SYNC1500 Operators Manual

Revision 2.0 – 07/27/2015

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

IMPORTANT NOTICE
ON
HARDWARE COMPATIBILITY

The SYNC1500 is a PCI Express (PCIe) product and is a PCI Local Bus compliant device that implements a PCIe Gen1 x1 bus connection. As such the SYNC1500 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 SYNC1500 as well as storage of base address size requirements for SYNC1500 operation.

The host computer that the SYNC1500 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 SYNC1500 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 SYNC1500 inoperable. It is beyond the ability of Signatec hardware or software to force a non-PnP system to operate an SYNC1500 board.

1 | Before Using the SYNC1500

1.1 | Package Contents

The SYNC1500 package will normally contain (as a minimum) the following:

- SYNC1500 circuit board assembly
- SYNC1500 Software Disk (includes manuals)
- One MMCX to BNC coaxial cable (48 inches in length)
- MMCX to SMA coaxial cables (36 inches in length) in sufficient quantity to connect to each digitizer unit ordered with the SYNC1500

1.2 | Unpacking and Handling

The SYNC1500, 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 your body of any charge buildup by touching some large metal object which ideally is at earth potential.

1.3 | Checking for Damage

Carefully inspect the SYNC1500 package for any sign of physical damage such as dents in the box during shipment in addition to any noticeable physical damage on the card. Any such damage must be reported within 15 days from the date of actual shipment in order to be covered by warranty. To report such damage, contact Signatec, explain the nature of the damage, and request a RMA number for returning the merchandise.

1.4 | Warranty

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

1.5 | System Requirements

The SYNC1500 requires the following minimum hardware configuration:

- Availability of one open PCIe x1 slot for SYNC1500 card installation (can also be installed in larger PCIe x4, x8 or x16 slots)
- Intel Core Series or higher CPU
- Minimum of 4 GB system RAM
- Plug-n-Play system BIOS
- 32-bit or 64-bit Operating System: Windows 7 or Windows 8.x

1.6 | Software

The main SYNC1500 software installation installs several software components on the host system:

- A kernel-mode driver which allows the operating system to communicate with the SYNC1500 hardware.
- A user-mode library that interfaces with the driver. All SYNC1500 software accesses the SYNC1500 hardware through this library. This library exports a number of functions that may be used by custom software to control the SYNC1500 hardware. This library is documented in detail in the SYNC1500 Software Development Reference section.
- Several example SYNC1500 applications that demonstrate how to write SYNC1500 client programs. These example programs are located in the Examples directory of the main SYNC1500 software installation folder.
- The SYNC1500 Control Application. This is a full featured Windows application that configures and operates SYNC1500 devices.

1.7 | Physical Layout

The SYNC1500 is shown in Figure 1. The SYNC1500 features a total of 17 MMCX connectors on the bracket as shown with the identified function for each connector. Note that the SYNC1500 product bracket will have abbreviated labeling for the identified connections shown.

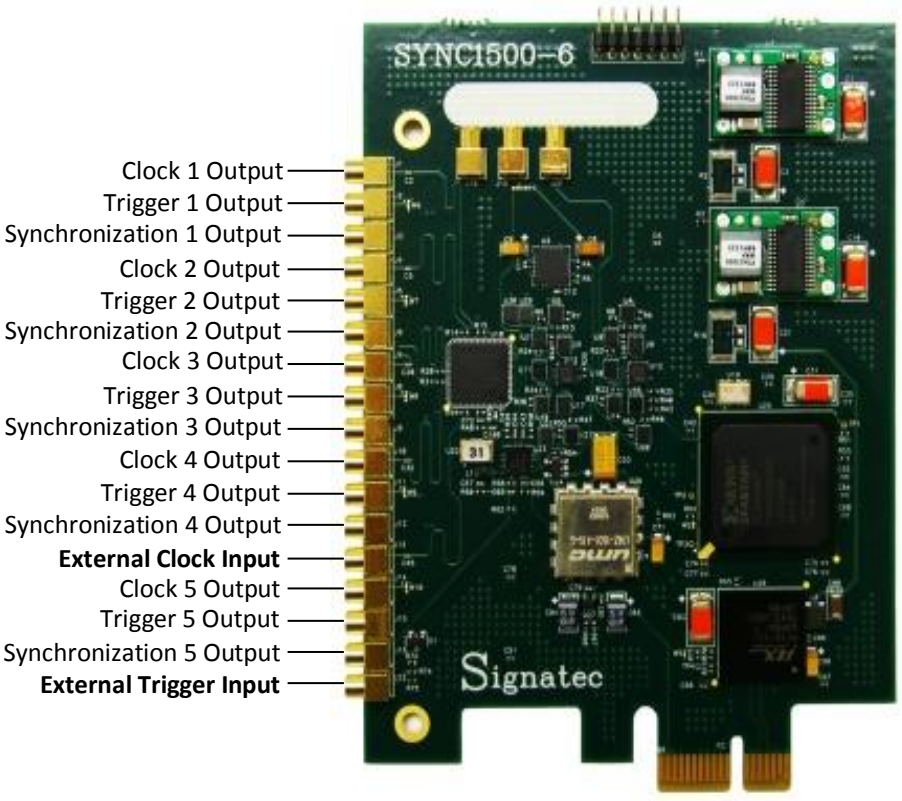

Figure 1: SYNC1500 Clock/Trigger Driver Card

2 | Functional Description

2.1 | Overview

The SYNC1500 was created to allow for both high board (or channel) count synchronized A/D and D/A systems as well as increased system scalability. With the SYNC1500 card module, synchronized channel record or playback systems are no longer bound to single chassis configurations and can span multiple chassis, which allows for significantly greater I/O and processing resources versus a single chassis system.

The SYNC1500 provides clock and trigger distribution for up to 5 data acquisition and arbitrary waveform generator boards. When utilized in conjunction with the Signatec PX1500-4 for example, a system can be mechanized incorporating up to 20 channels of fully synchronized, very-high-speed data acquisition.

The SYNC1500 board is designed for PC systems and utilizes a single lane PCI Express bus (PCIe) Gen1 interface. This PCIe Gen1 x1 interface is plug-in compatible with all PCIe lane configurations (x1, x4, x8 and x16).

An onboard frequency synthesized clock allows the output clock rate to be set to any value from 25 MHz to 1500 MHz, offering maximum flexibility for clock rate selection. The synthesized clock is locked to an onboard 10 MHz reference. An externally provided 10 MHz reference can also be selected.

Software is provided for setting up the various SYNC1500 board features including the output clock frequency (when using the onboard clock source).

The following figure is a simplified functional block diagram of the SYNC1500:

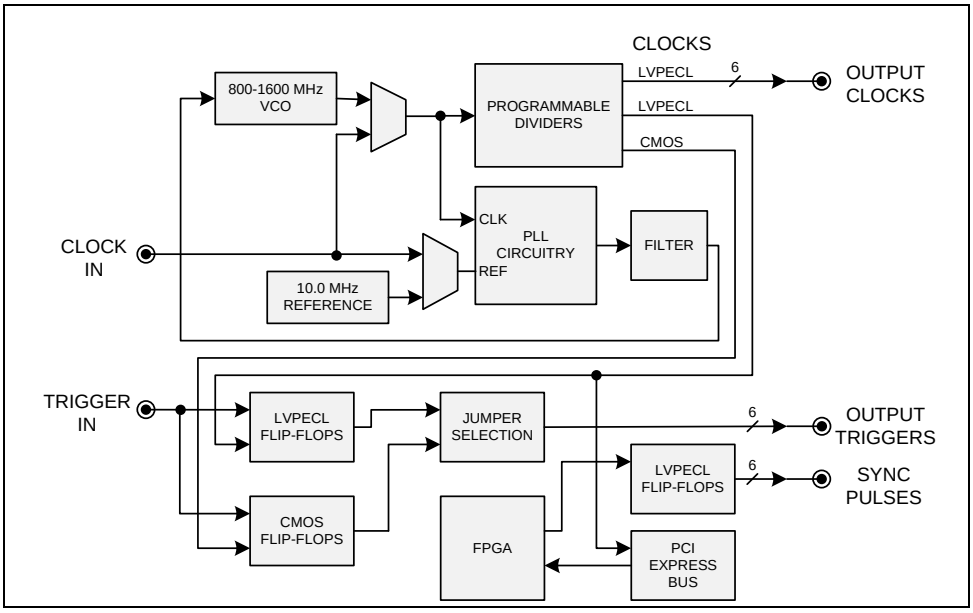

Figure 2: SYNC1500 Functional Block Diagram

2.2 | Output Clock Generation

The SYNC1500 can be configured to use one of two clock sources: internal or external. Whichever clock source is selected will be used to drive all enabled output clocks. Each output clock has an enable that must be turned on in order for the clock to be output.

Output clocks are enabled by calling SetClockOutputEnableSYNC6 or SetClockOutputEnableMaskSYNC6. The SetClockSourceSYNC6 function is used to select the clock source. By default, all clock outputs are enabled and the internal clock is selected as the clock source.

2.2.1 | Clock Source – Internal

The internal clock is a synthesized clock that can be configured for any integer multiple of 20 kHz in the range of [25, 1500] MHz. The error of the clock is ± 62.5 PPM (parts per million), but when locked to the internal reference clock this error is reduced to the internal reference clock accuracy error, which is 5 PPM max.

The SetInternalClockRateSYNC6 function is used to set the internal clock rate. By default, the internal clock rate is configured for 1500 MHz.

The internal clock is locked to a 10 MHz (± 5 PPM max) reference clock and is used in conjunction with the phase lock loop (PLL), which is used to maintain the desired internal clock rate. The SYNC1500 can be configured to use an externally supplied 10 MHz (± 50 PPM max) reference clock. When selected, the external 10 MHz clock is provided by the External Clock connector.

The SetClockRefSourceSYNC6 function is used to select the 10 MHz reference clock source. By default, the internal 10 MHz reference is selected as the internal clock reference source.

2.2.2 | Clock Source – External

When the external clock is selected as the clock source, the SYNC1500 will use the clock provided on the card's External Clock Input connector to drive the output clocks.

When using the external clock, the operator should ensure that the SYNC1500 software is kept up-to-date with the specified external clock rate that is applied to the card. The SYNC1500 needs to internally synchronize itself with the external clock and the software will need to know the external clock rate so that it can properly configure the SYNC1500 hardware.

The SetExternalClockRateSYNC6 function is used to set the external clock rate. By default, the external clock rate is configured for 1500 MHz.

When the external clock is selected, two programmable clock dividers are available. Clock divider #1 can be any integer in the range [1, 32] and clock divider #2 can be any integer in the range [1, 6]. The clock dividers operate in series for an effective clock division of clock divider #1 x clock divider #2. The two clock dividers can combine for 108 unique effective clock divider settings ranging from 1 (no division) to 192.

The SetExtClockDividersSYNC6 function is used to select the external clock dividers. Both external clock dividers default to 1.

2.3 | Output Trigger Generation

The SYNC1500 can be configured to use one of two trigger sources: external or internal software. Whichever trigger source is selected will be used to drive all output trigger connections.

2.3.1 | Trigger Source – External

When the output trigger source is selected as external, an external trigger source (from end user's provided equipment) must be connected to the SYNC1500 External Trigger Input connector. This supplied external trigger source will then be broadcasted by the SYNC1500 to all of the connected devices in a synchronous way.

The SYNC1500 will detect a trigger on the positive-going edge of a TTL level digital pulse on the external trigger input connector. A detected trigger event will then result in a trigger pulse being generated on all SYNC1500 trigger outputs. The format of the output trigger pulse depends on the Output Trigger Configuration.

All output triggers have a common enable that must be turned on in order for the trigger to be output on the trigger outputs. The SetTriggerEnableSYNC6 function is used to set the output trigger enable.

2.3.2 | Trigger Source – Internal Software

When the output trigger source is selected as internal software, a generated software issued trigger event is used to result in a trigger pulse being generated on all SYNC1500 trigger outputs. The format of the output trigger pulse depends on the Output Trigger Configuration.

All output triggers have a common enable that must be turned on in order for the trigger to be output on the trigger outputs. The SetTriggerEnableSYNC6 function is used to set the output trigger enable.

The IssueSoftwareTriggerSYNC6 function is used to generate software issued trigger events.

2.4 | Output Trigger Configuration

Each output trigger on the SYNC1500 can be factory hardware configured to use one of two output trigger formats. These configurations are factory hardware-configured and may not be changed via software. In general, all triggers will have the same configuration, but this is not a hard requirement. The currently defined output trigger configurations are detailed in this table:

Configuration	Configuration Details	Signatec Devices Using Configuration
LVPECL	Output trigger pulse will be a LVPECL pulse	PX1500, PXDAC4800
LVTTL	Output trigger pulse will be a LVTTL pulse	PX14400

2.5 | External Device Synchronization

The SYNC1500 has the ability to synchronize externally connected devices. All synchronization outputs have a common enable that must be turned on in order for the synchronization pulse to be output.

The SYNC1500 software automatically manages synchronization of external devices by issuing synchronization requests when changes in the SYNC1500 hardware settings require it.

The SyncExternalDevicesSYNC6 function is used to synchronize all external devices.

2.6 | Cabled Connections to Digitizers/Arbitrary Waveform Generators

The SYNC1500 is typically supplied with 3-foot length (36 inches / 914.4 mm) 50Ω RG-316 MMCX straight male plug to SMA straight male cables in sufficient quantity to connect each digitizer/arbitrary waveform generator (AWG) unit ordered with the SYNC1500.

The MMCX plug ends of the supplied cables should be pushed straight into the MMCX connectors on the SYNC1500. To disconnect the MMCX cable from the SYNC1500, pull the MMCX plug straight out. The SMA male ends of the supplied cables should be aligned and tightened to the SMA connectors on the digitizer/AWG.

For synchronizing multiple PX14400/PX1500 digitizers or PXDAC4800 arbitrary waveform generators, the SYNC1500 must drive 2 of the SMA inputs of each connected digitizer/AWG device: the External Clock Input and the External Trigger Input.

To connect the 1st digitizer/AWG device unit:

- Connect the SYNC1500 Clock 1 Output (**CLK 1**) to the 1st digitizer/AWG External Clock Input.
- Connect the SYNC1500 Trigger 1 Output (**TRIG 1**) to the 1st digitizer/AWG External Trigger Input.

To connect the 2nd digitizer/AWG device unit:

- Connect the SYNC1500 Clock 2 Output (**CLK 2**) to the 2nd digitizer/AWG External Clock Input.
- Connect the SYNC1500 Trigger 2 Output (**TRIG 2**) to the 2nd digitizer/AWG External Trigger Input.

To connect the 3rd digitizer/AWG device unit:

- Connect the SYNC1500 Clock 3 Output (**CLK 3**) to the 3rd digitizer/AWG External Clock Input.
- Connect the SYNC1500 Trigger 3 Output (**TRIG 3**) to the 3rd digitizer/AWG External Trigger Input.

To connect the 4th digitizer/AWG device unit:

- Connect the SYNC1500 Clock 4 Output (**CLK 4**) to the 4th digitizer/AWG External Clock Input.
- Connect the SYNC1500 Trigger 4 Output (**TRIG 4**) to the 4th digitizer/AWG External Trigger Input.

To connect the 5th digitizer/AWG device unit:

- Connect the SYNC1500 Clock 5 Output (**CLK 5**) to the 5th digitizer/AWG External Clock Input.
- Connect the SYNC1500 Trigger 5 Output (**TRIG 5**) to the 5th digitizer/AWG External Trigger Input.

NOTE: It is **NOT** necessary to supply any connections to the SYNC1500 card's Synchronization Output (SYNC) channels, as these connections are **NOT** required for Revision 2+ based Signatec digitizer/AWG products.

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

2.7 | Cabled Connection to External Trigger Source

The SYNC1500 is typically supplied with a single 4-foot length (48 inches / 1219.2 mm) 50Ω RG-316 MMCX straight male plug to BNC straight male plug cable for use in connecting an external trigger source to the SYNC1500.

The MMCX plug end of the supplied cable should be pushed straight into the MMCX connector on the SYNC1500. To disconnect the MMCX cable from the SYNC1500, pull the MMCX plug straight out. The BNC straight plug end of the supplied cable should be aligned and tightened to the BNC connector of the end user's equipment that will be providing the external trigger source.

To connect the external trigger source:

- Connect the external trigger source output (from end user's provided equipment) to the SYNC1500 External Trigger Input (**TRIG IN**).

3 | SYNC1500 Software Development Reference

3.1 | Windows Software Installation

To install the SYNC1500 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. Windows Administrative rights are required for the software installation.

This installation will install all SYNC1500 software and documentation to the folder selected during installation that includes SYNC1500 drivers, libraries, documentation, components, applications, utilities, and programming examples. Some of the more important items are:

SYNC1500 dynamic link library: this library is the interface to the SYNC1500 driver. User applications will use the functions exported by this library to connect their software to the SYNC1500 device. The library (SYNC1500.dll) is installed to the product installation folder and the system library search path is updated to include this directory.

SYNC1500 Operators Manual: this is the document that you are currently reading and is installed to the Documentation\ folder of the install folder.

The SYNC1500 Control Application: this is a full featured Windows application that configures and operates SYNC1500 devices. The complete fully commented C source code project for this application is also provided and installed to the Examples\ folder of the install folder.

3.2 | Developing SYNC1500 Software

PC applications interface the SYNC1500 through the SYNC1500 library. This library exports a variety of C functions that you can use in your own programs that are document within this manual. The following sections describe how to set up the library and program the SYNC1500.

3.2.1 | Setting up the Build Environment – Windows Platform

The SYNC1500 library is composed of three parts: the header file (sync1500.h), the import library (SYNC1500.lib), and the dynamic link library (SYNC1500.dll). The header and import library are required to build SYNC1500 applications and the dynamic link library is required to run SYNC1500 applications.

The header file, sync1500.h, is the C interface to the library. Your SYNC1500 applications will "#include" this file to define SYNC1500 data structures, function prototypes, constants, and macros. This file is installed to the include\ folder of the SYNC1500 install directory. It is recommended that you add this path to your compiler's header file search path[1]. Adding the header file to the include file search path will allow you to keep a single copy of sync1500.h on your system and access it like a standard C header file. (That is, you can do a "#include <sync1500.h>" instead of a "#include "*Path-to-sync1500.h*".) The same header file is used for both 32- and 64-bit code.

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

The import library, SYNC1500.lib (32-bit code) or SYNC1500_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 SYNC1500 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" SYNC6PP_NO_LINK_LIBRARY before including sync1500.h.)

The dynamic link library, SYNC1500.dll (32-bit code) or SYNC1500_64.dll (64-bit code), is installed to the SYNC1500 software installation folder (C:\Program Files\Signatec\SYNC1500 by default). During software installation, the system library search path is updated to include this folder so the SYNC1500 library can be located when needed.

The 64-bit SYNC1500 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 SYNC1500 software and need to build for a 32-bit architecture.

3.2.2 | Library Functions That Use Character Strings

For library functions that accept or generate character strings (e.g. GetErrorTextSYNC6), 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 'ASYNC6' and the UNICODE version function names are suffixed with 'WSYNC6'.

Macros have been defined so that library users can write generic code that will work with either version. Consider GetErrorTextSYNC6 as an example. The library implements two versions of this function: GetErrorTextASYNC6 and GetErrorTextWSYNC6. If the _UNICODE symbol is defined, then the GetErrorTextSYNC6 macro will expand to GetErrorTextWSYNC6, else it will default to GetErrorTextASYNC6. In this manual library functions are documented using the character-size agnostic type TCHAR.

```
int GetErrorTextASYNC6 (int res, char** bufpp, unsigned int flags);
int GetErrorTextWSYNC6 (int res, wchar_t** bufpp, unsigned int flags);

#ifdef _UNICODE
# define GetErrorTextSYNC6      GetErrorTextWSYNC6
#else
# define GetErrorTextSYNC6      GetErrorTextASYNC6
#endif

// Effective library function prototype:
int GetErrorTextSYNC6 (int res, TCHAR** bufpp, unsigned int flags);
```

3.3 | Library Utility Functions

The functions in this section are generic library utility functions.

3.3.1 | DumpLibErrorSYNC6

Form

```
int DumpLibErrorSYNC6 (int res, const TCHAR* pPreamble, HSYNC6 hBrd = SYNC6_INVALID_HANDLE, FILE* filp = NULL);
```

Description

Dump library error description to given file or stream.

Parameters

[in] *res*

The library error value for which to display the error description. This can be any (generic) SIG_*, or (SYNC1500-specific) SIG_SYNC6_* error value.

[in] *pPreamble*

A pointer to a NULL-terminated string with a preamble that will be displayed before the error description. This parameter may be NULL if no preamble is to be displayed.

[in] *hBrd*

A handle to the SYNC1500 device for which the error occurred. For some types of errors, the SYNC1500 handle used when the error was generated may contain additional context information. This parameter may be SYNC6_INVALID_HANDLE.

[in] *filp*

A pointer to a FILE* for the output stream that will be used. If NULL is passed for this parameter then standard output will be used.

Return Value

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

Remarks

This is a utility function that can be used to dump a library error description to a file or stream, which will typically be standard output.

The function will use the GetErrorTextSYNC6 function to generate error text for the given error and output it into the given file/stream.

DumpLibErrorSYNC6 is a macro that will expand to _DumpLibErrorART2W or _DumpLibErrorWRT2W depending on the native character type. See Library Functions That Use Character Strings section for details.

Sample Usage

```
HSYNC6 hBrd;
int res;

// …
res = SetClockSourceSYNC6 (hBrd, 910901 /* Invalid value */);
if (SIG_SUCCESS != res)
        DumpLibErrorSYNC6 (res, _T("Failed to set source clock: "), hBrd);
// …
```

Sample output

```
Failed to set source clock: (Device:SYNC1500 #320033) Function argument #2 was invalid.
```

Related Functions

GetErrorTextSYNC6

3.3.2 | GetErrorTextSYNC6

Form

```
int GetErrorTextSYNC6 (int res, TCHAR** bufpp, unsigned int flags, HSYNC6 hBrd = SYNC6_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 (SYNC1500-specific) SIG_SYNC6_* 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 FreeMemorySYNC6 library function

[in] *flags*

A set of flags that dictate the function behavior.

Flag	Interpretation
SYNC6ETF_IGNORE_SYSERROR (0x00000001)	Ignore system specific error information (Windows: GetLastError, Linux: errno)
SYNC6ETF_NO_SYSERROR_TEXT (0x00000002)	Do not generate any text for system specific error
SYNC6ETF_FORCE_SYSERROR (0x00000004)	Force inclusion of system error information even if it might not be relevant

[in] *hBrd*

A handle to the SYNC1500 device for which the error occurred. For some types of errors, the SYNC1500 handle used when the error was generated may contain additional context information. This parameter may be SYNC6_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.

GetErrorTextSYNC6 is a macro that will expand to GetErrorTextASYNC6 or GetErrorTextWSYNC6 depending on the native character type. See Library Functions That Use Character Strings section for details.

Related Functions

DisconnectFromDeviceSYNC6, ConnectToVirtualDeviceSYNC6

3.3.3 | SetUserDataSYNC6 / GetUserDataSYNC6

Form

int SetUserDataSYNC6 (HSYNC6 hBrd, void *data);
int GetUserDataSYNC6 (HSYNC6 hBrd, void **datap);

Description

Set or get user-defined data value associated with SYNC6 handle.

Parameters

[in] *hBrd*

A handle to the SYNC1500 board. This handle is obtained by calling the ConnectToDeviceSYNC6 function.

[in] *data*

A user-defined data value that will be associated with the given device handle.

[out] *datap*

A pointer to a void* variable that will receive the current user-defined data value.

Return Value

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

Remarks

Each SYNC1500 device handle can have a user-defined data value associated with it. When the connection to the device is established, the user-defined data value is initialized to 0. Interpretation of this value is entirely user-defined.

The user-defined data value is copied when a handle is duplicated by calling DuplicateHandleSYNC6.

3.4 | Device Enumeration and Connection Management

The functions in this section involve managing SYNC1500 device connections and general SYNC1500 device handle management.

3.4.1 | ConnectToDeviceSYNC6

Form

```
int ConnectToDeviceSYNC6 (HSYNC6 * phDev, unsigned int brdNum = 1);
```

Description

This function is used to establish a connection to a local SYNC1500 device, represented by a SYNC1500 device handle of type HSYNC6.

Parameters

[in] *pBrd*

A pointer to a HSYNC6 variable that will receive the SYNC1500 device handle. This SYNC1500 device handle is only valid in the current process. This handle should be treated as an opaque object; interpretation of the handle value is library implementation specific.

[in] *brdNum*

This parameter is used to select which SYNC1500 in the system to connect to. If this value is in the range [1, SYNC6_MAX_DEVICES (32)] 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 this range then it is assumed to be the SYNC1500 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 SYNC1500 hardware device in any way. The device driver's hardware register cache is consulted for software register values. This allows a thread to attach to a currently running SYNC1500 device in a non-intrusive manner.

Like other handle-based mechanisms, the actual value of a SYNC1500 device handle should be treated as an opaque value. The exception to this is a special value, SYNC6_INVALID_HANDLE, which is used to identify an invalid SYNC1500 device handle.

An application should close the connection to the SYNC1500 device by calling DisconnectFromDeviceSYNC6. Failure to disconnect from the device can result in resource leaks in the calling process.

Related Functions

DisconnectFromDeviceSYNC6, ConnectToVirtualDeviceSYNC6

3.4.2 | ConnectToVirtualDeviceSYNC6

Form

```
int ConnectToVirtualDeviceSYNC6 (HSYNC6 * phBrd, unsigned int serialNum, unsigned int brdNum);
```

Description

Establish a connection to a virtual (fake) SYNC1500 device.

Parameters

[in] *pBrd*
A pointer to a HSYNC6 variable that will receive the virtual SYNC1500 device handle.

[in] *serialNum*
The serial number to use for the virtual device. This can be any number; the SYNC1500 library ignores this value. This will be the number obtained via the GetSerialNumberSYNC6 function.

[in] *brdNum*
The board number to use for the virtual device. This can be any number; the SYNC1500 library ignores this value. This will be the number obtained by the GetOrdinalNumberSYNC6 function.

Return Value

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

Remarks

This function is used to establish a connection to a virtual SYNC1500 data acquisition device. A virtual device is one that is not connected to any real SYNC1500 hardware. Virtual devices are mainly used for software development and debugging.

Related Functions

DisconnectFromDeviceSYNC6

3.4.3 | DuplicateHandleSYNC6

Form

```
int DuplicateHandleSYNC6 (HSYNC6 hBrd, HSYNC6* phNew);
```

Description

Duplicate an SYNC1500 device handle.

Parameters

[in] *hBrd*
The SYNC1500 device handle to duplicate. This handle must have been obtained from the current process.

[out] *phNew*
A pointer to a HSYNC6 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.

Remarks

This function will create a new device handle to the SYNC1500 device associated with the source handle. All handle-specific data (software register cache state, user-defined data, etc) will be copied over to the new device handle.

This function has no direct effect on the SYNC1500 hardware.

Related Functions

ConnectToDeviceSYNC6

3.4.4 | DisconnectFromDeviceSYNC6

Form

```
int DisconnectFromDeviceSYNC6 (HSYNC6 hBrd);
```

Description

This function is used to release the handle for the SYNC1500 when it is no longer needed in the program.

Parameters

[in] *hBrd*
The SYNC1500 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 SYNC6_INVALID_HANDLE.

Return Value

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

Remarks

In general, closing a SYNC1500 device handle will have no effect on the underlying hardware. This allows one to connect/disconnect to a SYNC1500 device in an unobtrusive manner.

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

Related Functions

ConnectToDeviceSYNC6

3.4.5 | GetDeviceCountSYNC6

Form

```
int GetDeviceCountSYNC6();
```

Description

Obtain a count of all SYNC1500 devices present in the local system.

Return Value

Returns the number of physical SYNC1500 devices present in the local system. If this function returns zero then either no SYNC1500 devices are present or the SYNC1500 driver has not been installed.

3.4.6 | IsDeviceVirtualSYNC6

Form

```
int IsDeviceVirtualSYNC6 (HSYNC6 hBrd);
```

Description

Determine if given handle is attached to a virtual SYNC1500 device.

Parameters

[in] *hBrd*

The SYNC1500 device handle to check. A virtual SYNC1500 device handle is obtained by calling ConnectToVirtualDeviceSYNC6.

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 SYNC1500 device. A virtual SYNC1500 is not connected to real SYNC1500 hardware and is mainly used for software development and debugging.

Related Functions

ConnectToVirtualDeviceSYNC6, IsHandleValidSYNC6

3.4.7 | IsHandleValidSYNC6

Form

```
int IsHandleValidSYNC6 (HSYNC6 hBrd);
```

Description

Determines if the given SYNC1500 device handle is a valid, connected device handle.

Parameters

[in] *hBrd*

The SYNC1500 device handle to check. A SYNC1500 device handle is obtained by calling ConnectToDeviceSYNC6 or ConnectToVirtualDeviceSYNC6.

Return Value

Returns non-zero if given handle is a valid SYNC1500 device handle, or zero otherwise.

Remarks

A handle is valid if it is connected to a local or virtual SYNC1500 device.

Related Functions

ConnectToDeviceSYNC6, ConnectToVirtualDeviceSYNC6

3.5 | Device State and Configuration

The functions in this section are used to obtain state and configuration information on the SYNC1500.

3.5.1 | GetClockRateSYNC6

Form

```
int GetClockRateSYNC6 (HSYNC6 hBrd, double* pRateMHz);
```

Description

Obtain current clock rate considering clock source and dividers.

Parameters

[in] *hBrd*
A handle to the SYNC1500 board. This handle is obtained by calling the ConnectToDeviceSYNC6 function.

[out] *pRateMHz*
A pointer to the variable that will receive the effective clock rate.

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 obtain the current effective clock rate for all enabled clock outputs. This function considers the current clock source and any clock dividers (if applicable for the clock source).

Related Functions

SetClockSourceSYNC6 / GetClockSourceSYNC6, SetExtClockDividerXSYNC6 / GetExtClockDividerXSYNC6, SetExternalClockRateSYNC6 / GetExternalClockRateSYNC6

3.5.2 | GetDriverVersionSYNC6

Form

```
int GetDriverVersionSYNC6 (HSYNC6 hBrd, unsigned long long* verp);
```

Description

Obtains the version of the SYNC1500 kernel-mode driver.

Parameters

[in] *hBrd*
A handle to the SYNC1500 board. This handle is obtained by calling the ConnectToDeviceSYNC6 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 SYNC1500 driver 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

GetLibVersionSYNC6, GetSoftwareReleaseVersionSYNC6

3.5.3 | GetFirmwareVersionSYNC6

Form

```
int GetFirmwareVersionSYNC6 (HSYNC6 hBrd, unsigned long long* verp);
```

Description

Obtains the version of the SYNC1500 firmware.

Parameters

[in] *hBrd*
A handle to the SYNC1500 board. This handle is obtained by calling the ConnectToDeviceSYNC6 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 SYNC1500 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

GetHardwareRevisionSYNC6

3.5.4 | GetHardwareRevisionSYNC6

Form

```
int GetHardwareRevisionSYNC6 (HSYNC6 hBrd, unsigned long long* verp);
```

Description

Obtains the revision of the SYNC1500 hardware.

Parameters

[in] *hBrd*
A handle to the SYNC1500 board. This handle is obtained by calling the ConnectToDeviceSYNC6 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 SYNC1500 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

GetFirmwareVersionSYNC6

3.5.5 | GetLibVersionSYNC6

Form

```
int GetLibVersionSYNC6 (HSYNC6 hBrd, unsigned long long* verp);
```

Description

Obtains the version of the SYNC1500 user-mode library.

Parameters

[in] *hBrd*
A handle to the SYNC1500 board. This handle is obtained by calling the ConnectToDeviceSYNC6 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 SYNC1500 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

GetDriverVersionSYNC6, GetSoftwareReleaseVersionSYNC6

3.5.6 | GetOrdinalNumberSYNC6

Form

```
int GetOrdinalNumberSYNC6 (HSYNC6 hBrd, unsigned int* onp);
```

Description

Obtain the ordinal number of the SYNC1500 connected to the given handle.

Parameters

[in] *hBrd*
A handle to the SYNC1500 board. This handle is obtained by calling the ConnectToDeviceSYNC6 function.

[out] *onp*
A pointer to the variable that will receive the SYNC1500'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 SYNC1500's ordinal number is the number of the SYNC1500 in the system, as in the first, second, third, etc board. This order is determined by the system and may or may not be the same order as the physical PCIe slots.

Related Functions

GetSerialNumberSYNC6

3.5.7 | GetSerialNumberSYNC6

Form

```
int GetSerialNumberSYNC6 (HSYNC6 hBrd, unsigned int* snp);
```

Description

Obtain the SYNC1500 board's serial number.

Parameters

[in] *hBrd*
A handle to the SYNC1500 board. This handle is obtained by calling the ConnectToDeviceSYNC6 function.

[out] *snp*
A pointer to the variable that will receive the SYNC1500's serial number.

Return Value

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

Remarks

SYNC1500 serial numbers are unique over all SYNC1500 devices.

Related Functions

GetOrdinalNumberSYNC6

3.5.8 | GetSoftwareReleaseVersionSYNC6

Form

```
int GetSoftwareReleaseVersionSYNC6 (HSYNC6 hBrd, unsigned long long* verp);
```

Description

Obtains the version of the SYNC1500 software release.

Parameters

[in] *hBrd*
A handle to the SYNC1500 board. This handle is obtained by calling the ConnectToDeviceSYNC6 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 software release version is the version associated with the overall software package that installs the various SYNC1500 software components (driver, library, etc).

The SYNC1500 software release 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

GetLibVersionSYNC6, GetDriverVersionSYNC6

3.5.9 | ReadConfigEepromSYNC6

Form

```
int ReadConfigEepromSYNC6 (HSYNC6 hBrd, unsigned int eeprom_addr, unsigned short* eeprom_datap);
```

Description

Read an element from the SYNC1500 configuration EEPROM.

Parameters

[in] *hBrd*
A handle to the SYNC1500 board. This handle is obtained by calling the ConnectToDeviceSYNC6 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 SYNC1500 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

WriteConfigEepromSYNC6

3.5.10 | WriteConfigEepromSYNC6

Form

```
int WriteConfigEepromSYNC6 (HSYNC6 hBrd, unsigned int eeprom_addr, unsigned short eeprom_data);
```

Description

Write an element from the SYNC1500 configuration EEPROM.

Parameters

[in] *hBrd*
A handle to the SYNC1500 board. This handle is obtained by calling the ConnectToDeviceSYNC6 function.

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

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

Return Value

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

Related Functions

ReadConfigEepromSYNC6

3.6 | SYNC1500 Hardware Settings

The functions in this section control the getting and setting of the various SYNC1500 hardware settings. These settings include the numerous clock, trigger, and synchronization parameters.

3.6.1 | SetClockOutputEnableMaskSYNC6 / GetClockOutputEnableMaskSYNC6

Form

int SetClockOutputEnableMaskSYNC6 (HSYNC6 hBrd, unsigned int mask);
int GetClockOutputEnableMaskSYNC6 (HSYNC6 hBrd, int bFromCache = 1);

Description

Enable/disable specific clocks outputs.

Parameters

[in] *hBrd*
A handle to the SYNC1500 device to which the copied hardware settings will be applied.

[in] *bFromCache*
If non-zero, the setting will be read from the local device register cache associated with the given device 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 hardware device register read.

[in] *mask*
A bitmask containing the clock outputs to enable. A set bit will enable the respective clock output and a cleared bit will disable the respective clock output:

Bit	Interpretation
0x00000001 (1)	Output clock #1
0x00000002 (2)	Output clock #2
0x00000004 (4)	Output clock #3
0x00000008 (8)	Output clock #4
0x00000010 (16)	Output clock #5
0x00000020 (32)	Output clock #6

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 set the state of all clock output enables in one atomic operation. The related SetClockOutputEnableSYNC6 function is used to set a particular clock output enable.

Related Functions

SetClockOutputEnableSYNC6

3.6.2 | SetClockOutputEnableSYNC6 / GetClockOutputEnableSYNC6

Form

int SetClockOutputEnableSYNC6 (HSYNC6 hBrd, int nClock, int bEnable);
int GetClockOutputEnableSYNC6 (HSYNC6 hBrd, int nClock, int bFromCache = 1);

Description

Enable/disable a single specific output clock.

Parameters

[in] *hBrd*
A handle to the SYNC1500 device to which the copied hardware settings will be applied.

[in] *bFromCache*
If non-zero, the setting will be read from the local device register cache associated with the given device 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 hardware device register read.

[in] *nClock*
The output clock enable to set. This can be any integer value in the range [1,6].

[in] *bEnable*
If non-zero, the specified output clock will be enabled. If zero, the specified output clock will be disabled.

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 enable/disable a specific output clock. The SetClockOutputEnableMaskSYNC6 function may be used to set the state of all output clocks in one atomic operation.

Related Functions

SetClockOutputEnableMaskSYNC6

3.6.3 | SetClockRefSourceSYNC6 / GetClockRefSourceSYNC6

Form

int SetClockRefSourceSYNC6 (HSYNC6 hBrd, unsigned int val);
int GetClockRefSourceSYNC6 (HSYNC6 hBrd, int bFromCache = 1);

Description

Select the source of the 10MHz reference used for the internal clock.

Parameters

[in] *hBrd*

A handle to the SYNC1500 device to which the copied hardware settings will be applied.

[in] *bFromCache*

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

[in] *val*

The internal clock 10MHz reference clock source to select. This can be any of the following values:

Library Constant	Interpretation
SYNC6CLKREF_INT_10MHZ (0)	Internal 10MHz clock reference (Power-up default)
SYNC6CLKREF_EXT (1)	Externally provided 10MHz clock reference

Return Value

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

On success, GetClockRefSourceSYNC6 returns the current 10MHz reference source (SYNC6CLKREF_*).

Remarks

The 10MHz reference clock is only relevant when the internal clock is selected as the source clock.

Related Functions

SetClockSourceSYNC6

3.6.4 | SetClockSourceSYNC6 / GetClockSourceSYNC6

Form

int SetClockSourceSYNC6 (HSYNC6 hBrd, unsigned int val);
int GetClockSourceSYNC6 (HSYNC6 hBrd, int bFromCache = 1);

Description

Get or set the clock source used to drive the output clocks.

Parameters

[in] *hBrd*

A handle to the SYNC1500 device to which the copied hardware settings will be applied.

[in] *bFromCache*

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

[in] *val*

The source clock to use as the output clock. This can be any of the following values:

Library Constant	Interpretation
SYNC6CLKSRC_INTERNAL (0)	Internal VCO (Power-up default)
SYNC6CLKSRC_EXTERNAL (1)	External clock

Return Value

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

On success, GetClockSourceSYNC6 will return the current source clock selection (SYNC6CLKSRC_*).

Related Functions

SetInternalClockRateSYNC6, SetExternalClockRateSYNC6, SetExtClockDividersSYNC6

3.6.5 | SetExtClockDivider*X*SYNC6 / GetExtClockDivider*X*SYNC6

Form

int SetExtClockDividersSYNC6 (HSYNC6 hBrd, unsigned int div1, unsigned int div2);
int SetExtClockDivider1SYNC6 (HSYNC6 hBrd, unsigned int div1);
int GetExtClockDivider1SYNC6 (HSYNC6 hBrd, int bFromCache = 1);
int SetExtClockDivider2SYNC6 (HSYNC6 hBrd, unsigned int div2);
int GetExtClockDivider2SYNC6 (HSYNC6 hBrd, int bFromCache = 1);

Description

Set or get the external clock dividers applied to the external clock.

Parameters

[in] *hBrd*
A handle to the SYNC1500 device to which the copied hardware settings will be applied.

[in] *bFromCache*
If non-zero, the setting will be read from the local device register cache associated with the given device 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 hardware device register read.

[in] *div1*
The clock divider value to use for clock divider #1. This can be any integer value in the range [1, 32].

[in] *div2*
The clock divider value to use for clock divider #2. This can be any integer value in the range [1, 6].

Return Value

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

On success, GetExtClockDivider1SYNC6 will return the current clock divider #1 setting.
On success, GetExtClockDivider2SYNC6 will return the current clock divider #2 setting.

Remarks

The clock dividers set with these functions are only relevant when the external clock is selected as the source clock. The library automatically configures the SYNCH1500 hardware to use the external clock dividers when the external clock is selected as the clock source.

Clock divider #1 and clock divider #2 operate in series so the effective clock division is *div1* * *div2*.

Related Functions

SetExternalClockRateSYNC6, SetClockSourceSYNC6

3.6.6 | SetExternalClockRateSYNC6 / GetExternalClockRateSYNC6

Form

int SetExternalClockRateSYNC6 (HSYNC6 hBrd, double dRateMHz);
int GetExternalClockRateSYNC6 (HSYNC6 hBrd, double* ratep, int bFromCache = 1);

Description

Set or get the external clock rate.

Parameters

[in] *hBrd*

A handle to the SYNC1500 device to which the copied hardware settings will be applied.

[in] *bFromCache*

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

[in] *dRateMHz*

The frequency of the applied external clock rate in MHz.

[out] *ratep*

A pointer to a double variable that will receive the current external 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 setting is only relevant when using the external clock source.

If an external clock is being used to drive the SYNC1500 clock outputs, the library needs to know what the frequency of the external clock is so that it can properly configure the hardware for that rate.

Related Functions

SetClockSourceSYNC6

3.6.7 | SetInternalClockRateSYNC6 / GetInternalClockRateSYNC6

Form

int SetInternalClockRateSYNC6 (HSYNC6 hBrd, double dRateMHz);
int GetInternalClockRateSYNC6 (HSYNC6 hBrd, double* ratep, int bFromCache = 1);

Description

Set or get the internal clock rate.

Parameters

[in] *hBrd*

A handle to the SYNC1500 device to which the copied hardware settings will be applied.

[in] *bFromCache*

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

[in] *dRateMHz*

The frequency to use for the internal clock in MHz. This can be any value in the range [25, 1500] and will be internally realigned down to a multiple of 20kHz.

[out] *ratep*

A pointer to a double variable that will receive the current internal 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 setting is only relevant when using the internal clock source.

The internal VCO 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

SetClockSourceSYNC6

3.6.8 | SetSyncPulseEnableSYNC6 / GetSyncPulseEnableSYNC6

Form

int SetSyncPulseEnableSYNC6 (HSYNC6 hBrd, unsigned int bEnable);
int GetSyncPulseEnableSYNC6 (HSYNC6 hBrd, int bFromCache = 1);

Description

Enable or disable synchronization pulse output.

Parameters

[in] *hBrd*

A handle to the SYNC1500 device to which the copied hardware settings will be applied.

[in] *bFromCache*

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

[in] *bEnable*

If non-zero, the synchronization pulse will be enabled. If zero, no synchronization pulse will be sent to connected devices.

Return Value

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

On success, GetSyncPulseEnableSYNC6 will return the current synchronization pulse enable.

Remarks

This setting controls the enable of the synchronization pulse only. A synchronization pulse is generated by calling SyncExternalDevicesSYNC6. If the synchronization pulse is disabled then issuing a synchronization pulse will have no effect.

Related Functions

SyncExternalDevicesSYNC6

3.6.9 | SetTriggerEnableSYNC6 / GetTriggerEnableSYNC6

Form

int SetTriggerEnableSYNC6 (HSYNC6 hBrd, unsigned int bEnable);
int GetTriggerEnableSYNC6 (HSYNC6 hBrd, int bFromCache = 1);

Description

Enable or disable trigger output.

Parameters

[in] *hBrd*

A handle to the SYNC1500 device to which the copied hardware settings will be applied.

[in] *bFromCache*

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

[in] *bEnable*

If non-zero, the trigger output will be enabled. If zero, trigger output 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, GetTriggerEnableSYNC6 will return the current trigger enable.

Remarks

This setting controls the enable of the trigger pulse only. The generation of a trigger pulse depends on the trigger source setting.

If the trigger output is disabled then any external or software-triggered trigger events will not be delivered to connected devices.

3.7 | Device Synchronization and Triggering Functions

The functions in this section are used to trigger and synchronize the devices connected to the SYNC1500.

3.7.1 | IssueSoftwareTriggerSYNC

Form

```
int IssueSoftwareTriggerSYNC (HSYNC6 hBrd);
```

Description

Issue a software-generated trigger event to a SYNC1500.

Parameters

[in] *hBrd*
A handle to the SYNC1500 board. This handle is obtained by calling the ConnectToDeviceSYNC6 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 IssueSoftwareTriggerSYNC allows the user to issue a software trigger to the SYNC1500 board. After a software trigger has been issued, the SYNC1500 will immediately send out a trigger pulse to all enabled trigger outputs.

Related Functions

SetTriggerEnableSYNC6 / GetTriggerEnableSYNC6

3.7.2 | SyncExternalDevicesSYNC6

Form

```
int SyncExternalDevicesSYNC6 (HSYNC6 hBrd);
```

Description

Synchronize all external devices.

Parameters

[in] *hBrd*
A handle to the SYNC1500 device to which the copied hardware settings will be applied.

Return Value

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

Remarks

This function will only have effect if the synchronization pulse is enabled.

Calling this function will result in a synchronization pulse sent to all external devices connected to the SYNC1500 device.

Explicitly calling this function is not necessary; the underlying SYNC1500 library will automatically send synchronization pulses as they are needed when clock settings are changed.

Related Functions

SetSyncPulseEnableSYNC6

3.8 | Device Register State Functions

The functions in this section involve manipulation of SYNC1500 device registers.

3.8.1 | CopyHardwareSettingsSYNC6

Form

```
int CopyHardwareSettingsSYNC6 (HSYNC6 hBrdDst, HSYNC6 hBrdSrc);
```

Description

Copy hardware settings from another SYNC1500 device.

Parameters

[in] *hBrdDst*
A handle to the SYNC1500 device to which the copied hardware settings will be applied.

[in] *hBrdSrc*
A handle to the SYNC1500 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 SYNC1500 associated with the source device to the SYNC1500 associated with the destination device. The function will use the hardware settings as they are defined in the handle-specific software register cache.

Related Functions

RewriteHardwareSettingsSYNC6, RefreshLocalRegisterCacheSYNC6

3.8.2 | LoadSettingsFromFileXmlSYNC6

Form

```
int LoadSettingsFromFileXmlSYNC6 (HSYNC6 hBrd, unsigned int flags, const TCHAR* bufp);
```

Description

Load hardware settings from an XML file.

Parameters

[in] *hBrd*
A handle to the SYNC1500 board. This handle is obtained by calling the ConnectToDeviceSYNC6 function.

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

Library Constant	Value	Interpretation
SYNC6XMLSET_NO_PRELOAD_DEFAULTS	0x00000004	Do not set default hardware settings prior to loading settings

[in] *bufp*
A pointer to a NULL-terminated string containing the pathname of the XML file containing the settings data. The SaveSettingsToFileXmlSYNC6 function is used to generate this data.

Return Value

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

Remarks

This function is used to apply hardware settings saved by a previous call to SaveSettingsToFileXmlSYNC6.

LoadSettingsFromFileXmlSYNC6 is a macro that will expand to LoadSettingsFromFileXmlASYNC6 or LoadSettingsFromFileXmlWSYNC6 depending on the native character type. See Library Functions That Use Character Strings section for details.

Related Functions

SaveSettingsToFileXmlSYNC6

3.8.3 | LoadSettingsFromStringXmlSYNC6

Form

```
int LoadSettingsFromStringXmlSYNC6 (HSYNC6 hBrd, unsigned int flags, const TCHAR* bufp);
```

Description

Load hardware settings from an XML buffer.

Parameters

[in] *hBrd*
A handle to the SYNC1500 board. This handle is obtained by calling the ConnectToDeviceSYNC6 function.

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

Library Constant	Value	Interpretation
SYNC6XMLSET_NO_PRELOAD_DEFAULTS	0x00000004	Do not set default hardware settings prior to loading settings

[in] *bufp*
A pointer to a NULL-terminated string containing the XML data containing the settings data. The SaveSettingsToStringXmlSYNC6 function is used to generate this data.

Return Value

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

Remarks

This function is used to apply hardware settings saved by a previous call to SaveSettingsToStringXmlSYNC6.

LoadSettingsFromStringXmlSYNC6 is a macro that will expand to LoadSettingsFromStringXmlASYNC6 or LoadSettingsFromStringXmlWSYNC6 depending on the native character type. See Library Functions That Use Character Strings section for details.

Related Functions

SaveSettingsToStringXmlSYNC6

3.8.4 | RefreshLocalRegisterCacheSYNC6

Form

```
int RefreshLocalRegisterCacheSYNC6 (HSYNC6 hBrd, int bFromHardware = 0);
```

Description

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

Parameters

[in] *hBrd*

A handle to the SYNC1500 board. This handle is obtained by calling the ConnectToDeviceSYNC6 function.

[in] *bFromHardware*

If this parameter is non-zero then register content will be updated from the SYNC1500 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 SYNC1500 device register cache associated with the given SYNC1500 device handle. The register values are obtained from the kernel-level register cache maintained by the SYNC1500 device driver. Calling this function has no effect on any underlying SYNC1500 hardware.

This function is automatically invoked by the ConnectToDeviceSYNC6 function as part of the device connection procedure.

Related Functions

RewriteHardwareSettingsSYNC6

3.8.5 | RewriteHardwareSettingsSYNC6

Form

```
int RewriteHardwareSettingsSYNC6 (HSYNC6 hBrd);
```

Description

Bring hardware settings up to date with current cache settings.

Parameters

[in] *hBrd*
A handle to the SYNC1500 board. This handle is obtained by calling the ConnectToDeviceSYNC6 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 SYNC1500 handle.

Related Functions

RefreshLocalRegisterCacheSYNC6

3.8.6 | SaveSettingsToFileXmlSYNC6

Form

```
int SaveSettingsToFileXmlSYNC6 (HSYNC6 hBrd, unsigned int flags, TCHAR* pathnamep, TCHAR* encodingp = "UTF-8");
```

Description

Save board settings to an XML file.

Parameters

[in] *hBrd*
A handle to the SYNC1500 board. This handle is obtained by calling the ConnectToDeviceSYNC6 function.

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

Library Constant	Value	Interpretation
SYNC6XMLSET_NODE_ONLY	0x00000001	Serialize to a node only; do not include XML header info
SYNC6XMLSET_FORMAT_OUTPUT	0x00000002	Pretty-print XML output; add newlines and indentation. This will generate nicer, human-readable output.

[out] *pathnamep*
A pointer to a NULL-terminated string containing the pathname of the XML file.

[out] *encodingp*
A pointer to a NULL-terminated string containing the encoding to use for the generated XML file. This can be any encoding supported by the iconv library: ASCII, UTF-8, UTF16, char, wchar_t, etc.

Return Value

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

Remarks

This function is used to save hardware settings to an XML file. These settings can later be reloaded by calling the LoadSettingsFromFileXmlSYNC6 function.

SaveSettingsToFileXmlSYNC6 is a macro that will expand to SaveSettingsToFileXmlASYNC6 or SaveSettingsToFileXmlWSYNC6 depending on the native character type. See Library Functions That Use Character Strings section for details.

Related Functions

LoadSettingsFromFileXmlSYNC6

3.8.7 | SaveSettingsToStringXmlSYNC6

Form

int SaveSettingsToStringXmlSYNC6 (HSYNC6 hBrd, unsigned int flags, TCHAR** bufpp);

Description

Save board settings to XML format in a library allocated-buffer.

Parameters

[in] *hBrd*

A handle to the SYNC1500 board. This handle is obtained by calling the ConnectToDeviceSYNC6 function.

[in] *flags*

Flags that affect how settings are loaded:

Library Constant	Value	Interpretation
SYNC6XMLSET_NODE_ONLY	0x00000001	Serialize to a node only; do not include XML header info
SYNC6XMLSET_FORMAT_OUTPUT	0x00000002	Pretty-print XML output; add newlines and indentation. This will generate nicer, human-readable output.

[out] *bufp*

A pointer to a TCHAR* variable that will receive the address of a library-allocated buffer containing the XML data. It is the caller's responsibility to free this memory with FreeMemorySYNC6 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 save hardware settings to an XML buffer. These settings can later be reloaded by calling the LoadSettingsFromStringXmlSYNC6 function.

SaveSettingsToStringXmlSYNC6 is a macro that will expand to SaveSettingsToStringXmlASYNC6 or SaveSettingsToStringXmlWSYNC6 depending on the native character type. See Library Functions That Use Character Strings section for details.

Related Functions

LoadSettingsFromStringXmlSYNC6, SaveSettingsToFileXmlSYNC6

3.8.8 | SetPowerupDefaultsSYNC6

Form

```
int SetPowerupDefaultsSYNC6 (HSYNC6 hBrd);
```

Description

Restores all SYNC1500 settings to power-up default values.

Parameters

[in] *hBrd*
A handle to the SYNC1500 board. This handle is obtained by calling the ConnectToDeviceSYNC6 function.

Return Value

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

Remarks

Calling this function will reset all hardware settings to their default power up values:

- Clock source set to internal.
- Source for internal clock's 10MHz reference set to internal.
- Internal and external clock rates set to 1500 MHz.
- External clock dividers both set to 1.
- Trigger, synchronization pulse, and all output clocks enabled.

3.9 | Memory Management Functions

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

3.9.1 | FreeMemorySYNC6

Form

```
int FreeMemorySYNC6 (void* p);
```

Description

Free memory allocated by the SYNC1500 library.

Parameters

[in] *p*
The address of the SYNC1500 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 SYNC1500 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 SYNC1500 library may have allocated from another heap and freeing could result in memory corruption or an application crash.

3.10 | SYNC1500 Library Data Types

The following sections define some of the data types used by the SYNC1500 library.

3.10.1 | Data Type: HSYNC6

This data type is used to represent a handle to a SYNC1500 device.

A special library constant SYNC6_INVALID_HANDLE (NULL) represents an invalid SYNC1500 device handle. Any other value should be considered opaque; this value only has relevance to the SYNC1500 library implementation.

SYNC1500 handles are only valid in the process for which they are obtained.

The DisconnectFromDeviceSYNC6 library function should be called to release a SYNC1500 device handle when it is no longer needed. This will free handle-specific resources allocated by the library.

4 | Appendix A – SYNC1500 Specifications

Input Signal Connections

Trigger	: MMCX (miniature RF)
Clock	: MMCX

Output Signal Connections

Clocks (5)	: MMCX
Triggers (5)	: MMCX
Sync Pulses (5)	: MMCX

Trigger Input

Signal Type	: Digital, TTL Level
Impedance	: 200 ohms
Active Edge	: Positive

Clock Input

Signal Type	: Analog or Digital
Coupling	: AC
Impedance	: 50 ohms
Frequency	: 10 MHz to 1500 MHz
Amplitude	: 800 mV p-p (-200/+700)

Internal Synthesized Clock

Frequency range	: 25.0 - 1500 MHz
Resolution	: better than 62.5 PPM
Accuracy	: better than 5 PPM

Reference Clock

Internal	: 10.0 MHz, ± 5 PPM
External	: 10.0 MHz, ± 50 PPM (required for lock)

Output Clocks

Coupling	: AC
Required Termination	: 50 ohms
Amplitude	: 800 mV p-p (typical)

Output Triggers

Signal Type	: LVPECL (+3.3V); for use with PX1500/PXDAC4800 LVTTL (+3.3V); for use with PX14400
Required Termination	: 50 ohms to +1.3V (LVPECL only)

Output Sync Pulses

Signal Type	: LVPECL (+3.3V)
Required Termination	: 50 ohms to +1.3V

Absolute Maximum Ratings

Trigger Input	: -0.2V to +3.5V
Clock Input	: 3V p-p
Ambient Temperature	: 0 to 50°C

Part Numbers

SYNC1500-P : LVPECL Output Triggers (for use with PX1500/PXDAC4800)

SYNC1500-T : LVTTL Output Triggers (for use with PX14400)

Cables

The SYNC1500 is supplied with the following cables:

One 4-foot length (48 inches / 1219.2 mm) 50Ω RG-316 MMCX straight male plug to BNC straight male plug cable for use in connecting an external trigger source to the SYNC1500.

Multiple 3-foot length (36 inches / 914.4 mm) 50Ω RG-316 MMCX straight male plug to SMA straight male cables in sufficient quantity to connect each digitizer/arbitrary waveform generator unit ordered with the SYNC1500.

Documentation & Accessories

The SYNC1500 is supplied with a comprehensive operator's manual, which thoroughly describes the operation of both the hardware and the software. 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:

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

Specification Sheet Revision 2.0 – 07/27/2015
Specifications are subject to change without notice.

5 | Appendix B – SYNC1500 Library Error Codes

The table below lists all of the currently defined SYNC1500 library error codes.

The GetErrorTextSYNC6 library function can be used to generate a user-friendly string for any of these error codes.

Symbolic Constant	Error Code	Interpretation
SIG_SUCCESS	0	Operation successful
SIG_SYNC6_QUASI_SUCCESSFUL	512	Operation was quasi-successful; one or more items failed and one or more items succeeded.
SIG_ERROR	-1	Generic error; platform's system error may provide more info
SIG_INVALIDARG	-2	An invalid argument was specified
SIG_OUTOFBOUNDS	-3	An argument is out of valid bounds
SIG_NODEV	-4	Invalid board device
SIG_OUTOFMEMORY	-5	Error allocating memory
SIG_DMABUFALLOCFAIL	-6	Error allocating a DMA buffer
SIG_NOSUCHBOARD	-7	Board with given serial or ordinal number not found
SIG_NT_ONLY	-8	This feature is only available on Windows NT platforms.
SIG_INVALID_MODE	-9	Invalid operation for current operating mode
SIG_CANCELLED	-10	Operation was cancelled by user
SIG_SYNC6__FIRST	-512	First SYNC1500-specific error code value
SIG_SYNC6_NOT_IMPLEMENTED	-512	This operation is not currently implemented
SIG_SYNC6_INVALID_HANDLE	-513	An invalid SYNC1500 device handle (HSYNC6) was specified
SIG_SYNC6_INVALID_ARG_1	-514	Argument 1 is invalid
SIG_SYNC6_INVALID_ARG_2	-515	Argument 2 is invalid
SIG_SYNC6_INVALID_ARG_3	-516	Argument 3 is invalid
SIG_SYNC6_INVALID_ARG_4	-517	Argument 4 is invalid
SIG_SYNC6_INVALID_ARG_5	-518	Argument 5 is invalid
SIG_SYNC6_INVALID_ARG_6	-519	Argument 6 is invalid
SIG_SYNC6_INVALID_ARG_7	-520	Argument 7 is invalid
SIG_SYNC6_INVALID_ARG_8	-521	Argument 8 is invalid
SIG_SYNC6_BUSY	-522	Device is busy; try again later
SIG_SYNC6_XML_MALFORMED	-523	Invalid XML data was encountered
SIG_SYNC6_XML_INVALID	-524	XML data was well formed, but not valid
SIG_SYNC6_XML_GENERIC	-525	Generic XML releated error
SIG_SYNC6_RATE_TOO_FAST	-526	The specified rate is too fast
SIG_SYNC6_RATE_TOO_SLOW	-527	The specified rate is too slow
SIG_SYNC6_RATE_NOT_AVAILABLE	-528	The specified frequency is not available; see operator's manual
SIG_SYNC6_UNEXPECTED	-529	An unexpected error occurred; debug builds will have failed assertion
SIG_SYNC6_TIMED_OUT	-535	Operation timed out

Continued:

Symbolic Constant	Error Code	Interpretation
SIG_SYNC6_UNKNOWN_FW_FILE	-536	Unknown firmware file type
SIG_SYNC6_FIRMWARE_UPLOAD_FAILED	-537	Firmware upload failed
SIG_SYNC6_INVALID_FW_FILE	-538	Invalid firmware upload file
SIG_SYNC6_DEST_FILE_OPEN_FAILED	-539	Failed to open destination file
SIG_SYNC6_SOURCE_FILE_OPEN_FAILED	-540	Failed to open source file
SIG_SYNC6_FILE_IO_ERROR	-541	File IO error
SIG_SYNC6_INCOMPATIBLE_FIRMWARE	-542	Firmware is incompatible with SYNC1500
SIG_SYNC6_UNKNOWN_STRUCT_SIZE	-543	Unknown structure version specified to library function (X::struct_size)
SIG_SYNC6_INVALID_REGISTER	-544	An invalid hardware register read/write was attempted
SIG_SYNC6_DCM_SYNC_FAILED	-545	SYNC1500 firmware could not synchronize to acquisition clock
SIG_SYNC6_DISK_FULL	-546	Could not write all data; disk is full
SIG_SYNC6_INVALID_OBJECT_HANDLE	-547	An invalid object handle was used
SIG_SYNC6_PLL_LOCK_FAILED	-548	Phase lock loop (PLL) failed to lock; clock may be bad
SIG_SYNC6_NAMED_ITEM_NOT_FOUND	-549	Named item could not be found
SIG_SYNC6_NOT_IMPLEMENTED_IN_FIRMWARE	-550	Feature is not implemented in current firmware version; upgrade firmware
SIG_SYNC6_CANNOT_DETERMINE_FW_REQ	-551	Cannot determine which firmware needs to be uploaded; update software
SIG_SYNC6_REQUIRED_FW_NOT_FOUND	-552	Required firmware not found in firmware update file
SIG_SYNC6_FIRMWARE_IS_UP_TO_DATE	-553	Loaded firmware is up to date with firmware update file
SIG_SYNC6_NO_VIRTUAL_IMPLEMENTATION	-554	Operation not implemented for virtual devices
SIG_SYNC6_DEVICE_REMOVED	-555	SYNC1500 device has been removed from system
SIG_SYNC6_FLASH_WRITE_FAILURE	-556	Failed to write to SYNC1500 flash RAM
SIG_SYNC6_ACCESS_DENIED	-557	Access denied
SIG_SYNC6_PLL_LOCK_FAILED_UNSTABLE	-558	Phase lock loop (PLL) failed to lock; lock not stable
SIG_SYNC6_INVALIDARG_NULL_POINTER	-559	An invalid pointer was specified

6 | Appendix C – Revision History

Revision 1.0 (Initial public release)

Revision 1.1

- Updated manual cover page for DynamicSignals contact information.

Revision 2.0

- Updated Package Contents to identify all supplied items.
- Updated Warranty section for new 2 year warranty period.
- Updated System Requirements section for updated system specifications and supported OS versions.
- Added new Physical Layout section for connector identification.
- Updated Functional Description Overview section and sub-sections to reflect new updated maximum of up to 5 maximum supported connected devices (as internal onboard 6th device connection is no longer supported with current SYNC1500 hardware revision and modifications).
- Added new Trigger Source – Internal Software section and related library software function, as ability to generate a software based external trigger event for SYNC1500 has been added.
- Updated Output Trigger Configuration section to reflect updated hardware factory configured options.
- Added new Cabled Connections to Digitizers/Arbitrary Waveform Generators and new Cabled Connection to External Trigger Source sections, for general cable connection overview.
- Updated Functional Description Overview section and sub-sections and Software Development Reference section and sub-sections to remove all references to “remote” based operation of SYNC1500 devices, as remote operations with previous Signatec Service Manager software is no longer supported.
- Updated Appendix A and Appendix B for current SYNC1500 specifications.