI_BUF	Continuous Record Mode

All operating mode changes on the EC14150 should either be going to, or coming from Standby mode. This means that, for example, an operating mode change from RAM Acquisition mode directly to PCI Transfer mode is not permitted. The [SetOperatingModeEC14](#) function implementation automatically ensures that all operating mode changes go to or from Standby mode.

2.2.1 | Off Mode

When the device is in this mode, most EC14150 device hardware components are powered down. Specifically, the ADCs, front-end amplifiers, and internal clock generator components are all turned off. While in this operating mode, EC14150 hardware settings should not be changed since certain settings (acquisition clock settings for example) rely on components that may be powered down. The exception to this is that the [SetOperatingModeEC14](#) function may be called to put the board back into Standby mode, which will automatically power-up and restore all hardware settings.

See [Power Management](#) section for additional details about power management with respect to this operating mode.

2.2.2 | Standby Mode

This mode should be considered as the base mode for the EC14150. In this mode all circuitry, with the possible exception of the acquisition circuitry, is powered and the board is available for any command.

Placing the EC14150 into Standby mode will automatically cancel any active state of the board. This means any threads that are currently waiting for a DMA transfer ([WaitForTransferCompleteEC14](#)) or RAM acquisition ([WaitForAcquisitionCompleteEC14](#)) to complete will be resumed and return a SIG_CANCELLED (-10) error code. This includes threads running in other processes.

See [Power Management](#) section for additional details about power management with respect to this operating mode.

2.2.3 | RAM Acquisition Mode

This is the principal data acquisition mode. In this mode of acquisition, data is acquired directly to the EC14150 sample RAM. After the acquisition completes, the acquisition data can then be transferred to the host system for analysis.

There are a number of parameters that affect data acquisition behavior, all of which will be covered in subsequent sections. In general, the total amount of data to acquire is defined by the [Active Memory Region](#). The rate at which samples are acquired is defined by the various clock parameters, which are defined in the [ADC Clocking](#) section. Most users will want to synchronize the acquisition of the data to an external or internal event; this is controlled by the trigger mode and trigger timing parameters, discussed in the [Triggering the EC14150](#) section.

After the desired number of samples has been acquired, the EC14150 interrupts the host system. A program can wait (with an optional timeout) for the acquisition complete interrupt by calling the [WaitForAcquisitionCompleteEC14](#) library function.

Once data has been acquired to RAM, it can be transferred to the system with one of the [EC14150 Data Transfer](#) functions. It is not possible to transfer acquisition data while in RAM acquisition mode. Simultaneous acquisition and transfer is available with the Continuous Record mode, described below.

The [AcquireToBoardRamEC14](#) library function encapsulates a synchronous or asynchronous RAM acquisition operation.

2.2.4 | PCI Transfer Mode

This is the mode used to transfer acquisition data from the EC14150 RAM to host PC memory. This is the primary method of obtaining data previously acquired with the RAM Acquisition mode.

This mode should not be explicitly set by EC14150 client applications. Rather, higher-level library functions that manage the data transfer details should be used:

- [ReadSampleRamFastEC14](#)
- [ReadSampleRamBufEC14](#)
- [ReadSampleRamDualChannelBufEC14](#)
- [ReadSampleRamFileFastEC14](#)
- [ReadSampleRamFileBufEC14](#)

See the [EC14150 Data Transfers](#) section for more details on EC14150 data transfers.

2.2.5 | Continuous Record Mode

This mode is also referred to as PCI Buffered Acquisition mode. In this mode of operation the entire on-board RAM may be utilized as a giant FIFO to prevent data from being lost due to periodic hold-off of the PCIe data flow due to other system traffic. The maximum acquisition rate for a PCIe acquisition is dependent upon the hardware of the host system and other PCIe bus traffic. If the acquisition rate is higher than the PCIe data rate then it is possible that the on-board FIFOs could overflow and data could be lost. A FIFO-full flag is available to

be read at the end of the transfer to check for this FIFO overflow condition. The [GetFifoFullFlagEC14](#) library function may be used to read this flag.

Once the EC14150 is put into the PCI Buffered Acquisition mode and a trigger has been received, data from the ADC will flow through the RAM buffer to the PCIe bus. While in this mode, the [GetPciAcquisitionDataFastEC14](#) library function is repeatedly called to obtain fresh acquisition data.

The EC14150 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 [EC14150 Recording Session Management Routines](#) section for details on this feature.

2.3 | Analog Input Circuitry

2.3.1 | Active Channels

The EC14150 incorporates two identical analog input channels with 50 Ω impedance that are hardware configured for either AC-coupling (EC14150A) or DC-coupling (EC14150D). ADC data can be captured in dual channel or single channel mode. In single channel mode the entire signal memory can be used to capture data from channel 1 only. Use the library function [SetActiveChannelsEC14](#) to set the number of active channels.

2.3.2 | Input Voltage Range

Input voltage range is specified in peak-to-peak voltage (V_{p-p}). For example, a 6.34 V_{p-p} range starts at -3.17V and extends to +3.17V.

The AC-coupled EC14150A supports 31 unique input voltage ranges for each input channel ranging from 200 mV_{p-p} to 6.34 V_{p-p} in roughly 1dB steps. To achieve the various input voltage ranges the EC14150A uses a programmable amplifier to properly amplify or attenuate the input signal. At the amplifier/attenuator circuit output is a 200 MHz low pass filter to help reject higher frequency signals from aliasing into the data captured by the ADC.

The DC-coupled EC14150D supports full scale input voltage ranges of 250 mV_{p-p} , 500 mV_{p-p} , 1 V_{p-p} , and 2 V_{p-p} . Range switching is mechanized via the programmable attenuator. The amplifier has a fixed gain of 5 and is followed by a 3-pole Bessel low-pass filter with a cutoff at the maximum Nyquist limit of 75 MHz.

The [SetInputVoltageRangeCh1EC14](#) and [SetInputVoltageRangeCh2EC14](#) functions are used to select the input voltage range for channels 1 and 2 respectively. See function documentation for a table of input voltage range selection values.

2.3.3 | Pre-Trigger Samples FIFO

The Pre-trigger Samples FIFO can be thought of as a programmable length shift register with a maximum length of 1k samples. It is used to capture pre-trigger samples. Before a trigger is received, the digitizers are active and data is continuously written into the shift register. After receiving a trigger, data samples start to be written into the Acquisition FIFO. See the [Pre-Trigger Samples](#) section for more detail.

Data is written into the SDRAM via the Acquisition FIFO and read from RAM via the I/O FIFO. The double-data-rate RAM operates at a clock rate of 180 MHz so it has a bandwidth capability of 1440 MB/s, which exceeds the maximum data rate possible for simultaneously acquiring to the RAM at 150 MHz on both channels and transferring this data from the RAM.

2.4 | ADC Clocking

An internal synthesized clock is the primary clock source for the ADCs. This synthesized clock on the EC14150 allows for users to dial in almost any frequency possible for the onboard ADCs with resulting sampling clock performance that matches or beats most fixed crystal oscillator performance. The ADC clock can also be supplied from the external clock input connector.

The synthesizer can generate any frequency from 45 to 128 MHz and most frequencies from 128 to 150 MHz. There are a few “hole” frequency ranges that cannot be reached with the internal clock:

- 128.92 to 129.82 MHz (Δ 900 kHz)
- 140.63 to 142.80 MHz (Δ 2.17 MHz)

When the 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 signal frequency is typically greater than 100 MHz.

The synthesizer clock is locked to an internal 10 MHz reference clock. The internal reference clock is accurate to better than 5ppm. This sets the ADC clock accuracy to also be within 5ppm.

The [SetAdcClockSourceEC14](#) library function is used to select the clock that will be used for data conversion. When using the internal, synthesized clock the [SetInternalClockRateEC14](#) library function is used to select the desired synthesized clock frequency.

When using the external clock as the ADC clock, the EC14150 firmware needs to be aware of the actual external clock frequency so that it can properly synchronize to it. The [SetExternalClockRateEC14](#) library function is used to specify the external clock frequency.

If the external clock input is the ADC clock source, it may be divided by any integer value from 1 to 32. The [SetExtClockDividerEC14](#) library function is used to specify the external clock divider setting.

For all clock sources the effective digitization rate can be further reduced via a clock division emulation implemented by discarding samples after conversion. This second effective division can be set from 2 to 32 in factors of 2 and is set with the [SetPostAdcClockDividerEC14](#) library function.

2.5 | Active Memory Region

The active memory region defines the region of EC14150 RAM where subsequent RAM acquisitions or data transfers will originate. This region is defined by an offset (Start Sample) and length (Sample Count).

2.5.1 | Start Sample

The starting sample setting determines where the first data sample will be stored when acquiring data into RAM or when transferring data into RAM from one of the buses. It also sets the starting sample for reading data from RAM. The [SetStartSampleEC14](#) function can be used to set the starting sample.

Note: When the EC14150 is placed in Standby mode the address counter is automatically reset to zero. In most applications, data is acquired to the EC14150 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.

2.5.2 | Sample Count

This setting determines the total number of samples for subsequent RAM acquisition or PCI transfer operations. The Sample Count is set via the [SetSampleCountEC14](#) function.

2.6 | Triggering the EC14150

2.6.1 | Trigger Modes

Two trigger modes are implemented on the EC14150. The [SetTriggerModeEC14](#) function in the EC14150 function library may be used to select the trigger mode.

2.6.1.1 | Post-Trigger Mode

Post-Trigger mode is a single-shot mode in which a single trigger signal causes the EC14150 to start acquiring data. In this mode all (or most) of the data that is acquired occurs AFTER the trigger. Data acquisition will continue until a total number of samples are acquired. This is referred to as a single-shot mode because just one trigger is required to start and complete the acquisition. Notification of acquisition completion is indicated by the Samples Complete interrupt (see [WaitForAcquisitionCompleteEC14](#)).

2.6.1.2 | Segmented Trigger Mode

When Segmented Trigger mode is selected, each trigger event causes a fixed number of samples to be acquired. The EC14150 then stops and waits for another trigger event. This is repeated until a total number of samples have been acquired. The amount of samples acquired per trigger is called the segment size. To be meaningful, the segment size must always be smaller than the active memory size. Refer to the function [SetSegmentSizeEC14](#) for details concerning the segment sizes available and how they are selected.

2.6.2 | Trigger Sources

The trigger source defines where trigger events originate. The EC14150 has two types of trigger sources, internal and external.

Trigger events are only considered when the EC14150 is in one of the acquisition operating modes. Trigger events that occur prior to entering acquisition mode are not recognized. (A slight exception to this is the software-generated trigger, which may be issued prior to entering an acquisition mode. In this case acquisition will begin immediately upon entering acquisition mode.)

The trigger source is selected with the [SetTriggerSourceEC14](#) library function.

2.6.2.1 | External Trigger

When the external trigger is selected as the trigger source, a trigger event is defined by a TTL level pulse on the EC14150's external trigger input. Further, the board can be configured to trigger on the positive- or negative-going edge of the pulse with the [SetTriggerDirectionExtEC14](#) function.

2.6.2.2 | Internal Trigger

Triggering may also be set to be "internal" on either of the input channels. In this case a trigger event is defined by the selected channel's ADC value crossing one of two programmable trigger-levels in a given direction. The two trigger levels, A and B, are independent of one another.

The [SetTriggerLevelAEC14](#) and [SetTriggerLevelBEC14](#) are used to configure the A and B trigger levels, respectively.

2.6.2.3 | Software Generated Trigger

The EC14150 requires a trigger signal to start data acquisition. In addition to the trigger sources above, it is also possible to generate an acquisition trigger via software. The software trigger is functional whether internal or external triggering is selected. The [IssueSoftwareTriggerEC14](#) library function is used to generate a software trigger.

2.6.3 | Trigger Timing Options

In addition to the trigger modes described above, the following trigger option exists that can affect which samples are acquired relative to the trigger event. Data capture begins on detection of a trigger event. The following trigger option allows for starting the data capture slightly before the actual trigger event.

2.6.3.1 | Pre-trigger Samples

With this trigger option, data from the ADC is routed to a "data pipeline" of programmable length. The digitization and shifting of data through the pipeline is always active (while in the acquisition mode) so that the pipeline is full when a trigger occurs. The pipeline therefore holds pre-trigger data. When the trigger occurs, the output from the pipeline is written to RAM or the PCIe bus, depending on the current acquisition mode.

Data is stored until either the desired total samples is attained or, in the case of segmented trigger mode, the segment size limit is reached. The data written to the RAM (or output to the buses) consists of the defined number of pre-trigger samples immediately preceding the trigger, with the remaining number of samples occurring immediately after the trigger. This mode is useful for seeing "backward in time", i.e. seeing the waveform as it existed before the trigger signal occurred.

The amount of pre-trigger samples can be specified with the [SetPreTriggerSamplesEC14](#) function.

2.6.4 | 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.

2.6.4.1 | External Clock and Trigger

Supplying an external clock to the EC14150 as well as a trigger that is synchronized to that clock will eliminate all trigger jitter.

2.6.4.2 | 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 EC14150's internal digitizer clock. The clock output frequency will be equal to the digitizer clock frequency. The function [SetDigitalIoModeEC14](#) can be used to set the digital IO mode for clock out operation. The digital IO port is enabled with the [SetDigitalIoEnableEC14](#) function.

2.6.4.3 | 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 function [SetDigitalIoModeEC14](#) can be used to set the digital IO for synchronized trigger operation. The digital IO port is enabled with the [SetDigitalIoEnableEC14](#) function.

2.7 | EC14150 Data Transfers

Data transfers between the EC14150 and host system are implemented using Direct Memory Access (DMA) transfers. A DMA transfer is carried out using the system's DMA controller without intervention from the system processor. This is the only method available for burst transfers on the PCIe bus in a standard laptop or PC. The EC14150 library, driver, and hardware handle all aspects of doing DMA transfers.

The EC14150 library implements two general categories of data transfer functions: fast and buffered.

2.7.1 | 'Fast' Data Transfers

As previously stated, all data transfers between the EC14150 and host system are done via DMA transfer. In order to do a DMA transfer a DMA buffer must first be allocated. A DMA buffer is a special region of system RAM that is both physically contiguous and non-paged. 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 [AllocateDmaBufferEC14](#) library function is used to allocate a DMA buffer and map it into the address space of the current process. The [FreeDmaBufferEC14](#) is used to free the DMA buffer when it is no longer needed. Once the DMA buffer has been allocated, the EC14150 hardware can then transfer data directly to that buffer without any intermediate buffering by the system.

The library implements these types of transfers with the 'Fast' data transfer functions:

- [ReadSampleRamFastEC14](#)
- [ReadSampleRamFileFastEC14](#)
- [GetPciAcquisitionDataFastEC14](#)

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.

These ‘fast’ routines should be used when moving data from the EC14150 is speed critical, such as in PCI-buffered acquisition recordings.

2.7.2 | ‘Buffered’ Data Transfers

The ‘buffered’ data transfer functions allow arbitrary length/offset data transfers to arbitrary memory locations. 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 will result in a small degradation in transfer speed, but it does have the benefit that there are no offset/length alignment restrictions and any type of memory can be used for the data transfer. The ‘buffered’ data transfer functions are:

- [ReadSampleRamBufEC14](#)
- [ReadSampleRamDualChannelBufEC14](#)
- [ReadSampleRamFileBufEC14](#)
- [GetPciAcquisitionDataBufEC14](#)

These buffered routines can be used when speed is not an issue, such as transferring data after a RAM acquisition or saving data to disk.

2.7.3 | 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.

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 [WaitForTransferCompleteEC14](#) 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](#). Also, the AcqRecordManualEC14 example application (located in the Examples directory of the EC14150 software installation folder) is an example that uses this method.

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.

2.8 | EC14150 Acquisition Data Format

2.8.1 | Sample Format

The resolution of the onboard 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.

EC14150 data samples are little-endian, unsigned 16-bit integral values.

The EC14150 library uses the 'ec14_sample_t' data type to represent an EC14150 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 value 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 65532 is equivalent to the maximum input voltage range. A sample value of 16384 is equivalent to approximately 0V.

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 (e.g. 2, 4, 1, 0.200, etc...) in peak-to-peak voltage
- S is the sample value.

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

$$V_s = -1/2 + ((8192 / 65532) * 1) = -0.375V \text{ or } -375mV$$

2.8.2 | Multichannel Data Format

When acquiring dual-channel data, the EC14150 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 EC14150 library implements routines to interleave ([InterleaveDataEC14](#)) or de-interleave ([DeInterleaveDataEC14](#)) dual-channel data.

2.8.3 | EC14150 Sample Data Files (*.rd16)

Signatec software, such as the EC14150 Scope Application, as well as certain EC14150 library routines has the ability to save EC14150 acquisition data to a file. The native file format used by these entities is the RD16 file format. The RD16 moniker is derived from “Raw Data 16-bit”. RD16 files are identified by the ‘.rd16’ file extension.

RD16 files are binary files that contain only 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. This simple file format has two big advantages. First, it’s very fast to write these files since data is written to the file exactly as it is received from the EC14150. If the underlying file system (e.g. a high-speed RAID system) can keep up with the data rate, data can be streamed from the EC14150 card to the file at the full acquisition rate. The second advantage is that this file format is very generic which makes it easy for other software to get at the data. This includes custom software, or other software environments such as MATLAB.

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

The main disadvantage of the RD16 file format is that, since no context information is stored in the file, it may not be apparent what kind of data is in the file. Is it single or dual channel? What was the input voltage range? What is the sampling rate? To get around this problem, Signatec software can also be configured to generate a small auxiliary context file (XML format) that sits in the same directory as the RD16 file. This auxiliary file can contain information such as the channel count, input voltage range, sampling rate, source board, operator notes, or even user-defined data.

By convention, the name of the auxiliary data file is the full pathname of the 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 auxiliary data would be written to “C:\Data\Recordings\MyData.rd16.srdc”. The EC14150 library has routines that can be used to read and write these files.

SRDC files are saved in XML format so they can easily be read in by any XML-aware software.

See the [Signatec Recorded Data Context \(SRDC\) Information](#) section for more details on these files.

2.9 | Digital IO

The Digital IO MMCX connector on the EC14150 bracket can supply a selection of digital signals that may be used to synchronize external events or to monitor internal processes. Among the available signals are the digitizer clock and the synchronized trigger. See the function [SetDigitalIoModeEC14](#) for a list of available signals and how to select them.

The digital IO connector must be enabled before any digital input/output can be received/sent. The [SetDigitalIoEnableEC14](#) function is used to enable the digital IO port.

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 EC14150.

2.10 | Power Management

The EC14150 hardware and software attempt to minimize power consumption by powering down unused hardware components when the board is idle in Standby mode. This behavior can be overridden by turning on the power-down override, which is set with the [SetPowerDownOverrideEC14](#) function. By default, the power-down override is not set. When the power-down override is set, acquisition setup times will be faster (on the order of milliseconds) since hardware items are powered up and ready to go, but power consumption will be greater.

When the EC14150 is put into the Off operating mode, all unnecessary hardware components are powered down. The power-down override has no effect in this operating mode.

Hardware Component	Power Up/Down	Off Mode	Standby Mode	Controller
ADCs	Total	Off	Off ¹	Software
ADC output enables	Total	Off	Off	Software
Analog input amplifiers	Total	Off	Off	Software
Clock generator	Partial	Off	Off	Software
Internal acquisition logic	Partial	Off	On	Hardware
Digital IO	Total	Off	Unchanged	Software

¹ Powered down only if power-down override is not enabled.

3 | EC14150 Software Development Reference

3.1 | Windows Software Installation

To install the EC14150 Windows software and documentation just insert the Signatec Product Software CD into your CD-ROM 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.

This installation will install all EC14150 software and documentation to the folder selected during installation that includes EC14150 drivers, libraries, documentation, components, applications, utilities, and programming examples. Some of the more important items are:

EC14150 dynamic link library: this library is the interface to the EC14150 driver. User applications will use the functions exported by this library to connect their software to the EC14150 device. The library (EC14.dll) is installed to the product installation folder and the system library search path is updated to include this directory.

EC14150 Operators Manual: this is the document that you are currently reading and is installed to the Documentation\ folder of the install folder.

The EC14150 Scope application: this is a full-featured application that demonstrates how to use the various features of the EC14150. This application is also a great starting point to getting familiar with the EC14150. Please refer to the separate EC14150 Scope Application Manual for further details on this software application. The fully commented source code for this application is installed to the Source\Examples\EC14150 Scope Application\ folder of the install folder.

Various fully commented example source code projects demonstrating how to interface with the EC14150 located in the Source\ folder of the install folder.

3.2 | Linux Software Installation

The EC14150 software disk also contains EC14150 Linux Software with a tar file format of sig_ec14.X.Y.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 sig_ec14.X.Y.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 EC14150 driver
- doc : source for documentation regarding EC14150 development and usage
- examples : source for EC14150 example applications
- libsig_ec14 : source for EC14150 shared software function library
- olde : source for legacy driver load/unload scripts (should not be needed if PCIe hotplug subsystem if functioning as the EC14150 driver is automatically loaded and unloaded)
- util : source for example EC14150 utility applications and scripts; utility programs are installed to /user/bin so they are available globally and all EC14150 utilities begin with “ec14_”

Note that the EC14150 Linux software does not include a full EC14150 Scope Application utility. The EC14150 Scope Application utility is also available with Windows software.

Please read any Release Notes or README documents that may be installed in the root EC14 folder.

You must be running the kernel on which your EC14150 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 EC14 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 utility application running `'make util'`. This will compile all utility applications. This step needs to be done after Step 3 because the utilities are dependent on the EC14 shared library being installed.
- 5.) The EC14 drive module (`sig_ec14.ko`) should be automatically loaded when an EC14 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_ec14          34412  0      (the actual numbers will vary)
```

To explicitly load or unload the EC14 driver, there are scripts in the `olde/load_scripts` directory.

3.3 | Developing EC14150 Software

PC/laptop applications interface the EC14150 through the EC14 library. This library exports a variety of functions that you can use in your own programs. These functions are documented in the [C Library Functions](#) section. The following sections describe how to set up the library and program the EC14150.

3.3.1 | Setting up the Build Environment – Windows Platform

The EC14150 library is composed of three parts: the header file (ec14.h), the import library (EC14.lib), and the dynamic link library (EC14.dll). The header and import library are required to build EC14150 applications and the dynamic link library is required to run EC14150 applications.

The header file, ec14.h, is the C interface to the library. Your EC14150 applications will “#include” this file to define EC14150 data structures, function prototypes, constants, and macros. This file is installed to the include\ folder of the EC14150 install directory. It is recommended that you add this path to your compiler’s header file search path². Adding the header file to the include file search path will allow you to keep a single copy of ec14.h on your system and access it like a standard C header file. (That is, you can do a “#include <ec14.h>” instead of a “#include “Path-to-ec14.h”.”) The same header file is used for both 32- and 64-bit code.

The import library, EC14.lib (32-bit code) or EC14_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 EC14150 install directory. 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” EC14PP_NO_LINK_LIBRARY before including ec14.h.)

The dynamic link library, EC14.dll (32-bit code) or EC14_64.dll (64-bit code), is installed to the EC14150 software installation folder (C:\Program Files\Signatec\EC14150 by default). During software installation, the system library search path is updated to include this folder so the EC14150 library can be located when needed.

The 64-bit EC14150 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 EC14150 software and need to build for a 32-bit architecture.

² 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.

3.3.2 | Setting up the Build Environment – Linux Platform

The EC14150 library is composed of two files: the header file (ec14.h) and library binary (libsig_ec14150.so). Both files are required to build EC14150 applications. Only the library binary is required to run EC14150 applications.

The header file, ec14.h, is the C interface to the library. Your EC14150 applications will “#include” this file to define EC14150 data structures, function prototypes, constants, and macros. The main EC14150 software build process will put the ec14150 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 <ec14.h>” instead of a “#include “Path-to-ec14.h”.)

EC14150 applications must be linked with the EC14150 library. This is done by adding the “-lsig_ec14150” linker option to the gcc/g++ command line. For an example of this, refer to the Makefile for one of the EC14150 example applications.

3.3.3 | Library Functions That Use Character Strings

For library functions that accept or generate character strings (e.g. [GetErrorTextEC14](#)), 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 ‘AEC14’ and the UNICODE version function names are suffixed with ‘WEC14’.

Macros have been defined so that library users can write generic code that will work with either version. Consider GetErrorTextEC14 as an example. The library implements two versions of this function: GetErrorTextAEC14 and GetErrorTextWEC14. If the _UNICODE symbol is defined, then the GetErrorTextEC14 macro will expand to GetErrorTextWEC14, else it will default to GetErrorTextAEC14. In this manual library functions are documented using the character-size agnostic type TCHAR.

```
int GetErrorTextAEC14 (int res, char** bufpp, unsigned int flags);  
int GetErrorTextWEC14 (int res, wchar_t** bufpp, unsigned int flags);
```

```
#ifdef _UNICODE  
# define GetErrorTextEC14    GetErrorTextWEC14  
#else  
# define GetErrorTextEC14    GetErrorTextAEC14  
#endif
```

```
// Effective library function prototype:  
int GetErrorTextEC14 (int res, TCHAR** bufpp, unsigned int flags);
```

3.4 | Library Utility Functions

The functions in this section are generic library utility functions.

3.4.1 | GetErrorTextEC14

Form

```
int GetErrorTextEC14 (int res, TCHAR** bufpp, unsigned int flags, HEC14 hBrd = EC14_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 (EC14150-specific) SIG_EC14_* 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 [FreeMemoryEC14](#) library function

[in] *flags*

A set of flags that dictate the function behavior.

Flag	Interpretation
EC14ETF_IGNORE_SYSERROR (0x00000001)	Ignore system specific error information (Windows: GetLastError, Linux: errno)
EC14ETF_NO_SYSERROR_TEXT (0x00000002)	Do not generate any text for system specific error
EC14ETF_FORCE_SYSERROR (0x00000004)	Force inclusion of system error information even if it might not be relevant

[in] *hBrd*

A handle to the EC14150 device for which the error occurred. For some types of errors, the EC14150 handle used when the error was generated may contain additional context information. This parameter may be EC14_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

[DisconnectFromDeviceEC14](#), [ConnectToVirtualDeviceEC14](#)

3.5 | EC14150 Device Enumeration and Connection Management

The functions in this section involve managing EC14150 device connections and general EC14150 device handle management.

3.5.1 | ConnectToDeviceEC14

Form

```
int ConnectToDeviceEC14 (HEC14* phDev, unsigned int brdNum = 1);
```

Description

This function is used to establish a connection to an EC14150 data acquisition device, represented by an EC14150 device handle of type HEC14.

Parameters

[in] *pBrd*

A pointer to a HEC14 variable that will receive the EC14150 device handle. This EC14150 device handle is only valid in the current process. This handle should be treated as an opaque object; interpretation of the handle is EC14150 library specific.

[in] *brdNum*

This parameter is used to select which EC14150 in the system to connect to. If this value is in the range [1, EC14_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 EC14150 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 EC14150 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 EC14150 device in a non-intrusive manner.

Like other handle-based mechanisms, the actual value of an EC14150 device handle should be treated as an opaque value. The exception to this is a special value, EC14_INVALID_HANDLE, which is used to identify an invalid EC14150 device handle.

An application should close the connection to the EC14150 device by calling the DisconnectFromDeviceEC14 library function. Failure to disconnect from the device can result in memory leaks in the calling process.

Related Functions

[DisconnectFromDeviceEC14](#), [ConnectToVirtualDeviceEC14](#)

3.5.2 | ConnectToVirtualDeviceEC14

Form

```
int ConnectToVirtualDeviceEC14 (HEC14* phBrd, unsigned int serialNum, unsigned int brdNum);
```

Description

Establish a connection to a virtual (fake) EC14150 device.

Parameters

[in] *pBrd*

A pointer to a HEC14 variable that will receive the virtual EC14150 device handle.

[in] *serialNum*

The serial number to use for the virtual device. This can be any number; the EC14150 library ignores this value.

[in] *brdNum*

The board number to use for the virtual device. This can be any number; the EC14150 library ignores this value.

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 EC14150 data acquisition device. A virtual device is one that is not connected to any real EC14150 hardware. Virtual devices are mainly used for software development and debugging.

Related Functions

[DisconnectFromDeviceEC14](#), [ConnectToDeviceEC14](#)

3.5.3 | DuplicateHandleEC14

Form

```
int DuplicateHandleEC14 (HEC14 hBrd, HEC14* phNew);
```

Description

Duplicate an EC14150 device handle.

Parameters

[in] *hBrd*

The EC14150 device handle to duplicate. This handle must have been obtained from the current process.

[out] *phNew*

A pointer to a HEC14 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

[ConnectToDeviceEC14](#)

3.5.4 | DisconnectFromDeviceEC14

Form

```
int DisconnectFromDeviceEC14 (HEC14 hBrd);
```

Description

This function is used to release the handle for the EC14150 when it is no longer needed in the program. The function also frees any utility DMA buffers allocated by the library for the given EC14150 device.

Parameters

[in] *hBrd*

The EC14150 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 EC14_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 an EC14150 device handle will have no effect on the underlying hardware. This allows one to connect/disconnect to an EC14150 device in an unobtrusive manner. A slight exception to this is that the EC14150 driver may interact with the EC14150 hardware when all connections to a particular EC14150 (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 board into Standby or Off 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

[ConnectToDeviceEC14](#)

3.5.5 | GetDeviceCountEC14

Form

```
int GetDeviceCountEC14();
```

Description

Obtain a count of all EC14150 devices present in the local system.

Return Value

Returns the number of physical EC14150 devices present in the local system. If this function returns zero then either no EC14150 devices are present or the EC14150 driver has not been installed.

3.5.6 | IsDevicePresentEC14

Form

```
int IsDevicePresentEC14 (HEC14 hBrd);
```

Description

Determines if a given EC14150 device is still connected to system.

Parameters

[in] *hBrd*

The EC14150 device handle to check. An EC14150 device handle is obtained by calling the [ConnectToDeviceEC14](#) or [ConnectToVirtualDeviceEC14](#) function.

Return Value

Returns a positive value if the given handle is associated with an EC14150 device that is connected to the system, zero if the handle is associated with an EC14150 device that has been removed from the system, or a negative [library error code](#) on error.

Remarks

Use this function to determine if the given handle is associated with an EC14150 device that is connected to the system. If this function returns zero, then the underlying EC14150 device associated with the given handle has been unplugged from the system.

Note that when an EC14150 device is removed from the system, all open EC14150 device handles associated with that device become invalid and should be closed. These handles will remain invalid even if an EC14150 is reconnected to the system. In this case, new handles will need to be obtained by calling `ConnectToDeviceEC14`.

Related Functions

[IsHandleValidEC14](#)

3.5.7 | IsDeviceVirtualEC14

Form

```
int IsDeviceVirtualEC14 (HEC14 hBrd);
```

Description

Determines if a given EC14150 device handle is for a virtual device.

Parameters

[in] *hBrd*

The EC14150 device handle to check. An EC14150 device handle is obtained by calling the `ConnectToDeviceEC14` or `ConnectToVirtualDeviceEC14` 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 a negative [library error code](#) on error.

Remarks

Use this function to determine if the given handle is associated with a virtual EC14150 device. A virtual EC14150 is not connected to real EC14150 hardware and is mainly used for software development and debugging.

Related Functions

[IsHandleValidEC14](#)

3.5.8 | IsHandleValidEC14

Form

```
int IsHandleValidEC14 (HEC14 hBrd);
```

Description

Determines if the given EC14150 device handle is connected to a device

Parameters

[in] *hBrd*

The EC14150 device handle to check. An EC14150 device handle is obtained by calling the ConnectToDeviceEC14 or ConnectToVirtualDeviceEC14 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 a negative [library error code](#) on error.

Remarks

A handle is valid if it is connected to a local, remote, or virtual EC14150 device.

This function considers an EC14150 handle to be valid even if the associated EC14150 device has been removed from the system.

Related Functions

[IsDeviceVirtualEC14](#), [IsDevicePresentEC14](#)

3.6 | EC14150 Device State and Configuration

The functions in this section are used to obtain state and configuration information on the EC14150.

3.6.1 | GetAcqInProgressFlagEC14

Form

```
int GetAcqInProgressFlagEC14 (HEC14 hBrd);
```

Description

Get status of the acquisition in progress flag.

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

Return Value

Returns 1 if an acquisition is in progress, 0 if not acquiring, or one of the [library error codes](#) (which are all negative) on error.

3.6.2 | GetActualAdcAcqRateEC14

Form

```
int GetActualAdcAcqRateEC14 (HEC14 hBrd, double* pRateMHz);
```

Description

Obtain the actual ADC acquisition rate in MHz.

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

[in] *pRate*

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 [GetEffectiveAcqRateEC14](#) function.

Related Functions

[GetEffectiveAcqRateEC14](#)

3.6.3 | GetBoardNameEC14

Form

```
int GetBoardNameEC14 (HEC14 hBrd, TCHAR** bufpp, int flags _EC14_DEFAULT(0));
```

Description

Obtain a user friendly board name.

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

[in] *bufpp*

A pointer to a char pointer to which the board name string will be saved. Here is the board name format:

BoardModel SerialNumber

Ex: EC14150 123456

[in] *flags*

A set of flags that defines the format of the returned board name. Currently defined flags are:

Library constant	Value	Interpretation
EC14GBNF_NO_SN	0x00000001	Do not include the serial number.
EC14GBNF_USE_ORD_NUM	0x00000002	Use ordinal number instead of the serial number, if serial number is not excluded.
EC14GBNF_INC_VIRT_STATUS	0x00000004	Include virtual status if the device is virtual.
EC14GBNF_ALPHANUMERIC_ONLY	0x00000008	Add “#” character as ahead of the serial number in the board name.
EC14GBNF_USE_UNDERSCORES	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.
--	--	--

Return Value

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

Related Functions

[GetEffectiveAcqRateEC14](#)

3.6.4 | GetBoardRevisionEC14

Form

```
int GetBoardRevisionEC14 (HEC14 hBrd, unsigned int* revp, unsigned int* sub_rev);
```

Description

Obtain board revision and/or sub-revision.

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

[in] *revp*

A pointer to an unsigned integer variable that will receive the board's revision. This parameter may be NULL. At the time of this writing, there are two possible board revisions:

- EC14BRDREV_EC14150 (0) - Original EC14150 (AC-coupled 14-bit 150MHz digitizer)
- EC14BRDREV_EC14150D (1) – DC-coupled EC14150

[in] *sub_rev*

A pointer to an unsigned integer variable that will receive the board's sub revision. Interpretation of the sub revision is intended to be relative to the board revision. At the time of this writing, no sub revisions have been formally defined and zero will be returned. 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.

3.6.5 | GetEffectiveAcqRateEC14

Form

```
int GetEffectiveAcqRateEC14 (HEC14 hBrd, double* pRateMHz);
```

Description

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

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

[in] *pRate*

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

[GetActualAdcAcqRateEC14](#), [SetPostAdcClockDividerEC14](#)

3.6.6 | GetFifoFullFlagEC14

Form

```
int GetFifoFullFlagEC14 (HEC14 hBrd);
```

Description

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

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) 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 `GetFifoFullFlagEC14` 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 PCI 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.

3.6.7 | `GetFirmwareVersionEC14`

Form

```
int GetFirmwareVersionEC14 (HEC14 hBrd, unsigned long long* verp);
```

Description

Obtains the version of the EC14150 firmware.

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) 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 EC14150 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

[GetHardwareRevisionEC14](#)

3.6.8 | GetHardwareRevisionEC14

Form

```
int GetHardwareRevisionEC14 (HEC14 hBrd, unsigned long long* verp);
```

Description

Obtains the revision of the EC14150 hardware.

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) 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 EC14150 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

[GetFirmwareVersionEC14](#)

3.6.9 | GetOrdinalNumberEC14

Form

```
int GetOrdinalNumberEC14 (HEC14 hBrd, unsigned int* onp);
```

Description

Obtain the ordinal number of the EC14150 connected to the given handle.

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

[out] *onp*

A pointer to the variable that will receive the EC14150'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

An EC14150's ordinal number is the number of the EC14150 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 PCI slots.

Related Functions

[GetSerialNumberEC14](#)

3.6.10 | GetPllLockStatusEC14

Form

```
int GetPllLockStatusEC14 (HEC14 hBrd);
```

Description

Get lock status of the PLL.

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) 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.

3.6.11 | GetSampleRamSizeEC14

Form

```
int GetRamSizeEC14 (HEC14 hBrd , unsigned int* samplesp);
```

Description

Get the size of the RAM in samples.

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

[in] *samplesp*

A pointer to an unsigned int variable that will receive the size of the RAM in samples.

Return Value

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

3.6.12 | GetSamplesCompleteFlagEC14

Form

```
int GetSamplesCompleteFlagEC14 (HEC14 hBrd);
```

Description

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

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) 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 EC14150 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 WaitForAcquisitionCompleteEC14 function is a more reliable way of waiting for acquisition complete.

3.6.13 | GetSerialNumberEC14

Form

```
int GetSerialNumberEC14 (HEC14 hBrd, unsigned int* snp);
```

Description

Obtain the EC14150 board's serial number.

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

[out] *snp*

A pointer to the variable that will receive the EC14150'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

[GetOrdinalNumberEC14](#)

3.6.14 | GetVersionTextEC14

Form

```
int GetVersionTextEC14 (HEC14 hBrd, unsigned int ver_type, TCHAR** bufpp, unsigned int flags);
```

Description

Obtain the specified item version in string format.

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

[in] *ver_type*

An unsigned int variable specifying the item for which the version is to be retrieved. Use the following values:

Library constant	Value	Interpretation
EC14VERID_FIRMWARE	0x00000000	Obtain firmware version.
EC14VERID_HARDWARE	0x00000001	Obtain hardware version.
EC14VERID_DRIVER	0x00000002	Obtain driver version.
EC14VERID_LIBRARY	0x00000003	Obtain library version.
EC14VERID_EC14_SOFTWARE	0x00000004	Obtain software version.

[out] *bufpp*

A pointer to a string variable that will receive the requested version. Can be of char or wchar type, depending if Unicode is defined or not.

Return Value

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

3.6.15 | InAcquisitionModeEC14

Form

```
int InAcquisitionModeEC14 (HEC14 hBrd);
```

Description

Get the acquisition mode status of the device.

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

Return Value

Returns 1 if device is acquiring, 0 if not acquiring, or one of the [library error codes](#) (which are all negative) on error.

3.6.16 | InIdleModeEC14

Form

```
int InIdleModeEC14 (HEC14 hBrd);
```

Description

Get the stand-by mode status of the device.

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

Return Value

Returns 1 if device is in stand-by mode, 0 if not in stand-by mode, or one of the [library error codes](#) (which are all negative) on error.

3.6.17 | ReadConfigEepromEC14

Form

```
int ReadConfigEepromEC14 (HEC14 hBrd, unsigned int eeprom_addr, unsigned short* eeprom_datap);
```

Description

Read an element from the EC14150 configuration EEPROM

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) 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 EC14150 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

[WriteConfigEepromEC14](#)

3.6.18 | SetUserDataEC14 / GetUserDataEC14

Form

```
int SetUserDataEC14 (HEC14 hBrd, void *datap);  
int GetUserDataEC14 (HEC14 hBrd, void **datappp);
```

Description

Set or get user-defined data value associated with EC14 handle.

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

[in] *datap*

A pointer to a variable (any type) selected by the user.

[out] *datappp*

A pointer to a variable pointer (any type). User must keep track of the type of the variable when originally set by "SetUserDataEC14".

Return Value

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

3.6.19 | ValidateConfigEepromEC14

Form

```
int ValidateConfigEepromEC14 (HEC14 hBrd);
```

Description

Run an integrity check on the configuration EEPROM data..

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

Return Value

Returns SIG_SUCCESS (0) if EEPROM data is valid, or one of the [library error codes](#) (which are all negative) on error.

3.6.20 | WriteConfigEepromEC14

Form

```
int WriteConfigEepromEC14 (HEC14 hBrd, unsigned int eeprom_addr, unsigned short eeprom_data);
```

Description

Write an element from the EC14150 configuration EEPROM.

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

[in] *eeeprom_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] *eeeprom_data*

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

[ReadConfigEepromEC14](#)

3.7 | EC14150 Hardware Settings

The functions in this section control the getting and setting of the various EC14150 hardware settings. These settings include the numerous data acquisition, clock, and trigger parameters.

3.7.1 | GetInputVoltRangeFromSettingEC14

Form

```
int GetInputVoltRangeFromSettingEC14 (int vr_sel, double* vpp);
```

Get the input voltage range, in peak-to-peak voltage, represented by an input voltage selection enumeration (EC14VOLTRNG_*).

Parameters

[in] *vr_sel*

The input voltage range selection value to obtain peak-to-peak input voltage for. See [SetInputVoltageRangeCh1EC14](#).

[in] *vpp*

A pointer to a double variable that will receive the peak-to-peak input voltage represented by the EC14VOLTRNG_* 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

[GetInputVoltageRangeVoltsChXEC14](#)

3.7.2 | GetInputVoltageRangeVoltsChXEC14

Form

```
int GetInputVoltageRangeVoltsCh1EC14 (HEC14 hBrd, double* voltsp, int bFromCache = 1);
```

```
int GetInputVoltageRangeVoltsCh2EC14 (HEC14 hBrd, double* voltsp, int bFromCache = 1);
```

Get the input voltage range, in peak-to-peak voltage, for channel 1 or 2.

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given EC14150 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 EC14150 device register read.

[in] *voltsp*

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

Return Value

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

Remarks

The [SetInputVoltageRangeCh1EC14](#) and SetInputVoltageRangeCh2EC14 functions are used to select the input voltage range selection.

The GetInputVoltageRangeChXEC14 functions differ from the GetInputVoltageRangeVoltsChXEC14 functions in that the former obtain the input voltage range selection (EC14VOLTRNG_*) value and the latter obtain the actual peak-to-peak input voltage range.

Related Functions

[SetInputVoltageRangeCh1EC14](#), [SelectInputVoltRangeEC14](#)

3.7.3 | IssueSoftwareTriggerEC14

Form

```
int IssueSoftwareTriggerEC14 (HEC14 hBrd);
```

Description

Issue a software-generated trigger event to an EC14150.

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

Return Value

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

Remarks

The function `IssueSoftwareTriggerEC14` allows the user to issue a software trigger to the EC14150 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

[SetTriggerLevelAEC14](#), [SetTriggerModeEC14](#), [SetTriggerSourceEC14](#)

3.7.4 | SelectInputVoltRangeEC14

Form

```
int SelectInputVoltRangeEC14 (HEC14 hBrd, double vr_pp, int bCh1 = EC14_TRUE, int bCh2 = EC14TRUE, int sel_how = EC14VRSEL_NOT_LT );
```

Description

Select input voltage ranges for channel 1 and/or 2 input.

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) 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:

EC14150 Library Constant	Interpretation
EC14VRSEL_NOT_LT (0)	Select voltage range not less than given value, <i>vr_pp</i> .
EC14VRSEL_NOT_GT (1)	Select voltage range not greater than given value, <i>vr_pp</i> .
EC14VRSEL_CLOSEST (2)	Select voltage range closest to given value, <i>vr_pp</i> .

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 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.

The SetInputVoltageRangeChXEC14 functions differ from the SelectInputVoltRangeEC14 function in that the former set the input voltage range from an enumeration representing a particular input voltage range (EC14VOLTRNG_*) and the latter sets the input voltage range given a desired peak-to-peak voltage.

Related Functions

[SetInputVoltageRangeCh1EC14](#)

3.7.5 | SetActiveChannelsEC14 / GetActiveChannelsEC14

Form

int SetActiveChannelsEC14 (HEC14 hBrd, unsigned int val);
int GetActiveChannelsEC14 (HEC14 hBrd, int bFromCache = 1);

Description

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

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given EC14150 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 EC14150 device register read.

[in] *val*

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

EC14150 Library Constant	Interpretation
EC14CHANNEL_DUAL (0)	Channel 1 and Channel 2
EC14CHANNEL_ONE (1)	Single Channel (Channel 1 only)

Return Value

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

On success, GetActiveChannelsEC14 will return the current active channel selection.

3.7.6 | SetAdcClockSourceEC14 / GetAdcClockSourceEC14

Form

```
int SetAdcClockSourceEC14 (HEC14 hBrd, unsigned int val);  
int GetAdcClockSourceEC14 (HEC14 hBrd, int bFromCache = 1);
```

Description

Set or get the EC14150's source ADC clock setting.

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given EC14150 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 EC14150 device register read.

[in] *val*

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

EC14150 Library Constant	Interpretation
EC14CLKSRC_INT_VCO (0)	Internal 1.5GHz voltage-controlled oscillator (VCO)
EC14CLKSRC_EXTERNAL (1)	External clock supplied on the EC14150's external clock input

Return Value

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

On success, GetAdcClockSourceEC14 will return the current ADC clock source selection.

Remarks

Do not change the clock source or frequency while an acquisition is in progress.

Internal 1.5GHz Voltage-Controlled Oscillator

When this clock source is selected, the acquisition rate is defined by the [SetInternalClockRateEC14](#) 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 SetExternalClockRateEC14 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.

Related Functions

[SetPostAdcClockDividerEC14](#), [SetInternalClockRateEC14](#), [SetExternalClockRateEC14](#), [SetExtClockDividerEC14](#)

3.7.7 | SetDcOffsetChXEC14 / GetDcOffsetChXEC14

Prototype

<code>int SetDcOffsetCh1EC14 (HPX14 hBrd, unsigned int val);</code>
<code>int GetDcOffsetCh1EC14 (HPX14 hBrd, int bFromCache = 1);</code>
<code>int SetDcOffsetCh2EC14 (HPX14 hBrd, unsigned int val);</code>
<code>int GetDcOffsetCh2EC14 (HPX14 hBrd, int bFromCache = 1);</code>

Description

Set or get the DC-offset for channel 1 or 2. (DC-coupled devices only)

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given EC14150 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 EC14150 device register read.

[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.

Return Value

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

On success, GetDcOffsetCh1EC14 returns the current channel 1 DC-offset.

On success, GetDcOffsetCh2EC14 returns the current channel 2 DC-offset.

Remarks

These functions are only relevant for DC-coupled EC14150 devices. It is safe to call these functions for AC-coupled devices.

3.7.8 | SetDigitalIoEnableEC14 / GetDigitalIoEnableEC14

Form

int SetDigitalIoEnableEC14 (HEC14 hBrd, int bEnable);
int GetDigitalIoEnableEC14 (HEC14 hBrd, int bFromCache = 1);

Description

Enable/disable the EC14150's digital output

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given EC14150 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 EC14150 device register read.

[in] *bEnabled*

If this parameter is nonzero then the EC14150 digital output will be enabled. If this parameter is zero then digital output will be disabled. The actual digital output data is determined by the SetDigitalIoModeEC14 function.

Return Value

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

On success, GetDigitalIoEnableEC14 returns the current digital output enable setting.

Related Functions

[SetDigitalIoModeEC14](#)

3.7.9 | SetDigitalIoModeEC14 / GetDigitalIoModeEC14

Form

int SetDigitalIoModeEC14 (HEC14 hBrd, unsigned int val);
int GetDigitalIoModeEC14 (HEC14 hBrd, int bFromCache = 1);

Description

Set or get the digital output mode; determines what is driven on EC14150 digital output.

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given EC14150 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 EC14150 device register read.

[in] *val*

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

EC14150 Library Constant	Digital Input/Output	Interpretation
EC14DIGIO_OUT_0V (0)	Output	0V
EC14DIGIO_OUT_SYNC_TRIG (1)	Output	Synchronized Trigger
EC14DIGIO_OUT_ADC_CLOCK (2)	Output	ADC Clock
EC14DIGIO_OUT_3_3V (3)	Output	3.3V
EC14DIGIO_IN_TS_GEN (4)	Input	Digital input pulse requests a timestamp*
EC14DIGOUT_RESERVED_5 (5)	Assumed output	Reserved
EC14DIGOUT_RESERVED_6 (6)	Assumed output	Reserved
EC14DIGOUT_RESERVED_7 (7)	Assumed output	Reserved

Return Value

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

On success, GetDigitalIoModeEC14 returns the current digital output mode setting.

Remarks

This setting is used to define what gets driven on the EC14150 digital output. This setting is only applicable when the digital output is enabled, which is done by calling the SetDigitalIoEnableEC14 function.

*Timestamps not currently available.

Related Functions

[SetDigitalIoEnableEC14](#)

3.7.10 | SetExtClockDividerEC14 / GetExtClockDividerEC14

Form

int SetExtClockDividerEC14 (HEC14 hBrd, unsigned int val);
int GetExtClockDividerEC14 (HEC14 hBrd, int bFromCache = 1);

Description

Set or get the external clock divider value setting.

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given EC14150 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 EC14150 device register read.

[in] *val*

The divider to use for the external clock. This can be any value from 1 to 32.

Return Value

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

On success, GetExtClockDividerEC14 returns the external clock divider.

Remarks

This clock divider value is ignored when the internal clock is selected.

The EC14150 library will automatically apply this clock divider value when the clock source is changed to external clock.

Related Functions

[SetAdcClockSourceEC14](#), [SetExternalClockRateEC14](#)

3.7.11 | SetExternalClockRateEC14 / GetExternalClockRateEC14

Form

int SetExternalClockRateEC14 (HEC14 hBrd, double dRateMHz);
int GetExternalClockRateEC14 (HEC14 hBrd, double* ratep, int bFromCache = 1);

Description

Set or get the assumed external clock rate in MHz.

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given EC14150 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 EC14150 device register read.

[in] *dRateMHz*

The assumed external clock rate in MHz. 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.

Return Value

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

Remarks

The assumed external clock rate has no direct effect on the underlying EC14150 hardware. It exists solely as a convenience for the applications programming. The updating of this setting is up to the end user since the software cannot know the external clock rate.

Though there is no direct hardware effect of this setting, it is important to keep the EC14150 software up-to-date with the external clock rate. When a change in the acquisition clock frequency is made, the EC14150 firmware needs to be resynchronized with this new frequency. Ensuring this setting guarantees that this synchronization takes place.

The GetEffectiveClockRateEC14 function implementation will use this assumed rate when calculating the effective clock rate when the external clock is selected.

Related Functions

[SetAdcClockSourceEC14](#), [SetExtClockDividerEC14](#)

3.7.12 | SetInputVoltageRangeChXEC14 / GetInputVoltageRangeChXEC14

Form

int SetInputVoltageRangeCh1EC14 (HEC14 hBrd, unsigned int val);
int SetInputVoltageRangeCh2EC14 (HEC14 hBrd, unsigned int val);
int GetInputVoltageRangeCh1EC14 (HEC14 hBrd, int* vr_selp, int bFromCache = 1);
int GetInputVoltageRangeCh2EC14 (HEC14 hBrd, int* vr_selp, int bFromCache = 1);

Description

Set or get the input voltage range selection for channel 1 or 2

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given EC14150 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 EC14150 device register read.

[in] *val*

The input voltage range selection to set. EC14150D devices (DC-coupled EC14150) may only select EC14VOLTRNGD_* selections. Normal, AC-coupled EC14150 devices may not select EC14VOLTRNGD_* selection.

Library Constant	Coupling	Value	Interpretation
EC14VOLTRNG_6_340	AC	0	6.339572770 V _{p-p}
EC14VOLTRNG_5_650	AC	1	5.650150178 V _{p-p}
EC14VOLTRNG_5_036	AC	2	5.035701647 V _{p-p}
EC14VOLTRNG_4_488	AC	3	4.488073817 V _{p-p}
EC14VOLTRNG_4_000	AC	4	4.000000000 V _{p-p}
EC14VOLTRNG_3_565	AC	5	3.565003753 V _{p-p}
EC14VOLTRNG_3_177	AC	6	3.177312939 V _{p-p}
EC14VOLTRNG_2_832	AC	7	2.831783138 V _{p-p}
EC14VOLTRNG_2_524	AC	8	2.523829378 V _{p-p}
EC14VOLTRNG_2_250	AC	9	2.249365301 V _{p-p}
EC14VOLTRNG_2_005	AC	10	2.004748935 V _{p-p}
EC14VOLTRNG_1_787	AC	11	1.786734369 V _{p-p}
EC14VOLTRNG_1_592	AC	12	1.592428682 V _{p-p}
EC14VOLTRNG_1_419	AC	13	1.419253557 V _{p-p}
EC14VOLTRNG_1_265	AC	14	1.264911064 V _{p-p}
EC14VOLTRNG_1_127	AC	15	1.127353173 V _{p-p}
EC14VOLTRNG_1_005	AC	16	1.004754573 V _{p-p}
EC14VOLTRNG_0_895	AC	17	0.895488455 V _{p-p}
EC14VOLTRNG_0_798	AC	18	0.798104926 V _{p-p}

Continued:

Library Constant	Coupling	Value	Interpretation
EC14VOLTRNG_0_711	AC	19	0.711311764 V _{p-p}
EC14VOLTRNG_0_634	AC	20	0.633957277 V _{p-p}
EC14VOLTRNG_0_565	AC	21	0.565015018 V _{p-p}
EC14VOLTRNG_0_504	AC	22	0.503570165 V _{p-p}
EC14VOLTRNG_0_449	AC	23	0.448807382 V _{p-p}
EC14VOLTRNG_0_400	AC	24	0.400000000 V _{p-p}
<i>Input voltage ranges below have slightly lower SNR relative to input voltage ranges above.</i>			
EC14VOLTRNG_0_356	AC	25	0.355655882 V _{p-p}
EC14VOLTRNG_0_317	AC	26	0.316978638 V _{p-p}
EC14VOLTRNG_0_283	AC	27	0.282507509 V _{p-p}
EC14VOLTRNG_0_252	AC	28	0.251785082 V _{p-p}
EC14VOLTRNG_0_224	AC	29	0.224403691 V _{p-p}
EC14VOLTRNG_0_200	AC	30	0.200000000 V _{p-p}
<i>Input voltage ranges below are only valid for EC14150D (DC-coupled) devices.</i>			
EC14VOLTRNGD_2_00	DC	31	2.0 V _{p-p}
EC14VOLTRNGD_1_00	DC	32	1.0 V _{p-p}
EC14VOLTRNGD_0_50	DC	33	0.5 V _{p-p}
EC14VOLTRNGD_0_25	DC	34	0.25 V _{p-p}

[out] *vr_selp*

A pointer to the variable that will receive the current input voltage range selection.

Return Value

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

Remarks

The GetInputVoltageRangeChXEC14 functions differ from the GetInputVoltageRangeVoltsChXEC14 functions in that the former obtain the input voltage range selection (EC14VOLTRNG_*) value and the latter obtain the actual peak-to-peak input voltage range.

The SetInputVoltageRangeChXEC14 functions differ from the SelectInputVoltRangeEC14 function in that the former set the input voltage range from an enumeration representing a particular input voltage range (EC14VOLTRNG_*) and the latter sets the input voltage range given a desired peak-to-peak voltage.

AC/DC Coupling

AC-coupled devices (EC14150A) and DC-coupled devices (EC14150D) have different input voltage range selections. The reason for this is that they have completely different analog front ends. To programmatically determine if an EC14150 device is AC- or DC-coupled, the [GetBoardRevisionEC14](#) function can be called to

obtain the board's revision. A revision of EC14BRDREV_EC14150 (0) means that the EC14150 device is AC-coupled. A revision of EC14BRDREV_EC14150D (1) means that the device is DC-coupled (i.e. an EC14150D).

Related Functions

[SelectInputVoltRangeEC14](#), [GetInputVoltRangeFromSettingEC14](#), [GetInputVoltageRangeVoltsChXEC14](#)

3.7.13 | SetInternalClockRateEC14 / GetInternalClockRateEC14

Form

int SetInternalClockRateEC14 (HEC14 hBrd, double dRateMHz);
int GetInternalClockRateEC14 (HEC14 hBrd, double* ratep, int bFromCache = 1);

Set or get the ADC clock rate; only applies to internal 1.5GHz VCO

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given EC14150 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 EC14150 device register read.

[in] *dRateMHz*

The clock rate (i.e. sampling rate), in MHz, to use when the 1.5GHz VCO is selected as the source clock. This value must be in the range of 45 and 150.

[out] *ratep*

A pointer to the variable that will receive the current internal clock rate when the 1.5GHz VCO is selected as the ADC clock source.

Return Value

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

Remarks

The 1.5GHz VCO ADC clock source is a voltage controlled oscillator. 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.

Related Functions

[SetAdcClockSourceEC14](#)

3.7.14 | SetOperatingModeEC14 / GetOperatingModeEC14

Form

```
int SetOperatingModeEC14 (HEC14 hBrd, unsigned int val);  
int GetOperatingModeEC14 (HEC14 hBrd, int bFromCache = 1);
```

Description

Set or get the EC14150's operating mode.

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given EC14150 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 EC14150 device register read.

[in] *val*

The operating mode to switch to. This can be any of the following values:

EC14150 Library Constant	Interpretation
EC14MODE_OFF (0)	Off (power-down) mode
EC14MODE_STANDBY (1)	Standby (ready) mode
EC14MODE_ACQ_RAM (2)	RAM acquisition mode.
EC14MODE_RAM_READ_PCI (3)	EC14150 RAM read to PCI; transfer data from EC14150 RAM to host PC. <u>Never set this mode directly; see Remarks.</u>
EC14MODE_ACQ_PCI_BUF (4)	RAM-buffered PCI acquisition mode; data buffered through EC14150 RAM. <u>Never set this mode directly; see Remarks.</u>

Return Value

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

On success, GetOperatingModeEC14 returns the current operating mode setting.

Remarks

For most all active operating modes the EC14150 library implements higher level routines that automatically manage the operating mode. These higher-level routines are shown in the following table.

Underlying Operating Mode	Wrapper Function(s)
EC14MODE_ACQ_RAM	AcquireToBoardRamEC14
EC14MODE_ACQ_PCI_BUF	BeginBufferedPciAcquisitionEC14
EC14MODE_RAM_READ_PCI	ReadSampleRamFastEC14 ReadSampleRamBufEC14

When entering OFF mode, the EC14150 library will automatically disable digital IO by calling the [SetDigitalIoEnableEC14](#) function.

Related Functions

[SetPowerDownOverrideEC14](#)

3.7.15 | [SetPostAdcClockDividerEC14](#) / [GetPostAdcClockDividerEC14](#)

Form

<code>int SetPostAdcClockDividerEC14 (HEC14 hBrd, unsigned int div);</code>
<code>int GetPostAdcClockDividerEC14 (HEC14 hBrd, int bFromCache = 1);</code>

Description

Set or get the EC14150's post-ADC clock divider; effective clock division by dropping samples.

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given EC14150 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 EC14150 device register read.

[in] *div*

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

EC14150 Library Constant	Effective Clock Divider
EC14POSTADCCLKDIV_01 (0)	1
EC14POSTADCCLKDIV_02 (1)	2
EC14POSTADCCLKDIV_04 (2)	4
EC14POSTADCCLKDIV_08 (3)	8
EC14POSTADCCLKDIV_16 (4)	16
EC14POSTADCCLKDIV_32 (5)	32

Return Value

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

On success, GetPostAdcClockDividerEC14 returns the current post-ADC clock divider setting.

Related Functions

[SetAdcClockSourceEC14](#), [SetInternalClockRateEC14](#), [SetExternalClockRateEC14](#)

3.7.16 | SetPowerDownOverrideEC14 / GetPowerDownOverrideEC14

Form

int SetPowerDownOverrideEC14 (HEC14 hBrd, unsigned int val);
int GetPowerDownOverrideEC14 (HEC14 hBrd, int bFromCache = 1);

Description

Set or get the power-down override; prevents auto-shutdown of components when idle.

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given EC14150 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 EC14150 device register read.

[in] *val*

If non-zero, the power-down override will be enabled. If zero, the power-down override will be disabled.

Return Value

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

On success, GetPowerDownOverrideEC14 returns the current power-down override setting.

Remarks

When the power-down override is enabled, the EC14150 will not automatically power down unused hardware components while in Standby operating mode.

When the EC14150 is put into the OFF operating mode, this setting has no effect; hardware components will always be powered down.

Changing the status of the power-down enable bit will not have an immediate change on current power settings. Component power settings are changed on relevant operating mode transitions.

Related Functions

[SetOperatingModeEC14](#)

3.7.17 | SetPreTriggerSamplesEC14 / GetPreTriggerSamplesEC14

Form

<code>int SetPreTriggerSamplesEC14 (HEC14 hBrd, unsigned int val);</code>
<code>int GetPreTriggerSamplesEC14 (HEC14 hBrd, int bFromCache = 1);</code>

Description

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

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given EC14150 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 EC14150 device register read.

[in] *val*

The number of pre-trigger samples to set. This can be any even number not greater than 1024. The function will clip this value at the maximum allowed value if necessary. If this value is 0 then no pre-trigger samples are collected.

Return Value

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

On success, GetPreTriggerSamplesEC14 returns the current pre-trigger sample count setting.

Remarks

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

3.7.18 | SetSegmentSizeEC14 / GetSegmentSizeEC14

Form

int SetSegmentSizeEC14 (HEC14 hBrd, unsigned int segSize);
int GetSegmentSizeEC14 (HEC14 hBrd, int bFromCache = 1);

Description

Set or get the EC14150's segment size for use with Segmented triggering mode.

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given EC14150 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 EC14150 device register read.

[in] *segSize*

The segment size, in samples, to set. This can be any even value not larger than 4194304.

Return Value

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

On success, GetSegmentSizeEC14 returns the current segment size setting.

Remarks

This setting has no effect when the EC14150 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 2-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 number of samples acquired on channel 1 only.

When the EC14150 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 EC14150 reaches the specified total samples count (via [SetSampleCountEC14](#)). To be meaningful, the segment size should always be set smaller than the active memory size.

Related Functions

[SetTriggerModeEC14](#)

3.7.19 | SetSampleCountEC14 / GetSampleCountEC14

Form

int SetSampleCountEC14 (HEC14 hBrd, unsigned int val);
int GetSampleCountEC14 (HEC14 hBrd, int bFromCache = 1);

Description

Set or get the EC14150's total sample count; defines length of subsequent acquisitions/transfers.

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given EC14150 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 EC14150 device register read.

[in] *val*

The new total sample count. This should be a multiple of 16. This value will be clipped to the largest valid sample count value, 1073741808. Note that this is larger than the EC14150 RAM size. For dual channel data acquisitions, this value represents the combined total of samples to acquire for both channels. This value can be EC14_FREE_RUN (0) to specify a free-run (infinite) sample count. See Remarks.

Return Value

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

On success, GetSampleCountEC14 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.

In the PCI acquisition mode it is possible to acquire indefinitely. This is done by specifying the special value EC14_FREE_RUN (0). To terminate a free-run acquisition, return the board to Standby mode.

Related Functions

[SetStartSampleEC14](#)

3.7.20 | SetStartSampleEC14 / GetStartSampleEC14

Form

int SetStartSampleEC14 (HEC14 hBrd, unsigned int val);
int GetStartSampleEC14 (HEC14 hBrd, int bFromCache = 1);

Description

Set or get the EC14150's start sample; defines starting EC14150 RAM address of subsequent acquisitions/transfers.

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given EC14150 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 EC14150 device register read.

[in] *val*

The new starting sample. This should be a multiple of 4 samples. This value will be clipped to the largest valid starting sample, 268435452.

Return Value

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

On success, GetStartSampleEC14 returns the current starting sample setting.

Remarks

This function sets the EC14150 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 EC14150 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 EC14150 is placed in Standby mode.

Related Functions

[SetSampleCountEC14](#)

3.7.21 | SetTriggerDirectionAEC14 / GetTriggerDirectionAEC14

Form

int SetTriggerDirectionAEC14 (HEC14 hBrd, unsigned int val);
int GetTriggerDirectionAEC14 (HEC14 hBrd, int bFromCache = 1);

Description

Set or get trigger A direction to use when validating the trigger.

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given EC14150 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 EC14150 device register read.

[in] *val*

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

EC14150 Library Constant	Interpretation
EC14TRIGDIR_POS (0)	Trigger occurs on positive going signal (Power-up default)
EC14TRIGDIR_NEG (1)	Trigger occurs on negative going signal

Return Value

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

On success, GetTriggerDirectionAEC14 returns the current trigger A direction setting.

Remarks

The EC14150 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

[SetTriggerLevelAEC14](#), [SetTriggerSourceEC14](#)

3.7.22 | SetTriggerDirectionBEC14 / GetTriggerDirectionBEC14

Form

```
int SetTriggerDirectionBEC14 (HEC14 hBrd, unsigned int val);  
int GetTriggerDirectionBEC14 (HEC14 hBrd, int bFromCache = 1);
```

Description

Set or get trigger B direction to use when validating the trigger.

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given EC14150 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 EC14150 device register read.

[in] *val*

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

EC14150 Library Constant	Interpretation
EC14TRIGDIR_POS (0)	Trigger occurs on positive going signal (Power-up default)
EC14TRIGDIR_NEG (1)	Trigger occurs on negative going signal

Return Value

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

On success, GetTriggerDirectionBEC14 returns the current trigger B direction setting.

Remarks

The EC14150 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

[SetTriggerLevelBEC14](#), [SetTriggerSourceEC14](#)

3.7.23 | SetTriggerDirectionExtEC14 / GetTriggerDirectionExtEC14

Form

```
int SetTriggerDirectionExtEC14 (HEC14 hBrd, unsigned int val);  
int GetTriggerDirectionExtEC14 (HEC14 hBrd, int bFromCache = 1);
```

Description

Set or get external trigger direction to use when validating the trigger.

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given EC14150 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 EC14150 device register read.

[in] *val*

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

EC14150 Library Constant	Interpretation
EC14TRIGDIR_POS (0)	Trigger occurs on positive-going edge of TTL signal (Power-up default)
EC14TRIGDIR_NEG (1)	Trigger occurs on negative-going edge of TTL signal

Return Value

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

On success, GetTriggerDirectionExtEC14 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

[SetTriggerSourceEC14](#)

3.7.24 | SetTriggerLevelAEC14 / GetTriggerLevelAEC14

Form

int SetTriggerLevelAEC14 (HEC14 hBrd, unsigned int val);
int GetTriggerLevelAEC14 (HEC14 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 EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given EC14150 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 EC14150 device register read.

[in] *val*

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

Return Value

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

On success, GetTriggerLevelAEC14 returns the current trigger A level setting.

Remarks

The EC14150 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 ([SetTriggerDirectionAEC14](#)).

Related Functions

[SetTriggerDirectionAEC14](#), [SetTriggerSourceEC14](#)

3.7.25 | SetTriggerLevelBEC14 / GetTriggerLevelBEC14

Form

<code>int SetTriggerLevelBEC14 (HEC14 hBrd, unsigned int val);</code>
<code>int GetTriggerLevelBEC14 (HEC14 hBrd, int bFromCache = 1);</code>

Description

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

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given EC14150 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 EC14150 device register read.

[in] *val*

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

Return Value

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

On success, GetTriggerLevelBEC14 returns the current trigger B level setting.

Remarks

The EC14150 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 ([SetTriggerDirectionBEC14](#)).

Related Functions

[SetTriggerDirectionBEC14](#), [SetTriggerSourceEC14](#)

3.7.26 | SetTriggerModeEC14 / GetTriggerModeEC14

Form

```
int SetTriggerModeEC14 (HEC14 hBrd, unsigned int val);  
int GetTriggerModeEC14 (HEC14 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 EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given EC14150 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 EC14150 device register read.

[in] *val*

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

EC14150 Library Constant	Interpretation
EC14TRIGMODE_POST_TRIGGER (0)	Trigger event starts a single data acquisition (Power-up default)
EC14TRIGMODE_SEGMENTED (1)	Each trigger event begins a new statically sized data acquisition

Return Value

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

On success, GetTriggerModeEC14 returns the current trigger mode setting.

Related Functions

[SetTriggerSourceEC14](#)

3.7.27 | SetTriggerSourceEC14 / GetTriggerSourceEC14

Form

```
int SetTriggerSourceEC14 (HEC14 hBrd, unsigned int val);  
int GetTriggerSourceEC14 (HEC14 hBrd, int bFromCache = 1);
```

Description

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

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

[in] *bFromCache*

If non-zero, the setting will be read from the local device register cache associated with the given EC14150 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 EC14150 device register read.

[in] *val*

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

EC14150 Library Constant	Interpretation
EC14TRIGSRC_INT_CH1 (0)	Internal trigger, channel 1 (Power-up default)
EC14TRIGSRC_INT_CH2 (1)	Internal trigger, channel 2
EC14TRIGSRC_EXT (2)	External trigger

Return Value

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

On success, GetTriggerSourceEC14 returns the current trigger source setting.

Remarks

This function allows the user to select the EC14150 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.

Related Functions

[SetTriggerLevelAEC14](#), [SetTriggerDirectionAEC14](#), [SetTriggerModeEC14](#)

3.8 | Device Register State Functions

The functions in this section involve manipulation of EC14150 device registers.

3.8.1 | CopyHardwareSettingsEC14

Form

```
int CopyHardwareSettingsEC14 (HEC14 hBrdDst, HEC14 hBrdSrc);
```

Description

Copy hardware settings from another EC14150 device.

Parameters

[in] *hBrdDst*

A handle to the EC14150 device to which the copied hardware settings will be applied.

[in] *hBrdSrc*

A handle to the EC14150 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 EC14150 associated with the source device to the EC14150 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 board is placed into Standby operating mode before any settings are applied.

Related Functions

[RewriteHardwareSettingsEC14](#), [RefreshLocalRegisterCacheEC14](#)

3.8.2 | ReadAllDeviceRegistersEC14

Form

```
int ReadAllDeviceRegistersEC14 (HEC14 hBrd);
```

Description

Read all device registers from hardware; updates all register caches.

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) 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

[RewriteHardwareSettingsEC14](#), [RefreshLocalRegisterCacheEC14](#)

3.8.3 | RefreshLocalRegisterCacheEC14

Form

```
int RefreshLocalRegisterCacheEC14 (HEC14 hBrd, int bFromHardware = 0);
```

Description

Refresh local device register cache from driver's cache; no hardware read.

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

[in] *bFromHardware*

If this parameter is non-zero then register content will be updated from the EC14150 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 EC14150 device register cache associated with the given EC14150 device handle. The register values are obtained from the kernel-level register cache maintained by the EC14150 device driver. Calling this function has no effect on any underlying EC14150 hardware.

This function is automatically invoked by the [ConnectToDeviceEC14](#) function as part of the device connection procedure.

Related Functions

[RewriteHardwareSettingsEC14](#)

3.8.4 | RewriteHardwareSettingsEC14

Form

```
int RewriteHardwareSettingsEC14 (HEC14 hBrd);
```

Description

Bring hardware settings up to date with current cache settings.

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) 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 EC14150 handle.

Calling this function will place the board in the Standby operating mode regardless of the operating mode in the register cache.

Related Functions

[RefreshLocalRegisterCacheEC14](#)

3.8.5 | SetPowerupDefaultsEC14

Form

```
int SetPowerupDefaultsEC14 (HEC14 hBrd);
```

Description

Restores all EC14150 settings to power-up default values.

Parameters

[in] *hBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

Return Value

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

Remarks

The board 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 EC14150 settings, these equate to the '0' values. A few exceptions to this:

- Trigger levels are set to midscale (EC14_TRIGGER_LEVEL_MIDSCALE).
- Internal acquisition rate is set to 150MHz.
- The board is kept in Standby mode.

3.9 | Memory/DMA Buffer Allocation Routines

These functions in this section pertain to memory allocation and freeing.

3.9.1 | AllocateDmaBufferEC14

Form

```
int AllocateDmaBufferEC14 (HEC14 hBrd, unsigned int samples, ec14\_sample\_t** bufpp);
```

Description

Allocate a DMA buffer for use with DMA transfers.

Parameters

[in] *pBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) 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 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 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 PC is via a DMA transfer. A DMA buffer must be freed via the [FreeDmaBufferEC14](#) function when it is no longer needed.

The DMA buffer may only be used with the EC14150 device that it was allocated for. A DMA buffer can be used with another EC14150 device handle (HEC14) of the same process as long as both device handles refer to the same underlying EC14150 device.

All 'Fast' EC14150 library functions (e.g. [ReadSampleRamFastEC14](#)) 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.

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 EC14150 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 EC14150 software will ensure that all DMA buffers are properly cleaned up when a process exits, gracefully or not.

Related Functions

[FreeDmaBufferEC14](#), [ReadSampleRamFastEC14](#), [WriteSampleRamFastEC14](#)

3.9.2 | FreeDmaBufferEC14

Form

```
int FreeDmaBufferEC14 (HEC14 hBrd, ec14\_sample\_t* bufp);
```

Description

Free a DMA buffer previously allocated by the [AllocateDmaBufferEC14](#) function.

Parameters

[in] *pBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) 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 EC14150 driver will automatically free any DMA buffers that have not been freed by a process when it exits.

Related Functions

[AllocateDmaBufferEC14](#)

3.9.3 | FreeMemoryEC14

Form

```
int FreeMemoryEC14 (void* p);
```

Description

Free memory allocated by the EC14150 library.

Parameters

[in] *p*
The address of the EC14150 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 EC14150 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 EC14150 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; use [FreeDmaBufferEC14](#)

3.10 | Data Acquisition Routines

The functions in this section are used to perform data acquisitions.

3.10.1 | AcquireToBoardRamEC14

Form

```
int AcquireToBoardRamEC14 (HEC14 hBrd, unsigned int samp_start, unsigned int samp_count, unsigned int timeout_ms = 0, int bAsynchronous = 0);
```

Description

Acquire data to EC14150 RAM.

Parameters

[in] *pBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

[in] *samp_start*

The EC14150 RAM sample at which to put the first acquired sample. This parameter has the same restrictions defined in the [SetStartSampleEC14](#) 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 [SetSampleCountEC14](#) 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 EC14150 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 EC14150 sample RAM to the host PC by calling ReadSampleRamFastEC14 or ReadSampleRamBufEC14.

An acquisition may also be cancelled by putting the board 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 `IsAcquisitionInProgressEC14` function to determine if the acquisition is still in progress. To wait (sleep) for the acquisition to complete, call the `WaitForAcquisitionCompleteEC14` function.

Related Functions

[IsAcquisitionInProgressEC14](#), [WaitForAcquisitionCompleteEC14](#)

3.10.2 | BeginBufferedPciAcquisitionEC14

Form

```
int BeginBufferedPciAcquisitionEC14 (HEC14 hBrd , unsigned int samp_count = EC14_FREE_RUN);
```

Description

Begin an EC14150 RAM buffered PCI acquisition.

Parameters

[in] *pBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) 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 an EC14150 RAM buffered PCI data acquisition. In this type of data acquisition, the EC14150 sample RAM (128 mibi-samples) is used as a large FIFO to buffer the data as it is consumed via DMA transfer over the PCI/PCI-X bus.

Once the acquisition has been started, the `GetPciAcquisitionDataFastEC14` 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 `EndBufferedPciAcquisitionEC14` function.

This function takes care of setting up the active memory region and all mode changes.

Related Functions

[GetPciAcquisitionDataFastEC14](#), [EndBufferedPciAcquisitionEC14](#)

3.10.3 | EndBufferedPciAcquisitionEC14

Form

```
int EndBufferedPciAcquisitionEC14 (HEC14 hBrd);
```

Description

End a buffered PCI acquisition.

Parameters

[in] *pBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) 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 EC14150 RAM buffered PCI acquisition started by a previous call to [BeginBufferedPciAcquisitionEC14](#).

Related Functions

[BeginBufferedPciAcquisitionEC14](#), [GetPciAcquisitionDataFastEC14](#)

3.10.4 | IsAcquisitionInProgressEC14

Form

```
int IsAcquisitionInProgressEC14 (HEC14 hBrd);
```

Description

Determine if an asynchronous RAM acquisition is currently in progress.

Parameters

[in] *pBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) 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 not used for PCI-based acquisitions. Instead, the `IsTransferInProgressEC14` is used to determine if a DMA transfer is currently in progress.

This function is used to determine a RAM acquisition (`AcquireToBoardRamEC14`) 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 EC14150 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.

Related Functions

[WaitForAcquisitionCompleteEC14](#)

3.10.5 | WaitForAcquisitionCompleteEC14

Form

```
int WaitForAcquisitionCompleteEC14 (HEC14 hBrd, unsigned int timeout_ms = 0);
```

Description

Wait for a RAM acquisition operation to complete with optional timeout.

Parameters

[in] *pBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) 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_EC14_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 EC14150 into the Standby operating mode.

Remarks

This function is used when doing asynchronous data acquisition operations. This function is not used for PCI-based acquisitions. Instead, the WaitForTransferCompleteEC14 is used to wait for a DMA transfer to finish.

This function can be used to wait a RAM acquisition (via AcquireToBoardRamEC14) to complete.

Calling this function while any of the above operations are in progress (see [IsAcquisitionInProgressEC14](#)) 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.

This function will automatically put the EC14150 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.

Related Functions

[IsAcquisitionInProgressEC14](#)

3.11 | Data Transfer Routines

The functions in this section are used to perform data transfers.

3.11.1 | GetPciAcquisitionDataBufEC14

Form

```
int GetPciAcquisitionDataBufEC14 (HEC14 hBrd, unsigned int samples, ec14\_sample\_t* bufp, int bAsynchronous = 0);
```

Description

Obtain fresh acquisition data during a PCI data acquisition using buffered transfers.

Parameters

[in] *pBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) 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 EC14 into Standby mode, usually from another thread (or process).

Remarks

This function is identical in usage to the [GetPciAcquisitionDataFastEC14](#) function. The main difference between the two functions is the underlying transfer method. Most all users will want to use the 'fast' transfer functions for PCI buffered acquisitions to maximize throughput so the PCI FIFOs do not overflow. This function is implemented for use on platforms in which native memory access is not available (e.g. Visual Basic .NET).

This function is used to obtain data from the acquisition started by a previous call to [BeginBufferedPciAcquisitionEC14](#).

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 `IsTransferInProgressEC14` function to determine if the transfer is still in progress. To wait (sleep) for the transfer to complete, call the `WaitForTransferCompleteEC14` function.

Related Functions

[BeginBufferedPciAcquisitionEC14](#), [EndBufferedPciAcquisitionEC14](#), [IsTransferInProgressEC14](#),
[WaitForTransferCompleteEC14](#)

3.11.2 | GetPciAcquisitionDataFastEC14

Form

```
int GetPciAcquisitionDataFastEC14 (HEC14 hBrd, unsigned int samples, ec14 sample t* dma_bufp, int bAsynchronous = 0);
```

Description

Obtain fresh acquisition data during a PCI data acquisition using fast, unbuffered transfers.

Parameters

[in] *pBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) 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 EC14 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 [BeginBufferedPciAcquisitionEC14](#).

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 `IsTransferInProgressEC14` function to determine if the transfer is still in progress. To wait (sleep) for the transfer to complete, call the `WaitForTransferCompleteEC14` function.

Related Functions

[BeginBufferedPciAcquisitionEC14](#), [EndBufferedPciAcquisitionEC14](#), [IsTransferInProgressEC14](#),
[WaitForTransferCompleteEC14](#)

3.11.3 | IsTransferInProgressEC14

Form

```
int IsTransferInProgressEC14 (HEC14 hBrd);
```

Description

Determine if an asynchronous data transfer is currently in progress.

Parameters

[in] *pBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) 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 EC14150 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 EC14150 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

[WaitForTransferCompleteEC14](#)

3.11.4 | ReadSampleRamFastEC14

Form

```
int ReadSampleRamFastEC14 (HEC14 hBrd, unsigned int sample_start, unsigned int sample_count,  
ec14\_sample\_t* dma_bufp, int bAsynchronous = 0);
```

Description

Transfer data in EC14150 RAM to a DMA buffer on the host PC.

Parameters

[in] *pBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

[in] *sample_start*

The index of the first sample to copy. This parameter has the same restrictions defined in the [SetStartSampleEC14](#) function.

[in] *sample_count*

The total number of samples to copy. This parameter has the same restrictions defined in the [SetSampleCountEC14](#) 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 [AllocateDmaBufferEC14](#) 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 EC14150 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 [AllocateDmaBufferEC14](#) function.

This function handles all necessary operating mode changes and active memory settings.

The difference between this function and the `ReadSampleRamBufEC14` function is that `ReadSampleRamBufEC14` 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 `IsTransferInProgressEC14` function to determine if the transfer has completed. To wait (sleep) for the transfer to complete, call the `WaitForTransferCompleteEC14` function.

Related Functions

[AllocateDmaBufferEC14](#), [ReadSampleRamBufEC14](#), [IsTransferInProgressEC14](#), [WaitForTransferCompleteEC14](#)

3.11.5 | ReadSampleRamBufEC14

Form

```
int ReadSampleRamBufEC14 (HEC14 hBrd, unsigned int sample_start, unsigned int sample_count, ec14\_sample\_t* bufp, int bAsynchronous = 0);
```

Description

Transfer data in EC14150 RAM to a buffer on the PC; any boundary or alignment at the expense of speed.

Parameters

[in] *pBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

[in] *start_sample*

The first sample in the EC14150 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 EC14150 RAM. This value need not be on any particular boundary.

[out] *bufp*

A pointer to the buffer that will receive the EC14150 data. This buffer does not have to be a DMA buffer. (If you do have a DMA buffer you should use [ReadSampleRamFastEC14](#) 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 [ReadSampleRamFastEC14](#) 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 [ReadSampleRamFastEC14](#) is that the [ReadSampleRamFastEC14](#) 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 [ReadSampleRamFastEC14](#). The advantage of this is that [ReadSampleRamFastEC14](#) can run faster since no intermediate buffering is required.

To transfer and automatically de-interleave dual-channel data use the [ReadSampleRamDualChannelBufEC14](#) 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 [IsTransferInProgressEC14](#) function to determine if the transfer has completed. To wait (sleep) for the transfer to complete, call the [WaitForTransferCompleteEC14](#) function.

Related Functions

[ReadSampleRamFastEC14](#), [ReadSampleRamDualChannelBufEC14](#)

3.11.6 | [ReadSampleRamDualChannelBufEC14](#)

Form

```
int ReadSampleRamDualChannelBufEC14 (HEC14 hBrd, unsigned int sample_start, unsigned int sample_count, ec14\_sample\_t* buf_ch1p, ec14\_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] *pBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

[in] *start_sample*

The first sample in the EC14150 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 EC14150 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 [ReadSampleRamFastEC14](#) 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 [ReadSampleRamBufEC14](#) 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 [IsTransferInProgressEC14](#) function to determine if the transfer has completed. To wait (sleep) for the transfer to complete, call the [WaitForTransferCompleteEC14](#) function.

Related Functions

[ReadSampleRamBufEC14](#)

3.11.7 | ReadSampleRamFileFastEC14

Form

```
int ReadSampleRamFileFastEC14 (HEC14 hBrd, unsigned int sample_start, unsigned int sample_count, ec14\_sample\_t* dma_bufp, unsigned int dma_buf_samples, EC14S\_FILE\_WRITE\_PARAMS* paramsp);
```

Description

Transfer data in EC14150 RAM to a file on the host PC.

Parameters

[in] *pBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

[in] *sample_start*

The index of the first sample to copy. This parameter has the same restrictions defined in the [SetStartSampleEC14](#) function.

[in] *sample_count*

The total number of samples to copy. This parameter has the same restrictions defined in the [SetSampleCountEC14](#) function and must be an integer multiple of 1024 samples.

[in] *dma_bufp*

The address of a DMA buffer previously allocated by the [AllocateDmaBufferEC14](#) function. This DMA buffer is used for the data transfer between the EC14150 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 [EC14S_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 EC14150 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 [AllocateDmaBufferEC14](#) function.

This function handles all necessary operating mode changes and active memory settings.

Related Functions

[AllocateDmaBufferEC14](#), [ReadSampleRamBufEC14](#)

3.11.8 | ReadSampleRamFileBufEC14

Form

```
int ReadSampleRamFileBufEC14 (HEC14 hBrd, unsigned int sample_start, unsigned int sample_count, EC14S\_FILE\_WRITE\_PARAMS* paramsp);
```

Description

Transfer data in EC14150 RAM to a file on the host PC; any boundary or alignment at the expense of speed.

Parameters

[in] *pBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) 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 [EC14S_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

[ReadSampleRamBufEC14](#), [ReadSampleRamFileFastEC14](#)

3.11.9 | WaitForTransferCompleteEC14

Form

```
int WaitForTransferCompleteEC14 (HEC14 hBrd, unsigned int timeout_ms = 0);
```

Description

Wait for a DMA transfer operation to complete with optional timeout.

Parameters

[in] *pBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) 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_EC14_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 EC14150 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 [IsTransferInProgressEC14](#)) 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

[IsTransferInProgressEC14](#)

3.12 | Data Manipulation Routines

3.12.1 | DeInterleaveDataEC14

Form

```
int DeInterleaveDataEC14 (const ec14\_sample\_t* srcp,  
                          unsigned int samples_in, ec14\_sample\_t* dst_ch1p, ec14\_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 EC14150, sample data is interleaved: Channel 1 Sample 1, Channel 2 Sample 1, Channel 1 Sample 2, Channel 2 Sample 2, ... Use this function to extract out one or both channels of data into separate buffers.

Related Functions

[InterleaveDataEC14](#)

3.12.2 | InterleaveDataEC14

Form

```
int InterleaveDataEC14 (const ec14\_sample\_t* src_ch1p, const ec14\_sample\_t* src_ch2p, unsigned int  
samps_per_chan, ec14\_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

[DeInterleaveDataEC14](#)

3.13 | EC14150 Recording Session Management Routines

The EC14150 library implements a high-level recording interface that simplifies coding for recording EC14150 acquisition data to permanent storage. Using this interface, all aspects of EC14150 hardware interaction and output file management is managed internally by the library.

The library currently supports two types of recordings: PCI Acquisition recordings and RAM Acquisition/Transfer recordings.

3.13.1 | AbortRecordingSessionEC14

Form

```
int AbortRecordingSessionEC14 (HEC14RECORDING hRec)
```

Description

Abort the current recording session; stops all recording and closes all files.

Parameters

[in] *hRec*

A handle to an EC14150 recording session. This handle is obtained by calling the CreateRecordingSessionEC14 function.

Return Value

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

Related Functions

[CreateRecordingSessionEC14](#)

3.13.2 | ArmRecordingSessionEC14

Form

```
int ArmRecordingSessionEC14 (HEC14RECORDING hRec)
```

Description

Arm device for recording.

Parameters

[in] *hRec*

A handle to an EC14150 recording session. This handle is obtained by calling the CreateRecordingSessionEC14 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 EC14150 for recording. This function is only needed if the EC14RECSEF_DO_NOT_ARM recording flag is specified during creation of the recording session.

Related Functions

[CreateRecordingSessionEC14](#)

3.13.3 | CreateRecordingSessionEC14

Form

```
int CreateRecordingSessionEC14 (HEC14 hBrd, EC14S\_REC\_SESSION\_PARAMS\* rec_paramsp, HEC14RECORDING\* handlep)
```

Description

Create an EC14150 acquisition recording session.

Parameters

[in] *hBrd*

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] *rec_paramsp*

A pointer to a [EC14S_REC_SESSION_PARAMS](#) structure that define the parameters of the data recording session. This structure and its members are detailed in the [Structure EC14S_REC_SESSION_PARAMS](#) section.

[out] *handlep*

A pointer to a HEC14RECORDING 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 an EC14150 recording session. A recording session is an EC14150 library interface used to fully manage the recording of EC14150 acquisition data to permanent storage.

Related Functions

[DeleteRecordingSessionEC14](#)

3.13.4 | DeleteRecordingSessionEC14

Form

```
int DeleteRecordingSessionEC14 (HEC14RECORDING hRec)
```

Description

Delete an EC14150 recording session.

Parameters

[in] *hRec*

A handle to an EC14150 recording session. This handle is obtained by calling the CreateRecordingSessionEC14 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 AbortRecordingSessionEC14.

Related Functions

[CreateRecordingSessionEC14](#)

3.13.5 | GetRecordingSessionOutFlagsEC14

Form

```
int GetRecordingSessionOutFlagsEC14 (HEC14RECORDING hRec, unsigned int* flagsp)
```

Description

Obtain output flags of the specified recording session. To be used when the recording session is stopped, and before deleting the session.

Parameters

[in] *hRec*

A handle to an EC14150 recording session. This handle is obtained by calling the CreateRecordingSessionEC14 function.

[out] *flagsp*

A pointer to an unsigned int variable that will receive the flags. Here are the possible flag values:

EC14150 Library Constant	Value	Interpretation
EC14FILWOUTF_TIMESTAMP_FIFO_OVERFLOW	0x00000001	Timestamp FIFO overflowed during write/record process.
EC14FILWOUTF_NO_TIMESTAMP_DATA	0x00000002	No timestamp data was available during the operation.

Return Value

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

3.13.6 | GetRecordingSessionProgressEC14

Form

```
int GetRecordingSessionProgressEC14 (HEC14RECORDING hRec, EC14S_REC_SESSION_PROG* prog, unsigned flags = 0)
```

Description

Obtain progress/status for current recording session.

Parameters

[in] *hRec*

A handle to an EC14150 recording session. This handle is obtained by calling the CreateRecordingSessionEC14 function.

[out] *prog*

A pointer to an EC14S_REC_SESSION_PROG structure that will receive the current progress/status of the EC14150 recording session. The caller should initialize the struct_size field of this structure before calling this function.

[in] *flags*

A set of flags (EC14RECPROGF_*) that control function behavior. Currently, only a single flag is defined: EC14RECPROGF_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.

Remarks

This function is used to arm the EC14150 for recording. This function is only needed if the EC14RECSESF_DO_NOT_ARM recording flag is specified during creation of the recording session.

Related Functions

[CreateRecordingSessionEC14](#), [GetRecordingSnapshotEC14](#)

3.13.7 | GetRecordingSnapshotEC14

Form

```
int GetRecordingSnapshotEC14 (HEC14RECORDING hRec, ec14\_sample\_t* bufp, unsigned int samples, unsigned int* samples_gotp, unsigned int* ss_countp)
```

Description

Obtain data snapshot from current recording.

Parameters

[in] *hRec*

A handle to an EC14150 recording session. This handle is obtained by calling the CreateRecordingSessionEC14 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

[CreateRecordingSessionEC14](#), [GetRecordingSessionProgressEC14](#)

3.14 | Signatec Recorded Data Context (SRDC) Information

The EC14150 library supports the saving of EC14150 acquisition data to external files. This is usually done by running a recording (see [EC14150 Recording Session Management Routines](#)) or by copying data from EC14150 RAM (see [ReadSampleRamFastEC14](#)). In both cases, by default, only sample data is written to disk; no additional header or context information is written to the file. There are two primary reasons for this; first and foremost, it allows for fast access to the underlying data. This is important if the data is to be used for a corresponding playback and needs to be pulled off disk quickly. The other reason is that it keeps things generic; introducing extra context information may be incompatible with other software systems that will read the sample data.

The disadvantage to acquisition-data-only files is that it can get confusing trying to keep up with what kind of data is in recorded data files (*.rd16) since things like channel count, segment size, etc are not stored anywhere. This is where Signatec Recorded Data Context (SRDC) information comes in. This information is additional context information that describes the data in a corresponding data file. This context data is stored in XML format and can be extended to include any number of user defined items.

The EC14150 library can generate and store this SRDC data in one of two places: an auxiliary file or in an alternate file stream of the data file. In either case, the same data is generated; it's just the location of that data that varies.

Auxiliary File

This is the most portable method of SRDC data storage. Using this method, the SRDC data is stored in a secondary file located in the same directory as the data file that it describes. By convention, the name of the file will be the name of the data file appended with '.srdc'. As an example, if your output data file was "C:\Data\MyRecording.rd16" then the corresponding SRDC data file would be named "C:\Data\MyRecording.rd16.srdc". This naming convention allows software to quickly check to see if SRDC information is available for a given .rd16 file.

Signatec Recorded Data Context Format

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>EC14150</SourceBoard>
  <SourceBoardSerialNum>100147</SourceBoardSerialNum>
<!--Data sample information-->
  <ChannelCount>1</ChannelCount>
  <ChannelId>1</ChannelId>
  <SampleSizeBytes>2</SampleSizeBytes>
  <SampleSizeBits>16</SampleSizeBits>
  <SampleFormat>Unsigned</SampleFormat>
<!--Data acquisition information-->
  <SamplingRateMHz>150</SamplingRateMHz>
  <PeakToPeakInputVoltRange>1.0</PeakToPeakInputVoltRange>
  <InputGain_dB>6</InputGain_dB>
  <SegmentSize>0</SegmentSize>
  <PreTriggerSampleCount>0</PreTriggerSampleCount>
```

```

<!--User-defined items-->
<OperatorNotes>Sample operator notes....</OperatorNotes>
<SomeUserDefinedItem>User defined data abc123</SomeUserDefinedItem>
</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's an item "ChannelCount" with a value of "1". 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 EC14150 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 an EC14150, this value will be "EC14150".	<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 an EC14150.	2
SampleSizeBits	The size of a data sample in bits. This will be 16 for all data generated by an EC14150. (2 least significant bits will always be zero.)	16
SampleFormat	The format of a data sample. May be either "Signed" or "Unsigned". EC14150 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

Continued:

Item Name	Value Interpretation	Default Value
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 <i>FileFormat</i> 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>

The following library function are used to programmatically interact with SRDC data files

3.14.1 | CloseSrdcFileEC14

Form

```
int CloseSrdcFileEC14 (HEC14SRDC hFile)
```

Description

Close given SRDC file without updating contents.

Parameters

[in] *hFile*

A handle to a SRDC file previously opened by calling the [OpenSrdcFileEC14](#) 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 disk. Call the [SaveSrdcFileEC14](#) function to ensure all modifications are written to disk.

Related Functions

[OpenSrdcFileEC14](#), [SaveSrdcFileEC14](#)

3.14.2 | EnumSrdclItemsEC14

Form

```
int EnumSrdclItemsAEC14 (HEC14SRDC 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 [OpenSrdcFileEC14](#) 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
EC14SRDCENUM_SKIP_STD	0x00000001	Skip standard SRDC items; use to obtain only user-defined items.
EC14SRDCENUM_SKIP_USER_DEFINED	0x00000002	Skip user-defined SRDC items; use to obtain only standard items.
EC14SRDCENUM_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 | GetRecordedDataInfoEC14

Form

```
int GetRecordedDataInfoEC14 (const char* pathnamep, EC14S\_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 `EC14S_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 `FreeMemoryEC14` 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 [EC14S_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 | GetSrdcItemEC14

Form

```
int GetSrdcItemEC14 (HEC14SRDC 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 [OpenSrdcFileEC14](#) 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 [FreeMemoryEC14](#) 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_EC14_NAMED_ITEM_NOT_FOUND will be returned.

Related Functions

[SetSrdcltemEC14](#)

3.14.5 | IsSrdcFileModifiedEC14

Form

int IsSrdcFileModifiedEC14 (HEC14SRDC 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 [OpenSrdcFileEC14](#) 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 [SetSrdcltemEC14](#) function is used to modify the value of a specific SRDC data item. The RefreshSrdcParametersEC14 function can also change one or more standard SRDC data item values.

Saving SRDC data via the [SaveSrdcFileEC14](#) function will reset the modified state of all SRDC items.

3.14.6 | OpenSrdcFileEC14

Form

```
int OpenSrdcFileEC14 (HEC14 hBrd, HEC14SRDC* handlep, const TCHAR* pathnamep, unsigned flags = 0)
```

Description

Open a new or existing .srdc file.

Parameters

[in] *pBrd*

A handle to the EC14150 board. This handle is obtained by calling the [ConnectToDeviceEC14](#) function.

[out] *handlep*

A pointer to a HEC14SRDC 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 (EC14SRDOF_*) that define function behavior. Currently defined flags are:

Flag	Value	Interpretation
EC14SRDCOF_QUICK_SET	0x00000001	Opens/create file, refresh and write settings, then close file. See Remarks.
EC14SRDCOF_OPEN_EXISTING	0x00000002	Open existing SRDC file; function will fail if file does not exist.
EC14SDRCOF_CREATE_NEW	0x00000004	Create a new SDRC file; will ignore and overwrite previously existing data.
EC14SRDCOF_PATHNAME_IS_REC_DATA	0x00000008	Given pathname is recorded data file; have library search for SRDC data. Using this flag implies EC14SRDCOF_OPEN_EXISTING. This flag cannot be used with EC14SDRCOF_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 EC14SRDCOF_OPEN_EXISTING and EC14SRDCOF_CREATE_NEW are mutually exclusive. The library will return an error (SIG_EC14_INVALID_ARG_4) if both are set. Also note that use of EC14SRDCOF_PATHNAME_IS_REC_DATA implies EC14SRDCOF_OPEN_EXISTING.

If the EC14SRDCOF_QUICK_SET flag is set, then the function will open the file (if it exists), refresh standard SRDC context items (by calling [RefreshSrdcParametersEC14](#)), save changes, and close the file. In this case, no SRDC file handle is returned.

Related Functions

[SaveSrdcFileEC14](#), [CloseSrdcFileEC14](#), [SetSrdcItemEC14](#), [GetSrdcItemEC14](#), [RefreshSrdcParametersEC14](#)

3.14.7 | RefreshSrdcParametersEC14

Form

```
int RefreshSrdcParametersEC14 (HEC14SRDC 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 [OpenSrdcFileEC14](#) 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, and PreTriggerSampleCount.

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 SaveSrdcFileEC14 function is called.

Related Functions

[OpenSrdcFileEC14](#), [SaveSrdcFileEC14](#)

3.14.8 | SaveSrdcFileEC14

Form

```
int SaveSrdcFileEC14 (HEC14SRDC hFile, const TCHAR* pathnamep)
```

Description

Write SRDC data to file.

Parameters

[in] *hFile*

A handle to a SRDC file, previously opened by calling the [OpenSrdcFileEC14](#) 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

[OpenSrdcFileEC14](#)

3.14.9 | SetSrdcltemEC14

Form

```
int SetSrdcltemEC14 (HEC14SRDC 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 [OpenSrdcFileEC14](#) 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 disk until the SaveSrdcFileEC14 function is called.

Modifying the value of a SRDC item will result in that item's internal "modified" bit to be set.

Related Functions

[OpenSrdcFileEC14](#), [GetSrdcItemEC14](#), [SaveSrdcFileEC14](#)

3.15 | EC14150 Library Data Types

The following sections define some of the data types used by the EC14150 library.

3.15.1 | Data Type: HEC14

This data type is used to represent a handle to an EC14150 device.

A special library constant `EC14_INVALID_HANDLE` (NULL) represents an invalid EC14150 device handle. Any other value should be considered opaque; this value only has relevance to the EC14150 library implementation.

EC14150 handles are only valid in the process for which they are obtained.

The `DisconnectFromDeviceEC14` library function should be called to release an EC14150 device handle when it is no longer needed. This will free handle-specific resources allocated by the library.

3.15.2 | Data Type: HEC14RECORDING

This data type is used to represent a handle to a [EC14150 Recording Session](#).

A special library constant `INVALID_HEC14RECORDING_HANDLE` (NULL) represents an invalid EC14150 Recording Session handle. Any other value should be considered opaque; this value only has relevance to the EC14150 library implementation.

EC14150 Recording Session Handles are only valid in the process for which they are obtained.

3.15.3 | Data Type: HEC14SRDC

This data type is used to represent a handle to a [Signatec Recorded Data Context \(SRDC\)](#) file.

A special library constant `EC14_INVALID_HEC14SRDC` (NULL) represents an invalid SRDC file handle. Any other value should be considered opaque; this value only has relevance to the EC14150 library implementation.

3.15.4 | Data Type: `ec14_sample_t`

This data type is used to represent an EC14150 data sample. It is typically defined as an 'unsigned short' (unsigned 16-bit integral value).

Currently, the EC14150 will only generate little-endian, unsigned data.

3.15.5 | Structure: EC14S_FILE_WRITE_PARAMS

This structure is used by a few of the EC14150 library functions that save EC14150 board data (either from EC14150 RAM or from an EC14150 recording session) to one or more files on the host system.

Structure definition

```
typedef struct _EC14S_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;                 // EC14FILWF_*
    unsigned int    init_bytes_skip;       // Initial bytes to skip in file
    unsigned long long max_file_seg;       // Max file size in samples; binary only

    EC14_FILEIO_CALLBACK pfnCallback;     // Optional progress callback
    uintptr_t        callbackCtx;         // User-defined callback context

    unsigned int*    flags_out;            // Output flags; EC14FILWOUTF_*
    const char*      operator_notes;       // User-defined notes
    const char*      ts_filenamep;        // Timestamp filename override
} EC14S_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
EC14FILWF_APPEND	0x00000001	Append if file already exists; default is to overwrite existing file.
EC14FILWF_ALWAYS_SKIPBYTES	0x00000002	Use <i>bytes_skip</i> even if appending to a file. By default, if appending to an existing file, the <i>bytes_skip</i> field is ignored.

Continued:

Library constant	Value	Interpretation
EC14FILWF_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 recording PCI acquisition data.
EC14FILWF_NO_UNBUFFERED_IO	0x00000008	<i>Windows:</i> Don't try to use fast, unbuffered IO. If this flag is not set then the library may try to use the <code>FILE_FLAG_NO_BUFFERING</code> 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.
EC14FILWF_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.
EC14FILWF_HEX_OUTPUT	0x00000020	Use hexadecimal output; applies to text saves
EC14FILWF_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 <i>pathname</i> field. Channel 2 data will be written to the file specified by the <i>pathname2</i> field. Either one of these parameters can be NULL if that channel is not needed.
EC14FILWF_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.

Continued:

Library constant	Value	Interpretation
EC14FILWF_GENERATE_SRDC_FILE	0x00000100	Generate a Signatec Recorded Data Context (.srdc) file for each output file.
EC14FILWF_EMBED_SRDC_AS_AFS	0x00000200	Embed Signatec Recorded Data Context information in data file as NTFS alternate file stream (:SRDC).
EC14FILWF_SAVE_TIMESTAMPS	0x00000400	Save EC14150 timestamp data in external file.*
EC14FILWF_TIMESTAMPS_AS_TEXT	0x00000800	Save timestamps as newline-delimited text; use with EC14FILWF_SAVE_TIMESTAMPS flag.*
EC14FILWF_USE_TS_FIFO_OVFL_MARKER	0x00001000	Use timestamp FIFO overflow marker: F1F0F1F0F1F0F1F0 F1F0F1F0F1F0F1F0.*
EC14FILWF_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: EC14 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 `GetRecordingSessionOutFlagsEC14` 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.

EC14150 Library Constant	Value	Interpretation
EC14FILWOUTF_TIMESTAMP_FIFO_OVERFLOW	0x00000001	Timestamp FIFO overflowed during write/record process.*
EC14FILWOUTF_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 `EC14FILWF_GENERATE_SRDC_FILE` or `EC14FILWF_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 `EC14FILWF_SAVE_TIMESTAMPS` flag is set in the flags member. See remarks below for details on how the timestamp data file is named.

Remarks

***Timestamps not currently available.**

Related Functions

[ReadSampleRamFileFastEC14](#), [ReadSampleRamFileBufEC14](#), [CreateRecordingSessionEC14](#)

3.15.6 | Structure: EC14S_REC_SESSION_PARAMS

This structure is used by the [CreateRecordingSessionEC14](#) function to specify the various parameters that define how an EC14150 recording session will run.

Structure definition

```
typedef struct _EC14S_REC_SESSION_PARAMS_tag
{
    unsigned int      struct_size;           // Size of structure in bytes

    int               rec_flags;             // EC14RECSESF_*
    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

    EC14S_FILE_WRITE_PARAMS* filwp;         // Defines output file(s)

    ec14_sample_t*     dma_bufp;             // Optional DMA buffer address
    unsigned           dma_buf_samples;      // Size of DMA buffer in samps

    // Data snapshot parameters; valid when EC14RECSESF_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

    EC14_REC_CALLBACK  pfnCallback;          // Optional callback
    void*              callbackCtx;          // Context data for callback
} EC14S_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 EC14150 recording session behavior. Currently defined flags are:

Library Constant	Value	Interpretation
EC14RECSESF_REC_PCI_ACQ	0x00000001	Do an EC14150 RAM-buffered PCI acquisition recording
EC14RECSESF_REC_RAM_ACQ	0x00000002	Do an EC14150 RAM acquisition/transfer recording
EC14RECSESF_REC__MASK	0x0000000F	Mask for session recording type
EC14RECSESF_DO_NOT_ARM	0x00000010	Do not auto arm recording; will be done with ArmRecordingSessionEC14
EC14RECSESF_DO_SNAPSHOTS	0x00000020	Periodically obtain snapshots of recording data

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 `AbortRecordingSessionEC14` 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 [AcquireToBoardRamEC14](#) function. This parameter is ignored for PCI-based recordings.

xfer_samples

Defines the DMA transfer length that should be used for PCI-based acquisition recordings. Pass zero to have the library determine what the transfer size should be.

filwp

A pointer to a [EC14S_FILE_WRITE_PARAMS](#) structure that defines how data will be saved to persistent storage. See the [Structure EC14S_FILE_WRITE_PARAMS](#) section for details on this structure. Note that the EC14150 library supports many different ways of saving data and some of these (saving as text for instance) can be quite slow. For PCI-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 `EC14RECSESF_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 PCI acquisitions, 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 `EC14RECSESF_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 EC14150 library will default back to a simple, approximate-timed approach for recording data snapshot updates. This field is only considered when the `EC14RECSESF_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 EC14150 library currently supports two types of recording sessions: PCI Acquisition and RAM Acquisition/Transfer.

PCI Acquisition Recording Sessions

A PCI Acquisition recording session uses the EC14150's [RAM-Buffered PCI Acquisition](#) operating mode to acquire data directly to the PCI bus, using the onboard RAM as a large FIFO to help buffer the data against software latencies introduced by non-real-time environments like Windows and Linux.

A PCI Acquisition recording session is specified by setting the EC14RECSESF_REC_PCI_ACQ flag in the *rec_flags* field.

RAM Acquisition/Transfer Recording Sessions

A RAM Acquisition/Transfer recording session uses the EC14150's [RAM Acquisition](#) operating mode to acquire data into EC14150 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.

It is important to note that with this type of recording, multiple, independent acquisitions are performed so if a continuous, uninterrupted stream of data is needed the PCI Acquisition recording session should be used.

A RAM Acquisition/Transfer recording session is specified by setting the EC14RECSESF_REC_RAM_ACQ flag in the *rec_flags* field.

3.15.7 | Structure: EC14S_RECORDERD_DATA_INFO

This structure is used by the [GetRecordedDataInfoEC14](#) function to contain details on acquisition data that was previously saved to disk.

Structure definition

```
typedef struct _EC14S_RECORDERD_DATA_INFO_tag
{
    unsigned int      struct_size;          // Size of structure in bytes

    char              boardName[16];        // Name of board: "EC14150"
    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)
} EC14S_RECORDERD_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 EC14150 generated data this field will contain "EC14150".

boardSerialNum

The serial number of the board that generated the acquisition data.

channelCount

The channel count of the underlying data. For EC14150 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 EC14150 data, this will always be 2.

sampSizeBits

The size of a single data sample in bits. Though the EC14150 only has 14 effective bits, it will always use 16 for this field since it's the lower bits that are zero instead of the upper 2 bits.

bSignedSamples

This field will be non-zero if the underlying data samples are signed ($\pm N$). By default, EC14150 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 EC14150 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.

3.15.8 | Callback Function: EC14_FILEIO_CALLBACK

This item defines the signature of a callback function that is used when saving EC14150 data to file.

Prototype

```
int SomeCallbackFunction (HEC14 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 EC14150 device for which data is being saved.

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 – EC14150A Specifications

External Signal Connections

Analog Input, Channel 1	: MMCX (miniature RF)
Analog Input, Channel 2	: MMCX
Clock Input	: MMCX
Trigger Input	: MMCX
Digital I/O	: MMCX

Analog Inputs

Full-Scale Volt. Ranges	: 200mV to 6.3V (1 dB increments)
Impedance	: 50 ohms
Bandwidth	: 200 MHz (LP filter)
Coupling	: AC

External Trigger

Signal Type	: digital, TTL signal level
Impedance	: >10k ohms
Bandwidth	: 50 MHz

Internal Synthesized Clock

Frequency range	: 45.0 - 150 MHz
Resolution	: better than 5 PPM
Accuracy	: better than 5 PPM
Unsettable ranges	: 128.9-129.8, 140.6-142.8 MHz

External Clock

Signal Type	: sine wave or square wave
Coupling	: AC
Impedance	: 50 ohms
Frequency	: 10 MHz to 150 MHz
Amplitude	: 100 mV p-p to 2.0 V p-p

Post ADC Clock Divider

Divider Settings	: 1, 2, 4, 8, 16, 32
------------------	----------------------

Reference Clock

Internal	: 10.0 MHz, ± 5 ppm max.
External	: 10.0 MHz, ± 50 ppm max (required for lock)

Digitizer

Resolution	: 14 bits
Aperture Jitter	: 0.07 pS typical
Clock Rate	: 45 to 150 MHz

Trigger Modes

Post Trigger	: single start trigger fills active memory
Segmented	: start trigger for each memory segment

Trigger Options

Pretrigger Samples	: samples prior to trigger are stored; Single Channel: 8k max.; Dual Channel: 4k max per channel
--------------------	--

Memory

Active Size	: Up to 256 MegaSamples
Segment Size	: Up to 128 Megasamples
Segment re-arm time ¹	: 150 nanoseconds
Addressing	: DMA transfer from starting address

I/O Addressing

PCI Controller Address	: 64 bytes, Plug and Play selected
Control/Status Registers	: 32 bytes, Plug and Play selected

Performance

SFDR	: 80dB, 1 to 50 MHz : 78dB @ 100 MHz : 77dB @ 170 MHz
SNR (1-200 MHz)	: 67dB @ 20dB gain : 66dB @ 10dB gain : 65dB @ 0dB gain

Power Requirements

+3.3V	: 1.5 Amps max.
-------	-----------------

Absolute Maximum Ratings

Analog Inputs	: ± 5 volts
Trigger Input	: -0.2 to +4.0 volts DC
Clock Input	: 5 volts peak to peak
Ambient Temperature	: 0 to 50 °C

Definition of Terms

SNR: Signal to Noise Ratio - The ratio of the fundamental sinusoidal power to the noise power. For this data sheet, noise power is considered to be the power from all spectral components with the exception of the fundamental signal, the first harmonic, and the second harmonic.

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.

EC14150A Part Numbers

EC14150A

Documentation & Accessories

The EC14150A is supplied with a comprehensive operator's manual, which thoroughly describes the operation of both the hardware and the software. Also supplied are two four-foot coaxial cables with MMCX to BNC connectors. Additional cables may be purchased. Supplied software disks contain a function library for Microsoft Visual C/C++, example programs, and all source code to examples.

Product Warranty

All Signatec products carry a standard full 2-year warranty. During the warranty period, DynamicSignals will repair or replace any defective product at no cost to the customer. Warranties do not cover customer misuse or abuse of the products.

Notes

1. In segmented mode, time from the end of a segment until a trigger will be accepted to begin another segment acquisition.

DynamicSignals reserves the right to make changes in this specification at any time without notice. The information furnished herein is believed to be accurate, however no responsibility is assumed for its use.

Data Sheet Revision 1.07 – 05/12/2014

5 | Appendix B – EC14150D Specifications

External Signal Connections

Analog Input, Channel 1	: MMCX (miniature RF)
Analog Input, Channel 2	: MMCX
Clock Input	: MMCX
Trigger Input	: MMCX
Digital I/O	: MMCX

Analog Inputs

Full-Scale Volt. Ranges	: 250mV, 500mV, 1.0V, 2.0V
Impedance	: 50 ohms
Bandwidth	: 75 MHz (Bessel LP filter, 3rd order)
Coupling	: DC

External Trigger

Signal Type	: digital, TTL signal level
Impedance	: >10k ohms
Bandwidth	: 50 MHz

Internal Synthesized Clock

Frequency range	: 45.0 - 150 MHz
Resolution	: better than 5 PPM
Accuracy	: better than 5 PPM
Unsettable ranges	: 128.9-129.8, 140.6-142.8 MHz

External Clock

Signal Type	: sine wave or square wave (mandatory : below 45 MHz)
Coupling	: AC
Impedance	: 50 ohms
Frequency	: 10 MHz to 150 MHz
Amplitude	: 100 mV p-p to 2.0 V p-p

Post ADC Clock Divider

Divider Settings	: 1, 2, 4, 8, 16, 32
------------------	----------------------

Reference Clock

Internal	: 10.0 MHz, ± 5 ppm max.
----------	------------------------------

Digitizer

Resolution	: 14 bits
Aperture Jitter	: 0.1 pS typical
Clock Rate	: 45 to 150 MHz

Trigger Modes

Post Trigger	: single start trigger fills active memory
Segmented	: start trigger for each memory segment

Trigger Options

Pretrigger Samples	: samples prior to trigger are stored; Single Channel: 8k max.; Dual Channel: 4k max per channel
--------------------	--

Memory

Active Size	: Up to 256 MegaSamples
Segment Size	: Up to 128 Megasamples
Segment re-arm time ¹	: 150 nanoseconds
Addressing	: DMA transfer from starting address

I/O Addressing

PCI Controller Address	: 64 bytes, Plug and Play selected
Control/Status Registers	: 32 bytes, Plug and Play selected

Performance

SFDR (DC – 75 MHz)	: 78dB (2V range at 95% F.S.) : 75dB (1V, 500mV ranges at 50% F.S.)
SNR (DC – 75 MHz)	: 70dB (2V, 1V, & 500mV ranges) : 64dB (250mV range)

Power Requirements

+3.3V	: 1.5 Amps max.
+1.5V	: 0.2 Amps max.

Absolute Maximum Ratings

Analog Inputs	: ± 2 volts
Trigger Input	: -0.2 to +4.0 volts DC
Clock Input	: 5 volts peak to peak
Ambient Temperature	: 0 to 50 C

Definition of Terms

SNR: Signal to Noise Ratio - The ratio of the fundamental sinusoidal power to the noise power. For this data sheet, noise power is considered to be the power from all spectral components with the exception of the fundamental signal, the first harmonic, and the second harmonic.

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.

EC14150D Part Numbers

EC14150D

Documentation & Accessories

The EC14150D is supplied with a comprehensive operator's manual, which thoroughly describes the operation of both the hardware and the software. Also supplied are two four-foot coaxial cables with MMCX to BNC connectors. Additional cables may be purchased. Supplied software disks contain a function library for Microsoft Visual C/C++, example programs, and all source code to examples.

Product Warranty

All Signatec products carry a standard full 2-year warranty. During the warranty period, DynamicSignals will repair or replace any defective product at no cost to the customer. Warranties do not cover customer misuse or abuse of the products.

Notes

1. In segmented mode, time from the end of a segment until a trigger will be accepted to begin another segment acquisition.

DynamicSignals reserves the right to make changes in this specification at any time without notice. The information furnished herein is believed to be accurate, however no responsibility is assumed for its use.

Data Sheet Revision 1.03 – 05/12/2014

6 | Appendix C – EC14150 Library Error Codes

The table below lists all of the currently defined EC14150 library error codes.

The `GetErrorTextEC14` library function can be used to generate a user-friendly string for any of these error codes.

Symbolic constant	Error Code	Interpretation
<code>SIG_SUCCESS</code>	0	Operation successful
<code>SIG_EC14_QUASI_SUCCESSFUL</code>	512	Operation was quasi-successful; one or more items failed and one or more items succeeded
<code>SIG_ERROR</code>	-1	Generic error; platform's system error may provide more info
<code>SIG_INVALIDARG</code>	-2	An invalid argument was specified
<code>SIG_OUTOFBOUNDS</code>	-3	An argument is out of valid bounds
<code>SIG_NODEV</code>	-4	Invalid board device
<code>SIG_OUTOFMEMORY</code>	-5	Error allocating memory
<code>SIG_DMABUFALLOCFAIL</code>	-6	Error allocating a DMA buffer
<code>SIG_NOSUCHBOARD</code>	-7	Board with given serial or ordinal number not found.
<code>SIG_NT_ONLY</code>	-8	This feature is only available on Windows NT platforms.
<code>SIG_INVALID_MODE</code>	-9	Invalid operation for current operating mode
<code>SIG_CANCELLED</code>	-10	Operation was cancelled by user
<code>SIG_EC14_FIRST</code>	-512	First EC14150-specific error code value
<code>SIG_EC14_NOT_IMPLEMENTED</code>	-512	This operation is not currently implemented
<code>SIG_EC14_INVALID_HANDLE</code>	-513	An invalid EC14150 device handle (HEC14) was specified
<code>SIG_EC14_BUFFER_TOO_SMALL</code>	-514	A specified buffer is too small
<code>SIG_EC14_INVALID_ARG_1</code>	-515	Argument 1 is invalid
<code>SIG_EC14_INVALID_ARG_2</code>	-516	Argument 2 is invalid
<code>SIG_EC14_INVALID_ARG_3</code>	-517	Argument 3 is invalid
<code>SIG_EC14_INVALID_ARG_4</code>	-518	Argument 4 is invalid
<code>SIG_EC14_INVALID_ARG_5</code>	-519	Argument 5 is invalid
<code>SIG_EC14_INVALID_ARG_6</code>	-520	Argument 6 is invalid
<code>SIG_EC14_INVALID_ARG_7</code>	-521	Argument 7 is invalid
<code>SIG_EC14_INVALID_ARG_8</code>	-522	Argument 8 is invalid
<code>SIG_EC14_XFER_SIZE_TOO_SMALL</code>	-523	Requested transfer size is too small
<code>SIG_EC14_XFER_SIZE_TOO_LARGE</code>	-524	Requested transfer size is too large
<code>SIG_EC14_INVALID_DMA_ADDR</code>	-525	Given address is not part of a DMA buffer
<code>SIG_EC14_WOULD_OVERRUN_BUFFER</code>	-526	Operation would overrun given buffer
<code>SIG_EC14_BUSY</code>	-527	Device is busy; try again later
<code>SIG_EC14_INVALID_CHAN_IMP</code>	-528	Incorrect function for board's channel implementation
<code>SIG_EC14_XML_MALFORMED</code>	-529	Invalid XML data was encountered

Continued:

Symbolic constant	Error Code	Interpretation
SIG_EC14_XML_INVALID	-530	XML data was well formed, but not valid
SIG_EC14_XML_GENERIC	-531	Generic XML related error
SIG_EC14_RATE_TOO_FAST	-532	The specified rate is too fast
SIG_EC14_RATE_TOO_SLOW	-533	The specified rate is too slow
SIG_EC14_RATE_NOT_AVAILABLE	-534	The specified frequency is not available; see operator's manual
SIG_EC14_UNEXPECTED	-535	An unexpected error occurred; debug builds will have failed assertion
SIG_EC14_SOCKET_ERROR	-536	A socket error occurred
SIG_EC14_NETWORK_NOT_READY	-537	Network subsystem is not ready for network communication
SIG_EC14_SOCKETS_TOO_MANY_TASKS	-538	Limit on number of tasks/processes using sockets has been reached
SIG_EC14_SOCKETS_INIT_ERROR	-539	Generic sockets implementation start up failure
SIG_EC14_NOT_REMOTE	-540	Not connected to a remote EC14150 device
SIG_EC14_TIMED_OUT	-541	Operation timed out
SIG_EC14_CONNECTION_REFUSED	-542	Connection refused by service; service may not be running
SIG_EC14_INVALID_CLIENT_REQUEST	-543	Received an invalid client request
SIG_EC14_INVALID_SERVER_RESPONSE	-544	Received an invalid server response
SIG_EC14_REMOTE_CALL_RETURNED_ERROR	-545	Remote service call returned with an error
SIG_EC14_UNKNOWN_REMOTE_METHOD	-546	Undefined method invoked on remote server
SIG_EC14_SERVER_DISCONNECTED	-547	Server closed the connection
SIG_EC14_REMOTE_CALL_NOT_AVAILABLE	-548	Remote call for this operation is not implemented or available
SIG_EC14_UNKNOWN_FW_FILE	-549	Unknown firmware file type
SIG_EC14_FIRMWARE_UPLOAD_FAILED	-550	Firmware upload failed
SIG_EC14_INVALID_FW_FILE	-551	Invalid firmware upload file
SIG_EC14_DEST_FILE_OPEN_FAILED	-552	Failed to open destination file
SIG_EC14_SOURCE_FILE_OPEN_FAILED	-553	Failed to open source file
SIG_EC14_FILE_IO_ERROR	-554	File IO error
SIG_EC14_INCOMPATIBLE_FIRMWARE	-555	Firmware is incompatible with EC14150
SIG_EC14_UNKNOWN_STRUCT_VER	-556	Unknown structure version specified to library function (X::struct_size)
SIG_EC14_INVALID_REGISTER	-557	An invalid hardware register read/write was attempted
SIG_EC14_FIFO_OVERFLOW	-558	An internal FIFO overflowed during acquisition; couldn't keep up with data rate
SIG_EC14_DCM_SYNC_FAILED	-559	EC14150 firmware could not synchronize to acquisition clock
SIG_EC14_DISK_FULL	-560	Could not write all data; disk is full
SIG_EC14_INVALID_OBJECT_HANDLE	-561	An invalid object handle was used
SIG_EC14_THREAD_CREATE_FAILURE	-562	Failed to create a thread

Continued:

Symbolic constant	Error Code	Interpretation
SIG_EC14_PLL_LOCK_FAILED	-563	Phase lock loop (PLL) failed to lock; clock may be bad
SIG_EC14_THREAD_NOT_RESPONDING	-564	Recording thread is not responding
SIG_EC14_REC_SESSION_ERROR	-565	A recording session error occurred
SIG_EC14_REC_SESSION_CANNOT_ARM	-566	Cannot arm recording session; already armed or stopped
SIG_EC14_SNAPSHOTS_NOT_ENABLED	-567	Snapshots not enabled for given recording session
SIG_EC14_SNAPSHOT_NOT_AVAILABLE	-568	No data snapshot is available
SIG_EC14_SRD_FILE_ERROR	-569	An error occurred while processing .SRD file or stream-embedded SRD data
SIG_EC14_NAMED_ITEM_NOT_FOUND	-570	Named item could not be found
SIG_EC14_CANNOT_FIND_SRDC_DATA	-571	Could not find Signatec Recorded Data Context info
SIG_EC14_NOT_IMPLEMENTED_IN_FIRMWARE	-572	Feature is not implemented in current firmware version; upgrade firmware
SIG_EC14_TIMESTAMP_FIFO_OVERFLOW	-573	Timestamp FIFO overflowed during recording
SIG_EC14_CANNOT_DETERMINE_FW_REQ	-574	Cannot determine which firmware needs to be uploaded; update software
SIG_EC14_REQUIRED_FW_NOT_FOUND	-575	Required firmware not found in firmware update file
SIG_EC14_FIRMWARE_IS_UP_TO_DATE	-576	Loaded firmware is up to date with firmware update file
SIG_EC14_NO_VIRTUAL_IMPLEMENTATION	-577	Operation not implemented for virtual devices
SIG_EC14_DEVICE_REMOVED	-578	EC14150 device has been removed from system
SIG_EC14_FLASH_WRITE_FAILURE	-579	Failed to write to EC14 flash RAM
SIG_EC14_ACCESS_DENIED	-580	Access denied
SIG_EC14_PLL_LOCK_FAILED_UNSTABLE	-581	Phase lock loop (PLL) failed to lock; lock not stable
SIG_EC14_UNREACHABLE	-582	Item could not be set with the given constraints
SIG_EC14_INVALID_MODE_CHANGE	-583	Invalid operating mode change; all changes must go or come from Standby.
SIG_EC14_DEVICE_NOT_READY	-584	Underlying device is not yet ready; try again shortly

7 | Appendix D – Revision History

Revision 1.1 (Initial public release)

Revision 1.2

- Added DC-coupled input voltage ranges to [SetInputVoltageRangeCh1EC14](#)
- Added documentation for [GetBoardRevisionEC14](#) function.

Revision 1.3

- Added documentation for :
 - [SetDcOffsetChXEC14 / GetDcOffsetChXEC14](#)

Revision 1.4

- Updated Warranty section for new 2 year warranty period.
- Updated System Requirements section for supported OS versions
- Updated Physical Layout section for connector identification and maximum sustained data rate specification.
- Updated Functional Description Overview, Active Channels, and Input Voltage Range sections to further detail differences between EC14150A (AC-Coupled) and EC14150D (DC-Coupled) configurations.
- Updated Software Development Reference section to further detail Windows and Linux Software Installations and Build Environment information.
- Added missing documentation for API functions:
 - [GetAcqInProgressFlagEC14](#)
 - [GetBoardNameEC14](#)
 - [GetPllLockStatusEC14](#)
 - [GetSampleRamSizeEC14](#)
 - [GetVersionTextEC14](#)
 - [InAcquisitionModeEC14](#)
 - [InIdleModeEC14](#)
 - [SetUserDataEC14 / GetUserDataEC14](#)
 - [ValidateConfigEepromEC14](#)
 - [GetRecordingSessionOutFlagsEC14](#)
- Removed timestamps and trigger delay samples documentation throughout manual as these features are not yet available.
- **Updated Appendix A and inserted new Appendix B for EC14150A and EC14150D specifications.**