SD1 3.x Software for M310xA / M330xA Digitizers

User's Guide

KEYSIGHT TECHNOLOGIES

Notices

Copyright Notice

Trademarks

UNIX is a registered trademark of UNIX System Laboratories in the U.S.A. and other countries. Target is copyrighted by Thru-Put Systems, Inc.

Manual Part Number

M3xxx-90004

Edition

1.01, September 2020

Available in electronic format only

Published by

Keysight Technologies, Inc.
1900 Garden of the Gods Road
Colorado Springs, CO 80907 USA

Technology Licenses

The hardware and/or software described in this document are furnished under a license and may be used or copied only in accordance with the terms of such license.

Declaration of Conformity

Declarations of Conformity for this product and for other Keysight products may be downloaded from the Web. Go to http://www.keysight.com/go/conformity and click on “Declarations of Conformity.” You can then search by product number to find the latest Declaration of Conformity.

U.S. Government Rights

The Software is "commercial computer software," as defined by Federal Acquisition Regulation ("FAR") 2.101. Pursuant to FAR 12.212 and 27.405-3 and Department of Defense FAR Supplement ("DFARS") 227.7202, the U.S. government acquires commercial computer software under the same terms by which the software is customarily provided to the public. Accordingly, Keysight provides the Software to U.S. government customers under its standard commercial license, which is embodied in its End User License Agreement (EULA), a copy of which can be found at http://www.keysight.com/find/sweula. The license set forth in the EULA represents the exclusive authority by which the U.S. government may use, modify, distribute, or disclose the Software. The EULA and the license set forth therein, does not require or permit, among other things, that Keysight: (1) Furnish technical information related to commercial computer software or commercial computer software documentation that is not customarily provided to the public; or (2) Relinquish to, or otherwise provide, the government rights in excess of these rights customarily provided to the public to use, modify, reproduce, release, perform, display, or disclose commercial computer software or commercial computer software documentation. No additional government requirements beyond those set forth in the EULA shall apply, except to the extent that those terms, rights, or licenses are explicitly required from all providers of commercial computer software pursuant to the FAR and the DFARS and are set forth specifically in writing elsewhere in the EULA. Keysight shall be under no obligation to update, revise or otherwise modify the Software. With respect to any technical data as defined by FAR 2.101, pursuant to FAR 12.211 and 27.404.2 and DFARS 227.7102, the U.S. government acquires no greater than Limited Rights as defined in FAR 27.401 or DFAR 227.7103-5 (c), as applicable in any technical data.

Warranty

THE MATERIAL CONTAINED IN THIS DOCUMENT IS PROVIDED "AS IS," AND IS SUBJECT TO BEING CHANGED, WITHOUT NOTICE, IN FUTURE EDITIONS. FURTHER, TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, KEYSIGHT DISCLAIMS ALL WARRANTIES, EITHER EXPRESS OR IMPLIED WITH REGARD TO THIS MANUAL AND ANY INFORMATION CONTAINED HEREIN, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. KEYSIGHT SHALL NOT BE LIABLE FOR ERRORS OR FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES IN CONNECTION WITH THE FURNISHING, USE, OR PERFORMANCE OF THIS DOCUMENT OR ANY INFORMATION CONTAINED HEREIN. SHOULD KEYSIGHT AND THE USER HAVE A SEPARATE WRITTEN AGREEMENT WITH WARRANTY TERMS COVERING THE MATERIAL IN THIS DOCUMENT THAT CONFLICT WITH THESE TERMS, THE WARRANTY TERMS IN THE SEPARATE AGREEMENT WILL CONTROL.

Safety Information

CAUTION

A CAUTION notice denotes a hazard. It calls attention to an operating procedure, practice, or the like that, if not correctly performed or adhered to, could result in damage to the product or loss of important data. Do not proceed beyond a CAUTION notice until the indicated conditions are fully understood and met.

WARNING

A WARNING notice denotes a hazard. It calls attention to an operating procedure, practice, or the like that, if not correctly performed or adhered to, could result in personal injury or death. Do not proceed beyond a WARNING notice until the indicated conditions are fully understood and met.

Contents

1. Understanding PXIe Digitizers Theory of Operation

Keysight M31/M33XX digitizers are part of the new M3XXXA family of FPGA-programmable AWGs and Digitizers. These high-performance digitizers with high channel density have an advanced data acquisition system (DAQ), includes easy-to-use programming libraries and provides optional real-time sequencing and decision making capability using the Hard Virtual Instrumentation (HVI) technology with precise timing and multi-module synchronization. Graphical FPGA programing allows for FPGA customization without HDL programming expertise and performance penalty. This is a new family of FPGA-Programmable Digitizers with precise multi-module synchronization and real time sequencing technology (HVI).

Section 1.1: Understanding Digitizer's operation

The M31/M33XXA Digitizers has a flexible and powerful input structure to acquire signals (Figure 1).

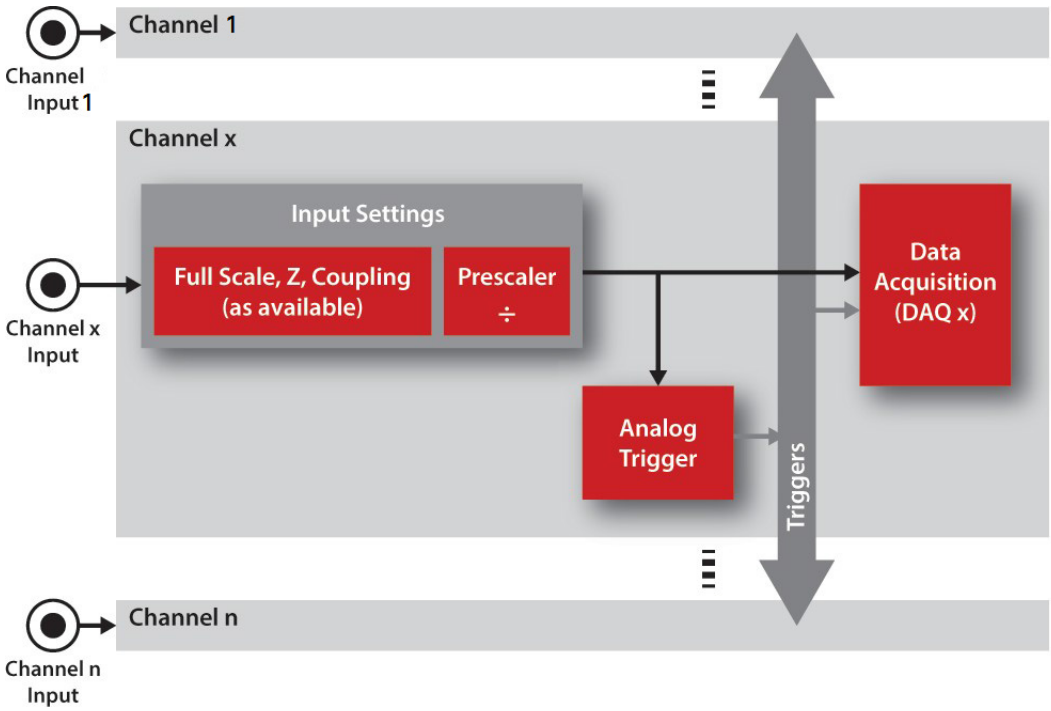


Figure 1 M31/M33XXA Digitizers input functional block diagram

1.1.1: Channel Numbering and Compatibility Mode

Table 1 shows Keysight standards for the output labeling (channel enumeration starts with CH1). Compatibility mode, which can be changed by open(), is available to support legacy modules and allows the channel numbering (channel enumeration) to start with either CH0 or CH1.

Modules are opened by default with the enumeration mode of its front panel. However, it is possible to open them in compatibility mode, forcing enumeration to the selected option. This option might be needed when different modules coexist.

Table 1 Compatibility mode options

Option	Description	Name	Value
Legacy	Channel enumeration starts with CH0	COMPATIBILITY_LEGACY	0
Keysight	Channel enumeration starts with CH1	COMPATIBILITY_KEYSIGHT	1

Legacy modules refer to SD1 modules that were manufactured by Signadyne before they were acquired by Keysight Technologies. If the hardware equipment configuration being used only contains modules from Keysight Technologies, channel enumeration should start with CH1.

1.1.2: Input settings

The M31/M33XXA Digitizers provides a block that allows the user to configure all the input settings such as input impedance, full scale, coupling, prescaler, and so on.

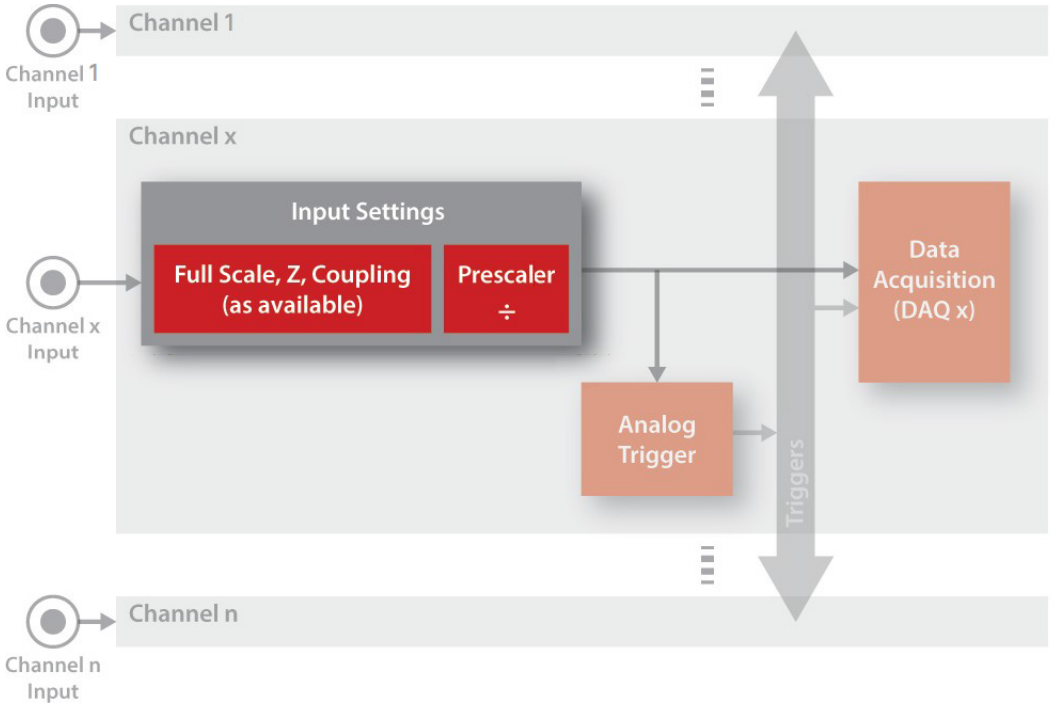


Figure 2 Input settings in the M31/M33XXA Digitizers functional block diagram

Full Scale, Impedance and Coupling

Depending on the product specifications the user can configure the input full scale value, the input impedance (Table 2) and the input coupling (Table 3). The full scale parameter adjusts automatically the input gain to maximize the input dynamic range.

Product-dependent Settings: This section describes all the possible input settings, but in reality they are product-dependent. Check the corresponding Product Data Sheet to see if they are applicable.

Table 2 Input Impedance options

Option	Description	Name	Value
High Impedance	Input impedance is high (value is product dependent, check the corresponding Data Sheet)	AIN_IMPEDANCE_HZ	0
50Ω	Input impedance is 50Ω	AIN_IMPEDANCE_50	1

Table 3 Input Coupling options

Option	Description	Name	Value
DC	DC coupling	AIN_COUPLING_DC	0
AC	AC coupling	AIN_COUPLING_AC	1

Prescaler

The prescaler is used to reduce the effective input sampling rate, capturing 1 out of n samples and discarding the rest. The resulting sampling rate is as follows:

$$\begin{cases} f_s = f_{CLKsys} & prescaler = 0 \\ f_s = \frac{f_{CLKsys}}{5 \cdot prescaler} & prescaler > 0 \end{cases}$$

Figure 3 Resulting sampling rate when prescaler is applied

$f_s = f_{CLKsys} / (1 + prescaler)$

where, f_s is final effective sampling frequency

f_{CLKsys} is the Clock System

prescaler is an integer value (4 bits)

NOTE Prescaler vs. Downsampling/Decimation: The prescaler cannot be considered as a full decimation or down-sampling block as it does not contain any filters and therefore, it generates aliasing. For applications where full downsampling is required, you must choose an IF Keysight Digitizer or Transceiver, which provides DDC (Digital Down Conversion).

Table 4 Programming functions related to the input prescaler

Function Name	Description	API function
channelPrescalerConfig	Configures the input prescaler	channelPrescalerConfig()

1.1.3: Analog Trigger

The analog trigger block processes the input data and generates a digital trigger that can be used by any Data Acquisition Block (“Data Acquisition (DAQs)” on page 15).

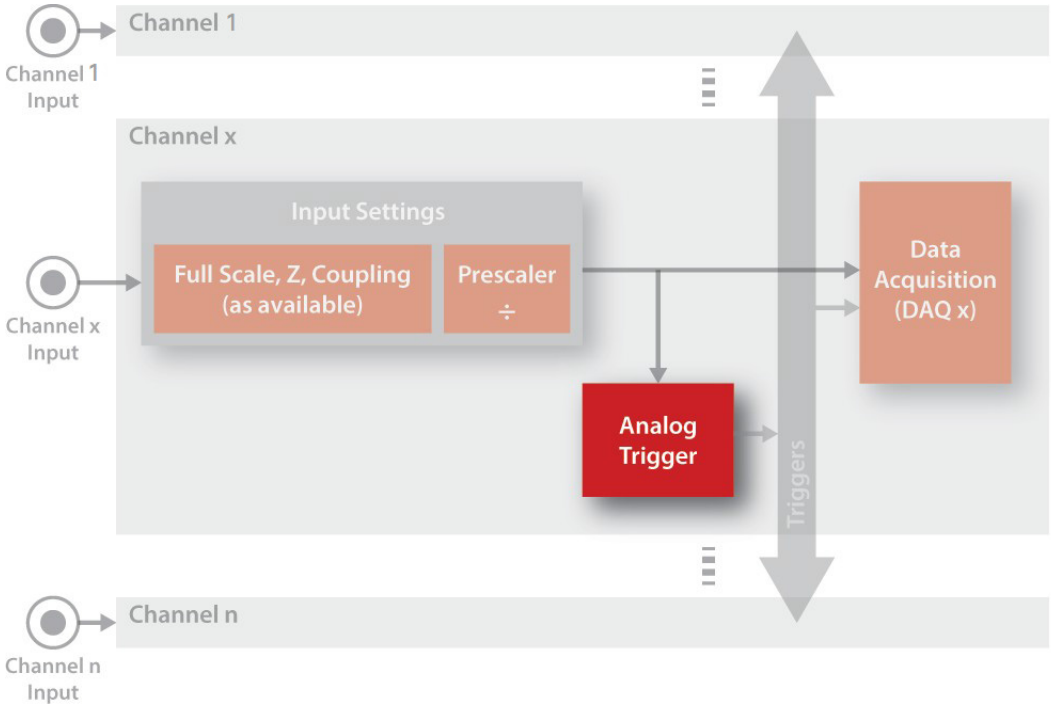


Figure 4 Analog trigger processor in the M31/M33XXA Digitizers functional block diagram

You can select the threshold and the trigger mode. The available trigger modes are shown in Table 5.

Table 5 Analog Trigger Mode options

Option	Description	Name	Value
Positive Edge	Trigger is generated when the input signal is rising and crosses the threshold	AIN_RISING_EDGE	1
Negative Edge	Trigger is generated when the input signal is falling and crosses the threshold	AIN_FALLING_EDGE	2
Both Edges	Trigger is generated when the input signal crosses the threshold, no matter if it is rising or falling	AIN_BOTH_EDGES	3

Table 6 Programming functions related to analog triggers

Function Name	Description	API function
channelTriggerConfig	Configures the analog trigger for each channel	channelTriggerConfig()

1.1.4: Data Acquisition (DAQs)

The Data Acquisition (DAQ) is a powerful and flexible block which acquires incoming words and sends them to the user PC using dedicated DMA channels (Figure 5).

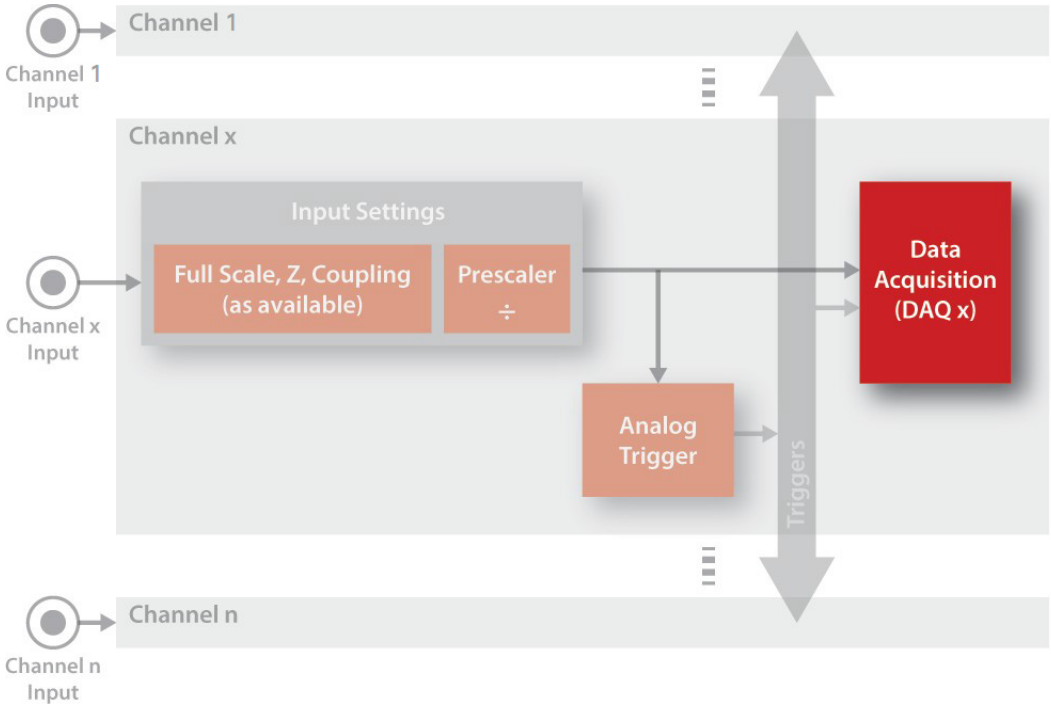


Figure 5 DAQ units in the M31/M33XXA Digitizers functional block diagram

Operation

The words acquisition requires two easy steps:

1 Configuration: A call to the function "DAQconfig()" allows the user to configure, among others, the trigger method (Table 7), the number of DAQ cycles to perform (number of triggers), the number of acquired words per cycle (DAQpointsPerCycle), and the number of words that must be acquired before interrupting the PC (DAQpoints). Figure 6 illustrates the flexibility of the DAQ block operation.

2 Data read:

a Using "DAQread()": a blocking/non-blocking function that returns the array of acquired words (DAQdata).

b Using a callback function: a user function that is automatically called when the configured amount of data (set with DAQpoints) is available.

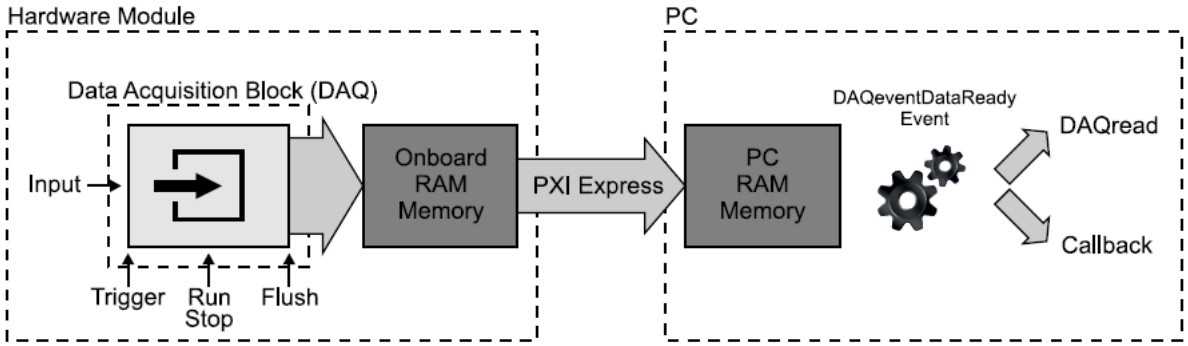


Figure 6 M31/M33XXA Digitizers words acquisition operation

Pausing and Resuming the DAQ: The function "DAQpause()" pauses the DAQ operation (triggers are discarded). The acquisition can be resumed calling "DAQresume()". A "DAQstop()" is performed automatically when the DAQ block reaches the specified number of acquisition cycles.

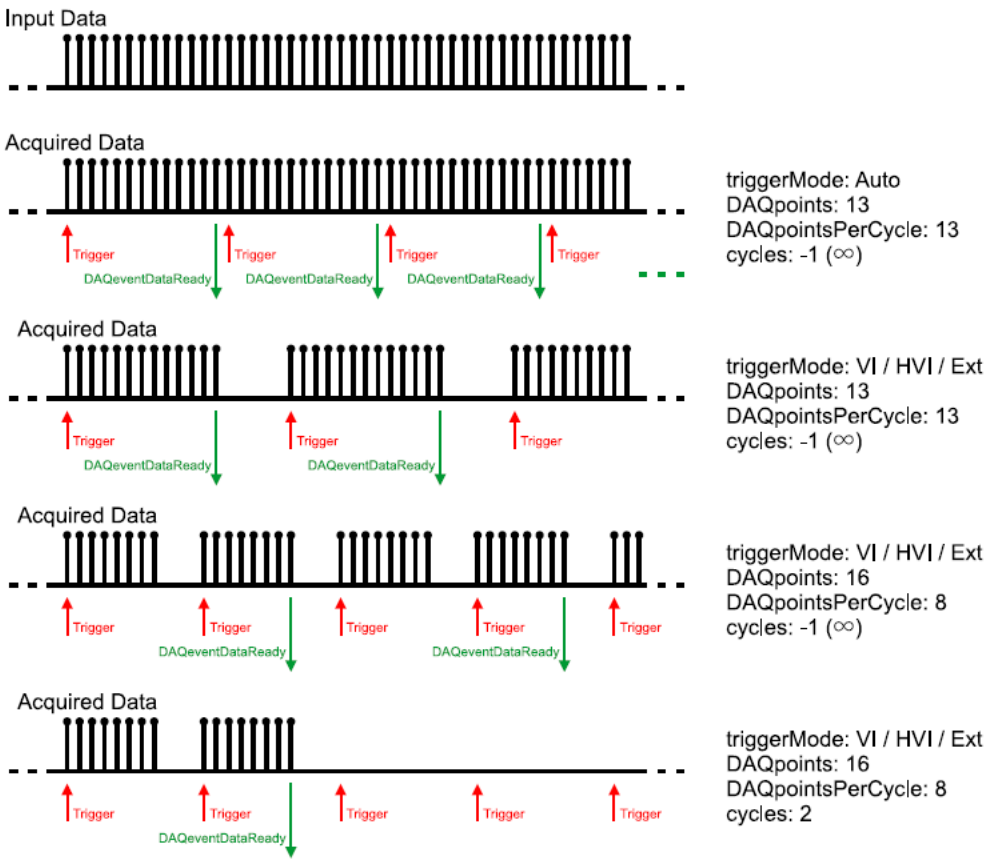


Figure 7 Examples of the DAQ operation

Onboard memory and DAQpoints selection: The acquired words are first stored in a DAQ buffer located in the module onboard RAM and then sent to the PC RAM via the ultra-fast PXI Express bus (Figure 6) using dedicated DMA channels. This DAQ buffer, and therefore the minimum amount of onboard RAM needed for an application, is directly the size of the data array, which you want to

receive in each "DAQread()" or callback function call. This data array, called (DAQdata), contains (DAQpoints) words. Therefore, its size is calculated as DAQpoints x 2 bytes/word (in the case of the M31/M33XXA Digitizers). The size of this DAQ buffer must be chosen according to two criteria:

- High speed transients: the DAQ onboard buffer can store bursts of words at higher rates than the PXIe and the computer can handle. Larger buffers allow handling longer high speed bursts.
- PC load: The DAQ buffer must be chosen to allow the PC enough time to process each DAQ buffer (DAQdata) and to handle interrupts and process-context switching tasks. For example, if a computer requires 500 ms to process DAQdata, the buffer size must be such that it can store more than 500 ms of input words without interrupting the computer. The DAQ block facilitates the task of adjusting the PC processing rate with the introduction of the DAQ cycles (Figure 6).

NOTE The amount of required PC RAM is double the amount of required onboard RAM.

Prescaler: The DAQ block has an input prescaler which can be configured to discard words, reducing the effective acquisition data rate "DAQconfig()".

DAQ counter: The DAQ block has a dedicated counter to store the number of acquired words since the last call to "DAQconfig()" or "DAQflush()". This counter can be read with "DAQcounterRead()".

DAQ Trigger

As previously explained, you can configure the trigger for the acquisition. The available trigger modes for the DAQ are shown in Table 7. Note that not all trigger methods are available in all modules.

Table 7 DAQ Trigger Mode options

Option	Description	Name	Value
Auto (Immediate)	The acquisition starts automatically after a call to function DAQstart	AUTOTRIG	0
Software / HVI	Software trigger. The acquisition is triggered by the function DAQtrigger, DAQtrigger provided that the DAQ is running. DAQtrigger can be executed from the user application (VI) or from an HVI.	SWHVITRIG	1
Hardware Digital Trigger	Hardware trigger. The DAQ waits for an external digital trigger (see Table 8 External Hardware Digital Trigger Source for the DAQ).	HWDIGTRIG	2
Hardware Analog Trigger	Hardware trigger. The DAQ waits for an external analog trigger (only products with analog inputs).	HWANATRIG	3

As shown in Table 7, you have the following options for hardware triggers:

- Hardware Digital Trigger: If the DAQ is set to use a digital hardware trigger, you must configure it using the function "DAQdigitalTriggerConfig()". The available digital hardware trigger options are shown in Table 8 and Table 9. If external I/O trigger is selected in "DAQdigitalTriggerConfig()", you must configure additional settings of this particular I/O line, such as input/output direction, sampling/synchronization options, etc. ("Working with I/O Triggers" on page 19).
- Hardware Analog Trigger: (Only products with analog inputs) If the DAQ is set to use an analog hardware trigger, you must configure it using the function "DAQanalogTriggerConfig()". The Analog Trigger Block of the corresponding analog input channel must also be configured.

Table 8 External Hardware Digital Trigger Source for the DAQ

Option	Description	Name	Value
External I/O Trigger	The DAQ trigger is a TRG connector/line of the product (I/O Triggers). PXI form factor only: this trigger can be synchronized to CLK10.	TRIG_EXTERNAL	0
PXI Trigger	PXI form factor only. The DAQ trigger is a PXI trigger line and it is synchronized to CLK10.	TRIG_PXI	1

Table 9 Trigger behavior for the DAQ

Option	Description	Name	Value
None	No trigger has been activated	TRIGGER_NONE	0
Active High	Trigger is active when it is at level high	TRIGGER_HIGH	1
Active Low	Trigger is active when it is at level Low	TRIGGER_LOW	2
Rising Edge	Trigger is active on the rising edge	TRIGGER_RISE	3
Falling Edge	Trigger is active on the falling edge	TRIGGER_FALL	4

Section 1.2: Working with I/O Triggers

The M3100A/M3102A PXIe Digitizers have general purpose input/output triggers (TRG connectors/lines). A trigger can be used as a general purpose digital IO or as a trigger input, and can be sampled using the options shown below in Trigger Synchronization/Sampling Options.

Table 10 I/O Trigger types and corresponding functions

Type	Description	Name	Value
Trigger Output (readable)	TRG operates as a general purpose digital output signal, that can be written by the user software.	AIN_TRG_OUT	0
Trigger Input	TRG operates as a trigger input, or as general purpose digital input signal, that can be read by the user software.	AIN_TRG_IN	1

Table 11 Trigger Synchronization options

Type	Description	Name	Value
Non-synchronized mode	The trigger is sampled with an internal 100 MHz clock.	SYNC_NONE	0
Synchronized mode	(PXI form factor only) The trigger is sampled using CLK10.	SYNC_CLK10	1

Section 1.3: Working with Clock System

The M3100A/M3102A PXIe Digitizer uses an internally generated high-quality clock (CLKref) which is phase-locked to the chassis clock. Therefore, this clock is an extremely jitter-cleaned copy of the chassis clock. This implementation achieves a jitter and phase noise above 100 Hz which is independent of the chassis clock, depending on it only for the absolute frequency precision and long term stability. A copy of CLKref is available at the CLK connector.

CLKref is used as a reference to generate CLKsys, the high-frequency clock used to sample data.

Chassis Clock Replacement for High-Precision Applications

For applications where clock stability and precision is crucial (for example: GPS, experimental physics, etc.), you can replace the chassis clock with an external reference.

In the case of PXI/PXIe, this is possible via a chassis clock input connector or with a PXI/PXIe timing module. These options are not available in all chassis; see the corresponding chassis specifications.

1.3.1: CLK Output Options

Table 12 Clock Output options

Options	Description	Name	Value
Disable	The CLK connector is disabled.	N/A	0 (default)
CLKref Output	A copy of the reference clock is available at the CLK connector.	N/A	1

1.3.2: FlexCLK Technology (models w/ variable sampling rate)

The sampling frequency of the M3100A/M3102A PXIe Digitizers (CLKsys frequency) can be changed using the advanced clocking system.

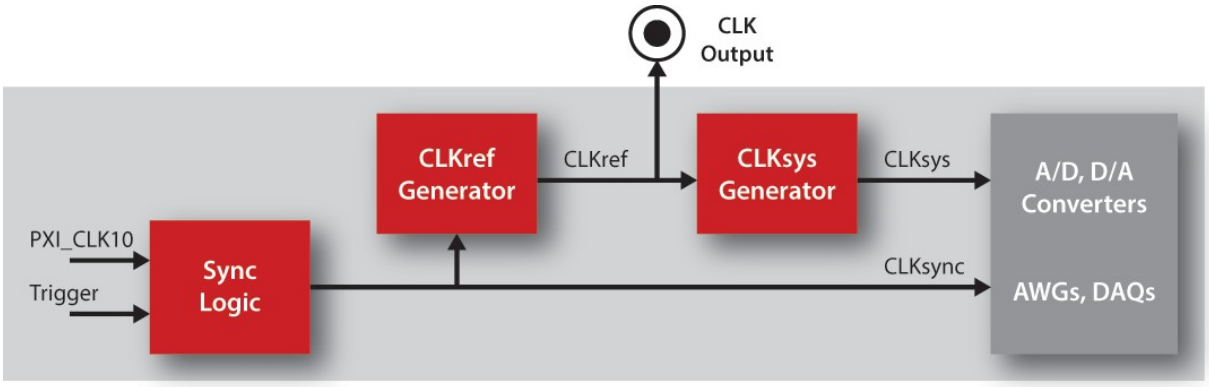


Figure 8 Block Diagram depicting the Advanced Flex Clock System in PXIe DIGs

FlexCLK System, where:

- CLKref is the internal reference clock, and is phase-locked to the chassis clock.
- CLKsys is the system clock used to sample data.

- CLKsync is an internal clock used for the synchronization features of the M3100A/M3102A PXIe Digitizers.
- PXI CLK10 is the 10 MHz clock of the PXI/PXIe backplane.

The CLKsys frequency can be changed within the range indicated in the Data Sheet of the corresponding product [clockSetFrequency()]. The CLKsync frequency changes with the CLKsys frequency as per the following equation:

$$f_{CLKsync} = \text{GreatestCommonDivisor}\ (f_{PXI\_CLK10}, f_{CLKsys} / 5)$$

The CLKsync frequency is returned by clockSetFrequency().

Table 13 CLKsync frequency mode options

Options	Description	Name	Value
Low Jitter Mode	The clock system is set to achieve the lowest jitter, sacrificing tuning speed.	CLK_LOW_JITTER	0
Fast Tuning Mode	The clock system is set to achieve the lowest tuning time, sacrificing jitter performance.	CLK_FAST_TUNE	1

2. Using the KS2201A PathWave Test Sync Executive Software

This chapter provides an introduction to the KS2201A PathWave Test Sync Executive Software and describes its implementation in the SD1 3.x API. For detailed information about the KS2201A PathWave Test Sync Executive Software and its API, refer to the *KS2201A PathWave Test Sync Executive Software User Guide*.

Section 2.1: Licensing for KS2201A PathWave Test Sync Executive software

Hardware license option (-HV1) on the M3xxxA modules

All M3xxxA modules support HVI technology. However, the hardware license option *-HV1* must be available on each module that is required to be programmed using the KS2201A PathWave Test Sync Executive software and for usability with SD1 3.x software. The newer M3xxxA cards are shipped with the newest versions of firmware and SD1 software, which support the PathWave Test Sync Executive software. During procurement, you may choose to procure the *-HV1* hardware option.

To use an older module with the KS2201A PathWave Test Sync Executive software, the firmware and SD1 software must be upgraded. KS2201A PathWave Test Sync Executive software requires that Keysight SD1 SFP software version 3.x be installed on the same machine. Also, the PXIe M3xxxA modules products must have Firmware versions greater than or equal to 4.0 (for M320xA AWGs / M330xA Combos) and greater than or equal to 2.0 (for M310xA Digitizers). For more information regarding the supported firmware and software versions, refer to the *SD1 3.x Software Startup Guide*.

Software license option for the KS2201A software

Refer to the *KS2201A PathWave Test Sync Executive User Guide* to know about the licenses that you must procure for the KS2201A PathWave Test Sync Executive software.

Section 2.2: Comparing ProcessFlow GUI with KS2201A HVI API

NOTE Beginning with SD1 3.x software release, the M3601A Hard Virtual Instrument (HVI) Design Environment (ProcessFlow) is replaced by the KS2201A PathWave Test Sync Executive Software for HVI integration. Both the GUI elements and the API functions from the former HVI design environment are not supported in the SD1 3.x software.

Table 14 summarizes the main operations necessary in an HVI design as performed from the point of view of the M3601A HVI GUI use model and the KS2201A HVI API use model. You may use this table of equivalence to transition from ProcessFlow to the KS2201A PathWave Test Sync Executive software.

Table 14 Differences in operations of ProcessFlow and KS2201A HVI API

Operations	ProcessFlow Use Model	KS2201A HVI API Use Model
HVI Design Flow	First, a ".HVIprj" project file must be created using M3601A GUI to design the required HVI sequences in form of flow-charts. A binary ".HVI" file is generated from the ".HVIprj" file once the HVI sequence design is final. The ".HVI" file must be open from code to integrate the HVI solution into the application code.	Application code must import the "keysight_pathwave_hvi" library to use the HVI API. HVI sequences can be created using programs directly into the application code without importing external files.
HVI Sequence	It is implemented by means of a graphical flowchart. Each HVI Engine in each instrument has a single or main HVI Sequence associated. All statements, both local and synchronized, are added to it graphically.	KtHviSequence class enables you to create a Local HVI sequence using programs that run "locally" on a specific HVI engine in a specific instrument. Local Sequences are accessed using 'SyncMultiSequenceBlock' statement placed in a SyncSequence (KtHviSyncSequence). The HVI top sequence is a SyncSequence that contains SyncStatements.
HVI SyncSequence	The concept of HVI SyncSequences is not available in the M3601A flowcharts.	KtHviSyncSequence class enables you to add synchronized operations (Sync Statements) common to all HVI engines within the HVI instance. The HVI top sequence is a SyncSequence that contains SyncStatements. Local instructions are added and executed within Local Sequences that can be accessed by adding a 'SyncMultiSequenceBlock' in a SyncSequence.
HVI Resources (Chassis, Triggers, M9031A modules, and so on)	Connected chassis are automatically recognized. M9031A boards are transparent to the M3601A software. PXI trigger resources that can be allocated to the HVI solution are chosen from the "Chassis settings" window.	HVI resources can be configured using "KtHviPlatform" class and all the classes inside it.
Program HVI Sequences	You may program HVI sequences by adding flowchart boxes using the M3601A GUI. Configure settings for statements in the "Properties" window of each flowchart box.	You may program both HVI SyncSequences and HVI (Local) Sequences with the API methods add_XXX(), where 'XXX' is the statement name.
HVI Compile, Load, Run	Once an ".HVI" file is open from a script, you can assign each sequence to an HW engine for it to be compiled, loaded to HW, and executed. Project ".HVIprj" files can be also tested directly from the M3601A GUI using the "Compile and Run" function.	API SW methods can compile the sequence using (hvi.compile()), load it to hardware using (hvi.load_to_hw()), run the sequence using (hvi.run()).

Section 2.3: Working with KS2201A PathWave Test Sync Executive software

Beginning with Keysight SD1 3.x release, the KS2201A PathWave Test Sync Executive software has been introduced to enhance the functionalities of one or more PXIe modules both individually and interactively. PathWave Test Sync Executive is a new API based environment for developing and running programs with a new generation of Keysight's Hard Virtual Instrument (HVI) technology. The KS2201A software enables programmatic development and execution of synchronous real-time operations across multiple instruments. It enables you to program multiple instruments together so they can act together with other instruments, like one instrument.

2.3.1: Overview on HVI technology

Keysight's Hardware Virtual Instrumentation (HVI) technology provides the capability to create time-deterministic execution sequences with precise synchronization by deploying FPGA hardware simultaneously among the constituent instruments. This makes the technology a powerful tool in MIMO systems, such as massive-scale quantum control networks.

A virtual instrument may be considered to function like any other instrument in the system; its main objective being to digitally sequence events and instructions in the application while synchronizing multiple modules. This instrument, (referred to in this document as the HVI instrument), accomplishes this by running one or more "engines" synchronously by referencing a common digital clock that all instruments (engines) operate on.

The KS2201A PathWave Test Sync Executive software provides you with the capability of designing HVI sequences using an Application Programming Interface (API) available in both Python and C# coding languages. The HVI Application Programming Interface (API) is the set of programming classes and methods that allows the user to create and program an HVI instance. HVI API currently supports Python (version 3.7 only). The HVI core functionality is extended by the PXIe M3xxxA modules using the SD1 API. The core HVI features and the SD1 API extensions that are specific to M3xxxA, allow a heterogeneous array of instruments and resources to coexist on a common framework.

All the PXIe M3xxxA modules support HVI technology. When Keysight SD1 is installed on a PXI system, it installs the drivers required to interact with the M3xxxA series modules. The SD1 API classes in the Keysight SD1 Library contain HVI add-on interfaces provided as an extension of the instrument. These add-on interfaces provide access to instrument specific HVI features such as triggering a digitizer acquisition, outputting a waveform, queuing a waveform, etc.

The primary HVI elements defined for the SD1 API and the corresponding API functions are described in the following sections.

Section 2.4: Understanding the HVI elements used in SD1 API

The HVI Core API exposes all HVI functions and defines base interfaces and classes, which are used to create an HVI, control the hardware execution flow, and operate with data, triggers, events and actions, but it alone does not include the ability to control operations specific to the M3xxxA product family. It is the HVI instrument extensions specific to M3xxxA modules that enable instrument functionalities in an HVI. Such functions are exposed by the module specific add-on HVI definitions. The SD1 API describes the instrument specific resources and operations that can be executed or used within HVI sequences.

Figure 9 & Figure 10 display the AWG and Digitizer specific HVI definitions, which are added to the SD1 library.

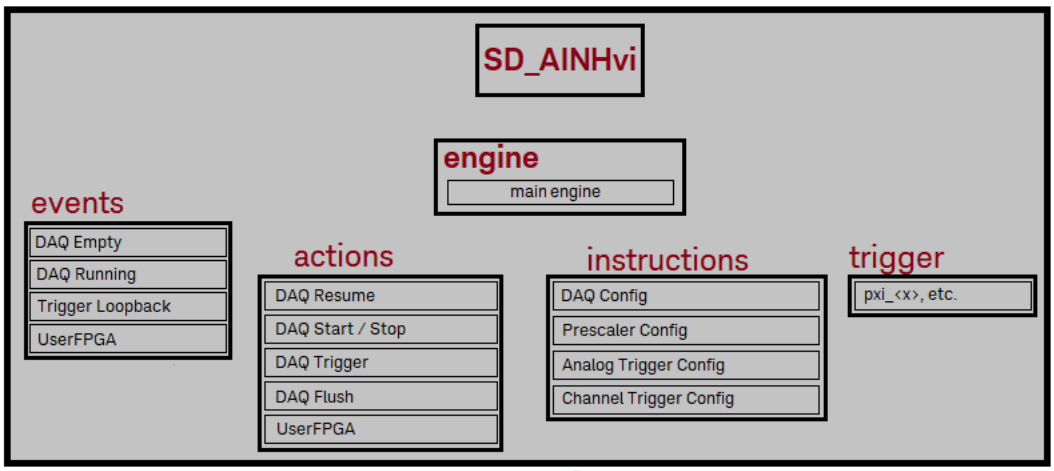


Figure 9 M310xA specific HVI definitions

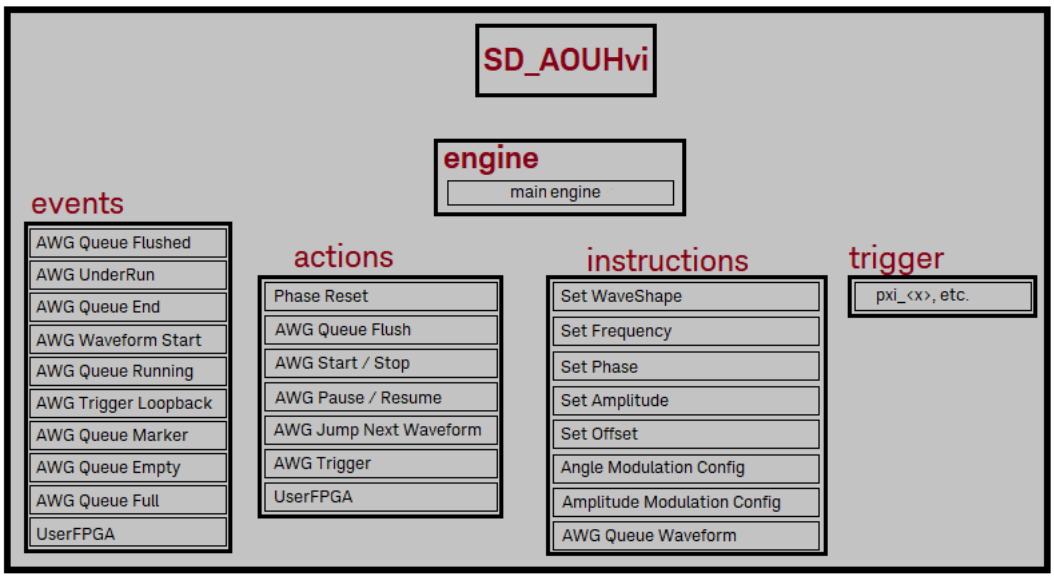


Figure 10 M320xA specific HVI definitions

2.4.1: Description of various HVI elements

HVI Engine

An HVI Engine block controls the functions of the instrument and the timing of operations. For HVI to control an SD1 module, the latter requires an HVI Engine. The HVI Engine is included directly in the instrument hardware or it can be programmed using the SD1 API into the Field programmable Gate Array (FPGA) on each module. The HVI Engine executes sequences, which are made up of Statements.

To define an HVI Engine in SD1 API, see the API syntax in Module HVI Engine.

HVI Actions

An HVI Action is defined for module-specific operations, such as playing waveform in an AWG or starting an acquisition in Digitizers.

To define an HVI Action in SD1 API, see the API syntax for various HVI actions in SD_AIN functions and SD_Module functions (specific to Pathwave FPGA).

HVI Events

An HVI Event is defined to occur when specific conditions are met during module-specific operations, such as when an AWG queue is flushed or when a DAQ is empty.

To define an HVI Event in SD1 API, see the API syntax for various HVI events in SD_AIN functions and SD_Module functions (specific to Pathwave FPGA).

HVI Trigger

An HVI Trigger is defined to activate the logic signal triggering source and perform various triggering operations, which are shared between instruments to initiate module-related operations, communicate states or other information.

To define an HVI Trigger in SD1 API, see the API syntax in Module HVI Triggers.

HVI Instructions

An HVI Instruction is defined to configure various settings related to the module. There are two types of HVI instructions:

- Product specific (custom) HVI instructions—can change a module's setting (such as amplitude, frequency, etc.) or trigger a functionality in the module (such as output a waveform, trigger a data acquisition, etc.).
- HVI core instructions (general purpose)—provide global, non-module specific or custom functions, such as register arithmetic, read/write general purpose I/O triggers, execution actions, etc.

To define an HVI Trigger in SD1 API, see the API syntax in SD_AIN functions.

FPGA sandbox registers

For the modules that contain an FPGA with a user-configurable sandbox, HVI can, potentially (if the configuration of the module allows it), access (read/write) the registers that you define in that sandbox. To accomplish this, you must obtain the "*.k7z*" file for the FPGA sandbox, generated by the PathWave FPGA application. This file contains all the necessary information to access the registers by name.

To define FPGA Action/Event in SD1 API, see the API syntax in User FPGA HVI Actions/Events.

Section 2.5: Implementing HVI in SD1 API - Sample Programs

The following section shows a sample program where HVI is implemented in SD1 API to create an HVI sequence, add HVI main engine, define HVI trigger, assign 'module hvi instruction set' functions followed by compiling the HVI sequence and loading it onto the hardware. Based on your requirements, you may add 'module hvi actions' as well as 'module hvi events' functions to the HVI sequence.

The SD1 API functions related to HVI are covered in Chapter 5, "Using Keysight SD1 API Command Reference".

Refer to the *KS2201A PathWave Test Sync Executive User Guide* to know more about the HVI Python API.

2.5.1: Sample program using Python for HVI instructions

```
import sys
import matplotlib.pyplot as plt
import keysight_hvi as kthvi
sys.path.append(r'C:\Program Files (x86)\Keysight\SD1\Libraries\Python')
import keysightSD1

dig = keysightSD1.SD_AIN()
dig.openWithSlot("M3102A", 1, 9)

sys_def = kthvi.SystemDefinition("mySystem")
sys_def.chassis.add_auto_detect()
sys_def.engines.add(dig.hvi.engines.main_engine, 'SdEngine0')

trigger_resources = [kthvi.TriggerResourceId.PXI_TRIGGER0,
kthvi.TriggerResourceId.PXI_TRIGGER1]
sys_def.sync_resources = trigger_resources
sys_def.non_hvi_core_clocks = [10e6]

sequencer = kthvi.Sequencer("mySequencer", sys_def)
sync_block = sequencer.sync_sequence.add_sync_multi_sequence_block("AWGsequence",10)
sequence = sync_block.sequences['SdEngine0']

test_channel = 1
test_daq_points = 500
test_daq_cycles = 10

dig.channelInputConfig(test_channel, 2, keysightSD1.AIN_Impedance.AIN_IMPEDANCE_50,
keysightSD1.AIN_Coupling.AIN_COUPLING_DC)
```

```
instrLabel = "daqConfig"
instruction0 = sequence.add_instruction(instrLabel, 20, dig.hvi.instruction_set.daq_config.id)
instruction0.set_parameter( dig.hvi.instruction_set.daq_config.channel.id, test_channel)
instruction0.set_parameter( dig.hvi.instruction_set.daq_config.cycles.id, test_daq_cycles)
instruction0.set_parameter( dig.hvi.instruction_set.daq_config.daq_points_per_cycle.id,
test_daq_points)
instruction0.set_parameter( dig.hvi.instruction_set.daq_config.trigger_delay.id, 0)
instruction0.set_parameter( dig.hvi.instruction_set.daq_config.trigger_mode.id,
dig.hvi.instruction_set.daq_config.trigger_mode.AUTOTRIG)
# Compile HVI sequences
try:
    hvi = sequencer.compile()
except kthvi.CompilationFailed as ex:
    compile_status = ex.compile_status
    print(compile_status.to_string())
    raise ex
print("HVI Compiled")

# Load HVI to HW: load sequences, configure actions/triggers/events, lock resources, etc.
hvi.load_to_hw()
print("HVI Loaded to HW")

# Execute HVI in non-blocking mode
# This mode allows SW execution to interact with HVI execution
hvi.run(hvi.no_wait)
print("HVI Running...")

print("Plotting measured data... Press enter to exit")
print("")
rawReadoutBufferI = dig.DAQread(test_channel, test_daq_points * test_daq_cycles, 200)

# Plot acquisition data
plt.clf()
plt.plot(rawReadoutBufferI, 'r-')
plt.show()
```

3. Using the PathWave FPGA Board Support Package (BSP)

This chapter provides an introduction to the PathWave FPGA Board Support Package (BSP) and describes its implementation in the SD1 3.x API. For detailed information about using the *PathWave FPGA 2020 Update 1.0* software along with BSP for supported modules, refer to the *PathWave FPGA Customer Documentation* and the *BSP guides* for the respective modules.

Section 3.1: Licensing for PathWave FPGA BSP support

All instruments that can be programmed with PathWave FPGA BSP will enable programming using the licensing option *-FP1*. You are required to procure an *-FP1* license option for each instrument that must be programmed. For example, if you are purchasing ten M3102A DIG cards, each M3102A requires an individual *-FP1* license option enabled.

New Hardware - Latest Software

The new modules for M3201A AWG 1G, M3202A AWG 500 and M3102A DIG500, where you have the *-FP1* license option enabled are supported only with SD1 software version 3.x (or later) installed on your machine. If you have SD1 version 2.x.x installed on your machine, you must upgrade the software for hardware compatibility. For more information regarding the supported firmware and software versions, refer to the *SD1 3.x Software Startup Guide*.

Latest Software - New Hardware

Keysight SD1 SFP software version 3.x or later recognize only those PXIe M3xxxA products that have Firmware versions greater than or equal to 4.0 (for M320xA AWGs / M330xA Combos) and greater than or equal to 2.0 (for M310xA Digitizers). You may either update the firmware on your modules or contact Keysight Sales to upgrade your hardware. After upgrading your hardware, you must have the *-FP1* license option enabled to make it compatible with the latest version of SD1 software.

NOTE Currently, PathWave FPGA BSP is not supported on M3100A DIG100, M3300A Combo500-100 and M3302A Combo500-500 modules, but will be supported in future releases of SD1 3.x software.

The M3201A/M3202A may be purchased as a 4 channel configuration, each of which may include an option to specify a variable sampling clock. Fixed sampling clock is supported, variable sampling clock support will be added in future.

The M3102A may be purchased as a 4 channel configuration, which has fixed sampling clock.

These modules may also be purchased to include one of two possible Xilinx FPGAs:

1 xc7k410tffg676-2

2 xc7k325tffg676-2

The xc7k410tffg676-2 part offers a substantial increase in the amount of FPGA resources available to the user for custom logic. The following table outline the FPGA resources, which are available for custom logic for each part.

Table 15 Available FPGA Resources for custom logic on each module

	xc7k325tffg676-2	xc7k410tffg676-2
Resource Type	4 Channel	4 Channel
Slice LUTs	82800	102000
Slice Registers	165600	204000
DSP Blocks	360	660
Block Ram (RAMB18)	360	660

Section 3.2: Comparing FPGAFlow with PathWave FPGA

NOTE **Beginning with SD1 3.x software release, the M3602A Graphical FPGA Development Environment (FPGAFlow) is replaced by the KF9000A PathWave FPGA software for FPGA programming. The files generated by FPGAFlow are not supported in the SD1 3.x software.**

The following sections describe the differences between FPGAFlow and PathWave FPGA software along with the new features introduced in the latter software.

3.2.1: Differences between FPGAFlow and PathWave FPGA

While most of the features are same, but there are subtle differences in the appearances and certain aspects of both the FPGA programming environments. Table 16 highlights such differences.

Table 16 Differences between FPGAFlow and PathWave FPGA

FPGAFlow	PathWave FPGA
Support limited to Legacy Signadyne boards	Support for larger number of Keysight developed boards
Uses one or more BitGen servers to generate the output bitstream	Uses Xilinx Vivado Design Suite locally on Host Machine to generate the output bitstream
Generates “sbp” file, which is loaded on the legacy module’s FPGA	Generates “k7z” file, which is loaded on the FPGA of the new modules

3.2.2: New features in PathWave FPGA

Apart from the differences listed above, the PathWave FPGA software has certain new elements:

- IPs can be defined using IP-XACT file. This gives you information about the IP name, version, interfaces, category, and so on.
- A repository of IPs that are using IP-XACT can be loaded concurrently.
- Block designs are now called sub-modules.
- Multiple boards are supported using the BSP technology.

Section 3.3: Working with PathWave FPGA software

Beginning with Keysight SD1 3.x release, the KF9000A PathWave FPGA Programming Environment (commonly known as PathWave FPGA), powered by *Xilinx Vivado Design Suite*, has been introduced for FPGA designing on supported Keysight modules.

Some applications require the use of custom on-board real-time processing, which might not be covered by the comprehensive off-the-shelf functionalities of standard hardware products. For these applications, Keysight supplies Option *-FP1* (Enabled FPGA Programming) that provide the capability to program the on-board FPGA. With PathWave FPGA, development time is dramatically reduced and you can focus exclusively on expanding the functionality of the standard instrument, instead of developing a complete new one.

PathWave FPGA is a graphical environment that provides a complete FPGA design flow for rapid FPGA development from design creation to simulation to Gateware deployment to Hardware/Gateware verification on Keysight hardware with Open FPGA. This environment provides a design flow from schematic to bitstream file generation with the click of a button.

NOTE **The PathWave FPGA 2020 is a licensed software. Contact Keysight Support for more information on procuring the respective licenses.**

Once you have installed the PathWave FPGA 2020 software, you can launch its user interface from the **Start** menu. Alternatively, you may use the corresponding command line or scripts to launch the application.

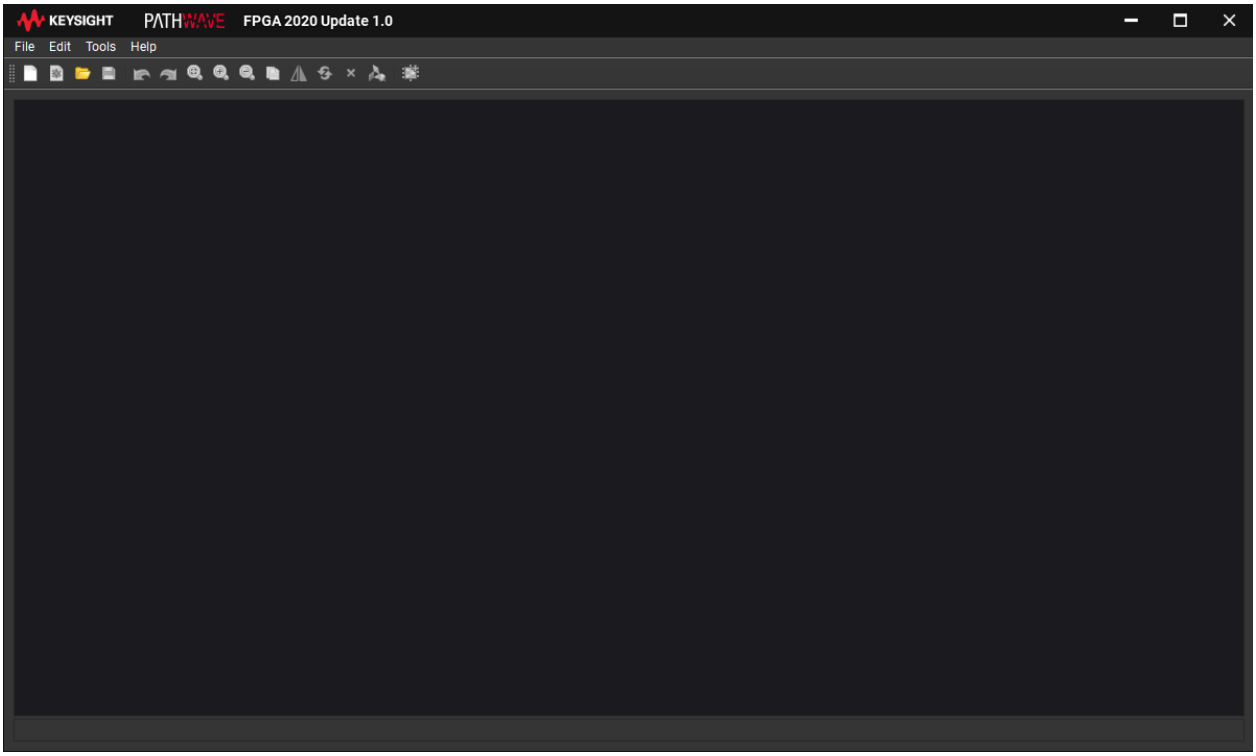


Figure 11 Default window for PathWave FPGA 2020 Update 1.0 software

In Keysight PathWave, FPGA code is represented as boxes (called blocks) with IO ports. An empty project contains the “Default Product Blocks”, and the “Design IO Blocks” that provide the outer interface of the design. You can add/remove blocks from the Keysight Block Library, External Blocks, or Xilinx IP cores.

To know about the required prerequisite software, system requirements and licensing information for the *PathWave FPGA 2020 Update 1.0* software, refer to the “*Getting Started*” section of the *PathWave FPGA Customer Documentation*. To view the installation procedure, refer to the “*Installing PathWave FPGA 2020 Update 1.0 software*” section and to view how to launch the software, refer to the “*Launching the PathWave FPGA BSP*” section in the *SD1 3.x Software Startup Guide*.

3.3.1: Understanding Partial Configuration (PR)

The FPGA configurable region utilizes Xilinx FPGA partial reconfiguration technology to allow you to design and configure a defined section of the supported M3xxxA FPGA without the need to power down or reboot the M3xxxA module or host computer, respectively.

PathWave FPGA supports the design flow in the Partial Reconfiguration (PR) technology. In a PR flow, a full FPGA reconfiguration is only necessary once for a given static region version. The sandboxes can be reconfigured anytime, without a full reconfiguration, and without stopping the current operation of the FPGA.

Partial Reconfiguration enables performing dynamic change of modules within an active design. By implementing multiple configurations in this flow, you can achieve full bitstream for each configuration and partial bitstream for each reconfigurable module.

While FPGA technology facilitates direct programming and re-programming of modules without going through re-fabrication with a modified design; the partial reconfiguration technique allows the modification of an operating FPGA design by loading a partial configuration file. The logic in the FPGA design is divided into two different types, reconfigurable logic and static logic. The static logic remains functional and is unaffected by the loading of a partial bitstream file, whereas the reconfigurable logic is replaced by the contents of that bitstream file.

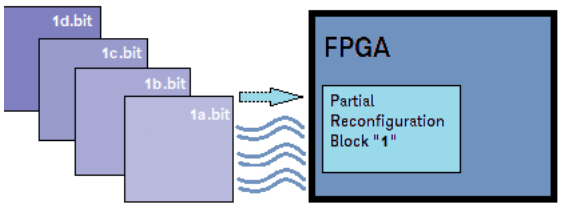


Figure 12 Block diagram depicting partial reconfiguration using bitstream files

Partial Reconfiguration (PR) is modifying a subset of logic in an operating FPGA design by downloading a partial bitstream. In some cases, a complete FPGA reconfiguration might be preferable (better routing, timing closure, and so on). This is also supported by PathWave FPGA, as long as a reboot is not required after the reconfiguration process.

The PathWave FPGA software does not, by itself, provide access to the waveform and digitizer controls for the supported Keysight M3xxxA PXIe modules. You must install the module-specific Board Support Package (BSP) to leverage the features within the PathWave FPGA software for FPGA designing and for loading your customized code onto the Keysight instrument.

The design part in Keysight FPGA consists of two regions: the static region and the sandbox region. The static region for each supported module is defined within BSP and cannot be modified. This region defines the implementation of the FPGA interfaces to external resources, and defines the

interfaces to the sandbox. A static region implementation can define one or more sandbox regions in an FPGA design. The sandbox region contains the user specific FPGA design. The interface of the sandbox depends on the static region implementation.

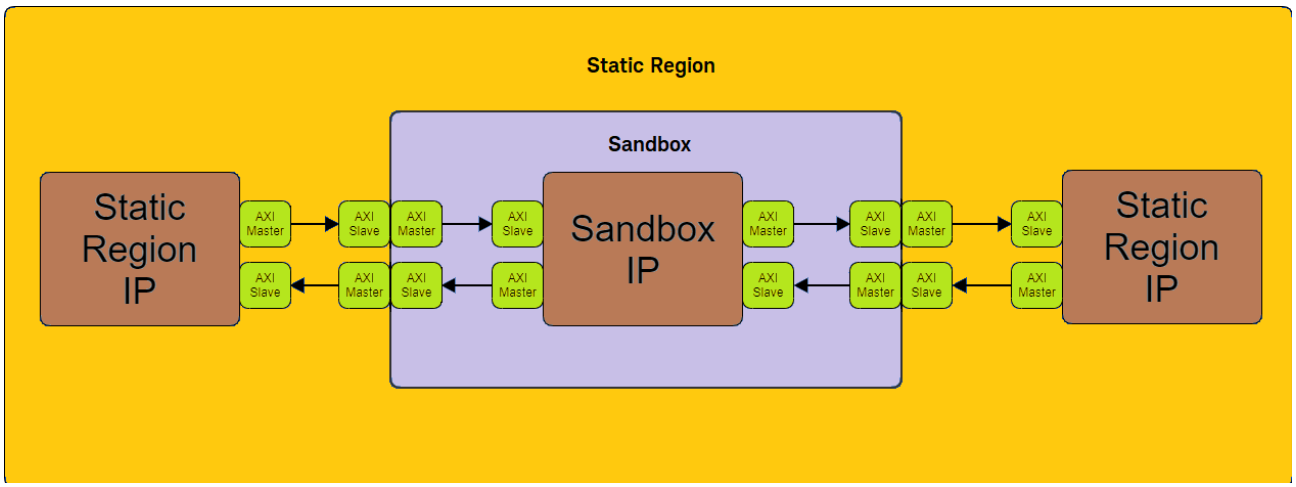


Figure 13 Block diagram representing layout of Static and Sandbox regions

Section 3.4: Using BSP with PathWave FPGA software

3.4.1: Understanding BSP composition

In embedded systems, the board support package (BSP) is the layer of software containing hardware-specific drivers and other routines that allow a particular operating system (traditionally a real-time operating system, or RTOS) to function in a particular hardware environment (a computer or CPU card), integrated with the RTOS itself. Third-party hardware developers, who wish to support a particular RTOS must create a BSP that allows that RTOS to run on their platform. In most cases the RTOS image and license, the BSP containing it, and the hardware are bundled together by the hardware vendor. BSPs are typically customizable, allowing you to specify the drivers and routines, which should be included in the software build based on their selection of hardware and software options. (*source: Wikipedia*)

The Board Support Package (BSP), installed separately from PathWave FPGA, comprises of two parts—an FPGA Support Package (FSP) and a Runtime Support Package (RSP). A BSP configuration file contains all the necessary information to identify the configuration, which includes static region implementation, an RSP implementation, build scripts, examples, project templates, and documentation.

- The FSP is that portion of the BSP that allows you to build a bitstream file for the target FPGA. It is consumed by PathWave FPGA to support design creation and sandbox compilation; everything that is performed without the physical hardware. The FPGA Support Package allows you, as a PathWave FPGA user, to create a design targeting your instrument. It includes descriptions of the sandbox interfaces, template PathWave FPGA projects, and files required for compiling a sandbox image from HDL sources. An FSP fulfills the 'design-time requirements' of PathWave FPGA, which covers everything prior to loading a bit image onto the instrument.
- The RSP is that portion of the BSP that allows you to control your target FPGA. It provides a 'C' API that you can use to load design images onto hardware, verify your FPGA bitstream image and perform simple register and streaming accesses to the sandbox. An RSP requires a hardware driver with the following capabilities:
  - Hardware discovery/enumeration
  - Program the FPGA sandbox regions
  - Read and write registers inside the sandbox
  - Read and write to streaming interfaces inside the sandbox (if the sandbox has streaming interfaces)

Both PathWave FPGA software and the BSP work together and cannot be used individually.

The block diagram shown in Figure 14 indicates how the bitstream file is generated using both the PathWave FPGA software and BSP together for any specific supported PXIe module.

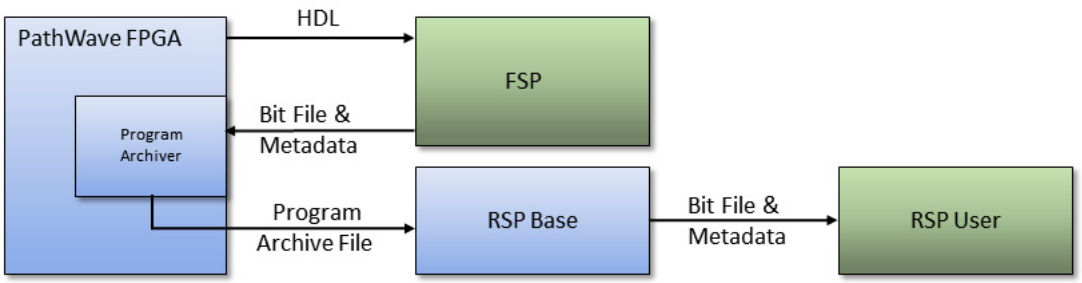


Figure 14 Bitstream file compilation flow using PathWave FPGA and BSP

PathWave FPGA manages bitstream file generation using a scripted flow of the FPGA tools. The scripts for building a specific design are provided by the FSP. The FSP build script can also add metadata, which are delivered to the RSP. The bitstream file and metadata are packaged into a Keysight PathWave FPGA program archive file with the filename extension **.k7z*.

To load a bitstream image on the FPGA, PathWave FPGA supplies the program archive file to the RSP. The RSP unpacks the contents of the program archive file and checks that the image is compatible. The original bitstream file and the metadata are then passed to the instrument-specific portion of the RSP, which eventually configures the FPGA with the bitstream file.

The output from a PathWave FPGA design compilation is saved in a Keysight PathWave FPGA program archive file, with a *k7z* extension. To understand how to generate the k7z file onto a module, see "Generating a k7z file using PathWave FPGA BSP" on page 39.

Hereafter, you may load the k7z file onto the required hardware using the Keysight SD1 3.x SFP software's user interface or the corresponding SD1 API functions. To understand how to load the k7z file onto a module, see "Loading k7z file into modules" on page 48.

For information regarding the Board Support Package for M3102A Digitizer modules, refer to *M3102A Digitizer PathWave FPGA User Guide*.

Section 3.5: Generating a k7z file using PathWave FPGA BSP

After you install BSP for one or more modules, you can proceed with creating a new project to design the sandbox region for the corresponding modules.

Following steps give you a quick glance into creating a new sandbox project using PathWave FPGA 2020 software for FPGA designing followed by generating a bitstream file for one or more supported PXIe modules. This example uses the BSP file for an M3102A module.

You can also perform most of the steps shown below using command line arguments. Refer to the "*Advanced Features*" section of the *PathWave FPGA Customer Documentation*. For information regarding the Board Support Package for M3102A Digitizer modules, refer to *M3102A PXIe Digitizers PathWave FPGA User Guide*.

1 On the main window of the PathWave FPGA 2020 software, click the icon for new project.

A new sandbox project dialog appears.

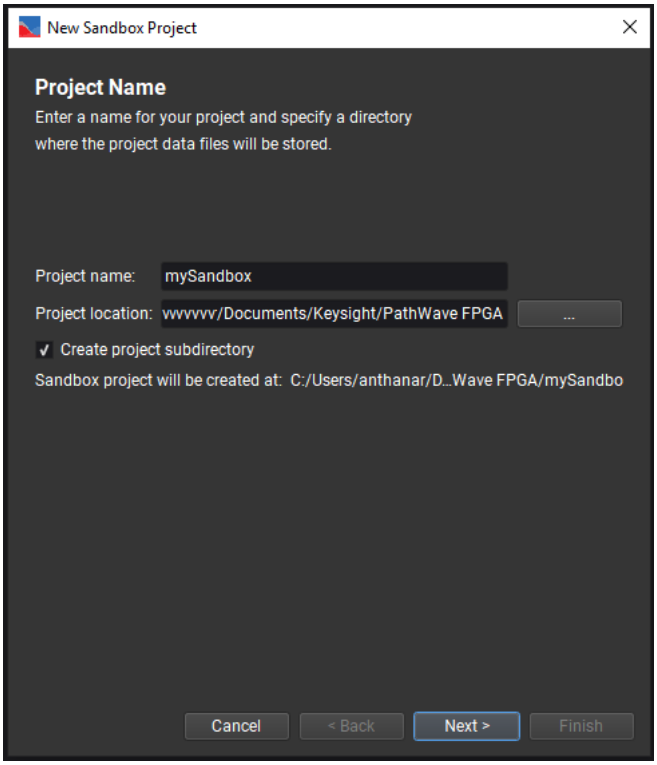


Figure 15 Creating a new Sandbox project

2 Click **Next >**.

3 For the Project Type, choose the BSP corresponding to one of the modules whose FPGA you wish to update.

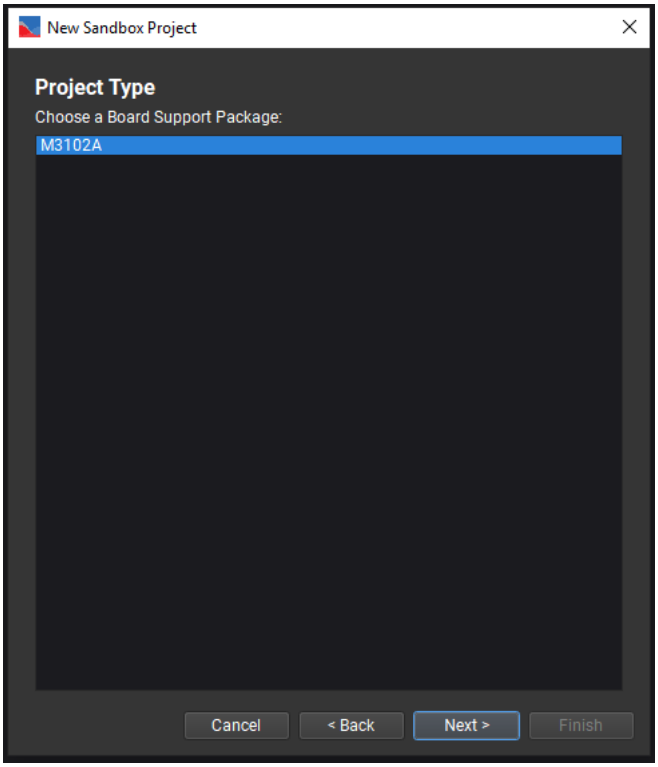


Figure 16 Selecting the module-specific BSP

4 Click **Next >**.

5 For the Project Options, choose the BSP options and Board Configurations you wish to include in your FPGA design logic for the selected module.

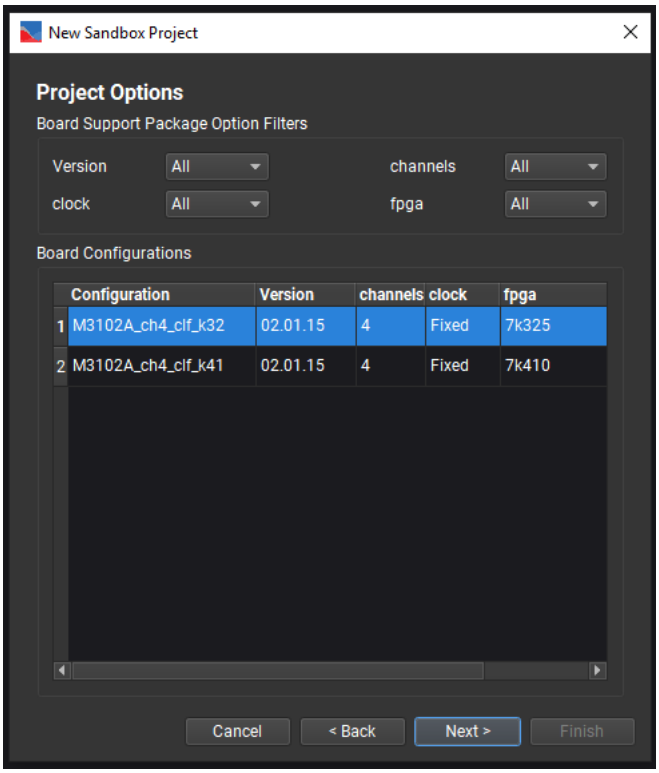


Figure 17 Selecting various options for FPGA design logic

6 Click **Next >**.

7 For the Project Template, either choose the default template offered within the PathWave FPGA design environment or choose blank to start a new custom design logic.

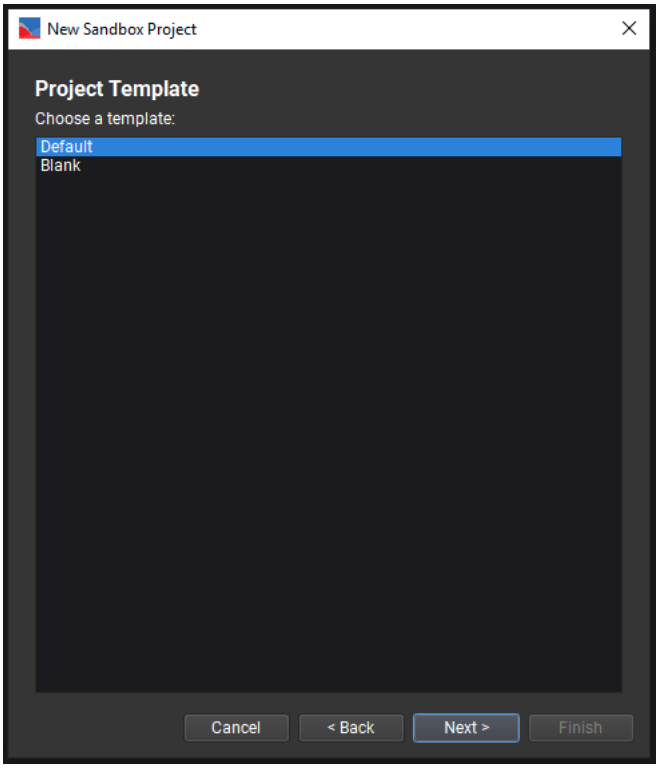


Figure 18 Selecting the PathWave FPGA project template

8 Click **Next >**.

9 Verify all information displayed in the Project Summary. To make any amendments, click **Back**. If the selected options for the corresponding BSP are satisfactory, click **Finish**.

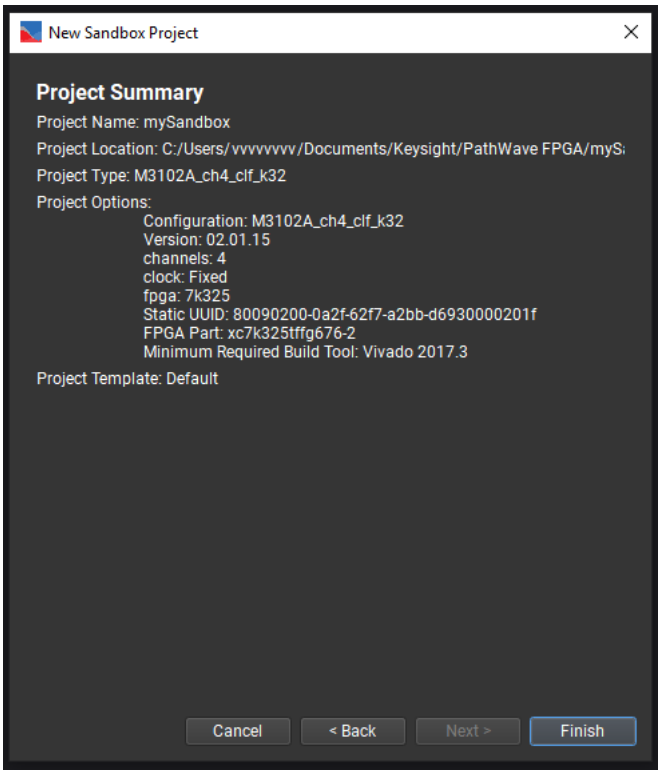


Figure 19 Viewing the project summary information

Various blocks are displayed, which are part of the default template in the PathWave FPGA 2020 software.

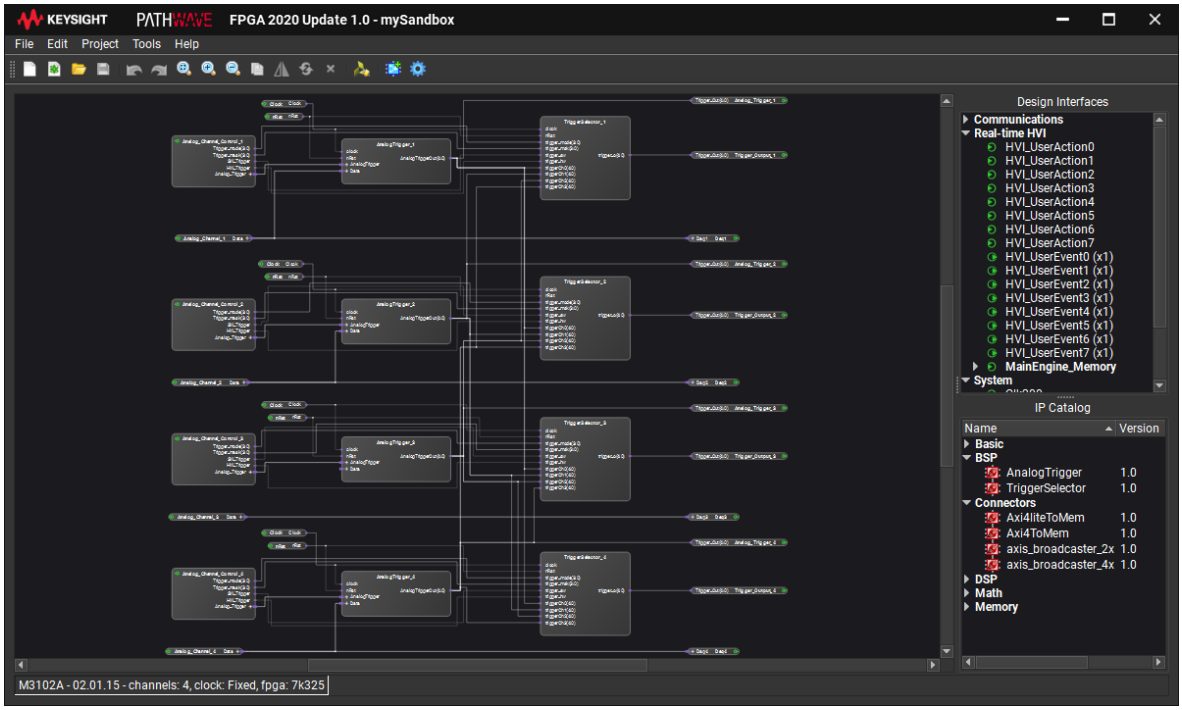


Figure 20 Viewing FPGA design blocks in the default template

10 Customize the configuration as per your requirements using one or more elements from the Design Interfaces and IP Catalog panels.

For more information regarding the configurable region of the M3xxxA FPGA interfaces and BSP IP Repository, refer to the BSP User Guide for the corresponding modules, which can be accessed via **Help** > **BSPs Help** menu options in the PathWave FPGA 2020 software. The guides for the supported modules are:

- *M3102A PXIe Digitizers*
- *M3201A PXIe Arbitrary Waveform Generators*
- *M3202A PXIe Arbitrary Waveform Generators*

11 After you have finished designing your logic, you can proceed with the *k7z* file generation. Click the **Generate Bit File...** icon as shown in Figure 21.

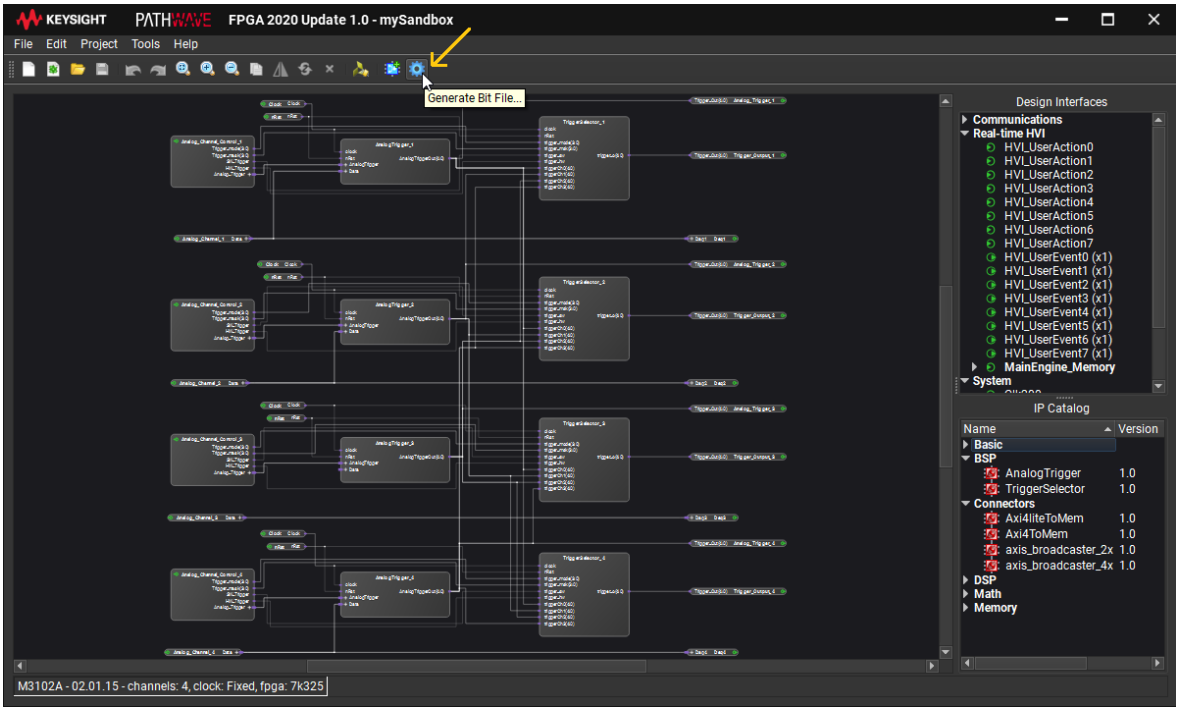
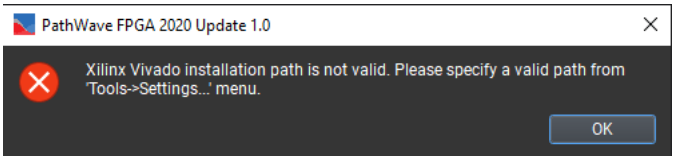


Figure 21 Initiating Bit File generation for the selected module

If you do not have the *Xilinx Vivado Design Suite* on the same machine where PathWave FPGA 2020 software and BSP are installed, the following error is prompted when you click the **Generate Bit File...** icon.



Refer to *PathWave FPGA Customer Documentation* for more information on installing the software and the licenses for the *Xilinx Vivado Design Suite*.

12 On the FPGA Hardware Build window that appears, click **Run**.

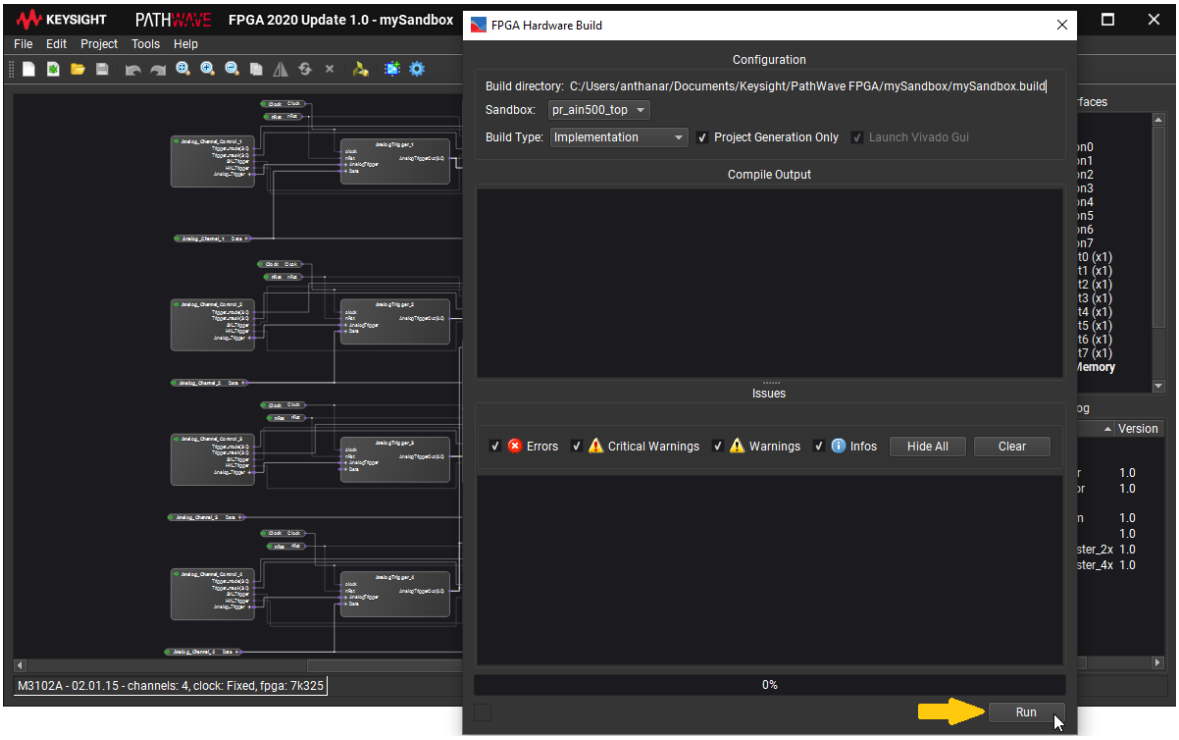


Figure 22 Generating FPGA Hardware Build

Depending on the configuration, the software takes some time before finishing the process of *k7z* file generation.

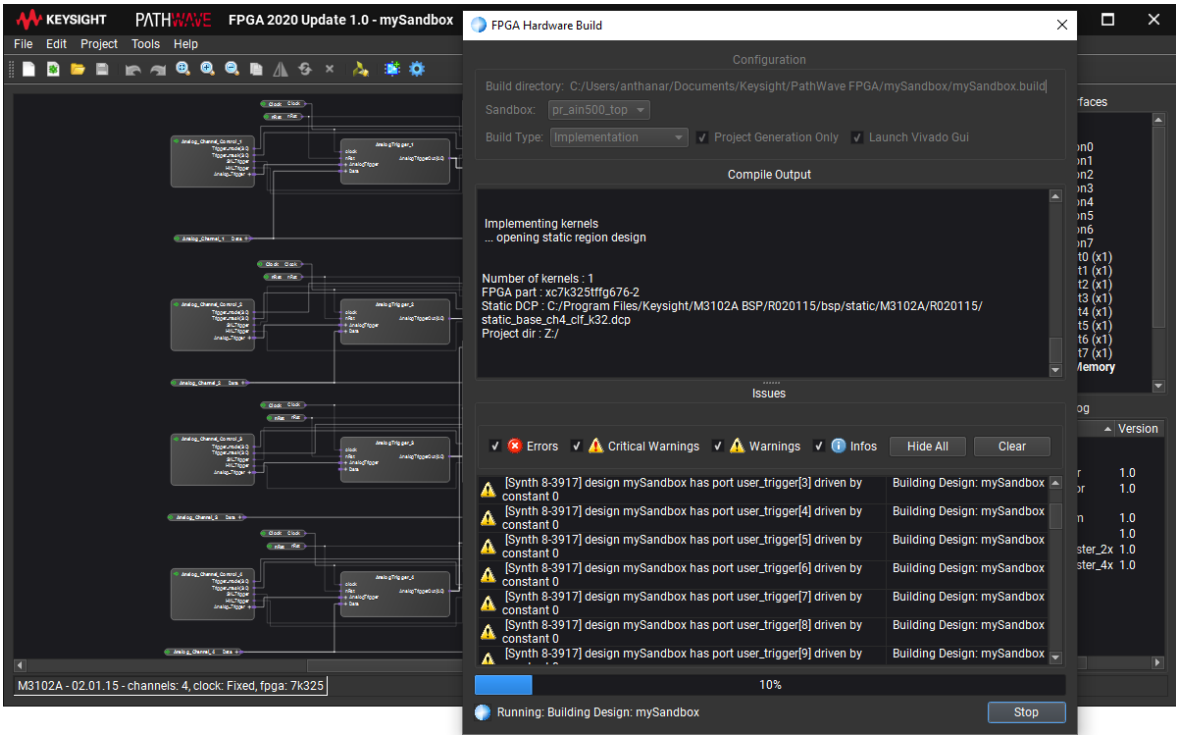


Figure 23 Progress status for the k7z file generation

Once the Bitstream file (which is the *k7z* file) is generated, your custom FPGA logic is ready to be loaded on the selected module using either the SD1 API or the **Load Firmware** feature (under FPGA menu of the Module panel in the SFP software).

Other than configuring and designing your FPGA Logic using the *PathWave FPGA 2020 Update 1.0* software, you can perform one or more of the following operations:

- Build your FPGA Logic
  - Generate the Bit File (covered briefly in this section)
  - Verify the Bit File
- Simulate your FPGA Logic
  - Simulation Testbench Designing
  - Test Bench Address Mapping
- Work with Advanced features
  - Using Command Line Arguments
  - Migrating a design to a new BSP
  - Changing a Submodule Project Target Hardware
  - Debugging in Hardware

For detailed instructions on performing these steps and to understand the features of the PathWave FPGA 2020 software, refer to the PathWave FPGA Help file accessible via the Help menu of the software. For information about the features and various elements along with understanding the workflow associated with the *PathWave FPGA 2020 Update 1.0* software, refer to the "*User Guide*" section of the *PathWave FPGA Customer Documentation*.

Section 3.6: Loading k7z file into modules

After a bitstream file (k7z) is generated using the PathWave FPGA software, you may use either the SD1 3.x software user interface or the SD1 API functions to load the FPGA on the corresponding module and also, verify design compatibility on that module.

3.6.1: Using SD1 SFP user interface to load FPGA

To load the FPGA using SD1 3.x software,

1 From the **Start** menu, launch Keysight SD1 SFP.

2 From the main menu of a specific module's dialog, click **FPGA** > **Load firmware...**.

Figure 24 Accessing FPGA firmware loader option in SD1

3 On the **Firmware Loader** dialog that appears, click the button to browse for the required k7z file for that particular module.

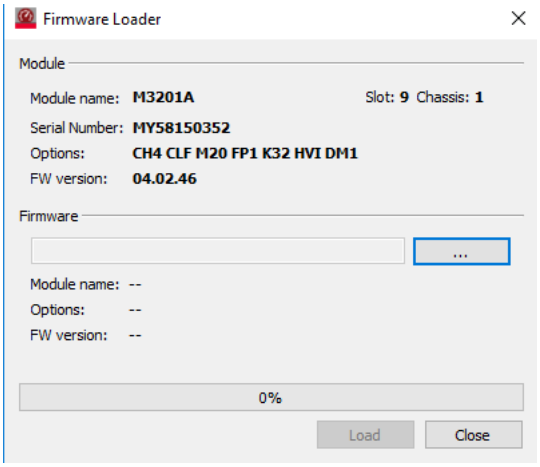


Figure 25 Firmware Loader window for the selected module

4 Click **Load** to initiate the process of FPGA loading.

If the progress bar displays “Loading successful” and “100%”, it indicates that the FPGA is updated as well as the FPGA design logic is compatible on hardware.

3.6.2: Using SD1 API to load FPGA

To load the FPGA using SD1 3.x API,

1 Import system and Keysight SD1 libraries.
2 Open the module with open().
3 Load FPGA sandbox using *.*k7z* file with FPGAload().
4 (Optional) Read/Write into registers using functions defined in SD_SandboxRegister functions.
5 (Optional) Perform FPGA related operations using other functions defined in SD_Module functions (specific to Pathwave FPGA).

Section 3.7: Implementing BSP using SD1 API - Sample Programs

3.7.1: Sample program using Python for read write on sandbox region

```
# ----------------------------
# import required libraries
import sys
sys.path.append('C:\Program Files (x86)\Keysight\SD1\Libraries\Python')
import keysightSD1

# Set details for the connected module
product=''
chassis=1
slot=9

# open a module
module=keysightSD1.SD_Module()
moduleID=module.openWithSlot(product,chassis,slot)
if moduleID &amp;lt; 0:
    print ("Module open error: ",moduleID)
else:
    print("Module is: ",moduleID)

# Loading FPGA sandbox using the *.k7z file
error = module.FPGAload(r'..\myHardwareTest.k7z')
numRegisters = 4

# Get list of Registers from the sandbox
registers = module.FPGAgetSandBoxRegisters(numRegisters)

# Print the register properties in register list
for register in registers:
    print(register.Name);
    print(register.Length);
    print(register.Address);
    print(register.AccessType);

registerName = 'Register_Bank_A'
# Get Sandbox Register with name "Register_Bank_A"
registerA = module.FPGAgetSandBoxRegister(registerName)
```

```
# Write data to Register_Bank_A
error = registerA.writeRegisterInt32(9)
registerNameB = 'Register_Bank_B'

# Get Sandbox Register with name "Register_Bank_B"
registerB = module.FPGAgetSandBoxRegister(registerNameB)

# Write data to Register_Bank_B
error = registerB.writeRegisterInt32(9)

registerNameC = 'Register_Bank_C'
# Get Sandbox Register with name "Register_Bank_C"
sandbox_register_C = module.FPGAgetSandBoxRegister(registerNameC)

# Read data from Register_Bank_B
error = sandbox_register_C.readRegisterInt32()

memoryMap = 'Host_mem_1'
# Get Sandbox memoryMap with name "Host_mem_1"
memory_Map = module.FPGAgetSandBoxRegister(memoryMap)
# Write buffer to memory map
   memory_Map.writeRegisterBuffer(0, [1,2,3, 4, 5, 6],
keysightSD1.SD_AddressingMode.AUTOINCREMENT, keysightSD1.SD_AccessMode.DMA)
# Read buffer from memory map
   c_value = memory_Map.readRegisterBuffer(0, 6, keysightSD1.SD_AddressingMode.AUTOINCREMENT,
keysightSD1.SD_AccessMode.NONDMA)
print(c_value)
module.close()
```

3.7.2: Sample program using .NET for read write on sandbox region

```
class Program
{
    static void Main(string[] args)
    {
    SD_AOU awg = new SD_AOU();
    // Open awg module
    awg.open("", 1, 9);
    //Loading FPGA sandbox using .K7z file
    int error = awg.FPGAload(@"..\myHardwareTest.k7z");
    //Get Sandbox register list
    List<SD_SandBoxRegister> registers = awg.FPGAGetSandBoxRegisters(4);
    //Print register properties
    foreach(SD_SandBoxRegister register in registers)
    {
              Console.WriteLine(register.Name);
       Console.WriteLine(register.Address);
       Console.WriteLine(register.Length);
       Console.WriteLine(register.AccessType);
    }
    int[] buffer = { 5, 4, 3, 2 };
    int[] registerBuffer = new int[4];
    //Write data buffer to register with index 0
    registers[0].WriteRegisterBuffer(0, buffer, SD_AddressingMode.AUTOINCREMENT,
SD_AccessMode.DMA);
    //Read buffer from register with index 0
    registers[0].ReadRegisterBuffer(0, registerBuffer, SD_AddressingMode.AUTOINCREMENT,
SD_AccessMode.DMA);
    SD_SandBoxRegister registerA= awg.FPGAGetSandBoxRegisterByName("Register_Bank_A");
    //Write data to register Register_Bank_A
    registerA.WriteRegisterInt32(2);
    int registerBValue = registerA.ReadRegisterInt32();
    SD_SandBoxRegister hostMem = awg.FPGAGetSandBoxRegisterByName("Host_mem_1");
    int indexOffset = 2;
    //Write data buffer to Host_mem_1 with indexOffset 2
    hostMem.WriteRegisterBuffer(indexOffset, buffer, SD_AddressingMode.AUTOINCREMENT,
SD_AccessMode.NONDMA);
    //Read data buffer to Host_mem_1 with indexOffset 2
    hostMem.ReadRegisterBuffer(indexOffset, registerBuffer, SD_AddressingMode.AUTOINCREMENT,
SD_AccessMode.DMA);
    }
}
```

4. Using Keysight SD1 3.x SFP Software

This chapter describes how to use Keysight SD1 SFP software.

Keysight M3201A/M3202A PXIe AWGs, M3100A/M3102A PXIe Digitizers, and M3300A/M3302A PXIe AWG/Digitizer Combos can be operated as classical bench-top instruments using Keysight SD1 SFP software; no programming is required.

Keysight SD1 SFP Software provides a fast and intuitive way of operating Keysight M3201A/M3202A PXIe AWGs, M3100A/M3102A PXIe Digitizers, and M3300A/M3302A PXIe AWG/Digitizer Combos.

Based on your preference, you may also perform various operations using the programming library provided within Keysight SD1 Core API. An API function is available for almost every control within the user interface. You can build and run custom scripts to carry out various operations / workflow, which can be performed by the SD1 SFP user interface.

The sections in this chapter cover the functionality of each SD1 SFP feature available in its user interface, along with describing the workflow required to perform some specific operations. Wherever applicable, the API function corresponding to the feature/control has been specified for reference.

To know about the programming libraries and the available API functions, see Chapter 5, “Using Keysight SD1 API Command Reference” followed by Chapter 6, “Using SD1 API functions in sample programs”.

Section 4.1: Installing the Keysight SD1 3.x Software Package

The Keysight SD1 3.x Software Package includes:

- Keysight SD1 3.x SFP Software
- KF9000A PathWave FPGA Programming Environment (commonly known as PathWave FPGA software)—required for FPGA logic designing
- PathWave FPGA Board Support Package (BSP)—required for FPGA logic designing
- KS2201A PathWave Test Sync Executive Software—required for integration with HVI technology

Prior to installing the software package components listed above, the following software must be installed on your machine that runs a Windows 10 (64-bit) Operating System.

Prerequisites

- Keysight IO Libraries Suite 2018 (version 18.0 or later)
- Microsoft .NET 3.5 or later
- Any API interface
  - Python 64-bit version 3.7.x or later
  - Any C#/.NET Compiler
  - Any C/C++ Compiler
- *Xilinx Vivado Design Suite* (required with PathWave FPGA software)
- License options
  - Option -FP1 (to enable FPGA programming through PathWave FPGA BSP)
  - Option -HV1 (to implement HVI technology using the PathWave Test Sync Executive software)

Refer to the *Keysight SD1 3.x Software Startup Guide* for download links and installation instructions.

Section 4.2: Launching the Keysight SD1 SFP software

After the Keysight SD1 3.x SFP software is installed, click **Start** > **Keysight** > **Keysight SD1 SFP**.

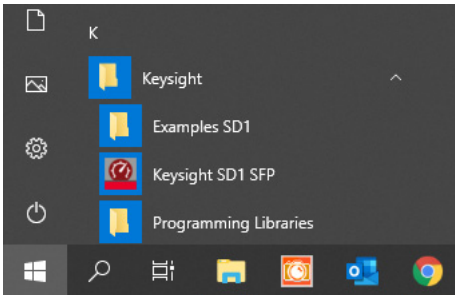


Figure 26 Launching SD1 SFP software from the Start menu

The Keysight SD1 3.x SFP software window is displayed, as shown in Figure 27 and Figure 28.

When SD1 SFP is launched, it identifies all Keysight PXIe hardware modules that are connected to the embedded controller or desktop computer and opens a corresponding soft front panel for each piece of hardware. As shown in Figure 27, the soft front panel for the M3201A AWG and M3302A (AWG & Digitizer) Combo are displayed by default, which indicates these modules are connected.

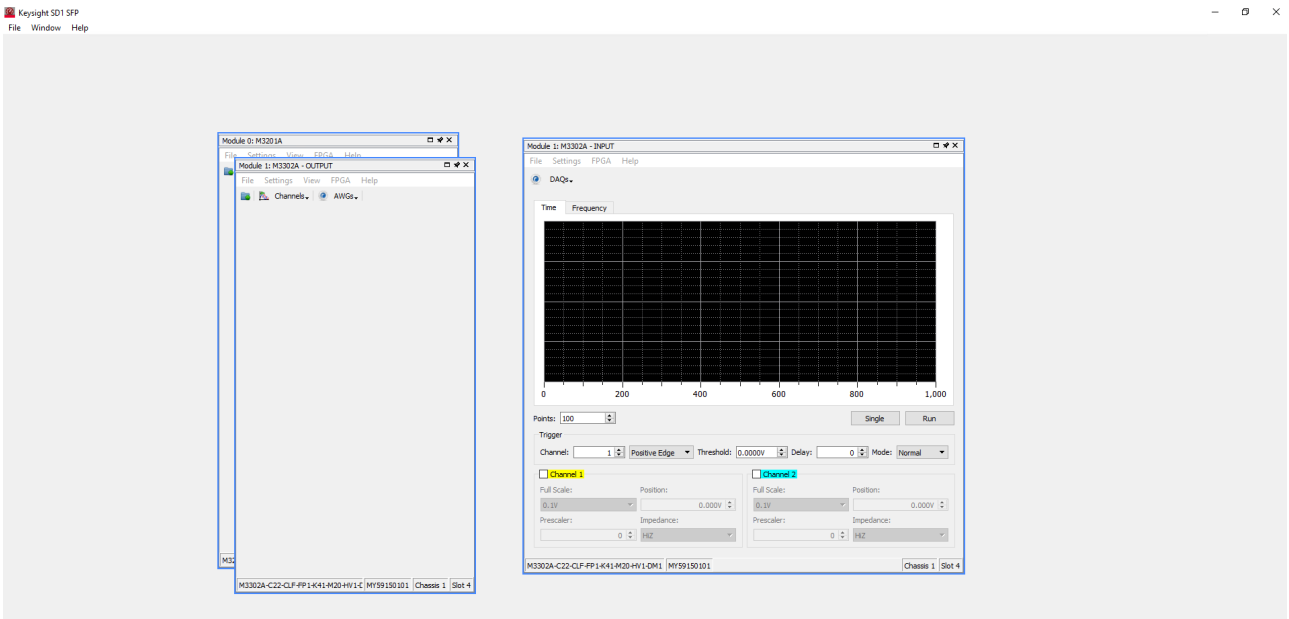


Figure 27 SD1 SFP software window (online mode with M3201A and M3302A cards connected)

If one or more modules are not inserted into the chassis (or disconnected), the soft front panel for each of such modules is displayed as “Demo module” on the Keysight SD1 3.x SFP software. See Figure 28.

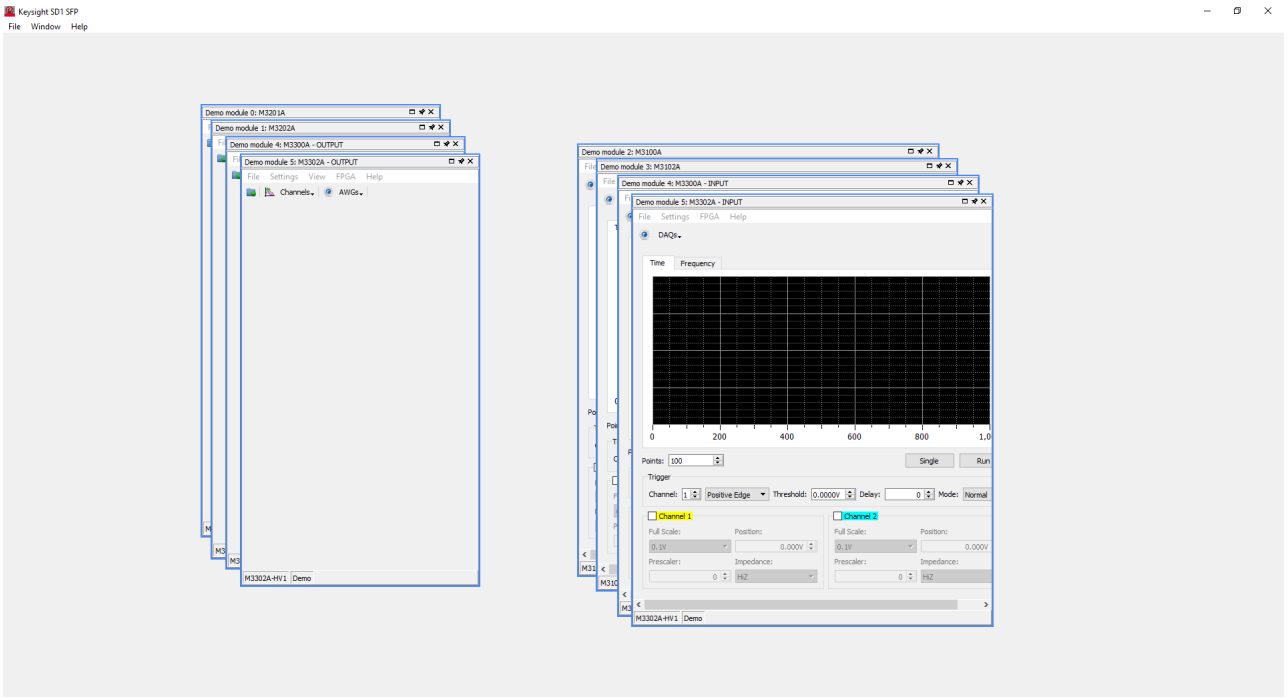


Figure 28 SD1 SFP software (offline mode without any card connected)

Section 4.3: Understanding the SD1 SFP features & controls

As the name indicates, the Soft Front Panel (SFP) provides you controls so that you may configure settings associated with the front panel features (namely, Channel, Clock and Trigger) on the connected PXIe AWG, Digitizer and Combo cards. Considering the basic functionality of an AWG and a Digitizer, the appearance and functions on each SFP window for AWG modules are different from that for the Digitizer modules.

The SFP windows that are specific to AWG modules correspond to:

- M3202A AWG 1G
- M3201A AWG 500
- AWG front panel of the M3302A AWG 500 DIG 500 Combo
- AWG front panel of the M3300A AWG 500 DIG 100 Combo

The SFP windows that are specific to Digitizer modules correspond to:

- M3102A DIG 500
- M3100A DIG 100
- DIG front panel of the M3302A AWG 500 DIG 500 Combo
- DIG front panel of the M3300A AWG 500 DIG 100 Combo

The SD1 SFP user interface, primarily, consists of three types of windows:

1 Main window—includes controls for displaying each module window and hardware configuration.
2 AWG Module SFP window—includes front panel controls for performing operations pertaining to AWGs.
3 Digitizer Module SFP window—includes front panel controls for performing operations pertaining to Digitizers.

In the offline (Demo) mode, even though the front panel controls appear to be available and even configurable, they are meant for demonstrative purposes only. On the other hand, when connected, each module has its respective SFP window and all controls and functions are available. The following sections describe the controls that appear on each window.

4.3.1: Understanding main window features and controls

Figure shows the SD1 SFP main window without any overlapping SFP window.

Figure 29 SD1 SFP main window

The controls under each menu item are further described in the order displayed above.

1. File

Figure 30 Controls in the File menu

- Exit—Click **File** > **Exit** to close the SD1 SFP main window along with any SFP windows that may be displayed.

2. Window

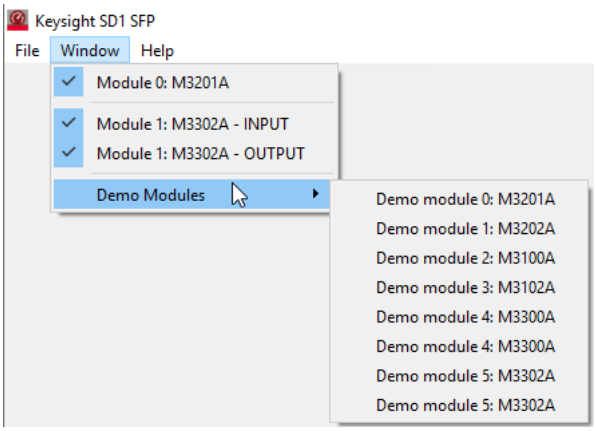


Figure 31 Controls in the Window menu

- The menu under **Window** lists the names assigned to the connected modules. By default, a tick mark appears against each entry, indicating that the corresponding SFP window is active. Any name that does not appear indicates that the module is not connected. An offline instance for each module is available in the **Demo Modules** sub-menu.
  1. To close any SFP window that is open, clear the tick from the corresponding entry.
  2. To open an SFP window for a module that is connected, click the corresponding entry.
- Demo Modules—The drop-down sub-menu under **Window** > **Demo Modules** lists the names of each module. You may perform the steps described above, but the corresponding SFP windows that appear are meant for demonstration purpose only.

3. Help

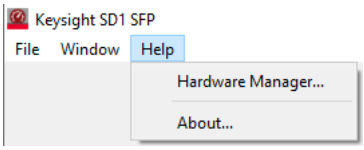


Figure 32 Controls in the Help menu

- Hardware Manager...—Click **Help** > **Hardware Manager...** to launch the **Hardware Manager** window. See “Understanding features in Hardware Manager” on page 60 for more information on features available in the Hardware Manager window.

- About—Click **Help** > **About** to view the About Keysight SD1 SFP window. This window conveys the SD1 SFP software's version installed on your machine.

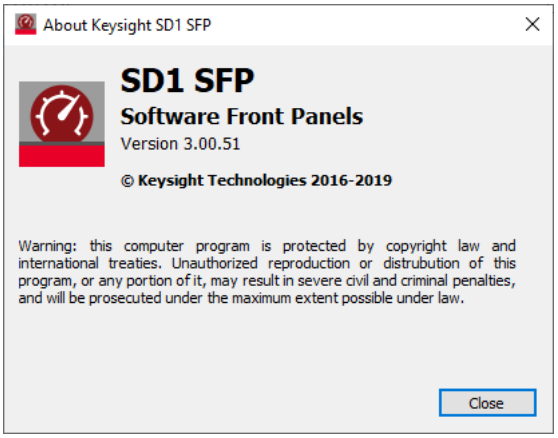


Figure 33 About Keysight SD1 SFP window

4.3.2: Understanding features in Hardware Manager

The features in this window are described below in the order shown in Figure 34.

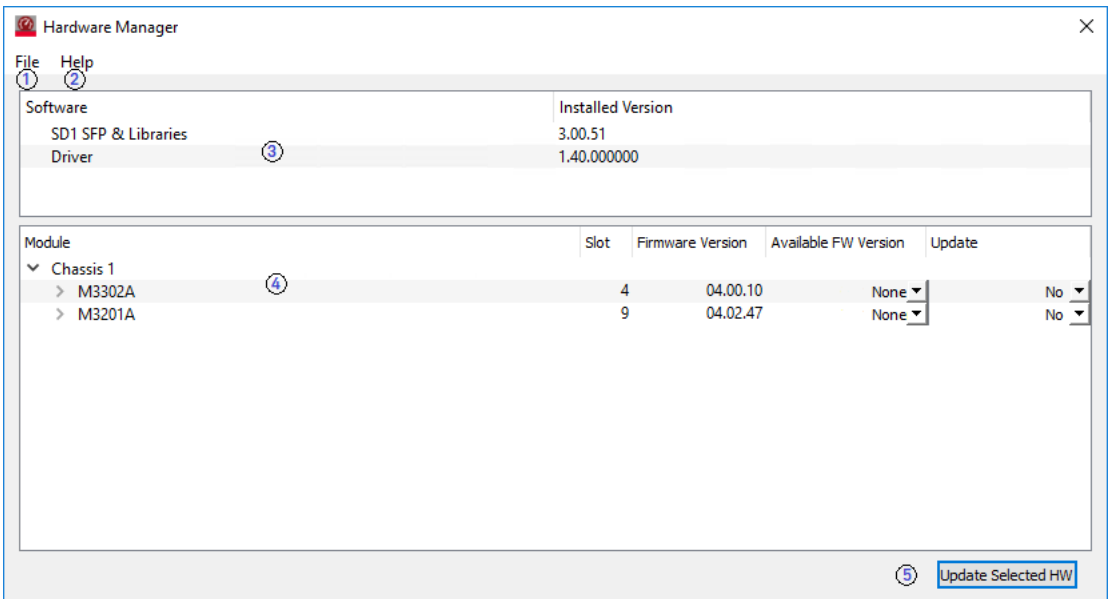


Figure 34 Hardware Manager window

1 **File**—The **File** menu has two items, which are:

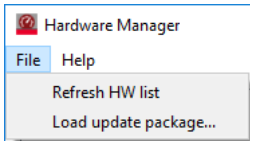


Figure 35 Controls in the File menu for Hardware Manager

- **Refresh HW list**—This feature requires an active network connection to fetch information from the database. Click **File** > **Refresh HW list** so that the SD1 SFP software checks the Firmware database for any new Firmware version for each M3xxxA module that is displayed in the **Modules** area below. If an updated firmware version is found corresponding to one or more modules, the firmware version is updated/refreshed in the **Available FW Version** list. Performing the **Refresh HW list** is recommended prior to proceeding with Firmware update using the SD1 SFP software on one or more connected modules.
- **Load update package...**—In case your machine is offline, that is, does not have an active network connection, this feature is useful to load an updated and pre-saved firmware package so that you can still proceed with firmware updates on one or more connected M3xxxA modules. Click **File** > **Load update package...** to access the **Load SDM package...** window, as shown in Figure 36. Navigate to the folder where the firmware update file is saved. Open the required SDM package file with extension **.sdpkg*. You may proceed with firmware update on the module which the SDM package file corresponds to.

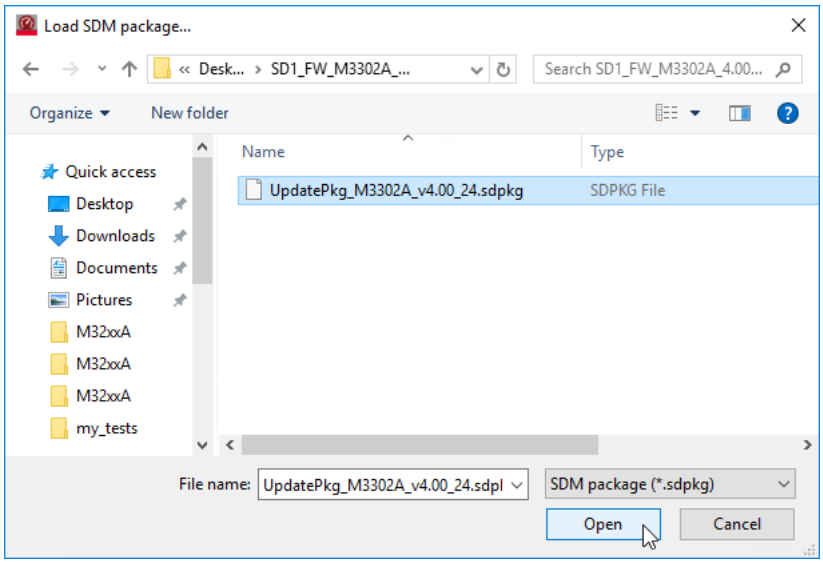


Figure 36 Accessing firmware update file from Load SDM package window

2 **Help**—The **Help** > **Database Version** to view the version of the Firmware update file Database server.

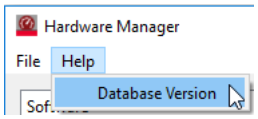


Figure 37 Control in the Help menu for Hardware Manager

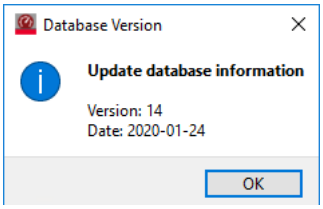


Figure 38 Database Version window

3 **Software**—This area displays the installed version of the Keysight SD1 SFP software and programming libraries along with that of the compatible driver.

NOTE **Keysight SD1 SFP 3.x software does not recognize cards that have Firmware version less than 4.0 (for M32xxA/M33xxA modules) or less than 2.0 (for M31xxA modules).**

4 **Module**—This area consists of the following information:

- Chassis—the Chassis number where the M3xxxA cards are inserted into. Related API function: "getChassis()".
- <M3xxxA>—the model number for one or more modules that are inserted into the specific Chassis number. The following information is displayed for each module. Related API function: "getType()".

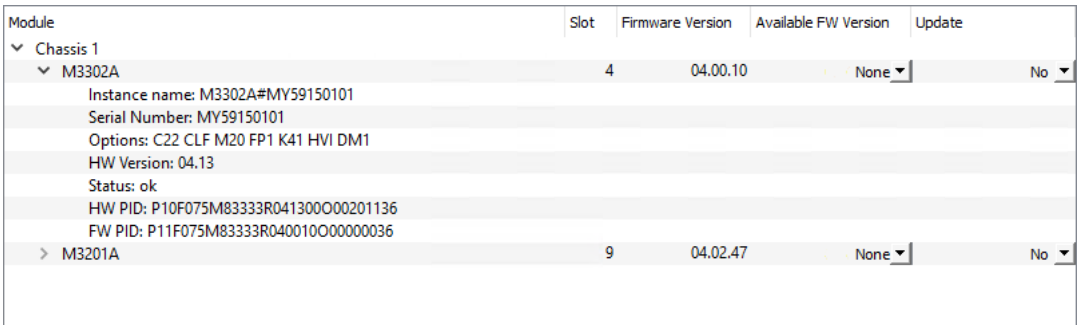


Figure 39 Information pertaining to each module in the Hardware Manager window

- Instance name—Product Name assigned to the software instance when the SD1 SFP detects the connected module. Format is <M3xxxA>#<serial-number>. Related API function: "getProductName()".
- Serial Number—Serial number of the connected module. Related API function: "getSerialNumber()".
- Options—Hardware license options that are enabled on this module. Related API function: "getOptions()".
- HW Version—Hardware version of the module. Related API function: "getHardwareVersion()".
- Status—Indicates the current status of the module. 'ok' indicates devices is working properly.
- HW PID—The Product ID number of the Hardware. Indicates which physical options are available on the product.
- FW PID—The Product ID number of the Firmware. Indicates which firmware elements are available on the product.

- Slot—the slot number in the Chassis where the M3xxxA module is inserted into. Related API function: "getSlot()".
- Firmware Version—the firmware version currently installed on each M3xxxA module. Related API function: "getFirmwareVersion()".
- Available FW Version—'None' if only the modules are connected but the machine does not have an active network connection; else, the latest firmware version number that is available on the Database server.
- Update—By default, the option is 'No' for each module. Change to 'Yes' for one or more modules where you wish to perform a firmware update.

5 **Update Selected HW**—Click the **Update Selected HW** button to proceed with firmware update on your selected module, where you have set the **Update** flag to **Yes**. The SD1 SFP software then displays a confirmation prompt asking you to verify if you wish to proceed with the firmware update using the version listed in the Available FW Version list for the selected module on the respective Chassis & Slot numbers. For firmware update instructions, refer to *SD1 3.x Software Startup Guide*.

Section 4.4: Understanding Digitizer SFP features & controls

For all those modules that are connected to the Chassis, the respective soft front panel for the M3100A/M3102A PXIe Digitizer and that for the Digitizer part of the M3300A/M3302A Combo appear automatically when SD1 SFP is launched. By default, the SFP window for the Digitizer appears, as shown for M3302A module in Figure 40.

NOTE **The controls are common across all Digitizer SFP windows. However, the number of DAQs / Channels that appear depends on the module type. M31xxA modules and the M3300A Combos support 4 DAQs/Channels whereas the M3302A Combos support 2 DAQs/Channels.**

On this window,

- the default name, which is assigned by the SD1 SFP user interface, is displayed on the bar at the top.
- the hardware options enabled on the module followed by the serial number, Chassis number and Slot number are displayed on the bar at the bottom.
- the Channel controls are displayed on the main window whereas the Data Acquisition (DAQ) controls are included within the main menu items, which are explained in the following section.
- The SFP is divided into **Time** domain and **Frequency** domain. The main menu is common to both domains.

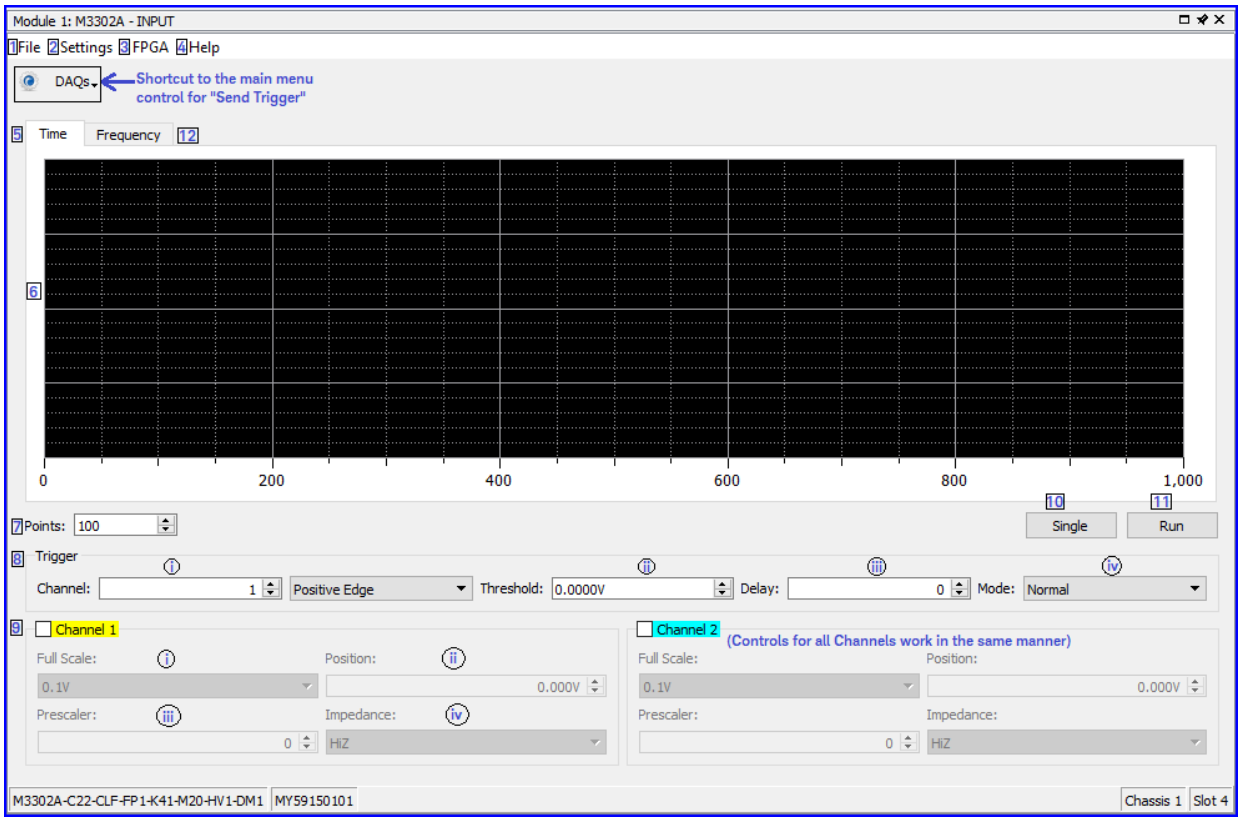


Figure 40 SFP (Time domain) for the connected M3302A after SD1 SFP software is launched

4.4.1: Understanding DIG SFP main menu features & controls

The controls under each menu item are further described in the order displayed in Figure 40. Images in this section pertain mainly to M3302A DIG module's SFP only, but are applicable across all DIG module SFP windows.

1. File

Note that the functionality of this feature is same as that for AWG SFP. The images shown here are specific to M3201A SFP; however, the functionality is the same across all SFP windows in the SD1 3.x software.

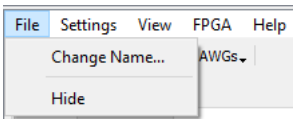


Figure 41 Controls in the File menu of the SFP

- Change Name...—Click **File** > **Change Name...** to open the **Change name module** dialog box.

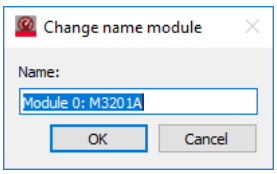


Figure 42 Change name module dialog box

  - In the **Name:** text field, enter an alternate name of your choice.
  - Click **OK**.

    The new name appears on the bar at the top of the corresponding module's SFP window.
- Hide—Click **File** > **Hide** to close the module's SFP window instance.
- To show/open the same window again, click **Window** > <*module name*> from the main menu of the Keysight SD1 SFP software. See description for "Window" in "Understanding main window features and controls" on page 58.

2. Settings

Here, the controls for DAQ are available in the upper part.

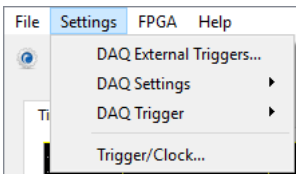


Figure 43 Controls in the Settings menu of the DIG SFP

- DAQ External Triggers...—Click **Settings** > **DAQ External Triggers...** to open the **Configure DAQ Triggers** dialog box. See "Setting the Configure DAQ Triggers dialog" on page 70 for more description.
  - Related API function: "DAQtriggerExternalConfig()"

- DAQ Settings—Click **Settings** > **DAQ Settings** > **DAQ n** to open the **DAQ n Settings** dialog box. See "Setting the DAQ Settings dialog" on page 71 for more description.
- DAQ Trigger—Click **Settings** > **DAQ Trigger** > **Send Trigger** to send a trigger to the selected DAQs in the same sub-menu. By default, all DAQs are selected.
  - Related API functions: "DAQtrigger()", "DAQtriggerMultiple()", "DAQtriggerConfig()"

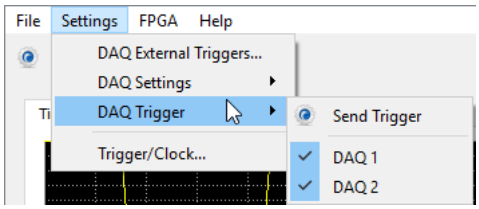


Figure 44 Controls in DAQ Trigger sub-menu

- Click one or more **DAQ n** (where, n = DAQ number) entries to clear the tick mark; thereby, refraining them from receiving the trigger when you click **Send Trigger**.
- Alternatively, you may use the shortcut controls to send trigger to DAQs.

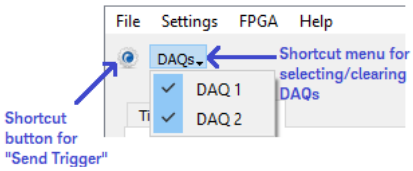


Figure 45 Shortcut controls for sending trigger to DAQs

- Trigger/Clock...—Click **Settings** > **Trigger/Clock...** to open the **Trigger / Clock Settings** dialog box. While the DAQ receives the data on trigger, the functionality of the user interface is the same as that for **Trigger / Clock Settings** dialog in AWG SFP modules. See Setting the Trigger / Clock Settings dialog for more information.
  - Related API functions: "triggerIOconfig()", "clockIOconfig()"

3. FPGA

This feature functions in the same manner across all SFPs.

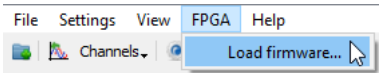


Figure 46 Control in the FPGA menu of the SFP

- Load firmware...—Click **FPGA** > **Load firmware...** to open the **Firmware Loader** dialog box, where you can load the bitstream file (**.k7z*) onto the FPGA sandbox region of the corresponding module. For more information on how to load the bitstream file onto the FPGA sandbox region, see "Using SD1 SFP user interface to load FPGA" on page 48.
  - Related API function: "FPGAload()"

4. Help

This feature functions in the same manner across all SFPs.

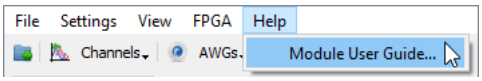


Figure 47 Control in the Help menu of the SFP

- Module User Guide...—Click **Help** > **Module User Guide...** to view the documentation corresponding to Digitizer/Combo modules in Online Help format. For more information, see "Accessing Online Help for SD1 3.x software" on page 182.

4.4.2: Other DIG SFP features & controls

5. Time

The **Time** domain, whose functionality is like that of Oscilloscope, is displayed by default. The controls in this tab are explained further.

6. Display

The display is similar to that of the Oscilloscope and plots the input waveform on the time scale. You can alter the way the waveform is plotted by using the Channel controls.

7. Points

Indicates the number of sample points to be acquired in a Data Acquisition. Make sure that the input integer is an even number.

8. Trigger

i **Channel**—Enter the number for the corresponding Channel n (where, n = Channel number) where the trigger is checked. Note that each Channel is color-coded, so that the applied trigger inherit the same color as shown on the selected Channel for easy identification, when triggers on multiple Channels are applied.

- **Positive Edge**—(default) indicates that the Trigger is generated when the input signal is rising and crosses the configured threshold value.
- **Negative Edge**—indicates that the Trigger is generated when the input signal is falling and crosses the configured threshold value.
- **Both Edges**—indicates that the Trigger is generated when the input signal crosses the configured threshold value, irrespective of whether it is rising or falling.

ii **Threshold**—indicates the threshold voltage level (in units of Volts), which the input signal must attain for the Trigger to be generated.

iii **Delay**—indicates the time delay (in units of seconds) on the input signal, after which the Trigger is generated.

iv **Mode**—select one of the following drop-down options:

- **Normal**—(default) Triggering is generated when the conditions defined for Edge, Threshold and Delay levels are met.
- **Auto**—Triggering is generated irrespective of the conditions defined for Edge, Threshold and Delay levels.

- **Slave**—Triggering is generated with one of the triggering source behaving as slave.

9. Channel

Select the check box for the corresponding Channel n (where, n = Channel number) where the input sampling points can be plotted as a waveform. Note that each Channel is color-coded, so that the waveforms inherit the same color as shown on the selected Channel for easy identification, when multiple waveforms are displayed.

i **Full Scale**—configures scalability of the input waveform on the Digitizer SFP display.

ii **Position**—configures positioning of the input waveform on the Digitizer SFP display.

iii **Prescaler**—configures the value by which the effective sampling rate must be reduced.

iv **Impedance**—configures the impedance value for AC coupling and DC coupling on the input waveform.

10. Single

- Click the **Single** button to capture a single acquisition of the waveform.

11. Run

- Click the **Run** button to capture a continuous acquisition of the waveform.

12. Frequency

The **Frequency** domain, whose functionality is like that of a spectrum analyzer, has common Channel controls described so far, but for the additional **Window** feature, which is available in this domain only, as shown in Figure 48.

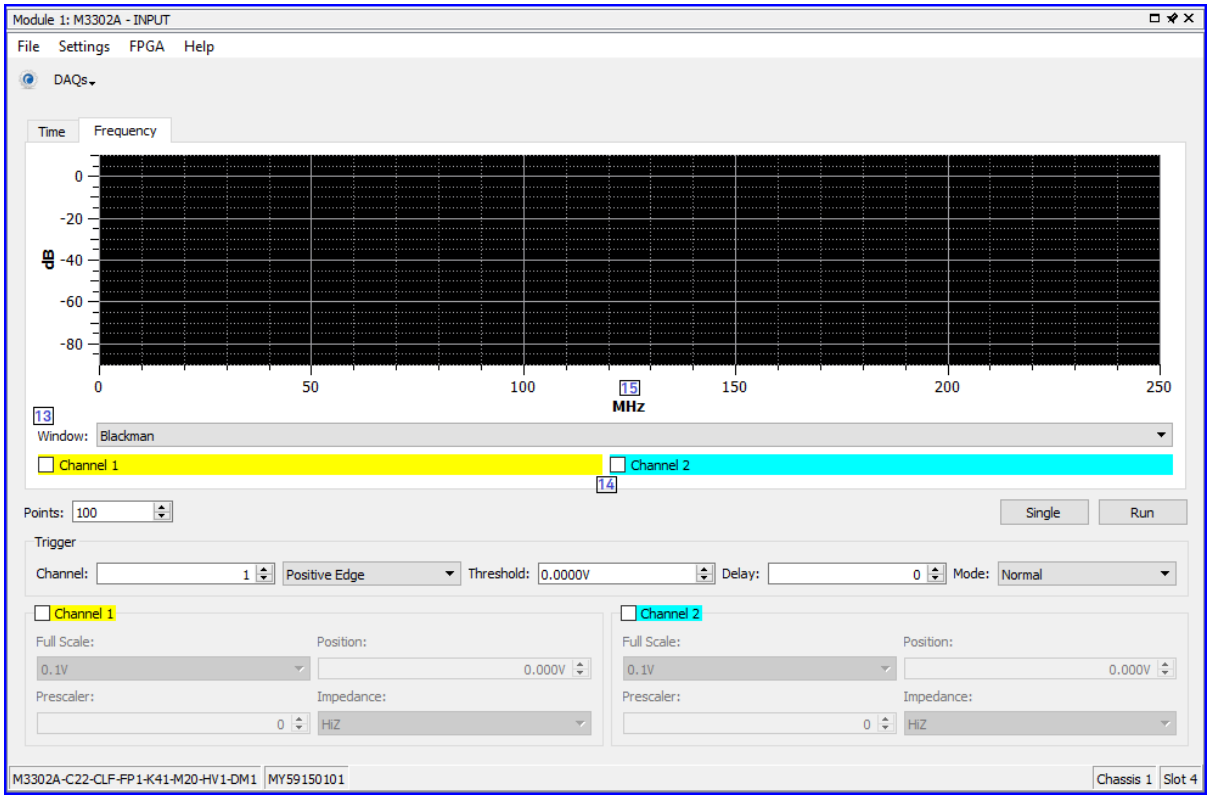


Figure 48 SFP (Frequency domain) for the connected M3302A

13. Window

Window is a mathematical function that is zero-valued outside of some chosen interval and it is used in applications including spectral analysis.

- Select one of the following window functions from the drop-down options to apply on the Digitizer signals:
  - **Rectangular**—Simplest B-spine window
  - **Bartlett**—Hybrid window
  - **Hanning**—Side-lobes roll off about 18 dB per octave
  - **Hamming**—Optimized to minimize the maximum nearest side lobe
  - **Blackman**—(default) Higher-order generalized cosine windows for applications that require windowing by the convolution in the frequency-domain

14. Channel n

(n represents the Channel number)

- Select the check box for the corresponding Channel number on the module, where the selected **Window** function must be applied.

15. Display

The display is similar to that of a Spectrum Analyzer and plots the input waveform on the time scale. You can alter the way the waveform is plotted by using the Channel controls, described earlier in this section.

4.4.3: Configuring DAQ and Trigger/Clock Setting dialogs

1. Setting the Configure DAQ Triggers dialog

If you are using an external trigger source to send trigger to one or more DAQs, you must configure the **Configure DAQ Triggers** dialog box.

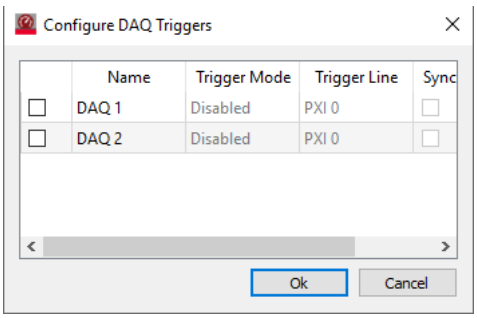


Figure 49 Configure DAQ Triggers dialog box

1 Select one or more rows to enable external triggering for DAQ n, where n = DAQ number.

2 **Trigger Mode** indicates the mode when the trigger signal must be sent. From the **Trigger Mode** drop-down options, select one of the following options:

- **Disabled**—(default) indicates that external triggering is disabled for the selected DAQ.
- **Active High**—indicates that the trigger signal must be sent when active high voltage on the signal is attained.

- **Active Low**—indicates that the trigger signal must be sent when active low voltage on the signal is attained.
- **Rising Edge**—indicates that the trigger signal must be sent when the rising edge on the signal is attained.
- **Falling Edge**—indicates that the trigger signal must be sent when the falling edge on the signal is attained.

3 **Trigger Line** indicates the line medium to be used for the trigger signal to be sent. This field is enabled for all **Trigger Mode** options except for **Disabled**. From the **Trigger Line** drop-down options, select one of the following options:

- **Extern 1** / **Extern 2**—indicates that line 1 / line 2 from the external triggering source is used to send the trigger signal.
- **PXI n** (where, n = 0 to 7)—indicates that lines 0 to 7 from the PXI backplane trigger source can be used to send the trigger signal. PXI0 is the default option.

4 **Sync**—By default, the setting to synchronize the DAQ with the trigger source is disabled. You may select the check box to enable synchronization.

5 Click **OK** to save any changes and return to the DIG module window.

2. Setting the DAQ Settings dialog

To indicate the triggering mode on each DAQ, you must configure the **DAQ n settings** dialog box (where, n = DAQ number). Also, see “DAQ Trigger” on page 17 for more information.

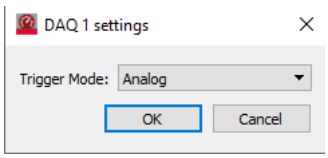


Figure 50 Configure DAQ Setting dialog box

- **Trigger Mode**—Select one of the following drop-down options:
  - **Auto**—The acquisition is triggered immediately after the DAQ has started, irrespective of the triggering source.
  - **Analog**—(default) The acquisition is triggered using an analog triggering source. The DAQ waits for an external analog trigger (applicable only for products with analog inputs).
  - **SW/HVI**—The acquisition is triggered using any triggering software or using an HVI function. In either case, the Digitizer must be connected and operational.
  - **External**—The acquisition is triggered using an external digital triggering source.
- Click **OK** to save any changes and return to the Digitizer module window.

3. Setting the Trigger / Clock Settings dialog

- While the DAQ acquires data on trigger, the functionality of the user interface is the same as that for **Trigger / Clock Settings** dialog in AWG SFP modules. If you are using an External Trigger / Clock source, you can control the point when the trigger must be applied. Use the **Trigger / Clock Settings** dialog to perform this action.

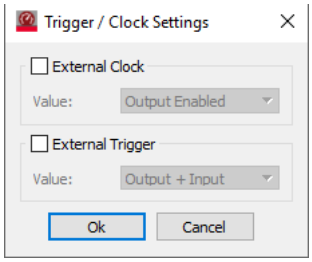


Figure 51 Trigger / Clock Settings dialog box

1 Select the check box for either **External Clock**, **External Trigger**, or both to indicate which external source is being used.

2 **External Clock**—From the **Value** drop-down options, select:

- **Output Disabled**—indicates that the signal from the external clock source is not included in the AWG output.
- **Output Enabled**—(default) indicates that the signal from the external clock source is included in the AWG output.

3 **External Trigger**—From the **Value** drop-down options, select:

- **Input**—Indicates that the signal from the external trigger source is used only to trigger the input waveform at the selected DIG Channels.
- **Output + Input**—(default) Indicates that the signal from the external trigger source is used to trigger both the DIG input and DAQ waveforms.

4 Click **OK** to save any changes and return to the DIG module window.

5. Using Keysight SD1 API Command Reference

Programs can run on an embedded controller or desktop computer and be controlled with Keysight SD1 Programming Libraries. Keysight supplies a comprehensive set of highly optimized software instructions that controls off-the-shelf functionalities of Keysight hardware. These software instructions are compiled into the Keysight SD1 Programming Libraries. The use of customizable software to create user-defined control, test and measurement systems is commonly referred as Virtual Instrumentation. In Keysight documentation, the concept of a Virtual Instrument (or VI) describes user software that uses programming libraries and is executed by a computer.

Section 5.1: Keysight Supplied Native Programming Libraries

Keysight provides ready-to-use native programming libraries for a comprehensive set of programming languages, such as C, Visual Studio (C#, VB), Python, etc., ensuring full software compatibility and seamless multi-vendor integration. Ready-to-use native libraries are supplied for the following programming languages and compilers:

Table 17 List of programming languages and compilers

Language	Compiler	Library	Files
C	Microsoft Visual Studio .NET	.NET Library	*.dll
	MinGW (Qt)	C Library	*.h, *.lib
	Any C compiler	C Library	*.h, *.lib
C++	Any C++ compiler	Header only	*.h
C#	Microsoft Visual Studio .NET	.NET Library	*.dll
Python	Any Python compiler	Python Library	*.py
Basic	Microsoft Visual Studio .NET	.NET Library	*.dll

Section 5.2: Support for Other Programming Languages

Keysight provides dynamic libraries, such as DLLs, that can be used in virtually any programming language. Dynamic-link libraries are compatible with any programming language that has a compiler capable of performing dynamic linking. Here are some case examples:

- Compilers not listed above.
- Other programming languages: Java, PHP, Perl, Fortran, Pascal.
- Computer Algebra Systems (CAS): Wolfram Mathematica, Maplesoft Maple.
- DLL function prototypes—The exported functions of the dynamic libraries have the same prototype as their counterparts of the static libraries.
- Function Parameters—Some of the parameters of the library functions are language dependent. The table of input and output parameters for each function is a conceptual description, therefore, the user must check the specific language function to see how to use it. One example are the ID parameters (moduleID, etc.), which identify objects in non object-oriented languages. In object-oriented languages, the objects are identified by their instances and therefore the IDs are not present.
- Function Names—Some programming languages like C# have a feature called function overloading or polymorphism, that allows creating several functions with the same name, but with different input/output parameters. In languages without this feature, functions with different parameters must have different names.

Section 5.3: Functions in SD1 Programming Libraries

The functions available in Keysight SD1 Programming Libraries are listed below.

- SD_Module functions
- SD_AIN functions
- SD_Module functions (specific to Pathwave FPGA)
- SD_SandboxRegister functions

Note that there are a few API functions that are specific to M3300A/M3302A PXIe Combo Cards. The syntax for these API functions use the “SD_AIO” class, which represents Combo modules and use the “port” parameter to distinguish between the AWG and the Digitizer within. These API functions have been developed with the flexibility to exclusively use the “SD_AOU” or “SD_AIN” class specifically for AWG or Digitizer cards, respectively.

Table 18 List of SD_Module Functions

Function name	Short description
open	Initializes a hardware module and must be called before using any other module-related function.
close	Releases all resources that were allocated for a module with open and must always be called before exiting the application.
moduleCount	Returns the number of Keysight SD1 modules in the system.
getProductName	Returns the product name of the specified module.
getSerialNumber	Returns the serial number of the specified module.
getChassis	Returns the chassis number of where a module is located.
getSlot	Returns the slot number of where a module is located.
PXItriggerWrite	Sets the digital value of a PXI trigger in the PXI backplane. Only available in PXI/PXI Express form factors.
PXItriggerRead	Reads the digital value of a PXI trigger in the PXI backplane. Only available in PXI/PXI Express form factors.
getFirmwareVersion	Returns the firmware version of the module.
getHardwareVersion	Returns the hardware version of the module.
getOptions	Returns all the available option for selected card.
getTemperature	Returns the temperature of the module.
getType	Returns the type of module.
isOpen	Returns true if module is opened successfully.
translateTriggerIOtoExternalTriggerLine	Returns the translated external trigger value.
translateTriggerPXItoExternalTriggerLine	Returns the translated PXI trigger value.
runSelfTest	Performs a self test on the open module.

Table 19 List of SD_AIN Functions

Function name	Short description
channelCoupling	Returns the coupling of the specified channel.
channelFullScale	Returns fullScale value for the specified channel.
channelImpedance	Returns the impedance of the specified channel.
channelInputConfig	Configures the input full scale, impedance and coupling.
channelMaxFullScale	Returns maximum fullScale value for the specified input coupling and impedance.
channelMinFullScale	Returns minimum fullScale value for the specified input coupling and impedance.
channelPrescaler	Returns the prescaler value for the specified channel.
channelPrescalerConfig	Configures the input prescaler.
channelPrescalerConfigMultiple	Configures prescaler on the selected Digitizer channels.
channelTriggerConfig	Configures the analog trigger block for each channel Analog Trigger.
DAQanalogTriggerConfig	Configures the analog hardware trigger for the selected DAQ Trigger.
DAQconfig	Configures the acquisition of words Data Acquisition (DAQs) in two possible reading modes.
DAQdigitalTriggerConfig	Configures the digital hardware triggers for the selected DAQ Trigger.
DAQread	Reads the words acquired with the selected DAQ Data Acquisition (DAQs).
DAQstart	Starts an acquisition on the selected DAQ Data Acquisition (DAQ).
DAQstartMultiple	Starts an acquisition on the selected DAQs Data Acquisition (DAQs).
DAQstop	Stops the Data Acquisition (DAQ).
DAQstopMultiple	Stops multiple channels of Data Acquisition (DAQs).
DAQpause	Pauses the Data Acquisition (DAQ).
DAQpauseMultiple	Pauses multiple channels of Data Acquisition (DAQs).
DAQresume	Resumes acquisition on the selected DAQ.
DAQresumeMultiple	Resumes multiple channels of acquisition on the selected DAQs.
DAQflush	Flushes the acquisition buffers and resets the acquisition counter included in a data acquisition block.
DAQflushMultiple	Flushes the acquisition buffers and resets the acquisition counter included in a Data Acquisition block.
DAQnPoints	Returns the counter of DAQ.
DAQtrigger	Triggers the acquisition of words in the selected DAQs provided that they are configured for VI/HVI Trigger.
DAQtriggerMultiple	Triggers the acquisition of words in the selected DAQ provided that they are configured for VI/HVI Trigger.
DAQtriggerConfig	Configures DAQ for the selected DAQ Trigger.
DAQcounterRead	Reads the number of words acquired by the selected DAQ (Data Acquisition (DAQs)) since the last call to DAQflush or DAQ.
triggerIOconfig	Configures the trigger connector/line direction (I/O Triggers).

Function name	Short description
triggerIOwrite	Sets the trigger output and synchronization mode.
triggerIOread	Reads the trigger input (I/O Triggers).
clockSetFrequency	Sets the module clock frequency.
clockGetFrequency	Returns the frequency of Clock System.
clockGetSyncFrequency	Returns the frequency of the synchronization clock.
clockIOconfig	Configures the operation of the clock output connector.
clockResetPhase	Sets the module in a synchronized state, waiting for the first trigger to reset the phase of the internal clocks CLKsync and CLKsys.
DAQbufferPoolConfig	Configures buffer pool that will be filled with the data of the channel to be transferred to PC.
DAQbufferAdd	Adds an additional buffer to the channel's previously configured pool.
DAQbufferGet	Retrieves a filled buffer from the channel buffer pool. You have to call DAQbufferAdd with this buffer so the buffer can be used again.
DAQbufferPoolRelease	Releases the channel buffer pool and its resources. After this function is called, you need to call DAQbufferRemove consecutively to get all buffers back and release them.
DAQbufferRemove	Requests that a buffer be removed from the channel buffer pool. If a NULL pointer is returned, no more buffers remain in the buffer pool. Returned buffer is a previously added buffer from the user and the user has to release/delete it.
DAQtriggerExternalConfig	Configures DAQ for the external trigger source.
FFT	Calculates the FFT of data captured by DAQread for the selected channel. FFT frequency range goes from 0 to fs/2.
voltsToInt	Converts threshold in ChannelTriggerConfig() from 'double' to 'integer'.

Table 20 List of SD_Module Functions (specific to PathWave FPGA)

Function name	Short description
FPGAgetSandBoxRegister	Get sandbox register from name if programmable FPGA is loaded from.k7z file.
FPGAgetSandBoxRegisters	Get sandbox registers list if programmable FPGA is loaded from.k7z file.
FPGAload	Loads the bit file (*.k7z), which is generated using PathWave FPGA, onto the respective module's FPGA.
FPGAreset	Sends a reset signal to the Sandbox region.
FPGATriggerConfig	Configures PXI or FrontPanel triggers coming in or going out from sandbox region.
User FPGA HVI Actions/Events	Defines the HVI Actions & Events, which are available for sending information to and receiving information from the FPGA sandbox.

Table 21 List of SD_Module HVI Functions

Function name	Short description
Module HVI Engine	Enables HVI Engines located in the M3xxxA module's FPGA.
Module HVI Triggers	Activates PXI or Front Panel triggers for HVI sequences.

Table 22 List of SD_SandboxRegister Functions

Function name	Short description
readRegisterBuffer	Reads data buffer from a sandbox register bank or memory map.
readRegisterInt32	Reads int32 data from a sandbox register bank or memory map.
writeRegisterBuffer	Writes data buffer to a sandbox register bank or memory map.
writeRegisterInt32	Writes int32 data to a sandbox register bank or memory map.
Properties	Displays properties associated with registers.

5.3.1: Common References to parameter values

Following certain parameters and their respective values, which are required for one or more API functions.

Table 23 Clock connector options and corresponding values

Options	Description	Value
Disable	The CLK connector is disabled.	0
CLKrefOutput	The reference clock is available at the CLK connector.	1

Table 24 Analog Trigger Mode options

Option	Description	Name	Value
Positive Edge	Trigger is generated when the input signal is rising and crosses the threshold	AIN_RISING_EDGE	1
Negative Edge	Trigger is generated when the input signal is falling and crosses the threshold	AIN_FALLING_EDGE	2
Both Edges	Trigger is generated when the input signal crosses the threshold, no matter if it is rising or falling	AIN_BOTH_EDGES	3

Table 25 DAQ Trigger Mode options

Option	Description	Name	Value
Auto (Immediate)	The acquisition starts automatically after a call to function DAQstart	AUTOTRIG	0
Software / HVI	Software trigger. The acquisition is triggered by the function DAQtrigger, DAQtrigger provided that the DAQ is running. DAQtrigger can be executed from the user application (VI) or from an HVI.	SWHVITRIG	1
Hardware Digital Trigger	Hardware trigger. The DAQ waits for an external digital trigger (see Table 26 External Hardware Digital Trigger Source for the DAQ).	HWDIGTRIG	2
Hardware Analog Trigger	Hardware trigger. The DAQ waits for an external analog trigger (only products with analog inputs).	HWANATRIG	3

Table 26 External Hardware Digital Trigger Source for the DAQ

Option	Description	Name	Value
External I/O Trigger	The DAQ trigger is a TRG connector/line of the product (I/O Triggers). PXI form factor only: this trigger can be synchronized to CLK10.	TRIG_EXTERNAL	0
PXI Trigger	PXI form factor only. The DAQ trigger is a PXI trigger line and it is synchronized to CLK10.	TRIG_PXI	0-7

Table 27 Trigger behavior for the DAQ

Option	Description	Name	Value
None	No trigger has been activated	TRIGGER_NONE	0
Active High	Trigger is active when it is at level high	TRIGGER_HIGH	1
Active Low	Trigger is active when it is at level Low	TRIGGER_LOW	2
Rising Edge	Trigger is active on the rising edge	TRIGGER_RISE	3
Falling Edge	Trigger is active on the falling edge	TRIGGER_FALL	4

Table 28 Trigger Direction options and corresponding values

Options	Description	Name	Value
Trigger Output (readable)	TRG operates as a general purpose digital output signal, that can be written by the user software.	AIN_TRG_OUT	0
Trigger Input	TRG operates as a trigger input, or as a general purpose digital input signal, that can be read by the user software.	AIN_TRG_IN	1

Table 29 Sync mode options and corresponding values

Options	Description	Name	Value
Non-synchronized mode	The trigger is sampled with an internal 100 MHz clock.	SYNC_NONE	0
Synchronized mode	(PXI form factor only) The trigger is sampled using CLK10*.	SYNC_CLK10	1

* In synchronized mode, the trigger is synchronized to the nearest clock edge of the 10 MHz clock from the PXI chassis backplane. If using an external trigger, it should also be synchronized to the same 10 MHz reference. The trigger is sampled using CLKsync. If it is a multiple of 10 MHz, the maximum processing time would be less than 100 ns, varying depending on trigger arrival. If CLKsync is less than 10 MHz, the processing time will be greater than 100 ns.

Table 30 Variable clock system operation mode options and corresponding values

Options	Description	Name	Value
Low JitterMode	The clock system is set to achieve the lowest jitter, sacrificing tuning speed.	CLK_LOW_JITTER	0
Fast Tuning Mode	The clock system is set to achieve the fastest tuning time, sacrificing jitter performance.	CLK_FAST_TUNE	1

Table 31 SD_ResetMode attribute values

Attribute	Value
LOW	0 (to exit reset mode / “HIGH” state of the sandbox region)
HIGH	1 (to initiate reset mode and keep the sandbox region in the same mode until a “LOW” state is used to exit reset mode)
PULSE	2 (to reset the sandbox region by sending a “pulse”)

Table 32 FPGATriggerDirection attribute values

Attribute	Value
IN	0
INOUT	1

Table 33 TriggerPolarity attribute values

Attribute	Value
ACTIVE_LOW	0
ACTIVE_HIGH	1

Table 34 Window Types Used in FFT Functions

Option	Description	Name	Value
Rectangular	Simplest B-spine window	WINDOW_RECTANGULAR	0
Bartlett	Hybrid window	WINDOW_BARTLETT	1
Hanning	Side-lobes roll off about 18 dB per octave	WINDOW_HANNING	2
Hamming	Optimized to minimize the maximum nearest side lobe	WINDOW_HAMMING	3
Blackman	Higher-order generalized cosine windows for applications that require windowing by the convolution in the frequency-domain	WINDOW_BLACKMAN	4

5.3.2: Latency in Digitizers for various HVI Actions & Instructions

Table 35 displays the HVI actions and instructions along with the corresponding latency measured for Digitizers. The latency value for each instruction or action is determined and measured based on the GateWare design. You must use the measured values for the instructions as shown in the following table.

Table 35 Latency (in nanoseconds) for HVI Actions & Instructions

HVI Actions / Instructions	Latency in M3102A (DIG 500)	Comments
ChannelPrescalerConfig	30	Notes 2, 3
ChannelTriggerConfig	30	Notes 2, 3
DAQConfig	80	Notes 1, 3
DAQFlush	170	Notes 2, 3
DAQTrigger	330	Notes 2, 3
DAQStart	120	Notes 2, 3

Notes:

1. "DAQConfig" is a multi-step instruction, which takes 20ns to run.

2. This HVI action / instruction takes 10ns to run.

3. The HVI actions / instructions in Digitizers are mutually exclusive, that is, cannot be run simultaneously.

HVI related latency in FPGA User Sandbox

The FPGA registers and Memory map access latency from HVI for both AWG and Digitizers is measured as: *HVI_FPGA_ProductDelay* = 1 cycles (10ns). Refer to the *SD1 3.x Software for M320xA / M330xA AWG User's Guide* for latency information in AWGs. Table 36 summarizes the latency for all FPGA read/write instructions. For more information regarding latencies related to HVI, refer to *Chapter 7: HVI Time Management and Latency* in the *PathWave Test Sync Executive User Manual*.

Table 36 Minimum delay required (in nanoseconds) for FPGA User Sandbox instructions

Instructions	Minimum Delay required
fpga_array_read	60
fpga_array_read (Address from HviRegister)	80
fpga_array_write	30
fpga_array_write (Address or data from HviRegister)	50
fpga_register_read	60
fpga_register_write	30
fpga_register_write (Address or data from HviRegister)	50

Section 5.4: SD_Module functions

5.4.1: open

Description Initializes a hardware module and must be called before using any other module-related function. A module can be opened using the serial number or the chassis and slot number. Using the serial number ensures the same module is always opened regardless of its chassis or slot location.

Parameters

Table 37 Input parameters for open

Input Parameter name	Description
productName	Module's product name (for example, "M3102A" or "M3202A"). The product name can be found on the product or can be retrieved with getProductName.
serialNumber	Module's serial number (for example, "ES5641"). The serial number can be found on the product or can be retrieved with getSerialNumber.
chassis	Chassis number where the module is located. The chassis number can be found in Keysight SD1 software or can be retrieved with getChassis.
nSlot	Slot number in the chassis where the module is located. The slot number can be found on the chassis or can be retrieved with getSlot.
compatibility	Forces the channel numbers to be compatible with legacy models. Channel numbering (channel enumeration) can start as CH0 or CH1. See Channel Numbering and Compatibility Mode.
partNumber	Name of the module such as "M3102A" or "M3202A".
options	The format of the <options> string is "optionName1=value1, optionName2=value2, and so on." Valid Values are: simulate=true|false. If 'true', the module is open as a simulated (demo) card. channelNumbering=Signadyne|Keysight, HVI2=true|false and Factory=XX. Signadyne modules label their channel numbers starting from 0, Keysight modules start from 1.

Table 38 Output parameters for open

Output Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier or a negative number that indicates an error, see Description of SD1 Error IDs.
errorOut	See Description of SD1 Error IDs.

Syntax

Table 39 API syntax for open

Language	Syntax

C	`int SD_Module_openWithSerialNumber(const char* productName, const char* serialNumber);`
	`int SD_Module_openWithSlot(const char* productName, int chassis, int nSlot);`
	`int SD_Module_openWithSerialNumberCompatibility(const char* productName, const char* serialNumber, int compatibility);`
	`int SD_Module_openWithSlotCompatibility(const char* productName, int chassis, int nSlot, int compatibility);`
	`int SD_Module_openWithOptions(const char *partNumber, int chassis, int nSlot, const char *options);`

C++	`int SD_Module::open(std::string partNumber, int nChassis, int nSlot, int compatibility);`
	`int SD_Module::open(std::string partNumber, std::string serialNumber, int compatibility);`
	`int SD_Module::open(std::string partNumber, int nChassis, int nSlot);`
	`int SD_Module::open(std::string partNumber, std::string serialNumber);`
	`int SD_Module::open(std::string partNumber, int nChassis, int nSlot, std::string options);`

.NET	`int SD_Module::open(string productName, string serialNumber);`
	`int SD_Module::open(string productName, int chassis, int nSlot);`
	`int SD_Module::open(string productName, string serialNumber, int compatibility);`
	`int SD_Module::open(string productName, int chassis, int nSlot, int compatibility);`
	`int SD_Module::open(string partNumber, int nChassis, int nSlot, string options);`

Python	`SD_Module.openWithSerialNumber(productName, serialNumber)`
	`SD_Module.openWithSlot(productName, chassis, nSlot)`
	`SD_Module.openWithSerialNumberCompatibility(productName, serialNumber, compatibility)`
	`SD_Module.openWithSlotCompatibility(productName, chassis, nSlot, compatibility)`
	`SD_Module.openWithOptions(partNumber, nChassis, nSlot, options)`

HVI	Not available

5.4.2: close

Description Releases all resources that were allocated for a module with open and must always be called before exiting the application.

Parameters

Table 40 Input parameters for close

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.

Table 41 Output parameters for close

Output Parameter name	Description
errorOut	See Description of SD1 Error IDs.

Syntax

Table 42 API syntax for close

Language	Syntax
C	`int SD_Module_close(int moduleID);`
C++	`int SD_Module::close();`
.NET	`int SD_Module::close();`
Python	`SD_Module.close()`
HVI	Not available

5.4.3: moduleCount

Description

Returns the number of Keysight SD1 modules (M31xxA/M32xxA/M33xxA) installed in the system.

NOTE (Specific to object-oriented programming languages only) moduleCount is a static function.

Parameters

Table 43 Output parameters for moduleCount

Output Parameter name	Description
nModules	Number of Keysight SD1 modules installed in the system. Negative numbers indicate an error. See Description of SD1 Error IDs.
errorOut	See Description of SD1 Error IDs.

Syntax

Table 44 API syntax for moduleCount

Language	Syntax
C	`int SD_Module_count();`
C++	`int SD_Module::moduleCount();`
.NET	`int SD_Module::moduleCount();`
Python	`SD_Module.moduleCount()`
HVI	Not available

5.4.4: getProductName

Description

Returns the product name of the specified module.

NOTE (Specific to object-oriented programming languages only) getProductName is a static function. Object-oriented languages also have a non-static function without arguments.

Parameters

Table 45 Input parameters for getProductName

Input Parameter name	Description
index	Module index. It must be in the range (0 to nModules-1), where nModules is returned by moduleCount.
chassis	Chassis number where the module is located. The chassis number can be found in Keysight SD1 software or can be retrieved with getChassis.
slot	Slot number in the chassis where the module is located. The slot number can be found on the chassis or can be retrieved with getSlot.
moduleId	Module identifier returned by open function.

Table 46 Output parameters for getProductName

Output Parameter name	Description
productName	Product name of the specified module. This product name can be used in open.
errorOut	See Description of SD1 Error IDs.

Syntax

Table 47 API syntax for getProductName

Language	Syntax

C	`int SD_Module_getProductNameByIndex(int index, char *productName);`
	`int SD_Module_getProductNameBySlot(int chassis, int slot, char* productName);`
	`char* SD_Module_getProductName(int moduleId, char * productName);`

C++	`std::string SD_Module::getProductName();`
	`std::string SD_Module::getProductName(int index);`
	`std::string SD_Module::getProductName(int chassis, int slot);`

.NET	`string SD_Module::getProductName(int index);`
	`string SD_Module::getProductName(int chassis, int slot);`
	`string SD_Module::getProductName();`

Python	`SD_Module.getProductNameByIndex(index)`
	`SD_Module.getProductNameBySlot(chassis, slot)`
	`SD_Module.getProductName()`

HVI	Not available

5.4.5: getSerialNumber

Description

Returns the serial number of the specified module.

NOTE (Specific to object-oriented programming languages only) getSerialNumber is a static function. Object-oriented languages also have a non-static function without arguments.

Parameters

Table 48 Input parameters for getSerialNumber

Input Parameter name	Description
index	Module index. It must be in the range (0 to nModules-1), where nModules is returned by moduleCount.
chassis	Chassis number where the module is located. The chassis number can be found in Keysight SD1 software or can be retrieved with getChassis.
slot	Slot number in the chassis where the module is located. The slot number can be found on the chassis or can be retrieved with getSlot.
moduleId	Module identifier returned by open function.

Table 49 Output parameters for getSerialNumber

Output Parameter name	Description
serialNumber	Serial number of the specified module. This serial number can be used in open.
errorOut	See Description of SD1 Error IDs.

Syntax

Table 50 API syntax for getSerialNumber

Language	Syntax

C	`int SD_Module_getSerialNumberByIndex(int index, char *serialNumber);`
	`int SD_Module_getSerialNumberBySlot(int chassis, int slot, char *serialNumber);`
	`char* SD_Module_getSerialNumber(int moduleId, char* serialNumber);`

C++	`std::string SD_Module::getSerialNumber(int index);`
	`std::string SD_Module::getSerialNumber();`
	`std::string SD_Module::getSerialNumber(int chassis, int slot);`

.NET	`string SD_Module::getSerialNumber(int index);`
	`string SD_Module::getSerialNumber(int chassis, int slot);`
	`string SD_Module::getSerialNumber();`

Python	`SD_Module.getSerialNumberByIndex(index)`
	`SD_Module.getSerialNumberBySlot(chassis, slot)`
	`SD_Module.getSerialNumber()`

HVI	Not available

5.4.6: getChassis

Description

Returns the chassis number of where a module is located.

NOTE (Specific to object-oriented programming languages only) getChassis is a static function. Object-oriented languages also have a non-static function without arguments.

Parameters

Table 51 Input parameters for getChassis

Input Parameter name	Description
moduleID	Module identifier, returned by open.
index	Module index. It must be in the range (0 to nModules-1), where nModules is returned by moduleCount.

Table 52 Output parameters for getChassis

Output Parameter name	Description
chassis	Chassis number of where a module is located. Negative numbers indicate an error. See Description of SD1 Error IDs.
errorOut	See Description of SD1 Error IDs.

Syntax

Table 53 API syntax for getChassis

Language	Syntax
C	`int SD_Module_getChassisByIndex(int index);` `int SD_Module_getChassis(int moduleID);`
C++	`int SD_Module::getChassis(int index);` `int SD_Module::getChassis();`
.NET	`int SD_Module::getChassis(int index);` `int SD_Module::getChassis();`
Python	`SD_Module.getChassis()` `SD_Module.getChassisByIndex(index)`
HVI	Not available

5.4.7: getSlot

Description

Returns the slot number of where a module is located in the chassis.

NOTE (Specific to object-oriented programming languages only) getSlot is a static function.

Parameters

Table 54 Input parameters for getSlot

Input Parameter name	Description
moduleID	Module identifier, returned by open.
index	Module index. It must be in the range (0 to nModules-1), where nModules is returned by moduleCount.

Table 55 Output parameters for getSlot

Output Parameter name	Description
slot	Slot number of where the module is located in the chassis. Negative numbers indicate an error. See Description of SD1 Error IDs.
errorOut	See Description of SD1 Error IDs.

Syntax

Table 56 API syntax for getSlot

Language	Syntax
C	`int SD_Module_getSlotByIndex(int index);` `int SD_Module_getSlot(int moduleID);`
C++	`static int SD_Module::getSlot(int index);` `int SD_Module::getSlot();`
.NET	`static int SD_Module::getSlot(int index);` `int SD_Module::getSlot();`
Python	`SD_Module.getSlot()` `SD_Module.getSlotByIndex(index)`
HVI	Not available

5.4.8: PXItriggerWrite

Description

Sets the digital value of a PXI trigger in the PXI backplane.

NOTE This function is only available in PXI/PXI Express form factors.

Parameters

Table 57 Input parameters for PXItriggerWrite

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
nPXItrigger	PXI trigger line number. Input Values 0 to 7.
value	Digital value with negated logic: 0 (ON) or 1 (OFF)

Table 58 Output parameters for PXItriggerWrite

Output Parameter name	Description
errorOut	See Description of SD1 Error IDs.

Syntax

Table 59 API syntax for PXItriggerWrite

Language	Syntax
C	`int SD_Module_PXItriggerWrite(int moduleID, int nPXItrigger, int value);`
C++	`int SD_Module::PXItriggerWrite(int nPXItrigger, int value);`
.NET	`int SD_Module::PXItriggerWrite(int nPXItrigger, int value);`
Python	`SD_Module.PXItriggerWrite(nPXItrigger, value)`
HVI	Not available

5.4.9: PXItriggerRead

Description

Reads the digital value of a PXI trigger in the PXI backplane.

NOTE This function is only available in PXI/PXI Express form factors.

Parameters

Table 60 Input parameters for PXItriggerRead

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
nPXItrigger	PXI trigger line number. Input Values 0 to 7.

Table 61 Output parameters for PXItriggerRead

Output Parameter name	Description
value	Digital value with negated logic: 0 (ON) or 1 (OFF). Negative numbers indicate an error. See Description of SD1 Error IDs.
errorOut	See Description of SD1 Error IDs.

Syntax

Table 62 API syntax for PXItriggerRead

Language	Syntax
C	`int SD_Module_PXItriggerRead(int moduleID, int nPXItrigger);`
C++	`int SD_Module::PXItriggerRead(int nPXItrigger);`
.NET	`int SD_Module::PXItriggerRead(int nPXItrigger);`
Python	`SD_Module.PXItriggerRead(nPXItrigger)`
HVI	Not available

5.4.10: getFirmwareVersion

Description Returns the Firmware Version of the module.

Parameters

Table 63 Input parameters for getFirmwareVersion

Input Parameter name	Description
moduleID	Module identifier, returned by open.

Table 64 Output parameters for getFirmwareVersion

Output Parameter name	Description
firmwareVersion	Firmware Version of the specified module.
errorOut	See Description of SD1 Error IDs.

Syntax

Table 65 API syntax for getFirmwareVersion

Language	Syntax
C	`int SD_Module_getFirmwareVersion(int moduleID, char *firmwareVersion);`
C++	`std::string SD_Module::getFirmwareVersion();`
.NET	`String SD_Module::getFirmwareVersion();`
Python	`SD_Module.getFirmwareVersion()`
HVI	Not available

5.4.11: getHardwareVersion

Description Returns the hardware version of the module.

Parameters

Table 66 Input parameters for getHardwareVersion

Input Parameter name	Description
moduleID	Module identifier, returned by open.

Table 67 Output parameters for getHardwareVersion

Output Parameter name	Description
hardwareVersion	Hardware Version of the specified module.
errorOut	See Description of SD1 Error IDs.

Syntax

Table 68 API syntax for getHardwareVersion

Language	Syntax
C	`int SD_Module_getHardwareVersion(int moduleID, char *hardwareVersion);`
C++	`std::string SD_Module::getHardwareVersion();`
.NET	`String SD_Module::getHardwareVersion();`
Python	`SD_Module.getHardwareVersion()`
HVI	Not available

5.4.12: getOptions

Description

Returns all the available option for selected card. It has two uses:

1 normal—the user calls getOptions("<OptionKey>"). This would return the values listed below:

- the value of desired option (if it exists).
- "NONE", if the value doesn't exist.
- NULL pointer, if there is not enough space to write all the data or you are trying to access an unopened module.

2 help - the user calls getOptions("?"). This would return the values listed below:

- list of all the possible keys (available or not) the function can be queried.
- NULL pointer, if there is not enough space to write all the data.

Parameters

Table 69 Input parameters for getOptions

Input Parameter name	Description
moduleID	Module identifier, returned by open.
optionKey	Possible values are: ▪ "model" ▪ "channels" ▪ "clock" ▪ "memory" ▪ "modulation" ▪ "dual_modulation" ▪ "up_modulation" ▪ "down_modulation" ▪ "onboard_dc" ▪ "streaming" ▪ "fpga" ▪ "fpga_programmable" ▪ "hvi"
varToFill	empty char array of options.
vartoFillSize	size of the char array.
objectType	type of module. Refer to getType.

Table 70 Output parameters for getOptions

Output Parameter name	Description
options	The options available for the selected card.
errorOut	See Description of SD1 Error IDs.

Syntax

Table 71 API syntax for getOptions

Language	Syntax
C	`const char* SD_Module_getOptions(int moduleID, const char*optionkey, char*varToFill, int varToFillSize, int objectType);`
C++	`std::string SD_Module::getOptions(std::string optionkey);`
.NET	`string SD_Module::getOptions(string optionkey);`
Python	`SD_Module.getOptions(optionkey)`
HVI	Not available

5.4.13: getTemperature

Description Returns the temperature of the module.

Parameters

Table 72 Input parameters for getTemperature

Input Parameter name	Description
moduleID	Module identifier, returned by open.

Table 73 Output parameters for getTemperature

Output Parameter name	Description
temperature	Temperature of the module.
errorOut	See Description of SD1 Error IDs.

Syntax

Table 74 API syntax for getTemperature

Language	Syntax
C	`double SD_Module_getTemperature(int moduleId);`
C++	`double SD_Module::getTemperature();`
.NET	`double SD_Module::getTemperature();`
Python	`SD_Module.getTemperature()`
HVI	Not available

5.4.14: getType

Description

Returns the type of module.

NOTE (Specific to object-oriented programming languages only) getType is a static function. Object-oriented languages also have a non-static function without arguments.

Parameters

Table 75 Input parameters for getType

Input Parameter name	Description
moduleID	Module identifier, returned by open.
chassis	Chassis number where the module is located. The chassis number can be found in Keysight SD1 software or can be retrieved with getChassis.
slot	Slot number in the chassis where the module is located. The slot number can be found on the chassis or can be retrieved with getSlot.
index	Module index. It must be in the range (0 to nModules-1), where nModules is returned by moduleCount.

Table 76 Output parameters for getType

Output Parameter name	Description
type	Type of module. Possible values are: ▪ 2–indicates that the module is an AWG. ▪ 6–indicates that the module is a DIG. ▪ 7–indicates that the module is a COMBO.
errorOut	See Description of SD1 Error IDs.

Syntax

Table 77 API syntax for getType

Language	Syntax

C	`int SD_Module_getTypeBySlot(int chassis, int slot);`
	`int SD_Module_getTypeByIndex(int index);`
	`int SD_Module_getType(int moduleID);`

C++	`int SD_Module::getType(int chassis, int slot);`
	`int SD_Module::getType(int index);`
	`int SD_Module::getType();`

.NET	`int SD_Module::getType(int chassis, int slot);`
	`int SD_Module::getType(int index);`
	`int SD_Module::getType();`

Python	`SD_Module.getTypeBySlot(chassis,slot)`
	`SD_Module.getTypeByIndex(index)`
	`SD_Module.getType()`

HVI	Not available

5.4.15: isOpen

Description Returns 'true' if module is opened successfully.

Parameters

Table 78 Input parameters for isOpen

Input Parameter name	Description
Not available	Not available

Table 79 Output parameters for isOpen

Output Parameter name	Description
outValue	'True' if module is opened, else 'False'.
errorOut	See Description of SD1 Error IDs.

Syntax

Table 80 API syntax for isOpen

Language	Syntax
C	Not available
C++	`bool SD_Module::isOpen();`
.NET	`bool SD_Module::isOpen();`
Python	`SD_Module.isOpen()`
HVI	Not available

5.4.16: translateTriggerIOtoExternalTriggerLine

Description Returns the translated external trigger value.

Parameters

Table 81 Input parameters for translateTriggerIOtoExternalTriggerLine

Input Parameter name	Description
moduleId	Module identifier, returned by open.
trigger	trigger IO number. Valid inputs are 1 (for both AWG and DIG) or 1 & 2 (for COMBO).

Table 82 Output parameters for translateTriggerIOtoExternalTriggerLine

Output Parameter name	Description
translatedValue	Translated external trigger value.
errorOut	See Description of SD1 Error IDs.

Syntax

Table 83 API syntax for translateTriggerIOtoExternalTriggerLine

Language	Syntax
C	`int SD_Module_translateTriggerIOtoExternalTriggerLine(int moduleID, int trigger);`
C++	`int SD_Module::translateTriggerIOtoExternalTriggerLine(int trigger);`
.NET	`int SD_Module::translateTriggerIOtoExternalTriggerLine(int trigger);`
Python	`SD_Module.translateTriggerIOtoExternalTriggerLine(trigger)`
HVI	Not available

5.4.17: translateTriggerPXItoExternalTriggerLine

Description Returns the translated PXI trigger value.

Parameters

Table 84 Input parameters for translateTriggerPXItoExternalTriggerLine

Input Parameter name	Description
moduleId	Module identifier, returned by open.
trigger	PXI trigger number. Input Values 0 to 7.

Table 85 Output parameters for translateTriggerPXItoExternalTriggerLine

Output Parameter name	Description
translatedValue	returns translated PXI trigger value.
errorOut	See Description of SD1 Error IDs.

Syntax

Table 86 API syntax for translateTriggerPXItoExternalTriggerLine

Language	Syntax
C	`int SD_Module_translateTriggerPXItoExternalTriggerLine(int moduleID, int trigger);`
C++	`int SD_Module::translateTriggerPXItoExternalTriggerLine(int trigger);`
.NET	`int SD_Module::translateTriggerPXItoExternalTriggerLine(int trigger);`
Python	`SD_Module.translateTriggerPXItoExternalTriggerLine(trigger)`
HVI	Not available

5.4.18: runSelfTest

Description Performs a self test on the open module.

Parameters **Table 87 Output parameters for runSelfTest**

Output Parameter name	Description
errorOut	See Description of SD1 Error IDs.

Syntax **Table 88 API syntax for runSelfTest**

Language	Syntax
C	`int SD_Module_runSelfTest();`
C++	`int SD_Module::runSelfTest();`
.NET	`int SD_Module::runSelfTest();`
Python	`SD_Module.runSelfTest()`
HVI	Not available

Section 5.5: SD_AIN functions

5.5.1: channelCoupling

Description Returns the coupling of the specified channel.

Parameters **Table 89** **Input parameters for channelCoupling**

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
Channel	Input Channel number.

Table 90 **Output parameters for channelCoupling**

Output Parameter name	Description
Coupling	Coupling of the Channel.
errorOut	See Description of SD1 Error IDs.

Syntax **Table 91** **API syntax for channelCoupling**

Language	Syntax
C	`int SD_AIN_channelCoupling(int moduleID, int channel);`
C++	`int SD_AIN::channelCoupling(int channel);`
.NET	`int SD_AIN::channelCoupling(int channel);`
Python	`SD_AIN.channelCoupling(channel)`
HVI	Not available.

5.5.2: channelFullScale

Description Returns fullScale value for the specified channel.

Parameters

Table 92 Input parameters for channelFullScale

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
Channel	Input Channel number.

Table 93 Output parameters for channelFullScale

Output Parameter name	Description
fullScale	Full Scale value of the Channel.
errorOut	See Description of SD1 Error IDs.

Syntax

Table 94 API syntax for channelFullScale

Language	Syntax
C	`double SD_AIN_channelFullScale(int moduleID, int channel);`
C++	`double SD_AIN::channelFullScale(int channel);`
.NET	`double SD_AIN::channelFullScale(int channel);`
Python	`SD_AIN.channelFullScale(channel)`
HVI	Not available.

5.5.3: channelImpedance

Description Returns the impedance on the specified channel.

Parameters

Table 95 Input parameters for channelImpedance

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
Channel	Input Channel number.

Table 96 Output parameters for channelImpedance

Output Parameter name	Description
Impedance	Impedance of the Channel.
errorOut	See Description of SD1 Error IDs.

Syntax

Table 97 API syntax for channelImpedance

Language	Syntax
C	`int SD_AIN_channelImpedance(int moduleID, int channel);`
C++	`int SD_AIN::channelImpedance(int channel);`
.NET	`int SD_AIN::channelImpedance(int channel);`
Python	`SD_AIN.channelImpedance(channel)`
HVI	Not available.

5.5.4: channelInputConfig

Description

Configures the input full scale, impedance and coupling as applicable according to the product Full Scale, Impedance and Coupling.

Parameters

Table 98 Input parameters for channelInputConfig

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
Channel	Input Channel number.
fullScale	Input full scale, in volts.
impedance	Input impedance
coupling	Input coupling

Table 99 Output parameters for channelInputConfig

Output Parameter name	Description
errorOut	See Description of SD1 Error IDs.

Syntax

Table 100 API syntax for channelInputConfig

Language	Syntax
C	`int SD_AIN_channelInputConfig(int moduleID, int channel, double fullScale, int impedance, int coupling);`
C++	`int SD_AIN::channelInputConfig(int channel, double fullScale, int impedance, int coupling);`
.NET	`int SD_AIN::channelInputConfig(int channel, double fullScale, int impedance, int coupling);`
Python	`SD_AIN.channelInputConfig(channel, fullScale, impedance, coupling)`
HVI	Not available.

5.5.5: channelMaxFullScale

Description Returns maximum fullScale value for the specified input coupling and impedance.

Parameters

Table 101 Input parameters for channelMaxFullScale

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
impedance	Input impedance.
coupling	Input coupling.

Table 102 Output parameters for channelMaxFullScale

Output Parameter name	Description
fullScale	FullScale value.
errorOut	See Description of SD1 Error IDs.

Syntax

Table 103 API syntax for channelMaxFullScale

Language	Syntax
C	`double SD_AIN_channelMaxFullScale(int moduleID, int impedance, int coupling);`
C++	`double SD_AIN::channelMaxFullScale(int impedance, int coupling);`
.NET	`double SD_AIN::channelMaxFullScale(int impedance, int coupling);`
Python	`SD_AIN.channelMaxFullScale(impedance, coupling)`
HVI	Not available.

5.5.6: channelMinFullScale

Description Returns minimum fullScale value for the specified input coupling and impedance.

Parameters

Table 104 Input parameters for channelMinFullScale

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
impedance	Input impedance.
coupling	Input coupling.

Table 105 Output parameters for channelMinFullScale

Output Parameter name	Description
fullScale	FullScale value.
errorOut	See Description of SD1 Error IDs.

Syntax

Table 106 API syntax for channelMinFullScale

Language	Syntax
C	`double SD_AIN_channelMinFullScale(int moduleID, int impedance, int coupling);`
C++	`double SD_AIN::channelMinFullScale(int impedance, int coupling);`
.NET	`double SD_AIN::channelMinFullScale(int impedance, int coupling);`
Python	`SD_AIN.channelMinFullScale(impedance, coupling)`
HVI	Not available

5.5.7: channelPrescaler

Description Returns the prescaler value for the specified channel.

Parameters

Table 107 Input parameters for channelPrescaler

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
Channel	Input Channel number.

Table 108 Output parameters for channelPrescaler

Output Parameter name	Description
errorOut	See Description of SD1 Error IDs.

Syntax

Table 109 API syntax for channelPrescaler

Language	Syntax
C	`int SD_AIN_channelPrescaler(int moduleID, int channel);`
C++	`int SD_AIN::channelPrescaler(int channel);`
.NET	`int SD_AIN::channelPrescaler(int channel);`
Python	`SD_AIN.channelPrescaler(channel)`
HVI	Not available.

5.5.8: channelPrescalerConfig

Description Configures the input Prescaler.

Parameters **Table 110 Input parameters for channelPrescalerConfig**

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
Channel	Input Channel number.
prescaler	Input Prescaler value.

Table 111 Output parameters for channelPrescalerConfig

Output Parameter name	Description
errorOut	See Description of SD1 Error IDs.

Syntax **Table 112 API syntax for channelPrescalerConfig**

Language	Syntax
C	`int SD_AIN_channelPrescalerConfig(int moduleID, int nChannel, int prescaler);`
C++	`int SD_AIN::channelPrescalerConfig(int nChannel, int prescaler);`
.NET	`int SD_AIN::channelPrescalerConfig(int nChannel, int prescaler);`
Python	`SD_AIN.channelPrescalerConfig(nChannel, prescaler)`
HVI	Refer to Table 113 HVI syntax and corresponding values for channelPrescalerConfig.

Table 113 HVI syntax and corresponding values for channelPrescalerConfig

Language	Syntax	Parameter	Possible values
Python	`module.hvi.instruction_set.channel_prescaler_config.id`	`channel.id`	• `Channel number`
		`prescaler.id`	• `Positive integer, used to determine input sampling rate.`
.NET	`module.Hvi.InstructionSet.ChannelPrescalerConfig.Id`	`Channel.Id`	• `Channel number`
		`Prescaler.Id`	• `Positive integer, used to determine input sampling rate.`

5.5.9: channelPrescalerConfigMultiple

Description Configures prescaler value on more than one selected Digitizer Channels.

Parameters

Table 114 Input parameters for channelPrescalerConfigMultiple

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
mask	Mask to select the channels.
prescaler	Input Prescaler value.

Table 115 Output parameters for channelPrescalerConfigMultiple

Output Parameter name	Description
errorOut	See Description of SD1 Error IDs.

Syntax

Table 116 API syntax for channelPrescalerConfigMultiple

Language	Syntax
C	`int SD_AIN_channelPrescalerConfigMultiple(int moduleID, int mask, int prescaler);`
C++	`int SD_AIN::channelPrescalerConfigMultiple(int mask, int prescaler);`
.NET	`int SD_AIN::channelPrescalerConfigMultiple(int mask, int prescaler);`
Python	`SD_AIN.channelPrescalerConfigMultiple(mask, prescaler)`
HVI	Not available

5.5.10: channelTriggerConfig

Description Configures the analog trigger block for each channel.

Parameters

Table 117 Input parameters for channelTriggerConfig

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
nChannel	Input Channel number.
analogTriggerMode	Analog Trigger mode. Refer to Table 24 Analog Trigger Mode options.
threshold	Threshold level, in volts.

Table 118 Output parameters for channelTriggerConfig

Output Parameter name	Description
errorOut	See Description of SD1 Error IDs.

Syntax

Table 119 API syntax for channelTriggerConfig

Language	Syntax
C	`int SD_AIN_channelTriggerConfig(int moduleID, int nChannel, int analogTriggerMode, double threshold);`
C++	`int SD_AIN::channelTriggerConfig(int nChannel, int analogTriggerMode, double threshold);`
.NET	`int SD_AIN::channelTriggerConfig(int nChannel, int analogTriggerMode, double threshold);`
Python	`SD_AIN.channelTriggerConfig(nChannel, analogTriggerMode, threshold)`
HVI	Refer to Table 120 HVI syntax and corresponding values for channelTriggerConfig.

Table 120 HVI syntax and corresponding values for channelTriggerConfig

Language	Syntax	Parameter	Possible values
Python	`module.hvi.instruction_set.channel_trigger_config.id`	`channel.id`	• `Channel number`
		`analog_trigger_mode.id`	• `AIN_RISING_EDGE` • `AIN_FALLING_EDGE` • `AIN_BOTH_EDGES`
		`threshold.id`	• `The threshold in volts that the DAQ will begin its acquisition at.`

Language	Syntax	Parameter	Possible values
.NET	module.Hvi.InstructionSet.ChannelTriggerConfig.Id	Channel.Id	• Channel number
		AnalogTriggerMode.Id	• AIN_RISING_EDGE • AIN_FALLING_EDGE • AIN_BOTH_EDGES
		Threshold.Id	• The threshold in volts that the DAQ will begin its acquisition at.

NOTE Use the voltsToInt() API function to convert threshold from double to integer.

5.5.11: DAQanalogTriggerConfig

Description

Configures the analog hardware trigger for the selected DAQ Trigger. This feature is available for Data Acquisition Blocks (DAQs), which are included in products with analog inputs only.

Parameters

Table 121 Input parameters for DAQanalogTriggerConfig

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
nDAQ	DAQ Channel number.
analogTriggerMask	Enter Channel Mask ID to enable analog triggering on specified channel.

Table 122 Output parameters for DAQanalogTriggerConfig

Output Parameter name	Description
errorOut	See Description of SD1 Error IDs.

Syntax

Table 123 API syntax for DAQanalogTriggerConfig

Language	Syntax
C	`int SD_AIN_DAQanalogTriggerConfig(int moduleID, int nDAQ, int analogTriggerMask);`
C++	`int SD_AIN::DAQanalogTriggerConfig(int nDAQ, int analogTriggerMask);`
.NET	`int SD_AIN::DAQanalogTriggerConfig(int nDAQ, int analogTriggerMask);`
Python	`SD_AIN.DAQanalogTriggerConfig(nDAQ, analogTriggerMask)`
HVI	Refer to Table 124 HVI syntax and corresponding values for DAQanalogTriggerConfig.

Table 124 HVI syntax and corresponding values for DAQanalogTriggerConfig

Language	Syntax	Parameter	Possible values
Python	`module.hvi.instruction_set.daq_analog_trigger_config.id`	`channel.id`	• `Channel number`
		`analog_trigger_mask.id`	• `Enter Channel ID to enable analog triggering on specified channel.`
.NET	`module.Hvi.InstructionSet.DaqAnalogTriggerConfig.Id`	`Channel.Id`	• `Channel number`
		`AnalogTriggerMask.Id`	• `Enter Channel ID to enable analog triggering on specified channel.`

5.5.12: DAQconfig

Description

Configures the acquisition of words Data Acquisition (DAQs) in two possible reading modes:

- Blocking: Use "DAQread()" to read the words. DAQread is a blocking function that is released when the amount of words specified in (DAQpoints) is acquired or when timeout elapses. This mode is enabled when a callback function is not specified (it is set to null).
- Non-blocking: You may specify a callback function which is called whenever the (DAQeventDataReady) event is signaled or when timeout elapses. In the latter condition, there may be words available, but less than the amount specified in (DAQpoints).

NOTE

The input for 'DAQpointsPerCycle' must always be an even number. If you enter an odd number as input, the SD1 API returns the following message before continuing operation.

"Warning: DAQ supports only even number of 'DAQpointsPerCycle'. The input value is reduced by 1."

Parameters

Table 125 Input parameters for DAQconfig

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
nDAQ	DAQ Channel number.
analogTriggerMask	Enter Channel Mask ID to enable analog triggering on specified channel.
triggerMode	Trigger mode.
triggerDelay	Number of samples that trigger is delayed (or advanced if negative).
DAQpointsPerCycle	Number of words to acquire per trigger.
cycles	Number of acquisition cycles. Each cycle requires a trigger specified by the triggerMode parameter. A negative number indicates continuous acquisition.

Table 126 Output parameters for DAQconfig

Output Parameter name	Description
errorOut	See Description of SD1 Error IDs and Description of SD1 Warning IDs.

Syntax

Table 127 API syntax for DAQconfig

Language	Syntax
C	`int SD_AIN_DAQconfig(int moduleID, int nDAQ, int DAQpointsPerCycle, int cycles, int triggerDelay, int triggerMode);`
C++	`int SD_AIN::DAQconfig(int nDAQ, int nDAQpointsPerCycle, int nCycles, int triggerDelay, int triggerMode);`
.NET	`int SD_AIN::DAQconfig(int nDAQ, int DAQpointsPerCycle, int cycles, int triggerDelay, int triggerMode);`
Python	`SD_AIN.DAQconfig(nDAQ, DAQpointsPerCycle, cycles, triggerDelay, triggerMode)`
HVI	Refer to Table 128 HVI syntax and corresponding values for DAQconfig.

Table 128 HVI syntax and corresponding values for DAQconfig

Language	Syntax	Parameter	Possible values
Python	module.hvi.instruction_set.daq_config.id	channel.id	• Channel number
		cycles.id	• Number of acquisition cycles. Each cycle requires a trigger specified by triggerMode. A negative number means continuous acquisition.
		daq_points_per_cycle.id	• Number of words to acquire per trigger.
		trigger_mode.id	• AUTOTRIG • SWHVITRIG • HWANATRIG • HWDIGTRIG
		trigger_delay.id	• Number of samples that trigger is delayed (or advanced if negative)
.NET	module.Hvi.InstructionSet.DaqConfig.Id	Channel.Id	• Channel number
		Cycles.Id	• Number of acquisition cycles. Each cycle requires a trigger specified by triggerMode. A negative number means continuous acquisition.
		DaqPointsPerCycle.Id	• Number of words to acquire per trigger.
		TriggerMode.Id	• AUTOTRIG • SWHVITRIG • HWANATRIG • HWDIGTRIG
		TriggerDelay.Id	• Number of samples that trigger is delayed (or advanced if negative)

5.5.13: DAQdigitalTriggerConfig

Description Configures the digital hardware triggers for the selected DAQ Trigger.

Parameters

Table 129 Input parameters for DAQdigitalTriggerConfig

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
nDAQ	DAQ Channel number.
triggerSource	Hardware Digital Trigger Source. Refer to Table 26 External Hardware Digital Trigger Source for the DAQ.
triggerBehavior	Refer to Table 27 Trigger behavior for the DAQ.

Table 130 Output parameters for DAQdigitalTriggerConfig

Output Parameter name	Description
errorOut	See Description of SD1 Error IDs.

Syntax

Table 131 API syntax for DAQdigitalTriggerConfig

Language	Syntax
C	`int SD_AIN_DAQdigitalTriggerConfig(int moduleID, int nDAQ, int triggerSource, int triggerBehavior);`
C++	`int SD_AIN::DAQdigitalTriggerConfig(int nDAQ, int triggerSource, int triggerBehavior);`
.NET	`int SD_AIN::DAQdigitalTriggerConfig(int nDAQ, int triggerSource, int triggerBehavior);`
Python	`SD_AIN.DAQdigitalTriggerConfig(nDAQ, triggerSource, triggerBehavior)`
HVI	Not available

5.5.14: DAQread

Description

Reads the words acquired with the selected DAQ's Data Acquisition (DAQs). It can be used only after calling the function "DAQconfig()" and when a callback function is not configured. DAQread is a blocking function released when the configured amount of words is acquired, or when the configured timeout elapses (if timeout is set to "0", DAQread waits until DAQpoints are acquired). If the timeout elapses, there may be words available, but less than the configured amount.

NOTE

The input for 'nPoints' must always be an even number. If you enter an odd number as input, the SD1 API returns the following message before continuing operation.

"Warning: DAQ supports only even number of 'DAQpointsPerCycle'. The input value is reduced by 1."

Parameters

Table 132 Input parameters for DAQread

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
nDAQ	DAQ Channel number.
dataBuffer	Array to be filled with acquired words.
nPoints	Size (number of words) of DAQdata.
timeOut	Timeout in ms when waiting for the amount of words specified in DAQpoints. "0" indicates infinite.

Table 133 Output parameters for DAQread

Output Parameter name	Description
dataBuffer	Array, which contains the acquired words.
DAQpoints	Count of words (DAQ data) that have been acquired.
errorOut	Returns the total number of points read. If the value returned is less than the specified input for DAQpoints, it indicates timeout has expired. If it is a negative value, see Description of SD1 Error IDs. Also, see Description of SD1 Warning IDs.

Syntax

Table 134 API syntax for DAQread

Language	Syntax
C	`int SD_AIN_DAQread(int moduleID, int nDAQ, short *dataBuffer, int nPoints, int timeOut);`
C++	`std::vector<short> SD_AIN::DAQread(int nDAQ, int nPoints, int timeOut, int &error);`
.NET	`int SD_AIN::DAQread(int nDAQ, short[] dataBuffer, int timeOut);`
Python	`SD_AIN.DAQread(self, nDAQ, nPoints, timeOut = 0)`
HVI	Not available

5.5.15: DAQstart

Description

Starts acquisition on the selected DAQ's Data Acquisition (DAQs). Acquisition starts when a trigger is received.

Parameters

Table 135 Input parameters for DAQstart

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
nDAQ	DAQ Channel number to be started.

Table 136 Output parameters for DAQstart

Output Parameter name	Description
errorOut	See Description of SD1 Error IDs.

Syntax

Table 137 API syntax for DAQstart

Language	Syntax
C	`int SD_AIN_DAQstart(int moduleID, int nDAQ);`
C++	`int SD_AIN::DAQstart(int nDAQ);`
.NET	`int SD_AIN::DAQstart(int nDAQ);`
Python	`SD_AIN.DAQstart(nDAQ)`
HVI	Refer to Table 138 HVI Action syntax for DAQstart for actions & Table 139 HVI Event syntax for DAQstart for events.

Table 138 HVI Action syntax for DAQstart

Language	Action Syntax	Parameters	Description
Python	`module.hvi.actions.daq<x>_start`	None	–
.NET	`module.Hvi.Actions.Daq<x>Start`		

Table 139 HVI Event syntax for DAQstart

Language	Event Syntax	Parameters	Description
Python	`module.hvi.events.daq<x>_empty`	None	Raises an event to the host when the DAQ is empty.
	`module.hvi.events.daq<x>_running`		Raises an event to the host whenever the DAQ is running
.NET	`module.Hvi.Events.Daq<x>Empty`	None	Raises an event to the host when the DAQ is empty.
	`module.Hvi.Events.Daq<x>Running`		Raises an event to the host whenever the DAQ is running.

where, x = Channel number in the syntax for both Actions and Events

5.5.16: DAQstartMultiple

Description Starts acquisition on the selected DAQs Data Acquisition (DAQs). Acquisition will start when a trigger is received.

Parameters

Table 140 Input parameters for DAQstartMultiple

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
DAQmask	Mask to select which DAQ channels are to be started (e.g. 0b0011 where LSB is CH1, bit 1 is CH2 and so on).

Table 141 Output parameters for DAQstartMultiple

Output Parameter name	Description
errorOut	See Description of SD1 Error IDs.

Syntax

Table 142 API syntax for DAQstartMultiple

Language	Syntax
C	`int SD_AIN_DAQstartMultiple(int moduleID, int DAQmask);`
C++	`int SD_AIN::DAQstartMultiple(int DAQmask);`
.NET	`int SD_AIN::DAQstartMultiple(int DAQmask);`
Python	`SD_AIN.DAQstartMultiple(DAQmask)`
HVI	Not available.

5.5.17: DAQstop

Description

Stops the words acquisition Data Acquisition (DAQs). After DAQstop, "DAQresume()" can be used to start acquisition again.

Parameters

Table 143 Input parameters for DAQstop

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
nDAQ	DAQ Channel number that must be stopped.

Table 144 Output parameters for DAQstop

Output Parameter name	Description
errorOut	See Description of SD1 Error IDs.

Syntax

Table 145 API syntax for DAQstop

Language	Syntax
C	`int SD_AIN_DAQstop(int moduleID, int nDAQ);`
C++	`int SD_AIN::DAQstop(int nDAQ);`
.NET	`int SD_AIN::DAQstop(int nDAQ);`
Python	`SD_AIN.DAQstop(nDAQ)`
HVI	Refer to Table 146 HVI Action syntax for DAQstop for actions.

Table 146 HVI Action syntax for DAQstop

Language	Action Syntax	Parameters	Description
Python	`module.hvi.actions.daq<x>_stop`	None	–
.NET	`module.Hvi.Actions.Daq<x>Stop`		

where, x = Channel number in the syntax for Actions

5.5.18: DAQstopMultiple

Description Stops the words acquisition on more than one Data Acquisition (DAQs) Channels.

Parameters

Table 147 Input parameters for DAQstopMultiple

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
DAQmask	Mask to select which DAQ channels are to be stopped (e.g. 0b0011 where LSB is CH1, bit 1 is CH2 and so on).

Table 148 Output parameters for DAQstopMultiple

Output Parameter name	Description
errorOut	See Description of SD1 Error IDs.

Syntax

Table 149 API syntax for DAQstopMultiple

Language	Syntax
C	`int SD_AIN_DAQstopMultiple(int moduleID, int DAQmask);`
C++	`int SD_AIN::DAQstopMultiple(int DAQmask);`
.NET	`int SD_AIN::DAQstopMultiple(int DAQmask);`
Python	`SD_AIN.DAQstopMultiple(DAQmask)`
HVI	Not available.

5.5.19: DAQpause

Description

DAQpause and DAQstop have the same functionality in the Digitizer. “DAQstop()” function should be used now. To keep compatibility, DAQpause can be referred to as DAQstop. After DAQstop is applied, “DAQresume()” can be used to continue acquisition without losing previously captured samples. If “DAQstart()” or “DAQflush()” is used, the captured samples are erased.

Parameters

Table 150 Input parameters for DAQpause

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
nDAQ	DAQ Channel number that must be paused.

Table 151 Output parameters for DAQpause

Output Parameter name	Description
errorOut	See Description of SD1 Error IDs.

Syntax

Table 152 API syntax for DAQpause

Language	Syntax
C	`int SD_AIN_DAQpause(int moduleID, int nDAQ);`
C++	`int SD_AIN::DAQpause(int nDAQ);`
.NET	`int SD_AIN::DAQpause(int nDAQ);`
Python	`SD_AIN.DAQpause(nDAQ)`
HVI	Not available.

5.5.20: DAQpauseMultiple

Description

Pauses the words acquisition on more than one Data Acquisition (DAQs). Acquisition can be resumed using "DAQresumeMultiple()".

Parameters

Table 153 Input parameters for DAQpauseMultiple

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
DAQmask	Mask to select which DAQ channels are to be paused (e.g. 0b0011 where LSB is CH1, bit 1 is CH2 and so on).

Table 154 Output parameters for DAQpauseMultiple

Output Parameter name	Description
errorOut	See Description of SD1 Error IDs.

Syntax

Table 155 API syntax for DAQpauseMultiple

Language	Syntax
C	`int SD_AIN_DAQpauseMultiple(int moduleID, int DAQmask);`
C++	`int SD_AIN::DAQpauseMultiple(int DAQmask);`
.NET	`int SD_AIN::DAQpauseMultiple(int DAQmask);`
Python	`SD_AIN.DAQpauseMultiple(DAQmask)`
HVI	Not available.

5.5.21: DAQresume

Description

Resumes acquisition on the selected Data Acquisition (DAQs).

Parameters

Table 156 Input parameters for DAQresume

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
nDAQ	DAQ Channel number that must be resumed.

Table 157 Output parameters for DAQresume

Output Parameter name	Description
errorOut	See Description of SD1 Error IDs.

Syntax

Table 158 API syntax for DAQresume

Language	Syntax
C	`int SD_AIN_DAQresume(int moduleID, int nDAQ);`
C++	`int SD_AIN::DAQresume(int nDAQ);`
.NET	`int SD_AIN::DAQresume(int nDAQ);`
Python	`SD_AIN.DAQresume(nDAQ)`
HVI	Refer to Table 159 HVI Action syntax for DAQresume for actions.

Table 159 HVI Action syntax for DAQresume

Language	Action Syntax	Parameters	Description
Python	`module.hvi.actions.daq<x>_resume`	None	–
.NET	`module.Hvi.Actions.Daq<x>Resume`		

where, x = Channel number in the syntax for Actions

5.5.22: DAQresumeMultiple

Description Resumes the words acquisition on more than one Data Acquisition (DAQs) Channels.

Parameters

Table 160 Input parameters for DAQresumeMultiple

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
DAQmask	Mask to select which DAQ channels are to be resumed (e.g. 0b0011 where LSB is CH1, bit 1 is CH2 and so on).

Table 161 Output parameters for DAQresumeMultiple

Output Parameter name	Description
errorOut	See Description of SD1 Error IDs.

Syntax

Table 162 API syntax for DAQresumeMultiple

Language	Syntax
C	`int SD_AIN_DAQresumeMultiple(int moduleID, int DAQmask);`
C++	`int SD_AIN::DAQresumeMultiple(int DAQmask);`
.NET	`int SD_AIN::DAQresumeMultiple(int DAQmask);`
Python	`SD_AIN.DAQresumeMultiple(DAQmask)`
HVI	Not available.

5.5.23: DAQflush

Description

Flushes the acquisition buffers and resets the acquisition counter included in a Data Acquisition block (“Data Acquisition (DAQs)”).

Parameters

Table 163 Input parameters for DAQflush

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
nDAQ	DAQ Channel number to be flushed.

Table 164 Output parameters for DAQflush

Output Parameter name	Description
errorOut	See Description of SD1 Error IDs.

Syntax

Table 165 API syntax for DAQflush

Language	Syntax
C	`int SD_AIN_DAQflush(int moduleID, int nDAQ);`
C++	`int SD_AIN::DAQflush(int nDAQ);`
.NET	`int SD_AIN::DAQflush(int nDAQ);`
Python	`SD_AIN.DAQflush(nDAQ)`
HVI	Refer to Table 166 HVI Action syntax for DAQflush for actions.

Table 166 HVI Action syntax for DAQflush

Language	Action Syntax	Parameters	Description
Python	`module.hvi.actions.daq<x>_flush`	None	–
.NET	`module.Hvi.Actions.Daq<x>Flush`		

where, x = Channel number in the syntax for Actions

5.5.24: DAQflushMultiple

Description

Flushes the acquisition buffers and resets the acquisition counters included in a Data Acquisition block (“Data Acquisition (DAQs)”).

Parameters

Table 167 Input parameters for DAQflushMultiple

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
DAQmask	Mask to select which DAQ channels are to be flushed (e.g. 0b0011 where LSB is CH1, bit 1 is CH2 and so on).

Table 168 Output parameters for DAQflushMultiple

Output Parameter name	Description
errorOut	See Description of SD1 Error IDs.

Syntax

Table 169 API syntax for DAQflushMultiple

Language	Syntax
C	`int SD_AIN_DAQflushMultiple(int moduleID, int DAQmask);`
C++	`int SD_AIN::DAQflushMultiple(int DAQmask);`
.NET	`int SD_AIN::DAQflushMultiple(int DAQmask);`
Python	`SD_AIN.DAQflushMultiple(DAQmask)`
HVI	Not available.

5.5.25: DAQnPoints

Description Returns the counter for the DAQs.

Parameters

Table 170 Input parameters for DAQnPoints

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
nDAQ	DAQ Channel number.

Table 171 Output parameters for DAQnPoints

Output Parameter name	Description
errorOut	See Description of SD1 Error IDs.

Syntax

Table 172 API syntax for DAQnPoints

Language	Syntax
C	`int SD_AIN_DAQnPoints(int moduleID, int nDAQ);`
C++	`int SD_AIN::DAQnPoints(int nDAQ);`
.NET	`int SD_AIN::DAQnPoints(int nDAQ);`
Python	`SD_AIN.DAQnPoints(nDAQ)`
HVI	Not available.

5.5.26: DAQtrigger

Description

Triggers the acquisition of words in the selected DAQs ("Data Acquisition (DAQs)") provided that they are configured for VI/HVI Trigger.

Parameters

Table 173 Input parameters for DAQtrigger

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
nDAQ	DAQ Channel number to be triggered.

Table 174 Output parameters for DAQtrigger

Output Parameter name	Description
errorOut	See Description of SD1 Error IDs.

Syntax

Table 175 API syntax for DAQtrigger

Language	Syntax
C	`int SD_AIN_DAQtrigger(int moduleID, int nDAQ);`
C++	`int SD_AIN::DAQtrigger(int nDAQ);`
.NET	`int SD_AIN::DAQtrigger(int nDAQ);`
Python	`SD_AIN.DAQtrigger(nDAQ)`
HVI	Refer to Table 176 HVI Action syntax for DAQtrigger for actions & Table 177 HVI Event syntax for DAQtrigger for events.

Table 176 HVI Action syntax for DAQtrigger

Language	Action Syntax	Parameters	Description
Python	`module.hvi.actions.daq<x>_trigger`	None	–
.NET	`module.Hvi.Actions.Daq<x>Trigger`		

Table 177 HVI Event syntax for DAQtrigger

Language	Event Syntax	Parameters	Description
Python	`module.hvi.events.daq<x>_trigger_loopback`	None	This is a loopback of the DAQxTrigger action, it is a copy of the action. This can be used for debugging purposes.
.NET	`module.Hvi.Events.Daq<x>TriggerLoopback`		

where, x = Channel number in the syntax for both Actions and Events

5.5.27: DAQtriggerMultiple

Description

Triggers the acquisition of words in more than one DAQs (Data Acquisition (DAQs)) provided that they are configured for VI/HVI Trigger.

Parameters

Table 178 Input parameters for DAQtriggerMultiple

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
DAQmask	Mask to select which DAQ channels are to be triggered (e.g. 0b0011 where LSB is CH1, bit 1 is CH2 and so on).

Table 179 Output parameters for DAQtriggerMultiple

Output Parameter name	Description
errorOut	See Description of SD1 Error IDs.

Syntax

Table 180 API syntax for DAQtriggerMultiple

Language	Syntax
C	`int SD_AIN_DAQtriggerMultiple(int moduleID, int DAQmask);`
C++	`int SD_AIN::DAQtriggerMultiple(int DAQmask);`
.NET	`int SD_AIN::DAQtriggerMultiple(int DAQmask);`
Python	`SD_AIN.DAQtriggerMultiple(DAQmask)`
HVI	Not available.

5.5.28: DAQtriggerConfig

Description

Configures DAQ for the selected DAQ Trigger.

Parameters

Table 181 Input parameters for DAQtriggerConfig

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
nDAQ	DAQ Channel number.
digitalTriggerMode	Digital Trigger Mode. Refer to Table 25 DAQ Trigger Mode options.
digitalTriggerSource	Digital Trigger source. Refer to Table 26 External Hardware Digital Trigger Source for the DAQ.
analogTriggerMask	Enter Channel Mask ID to enable analog triggering on specified channel.

Table 182 Output parameters for DAQtriggerConfig

Output Parameter name	Description
errorOut	See Description of SD1 Error IDs.

Syntax

Table 183 API syntax for DAQtriggerConfig

Language	Syntax
C	`int SD_AIN_DAQtriggerConfig(int moduleID, int nDAQ, int digitalTriggerMode, int digitalTriggerSource, int analogTriggerMask);`
C++	`int SD_AIN::DAQtriggerConfig(int nDAQ, int digitalTriggerMode, int digitalTriggerSource, int analogTriggerMask);`
.NET	`int SD_AIN::DAQtriggerConfig(int nDAQ, int digitalTriggerMode, int digitalTriggerSource, int analogTriggerMask);`
Python	`SD_AIN.DAQtriggerConfig(nDAQ, digitalTriggerMode, digitalTriggerSource, analogTriggerMask)`
HVI	Not available

5.5.29: DAQcounterRead

Description

Reads the number of words acquired by the selected DAQ (Data Acquisition (DAQs)) since the last call to "DAQflush()" or DAQ.

Parameters

Table 184 Input parameters for DAQcounterRead

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
nDAQ	DAQ Channel number, whose counter is read.

Table 185 Output parameters for DAQcounterRead

Output Parameter name	Description
counter	Value of the DAQ counter or a negative number for errors.
errorOut	See Description of SD1 Error IDs.

Syntax

Table 186 API syntax for DAQcounterRead

Language	Syntax
C	`int SD_AIN_DAQcounterRead(int moduleID, int nDAQ);`
C++	`int SD_AIN::DAQcounterRead(int nDAQ);`
.NET	`int SD_AIN::DAQcounterRead(int nDAQ);`
Python	`SD_AIN.DAQcounterRead(nDAQ)`
HVI	Not available.

5.5.30: triggerIOconfig

Description Configures the trigger connector/line direction (Working with I/O Triggers).

Parameters

Table 187 Input parameters for triggerIOconfig

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
direction	Output (0) or Input (1). Refer to Table 28 Trigger Direction options and corresponding values.
port	Applicable for M33xxA COMBO cards only.

Table 188 Output parameters for triggerIOconfig

Output Parameter name	Description
errorOut	See Description of SD1 Error IDs.

Syntax

Table 189 API syntax for triggerIOconfig (on M310xA DIG modules)

Language	Syntax
C	`int SD_AIN_triggerIOconfig(int moduleID, int direction);`
C++	`int SD_AIN::triggerIOconfig(int direction);`
.NET	`int SD_AIN::triggerIOconfig(int direction);`
Python	`SD_AIN.triggerIOconfig(direction)`
HVI	Not available

Table 190 API syntax for triggerIOconfig (on M330xA Combo modules)

Language	Syntax
C	`int SD_AIO_triggerIOconfig(int moduleID, int port, int direction);`
C++	`int SD_AIO::triggerIOconfig(int port, int direction);`
.NET	`int SD_AIO::triggerIOconfig(int port, int direction);`
Python	`SD_AIO.triggerIOconfig(port, direction)`
HVI	Not available

5.5.31: triggerIOwrite

Description Sets the trigger output to be ON or OFF. The trigger must be configured as output using triggerIOconfig().

Parameters

Table 191 Input parameters for triggerIOwrite

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
value	Trigger output value: 1 (ON), 0 (OFF).
syncMode	Sampling/synchronization mode. '0' for immediate triggers or '1' to synchronize trigger to nearest CLK edge. Refer to Table 29 Sync mode options and corresponding values.
port	Applicable for M33xxA COMBO cards only.

Table 192 Output parameters for triggerIOwrite

Output Parameter name	Description
errorOut	See Description of SD1 Error IDs.

Syntax

Table 193 API syntax for triggerIOwrite (on M310xA DIG modules)

Language	Syntax
C	`int SD_AIN_triggerIOwrite(int moduleID, int value, int syncMode);`
C++	`int SD_AIN::triggerIOwrite(int value);` `int SD_AIN::triggerIOwrite(int value, int syncMode);`
.NET	`int SD_AIN::triggerIOwrite(int value);` `int SD_AIN::triggerIOwrite(int value, int syncMode);`
Python	`SD_AIN.triggerIOwrite(value, syncMode = 1)`
HVI	Not available

Table 194 API syntax for triggerIOwrite (on M330xA Combo modules)

Language	Syntax
C	`int SD_AIO_triggerIOwrite(int moduleID, int port, int value, int syncMode);`
C++	`int SD_AIO::triggerIOwrite(int port, int value, int syncMode);`
.NET	`int SD_AIO::triggerIOwrite(int port, int value, int syncMode);`
Python	`SD_AIO.triggerIOwrite(port, value, syncMode = 1)`
HVI	Not available

5.5.32: triggerIOread

Description Reads the trigger input (Working with I/O Triggers).

Parameters

Table 195 Input parameters for triggerIOread

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
port	Applicable for M33xxA COMBO cards only.

Table 196 Output parameters for triggerIOread

Output Parameter name	Description
value	Trigger output value: 1 (ON), 0 (OFF). Negative numbers indicate an error. See Description of SD1 Error IDs.
errorOut	See Description of SD1 Error IDs.

Syntax

Table 197 API syntax for triggerIOread (on M310xA DIG modules)

Language	Syntax
C	`int SD_AIN_triggerIORead(int moduleID);`
C++	`int SD_AIN::triggerIORead();`
.NET	`int SD_AIN::triggerIORead();`
Python	`SD_AIN.triggerIORead()`
HVI	Not available

Table 198 API syntax for triggerIOread (on M330xA Combo modules)

Language	Syntax
C	`int SD_AIO_triggerIORead(int moduleID, int port);`
C++	`int SD_AIO::triggerIORead(int port);`
.NET	`int SD_AIO::triggerIORead(int port);`
Python	`SD_AIO.triggerIORead(int port)`
HVI	Not available

5.5.33: clockSetFrequency

Description

Returns the frequency of the internal CLKsync signal in Hz. (See “CLKsys” in FlexCLK Technology (models w/ variable sampling rate)).

NOTE This function is only usable for modules with the variable clock Option CLV. Note that the CLV option is currently not supported in the M310xA/M330xA modules.

Parameters

Table 199 Input parameters for clockSetFrequency

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
frequency	Frequency in Hz. See Data sheet for complete specifications.
mode	Operation mode of the variable clock system. Refer to Table 30 Variable clock system operation mode options and corresponding values.
port	Applicable for M33xxA COMBO cards only.

Table 200 Output parameters for clockSetFrequency

Output Parameter name	Description
CLKsysFreq*	It returns the real frequency applied to the hardware in Hz. It may differ from the desired frequency due to hardware frequency resolution. Negative numbers indicate an error. See Description of SD1 Error IDs.
errorOut	See Description of SD1 Error IDs.

*In Keysight Programming Libraries v.1.57.61 or older, clockSetFrequency returns CLKsyncFreq, the frequency of the internal CLKsync in Hz.

Syntax

Table 201 API syntax for clockSetFrequency (on M310xA DIG modules)

Language	Syntax
C	`double SD_AIN_clockSetFrequency(int moduleID, double frequency, int mode);`
C++	`double SD_AIN::clockSetFrequency(double frequency, int mode);`
.NET	`double SD_AIN::clockSetFrequency(double frequency, int mode);`
Python	`SD_AIN.clockSetFrequency(frequency, mode)`
HVI	Not available

Table 202 API syntax for clockSetFrequency (on M330xA Combo modules)

Language	Syntax
C	`double SD_AIO_clockSetFrequency(int moduleID, int port, double frequency, int mode);`
C++	`double SD_AIO::clockSetFrequency(int port, double frequency, int mode);`
.NET	`double SD_AIO::clockSetFrequency(int port, double frequency, int mode);`
Python	`SD_AIO.clockSetFrequency(port, frequency, mode)`
HVI	Not available

5.5.34: clockGetFrequency

Description

Returns the value (in units of Hz) of the module's sample rate frequency (the real hardware's clock frequency). It may differ from the frequency set with the function "clockSetFrequency()", due to the hardware frequency resolution.

Parameters

Table 203 Input parameters for clockGetFrequency

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
port	Applicable for M33xxA COMBO cards only.

Table 204 Output parameters for clockGetFrequency

Output Parameter name	Description
CLKsysFreq*	It returns the real frequency applied to the hardware in Hz. It may differ from the desired frequency due to hardware frequency resolution. Negative numbers indicate an error. See Description of SD1 Error IDs.
errorOut	See Description of SD1 Error IDs.

*In Keysight Programming Libraries v.1.57.61 or older, clockSetFrequency returns CLKsyncFreq, the frequency of the internal CLKsync in Hz.

Syntax

Table 205 API syntax for clockGetFrequency (on M310xA DIG modules)

Language	Syntax
C	`double SD_AIN_clockGetFrequency(int moduleID);`
C++	`double SD_AIN::clockGetFrequency();`
.NET	`double SD_AIN::clockGetFrequency();`
Python	`SD_AIN.clockGetFrequency()`
HVI	Not available

Table 206 API syntax for clockGetFrequency (on M330xA Combo modules)

Language	Syntax
C	`double SD_AIO_clockGetFrequency(int moduleID, int port);`
C++	`double SD_AIO::clockGetFrequency(int port);`
.NET	`double SD_AIO::clockGetFrequency(int port);`
Python	`SD_AIO.clockGetFrequency(port)`
HVI	Not available

5.5.35: clockGetSyncFrequency

Description Returns the frequency of the internal CLKsync signal, in units of Hz.

Parameters

Table 207 Input parameters for clockGetSyncFrequency

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
port	Applicable for M33xxA COMBO cards only.

Table 208 Output parameters for clockGetSyncFrequency

Output Parameter name	Description
CLKsysFreq*	It returns the real frequency applied to the hardware in Hz. It may differ from the desired frequency due to hardware frequency resolution. Negative numbers indicate an error. See Description of SD1 Error IDs.
errorOut	See Description of SD1 Error IDs.

*In Keysight Programming Libraries v.1.57.61 or older, clockSetFrequency returns CLKsyncFreq, the frequency of the internal CLKsync in Hz.

Syntax

Table 209 API syntax for clockGetSyncFrequency (on M310xA DIG modules)

Language	Syntax
C	`int SD_AIN_clockGetSyncFrequency(int moduleID);`
C++	`double SD_AIN::clockGetSyncFrequency();`
.NET	`double SD_AIN::clockGetSyncFrequency();`
Python	`SD_AIN.clockGetSyncFrequency()`
HVI	Not available

Table 210 API syntax for clockGetSyncFrequency (on M330xA Combo modules)

Language	Syntax
C	`double SD_AIO_clockGetSyncFrequency(int moduleID, int port);`
C++	`double SD_AIO::clockGetSyncFrequency(int port);`
.NET	`double SD_AIO::clockGetSyncFrequency(int port);`
Python	`SD_AIO.clockGetSyncFrequency(port)`
HVI	Not available

5.5.36: clockIOconfig

Description Configures the operation of the clock output connector. See CLK Output Options.

Parameters

Table 211 Input parameters for clockIOconfig

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
clockConfig	Enable (value 1) or disable (value 0) the Clock Connector. Refer to Table 23 Clock connector options and corresponding values.
port	Applicable for M33xxA COMBO cards only.

Table 212 Output parameters for clockIOconfig

Output Parameter name	Description
errorOut	See Description of SD1 Error IDs.

Syntax

Table 213 API syntax for clockIOconfig (on M310xA DIG modules)

Language	Syntax
C	`int SD_AIN_clockIOconfig(int moduleID, int clockConfig);`
C++	`int SD_AIN::clockIOconfig(int clockConfig);`
.NET	`int SD_AIN::clockIOconfig(int clockConfig);`
Python	`SD_AIN.clockIOconfig(clockConfig)`
HVI	Not available

Table 214 API syntax for clockIOconfig (on M330xA Combo modules)

Language	Syntax
C	`int SD_AIO_clockIOconfig(int moduleID, int port, int clockConfig);`
C++	`int SD_AIO::clockIOconfig(int port, int clockConfig);`
.NET	`int SD_AIO::clockIOconfig(int port, int clockConfig);`
Python	`SD_AIO.clockIOconfig(port, clockConfig)`
HVI	Not available

5.5.37: clockResetPhase

Description

Sets the module in a synchronous state, waiting for the first trigger to reset the phase of the internal clocks CLKsync and CLKsys (see “Working with Clock System” on page 20).

Parameters

Table 215 Input parameters for clockResetPhase

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
triggerBehavior	DAQ external trigger behavior. Refer to Table 27 Trigger behavior for the DAQ.
PXItrigger	PXI trigger number. Refer to Table 26 External Hardware Digital Trigger Source for the DAQ.
skew	Skew between PXI CLK10 and CLKsync in multiples of 10 ns.
port	Applicable for M33xxA COMBO cards only.

Table 216 Output parameters for clockResetPhase

Output Parameter name	Description
errorOut	See Description of SD1 Error IDs.

Syntax

Table 217 API syntax for clockResetPhase (on M310xA DIG modules)

Language	Syntax
C	`int SD_AIN_clockResetPhase(int moduleID, int triggerBehavior, int PXItrigger, double skew);`
C++	`int SD_AIN::clockResetPhase(int triggerBehavior, int PXItrigger, double skew);`
.NET	`int SD_AIN::clockResetPhase(int triggerBehavior, int PXItrigger, double skew);`
Python	`SD_AIN.clockResetPhase(triggerBehavior, PXItrigger, skew)`
HVI	Not available

Table 218 API syntax for clockResetPhase (on M330xA Combo modules)

Language	Syntax
C	`int SD_AIO_clockResetPhase(int moduleID, int port, int triggerBehavior, int PXItrigger, double skew);`
C++	`int SD_AIO::clockResetPhase(int port, int triggerBehavior, int PXItrigger, double skew);`
.NET	`int SD_AIO::clockResetPhase(int port, int triggerBehavior, int PXItrigger, double skew);`
Python	`SD_AIO.clockResetPhase(port, triggerBehavior, PXItrigger, skew)`
HVI	Not available

5.5.38: DAQbufferPoolConfig

Description Configures buffer pool that will be filled with the data of the channel to be transferred to PC.

Parameters

Table 219 Input parameters for DAQbufferPoolConfig

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
nDAQ	DAQ Channel number to be configured.
dataBuffer	Buffer to use in buffer pool. Has to be created and released by user.
nPoints	Size of dataBuffer buffer.
timeOut	Maximum time used to fill each buffer. If '0', timeout is not set and buffer will not be delivered to the user until it's full.
callbackFunction	Callback that will be called each time a buffer is ready. It can be null. If callback is set, DAQbufferGet is useless.
callbackUserObj	Pointer to user object that will be passed in the callback as parameter called userObject; it can be null.

Table 220 Output parameters for DAQbufferPoolConfig

Output Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier or a negative number for errors.
errorOut	See Description of SD1 Error IDs.

Syntax

Table 221 API syntax for DAQbufferPoolConfig

Language	Syntax
C	`int SD_AIN_DAQbufferPoolConfig(int moduleID, int nDAQ, short* dataBuffer, int nPoints, int timeOut, callbackEventPtr callbackFunction, void *callbackUserObj);`
C++	`int SD_AIN::DAQbufferPoolConfig(int nDAQ, std::vector<short> &dataBuffer, int timeOut);`
.NET	`int SD_AIN::DAQbufferPoolConfig(int nDAQ, short[] dataBuffer, int timeOut);`
Python	`SD_AIN.DAQbufferPoolConfig(nDAQ, nPoints, timeOut)`
HVI	Not available

5.5.39: DAQbufferAdd

Description Adds an additional buffer to the channel's previously configured pool.

Parameters

Table 222 Input parameters for DAQbufferAdd

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
nDAQ	DAQ Channel number.
dataBuffer	Buffer to use in buffer pool. Has to be created and released by user.
nPoints	Size of dataBuffer buffer.

Table 223 Output parameters for DAQbufferAdd

Output Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier or a negative number for errors.
errorOut	See Description of SD1 Error IDs.

Syntax

Table 224 API syntax for DAQbufferAdd

Language	Syntax
C	`int SD_AIN_DAQbufferAdd(int moduleID, int nDAQ, short* dataBuffer, int nPoints);`
C++	`int SD_AIN::DAQbufferAdd(int nDAQ, std::vector<short> &dataBuffer);`
.NET	`int SD_AIN::DAQbufferAdd(int nDAQ, short[] dataBuffer);`
Python	Not available
HVI	Not available

5.5.40: DAQbufferGet

Description

Gets a filled buffer from the channel buffer pool. User has to call DAQbufferAdd() with this buffer to tell the pool that the buffer can be used again.

Parameters

Table 225 Input parameters for DAQbufferGet

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
nDAQ	DAQ Channel number.

Table 226 Output parameters for DAQbufferGet

Output Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier or a negative number for errors.
buffer	Buffer obtained.
readPointsOut	Number of points of the returned buffer.
errorOut	See Description of SD1 Error IDs.

Syntax

Table 227 API syntax for DAQbufferGet

Language	Syntax
C	`short* SD_AIN_DAQbufferGet(int moduleID, int nDAQ, int &readPointsOut, int &errorOut);`
C++	`short* SD_AIN::DAQbufferGet(int nDAQ, int &readPointsOut, int &error);`
.NET	`short[] SD_AIN::DAQbufferGet(int nDAQ, out int readPointsOut, out int errorOut);`
Python	`SD_AIN.DAQbufferGet(nDAQ) [*Note that the returned data array is a NumPY array*]`
HVI	Not available

5.5.41: DAQbufferPoolRelease

Description

Releases the channel buffer pool and its resources. After this call, you must call "DAQbufferRemove()" consecutively to get all buffers back and release them.

Parameters

Table 228 Input parameters for DAQbufferPoolRelease

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
nDAQ	DAQ Channel number

Table 229 Output parameters for DAQbufferPoolRelease

Output Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier or a negative number for errors.
errorOut	See Description of SD1 Error IDs.

Syntax

Table 230 API syntax for DAQbufferPoolRelease

Language	Syntax
C	`int SD_AIN_DAQbufferPoolRelease(int moduleID, int nDAQ);`
C++	`int SD_AIN::DAQbufferPoolRelease(int nDAQ);`
.NET	`int SD_AIN::DAQbufferPoolRelease(int nDAQ);`
Python	`SD_AIN.DAQbufferPoolRelease(nDAQ)`
HVI	Not available

5.5.42: DAQbufferRemove

Description

Ask for a buffer to be removed from the channel buffer pool. If NULL pointer is returned, no more buffers remains in buffer pool. Returned buffer is a previously added buffer from user and user has to release/delete it.

Parameters

Table 231 Input parameters for DAQbufferRemove

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
nDAQ	DAQ Channel number

Table 232 Output parameters for DAQbufferRemove

Output Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier or a negative number for errors.
buffer	The obtained buffer.
errorOut	See Description of SD1 Error IDs.

Syntax

Table 233 API syntax for DAQbufferRemove

Language	Syntax
C	`short* SD_AIN_DAQbufferRemove(int moduleID, int nDAQ);`
C++	`short* SD_AIN::DAQbufferRemove(int nDAQ);`
.NET	`short[] SD_AIN::DAQbufferRemove(int nDAQ);`
Python	Not available (DAQbuffer functions are not accessible)
HVI	Not available

5.5.43: DAQtriggerExternalConfig

Description

Configures DAQ for the external trigger source.

Parameters

Table 234 Input parameters for DAQtriggerExternalConfig

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
nDAQ	DAQ Channel number.
externalSource	External trigger source for the DAQ. Refer to Table 26 External Hardware Digital Trigger Source for the DAQ.
triggerBehavior	Refer to Table 27 Trigger behavior for the DAQ.
sync	Possible values are: SYNC_NONE = 0; SYNC_CLK10 = 1;

Table 235 Output parameters for DAQtriggerExternalConfig

Output Parameter name	Description
errorOut	See Description of SD1 Error IDs.

Syntax

Table 236 API syntax for DAQtriggerExternalConfig

Language	Syntax
C	`int SD_AIN_DAQtriggerExternalConfig(int moduleID, int nDAQ, int externalSource, int triggerBehavior, int sync);`
C++	`int SD_AIN::DAQtriggerExternalConfig(int nDAQ, int externalSource, int triggerBehavior, int sync);`
.NET	`int SD_AIN::DAQtriggerExternalConfig(int nDAQ, int externalSource, int triggerBehavior);` `int SD_AIN::DAQtriggerExternalConfig(int nDAQ, int externalSource, int triggerBehavior, int sync);`
Python	`SD_AIN.DAQtriggerExternalConfig(nDAQ, externalSource, triggerBehavior, sync=0)`
HVI	Not available.

5.5.44: FFT

Description Calculates the FFT of data captured by "DAQread()" for the selected channel.

Parameters **Table 237 Input parameters for FFT**

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
channel	Input Channel number.
data	Data previously acquired by DAQread().
size	Input data size.
resultSize	Size of the output buffers (module and phase).
dB	Scale (dB or linear).
windowType	Windowing option. Refer to Table 34 Window Types Used in FFT Functions.

Table 238 Output parameters for FFT

Output Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier or a negative number for errors.
result	Module (magnitude) of the FFT.
resultPhase	Phase of the FFT.
errorOut	See Description of SD1 Error IDs.

Syntax **Table 239 API syntax for FFT**

Language	Syntax
C	`int SD_AIN_FFT(int moduleID, int channel, short *data, int size, double *result, int resultSize, double *resultPhase, bool dB, int windowType);`
C++	`int SD_AIN::FFT(int channel, std::vector<short> data, std::vector<double> &result, bool dB, int windowType);` `int SD_AIN::FFT(int channel, std::vector<short> data, std::vector<double> &result, std::vector<double> &resultPhase, bool dB, int windowType);`
.NET	`int SD_AIN::FFT(int channel, short[] data, out double[] result, bool dB, int windowType);` `int SD_AIN::FFT(int channel, short[] data, out double[] result, out double[] resultPhase, bool dB, int windowType);`
Python	`SD_AIN.FFT(channel, data, dB, windowType) [*Note that the returned data array is a NumPY array*].`
HVI	Not available

5.5.45: voltsToInt

Description Helper method to convert voltage from ‘double’ / 'float' to ‘integer’. This conversion method is intended to be used with HVI native instructions only, when assigning literals to the registers for ‘threshold’ in channelTriggerConfig() function.

Parameters

Table 240 Input parameters for voltsToInt

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
nDAQ	DAQ Channel number
volts	Input threshold

Table 241 Output parameters for voltsToInt

Output Parameter name	Description
voltOut	Converted threshold in integer
errorOut	See Description of SD1 Error IDs.

Syntax

Table 242 API syntax for voltsToInt

Language	Syntax
C	`int SD_AIN_voltsToInt(int moduleID, int nDAQ, double volts);`
C++	`int SD_AIN::voltsToInt(int nDAQ, double volts);`
.NET	`int SD_AIN::voltsToInt(int channel, double volts);`
Python	`SD_AIN.voltsToInt(channel, volts)`
HVI	Not available

Section 5.6: SD_Module functions (specific to Pathwave FPGA)

5.6.1: FPGAgetSandBoxRegister

Description Gets sandbox register from name, if programmable FPGA is loaded from **.k7z* file.

Parameters

Table 243 Input parameters for FPGAgetSandBoxRegister

Input Parameter name	Description
moduleID	Module identifier, returned by open.
registerName	Name of the register.

Table 244 Output parameters for FPGAgetSandBoxRegister

Output Parameter name	Description
registerId	Register ID corresponding to the register name.
SD_SandBoxRegisterObj	SD_SandBoxRegister object.
errorOut	See Description of SD1 Error IDs.

Syntax

Table 245 API syntax for FPGAgetSandBoxRegister

Language	Syntax
C	`int SD_Module_FPGAgetRegisterId(int moduleID, const char* registerName);`
C++	`SD_SandBoxRegister SD_Module::FPGAGetSandBoxRegister(std::string registerName);`
.NET	`SD_SandBoxRegister SD_Module::FPGAGetSandBoxRegister(string registerName);`
Python	`SD_Module.FPGAgetSandBoxRegister(registerName)`
HVI	Not available

5.6.2: FPGAgetSandBoxRegisters

Description Gets list of the sandbox registers if programmable FPGA is loaded from **.k7z* file.

Parameters

Table 246 Input parameters for FPGAgetSandBoxRegisters

Input Parameter name	Description
moduleID	Module identifier, returned by open.
count / numberOfRegister	Number of Registers.

Table 247 Output parameters for FPGAgetSandBoxRegisters

Output Parameter name	Description
ListOut	List of SD_SandBoxRegister or null pointer.
errorOut	See Description of SD1 Error IDs.

Syntax

Table 248 API syntax for FPGAgetSandBoxRegisters

Language	Syntax
C	`int SD_Module_FPGAgetRegisterIds(int moduleID, int* registerIds, int numberOfRegister);`
C++	`std::list<SD_SandBoxRegister> SD_Module::FPGAGetSandBoxRegisters(int count);`
.NET	`List<SD_SandBoxRegister> SD_Module::FPGAGetSandBoxRegisters(int count);`
Python	`SD_Module.FPGAgetSandBoxRegisters(count)`
HVI	Not available

5.6.3: FPGAload

Description

Loads the bit file (*.k7z), which is generated using PathWave FPGA, onto the respective module's FPGA.

Parameters

Table 249 Input parameters for FPGAload

Input Parameter name	Description
moduleID	Module identifier, returned by open.
fileName	Name of the file. The file format should be (*.*k7z*)

Table 250 Output parameters for FPGAload

Output Parameter name	Description
errorOut	Returns SD_ERROR_NONE on success. See Description of SD1 Error IDs.

Syntax

Table 251 API syntax for FPGAload

Language	Syntax
C	`int SD_Module_FPGAload(int moduleID, const char *fileName);`
C++	`int SD_Module::FPGAload(std::string fileName);`
.NET	`int SD_Module::FPGAload(string fileName);`
Python	`SD_Module.FPGAload(fileName)`
HVI	Not available

5.6.4: FPGAreset

Description Sends a reset signal to the FPGA sandbox region.

Parameters

Table 252 Input parameters for FPGAreset

Input Parameter name	Description
moduleID	Module identifier, returned by open.
mode	SD_ResetMode values. Refer to Table 31 SD_ResetMode attribute values.

Table 253 Output parameters for FPGAreset

Output Parameter name	Description
errorOut	See Description of SD1 Error IDs.

Syntax

Table 254 API syntax for FPGAreset

Language	Syntax
C	`int SD_Module_FPGAreset(int moduleID, int mode);`
C++	`int SD_Module::FPGAreset(int mode);`
.NET	`int SD_Module::FPGAreset(SD_ResetMode mode);`
Python	`SD_Module.FPGAreset(mode)`
HVI	Not available

5.6.5: FPGATriggerConfig

Description Configures PXI or FrontPanel triggers coming in or going out from sandbox region.

Parameters

Table 255 Input parameters for FPGATriggerConfig

Input Parameter name	Description
moduleID	Module identifier, returned by open.
externalSource	AWG external trigger source.
direction	Refer to Table 32 FPGATriggerDirection attribute values.
polarity	Refer to Table 33 TriggerPolarity attribute values.
syncMode	Set to '0' for immediate trigger or '1' to synchronize with nearest CLK edge.
delay5Tclk	Delay to add before the marker pulse = delay x TCLKsys x 5 (syncMode selects the start point of the marker, after which the delay is added).

Table 256 Output parameters for FPGATriggerConfig

Output Parameter name	Description
errorOut	See Description of SD1 Error IDs.

Syntax

Table 257 API syntax for FPGATriggerConfig

Language	Syntax
C	`int SD_Module_FPGATriggerConfig(int moduleID, int externalSource, int direction, int polarity, int syncMode, int delay5Tclk);`
C++	`int SD_Module::FPGATriggerConfig(int externalSource, FpgaTriggerDirection direction, TriggerPolarity polarity, int syncMode, int delay5Tclk);`
.NET	`int SD_Module::FPGATriggerConfig(int externalSource, FpgaTriggerDirection direction, TriggerPolarity polarity, int syncMode, int delay5Tclk);`
Python	`SD_Module.FPGATriggerConfig(externalSource, direction, polarity, syncMode, delay5Tclk)`
HVI	Not available

5.6.6: User FPGA HVI Actions/Events

Description Defines the HVI Actions & Events, which are available for sending information to and receiving information from the FPGA sandbox.

Syntax

Table 258 HVI Action syntax for User FPGA HVI Actions

Language	Action Syntax	Parameters	Description
Python	`module.hvi.actions.user_fpga_<x>`	None	Customizable action that can relay and receive information from the FPGA sandbox.
.NET	`module.Hvi.Actions.FpgaUser<x>`		

where, 0 <= x = 7

Table 259 HVI Event syntax for User FPGA HVI Events

Language	Event Syntax	Parameters	Description
Python	`module.hvi.events.user_fpga_<x>`	None	Customizable event that can relay and receive information from the FPGA sandbox.
.NET	`module.Hvi.Events.FpgaUser<x>`		

where, 0 <= x = 7

5.6.7: Module HVI Engine

Description Enables HVI Engines located in the M3xxxA module's FPGA to control various functions of the M3xxxA modules, the timing of operations and execution of the HVI sequences.

Syntax **Table 260 HVI Engine syntax for M3xxxA modules**

Language	Action Syntax	Parameters	Description
Python	`module.hvi.engines.main_engine`	None	Enables HVI Engines located in the M3xxxA module's FPGA.
.NET	`module.Hvi.Engines.MainEngine`		

5.6.8: Module HVI Triggers

Description Activates PXI or Front Panel triggers that an HVI sequence can read/write or use to turn the trigger ON or OFF, write to the trigger line, get the hardware name or ID of the trigger resource, and configure settings for the trigger.

Syntax

Table 261 HVI Trigger syntax for M3xxxA modules

Language	Trigger Syntax	Value for trigger name	Description
Python	`module.hvi.triggers.<trigger_name>`	`pxi_<x>`	Activates the defined PXI trigger line.
		`front_panel_1`	Activates the Front Panel Trigger 1.
		`front_panel_2*`	Activates the Front Panel Trigger 2 on Combo cards.
.NET	`module.Hvi.Triggers.<TriggerName>`	`Pxi<x>`	Activates the defined PXI trigger line.
		`FrontPanel1`	Activates the Front Panel Trigger 1.
		`FrontPanel2*`	Activates the Front Panel Trigger 2 on Combo cards.

where, 0 <= x = 7

*Available for Combo modules only

Section 5.7: SD_SandboxRegister functions

5.7.1: readRegisterBuffer

Description Reads data buffer from a sandbox register bank or memory map.

Parameters **Table 262 Input parameters for readRegisterBuffer**

Input Parameter name	Description
moduleID	Module identifier, returned by open.
registerid	ID of the sandbox register.
indexOffset	Starting index of sandbox register.
bufferSize	Number of 32-bit words to write.
addressMode	Addressing modes: AUTOINCREMENT = 0, FIXED = 1
accessMode	Access Modes: NONDMA = 0, DMA=1

Table 263 Output parameters for readRegisterBuffer

Output Parameter name	Description
buffer	Buffer with the specified size.
errorOut	See Description of SD1 Error IDs.

Syntax **Table 264 API syntax for readRegisterBuffer**

Language	Syntax
C	`int SD_Module_FPGAreadRegisterBuffer(int moduleID, int RegisterId, int indexOffset, int *buffer, int bufferSize,int addressMode, int accessMode);`
C++	`std::vector<int> SD_SandBoxRegister::readRegisterBuffer(int indexOffset, int bufferSize, SD_AddressingMode addressMode, SD_AccessMode accessMode, int &error);`
.NET	`int SD_SandBoxRegister::readRegisterBuffer(int indexOffset, int[] buffer, SD_AddressingMode addressMode, SD_AccessMode accessMode);`
Python	`SD_SandBoxRegister.readRegisterBuffer(indexOffset, bufferSize, addressMode, accessMode)`
HVI	Not available

5.7.2: readRegisterInt32

Description Reads int32 data from a sandbox register bank or memory map.

Parameters

Table 265 Input parameters for readRegisterInt32

Input Parameter name	Description
moduleID	Module identifier, returned by open.
RegisterId	ID of the sandbox register.

Table 266 Output parameters for readRegisterInt32

Output Parameter name	Description
data	Register value.
errorOut	See Description of SD1 Error IDs.

Syntax

Table 267 API syntax for readRegisterInt32

Language	Syntax
C	`int SD_Module_FPGAreadRegisterInt32(int moduleID, int RegisterId, int &data);`
C++	`int SD_SandBoxRegister::readRegisterInt32();`
.NET	`int SD_SandBoxRegister::readRegisterInt32();`
Python	`SD_SandBoxRegister.readRegisterInt32()`
HVI	Not available

5.7.3: writeRegisterBuffer

Description Writes data buffer to a sandbox register bank or memory map.

Parameters **Table 268 Input parameters for writeRegisterBuffer**

Input Parameter name	Description
moduleID	Module identifier, returned by open.
registerid	ID of the sandbox register.
indexOffset	Starting index of sandbox register.
buffer	Data buffer to be written.
bufferSize	Size of the buffer.
addressMode	Addressing modes: AUTOINCREMENT = 0, FIXED = 1
accessMode	Access Modes: NONDMA = 0, DMA=1

Table 269 Output parameters for writeRegisterBuffer

Output Parameter name	Description
errorOut	See Description of SD1 Error IDs.

Syntax **Table 270 API syntax for writeRegisterBuffer**

Language	Syntax
C	`int SD_Module_FPGAwriteRegisterBuffer(int moduleID, int registerId, int indexOffset, int *buffer, int bufferSize,int addressMode, int accessMode);`
C++	`int SD_SandBoxRegister::writeRegisterBuffer(int indexOffset, std::vector<int> buffer, SD_AddressingMode addressMode, SD_AccessMode accessMode);`
.NET	`int SD_SandBoxRegister::writeRegisterBuffer(int indexOffset, int[] buffer, SD_AddressingMode addressMode, SD_AccessMode accessMode);`
Python	`SD_SandBoxRegister.writeRegisterBuffer(self, indexOffset, buffer, addressMode, accessMode)`
HVI	Not available

5.7.4: writeRegisterInt32

Description Writes int32 data to a sandbox register bank or memory map.

Parameters **Table 271 Input parameters for writeRegisterInt32**

Input Parameter name	Description
moduleID	Module identifier, returned by open.
RegisterId	ID of the sandbox register.
data	Data to write to register.

Table 272 Output parameters for writeRegisterInt32

Output Parameter name	Description
errorOut	See Description of SD1 Error IDs.

Syntax **Table 273 API syntax for writeRegisterInt32**

Language	Syntax
C	`int SD_Module_FPGAwriteRegisterInt32(int moduleID, int registerId, int data);`
C++	`int SD_SandBoxRegister::writeRegisterInt32(int data);`
.NET	`int SD_SandBoxRegister::writeRegisterInt32(int data);`
Python	`SD_SandBoxRegister.writeRegisterInt32(data)`
HVI	Not available

5.7.5: Properties

Description Properties associated with registers.

Property Name	Description	Python / .Net	C++
Name	Register symbol name used in the PathWave FPGA design.	SD_SandBoxRegister.Name	SD_SandBoxRegister.Name()
Address	Returned address offset in bytes, relative to sandbox address space.	SD_SandBoxRegister.Address	SD_SandBoxRegister.Address()
Length	Returned register width in bytes. Typically, '4'.	SD_SandBoxRegister.Length	SD_SandBoxRegister.Length()
Access Type	Returned access type. 'RW' or 'Memory Map'.	SD_SandBoxRegister.AccessType	SD_SandBoxRegister.AccessType()

6. Using SD1 API functions in sample programs

This chapter describes how to use the Keysight SD1 Programming Libraries with Python.

Section 6.1: Basic Work Flow for the Digitizer

In some programming languages, such as Python, *objects* must be created.

1 (Optional) Create a Digitizer object with "SD_AIN functions()".
2 Open the Digitizer module using "open()".
3 Configure the input Channel for Data Acquisition using "DAQconfig()".
4 Start Data Acquisition using "DAQstart()".
5 Read the acquired words on the selected DAQ Channel using "DAQread()".

Section 6.2: Implementing SD1 API functions – Sample Programs

Default locations for Keysight SD1 Programming Libraries, example programs, and waveform files:

- C:\Program Files (x86)\Keysight\SD1\Libraries
- C:\Users\Public\Documents\Keysight\SD1\Examples
- C:\Users\Public\Documents\Keysight\SD1\Examples\Waveforms

6.2.1: Sample program for the overall Digitizer work flow using Python

```
import sys
sys.path.append('C:\Program Files (x86)\Keysight\SD1\Libraries\Python')
import keysightSD1
# MODULE CONSTANTS
PRODUCT = ""
CHASSIS = 1
# change slot number to your value
SLOT = 6
CHANNEL = 1
# CREATE AND OPEN MODULE
module = keysightSD1.SD_AIN()
moduleID = module.openWithSlot(PRODUCT, CHASSIS, SLOT)
if moduleID < 0:
    print("Module open error:", moduleID)
else:
    print("Module opened:", moduleID)
    print("Module name:", module.getProductName())
    print("slot:", module.getSlot())
    print("Chassis:", module.getChassis())
    print()
# CONFIGURE AND START DAQ
POINTS_PER_CYCLE = 100
CYCLES = 500
TRIGGER_DELAY = 0
module.DAQconfig(CHANNEL, POINTS_PER_CYCLE, CYCLES, TRIGGER_DELAY,
keysightSD1.SD_TriggerModes.SWHVITRIG)
module.DAQstart(CHANNEL)
input("Press any key to provide trigger")
module.DAQtrigger(CHANNEL)
# READ DATA
TIMEOUT = 1
dataRead = module.DAQread(CHANNEL, POINTS_PER_CYCLE * CYCLES, TIMEOUT)
```

```
print(dataRead)
# exiting...
module.close()
print()
print("AIN closed")
# © Keysight Technologies, 2020
# All rights reserved.
# You have a royalty-free right to use, modify,
# reproduce and distribute this Sample Application (and/or any modified # version)
# in any way you find useful, provided that you agree that
# Keysight Technologies has no warranty, obligations or liability
# for any Sample Application Files.
#
# Keysight Technologies provides programming examples for illustration only.
# This sample program assumes that you are familiar with the programming language
# being demonstrated and the tools used to create and debug procedures.
# Keysight Technologies support engineers can help explain
# the functionality of Keysight Technologies software components and associated commands,
# but they will not modify these samples to provide added functionality
# or construct procedures to meet your specific needs.
# ----------
```

6.2.2: Sample program for Auto-triggering input waveform using Python

```
# Connect an AWG generator to the Digitizer
import sys
sys.path.append('C:\Program Files (x86)\Keysight\SD1\Libraries\Python')
import keysightSD1
import numpy as np
import matplotlib.pyplot as plt
import time

# CONFIGURATION CONSTANTS
FULL_SCALE = 2 # half peak to peak voltage
DELAY_IN = 0 # 250
READ_TIMEOUT = 100 # 0 means infinite timeout
DAQ_CH = 1 # DAQ channel
WAITER_TIMEOUT_SECONDS = 0.1

def waitUntilPointsRead(module, DAQchannel, totalPoints, timeOut):
    t0 = time.time()
    timeElapsed = 0
```

```
    totalPointsRead = 0
    while (totalPointsRead < totalPoints) and (timeElapsed < timeOut):
            totalPointsRead = module.DAQcounterRead(DAQchannel)
            if(totalPointsRead < totalPoints):
                time.sleep(.1) #input argument of time.sleep is in seconds
            timeElapsed = time.time() - t0

# MODULE CONSTANTS
PRODUCT = ""
CHASSIS = 1
SLOT_IN = 5 # digitizer slot in chassis

# CREATE AND OPEN MODULE IN
digitizer = keysightSD1.SD_AIN()
digitizerID = digitizer.openWithSlot(PRODUCT, CHASSIS, SLOT_IN)
if digitizerID < 0:
    print("Module open error:", digitizerID)

else:
    print("===== Digitizer =====")
    print("ID:\t\t", digitizerID)
    print("Product name:\t", digitizer.getProductName())
    print("Serial number:\t", digitizer.getSerialNumber())
    print("Chassis:\t", digitizer.getChassis())
    print("Slot:\t\t", digitizer.getSlot())
    print()

    plt.ion() #interactive mode
    plt.show(block=False)

    # CONFIGURATION CONSTANTS
    print("Test: Auto trigger, 1000 total points, 1 cycles, 1000 points per cycle...")
    NUM_POINTS_PER_CYCLE = 1000
    NUM_CYCLES = 1
    TOTAL_POINTS = NUM_POINTS_PER_CYCLE * NUM_CYCLES

    # DAQ CONFIGURATION
    digitizer.channelInputConfig(DAQ_CH, FULL_SCALE,
keysightSD1.AIN_Impedance.AIN_IMPEDANCE_50, keysightSD1.AIN_Coupling.AIN_COUPLING_DC)
    digitizer.DAQconfig(DAQ_CH, NUM_POINTS_PER_CYCLE, NUM_CYCLES, DELAY_IN,
keysightSD1.SD_TriggerModes.AUTOTRIG)
```

```
    # DAQ ACQUISITION
    digitizer.DAQflush(DAQ_CH)
    digitizer.DAQstart(DAQ_CH)
    waitUntilPointsRead(digitizer, DAQ_CH, TOTAL_POINTS, WAITER_TIMEOUT_SECONDS)
    readPoints = digitizer.DAQread(DAQ_CH, TOTAL_POINTS, READ_TIMEOUT)


    # STOP DAQ
    digitizer.DAQstop(DAQ_CH);

    # PLOT
    print("Plotting test...")
    plt.clf()
    plt.plot(readPoints, 'r-')
    plt.show()
    plt.pause(2)

    # exiting...
    digitizer.close()
    print()
    print("AIN closed")
# ----------------------------------------
# © Keysight Technologies, 2020
# All rights reserved.
# You have a royalty-free right to use, modify,
# reproduce and distribute this Sample Application (and/or any modified # version)
# in any way you find useful, provided that you agree that
# Keysight Technologies has no warranty, obligations or liability
# for any Sample Application Files.
#
# Keysight Technologies provides programming examples for illustration only.
# This sample program assumes that you are familiar with the programming language
# being demonstrated and the tools used to create and debug procedures.
# Keysight Technologies support engineers can help explain
# the functionality of Keysight Technologies software components and associated commands,
# but they will not modify these samples to provide added functionality
# or construct procedures to meet your specific needs.
# ----------
```

6.2.3: Sample program for DAQ multiple triggering using Python

```
# Connect an AWG generator to the Digitizer
import sys
sys.path.append('C:\Program Files (x86)\Keysight\SD1\Libraries\Python')
import keysightSD1
import numpy as np
import matplotlib.pyplot as plt
import time

# CONFIGURATION CONSTANTS
FULL_SCALE = 2 # half peak to peak voltage
DELAY_IN = 0 # 250
READ_TIMEOUT = 100 # 0 means infinite timeout
DAQ_CH = 1 # DAQ channel
WAITER_TIMEOUT_SECONDS = 0.1

def waitUntilPointsRead(module, DAQchannel, totalPoints, timeOut):
    t0 = time.time()
    timeElapsed = 0
    totalPointsRead = 0
    while (totalPointsRead < totalPoints) and (timeElapsed < timeOut):
            totalPointsRead = module.DAQcounterRead(DAQchannel)
            if(totalPointsRead < totalPoints):
                time.sleep(.1) #input argument of time.sleep is in seconds
            timeElapsed = time.time() - t0

# MODULE CONSTANTS
PRODUCT = ""
CHASSIS = 1
SLOT_IN = 5 # digitizer slot in chassis

# CREATE AND OPEN MODULE IN
digitizer = keysightSD1.SD_AIN()
digitizerID = digitizer.openWithSlot(PRODUCT, CHASSIS, SLOT_IN)
if digitizerID < 0:
    print("Module open error:", digitizerID)

else:
    print("===== Digitizer =====")
    print("ID:\t\t", digitizerID)
    print("Product name:\t", digitizer.getProductName())
```

```
    print("Serial number:\t", digitizer.getSerialNumber())
    print("Chassis:\t", digitizer.getChassis())
    print("Slot:\t\t", digitizer.getSlot())
    print()

    plt.ion() #interactive mode
    plt.show(block=False)

    # CONFIGURATION CONSTANTS
    print("Test: Software triggering, 1000 total points, 500 points per cycle, only two
cycles...")
    NUM_POINTS_PER_CYCLE = 500
    NUM_CYCLES = 2
    TOTAL_POINTS = 1000

    # DAQ CONFIGURATION
    digitizer.channelInputConfig(DAQ_CH, FULL_SCALE,
keysightSD1.AIN_Impedance.AIN_IMPEDANCE_50, keysightSD1.AIN_Coupling.AIN_COUPLING_DC)
    digitizer.DAQconfig(DAQ_CH, NUM_POINTS_PER_CYCLE, NUM_CYCLES, DELAY_IN,
keysightSD1.SD_TriggerModes.SWHVITRIG)

    DAQ_CH2 = DAQ_CH + 1
    digitizer.channelInputConfig(DAQ_CH2, FULL_SCALE,
keysightSD1.AIN_Impedance.AIN_IMPEDANCE_50, keysightSD1.AIN_Coupling.AIN_COUPLING_DC)
    digitizer.DAQconfig(DAQ_CH2, NUM_POINTS_PER_CYCLE, NUM_CYCLES, DELAY_IN,
keysightSD1.SD_TriggerModes.SWHVITRIG)
    mask = 3  # 00000011 (first and second channel)

    # DAQ ACQUISITION
    digitizer.DAQflushMultiple(mask)
    digitizer.DAQstartMultiple(mask)
    digitizer.DAQtriggerMultiple(mask)
    waitUntilPointsRead(digitizer, DAQ_CH, TOTAL_POINTS, WAITER_TIMEOUT_SECONDS)
    waitUntilPointsRead(digitizer, DAQ_CH2, TOTAL_POINTS, WAITER_TIMEOUT_SECONDS)
    print("total points read after first trigger (first cycle) DAQ1:",
digitizer.DAQcounterRead(DAQ_CH))
    print("total points read after first trigger (first cycle) DAQ2:",
digitizer.DAQcounterRead(DAQ_CH2))
    digitizer.DAQtriggerMultiple(mask)
    waitUntilPointsRead(digitizer, DAQ_CH, TOTAL_POINTS, WAITER_TIMEOUT_SECONDS)
    print("total points read after second trigger (second cycle) DAQ1:",
digitizer.DAQcounterRead(DAQ_CH))
    waitUntilPointsRead(digitizer, DAQ_CH2, TOTAL_POINTS, WAITER_TIMEOUT_SECONDS)
    print("total points read after second trigger (second cycle) DAQ2:",
digitizer.DAQcounterRead(DAQ_CH2))
```

```
    # read points
    readPoints = digitizer.DAQread(DAQ_CH, TOTAL_POINTS, READ_TIMEOUT)
    readPoints2 = digitizer.DAQread(DAQ_CH2, TOTAL_POINTS, READ_TIMEOUT)

    # STOP DAQ
    digitizer.DAQstopMultiple(mask)

    # PLOT
    print("Plotting test...")
    plt.clf()
    plt.plot(readPoints, 'r-',readPoints2,'g-')
    plt.show()
    plt.pause(2)

    # exiting...
    digitizer.close()
    print()
    print("AIN closed")
# ----------------------------------------
# © Keysight Technologies, 2020
# All rights reserved.
# You have a royalty-free right to use, modify,
# reproduce and distribute this Sample Application (and/or any modified # version)
# in any way you find useful, provided that you agree that
# Keysight Technologies has no warranty, obligations or liability
# for any Sample Application Files.
#
# Keysight Technologies provides programming examples for illustration only.
# This sample program assumes that you are familiar with the programming language
# being demonstrated and the tools used to create and debug procedures.
# Keysight Technologies support engineers can help explain
# the functionality of Keysight Technologies software components and associated commands,
# but they will not modify these samples to provide added functionality
# or construct procedures to meet your specific needs.
# ----------
```

6.2.4: Sample program for PXI triggering on DAQs using Python

```
# Connect an AWG generator to the Digitizer
import sys
sys.path.append('C:\Program Files (x86)\Keysight\SD1\Libraries\Python')
import keysightSD1
import numpy as np
import matplotlib.pyplot as plt
import time

# CONFIGURATION CONSTANTS
FULL_SCALE = 2 # half peak to peak voltage
DELAY_IN = 0 # 250
READ_TIMEOUT = 100 # 0 means infinite timeout
DAQ_CH = 1 # DAQ channel
WAITER_TIMEOUT_SECONDS = 0.1

def waitUntilPointsRead(module, DAQchannel, totalPoints, timeOut):
    t0 = time.time()
    timeElapsed = 0
    totalPointsRead = 0
    while (totalPointsRead < totalPoints) and (timeElapsed < timeOut):
            totalPointsRead = module.DAQcounterRead(DAQchannel)
            if(totalPointsRead < totalPoints):
                time.sleep(.1) #input argument of time.sleep is in seconds
            timeElapsed = time.time() - t0

# MODULE CONSTANTS
PRODUCT = ""
CHASSIS = 1
SLOT_IN = 5 # digitizer slot in chassis

# CREATE AND OPEN MODULE IN
digitizer = keysightSD1.SD_AIN()
digitizerID = digitizer.openWithSlot(PRODUCT, CHASSIS, SLOT_IN)
if digitizerID < 0:
    print("Module open error:", digitizerID)

else:
    print("===== Digitizer =====")
    print("ID:\t\t", digitizerID)
    print("Product name:\t", digitizer.getProductName())
```

```
    print("Serial number:\t", digitizer.getSerialNumber())
    print("Chassis:\t", digitizer.getChassis())
    print("Slot:\t\t", digitizer.getSlot())
    print()

    plt.ion() #interactive mode
    plt.show(block=False)

    print("Test: PXI external trigger, 1000 total points, 1 cycle...")
    NUM_CYCLES = 1
    TOTAL_POINTS = 1000

    # DAQ CONFIGURATION
    digitizer.channelInputConfig(DAQ_CH, FULL_SCALE,
keysightSD1.AIN_Impedance.AIN_IMPEDANCE_50, keysightSD1.AIN_Coupling.AIN_COUPLING_DC)
    digitizer.DAQconfig(DAQ_CH, TOTAL_POINTS, NUM_CYCLES, DELAY_IN,
keysightSD1.SD_TriggerModes.EXTTRIG)
    digitizer.DAQdigitalTriggerConfig(DAQ_CH,
keysightSD1.SD_TriggerExternalSources.TRIGGER_PXI2,
keysightSD1.SD_TriggerBehaviors.TRIGGER_RISE)

    # DAQ ACQUISITION
    digitizer.DAQflush(DAQ_CH)
    digitizer.DAQstart(DAQ_CH)
    waitUntilPointsRead(digitizer, DAQ_CH, TOTAL_POINTS, WAITER_TIMEOUT_SECONDS)
    print("total points read before trigger:", digitizer.DAQcounterRead(DAQ_CH))

    # PXI2 Trigger
    PXI2 = 2
    digitizer.PXItriggerWrite(PXI2, 1);
    digitizer.PXItriggerWrite(PXI2, 0);
    digitizer.PXItriggerWrite(PXI2, 1);
    waitUntilPointsRead(digitizer, DAQ_CH, TOTAL_POINTS, WAITER_TIMEOUT_SECONDS)
    print("total points read after trigger:", digitizer.DAQcounterRead(DAQ_CH))

    # read points
    readPoints = digitizer.DAQread(DAQ_CH, TOTAL_POINTS, READ_TIMEOUT)
    print("total points read: {}".format(readPoints.size))

    # STOP DAQ
    digitizer.DAQstop(DAQ_CH)
```

```
    # PLOT
    print("Plotting test...")
    plt.clf()
    plt.plot(readPoints, 'r-')
    plt.show()
    plt.pause(2)

    # exiting...
    digitizer.close()
    print()
    print("AIN closed")
# ----------------------------------------
# © Keysight Technologies, 2020
# All rights reserved.
# You have a royalty-free right to use, modify,
# reproduce and distribute this Sample Application (and/or any modified # version)
# in any way you find useful, provided that you agree that
# Keysight Technologies has no warranty, obligations or liability
# for any Sample Application Files.
#
# Keysight Technologies provides programming examples for illustration only.
# This sample program assumes that you are familiar with the programming language
# being demonstrated and the tools used to create and debug procedures.
# Keysight Technologies support engineers can help explain
# the functionality of Keysight Technologies software components and associated commands,
# but they will not modify these samples to provide added functionality
# or construct procedures to meet your specific needs.
# ----------
```

7. Understanding Error Codes in SD1 API

Section 7.1: Description of Error & Warning IDs

7.1.1: Description of SD1 Error IDs

Table 274 lists the error IDs that are displayed when one or more conditions are not met during the running of the SD1 API on either the AWG or the Digitizer modules.

Table 274 List of SD1 Errors, Error IDs and description

Error Define	Error No.	Error Description
SD_ERROR_OPENING_MODULE	-8000	Opening module
SD_ERROR_CLOSING_MODULE	-8001	Closing module
SD_ERROR_OPENING_HVI	-8002	Opening HVI
SD_ERROR_CLOSING_HVI	-8003	Closing HVI
SD_ERROR_MODULE_NOT_OPENED	-8004	Module not opened
SD_ERROR_MODULE_NOT_OPENED_BY_USER	-8005	Module not opened by user
SD_ERROR_MODULE_ALREADY_OPENED	-8006	Module already opened
SD_ERROR_HVI_NOT_OPENED	-8007	HVI not opened
SD_ERROR_INVALID_OBJECTID	-8008	Invalid objectID
SD_ERROR_INVALID_MODULEID	-8009	Invalid moduleID
SD_ERROR_INVALID_MODULEUSERNAME	-8010	Invalid moduleUsername
SD_ERROR_INVALID_HVIID	-8011	Invalid HVI ID
SD_ERROR_INVALID_OBJECT	-8012	Invalid object
SD_ERROR_INVALID_NCHANNEL	-8013	Invalid channelNumber
SD_ERROR_BUS_DOES_NOT_EXIST	-8014	Bus does not exist
SD_ERROR_BITMAP_ASSIGNED_DOES_NOT_EXIST	-8015	Any input assigned to the BitMap does not exist
SD_ERROR_BUS_INVALID_SIZE	-8016	Input size does not fit on this bus
SD_ERROR_BUS_INVALID_DATA	-8017	Input data does not fit on this bus
SD_ERROR_INVALID_VALUE	-8018	Invalid value
SD_ERROR_CREATING_WAVE	-8019	Creating waveform
SD_ERROR_NOT_VALID_PARAMETERS	-8020	Invalid parameters
SD_ERROR_AWG	-8021	AWG failed
SD_ERROR_DAQ_INVALID_FUNCTIONALITY	-8022	DAQ invalid functionality
SD_ERROR_DAQ_POOL_ALREADY_RUNNING	-8023	DAQ buffer pool is already running
SD_ERROR_UNKNOWN	-8024	Unknown error
SD_ERROR_INVALID_PARAMETERS	-8025	Invalid parameters
SD_ERROR_MODULE_NOT_FOUND	-8026	Module not found

Error Define	Error No.	Error Description
SD_ERROR_DRIVER_RESOURCE_BUSY	-8027	Driver resource busy
SD_ERROR_DRIVER_RESOURCE_NOT_READY	-8028	Driver resource not ready
SD_ERROR_DRIVER_ALLOCATE_BUFFER	-8029	Driver cannot allocate buffer
SD_ERROR_ALLOCATE_BUFFER	-8030	Cannot allocate buffer
SD_ERROR_RESOURCE_NOT_READY	-8031	Resource not ready
SD_ERROR_HARDWARE	-8032	Hardware error
SD_ERROR_INVALID_OPERATION	-8033	Invalid operation
SD_ERROR_NO_COMPILED_CODE	-8034	No compiled code in the module
SD_ERROR_FW_VERIFICATION	-8035	Firmware verification failed
SD_ERROR_COMPATIBILITY	-8036	Compatibility error
SD_ERROR_INVALID_TYPE	-8037	Invalid type
SD_ERROR_DEMO_MODULE	-8038	Demo module
SD_ERROR_INVALID_BUFFER	-8039	Invalid buffer
SD_ERROR_INVALID_INDEX	-8040	Invalid index
SD_ERROR_INVALID_NHISTOGRAM	-8041	Invalid histogram number
SD_ERROR_INVALID_NBINS	-8042	Invalid number of bins
SD_ERROR_INVALID_MASK	-8043	Invalid mask
SD_ERROR_INVALID_WAVEFORM	-8044	Invalid waveform
SD_ERROR_INVALID_STROBE	-8045	Invalid strobe
SD_ERROR_INVALID_STROBE_VALUE	-8046	Invalid strobe value
SD_ERROR_INVALID_DEBOUNCING	-8047	Invalid debouncing
SD_ERROR_INVALID_PRESCALER	-8048	Invalid prescaler
SD_ERROR_INVALID_PORT	-8049	Invalid port
SD_ERROR_INVALID_DIRECTION	-8050	Invalid direction
SD_ERROR_INVALID_MODE	-8051	Invalid mode
SD_ERROR_INVALID_FREQUENCY	-8052	Invalid frequency
SD_ERROR_INVALID_IMPEDANCE	-8053	Invalid impedance
SD_ERROR_INVALID_GAIN	-8054	Invalid gain
SD_ERROR_INVALID_FULLSCALE	-8055	Invalid full scale
SD_ERROR_INVALID_FILE	-8056	Invalid file
SD_ERROR_INVALID_SLOT	-8057	Invalid slot
SD_ERROR_INVALID_NAME	-8058	Invalid name
SD_ERROR_INVALID_SERIAL	-8059	Invalid serial number
SD_ERROR_INVALID_START	-8060	Invalid start

Error Define	Error No.	Error Description
SD_ERROR_INVALID_END	-8061	Invalid end
SD_ERROR_INVALID_CYCLES	-8062	Invalid cycles
SD_ERROR_HVI_INVALID_NUMBER_MODULES	-8063	Invalid number of modules on HVI
SD_ERROR_DAQ_P2P_ALREADY_RUNNING	-8064	DAQ P2P is already running
SD_ERROR_OPEN_DRAIN_NOT_SUPPORTED	-8065	Open drain not supported
SD_ERROR_CHASSIS_PORTS_NOT_SUPPORTED	-8066	Chassis port not supported
SD_ERROR_CHASSIS_SETUP_NOT_SUPPORTED	-8067	Chassis setup not supported
SD_ERROR_OPEN_DRAIN_FAILED	-8068	Open drain failed
SD_ERROR_CHASSIS_SETUP_FAILED	-8069	Chassis setup failed
SD_ERROR_INVALID_PART	-8070	Invalid part
SD_ERROR_INVALID_SIZE	-8071	Invalid size
SD_ERROR_INVALID_HANDLE	-8072	Invalid handle
SD_ERROR_NO_WAVEFORMS_IN_LIST	-8073	No waveforms in list
SD_ERROR_PATHWAVE_REGISTER_NOT_FOUND	-8074	PathWave register not found
SD_ERROR_HVI_DRIVER_ERROR	-8075	HVI driver error
SD_ERROR_BAD_MODULE_OPEN_OPTION	-8076	Bad Module open option
SD_ERROR_NOT_HVI2_MODULE	-8077	Not HVI2 Module
SD_ERROR_NO_FP_OPTION	-8078	Indicates that the module is not FPGA programmable. This error is returned when the FPGALoad API is called for any such hardware that does not have the -FP1 option.
SD_ERROR_FILE_DOES_NOT_EXIST	-8079	Indicates that the specified file path, passed as an argument does not exist.

7.1.2: Description of SD1 Warning IDs

Table 275 lists the warning IDs specific to Digitizer modules.

Table 275 List of SD1 Warnings, Warning IDs and Warning messages

Warning Define	Warning No.	Warning message displayed
SD_WARNING_DAQ_POINTS_ODD_NUM	-9000	DAQ supports only even number of 'DAQpointsPerCycle'. The input value is reduced by 1.

8. Documentation References

This section provides references to the links where you can access one or more documents, as required.

Section 8.1: Accessing Online Help for SD1 3.x software

The Online Help file for Digitizer/Combos can be accessed via the **Help** menu of the Digitizer SFPs for the M310xA modules and M330xA modules.

Alternatively, you may access this help file on the local disk where the SD1 3.x software is installed. By default, the Help file can be found in *C:\Program Files (x86)\Keysight\SD1\help\M31_M33XX_Digitizer*.

1 Navigate to folder where the SD1 files are installed.

2 Click the file "*DigitizerUser Guide.htm*", which is highlighted in Figure 52.

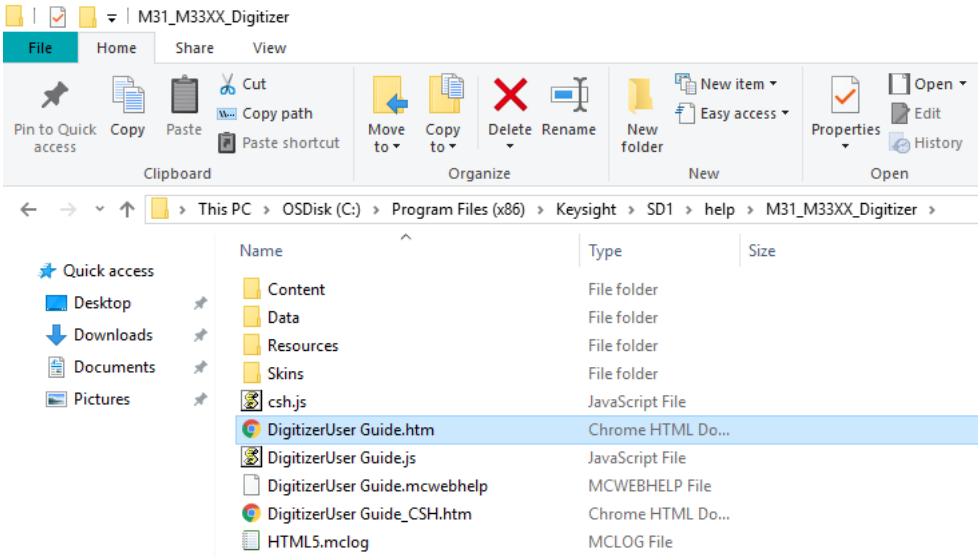


Figure 52 Accessing Digitizer Online Help on the local disk

The Online Help file for SD1 3.x Software is launched on the default browser on your machine. Note that the rest of the files need not be accessed as they contain the contents and formatting settings for the Online Help file.

Section 8.2: Links to other documents

Document Reference	Reference link
SD1 3.x Software Startup Guide	http://literature.cdn.keysight.com/litweb/pdf/M3XXX-90002.pdf
KS2201A PathWave Test Sync Executive User Guide	Visit https://www.keysight.com/.
KF9000A PathWave FPGA (Home Page)	https://www.keysight.com/us/en/products/software/pathwave-test-software/pathwave-fgpa-software.html *PathWave FPGA Customer Documentation* can be accessed from the **Help** menu of the design environment.
BSP Guides	
M3102A PXIe Digitizer	Accessed from the Help menu of the *PathWave FPGA 2020 Update 1.0* design environment.
M3201A PXIe Arbitrary Waveform Generator	Accessed from the Help menu of the *PathWave FPGA 2020 Update 1.0* design environment.
M3202A PXIe Arbitrary Waveform Generator	Accessed from the Help menu of the *PathWave FPGA 2020 Update 1.0* design environment.
Data Sheets	
M3100A PXIe Digitizers with Optional Real-Time Sequencing and FPGA Programming	https://www.keysight.com/us/en/assets/7018-05400/data-sheets/5992-1806.pdf
M3102A PXIe Digitizers with Optional Real-Time Sequencing and FPGA Programming	https://www.keysight.com/us/en/assets/7018-05399/data-sheets/5992-1805.pdf
M3201A PXIe Arbitrary Waveform Generator with Optional Real-Time Sequencing and FPGA Programming	https://www.keysight.com/us/en/assets/7018-05391/data-sheets/5992-1797.pdf
M3202A PXIe Arbitrary Waveform Generator with Optional Real-Time Sequencing and FPGA Programming	https://www.keysight.com/us/en/assets/7018-05392/data-sheets/5992-1798.pdf
M3300A PXIe Arbitrary Waveform Generator/Digitizer with Optional Real-Time Sequencing & FPGA Programming	https://www.keysight.com/us/en/assets/7018-05403/data-sheets/5992-1809.pdf
M3302A PXIe Arbitrary Waveform Generator/Digitizer with Optional Real-Time Sequencing & FPGA Programming	https://www.keysight.com/us/en/assets/7018-05402/data-sheets/5992-1808.pdf

Index

Edition 1.01, September 2020

M3xxx-90004

www.keysight.com