SD1 3.x Software for M320xA / M330xA Arbitrary Waveform Generators

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-90003

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 AWGs Theory of Operation

Keysight M3201A/M3202A PXIe Arbitrary Waveform Generators include an embedded Function Generator (FG), Arbitrary Waveform Generator (AWG), and modulator blocks; together, they form a powerful signal generator that is capable of generating standard waveforms (sinusoidal, triangular, square, and DC voltages) or arbitrary waveforms defined by the user and stored on its onboard RAM. With embedded modulator blocks, the output channels can be modulated in phase, frequency, amplitude, or IQ to create analog or digital modulation.

Section 1.1: Working with Signal Generation/Channel Structure

Each channel (Channel 1 to Channel n) has an identical structure that contains a Function Generator (FG), Arbitrary Waveform Generator (AWG), Frequency and Phase Angle Modulator, Amplitude and DC Offset Amplitude Modulator, and an IQ Modulator.

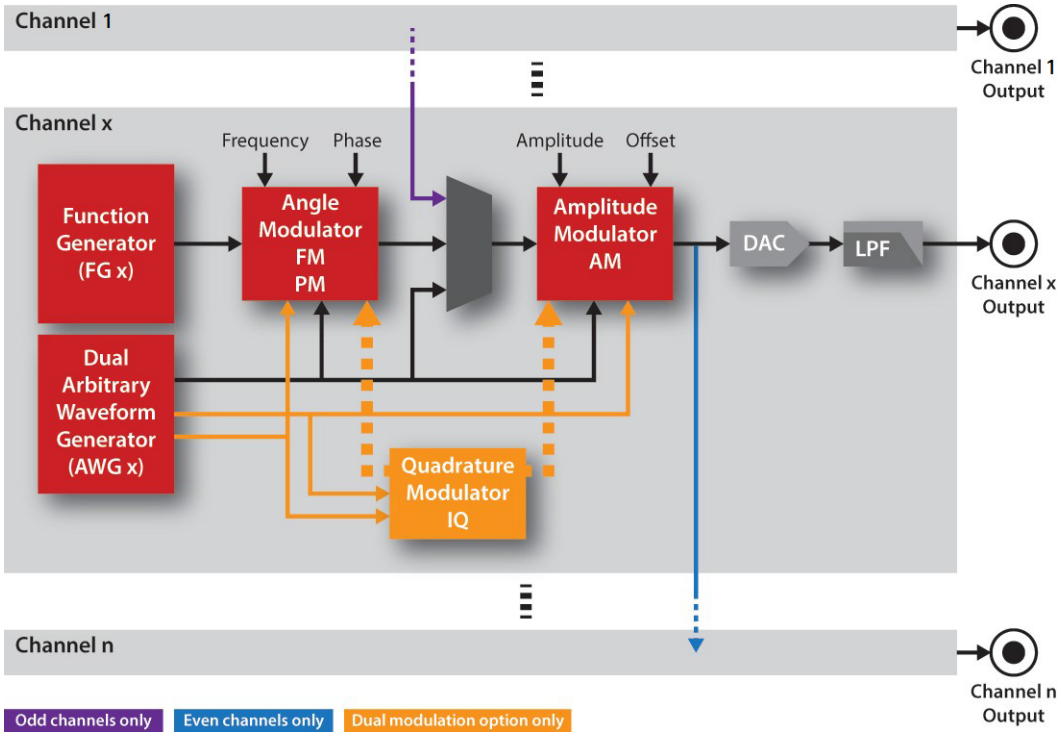


Figure 1 Workflow during signal generation

1.1.1: Channel Numbering and Compatibility Mode

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.

Table 1 API for Channels

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: Channel Waveshape Types

Each channel has a Function Generator (FG) block that generates basic periodic signals and an Arbitrary Waveform Generator (AWG) block that generates arbitrary waveforms.

FG vs AWG

When the generation of periodic signals is needed, an FG has many advantages over a pure AWG solution:

- The FG does not use onboard RAM.
- The "channelWaveShape()", "channelFrequency()" and "channelPhase()" functions can be changed in real time without having to modify a static waveform loaded in memory.
- Achieving the same precision in frequency and phase with a pure AWG solution requires a huge amount of memory.

Table 2 Channel Waveshape Types

WaveShape	Description	Name	Value
HiZ*	The output signal is set to HIZ. No output signal is provided.	AOU_HIZ	-1
No Signal	The output signal is set to 0. All other channel settings are maintained.	AOU_OFF (default)	0
Sinusoidal	Generated by the Function Generator.	AOU_SINUSOIDAL	1
Triangular	Generated by the Function Generator.	AOU_TRIANGULAR	2
Square	Generated by the Function Generator.	AOU_SQUARE	4
DC Voltage	Generated by the Amplitude Modulator.	AOU_DC	5
Arbitrary Waveform	Generated by the Arbitrary Waveform Generator (See AWG Waveform Types).	AOU_AWG	6
Partner Channel	Only for odd channels. It is the output of the previous channel (to create differential signals, and so on).	AOU_PARTNER	8

* Only available for Keysight M3202A PXIe AWG models

Using Partner Channel

AOU_PARTNER (8) is used to generate differential signals on alternate Channels (either 1 & 3 or 2 & 4). The signal comes from AWG mode and not the Function Generator. The Partner Channel feature is hidden in the SD1 SFP user interface and can be enabled using the SD1 API only.

To produce differential signals for channels 1 and 3:

1 (Optional) Create an AWG object with "SD_AOU functions()".
2 Open an AWG with "open()".
3 Flush waveforms with "waveformFlush()".
4 Set "channelWaveShape()" to AOU_AWG and AOU_PARTNER on alternate Channels.
For example, set Channels 1 & 3 to AOU_AWG and Channels 2 & 4 to AOU_PARTNER.
5 On each AWG Channel, set the amplitude using "channelAmplitude()".
6 On the alternate Partner Channels, set equal and opposite amplitude values using "channelAmplitude()".
For example, if you set an amplitude of 1V on Channels 1 & 3, the amplitude on Channels 2 & 4 must be -1V.

7 Load an AWG waveform using “waveformLoad()”.
8 Queue the waveform on the AWG Channels only using “AWGqueueWaveform()”.
The waveform on the Partner Channels are queued automatically.
9 Start the waveform on each Channel simultaneously using “AWGstartMultiple()”.
10 Trigger the waveform on each Channel simultaneously to play the waveforms using “AWGtriggerMultiple()”.
11 Stop the waveform on each Channel simultaneously using “AWGstopMultiple()”.

To view the corresponding programming steps, see “Sample Program for using AWG Partner Channel as Differential” on page 225.

Table 3 API function for Channel Waveshapes

Function Name	Comments	Reference
channelWaveShape	Sets the channel waveshape type.	channelWaveShape()

1.1.3: Signal Generation with the Function Generator

Each channel has a Function Generator (FG) that generates basic periodic signals (sinusoidal, triangular, square) and is commonly used to generate the RF carrier in modulation schemes. These periodic signals can be modulated in frequency, phase, amplitude, or IQ.

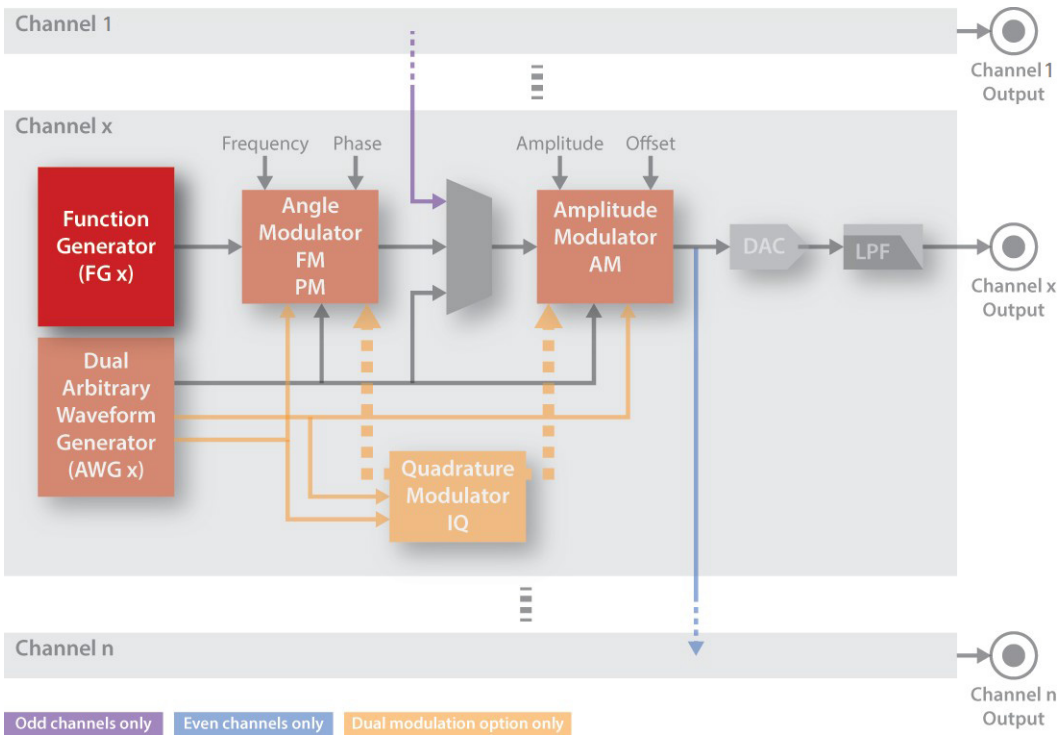


Figure 2 Workflow for Function Generator

Waveform harmonics

Non-sinusoidal wave shapes (triangular, square, and so on) have high frequency components that may fall outside the bandwidth of the reconstruction filter if the fundamental frequency is too high. In this situation, the output analog signal may suffer some distortion due to the missing harmonics, becoming a sinusoidal as the fundamental frequency approaches the cutoff frequency of the reconstruction filter.

1.1.4: Signal Generation with the Arbitrary Waveform Generator

Each channel has an Arbitrary Waveform Generator (AWG) block that generates arbitrary waveforms that can be sent directly to each output channel or they can be used as a modulating signal for the frequency, phase, amplitude, or IQ modulators.

See “Working with AWG Waveforms” on page 19.

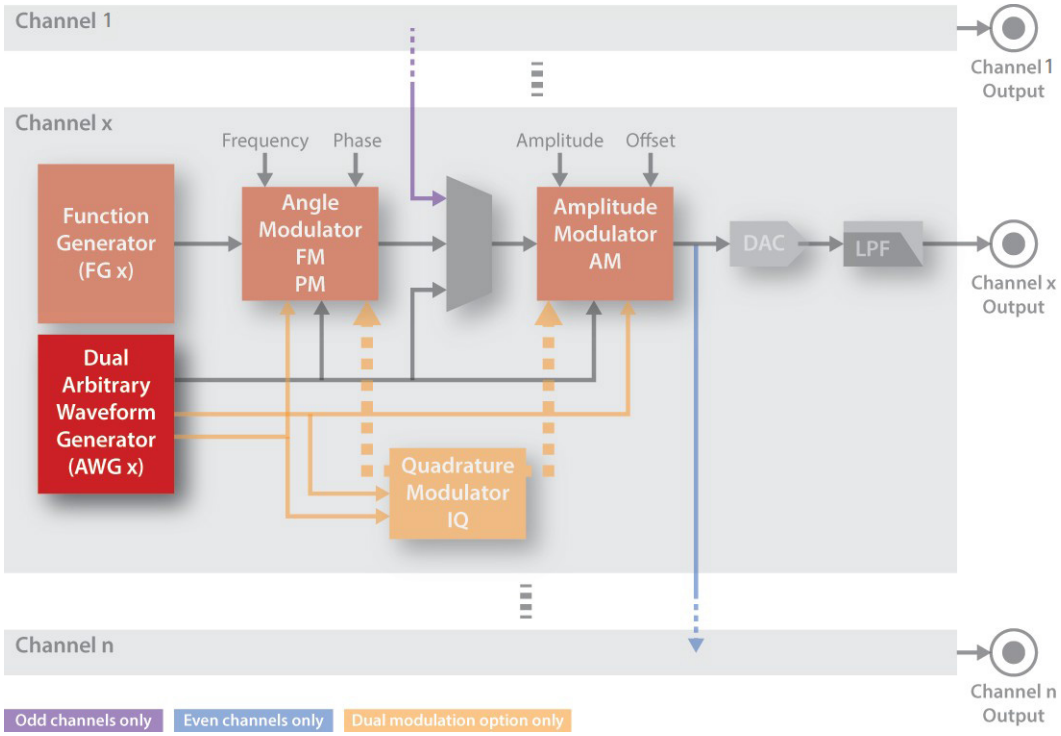


Figure 3 Workflow for Arbitrary Waveform Generator

1.1.5: Channel Frequency and Phase

Each channel has an Angle Modulator block that has a frequency and phase control. In angle modulation schemes, these controls set the frequency and phase of the carrier.

See "Frequency and Phase Modulation (Angle Modulator Block)" on page 29.

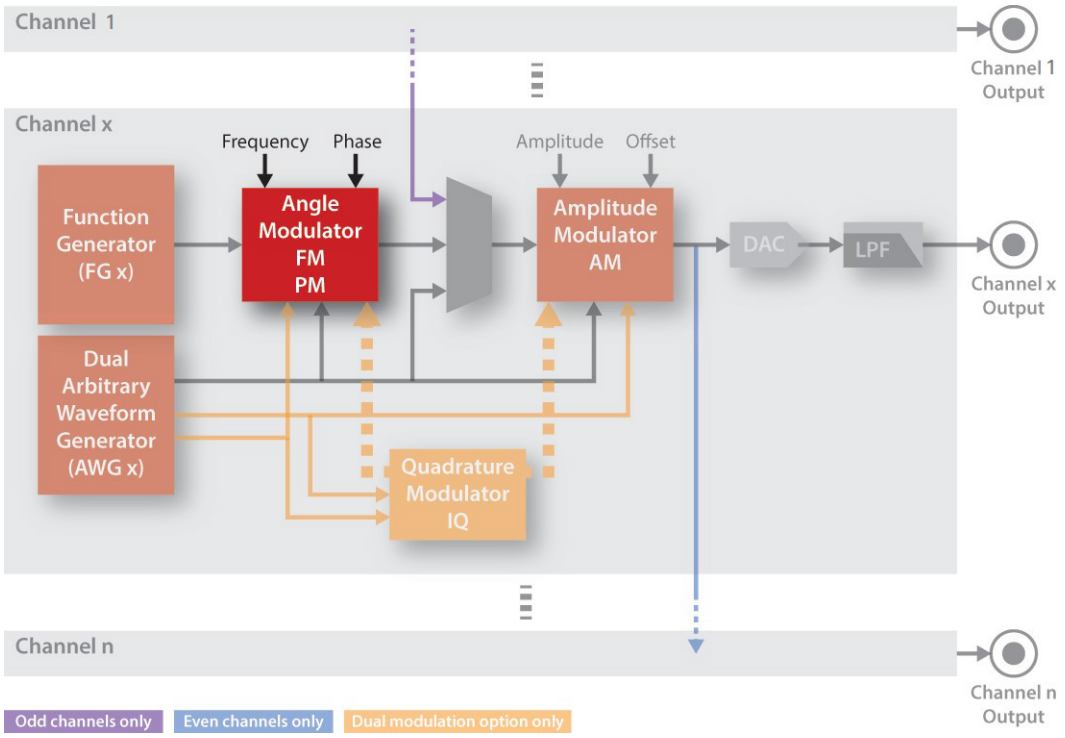


Figure 4 Workflow for Channel Frequency and Phase

Phase coherent vs. phase continuous

Changes in the output signal are always phase continuous, not phase coherent. For example, the frequency is changed from freq1 to freq2 and changed back to freq1, the phase will not be the initial one. To achieve phase coherent behavior, the channel accumulated phase can be reset using channelPhaseReset().

Additionally, in HVI operations, the execution time is deterministic, which allows you to calculate the new phase and adjust it after any frequency change.

Table 4 Programming Function for channel frequency and phase

Function Name	Comments	Reference
channelFrequency	Sets the frequency of the FG	channelFrequency()
channelPhase	Sets the phase of the FG	channelPhase()
channelPhaseReset	Resets the accumulated phase	channelPhaseReset()

1.1.6: Channel Amplitude and DC Offset

Each channel has an Amplitude Modulator block that has an amplitude and DC offset control. These controls have a combined range from 1.5 V to –1.5 V.

See “AM and DC Offset (Amplitude Modulator Block)” on page 31.

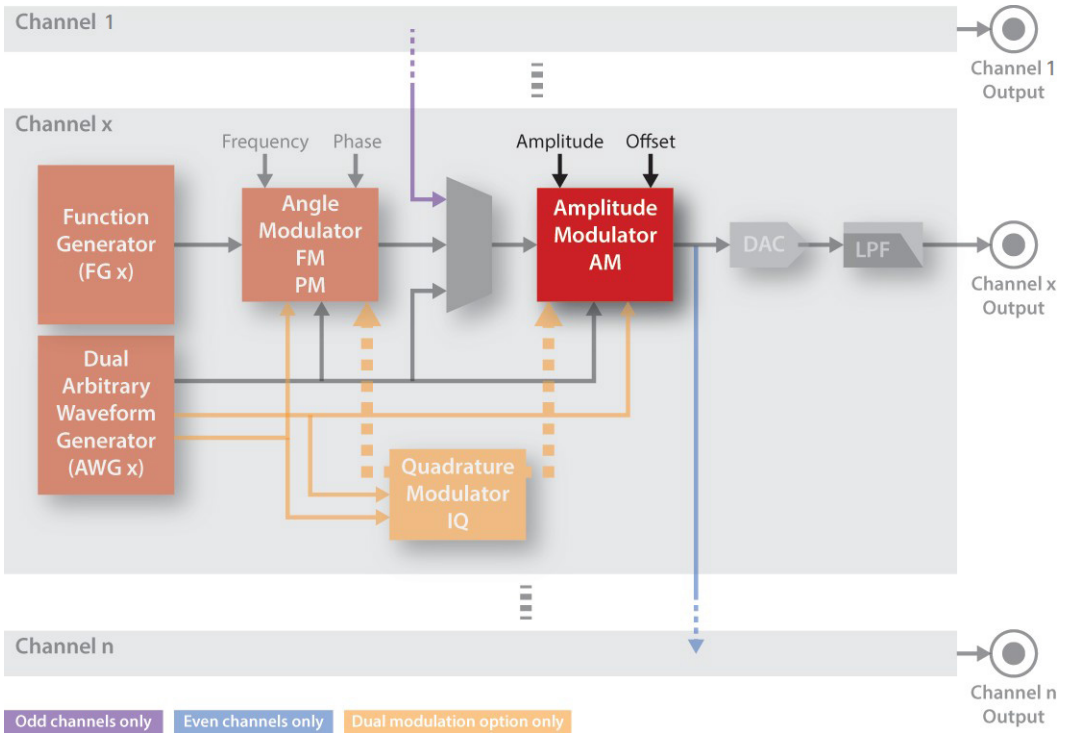


Figure 5 Workflow for Channel Amplitude and DC Offset

Table 5 Programming Function for channel amplitude and DC offset

Function Name	Comments	Reference
channelAmplitude	Sets the output amplitude	channelAmplitude()
channelOffset	Sets the output DC offset	channelOffset()

Section 1.2: Working with AWG Waveforms

Each channel has an Arbitrary Waveform Generator (AWG) block that generates arbitrary waveforms.

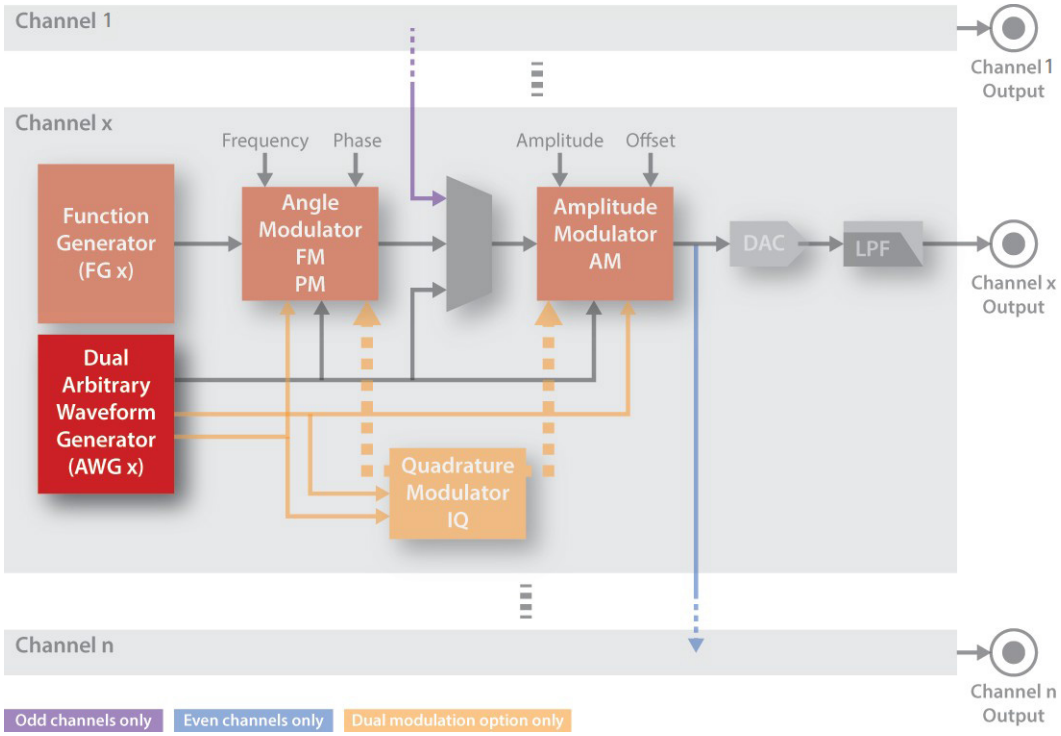


Figure 6 Workflow during AWG waveforms

1.2.1: AWG Programming Process

AWG block operation can be configured with a one-step programming process or with a step-by-step programming process.

One-Step AWG programming process

The "AWG()" function provides a one-step solution to load, queue, and run a single waveform directly from a file or from an array in the PC. This function simplifies the generation of a single waveform, but it does not allow control over the following aspects of waveform generation:

- The possibility to prepare the AWG queue with multiple waveforms in advance before starting the generation; this may be important to create more complex generation sequences.
- The possibility to have a variety of waveforms in the onboard RAM in order to queue them repeatedly in the AWGs in an efficient way.
- The precise moment when waveforms are transferred from a file to the PC RAM and to the onboard RAM. This may be important for long waveforms and time-critical applications.

Step-by-Step AWG programming process

Keysight SD1 Programming Libraries provide full control of all aspects of arbitrary waveform generation:

1 Create waveforms in the PC RAM with "new()" function. Waveforms can be created from points in an array or from points in a file stored on hard disk.

2 Transfer waveforms to a module's onboard RAM with waveformLoad().

3 Queue waveforms in an AWG with AWGqueueWaveform() to create the desired generation sequence.

4 Select the sync mode of the queue with AWGqueueSyncMode().

5 Start the generation with AWGstart() and provide triggers if required.

See "AWG Waveform Queue System" on page 20.

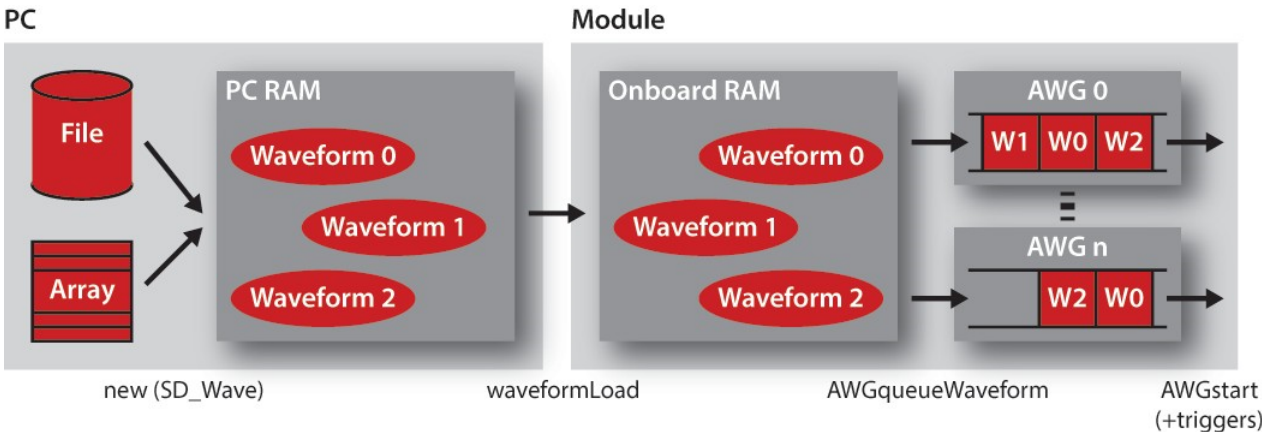


Figure 7 Waveform programming flow

1.2.2: AWG Waveform Queue System

Each AWG block has a flexible waveform queue system that can be used to configure complex generation sequences. In order to generate waveforms, they must be loaded into the module onboard RAM and queued in each corresponding AWG.

The AWG waveform queue system has the following advantages:

- Provides a way to generate a sequence of waveforms one after the other with no discontinuities.
- Allows selection of many parameters (trigger mode, start delay, cycles, prescaler, and so on) individually per queued waveform.
- Waveforms can have many instances in many AWG queues, but only one copy is required in the onboard RAM. This feature saves onboard RAM memory.

Each AWG queued waveform has the following parameters:

- **Trigger Mode**: selects the trigger for the waveform.
- **Start Delay**: adds an optional delay from the reception of the trigger to the beginning of the waveform generation.
- **Cycles**: the number of times the waveform is repeated. Using the appropriate trigger mode, the AWG can be set to require one trigger per cycle or just one trigger at the beginning of all cycles.

- **Cyclic Mode**: sets the repetition cycles for the complete queue. The default mode is “One Shot” and the complete queue will be reproduced one time. Complete queues with a number of waveforms with a limited number of cycles can be repeated with AWGqueueConfig().
  - For Cyclic mode, the minimum play time per cycle should be 1µs for M3201A/M330xA and 2µs for M3202A modules.
  - For fast trigger rates, it is recommended to consolidate all waveforms into a single bigger waveform.
- **Prescaler**: divides the effective waveform sampling rate.

See “AWG Prescaler and Sampling Rate” on page 22.

AWG Waveform Queue System Examples

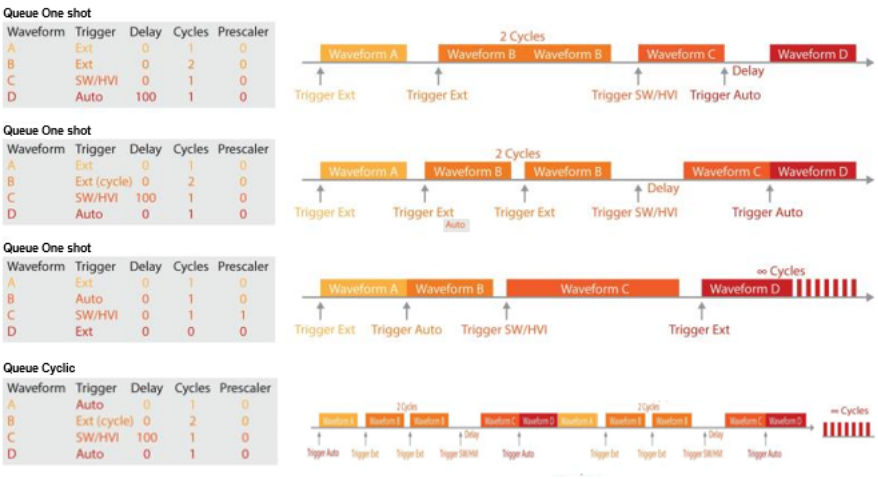


Figure 8 Flow in an AWG Waveform Queue System

Inter-waveform/inter-cycles discontinuities

The AWG Queue System allows you to queue many waveforms one after the other. It also provides the capability to set repetition cycles per waveform and repetition cycles for the complete queue. In all these cases, there are absolutely no discontinuities between waveforms or cycles, providing a continuous waveform generation from the last sample point of the finished waveform to the first sample point of the starting waveform.

HVI Generation Sequences

The queue system of the AWGs provide a way to create generation sequences. For more complex sequences, the best solution is to use Keysight’s PathWave Test Sync Executive software API. This API is based upon the Hard Virtual Instruments (HVIs) technology, using which you may achieve time deterministic sequences with nanosecond resolution, providing a hard real-time execution. For more information about the HVI technology, see Chapter 2, “Using the KS2201A PathWave Test Sync Executive software”.

1.2.3: AWG Prescaler and Sampling Rate

You can set a different prescaler value for each of the waveforms queued in the AWG. This prescaler reduces the effective sampling frequency of each individual waveform in the following manner:

M3201A

0=> fs = 500 MSa/s

>=1 => fs = 100/n MSa/s

M3202A

0=> fs = 1 GSa/s

1 => fs = 200 MSa/s

>1 => fs = 100/n MSa/s

$$\begin{cases} f_s = f_{CLKsys} & prescaler = 0 \\ f_s = \frac{f_{CLKsys}}{5 \cdot prescaler} & prescaler > 0 \end{cases} \quad (1)$$

where, f_S is final effective sampling frequency and prescaler is an integer value (0 to 4095).

An important advantage of this method is the possibility to change the sampling frequency in real time from one waveform to another, reducing waveform sizes and maximizing the flexibility of the AWG.

Prescaler vs. Upsampling

Note that reducing the effective sampling rate of the waveform with the prescaler is not the same as using a full Upsampler, as the prescaler does not contain any filter. Therefore, it generates aliasing inside the reconstruction filter bandwidth. For applications where full Upsampling is required, you must use an IF Generator or Transceiver with DUC (Digital Up Converter) capabilities.

1.2.4: AWG Trigger Mode

A different trigger mode can be configured in each queued waveform.

Table 6 AWG Trigger Mode options

Options	Description	Name	Value
Auto (Immediate)	The waveform is launched automatically after AWGstart, or when the previous waveform in the queue finishes	AUTOTRIG	0
Software / HVI*	Software trigger. The AWG is triggered by the AWGtrigger, provided that the AWG is running. AWGtrigger can be executed from the user application (VI) or from an HVI.	SWHVITRIG	1
Software / HVI (per cycle)	Software trigger. Identical to the previous option, but the trigger is required per each waveform cycle.	SWHVITRIG_CYCLE	5
External Trigger	Hardware trigger. The AWG waits for an external trigger.	EXTTRIG	2
External Trigger (per cycle)	Hardware trigger. Identical to the previous option, but the trigger is required per each waveform cycle.	EXTTRIG_CYCLE	6

* VIHVITRIG is equivalent, but is considered obsolete

Table 7 API functions for AWG Trigger Modes

Function Name	Comments	Reference
AWG	Provides a one-step method to load, queue, and start a single waveform	AWG()
AWGqueueWaveform	Queues a waveform in the specified AWG	AWGqueueWaveform()

If the queued waveforms are going to use any of the External Trigger modes, the source of this trigger must be configured using "AWGtriggerExternalConfig()".

External Trigger Connector/Line Usage

Apart from the AWG trigger settings, an external trigger connector/line may have additional settings (input/output direction, sampling/synchronization options, and so on) that must be configured for proper operation.

1.2.5: AWG External Trigger Source

Table 8 AWG External Trigger Source options

Options	Description	Name	Value
External I/O Trigger	The AWG trigger is a TRG connector/line of the module. PXI form factor only: this trigger can be synchronized to CLK10.	TRIG_EXTERNAL	0
PXI Trigger [0 to n]	PXI form factor only. The AWG external trigger is a PXI trigger line and it is synchronized to CLK10.	TRIG_PXI + Trigger No.	4000 + Trigger No.

1.2.6: AWG External Trigger Behavior

Table 9 AWG External Trigger Behavior options

Options	Description	Name	Value
Active High	Trigger is active when it is at level high	TRIG_HIGH	1
Active Low	Trigger is active when it is at level Low	TRIG_LOW	2
Rising Edge	Trigger is active on the rising edge	TRIG_RISE	3
Falling Edge	Trigger is active on the falling edge	TRIG_FALL	4

1.2.7: AWG Markers

A different marker can be configured for each AWG channel. All waveforms must already be queued using "AWGqueueMarkerConfig()" in one of the module's AWGs.

1.2.8: AWG FlexCLK Synchronization (models with variable sampling rate)

The internal diagram and the operation of the M3201A/M3202A PXIe AWGFlexCLK system is shown in "FlexCLK Technology (models w/ variable sampling rate)" on page 36. This advanced technology allows you to change the sampling frequency of the AWGs (CLKsys), while maintaining full synchronization capabilities due to the internal CLKsync signal. CLKsync is an internal signal used to start the AWGs and it is aligned with CLKsys and PXI CLK10. Its frequency depends on the FlexCLK, resulting in the following scenarios:

$$\mathbf{f}_{CLKsync} = \mathbf{f}_{PXI\_CLK10}$$

When both frequencies coincide, there is no phase uncertainty between both signals and a trigger synchronized with PXI CLK10 will start the AWGs always with the same skew, independently of the clock boot conditions. This ensures proper synchronization between different modules.

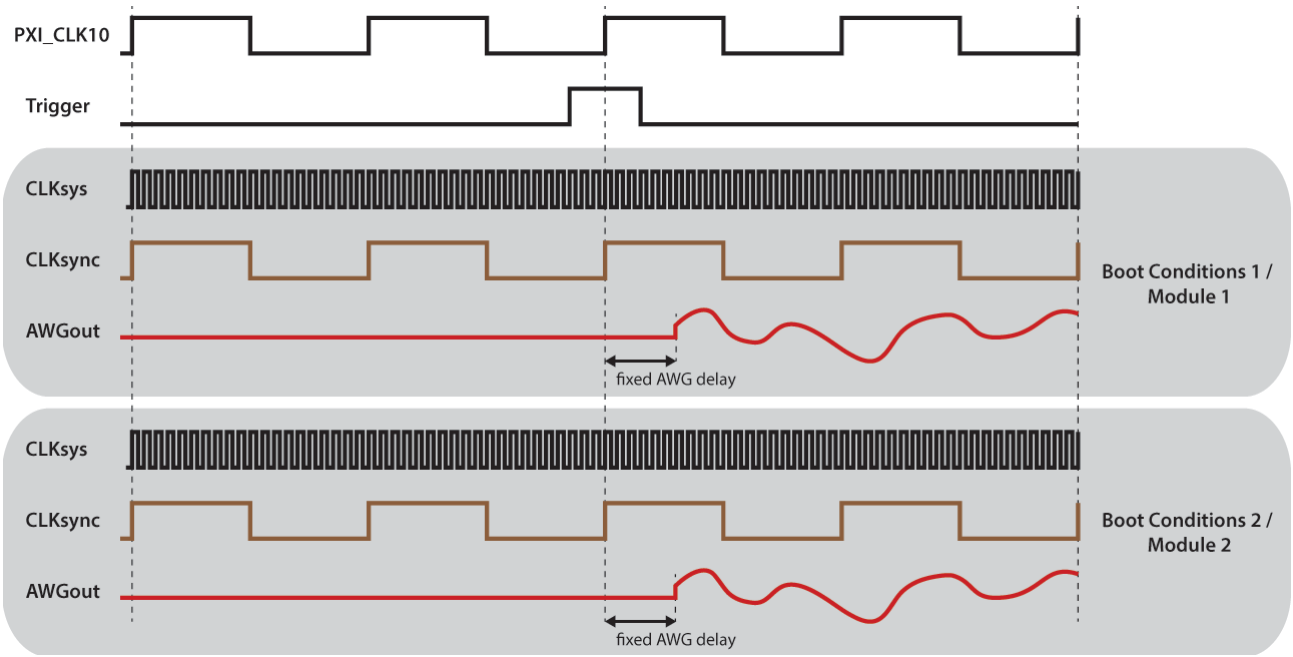


Figure 9 Block diagram depicting proper synchronization between modules

$$f_{CLKsync!} = f_{PXI\_CLK10}$$

When both frequencies do not coincide, both signals are still aligned, but there is a phase uncertainty due to their frequency difference. In this case, if a trigger synchronized with PXI CLK10 is sent to the system, there might be a skew between the start of different AWGs.

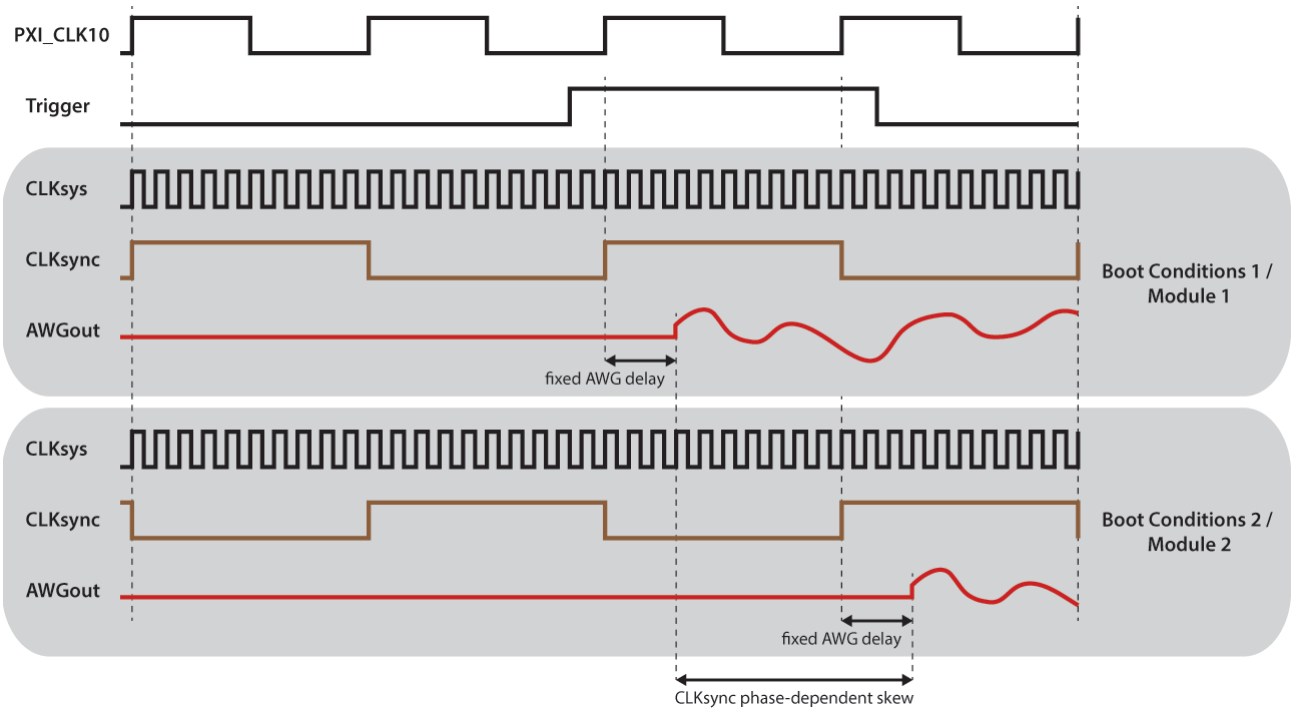


Figure 10 Block diagram depicting skew between the start of different AWG modules

This phase uncertainty can be easily corrected using clockResetPhase(). This function sets the modules in a sync mode and the next trigger is used to reset the phase of the CLKsync and CLKsys signals, not to start the AWGs. In this way, the phase uncertainty between different modules or between different boot conditions can be eliminated, resulting in a predictable and repeatable skew.

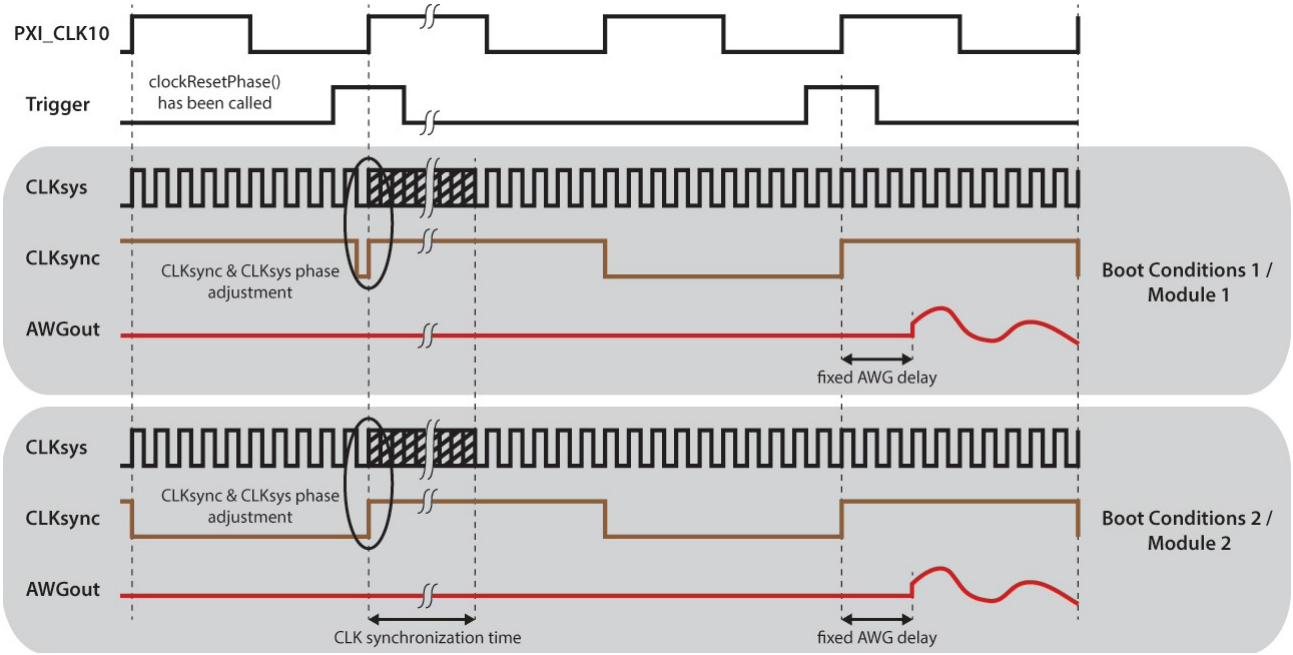


Figure 11 Block diagram depicting synchronization between modules using the clockResetPhase function

1.2.9: AWG Waveform Array and *.csv file structure

The two possible sources of waveforms are an array in the PC RAM and a file in the PC HDD. The memory array is just an array of waveform points, without header and without any particular structure. A waveform file is simply a text file with values separated by commas (*.csv).

One-component Waveform File

	Template	Example
Waveform Header	waveformName,name waveformPoints,nPoints waveformType,type	waveformName,myGaussian waveformPoints,100 waveformType,WAVE_ANALOG_16
Waveform Points	Point 0 Point 1 Point 2 Point 3 ... Point (nPoints-1)	0.59 0.37 -0.90 0.13 ... 0.34

Two-component Waveform File

	Template	Example
Waveform Header	waveformName,name waveformPoints,nPoints waveformType,type	waveformName,myDualGaussian waveformPoints,100 waveformType, WAVE_ANALOG_16_DUAL
Waveform Points	Point A0,Point B0 Point A1,Point B1 Point A2,Point B2 Point A3,Point B3 ... Point A(nPoints-1),Point B(nPoints-1)	0.47,0.87 -0.78,0.98 0.79,-0.47 1.00,0.15 ... -1.41,0.90

Figure 12 Block diagram depicting waveform file structure

The “waveformType” is a parameter that tells the module which kind of waveform it used to configure the AWG internally.

1.2.10: AWG Waveform Types

See “waveformType” in waveform file or in the description for the new() function.

Table 10 Waveform types and corresponding values

Waveform Type	Description	Name	Value
Analog 16 Bits	Analog normalized waveforms (-1 to 1) defined with doubles	WAVE_ ANALOG_16	0
Analog 32 Bits	Analog normalized waveforms (-1 to 1) defined with doubles	WAVE_ANALOG_32	1
Analog 16 Bits Dual	Analog normalized waveforms (-1 to 1) defined with doubles, with two components (A and B)	WAVE_ANALOG_16_DUAL	4
Analog 32 Bits Dual	Analog normalized waveforms (-1 to 1) defined with doubles, with two components (A and B)	WAVE_ANALOG_32_DUAL	6
IQ*	Analog normalized waveforms (-1 to 1) defined with doubles, with two components (I and Q)	WAVE_IQ	2
IQ Polar*	Analog waveforms (-1 to 1 module, -180 to +180 phase) defined with doubles, with two components (Magnitude and Phase)	WAVE_IQPOLAR	3
Digital	Digital waveforms defined with integers	WAVE_DIGITAL	5

*When using IQ or IQ Polar, each component will only play at a maximum rate of 500 MSa/s.

NOTE The M3201A PXIe AWG and both the M3300A & M3302A PXIe Combos do not support both Analog 32 Bits (Single) and Analog 32 Bits Dual waveforms.

Section 1.3: Working with Signal Modulation

Signals can be modulated in frequency, phase, amplitude, or IQ.

1.3.1: Frequency and Phase Modulation (Angle Modulator Block)

The output signal of the Function Generator (FG) block or the Arbitrary Waveform Generator (AWG) block can be modulated.

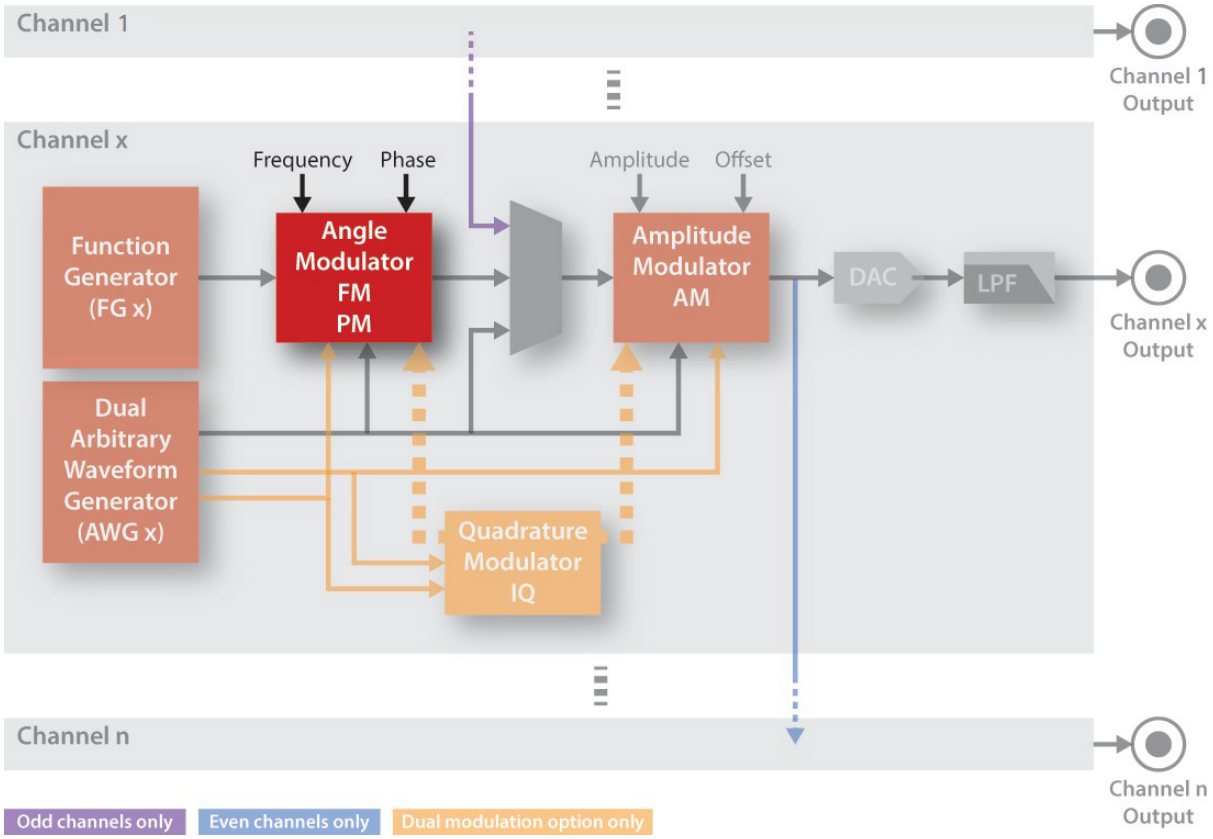


Figure 13 Block diagram depicting functionality of Angle Modulator

Table 11 Angle Modulation options

Options	Description	Name	Value
No Modulation	Modulation is disabled.	AOU_MOD_OFF (default)	0
Frequency Modulation	AWG is used to modulate the Channel frequency.	AOU_MOD_FM	1
Frequency Modulation (32 bits)*	AWG is used to modulate the Channel frequency.	AOU_MOD_FM_32b	1
Phase Modulation	AWG is used to modulate the Channel phase.	AOU_MOD_PM	2

*Models with Option DM1 dual modulation capability (amplitude and angle modulation simultaneously)

The modulating signal is generated using the AWG associated to that particular channel, for example, AWG1 for channel 1.

The angle modulator allows you to create any analog/digital frequency or phase modulation. For example: FM, FSK, PM, PSK, DPSK, and so on.

The output signal of a Channel using the frequency modulation is described using the following equation:

$$Output(t) = A \cdot cos[2\pi(f_c + G \cdot AWG(t)) \cdot t + \emptyset_c]$$

where,

- A is the channel amplitude (set by "channelAmplitude()")
- G is the deviation gain or peak frequency deviation (set by "modulationAngleConfig()"). Note that G is only used for 16-bit waveforms.
- AWG(t) is the normalized modulating signal generated by the AWGs.
- $cos[2\pi(f_c t) + \emptyset_c]$ is the carrier signal, generated with the Function Generators.

As an example, for the generation of an FM signal with an amplitude of 0.8 Vp, a modulation index (h) equal to 0.5 and a maximum frequency of the modulating signal of 10 MHz, the settings must be A=0.8 and G=5,000,000 (where, G = h*max.freq[AWG(t)]).

The output signal of a Channel using the phase modulation is described by the following equation:

$$Output(t) = A \cdot cos[2\pi f_c t + \emptyset_c + G \cdot AWG(t)]$$

where,

- A is the channel amplitude (set by "channelAmplitude()")
- G is the deviation gain or peak phase deviation (set by "modulationAngleConfig()"). Note that G is only used for 16-bit waveforms.
- AWG(t) is the normalized modulating signal generated by the AWGs.
- $cos[2\pi(f_c t) + \emptyset_c]$ is the carrier signal, generated with the Function Generators.

As an example, for the generation of a PM signal with an amplitude of 0.8 Vp and a modulation index (h) equal to 180, the settings must be A=0.8 and G=180 (G=h).

Programming Information

Table 12 API for Angle Modulation functions

Function	Comments	Details
modulationAngleConfig	Configures the angle modulator	modulationAngleConfig()
AWG functions	Control the modulating signal	AWG functions under SD_AOU functions
FG functions	Control the carrier signal	Channel functions under SD_AOU functions

1.3.2: AM and DC Offset (Amplitude Modulator Block)

The amplitude modulator can be used to modulate the amplitude or change the DC offset of the output signal.

Mutual Exclusions for Modulations

Internally, IQ modulation uses the amplitude and the angle modulators, so they cannot be used when the module is working in IQ mode.

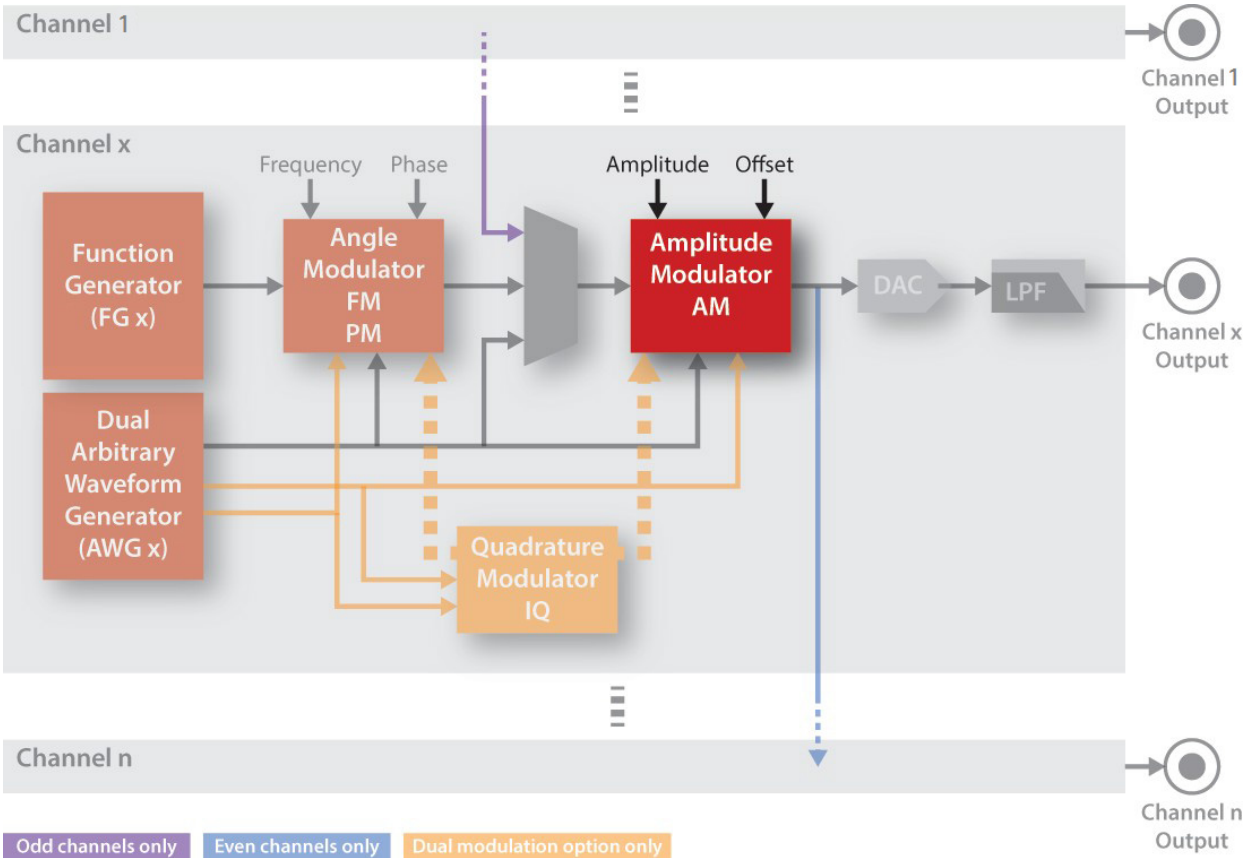


Figure 14 Block diagram depicting functionality of Amplitude Modulator

The modulating signal is generated using the AWG associated with a particular channel (for example, AWG1 corresponds to Channel 1). The Amplitude Modulator can be used to create analog/digital amplitude modulation (for example, AM, ASK, and so on).

The output signal of a channel using the Amplitude Modulator is described using the following equation:

$$Output(t) = (A + G \cdot AWG(t)) \cdot \cos(2\pi f_c t + \varnothing_c)$$

where,

- A is the channel amplitude (set by "channelAmplitude()")
- G is the deviation gain (set by "modulationAmplitudeConfig()"). Note that G is only used for 16-bit waveforms.
- AWG(t) is the normalized modulating signal generated by the AWGs.

- $\cos[2\pi(f_c t) + \varnothing_c]$ is the carrier signal, generated with the Function Generators.

As an example, for generation of an AM signal with an amplitude of 0.8 Vp and a modulation index (h) equal to 0.5, the settings must be A=0.8 and G=0.32 ($G = h \cdot A^2$).

The output signal of a channel using the amplitude modulator to modulate the offset is described using the following equation:

$$Output(t) = A \cdot \cos(2\pi f_c t + \varnothing_c) + G \cdot AWG(t)$$

where,

- A is the channel amplitude (set by "channelAmplitude()")
- G is the deviation gain (set by "modulationAmplitudeConfig()"). Note that G is only used for 16-bit waveforms.
- AWG(t) is the normalized modulating signal generated by the AWGs.
- $\cos[2\pi(f_c t) + \varnothing_c]$ is the carrier signal, generated with the Function Generators.

Options (Mutual Exclusions for Modulations)

The Amplitude Modulator can be used with signals coming from the Function Generator directly, the AWG, or the frequency/phase modulators. Therefore, the latter can be combined with amplitude modulation at will (although it only makes sense for frequency/amplitude modulations, because for phase/amplitude modulations there is a dedicated IQ operation, "IQ Modulation (Quadrature Modulator Block)" on page 33, which uses the internal amplitude and the angle modulators.

Table 13 Amplitude Modulation options

Functions	Description	Const.	Value
No Modulation	Modulation is disabled. The channel amplitude and offset are only set by the main registers.	AOU_MOD_OFF	0 (default)
Amplitude Modulation	The modulating signal is used to modulate the channel amplitude.	AOU_MOD_AM	1
Offset Modulation	The modulating signal is used to modulate the channel offset.	AOU_MOD_OFFSET	2

Programming Information

Table 14 API for Amplitude Modulation functions

Function	Comments	Details
modulationAmplitudeConfig	Configures the amplitude modulator	modulationAmplitudeConfig()
AWG functions	Control the modulating signal	AWG functions under SD_AOU functions
FG functions	Control the carrier signal	Channel functions under SD_AOU functions

1.3.3: IQ Modulation (Quadrature Modulator Block)

The output signal of the Function Generator can be modulated simultaneously in amplitude and angle (frequency or phase). This allows the creation of an IF signal with amplitude modulation while the frequency is scanned. However, the most common use of dual modulations is the amplitude and angle modulation decomposed to in-phase (I) and quadrature (Q) components, commonly known as IQ modulation. In order to make the creation of IQ modulations easier, there are dedicated programming functions for that purpose.

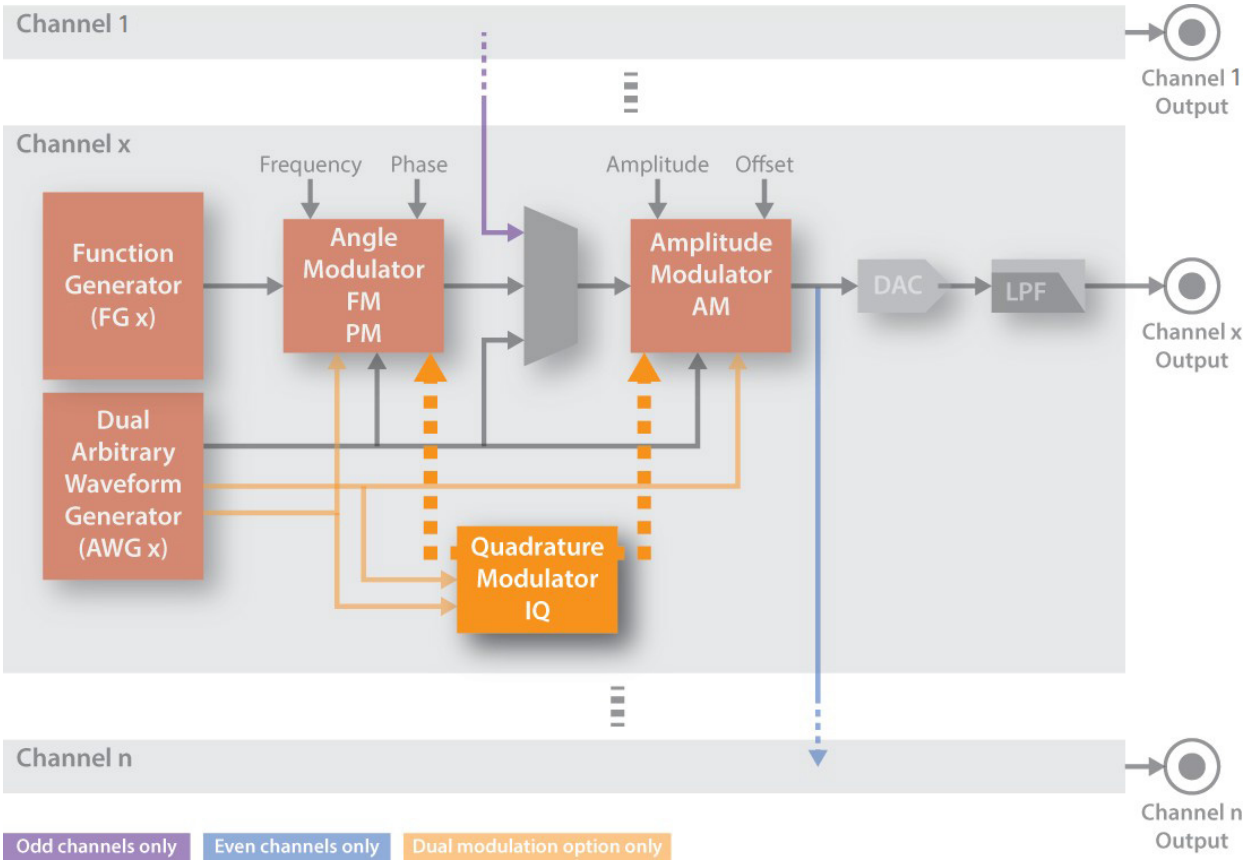


Figure 15 Block diagram depicting functionality of Quadrature (IQ) Modulator

The modulating signals are generated using the IQ Modulation (Quadrature Modulator Block) associated to that particular channel, for example, AWG1 for channel 1. In this case, the waveform loaded into the AWG is composed by the two waveforms (for example, I and Q components, see "AWG FlexCLK Synchronization (models with variable sampling rate)" on page 24).

NOTE IQ waveforms are sometimes called complex waveforms due to the mathematical form I(t)+iQ(t).

The output signal of a channel using the IQ modulator is described using the following equation:

$$Output(t) = (A / \sqrt{2}) \cdot [AWG_I(t) \cdot cos(2\pi f_c t + \emptyset_c) - AWG_Q(t) \cdot sin(2\pi f_c t + \emptyset_c)]$$

where,

- A is the channel amplitude (set by "channelAmplitude()")
- $AWG_I(t)$ and $AWG_Q(t)$ are the normalized modulating signals generated by the AWGs.
- $\cos[2\pi(f_c t) + \emptyset_c]$ and $\sin[2\pi(f_c t) + \emptyset_c]$ are the carrier signals, generated with the Function Generators.

You may also use the IQ modulator with amplitude and phase components:

$$Output(t) = A \cdot AWG_A(t) \cdot cos(2\pi f_c t + \emptyset_c + \pi/2 \cdot AWG_\emptyset(t))$$

where,

- A is the channel amplitude (set by "channelAmplitude()")
- $AWG_A(t)$ and $AWG_\emptyset(t)$ are the normalized modulating signals generated by the AWGs.
- $\cos[2\pi(f_c t) + \emptyset_c]$ is the carrier signal, generated with the Function Generators.

Mathematical Background of IQ Modulation

The IQ modulation is based on the decomposition of the output signal in sinus and cosinus:

$$A \cdot cos(2\pi f_c t + \emptyset_c + \emptyset) = A \cdot cos(2\pi f_c t + \emptyset_c)cos(\emptyset) - A \cdot sin(2\pi f_c t + \emptyset_c)sin(\emptyset)$$

where,

- $I = A \cdot \cos(\emptyset)$
- $Q = A \cdot \sin(\emptyset)$

The IQ components become very useful to set the amplitude (A) and the phase (Ø) of the output signal, resulting in IQ modulation (or QAM, Quadrature Amplitude Modulation).

$$A \cdot \cos(2\pi f_c t + \emptyset_c + \emptyset) = I \cdot \cos(2\pi f_c t + \emptyset_c) - Q \cdot \sin(2\pi f_c t + \emptyset_c)$$

where,

- $I = A \cdot \cos(\emptyset)$
- $Q = A \cdot \sin(\emptyset)$

Mutual Exclusions for Modulations

Internally, IQ modulation uses the amplitude and the angle modulators so they cannot be used when the module is working in IQ mode.

Programming Information

Table 15 API for IQ Modulation functions

Function	Comments	Details
modulationIQConfig	Configures the IQ modulator	modulationIQconfig()
AWG functions	Control the modulating signal	AWG functions under SD_AOU functions
FG functions	Control the carrier signal	Channel functions under SD_AOU functions

Section 1.4: Working with I/O Triggers

The M3201A/M3202A PXIe AWG has 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. Because the Keysight M3201A and M3202A PXIe AWGs have different output latencies, triggering both at the same time from HVI will always result in a 65 to 70 ns offset between their outputs.

Table 16 I/O Trigger types and corresponding values

Type	Description	Name	Value
Trigger Output (readable)	TRG operates as a general purpose digital output signal, that can be written by the user software.	AOU_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.	AOU_TRG_IN	1

Table 17 Trigger Synchronization types and corresponding values

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 1.5: Working with Clock System

The M3201A/M3202A PXIe AWG 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.

1.5.1: CLK Output Options

Table 18 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

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

The sampling frequency of the M3201A/M3202A PXIe AWG (CLKsys frequency) can be changed using the advanced clocking system.

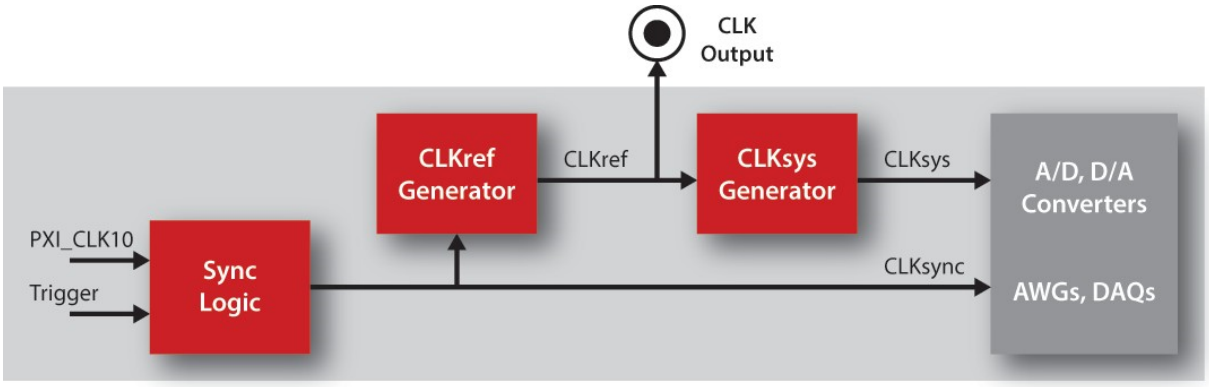


Figure 16 Block Diagram depicting the Advanced Flex Clock System in PXIe AWGs

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 M3201A/M3202A PXIe AWG.
- 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 19 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

1.5.3: CLKref Frequency in AWG Modules with Option CLV

NOTE The hardware option CLV is currently not supported in M3201A/M3202A AWG modules.

In M3201A-CLV modules, CLKref frequency (freqCLKref) changes as follows, as a function of CLKsys frequency (freqCLKsys):

- Between 400 MHz and 500 MHz => freqCLKref = freqCLKsys/40
- Between 300 MHz and 400 MHz => freqCLKref = freqCLKsys/30
- Between 200 MHz and 300 MHz => freqCLKref = freqCLKsys/20
- Between 150 MHz and 200 MHz => freqCLKref = freqCLKsys/15
- Between 100 MHz and 150 MHz => freqCLKref = freqCLKsys/10
- Below 100 MHz => error

In M3202A-CLV modules, CLKref frequecy (freqCLKref) changes as follows, as a function of CLKsys frequency (freqCLKsys):

- Between 800 MHz and 1000 MHz => freqCLKref = freqCLKsys/80
- Between 600 MHz and 800 MHz => freqCLKref = freqCLKsys/60
- Between 400 MHz and 600 MHz => freqCLKref = freqCLKsys/40
- Below 400 MHz => error

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 elements in KS2201A PathWave Test Sync Executive Software and its API, refer to the *KS2201A PathWave Test Sync Executive 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 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 KS2201A 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 20 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 20 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. This software has 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 17 & Figure 18 display the AWG and Digitizer specific HVI definitions, which are added to the SD1 library.

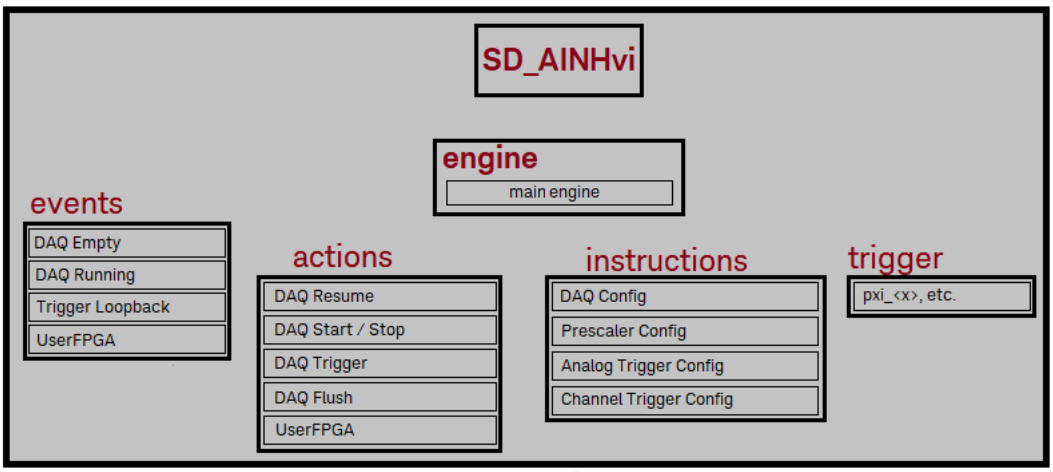


Figure 17 M310xA specific HVI definitions

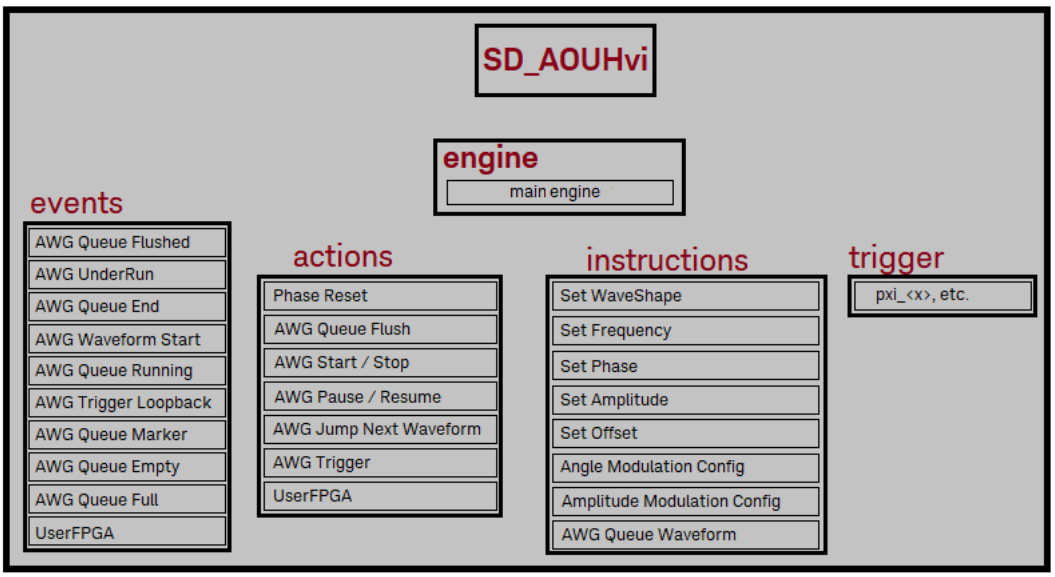


Figure 18 M320xA specific HVI definitions

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_AOU 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_AOU 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_AOU 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 keysight_hvi as kthvi

sys.path.append(r'C:\Program Files (x86)\Keysight\SD1\Libraries\Python')

import keysightSD1


awg = keysightSD1.SD_AOU()

awg.openWithSlot("M3201A", 1, 4)


sys_def = kthvi.SystemDefinition("mySystem")

sys_def.chassis.add_auto_detect()

sys_def.engines.add(awg.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


instrLabel = "awgSetWaveshape"
```

```
instruction0 = sequence.add_instruction(instrLabel, 20,
awg.hvi.instruction_set.set_waveshape.id)

instruction0.set_parameter(awg.hvi.instruction_set.set_waveshape.channel.id, test_channel)

instruction0.set_parameter(awg.hvi.instruction_set.set_waveshape.value.id,
awg.hvi.instruction_set.set_waveshape.value.AOU_SINUSOIDAL)


instrLabel = "awgSetFrequency"

instruction1 = sequence.add_instruction(instrLabel, 20,
awg.hvi.instruction_set.set_frequency.id)

instruction1.set_parameter(awg.hvi.instruction_set.set_frequency.channel.id, test_channel)

instruction1.set_parameter(awg.hvi.instruction_set.set_frequency.value.id, 1e7)


instrLabel = "awgSetAmplitude"

instruction1 = sequence.add_instruction(instrLabel, 20,
awg.hvi.instruction_set.set_amplitude.id)

instruction1.set_parameter(awg.hvi.instruction_set.set_amplitude.channel.id, test_channel)

instruction1.set_parameter(awg.hvi.instruction_set.set_amplitude.value.id, 1)


# 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...")
```

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 M3202A AWG cards, each M3202A 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 21 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 22 highlights such differences.

Table 22 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.

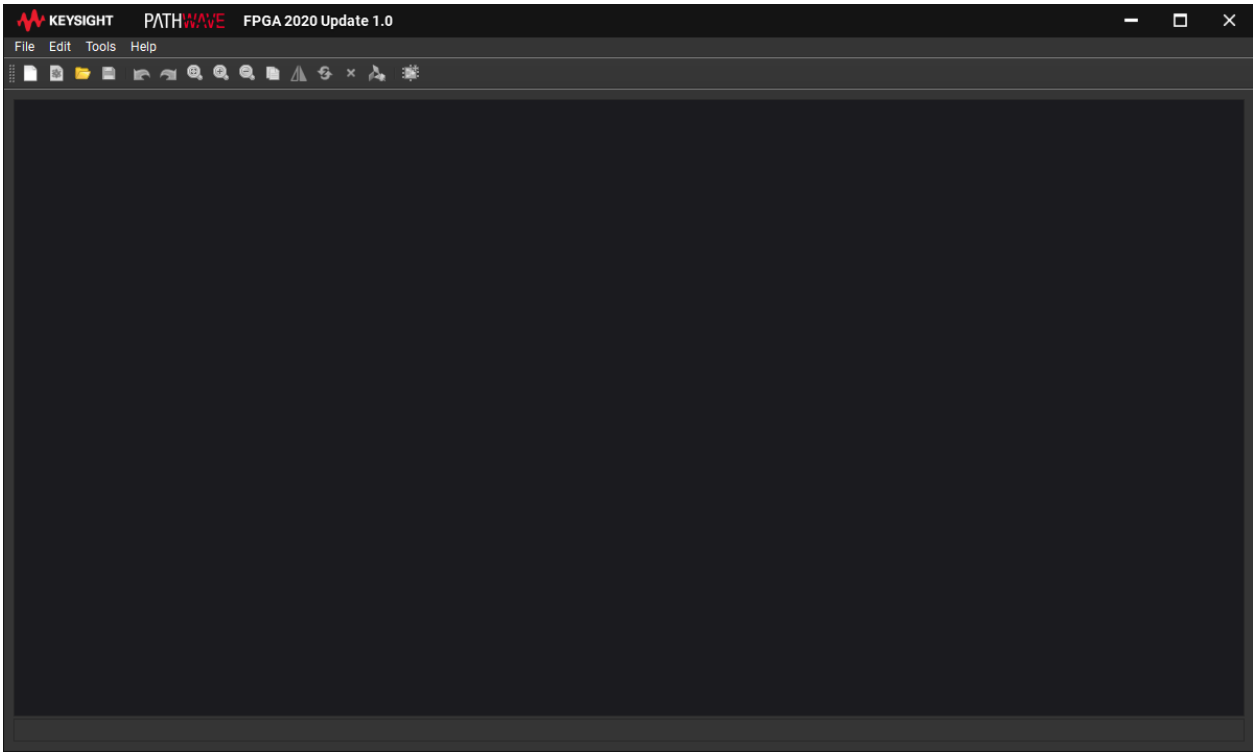


Figure 19 Default window for PathWave FPGA 2020 Update 1.0 software

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.

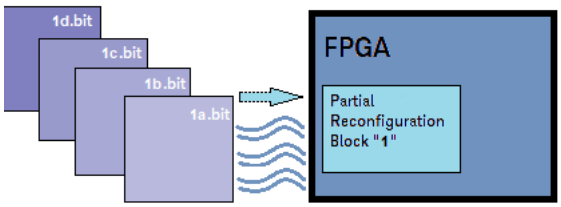


Figure 20 Block diagram depicting partial reconfiguration using bitstream files

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.

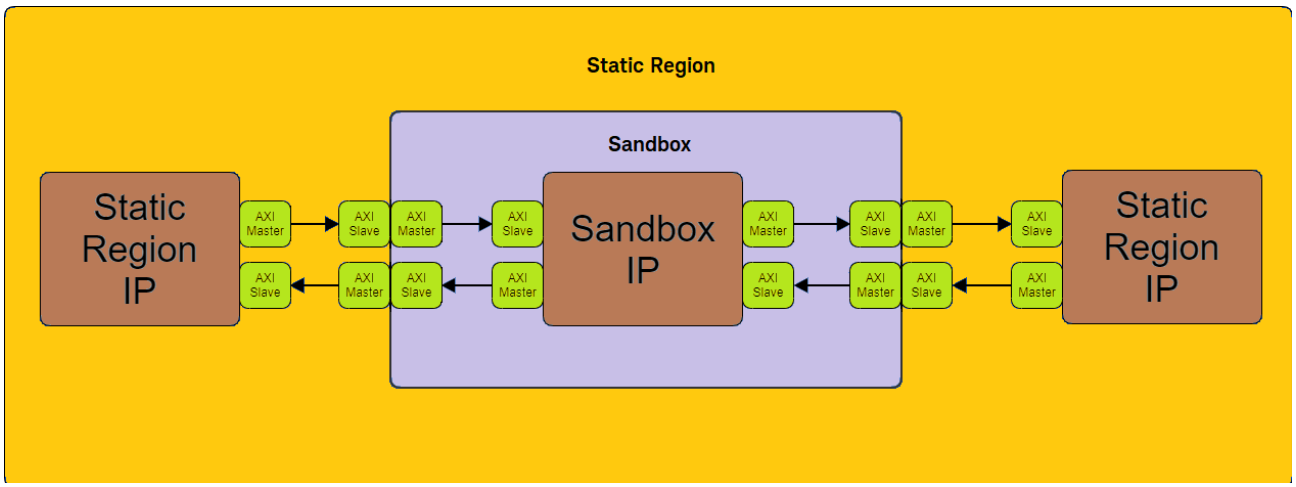


Figure 21 Block diagram representing layout of Static and Sandbox regions

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 22 indicates how the bitstream file is generated using both the PathWave FPGA software and BSP together for any specific supported PXIe module.

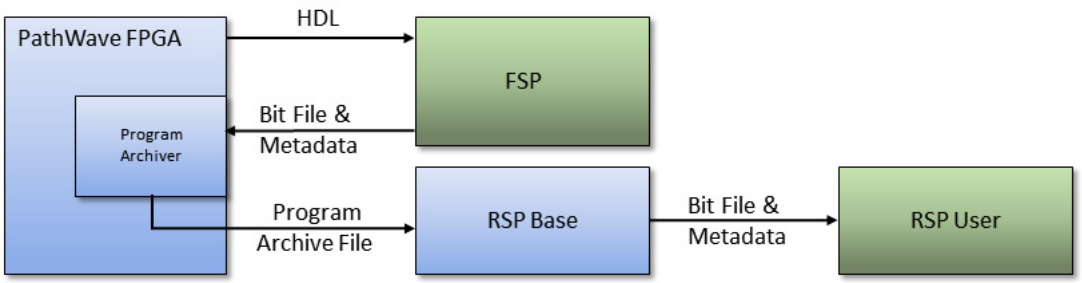


Figure 22 Bitstream file compilation flow using PathWave FPGA and BSP

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 55.

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 64.

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*.

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

A new sandbox project dialog appears.

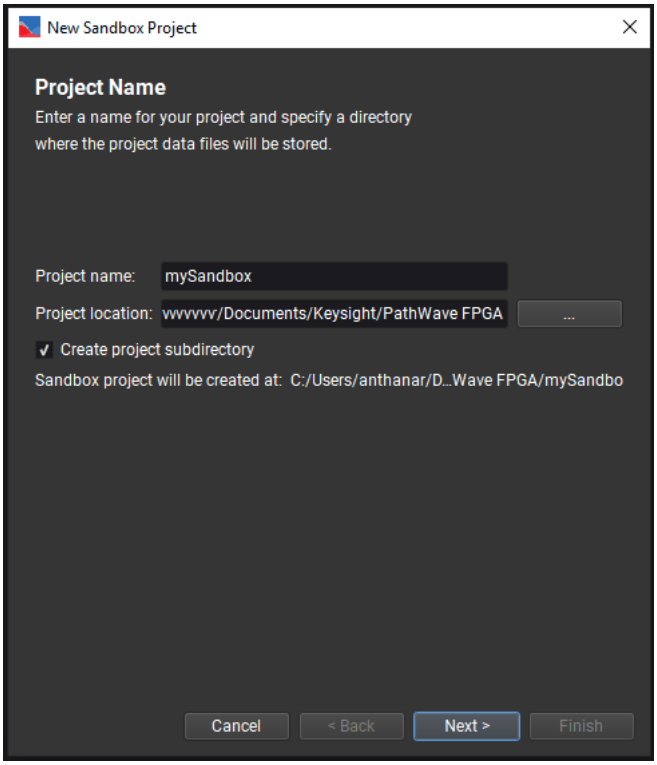


Figure 23 Creating a new Sandbox project

2 Click **Next >**.

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

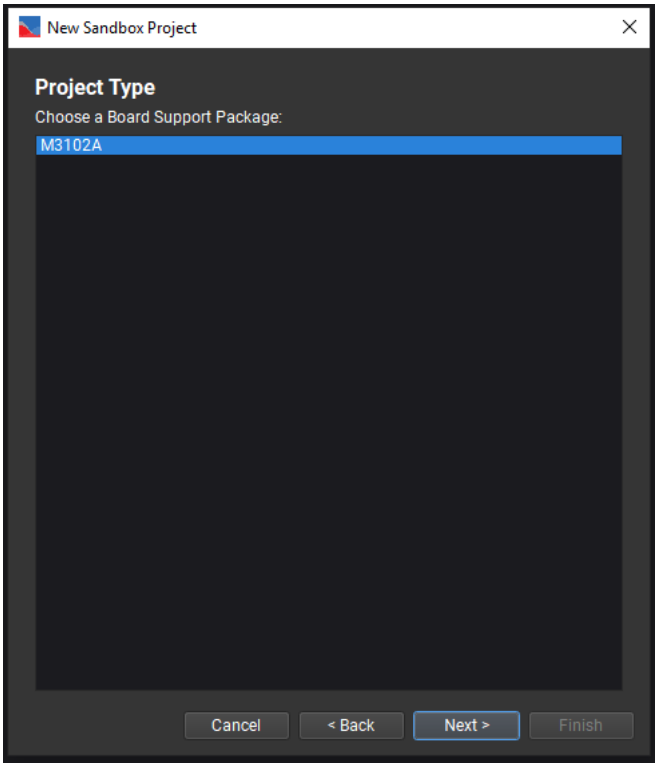


Figure 24 Selecting the module-specific BSP

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.

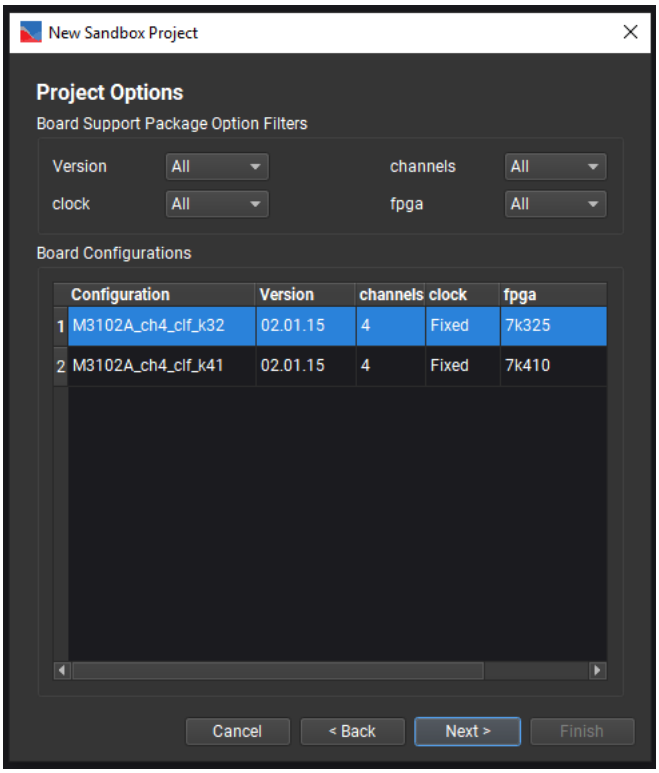


Figure 25 Selecting various options for FPGA design logic

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.

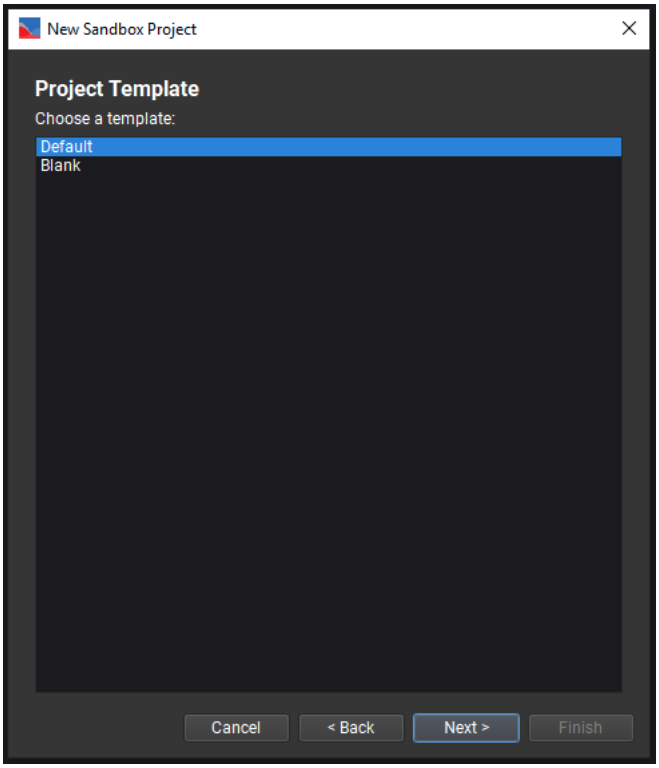


Figure 26 Selecting the PathWave FPGA project template

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**.

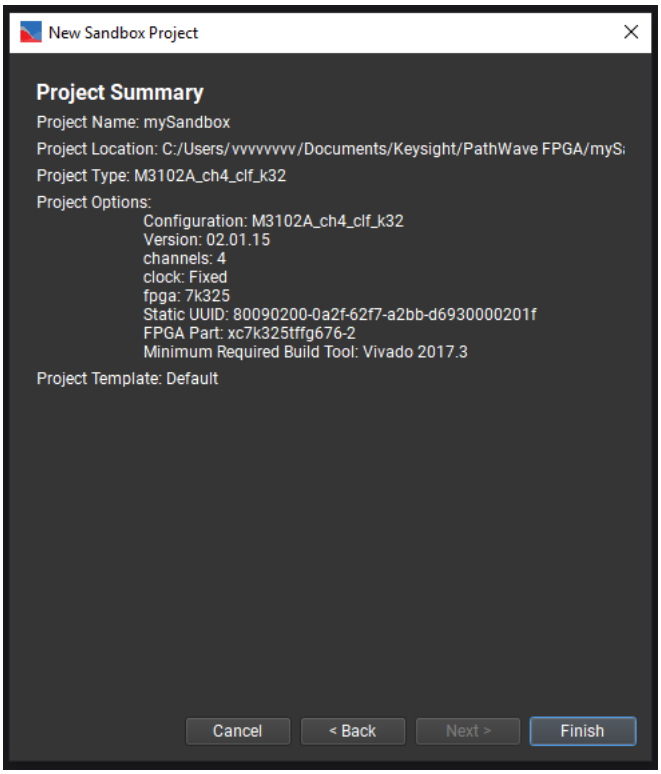


Figure 27 Viewing the project summary information

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

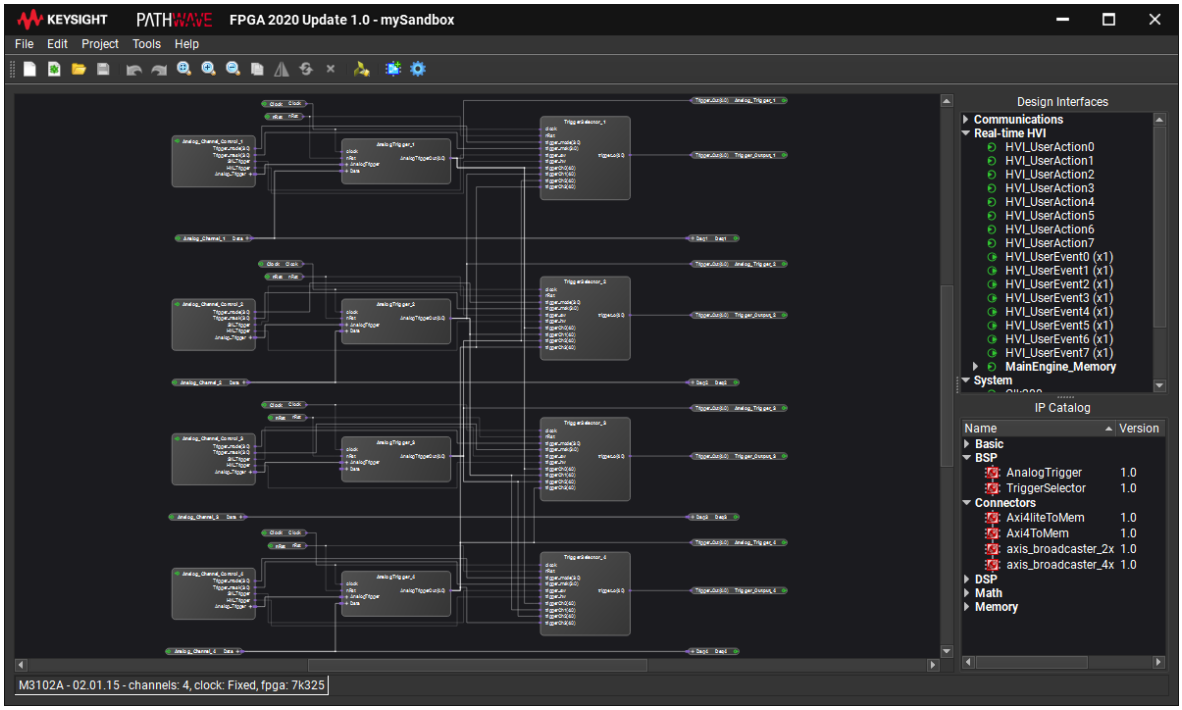


Figure 28 Viewing FPGA design blocks in the default template

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 29.

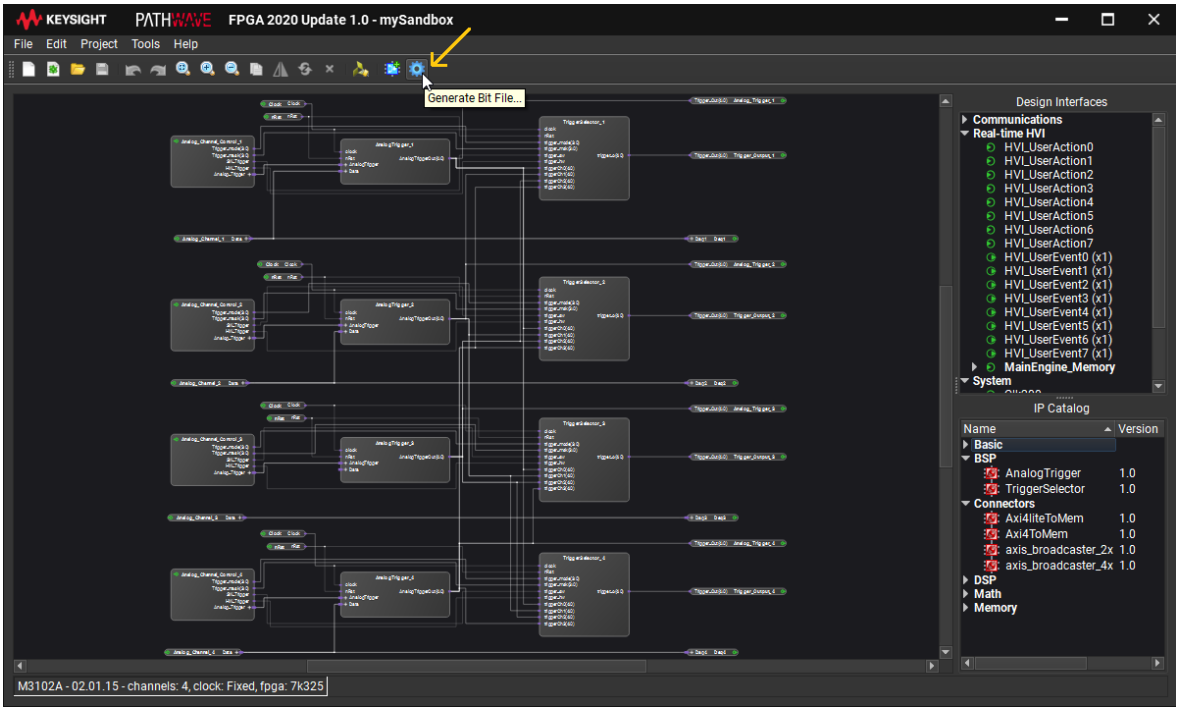
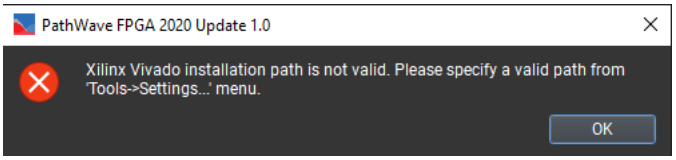


Figure 29 Initiating Bit File generation for the selected module

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**.

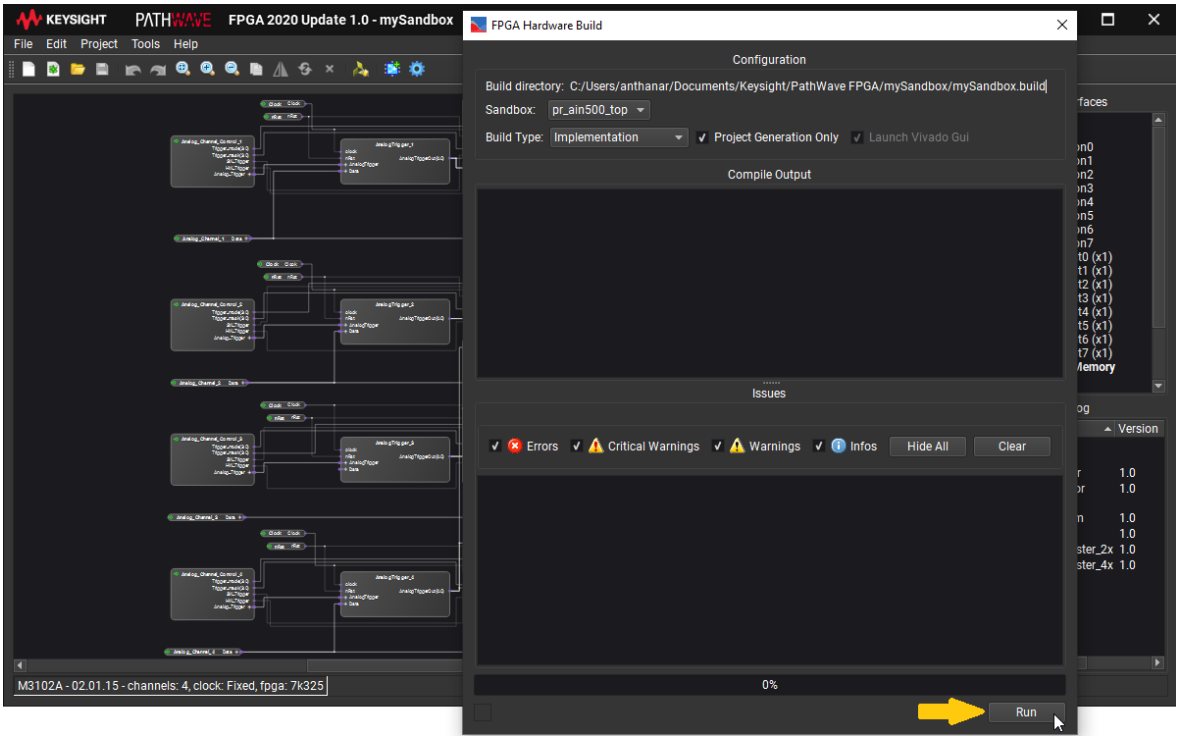


Figure 30 Generating FPGA Hardware Build

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

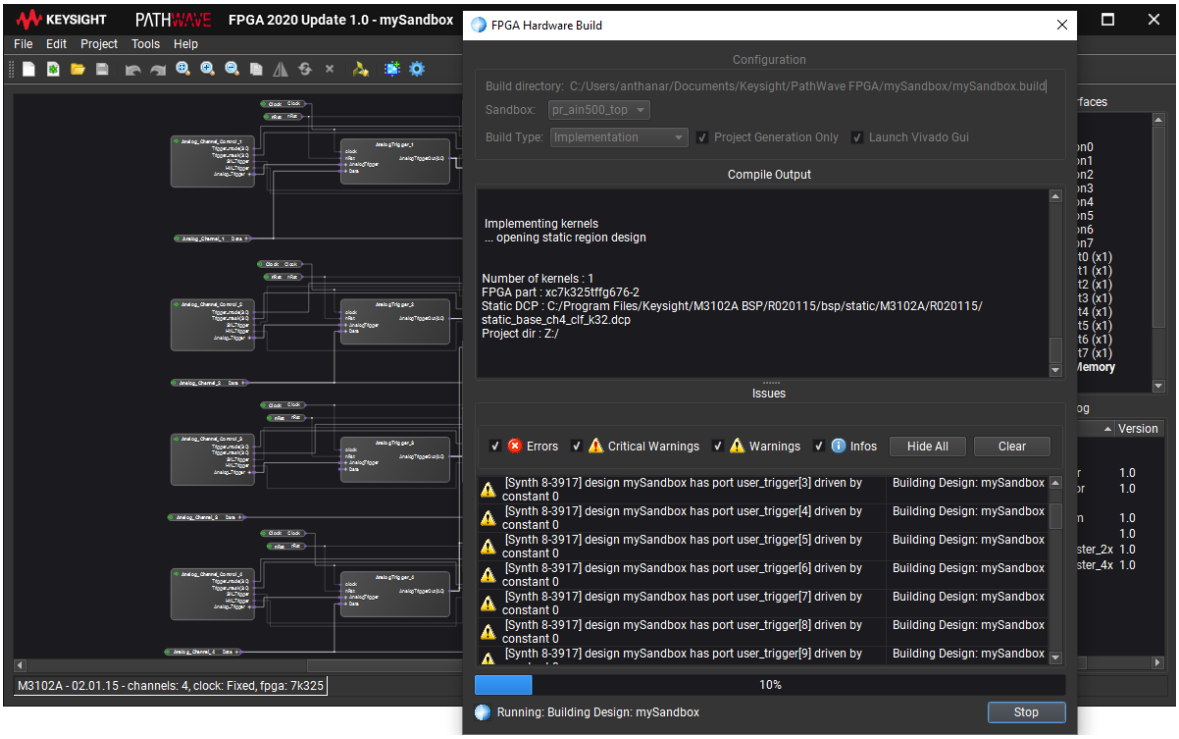


Figure 31 Progress status for the k7z file generation

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 32 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.

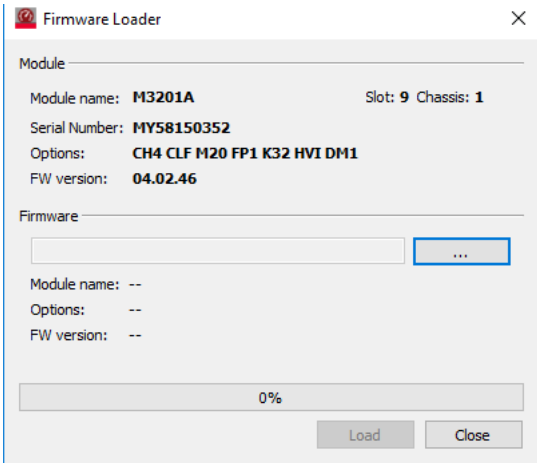


Figure 33 Firmware Loader window for the selected module

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 - Basic workflow

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 < 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 *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**.

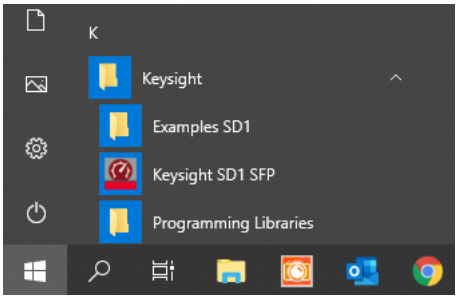


Figure 34 Launching SD1 SFP software from the Start menu

The Keysight SD1 3.x SFP software window is displayed, as shown in Figure 35 and Figure 36.

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 35, the soft front panel for the M3201A AWG and M3302A (AWG & Digitizer) Combo are displayed by default, which indicates these modules are connected.

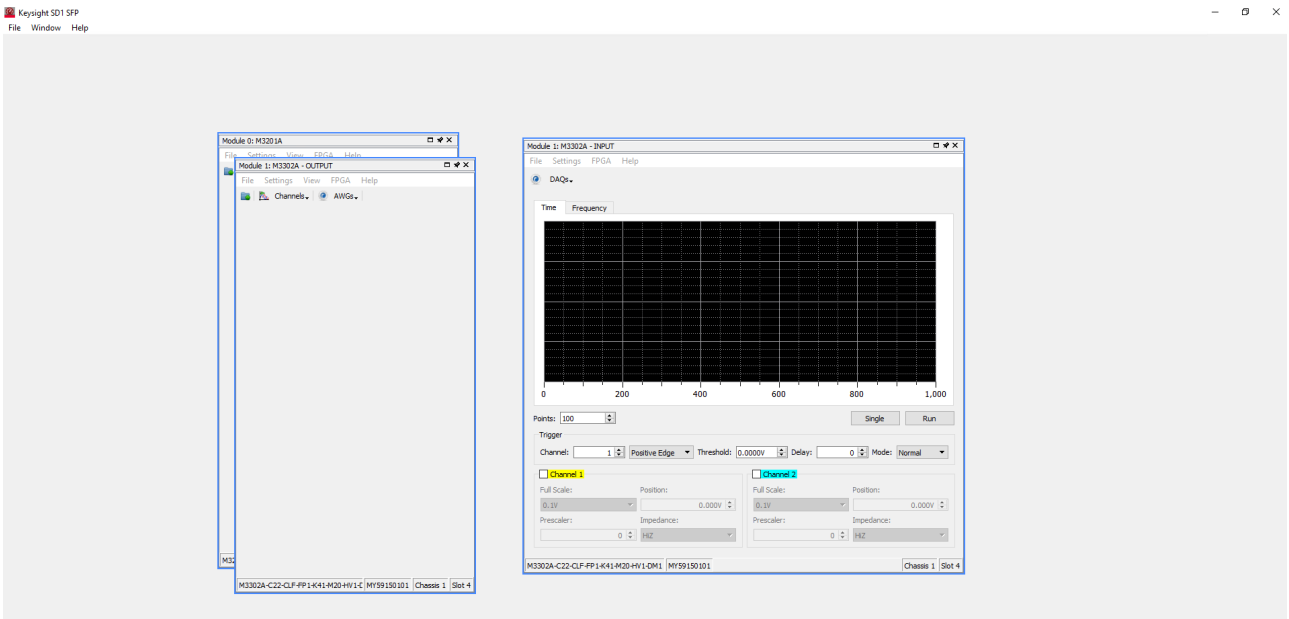


Figure 35 SD1 SFP software window (online mode with M3201A and M3302A cards connected)

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 36.

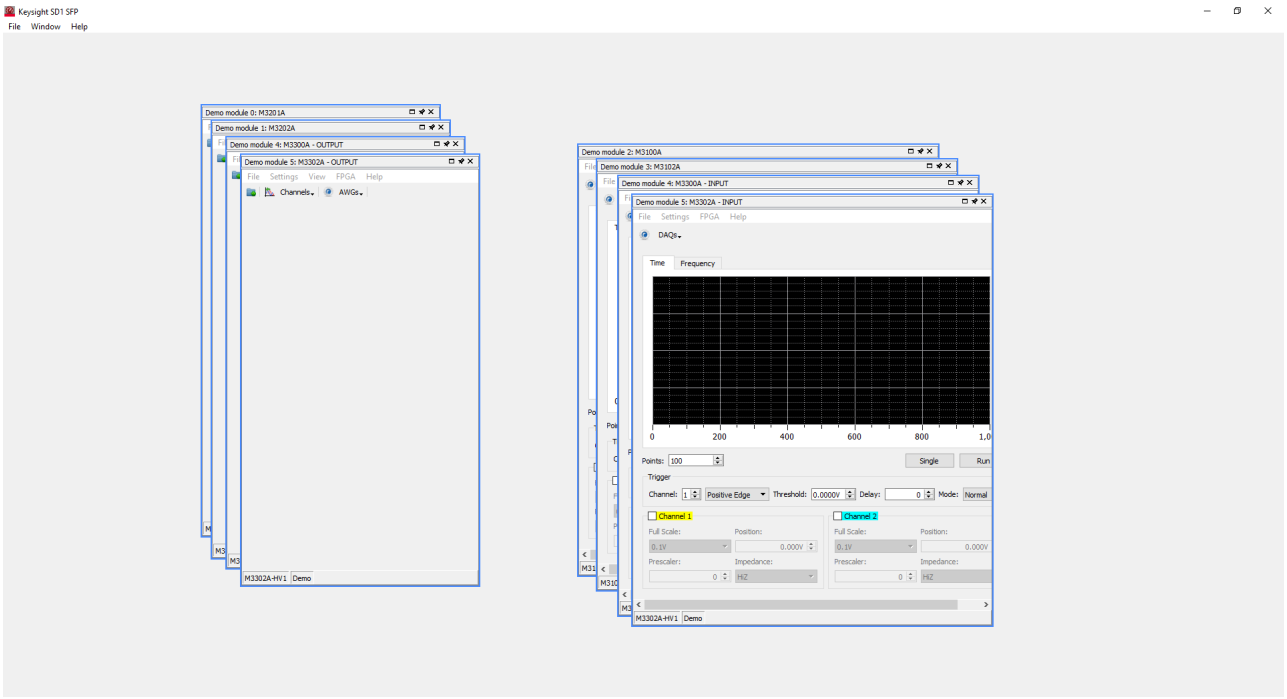


Figure 36 SD1 SFP software (offline mode without any card connected)

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

Figure 37 SD1 SFP main window

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

1. File

Figure 38 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

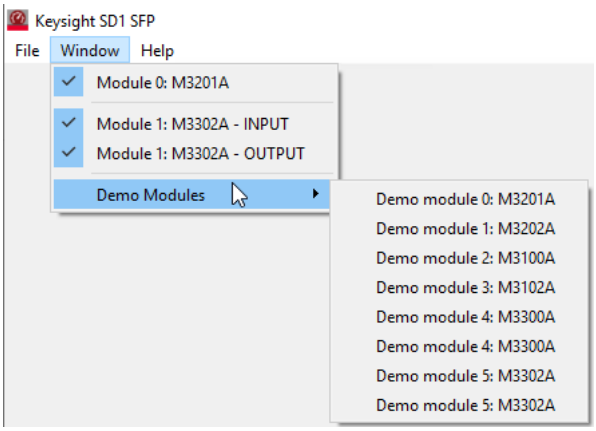


Figure 39 Controls in the Window menu

- 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

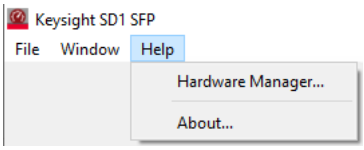


Figure 40 Controls in the Help menu

- Hardware Manager...—Click **Help** > **Hardware Manager...** to launch the **Hardware Manager** window. See "Understanding features in Hardware Manager" on page 78 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.

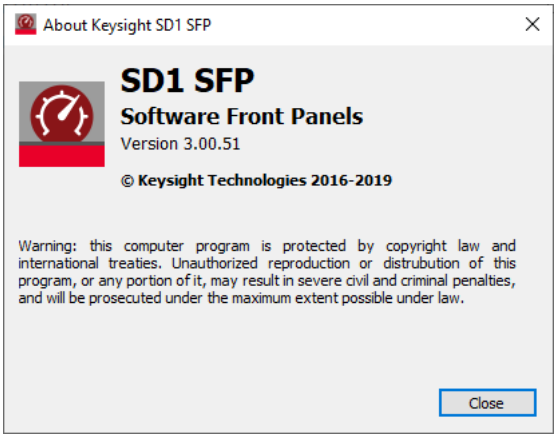


Figure 41 About Keysight SD1 SFP window

4.3.2: Understanding features in Hardware Manager

The features in this window are described below in the order shown in Figure 42.

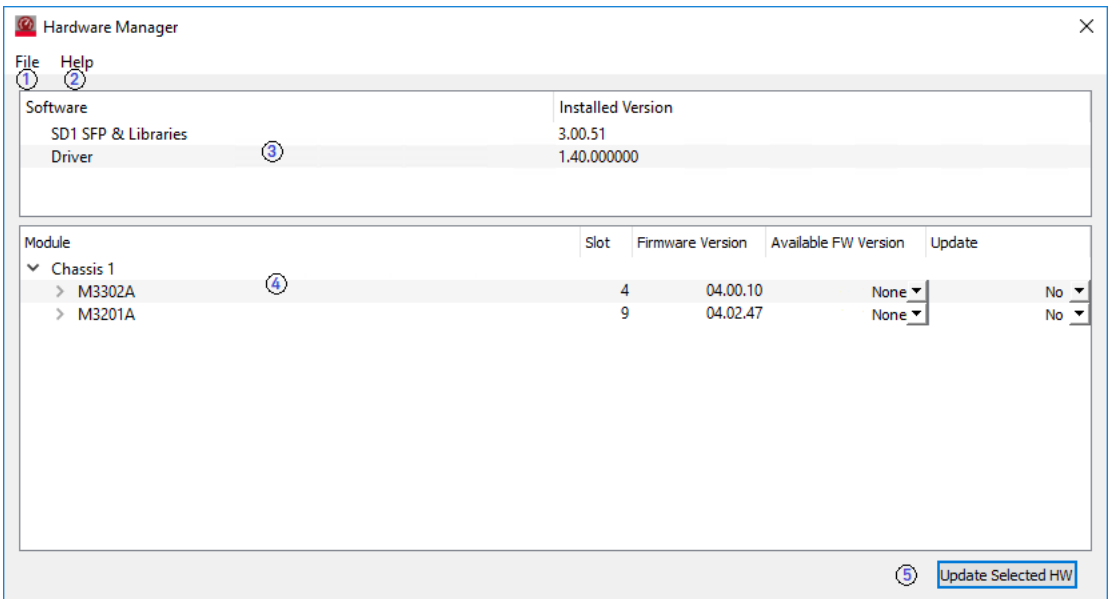


Figure 42 Hardware Manager window

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

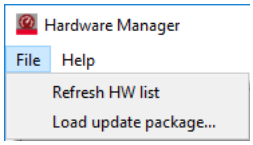


Figure 43 Controls in the File menu for Hardware Manager

- **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 44. 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.

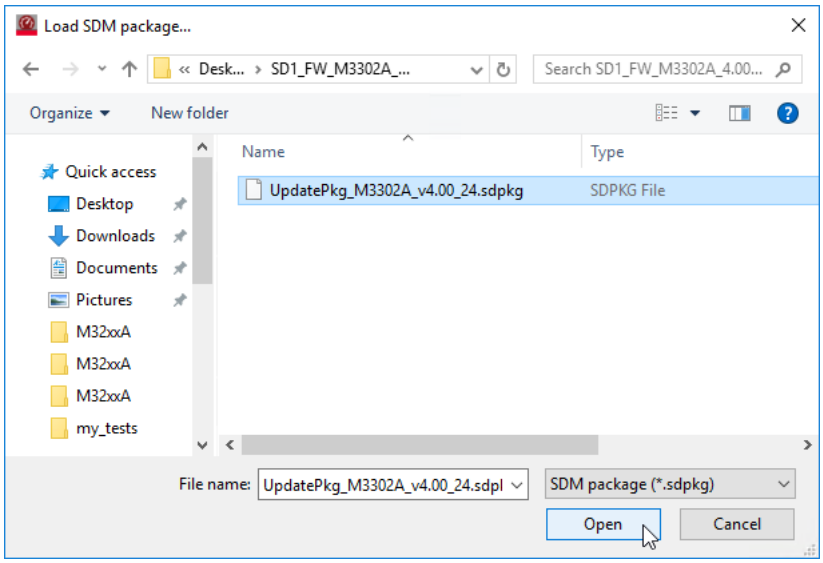


Figure 44 Accessing firmware update file from Load SDM package window

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

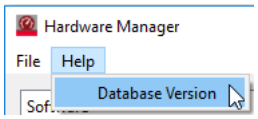


Figure 45 Control in the Help menu for Hardware Manager

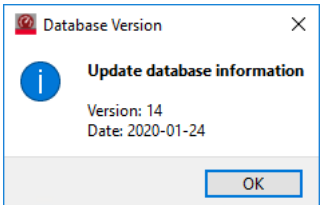


Figure 46 Database Version window

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()".

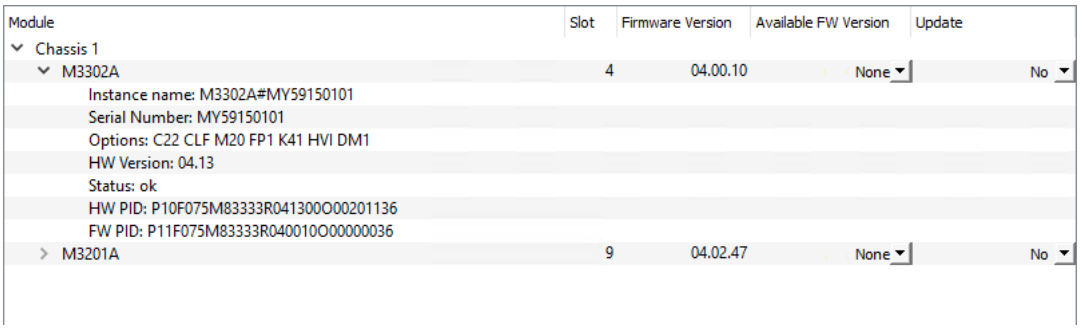


Figure 47 Information pertaining to each module in the Hardware Manager window

- 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 AWG SFP features & controls

For all those modules that are connected to the Chassis, the respective soft front panel for the M3201A/M3202A PXIe AWG and that for the AWG part of the M3300A/M3302A Combo appear automatically when SD1 SFP is launched. By default, the SFP window for the AWG appears blank, that is, without any configuration parameters, as shown for M3201A module in Figure 48.

NOTE **The controls are common across all AWG SFP windows. However, the number of AWGs / Channels that appear depends on the module type. M32xxA modules support only 4 AWGs/Channels whereas the M33xxA Combos support only 2 AWGs/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.
- most of the controls appear within the main menu items, which are explained in the following section.
- a shortcut for some of the controls, which are available under the main menu items, are displayed below the main menu.

Figure 48 SFP for the connected M3201A after SD1 SFP software is launched

4.4.1: Understanding AWG SFP main menu features & controls

The controls under each menu item are further described in the order displayed in Figure 48. Images in this section pertain mainly to M3201A AWG module's SFP only, but are applicable across all AWG module SFP windows.

1. File

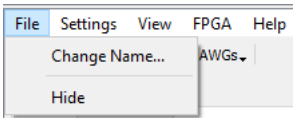


Figure 49 Controls in the File menu of the AWG SFP

- Change Name...—Click **File** > **Change Name...** to open the **Change name module** dialog box. This feature works in the same manner for Digitizer SFPs also.

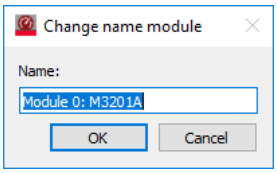


Figure 50 Change name module dialog box

  - 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 76.

2. Settings

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

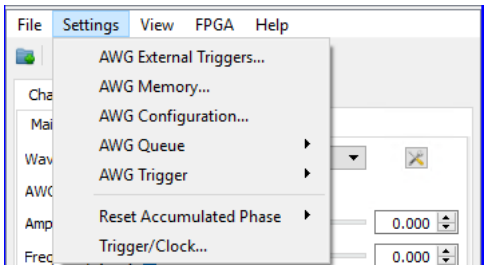


Figure 51 Controls in the Settings menu of the AWG SFP

- AWG External Triggers...—Click **Settings** > **AWG External Triggers...** to open the **Configure AWG Triggers** dialog box. See "Setting the Configure AWG Triggers dialog" on page 87 for more description.
  - Related API function: "AWGtriggerExternalConfig()"

- AWG Memory...—Click **Settings** > **AWG Memory...** to open the **Onboard waveform memory** dialog box. See "Setting the Onboard Waveform memory dialog" on page 87 for more description.
- AWG Configuration...—Click **Settings** > **AWG Configuration...** to open the **AWG Configuration** dialog box. See "Setting the AWG Configuration dialog" on page 89 for more description.
  - Related API functions: "AWGqueueSyncMode()", "AWGqueueConfig()", "AWGqueueConfigRead()"
- AWG Queue—Click **Settings** > **AWG Queue** > **AWGn...** (where, n = AWG number) to open the corresponding **AWG n Waveform Queue** dialog box. See "Setting the AWG Waveform Queue dialog" on page 90 for more description.
  - Related API functions: "AWGqueueWaveform()", "AWGflush()"

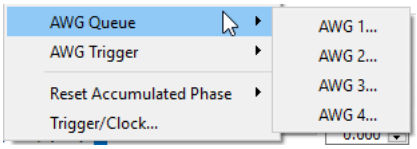


Figure 52 Controls in AWG Queue sub-menu

- AWG Trigger—Click **Settings** > **AWG Trigger** > **Send Trigger** to send a trigger to the selected AWGs in the same sub-menu. By default, all AWGs are selected.
  - Related API functions: "AWGtrigger()", "AWGtriggerMultiple()"

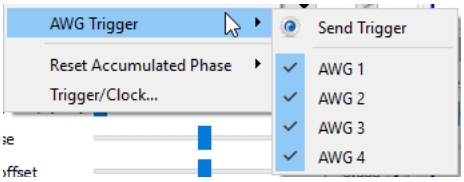


Figure 53 Controls in AWG Trigger sub-menu

  - Click one or more **AWG n** (where, n = AWG 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 AWGs.

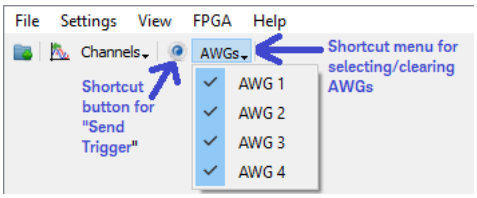


Figure 54 Shortcut controls for sending trigger to AWGs

- Reset Accumulated Phase—Click **Settings** > **Reset Accumulated Phase** > **Reset Phase** to reset any accumulated phase on the selected Channels in the same sub-menu. By default, all Channels are selected.
  - Related API functions: "channelPhaseReset()", "channelPhaseResetMultiple()"

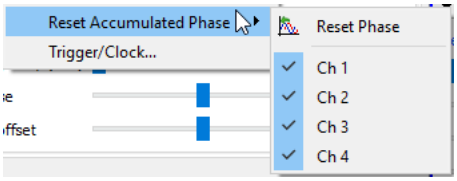


Figure 55 Controls in Reset Accumulated Phase sub-menu

- Click one or more **Ch n** (where, n = Channel number) entries to clear the tick mark; thereby, refraining them from undergoing phase reset when you click **Reset Phase**.
- Alternatively, you may use the shortcut controls to reset phase on the Channels.

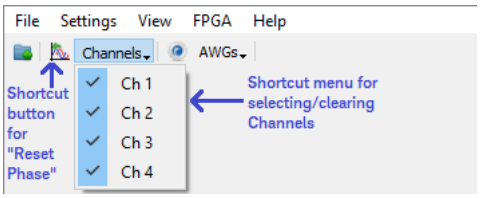


Figure 56 Shortcut controls for resetting accumulated phase on Channels

- Trigger/Clock...—Click **Settings** > **Trigger/Clock...** to open the **Trigger / Clock Settings** dialog box. See "Setting the Trigger / Clock Settings dialog" on page 91 for more description.
  - Related API functions: "triggerIOconfig()", "clockIOconfig()"

3. View

Figure 57 Control in the View menu of the AWG SFP

- New Panel...—Click **View** > **New Panel...** to open the **Add new panel** dialog box.

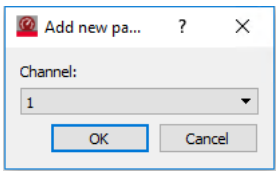


Figure 58 Add new panel dialog box

- From the **Channel:** drop-down options, select a Channel number, which you wish to configure on the connected AWG module.
- Click **OK**.

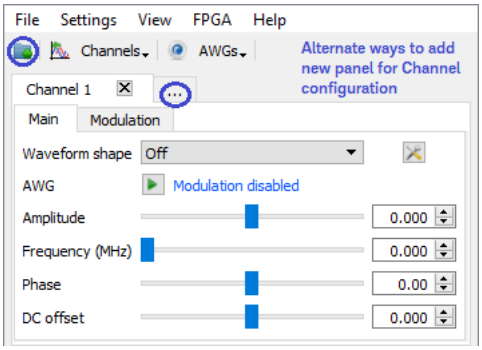


Figure 59 New panel for Channel configuration dialog box along with its various shortcut controls

- Repeat the previous steps to add more AWG Channels.
- Alternatively, you may click the shortcut icons shown in Figure 59 to open the **Add new panel** dialog box.

The description for the attributes in the channel configuration panel is covered in "Setting up the Channel configuration panel" on page 93.

4. FPGA

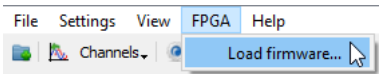


Figure 60 Control in the FPGA menu of the AWG SFP

- 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 64.
  - Related API function: "FPGAload()"

5. Help

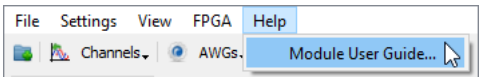


Figure 61 Control in the Help menu of the AWG SFP

- Module User Guide...—Click **Help** > **Module User Guide...** to view the documentation corresponding to AWG/Combo modules in Online Help format. For more information, see "Accessing Online Help for SD1 3.x software" on page 236.

4.4.2: Configuring AWG and Channel Setting dialogs

1. Setting the Configure AWG Triggers dialog

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

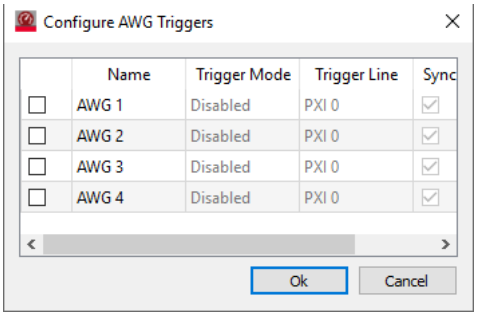


Figure 62 Configure AWG Triggers dialog box

1 Select one or more rows to enable external triggering for AWG n, where n = AWG 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 AWG.
- **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 trigger source can be used to send the trigger signal.

4 **Sync**—By default, the setting to synchronize the AWG with the trigger source is enabled. You may clear the check box to disable synchronization.

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

2. Setting the Onboard Waveform memory dialog

You must configure the **Onboard waveform memory** dialog box to load a waveform (in **.csv* file format) onto the module's onboard RAM.

By default, the Keysight SD1 SFP software provides some waveform files in **.csv* format as examples during installation. The default location for these files is *C:\Users\Public\Documents\Keysight\SD1\Examples\Waveforms*. You may also create and save waveform files of your own and save them in **.csv* format.

Figure 63 Onboard waveform memory dialog box

1 Click '**+**' to view and add one or more waveform files from the local disk of your machine.

Alternatively, if you click the cell in the **Source** column, a folder icon appears, which you can click to perform the same action.

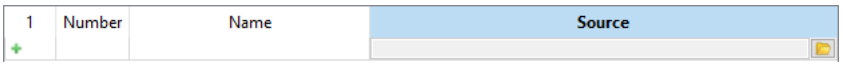


Figure 64 Alternate way of browsing for waveform files

2 From the **Open** window that appears, select a pre-defined waveform or navigate to select any other waveform file of your choice and click **Open**.

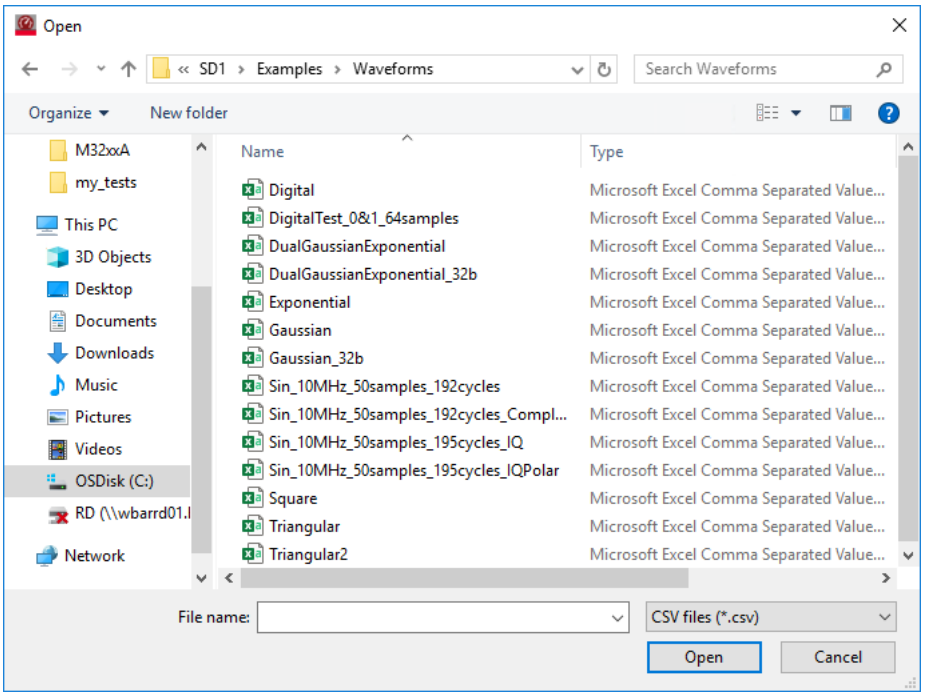


Figure 65 Default waveform files folder displayed in Open window

The waveform file is added to the first row of the **Onboard waveform memory** dialog box.

The SD1 SFP software auto-assigns numbers in the **Number** column to each waveform file you add. In the **Name** column, it displays the name defined for the waveform type in the CSV file. The cell in the **Source** column displays the file location on the local disk of your machine.

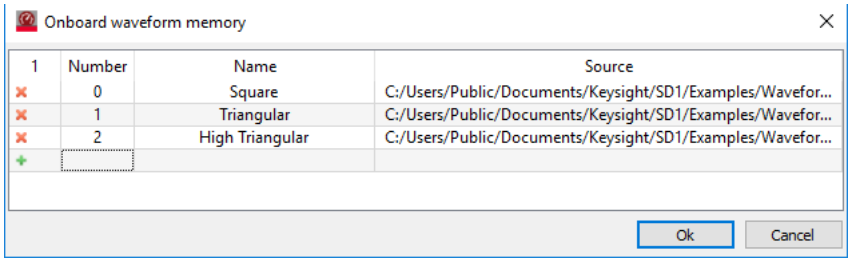


Figure 66 Onboard waveform memory dialog box appearance after waveform files are added

3 In this dialog box, you may modify the following elements:

a Click '**x**' that appears in column '**1**' to remove a row. The SD1 SFP software prompts a confirmation dialog. Click **Yes** to remove else **No** to retain.

b **Number**—Indicates the register number on the onboard memory where the waveform is loaded. Double-click to edit one or more numbers in their respective cell to change the order of listing of waveforms. The list is rearranged automatically to appear in ascending order of numbers.

c **Name**—Indicates the waveform type defined within the CSV file that you have opened. If required, double-click to edit one or more names in their respective cell. Note that changing the waveform name on the **Onboard waveform memory** dialog box does not change the name within the CSV file on disk.

4 Repeat the previous steps to add one or more waveform files to this window and modify number and names, if needed.

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

3. Setting the AWG Configuration dialog

Each AWG block has a flexible waveform queue system that can be used to configure complex generation sequences. To define the queue modes for each AWG, configure the **AWG Configuration** dialog box.

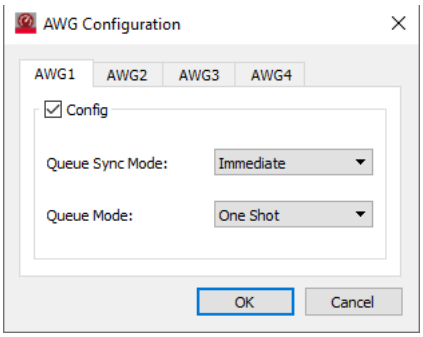


Figure 67 AWG Configuration dialog box

1 Click an **AWGn** (where, n = AWG number) tab to configure the queue modes. Note that the queue configuration options that appear under each AWG tab are the same.

2 Select the **Config** check box so that the **Queue Sync Mode** and **Queue Mode** fields become configurable.

3 **Queue Sync Mode**— This setting configures the synchronization mode of the queue. From the drop-down options, select:

- **Immediate**—(default) Set this option to queue waveform immediately after AWG has started, or when the previous waveform in the queue finishes.
- **Sync 100 (10MHz)**—Set this option to synchronize the AWG to the 10 MHz reference clock.

4 **Queue Mode**—This setting configures the repetition cycles for the complete queue. From the drop-down options, select:
   - **One Shot**—(default) Set this option for the complete queue to be reproduced one time only.
   - **Cyclic**—Set this option for the complete queue to be reproduced for more than one cycles.

5 Repeat steps 1 to 4 to configure AWG Queue modes for other AWG Channels.

6 Click **OK** to save any changes and return to the AWG module window.

4. Setting the AWG Waveform Queue dialog

To define the order in which the waveforms must be queued along with their timing settings on each AWG Channel, use the **AWG n Waveform Queue** (where, n = AWG Channel number). You can queue only those waveforms, which have already been loaded onto the Onboard Waveform Memory.

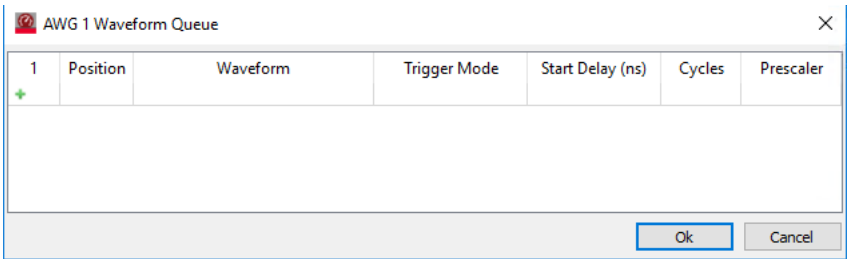


Figure 68 AWG Waveform Queue dialog box

1 Click '**+**' to add one or more waveform files from the Onboard Waveform Memory.

   Alternatively, if you double-click the cell in the **Name** column, it converts into a drop-down field, where you can select a waveform from the list that is already loaded in the Onboard Waveform Memory.

Figure 69 Alternate way of forming the waveform queue list

The SD1 SFP software auto-assigns position numbers in the **Position** column for each waveform that you have included in the list. The **Waveform** column displays the waveform name as you have set in the **Onboard Waveform Memory** window. By default, the **Trigger Mode** column is set to **AUTO** whereas the **Start Delay (ns)**, **Cycles** and **Prescaler** are set to '**0**'.

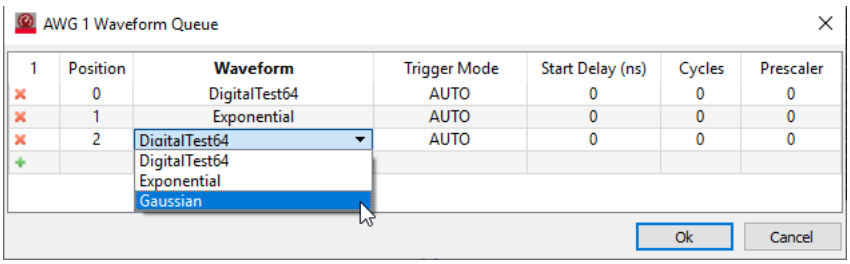


Figure 70 AWG Waveform Queue dialog box appearance after waveforms are added

2 In this dialog box, you may modify the following elements:

a Click '**x**' that appears in column '**1**' to remove a row. The SD1 SFP software prompts a confirmation dialog. Click **Yes** to remove else **No** to retain.

b **Position**—Indicates the position where the waveform must appear in the generated queue. Double-click to edit one or more numbers in their respective cell to change the order of positioning of waveforms in the queue. The list is rearranged automatically to appear in ascending order of numbers.

c **Waveform**—Indicates the waveform name as defined in the **Onboard waveform memory** dialog. If you click '**+**' to add waveforms on each row, it pulls the waveform name that appears in waveform memory register # 0 by default. To change the waveform type, double-click the cell in a specific row and select the waveform name from the drop-down list as shown in Figure 70.

d **Trigger Mode**—Indicates the triggering method to launch the waveforms queued in an AWG. Double-click the cell for each row to modify the value. Select one of the following options:

- **AUTO**—(default) Trigger signal is set to launch waveform immediately after the AWG has started or when the previous waveform in the queue finishes.
- **SW/HVI START**—Set this option so that the AWG is triggered by the condition set in the **Configure AWG Triggers** dialog in the SD1 SFP software, provided that the AWG is running.
- **SW/HVI CYCLE**—Set this option so that the AWG is triggered by the condition set in the **Configure AWG Triggers** dialog in the SD1 SFP software, provided that the AWG is running. However, in this case, trigger is required for each waveform cycle.
- **EXTERNAL START**—Set this option so that the AWG is triggered by an external Trigger source.
- **EXTERNAL CYCLE**—Set this option so that the AWG is triggered by an external Trigger source. However, a trigger is required for each waveform cycle.

e **Start Delay (ns)**—Set the delay between the trigger and the waveform launch in tens of ns.

f **Cycles**—Set the number of times the waveform is played once launched. The value '0' specifies infinite cycles.

g **Prescaler**—Set the waveform prescaler value, to reduce the effective sampling rate by a value of prescaler x 5.

3 Repeat the previous steps to add one or more waveforms to the queue and modify the settings as required.

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

5. Setting the Trigger / Clock Settings dialog

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.

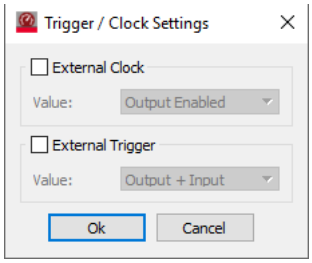


Figure 71 Trigger / Clock Settings dialog box

1 Select the check box for **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**—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 AWG waveform.
- **Output + Input**—Indicates that the signal from the external trigger source is used to trigger both the AWG input and output waveforms.

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

4.4.3: Setting up the Channel configuration panel

The main purpose of the AWG SFP window is to generate one or more waveforms per Channel. When you add one or more new panels to the SFP window, the Channel configuration area panel appears for the selected Channel. This section describes the various elements that are available in the highlighted region in Figure 72. You may add more Channels to configure the waveform settings, as needed. See “Understanding AWG SFP main menu features & controls” on page 83 to add new panels.

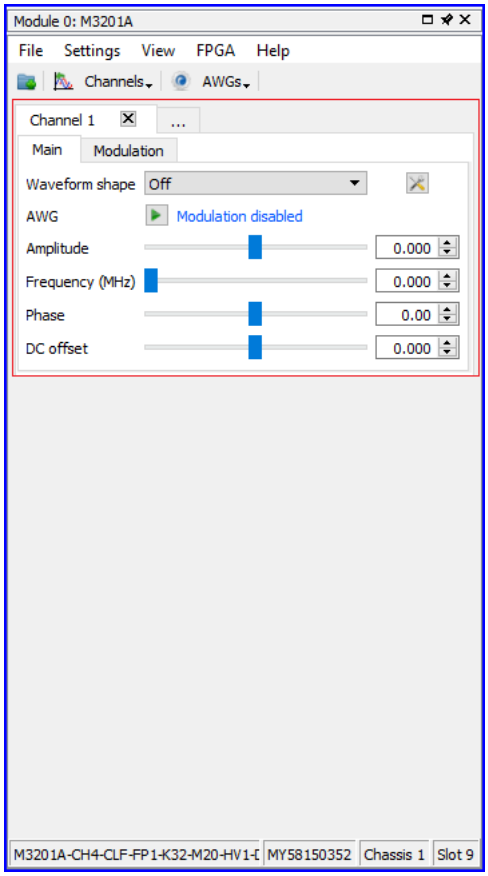


Figure 72 Channel configuration panel in M3xxxA AWG SFP window

Consider Figure 73 to understand each element of the Channel configuration panel in the order shown.

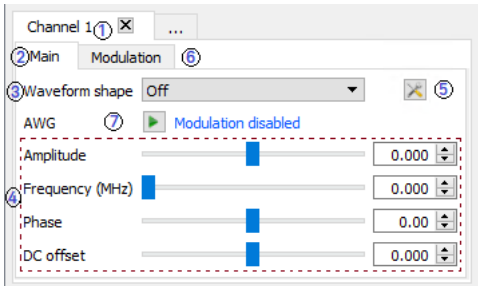


Figure 73 Various features on the Channel configuration panel

1. Channel n

- The title of each panel is set for the Channel number you select when adding a new panel. Here, n represents the Channel number, which is different for the type of M3xxxxA module you select.
  - M3202A AWG 1G – 4 Channels
  - M3200A AWG 500 – 4 Channels
  - M3302A AWG 500 DIG 500 – 2 AWG Channels
  - M3300A AWG 500 DIG 100 – 2 AWG Channels

The number of Channels depend on the selected module's AWG SFP window.

2. Main

- This tab contains the primary elements to generate waveforms using the internal Function Generator or the Arbitrary Waveform Generator along with the settings to configure the waveform's display characteristics.

3. Waveform shape

- This drop-down contains the primary elements to generate waveforms using the internal Function Generator or the Arbitrary Waveform Generator along with the settings to configure the waveform's display characteristics. Note that the SFP starts generating the waveforms using the Function Generator as soon as you select one from the list.
  - Related API function: "channelWaveShape()"

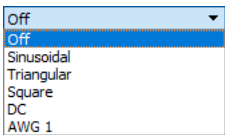


Figure 74 Options in the Waveform shape drop-down box

- **Off**—The output signal is set to '0'. All other channel settings are maintained.
- **Sinusoidal**—The output signal is generated as sinusoidal. It is generated by the Function Generator.
- **Triangular**—The output signal is generated as triangular. It is generated by the Function Generator.
- **Square**—The output signal is generated as square. It is generated by the Function Generator.

- **DC Voltage**—The output signal is generated as square. It is generated by the Amplitude Modulator.
- **AWG n** (n = AWG number for the respective Channel)—The output signal is generated based on the waveforms defined within the AWG Queue for the respective AWG. See "Setting the AWG Waveform Queue dialog" on page 90 for more information. It is generated by the Arbitrary Waveform Generator.
- Partner Channel—This is applicable for odd channels only. It requires a new panel to be added to generate the output from the previous channel (to create differential signals, and so on).
- **HiZ**—Selecting this check box indicates that the output signal is set to HIZ (High Impedance) and no output signal is provided. This feature is available for Keysight M3202A PXIe AWG models only.

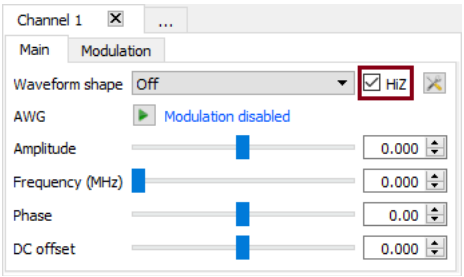


Figure 75 HiZ option available in M3202A PXIe AWG models

4. Controls for Amplitude, Frequency, Phase & DC offset

The four parameters define the waveform characteristics that is being generated.

- **Amplitude**—defines the amplitude range on the waveform generated by both the Function Generator and the Arbitrary Waveform Generator. The number of Waveform points set for each waveform type is multiplied by the value set in the user interface to determine the actual amplitude. Default unit is 'Volts' and the range is between '±1.5V' for all AWG modules.
  - Related API function: "channelAmplitude()"
- **Frequency (MHz)**—defines the frequency of transmission on the waveform generated by the Function Generator only. Default unit is 'MHz' and can be set to a maximum value of '200 MHz' for all AWG modules that have sampling rate of 500 MSa/s whereas M3202A AWG with 1GSa/s has a maximum frequency limit of '400 MHz'. Note that for the waveforms generated by the AWG, the frequency is defined within the CSV file and not controlled via the SD1 user interface.
  - Related API function: "channelFrequency()"
- **Phase**—defines the phase of the output waveform generated by the Function Generator only. Default unit is 'Degrees' and the range is between '±360' for all AWG modules. This phase is continuous and can be made coherent by using the "Reset Accumulated Phase" feature for the respective Channel number. See description for "Reset Accumulated Phase" in "Understanding AWG SFP main menu features & controls" on page 83 for more information. Note that for the waveforms generated by the AWG, the phase is defined within the CSV file and not controlled via the SD1 user interface.
  - Related API function: "channelPhase()"
- **DC offset**—defines the offset range on the waveform generated by both the Function Generator and the Arbitrary Waveform Generator. The number of Waveform points set for each waveform type is multiplied by the value set in the user interface to determine the actual offset. Default unit is 'Volts' and the range is between '±1.5V' for all AWG modules.
  - Related API function: "channelOffset()"

The controls for configuring values work in the same manner across all AWG SFP windows.

- Horizontal sliders—By default, markers in the center indicate that the respective parameter range between both negative and positive values, whereas the markers that appear on the left indicate only positive values.
- Editable value field—Click within the field where the value for each parameter is shown and edit the respective values.
- Vertical sliders—Included each value field is a vertical slider for incrementing or decrementing the values within the defined ranges.

5. Using the Channel Visualization Settings dialog

To access the **Channel Visualization Settings** dialog, click the icon on the Channel configuration panel.

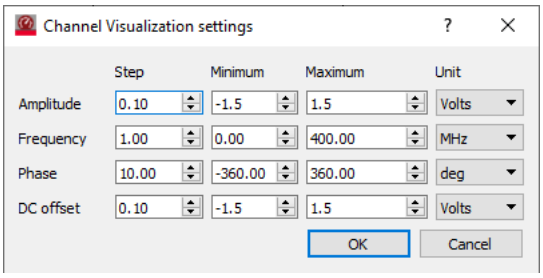


Figure 76 Default Channel Visualization Settings dialog for M3202A AWG 1G module

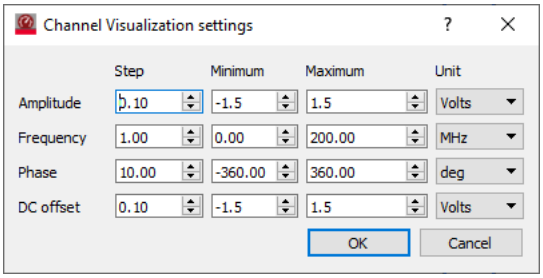


Figure 77 Default Channel Visualization Settings dialog for all other AWG modules

This dialog box provides advanced settings for each of the parameters that are displayed up front on the Channel configuration panel. Here, you can:

- Define minimum and maximum values within the defined ranges for each parameter
- Define custom step sizes used for incrementing/decrementing the values in the defined range
- Define other options as units for each parameter as shown in Table 23.
- Define values by entering in the value fields or using the vertical sliders.
- Click **OK** to save any changes and return to the AWG module window.

Table 23 Alternate units and range values for each parameter

Parameter	Alternate units	Range value	
		Minimum	Maximum
Amplitude	Volts	-1500.0	1500.0
Frequency	Hz	0.00	200000000.00 400000000.00 (for AWG 1G)
	kHz	0.00	200000.00 400000.00 (for AWG 1G)
Phase (default step size is 0.17)	rad (Radians)	-6.28	6.28
DC offset	Volts	-1500.0	1500.0

6. Modulation

On the Channel configuration panel, click the **Modulation** tab to view the settings to include one or more modulating signals in the output waveform from the Function Generator or the AWG.

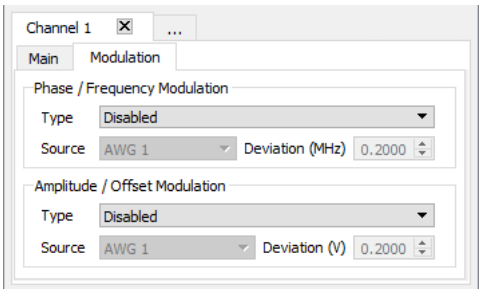


Figure 78 Default view of the Modulation tab

Figure 79 Options available in each segment of the Modulation tab

You may choose to include the following combinations of modulation signals from the drop-down options in each segment. If required, set the deviation value for the corresponding field:

- Disabled (default option, no modulation signal will be included)
- Frequency Modulation only—Related API function: "modulationAngleConfig()"
- Phase Modulation only—Related API function: "modulationAngleConfig()"
- Amplitude Modulation only—Related API function: "modulationAmplitudeConfig()"
- Offset Modulation only—Related API function: "modulationAmplitudeConfig()"
- Both Frequency and Amplitude Modulation
- Both Frequency and Offset Modulation
- Both Phase and Amplitude Modulation
- Both Phase and Offset Modulation

- IQ Modulation only (deviation is not required to be set)—Related API function: "modulationIQconfig()"

For more information regarding the Angle and Amplitude modulation techniques available here, see "Working with Signal Modulation" on page 29. Once configured, you may switch to the **Main** tab to view the applied modulation signal. See Figure 80 for an example.

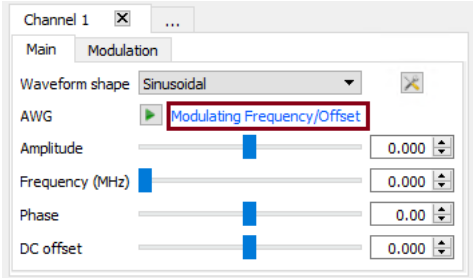


Figure 80 Single/Combination of modulation signals applied

7. AWG

As mentioned earlier, the standard waveforms using the Function Generator start generating as soon as you select a waveform from the drop-down options for **Waveform shape**. However, if you select the AWG n option, you must manually click the *Play* button, as shown in Figure 81, to start waveform generation using the AWG, as defined in the **AWG Waveform Queue** window. If modulation is applied, the modulation signal is added to the output waveform. As explained earlier, only **Amplitude** and **DC Offset** parameter values can be modified on these waveforms.

- Related API functions: "AWG()", "AWGstart()", "AWGstartMultiple()", "AWGpause()", "AWGpauseMultiple()", "AWGresume()", "AWGresumeMultiple()", "AWGstop()", "AWGstopMultiple()"

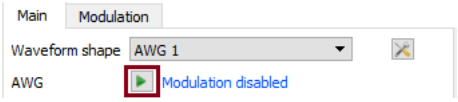


Figure 81 Play button to generate AWG waveforms only

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 24 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++	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_AOU functions
- SD_Wave 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 25 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 26 List of SD_AOU Functions

Function name	Short description
channelWaveShape	Sets the channel waveshape type.
channelFrequency	Sets the channel frequency for the periodic signals generated by the Function Generator.
channelPhase	Sets the channel phase for the periodic signals generated by the Function Generator.
channelPhaseReset	Resets the accumulated phase of the selected channel.
channelPhaseResetMultiple	Resets the accumulated phase of multiple selected channels, simultaneously.
channelAmplitude	Sets the amplitude of a channel.
channelOffset	Sets the DC offset of a channel.
modulationAngleConfig	Configures the modulation in frequency/phase for the selected channel.
modulationAmplitudeConfig	Configures the modulation in amplitude/offset for the selected channel.
modulationIQconfig	Sets the IQ modulation for the selected channel.
clockIOconfig	Configures the operation of the clock output connector.
waveformLoad	Loads a waveform into the module's onboard RAM.
waveformReLoad	Replaces a waveform located in the module's onboard RAM.
waveformFlush	Deletes all waveforms from the module's onboard RAM and flushes all AWG queues.
AWG	Provides a one-step method to load, queue, and start a single waveform.
AWGqueueWaveform	Queues a waveform in the specified AWG.
AWGflush	Empties the queue of the AWG.
AWGfreezeOnStopEnable	By default, when the queue is stopped the channel output returns to zero. If this feature is enabled, the output will maintain the last value.
AWGisFreezeOnStopEnabled	Returns the status of the AWGFreezeOnStop function.
AWGstart	Runs the AWG starting from the beginning of its queue.
AWGstartMultiple	Runs the AWG starting from the beginning of its queue, but acting on more than one AWG at once.
AWGpause	Pauses the AWG.
AWGpauseMultiple	Pauses the AWG, but acting on more than one AWG at once.
AWGresume	Resumes the AWG.
AWGresumeMultiple	Resumes the AWG, but acting on more than one AWG at once.
AWGstop	Stops the AWG, resetting the queue to its initial position.
AWGstopMultiple	Stops the AWG, but acting on more than one AWG at once.
AWGjumpNextWaveform	Forces the AWG to jump to the next waveform in the queue.
AWGjumpNextWaveformMultiple	Forces the AWG to jump to the next waveform in the queue, but acting on more than one AWG at once.
AWGisRunning	Returns if the AWG is running or stopped.
AWGnWFplaying	Returns if the AWG is running or stopped, but acting on more than one AWG at once.

Function name	Short description
AWGtriggerExternalConfig	Configures the external triggers of the AWG.
AWGtrigger	Triggers the AWG.
AWGtriggerMultiple	Triggers the AWG, but acting on more than one AWG at once.
triggerIOconfig	Configures the trigger line direction.
triggerIOwrite	Sets the trigger output.
triggerIOread	Reads the trigger input.
clockSetFrequency	Sets the module clock frequency. This function is only usable for modules with the variable clock Option CLV.
clockGetFrequency	Returns the value in Hz of the module sample rate frequency.
clockGetSyncFrequency	Returns the frequency of the internal CLKsync signal in Hz.
clockResetPhase	Set modules in sync state, waiting for first trigger to reset the phase of the internal clocks CLKsync and CLKsys.
AWGqueueConfig	Configures the cyclic mode of the queue.
AWGqueueConfigRead	Reads the value of the cyclic mode of the queue.
AWGqueueMarkerConfig	Configures the Marker generation for each AWG.
AWGqueueSyncMode	Configures the sync mode of the queue.
AWGqueueIsEmpty	Returns '1' if AWG queue is empty; '0' otherwise.
AWGqueueIsFull	Returns '1' if AWGqueue is full; '0' otherwise.
AWGqueueRemaining	Returns the size remaining on queue.
freqToInt	Converts frequency from 'double' to 'integer'.
freqGainToInt	Converts deviation gain (in frequency modulation) from 'double' to 'integer'.
phaseToInt	Converts phase from 'double' to 'integer'.
phaseGainToInt	Converts deviation gain (in phase modulation) from 'double' to 'integer'.
setDigitalFilterMode	Sets the digital filter mode.
voltsToInt	Converts voltage from 'double' to 'integer'.
waveformAddToList	Adds the waveform to the list.
waveformListLoad	Loads the list of waveform for the module.

Table 27 List of SD_Wave Functions

Function name	Short description
new	Creates a waveform object from data points contained in an array in memory or in a file.
delete	Removes a waveform created with the new function.
getStatus	Returns the waveform loading status.
getType	Returns the type of loaded waveform.

Table 28 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.
Module HVI Engine	Enables HVI Engines located in the M3xxxA module's FPGA.
Module HVI Triggers	Activates PXI or Front Panel triggers.

Table 29 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 tables display certain parameters and their values, required for one or more API functions.

Table 30 Channel Waveshape types and corresponding values

Waveshape	Description	Name	Value
HIZ	The output signal is set to HIZ (No output signal is provided.) *	AOU_HIZ	-1
No Signal	The output signal is set to 0. All other channel settings are maintained.	AOU_OFF (default)	0
Sinusoidal	Generated by the Function Generator	AOU_SINUSOIDAL	1
Triangular	Generated by the Function Generator	AOU_TRIANGULAR	2
Square	Generated by the Function Generator	AOU_SQUARE	4
DC Voltage	Generated by the Amplitude Modulator	AOU_DC	5
ArbitraryWaveform	Generated by the Arbitrary Waveform Generator. See AWG Waveform Types.	AOU_AWG	6
PartnerChannel	Only for odd channels. It is the output of the previous channel (to create differential signals, etc.)	AOU_PARTNER	8

*Only available for Keysight M3202A PXIe AWG model

Table 31 Angle modulation options and corresponding values

Options	Description	Name	Value
NoModulation	Modulation is disabled.	AOU_MOD_OFF (default)	0
FrequencyModulation	The AWG is used to modulate the channel frequency.	AOU_MOD_FM	1
FrequencyModulation (32 bits)	The AWG is used to modulate the channel frequency with 32 bit resolution	AOU_MOD_FM_32b	1
PhaseModulation	The AWG is used to modulate the channel phase.	AOU_MOD_PM	2

Table 32 Amplitude modulation options and corresponding values

Options	Description	Name	Value
NoModulation	Modulation is disabled. The channel amplitude and offset are only set by the main registers.	AOU_MOD_OFF (default)	0
AmplitudeModulation	The modulating signal is used to modulate the channel amplitude.	AOU_MOD_AM	1
OffsetModulation	The modulating signal is used to modulate the channel offset.	AOU_MOD_OFFSET	2

Table 33 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 34 Waveform types and corresponding values

Waveshape	Description	Name	Value
Analog 16 Bits	Analog normalized waveforms (-1 to 1) defined with doubles	WAVE_ ANALOG_16	0
Analog 32 Bits	Analog normalized waveforms (-1 to 1) defined with doubles	WAVE_ANALOG_32	1
Analog 16 Bits Dual	Analog normalized waveforms (-1 to 1) defined with doubles, with two components (A and B)	WAVE_ANALOG_16_DUAL	4
Analog 32 Bits Dual	Analog normalized waveforms (-1 to 1) defined with doubles, with two components (A and B)	WAVE_ANALOG_32_DUAL	6
IQ*	Analog normalized waveforms (-1 to 1) defined with doubles, with two components (I and Q)	WAVE_IQ	2
IQ Polar*	Analog waveforms (-1 to 1 module, -180 to +180 phase) defined with doubles, with two components (Magnitude and Phase)	WAVE_IQPOLAR	3
Digital	Digital waveforms defined with integers	WAVE_DIGITAL	5

*When using IQ or IQ Polar, each component will only play at a maximum rate of 500 MSa/s.

NOTE The M3201A PXIe AWG and both the M3300A & M3302A PXIe Combos do not support both Analog 32 Bits (Single) and Analog 32 Bits Dual waveforms.

Table 35 Trigger mode options and corresponding values

Waveshape	Description	Name	Value
Immediate (Auto)	The waveform is launched immediately after AWGstart, or when the previous waveform in the queue finishes.	AUTOTRIG	0
Software /HVI	Software trigger. The AWG is triggered by AWGtrigger provided that the AWG is running. AWGtrigger can be executed from the user application (VI) or from an HVI.	SWHVITRIG	1
Software / HVI (per cycle)	Software trigger. Identical to the previous option, but a trigger is required per each waveform cycle.	SWHVITRIG_CYCLE	5
External Trigger	Hardware trigger. The AWG waits for an external trigger source for the AWGs. (This is set by the "externalSource" in AWGtriggerExternalConfig.)	EXTTRIG	2
External Trigger (per cycle)	Hardware trigger. Identical to the previous option, but a trigger is required per each waveform cycle.	EXTTRIG_CYCLE	6

Table 36 External Trigger options and corresponding values

Options	Description	Name	Value
External I/OTrigger	The AWG trigger is a TRG connector/line of the module. PXI form factor only: this trigger can be synchronized to CLK10.	TRIG_EXTERNAL	0
PXITrigger [0 to n]	PXI form factor only. The AWG external trigger is a PXI trigger line and is synchronized to CLK10. For example, to use PXI trigger line 1on the PXI backplane, use '`TRIGGER_PXI1`' or '`4001`'.	TRIG_PXI + Trigger No.	4000 + Trigger No.

Table 37 External Trigger behavior and corresponding values

Behavior modes	Description	Name	Value
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 38 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.	AOU_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.	AOU_TRG_IN	1

Table 39 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 40 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 41 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 42 FPGATriggerDirection attribute values

Attribute	Value
IN	0
INOUT	1

Table 43 TriggerPolarity attribute values

Attribute	Value
ACTIVE_LOW	0
ACTIVE_HIGH	1

5.3.2: Latency in AWGs for various HVI Actions & Instructions

Table 44 displays the HVI actions and instructions along with the corresponding latency measured for AWGs. For AWGs, the latency value for each action and instruction is the measured delay between the HVI instructions/actions and the corresponding change in the analog output. A PXI or Front Panel trigger is used as reference and all instruction latencies are measured with respect to this trigger. For example, for the “AWG_Trigger” instruction, the time between the trigger and the AWG output in the Oscilloscope is measured.

Table 44 Latency (in nanoseconds) for HVI Actions & Instructions

HVI Actions / Instructions	Latency in M3201A (AWG 500)	Latency in M3202A (AWG 1G)	Comments
AWG_Pause	204	110	-
AWG_QueueFlush	204	119	-
AWG_Resume	207	111	-
AWG_Start	227	122	-
AWG_Trigger	205	112	-
AWG_JumpNextWaveform	1990	1828	-
AWG _Stop	204	120	-
ModulationAmplitudeConfig	201	106	Note 1
SetAmplitude	203	115	Note 1
SetFrequency	304	172	Note 1
SetOffset	204	115	Note 1
SetPhase	296	155	Note 1
SetWaveshape	242	132	Note 1
ModulationPhaseConfig	345	204	Note 1
AWGQueueWaveform	1590	1550	Note 1, 2

Note:

1. All these HVI instructions can be run simultaneously.

2. AWGQueueWavefirm is a multi-step instruction, which takes 20ns to run. Rest of the HVI instructions shown in the table take 10ns.

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). Table 45 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 45 Minimum delay required (in nanoseconds) for FPGA User Sandbox instructions

Instructions	Minimum Delay required	Extra latency in AWG 1G*
fpga_array_read	60	+15ns
fpga_array_read (Address from HviRegister)	80	+15ns
fpga_array_write	30	+10ns
fpga_array_write (Address or data from HviRegister)	50	+10ns
fpga_register_read	60	+15ns
fpga_register_write	30	+10ns
fpga_register_write (Address or data from HviRegister)	50	+10ns

* The M3202A (AWG 1G) instrument has an extra latency, which is in addition to that required for FPGA User Sandbox instructions.

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 46 Input parameters for open

Input Parameter name	Description
productName	Module's product name (for example, "M3202A" or "M3102A"). 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 "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 47 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 48 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 49 Input parameters for close

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

Table 50 Output parameters for close

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

Syntax

Table 51 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 52 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 53 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 54 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 55 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 56 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 57 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 58 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 59 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 60 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 61 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 62 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 63 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 64 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 65 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 66 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 67 Output parameters for PXItriggerWrite

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

Syntax

Table 68 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 69 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 70 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 71 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 72 Input parameters for getFirmwareVersion

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

Table 73 Output parameters for getFirmwareVersion

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

Syntax

Table 74 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 75 Input parameters for getHardwareVersion

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

Table 76 Output parameters for getHardwareVersion

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

Syntax

Table 77 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 78 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 79 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 80 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 81 Input parameters for getTemperature

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

Table 82 Output parameters for getTemperature

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

Syntax

Table 83 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 84 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 85 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 86 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 87 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 88 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 89 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 90 Output parameters for translateTriggerIOtoExternalTriggerLine

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

Syntax

Table 91 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 92 Input parameters for translateTriggerPXItoExternalTriggerLine

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

Table 93 Output parameters for translateTriggerPXItoExternalTriggerLine

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

Syntax

Table 94 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 95** **Output parameters for runSelfTest**

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

Syntax **Table 96** **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_AOU functions

5.5.1: channelWaveShape

Description Sets the waveshape type for the selected channel.

Parameters **Table 97 Input parameters for channelWaveShape**

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
nChannel	Channel number.
waveShape	Channel waveshape type. Refer to Table 30 Channel Waveshape types and corresponding values.

Table 98 Output parameters for channelWaveShape

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

Syntax **Table 99 API syntax for channelWaveShape**

Language	Syntax
C	`int SD_AOU_channelWaveShape(int moduleID, int nChannel, int waveShape);`
C++	`int SD_AOU::channelWaveShape(int nChannel, int waveShape);`
.NET	`int SD_AOU::channelWaveShape(int nChannel, int waveShape);`
Python	`SD_AOU.channelWaveShape(nChannel, waveShape)`
HVI	Refer to Table 100 HVI syntax and corresponding values for channelWaveShape.

Table 100 HVI syntax and corresponding values for channelWaveShape

Language	Syntax	Parameter	Possible values
Python	`module.hvi.instruction_set.set_waveshape.id`	`channel.id`	• `Channel number`
		`value.id`	• `AOU_OFF` • `AOU_SINUSOIDAL` • `AOU_TRIANGULAR` • `AOU_SQUARE` • `AOU_DC` • `AOU_AWG` • `AOU_PARTNER_CHANNEL`
.NET	`module.Hvi.InstructionSet.SetWaveShape.Id`	`Channel.Id`	• `Channel number`
		`Value.Id`	• `AOU_OFF` • `AOU_SINUSOIDAL` • `AOU_TRIANGULAR` • `AOU_SQUARE` • `AOU_DC` • `AOU_AWG` • `AOU_PARTNER_CHANNEL`

5.5.2: channelFrequency

Description Sets the frequency for the periodic signals generated by the Function Generator block on the selected channel.

Parameters

Table 101 Input parameters for channelFrequency

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
nChannel	Channel number.
frequency	Frequency in Hz. (Refer to the product's Data Sheet for frequency specifications.)

Table 102 Output parameters for channelFrequency

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

Syntax

Table 103 API syntax for channelFrequency

Language	Syntax
C	`double SD_AOU_channelFrequency(int moduleID, int nChannel, double frequency);`
C++	`double SD_AOU::channelFrequency(int nChannel, double frequency);`
.NET	`double SD_AOU::channelFrequency(int nChannel, double frequency);`
Python	`SD_AOU.channelFrequency(nChannel, frequency)`
HVI	Refer to Table 104 HVI syntax and corresponding values for channelFrequency.

Table 104 HVI syntax and corresponding values for channelFrequency

Language	Syntax	Parameter	Possible values
Python	`module.hvi.instruction_set.set_frequency.id`	`channel.id`	• `Channel number`
		`value.id`	• `Frequency value in Hertz`
.NET	`module.Hvi.InstructionSet.SetFrequency.Id`	`Channel.Id`	• `Channel number`
		`Value.Id`	• `Frequency value in Hertz`

NOTE Use the freqToInt() API function to convert frequency from double to integer.

5.5.3: channelPhase

Description Sets the phase, on the selected channel, for the periodic signals generated by the Function Generator block.

Parameters

Table 105 Input parameters for channelPhase

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
nChannel	Channel number.
phase	Phase in degrees. (Refer to the product's Data Sheet for phase specifications.)

Table 106 Output parameters for channelPhase

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

Syntax

Table 107 API syntax for channelPhase

Language	Syntax
C	`int SD_AOU_channelPhase(int moduleID, int nChannel, double phase);`
C++	`int SD_AOU::channelPhase(int nChannel, double phase);`
.NET	`int SD_AOU::channelPhase(int nChannel, double phase);`
Python	`SD_AOU.channelPhase(nChannel, phase)`
HVI	Refer to Table 104 HVI syntax and corresponding values for channelFrequency.

Table 108 HVI syntax and corresponding values for channelPhase

Language	Syntax	Parameter	Possible values
Python	`module.hvi.instruction_set.set_phase.id`	`channel.id`	• `Channel number`
		`value.id`	• `Phase value in degrees`
.NET	`module.Hvi.InstructionSet.SetPhase.Id`	`Channel.Id`	• `Channel number`
		`Value.Id`	• `Phase value in degrees`

NOTE Use the phaseToInt() API function to convert phase from double to integer.

5.5.4: channelPhaseReset

Description

Resets the accumulated phase on the selected channel. The accumulated phase is the result of the phase continuous operation of the Function Generator block.

Parameters

Table 109 Input parameters for channelPhaseReset

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

Table 110 Output parameters for channelPhaseReset

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

Syntax

Table 111 API syntax for channelPhaseReset

Language	Syntax
C	`int SD_AOU_channelPhaseReset(int moduleID, int nChannel);`
C++	`int SD_AOU::channelPhaseReset(int nChannel);`
.NET	`int SD_AOU::channelPhaseReset(int nChannel);`
Python	`SD_AOU.channelPhaseReset(nChannel)`
HVI	Refer to Table 112 HVI syntax and corresponding values for channelPhaseReset.

Table 112 HVI syntax and corresponding values for channelPhaseReset

Language	Syntax	Parameters	Possible values
Python	`module.hvi.actions.ch<x>_reset_phase`	Not available	Not available
.NET	`module.Hvi.Actions.Ch<x>ResetPhase`	Not available	Not available

where, x = channel number

5.5.5: channelPhaseResetMultiple

Description

Resets the accumulated phase on multiple selected channels, simultaneously. This accumulated phase is the result of the phase continuous operation of each channel's Function Generator block.

Parameters

Table 113 Input parameters for channelPhaseResetMultiple

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
channelMask	Mask to select the channels to reset (LSB is channel 1, bit 1 is channel 2, and so forth).

Table 114 Output parameters for channelPhaseResetMultiple

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

Syntax

Table 115 API syntax for channelPhaseResetMultiple

Language	Syntax
C	`int SD_AOU_channelPhaseResetMultiple(int moduleID, int channelMask);`
C++	`int SD_AOU::channelPhaseResetMultiple(int channelMask);`
.NET	`int SD_AOU::channelPhaseResetMultiple(int channelMask);`
Python	`SD_AOU.channelPhaseResetMultiple(channelMask)`
HVI	Not available

5.5.6: channelAmplitude

Description Sets the amplitude of a channel. See "Channel Amplitude and DC Offset" on page 18.

Parameters

Table 116 Input parameters for channelAmplitude

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
nChannel	Channel number.
amplitude	Amplitude in volts (–1.5 V to 1.5 V)

Table 117 Output parameters for channelAmplitude

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

Syntax

Table 118 API syntax for channelAmplitude

Language	Syntax
C	`int SD_AOU_channelAmplitude(int moduleID, int nChannel, double amplitude);`
C++	`int SD_AOU::channelAmplitude(int nChannel, double amplitude);`
.NET	`int SD_AOU::channelAmplitude(int nChannel, double amplitude);`
Python	`SD_AOU.channelAmplitude(nChannel, amplitude)`
HVI	Refer to Table 119 HVI syntax and corresponding values for channelAmplitude.

Table 119 HVI syntax and corresponding values for channelAmplitude

Language	Syntax	Parameter	Possible values
Python	`module.hvi.instruction_set.set_amplitude.id`	`channel.id`	• `Channel number`
		`value.id`	• `Amplitude value in volts, decimal values from -1.5 to 1.5.`
.NET	`module.Hvi.InstructionSet.SetAmplitude.Id`	`Channel.Id`	• `Channel number`
		`Value.Id`	• `Amplitude value in volts, decimal values from -1.5 to 1.5.`

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

5.5.7: channelOffset

Description Sets the DC offset of a channel. See "Channel Amplitude and DC Offset" on page 18.

Parameters

Table 120 Input parameters for channelOffset

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
nChannel	Channel number.
offset	DC offset in volts (–1.5 V to 1.5 V)

Table 121 Output parameters for channelOffset

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

Syntax

Table 122 API syntax for channelOffset

Language	Syntax
C	`int SD_AOU_channelOffset(int moduleID, int nChannel, double offset);`
C++	`int SD_AOU::channelOffset(int nChannel, double offset);`
.NET	`int SD_AOU::channelOffset(int nChannel, double offset);`
Python	`SD_AOU.channelOffset(nChannel, offset)`
HVI	Refer to Table 123 HVI syntax and corresponding values for channelOffset.

Table 123 HVI syntax and corresponding values for channelOffset

Language	Syntax	Parameter	Possible values
Python	`module.hvi.instruction_set.set_offset.id`	`channel.id`	• `Channel number`
		`value.id`	• `Amplitude offset value in volts, decimal values from -1.5 to 1.5`
.NET	`module.Hvi.InstructionSet.SetOffset.Id`	`Channel.Id`	• `Channel number`
		`Value.Id`	• `Amplitude offset value in volts, decimal values from -1.5 to 1.5`

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

5.5.8: modulationAngleConfig

Description

Configures the modulation in frequency or phase for the selected channel. See "Channel Frequency and Phase" on page 17.

Parameters

Table 124 Input parameters for modulationAngleConfig

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
nChannel	Channel number.
modulationType	Angle modulation options. Refer to Table 31 Angle modulation options and corresponding values.
deviationGain	Gain for the normalized modulating signal.

Table 125 Output parameters for modulationAngleConfig

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

Syntax

Table 126 API syntax for modulationAngleConfig

Language	Syntax
C	`int SD_AOU_modulationAngleConfig(int moduleID, int nChannel, int modulationType, double deviationGain);`
C++	`int SD_AOU::modulationAngleConfig(int nChannel, int modulationType, double deviationGain);`
.NET	`int SD_AOU::modulationAngleConfig(int nChannel, int modulationType, double deviationGain);`
Python	`SD_AOU.modulationAngleConfig(nChannel, modulationType, deviationGain)`
HVI	Refer to Table 127 HVI syntax and corresponding values for modulationAngleConfig.

Table 127 HVI syntax and corresponding values for modulationAngleConfig

Language	Syntax	Parameter	Possible values
Python	module.hvi.instruction_set.modulation_angle_config.id	channel.id	• Channel number
		modulation_type.id	• AOU_MOD_OFF • AOU_MOD_PM • AOU_MOD_FM
		deviation_gain.id	• Deviation gain for the normalized modulating signal
.NET	module.Hvi.InstructionSet.ModulationAngleConfig.Id	Channel.Id	• Channel number
		ModulationType.Id	• AOU_MOD_OFF • AOU_MOD_PM • AOU_MOD_FM
		DeviationGain.Id	• Deviation gain for the normalized modulating signal

NOTE Use the freqGainToInt() and the phaseGainToInt() API functions to convert deviation gain (in frequency modulation) and deviation gain (in phase modulation), respectively, from double to integer.

5.5.9: modulationAmplitudeConfig

Description Configures the modulation in amplitude or offset for the selected channel. See "Channel Amplitude and DC Offset" on page 18.

Parameters

Table 128 Input parameters for modulationAmplitudeConfig

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
nChannel	Channel number.
modulationType	Amplitude modulation options. Refer to Table 32 Amplitude modulation options and corresponding values.
deviationGain	Gain for the normalized modulating signal.

Table 129 Output parameters for modulationAmplitudeConfig

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

Syntax

Table 130 API syntax for modulationAmplitudeConfig

Language	Syntax
C	`int SD_AOU_modulationAmplitudeConfig(int moduleID, int nChannel, int modulationType, double deviationGain);`
C++	`int SD_AOU::modulationAmplitudeConfig(int nChannel, int modulationType, double deviationGain);`
.NET	`int SD_AOU::modulationAmplitudeConfig(int nChannel, int modulationType, double deviationGain);`
Python	`SD_AOU.modulationAmplitudeConfig(nChannel, modulationType, deviationGain)`
HVI	Refer to Table 131 HVI syntax and corresponding values for modulationAmplitudeConfig.

Table 131 HVI syntax and corresponding values for modulationAmplitudeConfig

Language	Syntax	Parameter	Possible values
Python	`module.hvi.instruction_set.modulation_amplitude_config.id`	`channel.id`	• `Channel number`
		`modulation_type.id`	• `AOU_MOD_OFF` • `AOU_MOD_AM` • `AOU_MOD_OFFSET`
		`deviation_gain.id`	• `Deviation gain for the normalized modulating signal`
.NET	`module.Hvi.InstructionSet.ModulationAmplitudeConfig.Id`	`Channel.Id`	• `Channel number`
		`ModulationType.Id`	• `AOU_MOD_OFF` • `AOU_MOD_AM` • `AOU_MOD_OFFSET`
		`DeviationGain.Id`	• `Deviation gain for the normalized modulating signal`

NOTE Use the voltsToInt() API function to convert deviation gain (either in amplitude or offset) from double to integer.

5.5.10: modulationIQconfig

Description

Configures the IQ modulation for the selected channel. See "IQ Modulation (Quadrature Modulator Block)" on page 33.

Parameters

Table 132 Input parameters for modulationIQConfig

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
nChannel	Channel number.
enable	Enable (value 1) or Disable (value 0) the IQ modulation.

Table 133 Output parameters for modulationIQConfig

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

Syntax

Table 134 API syntax for modulationIQConfig

Language	Syntax
C	`int SD_AOU_modulationIQconfig(int moduleID, int nChannel, int enable);`
C++	`int SD_AOU::modulationIQconfig(int nChannel, int enable);`
.NET	`int SD_AOU::modulationIQconfig(int nChannel, int enable);`
Python	`SD_AOU.modulationIQconfig(nChannel, enable)`
HVI	Not available

5.5.11: clockIOconfig

Description Configures the operation of the clock output connector. See “CLK Output Options” on page 36.

Parameters

Table 135 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 33 Clock connector options and corresponding values.
port	Applicable for M33xxA COMBO cards only.

Table 136 Output parameters for clockIOconfig

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

Syntax

Table 137 API syntax for clockIOconfig (on M32xxA AWG modules)

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

Table 138 API syntax for clockIOconfig (on M33xxA 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

5.5.12: waveformLoad

Description

Loads the specified waveform into the module's onboard RAM. Waveforms must first be created with new. Waveforms are loaded at different speeds depending on their length, longer waveforms will load at a faster rate. (For example, 1000 point waveforms will load at 2 MB/s, waveforms with 10,000 points will load at 20 MB/s, and waveforms with 1,000,000 points will load at 44 MB/s). Regardless of waveform size, a maximum of 1024 waveforms can be loaded into the module's onboard RAM.

Parameters

Table 139 Input parameters for waveformLoad

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
waveformID	Waveform identifier. See SD_Wave functions.
waveformObject	Pointer to the waveform object. See SD_Wave functions.
waveformType	Waveform type used in the waveform file and in this function, it defines the type of waveform created. This parameter is used to internally configure the AWG. Refer to Table 34 Waveform types and corresponding values. Also, see AWG Waveform Types.
waveformPoints	Number of points of the waveform, which must be a multiple of a certain number of points. See *AWG specifications* in the Data Sheet.
waveformDataRaw	Array with waveform points. In dual and IQ waveforms, the waveform points are interleaved (WaveformA0, WaveformB0, WaveformA1, etc.)
waveformNumber	Waveform number to identify the waveform in subsequent related function calls. This value must be in the (0 to n) range, and in order to optimize the memory usage, it should be as low as possible.
paddingMode	If 0, the waveform is loaded as is and the zeros are added at the end if the number of points is not a multiple of the number required by the AWG. If 1, the waveform is loaded n times (using DMA) until the total number of points is a multiple of the number of points required by the AWG (only for waveforms with an even number of points).

Table 140 Output parameters for waveformLoad

Output Parameter name	Description
availableRAM	Available onboard RAM in waveform points. Negative numbers indicate an error. See Description of SD1 Error IDs.
errorOut	See Description of SD1 Error IDs.

Syntax

Table 141 API syntax for waveformLoad

Language	Syntax
C	`int SD_AOU_waveformLoad(int moduleID, int waveformID, int waveformNumber, int paddingMode);` `int SD_AOU_waveformLoadArrayInt16(int moduleID, int waveformType, int waveformPoints, short *waveformDataRaw, int waveformNumber, int paddingMode);`

C++	`int SD_AOU::waveformLoad(int waveformType, std::vector<short> dataRaw, int waveformNumber, int paddingMode);`
	`int SD_AOU::waveformLoad(int waveformType, std::vector<short> dataRaw, int waveformNumber);`
	`int SD_AOU::waveformLoad(SD_Wave waveformObject, int waveformNumber, int paddingMode);`
	`int SD_AOU::waveformLoad(SD_Wave waveformObject, int waveformNumber);`

.NET	`int SD_AOU::waveformLoad(SD_Wave waveformObject, int waveformNumber, int paddingMode);`
	`int SD_AOU::waveformLoad(int waveformType, short[] waveformDataRaw, int waveformNumber, int paddingMode);`
	`int SD_AOU::waveformLoad(SD_Wave waveformObject, int waveformNumber);`
	`int SD_AOU::waveformLoad(int waveformType, short[] waveformDataRaw, int waveformNumber);`

Python	`SD_AOU.waveformLoad(waveformObject, waveformNumber, paddingMode)` `SD_AOU.waveformLoadInt16(waveformType, waveformDataRaw, waveformNumber, paddingMode)`
HVI	Not available

5.5.13: waveformReLoad

Description Replaces an existing waveform located in a module's onboard RAM. The size of the new waveform must be smaller than or equal to the existing waveform.

Parameters

Table 142 Input parameters for waveformReLoad

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
waveformID	Waveform identifier. See SD_Wave functions.
waveformObject	Pointer to the waveform object. See SD_Wave functions.
waveformType	Waveform type used in the waveform file and in this function, it defines the type of waveform created. This parameter is used to internally configure the AWG. Refer to Table 34 Waveform types and corresponding values. Also, see AWG Waveform Types
waveformPoints	Number of points of the waveform, which must be a multiple of a certain number of points. See *AWG specifications* in the Data Sheet.
waveformDataRaw	Array with waveform points. In dual and IQ waveforms, the waveform points are interleaved (WaveformA0, WaveformB0, WaveformA1, etc.).
waveformNumber	Waveform number to identify the waveform in subsequent related function calls. This value must be in the (0 to n) range, and in order to optimize the memory usage, it should be as low as possible.
paddingMode	If 0, the waveform is loaded as is and the zeros are added at the end if the number of points is not a multiple of the number required by the AWG. If 1, the waveform is loaded n times (using DMA) until the total number of points is a multiple of the number of points required by the AWG (only for waveforms with an even number of points).

Table 143 Output parameters for waveformReLoad

Output Parameter name	Description
availableRAM	Available onboard RAM in waveform points. Negative numbers indicate an error. See Description of SD1 Error IDs.
errorOut	See Description of SD1 Error IDs.

Syntax

Table 144 API syntax for waveformReLoad

Language	Syntax
C	`int SD_AOU_waveformReLoad(int moduleID, int waveformID, int waveformNumber, int paddingMode);` `int SD_AOU_waveformReLoadArrayInt16(int moduleID, int waveformType, int waveformPoints, short *waveformDataRaw, int waveformNumber, int paddingMode);`

C++	`int SD_AOU::waveformReLoad(int waveformType, std::vector<short> waveformDataRaw,int waveformNumber, int paddingMode);`
	`int SD_AOU::waveformReLoad(int waveformType, std::vector<short> waveformDataRaw, int waveformNumber);`
	`int SD_AOU::waveformReLoad(SD_Wave waveformObject, int waveformNumber, int paddingMode);`
	`int SD_AOU::waveformReLoad(SD_Wave waveformObject, int waveformNumber);`

.NET	`int SD_AOU::waveformReLoad(SD_Wave waveformObject, int waveformNumber, int paddingMode);`
	`int SD_AOU::waveformReLoad(int waveformType, short[] waveformDataRaw, int waveformNumber, int paddingMode);`
	`int SD_AOU::waveformReLoad(int waveformType, short[] dataRaw, int waveformNumber);`
	`int SD_AOU::waveformReLoad(SD_Wave waveformObject, int waveformNumber);`

Python	`SD_AOU.waveformReLoad(waveformObject, waveformNumber, paddingMode)` `SD_AOU.waveformReLoadArrayInt16(waveformType, waveformDataRaw, waveformNumber, paddingMode)`
HVI	Not available

5.5.14: waveformFlush

Description

Deletes all waveforms from the module's onboard RAM and flushes all the AWG queues. Also, see AWGflush().

Parameters

Table 145 Input parameters for waveformFlush

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

Table 146 Output parameters for waveformFlush

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

Syntax

Table 147 API syntax for waveformFlush

Language	Syntax
C	`int SD_AOU_waveformFlush(int moduleID);`
C++	`int SD_AOU::waveformFlush();`
.NET	`int SD_AOU::waveformFlush();`
Python	`SD_AOU.waveformFlush()`
HVI	Not available

5.5.15: AWG

Description Provides a one-step method to load, queue, and start a single waveform in one of the module's AWGs. The waveform can be loaded from an array of points in memory or from a file.

Parameters

Table 148 Input parameters for AWG

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
nAWG	AWG Channel number.
triggerMode	Trigger method to launch the waveforms queued in an AWG. Refer to Table 35 Trigger mode options and corresponding values.
startDelay	Defines the delay between the trigger and the waveform launch. Enter the value in multiples of 10 and the delay is calculated in units of seconds. For M3202A, you can attain a startDelay parameter value of up to 327.67 us, which is calculated as [(2^{16}) - 1] cycles] * 5ns = 65535 cycles * 5ns = 327.67 us. Similarly, for M3201A, you can attain a startDelay parameter value of up to 655.35 us, which is calculated as [(2^{16}) - 1] cycles] * 10ns = 65535 cycles * 10ns = 655.35 us. Also, note that this delay would be applied only in the first cycle of the queued WF.
cycles	Number of times the waveform is played once launched. (Zero specifies infinite cycles).
prescaler	Waveform prescaler value, to reduce the effective sampling rate by prescaler x 5.
waveformType	Waveform type used in the waveform file and in this function, it defines the type of waveform created. This parameter is used to internally configure the AWG. Refer to Table 34 Waveform types and corresponding values. Also, see AWG Waveform Types.
waveformPoints	Number of points of the waveform, which must be a multiple of a certain number of points. See *AWG specifications* in the Data Sheet.
waveformDataA	Array with waveform points. Analog waveforms are defined with floating point numbers, which correspond to a normalized amplitude (-1 to 1).
waveformDataB	Array with waveform points, only the waveforms which have a second component (for example, Q in IQ modulations defined in Cartesian, or phase in IQ modulations defined with polar).
waveformFile	File containing the waveform points.
paddingMode	If 0, the waveform is loaded as is and the zeros are added at the end if the number of points is not a multiple of the number required by the AWG. If 1, the waveform is loaded n times (using DMA) until the total number of points is a multiple of the number of points required by the AWG (only for waveforms with an even number of points).

Table 149 Output parameters for AWG

Output Parameter name	Description
availableRAM	Available onboard RAM in waveform points. Negative numbers indicate an error. See Description of SD1 Error IDs.
errorOut	See Description of SD1 Error IDs.

Syntax

Table 150 API syntax for AWG

Language	Syntax

C	int SD_AOU_AWGfromFile(int moduleID, int nAWG, const char *waveformFile, int triggerMode, int startDelay, int cycles, int prescaler, int paddingMode =0);
	int SD_AOU_AWGfromArray(int moduleID, int nAWG, int triggerMode, int startDelay, int cycles, int prescaler, int waveformType, int waveformPoints, double* waveformDataA, double* waveformDataB =0, int paddingMode =0);
	int SD_AOU_AWGfromArrayInteger(int moduleID, int nAWG, int triggerMode, int startDelay, int cycles, int prescaler, int waveformType, int waveformPoints, int* waveformDataA, int* waveformDataB, int paddingMode);

C++	int SD_AOU::AWG(int nAWG, std::string waveformFile, int triggerMode, int startDelay, int cycles, int prescaler, int paddingMode);
	int SD_AOU::AWG(int nAWG, std::string waveformFile, int triggerMode, int startDelay, int cycles, int prescaler);
	
	int SD_AOU::AWG(int nAWG, int triggerMode, int startDelay, int cycles, int prescaler, int waveformType, std::vector<double> waveformDataA, std::vector<double> waveformDataB, int paddingMode);
	int SD_AOU::AWG(int nAWG, int triggerMode, int startDelay, int cycles, int prescaler, int waveformType, std::vector<double> waveformDataA, std::vector<double> waveformDataB);
	int SD_AOU::AWG(int nAWG, int triggerMode, int startDelay, int cycles, int prescaler, int waveformType, std::vector<double> waveformDataA, int paddingMode);
	int SD_AOU::AWG(int nAWG, int triggerMode, int startDelay, int cycles, int prescaler, int waveformType, std::vector<double> waveformDataA);
	
	int SD_AOU::AWG(int nAWG, int triggerMode, int startDelay, int cycles, int prescaler, int waveformType, std::vector<int> waveformDataA, std::vector<int> waveformDataB, int paddingMode);
	int SD_AOU::AWG(int nAWG, int triggerMode, int startDelay, int cycles, int prescaler, int waveformType, std::vector<int> waveformDataA, std::vector<int> waveformDataB);
	int SD_AOU::AWG(int nAWG, int triggerMode, int startDelay, int cycles, int prescaler, int waveformType, std::vector<int> waveformDataA, int paddingMode);
	int SD_AOU::AWG(int nAWG, int triggerMode, int startDelay, int cycles, int prescaler, int waveformType, std::vector<int> waveformDataA);

Language	Syntax

.NET	int SD_AOU::AWG(int nAWG, string waveformFile, int triggerMode, int startDelay, int cycles, int prescaler);
	int SD_AOU::AWG(int nAWG, string waveformFile, int triggerMode, int startDelay, int cycles, int prescaler, int paddingMode);
	
	int SD_AOU::AWG(int nAWG, int triggerMode, int startDelay, int cycles, int prescaler, int waveformType, double[] waveformDataA, double[] waveformDataB, int paddingMode);
	int SD_AOU::AWG(int nAWG, int triggerMode, int startDelay, int cycles, int prescaler, int waveformType, double[] waveformDataA, double[] waveformDataB);
	int SD_AOU::AWG(int nAWG, int triggerMode, int startDelay, int cycles, int prescaler, int waveformType, double[] waveformDataA, int paddingMode);
	int SD_AOU::AWG(int nAWG, int triggerMode, int startDelay, int cycles, int prescaler, int waveformType, double[] waveformDataA);
	
	int SD_AOU::AWG(int nAWG, int triggerMode, int startDelay, int cycles, int prescaler, int waveformType, int[] waveformDataA, int[] waveformDataB, int paddingMode);
	int SD_AOU::AWG(int nAWG, int triggerMode, int startDelay, int cycles, int prescaler, int waveformType, int[] waveformDataA, int[] waveformDataB);
	int SD_AOU::AWG(int nAWG, int triggerMode, int startDelay, int cycles, int prescaler, int waveformType, int[] waveformDataA, int paddingMode);
	int SD_AOU::AWG(int nAWG, int triggerMode, int startDelay, int cycles, int prescaler, int waveformType, int[] waveformDataA);

Python	SD_AOU.AWGfromArray(nAWG, triggerMode, startDelay, cycles, prescaler, waveformType, waveformDataA, waveformDataB = None, paddingMode = 0) SD_AOU.AWGFromFile(nAWG, waveformFile, triggerMode, startDelay, cycles, prescaler, paddingMode = 0)
HVI	Not available

5.5.16: AWGqueueWaveform

Description

Queues the specified waveform in one of the module's AWGs. The waveform must be already loaded in the module's onboard RAM. See waveformLoad(). The number of waveforms that can be queued (regardless of cycles) is limited to one million. The AWGqueueWaveform() function can only be called one million times until you play and consequently, consume the queued waveforms.

Parameters

Table 151 Input parameters for AWGqueueWaveform

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
nAWG	AWG Channel number.
waveformNumber	Waveform to be queued into the AWG. It must be already loaded using waveformLoad.
triggerMode	Trigger method to launch the waveforms queued in an AWG. Refer to Table 35 Trigger mode options and corresponding values.
startDelay	Defines the delay between the trigger and the waveform launch. Enter the value in multiples of 10 and the delay is calculated in units of seconds. For M3202A, you can attain a startDelay parameter value of up to 327.67 us, which is calculated as [(2^{16}) - 1] cycles] * 5ns = 65535 cycles * 5ns = 327.67 us. Similarly, for M3201A, you can attain a startDelay parameter value of up to 655.35 us, which is calculated as [(2^{16}) - 1] cycles] * 10ns = 65535 cycles * 10ns = 655.35 us. Also, note that this delay would be applied only in the first cycle of the queued WF.
cycles	Number of times the waveform is played once launched. (Zero specifies infinite cycles).
prescaler	Waveform prescaler value, to reduce the effective sampling rate by prescaler x 5.

Table 152 Output parameters for AWGqueueWaveform

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

Syntax

Table 153 API syntax for AWGqueueWaveform

Language	Syntax
C	`int SD_AOU_AWGqueueWaveform(int moduleID, int nAWG, int waveformNumber, int triggerMode, int startDelay, int cycles, int prescaler);`
C++	`int SD_AOU::AWGqueueWaveform(int nAWG, int waveformNumber, int triggerMode, int startDelay, int cycles, int prescaler);`
.NET	`int SD_AOU::AWGqueueWaveform(int nAWG, int waveformNumber, int triggerMode, int startDelay, int cycles, int prescaler);`
Python	`SD_AOU.AWGqueueWaveform(nAWG, waveformNumber, triggerMode, startDelay, cycles, prescaler)`
HVI	Refer to Table 154 HVI syntax and corresponding values for AWGqueueWaveform.

Table 154 HVI syntax and corresponding values for AWGqueueWaveform

Language	Syntax	Parameter	Possible values
Python	module.hvi.instruction_set.queue_waveform.id	channel.id	• Channel number
		prescaler.id	• Integer value for prescaler
		start_delay.id	• Delay in nanoseconds
		cycles.id	• Number of cycles to play the selected waveform
		waveform_number.id	• Waveform number to be queued. Must already be loaded into on-board memory
		trigger_mode.id	• AUTOTRIG • SWHVITRIG • SWHVITRIG_CYCLE • EXTTRIG • EXTTRIG_CYCLE
.NET	module.Hvi.InstructionSet.QueueWaveform.Id	Channel.Id	• Channel number
		Prescaler.Id	• Integer value for prescaler
		StartDelay.Id	• Delay in nanoseconds
		Cycles.Id	• Number of cycles to play the selected waveform
		WaveformNumber.Id	• Waveform number to be queued. Must already be loaded into on-board memory
		TriggerMode.Id	• AUTOTRIG • SWHVITRIG • SWHVITRIG_CYCLE • EXTTRIG • EXTTRIG_CYCLE

5.5.17: AWGflush

Description

Empties the queue of the selected AWG channel. Waveforms are not removed from the module's onboard RAM.

Parameters

Table 155 Input parameters for AWGflush

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

Table 156 Output parameters for AWGflush

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

Syntax

Table 157 API syntax for AWGflush

Language	Syntax
C	`int SD_AOU_AWGflush(int moduleID, int nAWG);`
C++	`int SD_AOU::AWGflush(int nAWG);`
.NET	`int SD_AOU::AWGflush(int nAWG);`
Python	`SD_AOU.AWGflush(nAWG)`
HVI	Refer to Table 158 HVI Action syntax for AWGflush for actions & Table 159 HVI Event syntax for AWGflush for events.

Table 158 HVI Action syntax for AWGflush

Language	Action Syntax	Parameters	Description
Python	`module.hvi.actions.awg<x>_queue_flush`	None	Empties the queue of the selected AWG channel. Waveforms are not removed from the module's onboard RAM.
.NET	`module.Hvi.Actions.AWG<x>QueueFlush`		

where, x = Channel number

Table 159 HVI Event syntax for AWGflush

Language	Event Syntax	Parameters	Description
Python	`module.hvi.events.awg<x>_queue_flushed`	None	Raises an event if awg<x>'s queue is flushed.
.NET	`module.Hvi.Events.Awg<x>QueueFlushed`		

where, x = Channel number

5.5.18: AWGfreezeOnStopEnable

Description

By default, when the queue is stopped the channel output returns to zero. If this feature is enabled, the output will maintain the last value.

Parameters

Table 160 Input parameters for AWGfreezeOnStopEnable

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
nAWG	AWG Channel number.
mode	Values can be set either to 1 (enabled) or 0 (disabled).

Table 161 Output parameters for AWGfreezeOnStopEnable

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

Syntax

Table 162 API syntax for AWGfreezeOnStopEnable

Language	Syntax
C	`int SD_AOU_AWGfreezeOnStopEnable(int moduleID, int nAWG, int mode);`
C++	`int SD_AOU::AWGfreezeOnStopEnable(int nAWG, int mode);`
.NET	`int SD_AOU::AWGfreezeOnStopEnable(int nAWG, int mode);`
Python	`SD_AOU.AWGfreezeOnStopEnable(nAWG, mode)`
HVI	Not available

5.5.19: AWGisFreezeOnStopEnabled

Description Returns 'True' if AWGFreezeOnStop is enabled. See AWGfreezeOnStopEnable.

Parameters **Table 163 Input parameters for AWGisFreezeOnStopEnabled**

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

Table 164 Output parameters for AWGisFreezeOnStopEnabled

Output Parameter name	Description
valOut	Returns '1' if AWGfreezeOnStopEnable function is enabled; else '0' if disabled.
errorOut	See Description of SD1 Error IDs.

Syntax **Table 165 API syntax for AWGisFreezeOnStopEnabled**

Language	Syntax
C	`int SD_AOU_AWGisFreezeOnStopEnabled(int moduleID, int nAWG);`
C++	`int SD_AOU::AWGisFreezeOnStopEnabled(int nAWG);`
.NET	`int SD_AOU::AWGisFreezeOnStopEnabled(int nAWG);`
Python	`SD_AOU.AWGisFreezeOnStopEnabled(nAWG)`
HVI	Not available

5.5.20: AWGstart

Description

Starts the selected AWG from the beginning of its queue. The generation starts immediately or when a trigger is received, depending on the trigger selection of the first waveform in the queue and provided that at least one waveform is queued in the AWG. See AWGqueueWaveform() or AWG().

After calling AWGstart, there is a minimum amount of "*time delay*" (in nanoseconds) required before the AWG can be triggered. This minimum amount of time delay is dependent on the waveform size (number of samples) to be played by the AWG; this time delay could be up to 900 ns.

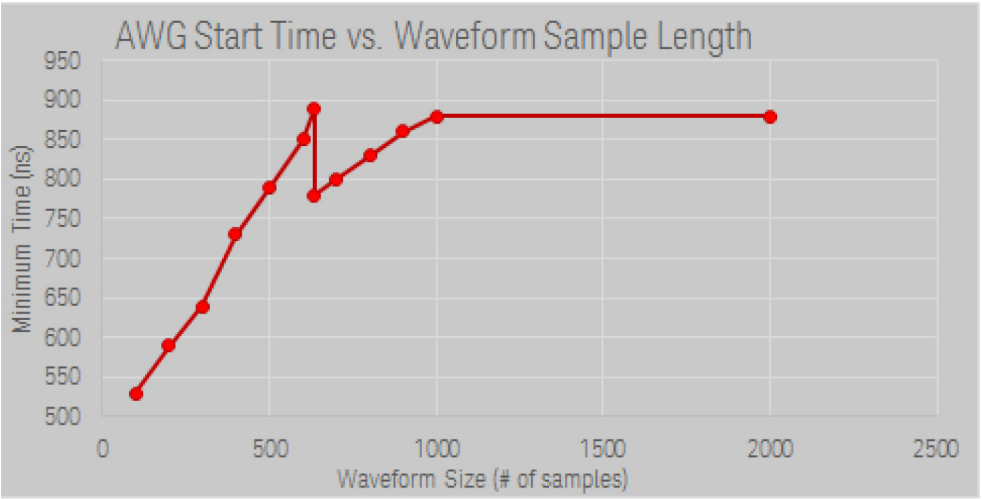


Figure 82 Time delay pattern after AWGstart is called

Parameters

Table 166 Input parameters for AWGstart

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

Table 167 Output parameters for AWGstart

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

Syntax

Table 168 API syntax for AWGstart

Language	Syntax
C	`int SD_AOU_AWGstart(int moduleID, int nAWG);`
C++	`int SD_AOU::AWGstart(int nAWG);`
.NET	`int SD_AOU::AWGstart(int nAWG);`
Python	`SD_AOU.AWGstart(nAWG)`
HVI	Refer to Table 169 HVI Action syntax for AWGstart for actions & Table 170 HVI Event syntax for AWGstart for events.

Table 169 HVI Action syntax for AWGstart

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

where, x = Channel number

Table 170 HVI Event syntax for AWGstart

Language	Event Syntax	Parameters	Description
Python	`module.hvi.events.awg<x>_underrun`	None	Playback FIFO is empty during a queue playback. This is signaling a race condition caused for playing a few short waveforms with no recovery time to avoid gaps between them.
	`module.hvi.events.awg<x>_queue_end`		Raises an event when awg <x> reaches the end of its queue.
	`module.hvi.events.awg<x>_waveform_start`		Raises an event if awg <x> is at the start of its waveform.
.NET	`module.Hvi.Events.AWG<x>Underrun`	None	Playback FIFO is empty during a queue playback. This is signaling a race condition caused for playing a few short waveforms with no recovery time to avoid gaps between them.
	`module.Hvi.Events.AWG<x>QueueEnd`		Raises an event when awg <x> reaches the end of its queue.
	`module.Hvi.Events.AWG<x>WfStart`		Raises an event if awg <x> is at the start of its waveform.

where, x = Channel number

5.5.21: AWGstartMultiple

Description

Starts the selected AWGs from the beginning of their queues. The generation will start immediately or when a trigger is received, depending on the trigger selection of the first waveform in their queues and provided that at least one waveform is queued in these AWGs. See AWGqueueWaveform() or AWG().

Parameters

Table 171 Input parameters for AWGstartMultiple

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
AWGmask	Mask to select the AWGs to be started (LSB is AWG 0, bit 1 is AWG 1, and so on).

Table 172 Output parameters for AWGstartMultiple

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

Syntax

Table 173 API syntax for AWGstartMultiple

Language	Syntax
C	`int SD_AOU_AWGstartMultiple(int moduleID, int AWGmask);`
C++	`int SD_AOU::AWGstartMultiple(int AWGmask);`
.NET	`int SD_AOU::AWGstartMultiple(int AWGmask);`
Python	`SD_AOU.AWGstartMultiple(AWGmask)`
HVI	Not available

5.5.22: AWGpause

Description

Pauses the selected AWG leaving the last waveform point at the output and ignoring all incoming triggers. The AWG can be resumed by calling AWGresume().

Parameters

Table 174 Input parameters for AWGpause

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

Table 175 Output parameters for AWGpause

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

Syntax

Table 176 API syntax for AWGpause

Language	Syntax
C	`int SD_AOU_AWGpause(int moduleID, int nAWG);`
C++	`int SD_AOU::AWGpause(int nAWG);`
.NET	`int SD_AOU::AWGpause(int nAWG);`
Python	`SD_AOU.AWGpause(nAWG)`
HVI	Refer to Table 177 HVI Action syntax for AWGpause for actions.

Table 177 HVI Action syntax for AWGpause

Language	Action Syntax	Parameters	Description
Python	`module.hvi.actions.awg<x>_pause`	None	None
.NET	`module.Hvi.Actions.Awg<x>Pause`		

where, x = Channel number

5.5.23: AWGpauseMultiple

Description

Pauses the selected AWGs leaving the last waveform point at the output of each channel, and ignoring all incoming triggers. The AWGs can be resumed by calling AWGresumeMultiple().

Parameters

Table 178 Input parameters for AWGpauseMultiple

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
AWGmask	Mask to select the AWG Channels to be paused (LSB is Channel 1, bit 1 is Channel 2, and so on).

Table 179 Output parameters for AWGpauseMultiple

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

Syntax

Table 180 API syntax for AWGpauseMultiple

Language	Syntax
C	`int SD_AOU_AWGpauseMultiple(int moduleID, int AWGmask);`
C++	`int SD_AOU::AWGpauseMultiple(int AWGmask);`
.NET	`int SD_AOU::AWGpauseMultiple(int AWGmask);`
Python	`SD_AOU.AWGpauseMultiple(AWGmask)`
HVI	Not available

5.5.24: AWGresume

Description

Resumes the operation of the selected AWG from the current position of the queue. The waveform generation can be paused by calling AWGpause().

Parameters

Table 181 Input parameters for AWGresume

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

Table 182 Output parameters for AWGresume

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

Syntax

Table 183 API syntax for AWGresume

Language	Syntax
C	`int SD_AOU_AWGresume(int moduleID, int nAWG);`
C++	`int SD_AOU::AWGresume(int nAWG);`
.NET	`int SD_AOU::AWGresume(int nAWG);`
Python	`SD_AOU.AWGresume(nAWG)`
HVI	Refer to Table 184 HVI Action syntax for AWGresume for actions.

Table 184 HVI Action syntax for AWGresume

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

where, x = Channel number

5.5.25: AWGresumeMultiple

Description

Resumes the operation of the selected AWGs from the current position of their respective queues. The waveform generation of multiple AWGs can be paused by calling AWGpauseMultiple().

Parameters

Table 185 Input parameters for AWGresumeMultiple

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
AWGmask	Mask to select the AWG Channels to be resumed (LSB is Channel 1, bit 1 is Channel 2, and so on).

Table 186 Output parameters for AWGresumeMultiple

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

Syntax

Table 187 API syntax for AWGresumeMultiple

Language	Syntax
C	`int SD_AOU_AWGresumeMultiple(int moduleID, int AWGmask);`
C++	`int SD_AOU::AWGresumeMultiple(int AWGmask);`
.NET	`int SD_AOU::AWGresumeMultiple(int AWGmask);`
Python	`SD_AOU.AWGresumeMultiple(AWGmask)`
HVI	Not available

5.5.26: AWGstop

Description

Stops the selected AWG, setting the output to zero, and resetting the AWG queue to its initial position. All following incoming triggers are ignored.

Parameters

Table 188 Input parameters for AWGstop

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

Table 189 Output parameters for AWGstop

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

Syntax

Table 190 API syntax for AWGstop

Language	Syntax
C	`int SD_AOU_AWGstop(int moduleID, int nAWG);`
C++	`int SD_AOU::AWGstop(int nAWG);`
.NET	`int SD_AOU::AWGstop(int nAWG);`
Python	`SD_AOU.AWGstop(nAWG)`
HVI	Refer to Table 191 HVI Action syntax for AWGstop for actions.

Table 191 HVI Action syntax for AWGstop

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

where, x = Channel number

5.5.27: AWGstopMultiple

Description

Stops the selected AWGs, sets their outputs to zero and resets their respective queues to the initial positions. All following incoming triggers are ignored.

Parameters

Table 192 Input parameters for AWGstopMultiple

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
AWGmask	Mask to select the AWG Channels to be stopped (LSB is Channel 1, bit 1 is Channel 2, and so on).

Table 193 Output parameters for AWGstopMultiple

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

Syntax

Table 194 API syntax for AWGstopMultiple

Language	Syntax
C	`int SD_AOU_AWGstopMultiple(int moduleID, int AWGmask);`
C++	`int SD_AOU::AWGstopMultiple(int AWGmask);`
.NET	`int SD_AOU::AWGstopMultiple(int AWGmask);`
Python	`SD_AOU.AWGstopMultiple(AWGmask)`
HVI	Not available

5.5.28: AWGjumpNextWaveform

Description

Forces a jump to the next waveform in the AWG queue. The jump is executed once the current waveform has finished a complete cycle.

Parameters

Table 195 Input parameters for AWGjumpNextWaveform

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

Table 196 Output parameters for AWGjumpNextWaveform

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

Syntax

Table 197 API syntax for AWGjumpNextWaveform

Language	Syntax
C	`int SD_AOU_AWGjumpNextWaveform(int moduleID, int nAWG);`
C++	`int SD_AOU::AWGjumpNextWaveform(int nAWG);`
.NET	`int SD_AOU::AWGjumpNextWaveform(int nAWG);`
Python	`SD_AOU.AWGjumpNextWaveform(nAWG)`
HVI	Refer to Table 198 HVI Action syntax for AWGjumpNextWaveform for actions.

Table 198 HVI Action syntax for AWGjumpNextWaveform

Language	Action Syntax	Parameters	Description
Python	`module.hvi.actions.awg<x>_jump_next_waveform`	None	None
.NET	`module.Hvi.Actions.Awg<x>JumpNextWaveform`		

where, x = Channel number

5.5.29: AWGjumpNextWaveformMultiple

Description

Forces a jump to the next waveform in the queue of several AWGs. The jumps are executed once the current waveforms have finished a complete cycle.

Parameters

Table 199 Input parameters for AWGjumpNextWaveformMultiple

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
AWGmask	Mask to select the AWG Channels (LSB is Channel 1, bit 1 is Channel 2, and so on).

Table 200 Output parameters for AWGjumpNextWaveformMultiple

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

Syntax

Table 201 API syntax for AWGjumpNextWaveformMultiple

Language	Syntax
C	`int SD_AOU_AWGjumpNextWaveformMultiple(int moduleID, int AWGmask);`
C++	`int SD_AOU::AWGjumpNextWaveformMultiple(int AWGmask);`
.NET	`int SD_AOU::AWGjumpNextWaveformMultiple(int AWGmask);`
Python	`SD_AOU.AWGjumpNextWaveformMultiple(AWGmask)`
HVI	Not available

5.5.30: AWGisRunning

Description Returns '1' if the AWG is running or '0' if it is stopped.

Parameters

Table 202 Input parameters for AWGisRunning

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

Table 203 Output parameters for AWGisRunning

Output Parameter name	Description
running	'1' if the AWG is running or '0' if it is stopped.
errorOut	See Description of SD1 Error IDs.

Syntax

Table 204 API syntax for AWGisRunning

Language	Syntax
C	`int SD_AOU_AWGisRunning(int moduleID, int nAWG);`
C++	`int SD_AOU::AWGisRunning(int nAWG);`
.NET	`int SD_AOU::AWGisRunning(int nAWG);`
Python	`SD_AOU.AWGisRunning(nAWG)`
HVI	Refer to Table 205 HVI Event syntax for AWGisRunning for events.

Table 205 HVI Event syntax for AWGisRunning

Language	Action Syntax	Parameters	Description
Python	`module.hvi.events.awg<x>_queue_running`	None	Raises an event if awg <x>'s queue is running.
.NET	`module.Hvi.Events.Awg<x>QueueRunning`		

where, x = Channel number

5.5.31: AWGnWFplaying

Description

Returns the waveformNumber (waveform identifier) of the waveform, which is currently being generated by the AWG.

Parameters

Table 206 Input parameters for AWGnWFplaying

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

Table 207 Output parameters for AWGnWFplaying

Output Parameter name	Description
waveformNumber	Waveform identifier. See waveformLoad.
errorOut	See Description of SD1 Error IDs.

Syntax

Table 208 API syntax for AWGnWFplaying

Language	Syntax
C	`int SD_AOU_AWGnWFplaying(int moduleID, int nAWG);`
C++	`int SD_AOU::AWGnWFplaying(int nAWG);`
.NET	`int SD_AOU::AWGnWFplaying(int nAWG);`
Python	`SD_AOU.AWGnWFplaying(nAWG)`
HVI	Not available

5.5.32: AWGtriggerExternalConfig

Description

Configures the external triggers for the selected AWG. The external trigger is used in case the waveform is queued with the 'external trigger mode' option.See AWGqueueWaveform().

Parameters

Table 209 Input parameters for AWGtriggerExternalConfig

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
nAWG	AWG Channel number.
externalSource	AWG external trigger source. Refer to Table 36 External Trigger options and corresponding values.
triggerBehavior	AWG external trigger behavior. Refer to Table 37 External Trigger behavior and corresponding values.
sync	Set to '0' for immediate trigger or '1' to synchronize with nearest CLK edge.

Table 210 Output parameters for AWGtriggerExternalConfig

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

Syntax

Table 211 API syntax for AWGtriggerExternalConfig

Language	Syntax
C	`int SD_AOU_AWGtriggerExternalConfig(int moduleID, int nAWG, int externalSource, int triggerBehavior, int sync);`
C++	`int SD_AOU::AWGtriggerExternalConfig(int nAWG, int externalSource, int triggerBehavior, int sync);` `int SD_AOU::AWGtriggerExternalConfig(int nAWG, int externalSource, int triggerBehavior);`
.NET	`int SD_AOU::AWGtriggerExternalConfig(int nAWG, int externalSource, int triggerBehavior, int sync);` `int SD_AOU::AWGtriggerExternalConfig(int nAWG, int externalSource, int triggerBehavior);`
Python	`SD_AOU.AWGtriggerExternalConfig(nAWG, externalSource, triggerBehavior, sync)`
HVI	Not available

5.5.33: AWGtrigger

Description

Triggers the selected AWG. The waveform waiting in the current position of the queue is launched provided it is configured with VI/HVI Trigger (triggerMode = 1 or 5).

Parameters

Table 212 Input parameters for AWGtrigger

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

Table 213 Output parameters for AWGtrigger

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

Syntax

Table 214 API syntax for AWGtrigger

Language	Syntax
C	`int SD_AOU_AWGtrigger(int moduleID, int nAWG);`
C++	`int SD_AOU::AWGtrigger(int nAWG);`
.NET	`int SD_AOU::AWGtrigger(int nAWG);`
Python	`SD_AOU.AWGtrigger(nAWG)`
HVI	Refer to Table 215 HVI Action syntax for AWGtrigger for actions & Table 216 HVI Event syntax for AWGtrigger for events.

Table 215 HVI Action syntax for AWGtrigger

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

where, x = Channel number

Table 216 HVI Event syntax for AWGtrigger

Language	Event Syntax	Parameters	Description
Python	`module.hvi.events.awg<x>_trigger_loopback`	None	None
.NET	`module.Hvi.Events.AWG<x>TriggerLoopback`		

where, x = Channel number

5.5.34: AWGtriggerMultiple

Description

Triggers the selected AWGs. The waveforms waiting in the current position of their respective queues is launched provided they are configured with an VI/HVI Trigger (triggerMode = 1 or 5).

Parameters

Table 217 Input parameters for AWGtriggerMultiple

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
AWGmask	Mask to select the AWG Channels to be triggered (LSB is Channel 1, bit 1 is Channel 2, and so on).

Table 218 Output parameters for AWGtriggerMultiple

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

Syntax

Table 219 API syntax for AWGtriggerMultiple

Language	Syntax
C	`int SD_AOU_AWGtriggerMultiple(int moduleID, int AWGmask);`
C++	`int SD_AOU::AWGtriggerMultiple(int AWGmask);`
.NET	`int SD_AOU::AWGtriggerMultiple(int AWGmask);`
Python	`SD_AOU.AWGtriggerMultiple(AWGmask)`
HVI	Not available

5.5.35: triggerIOconfig

Description Configures the trigger line direction. See "Working with I/O Triggers" on page 35.

Parameters

Table 220 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 38 Trigger Direction options and corresponding values.
port	Applicable for M33xxA COMBO cards only.

Table 221 Output parameters for triggerIOconfig

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

Syntax

Table 222 API syntax for triggerIOconfig (on M32xxA AWG modules)

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

Table 223 API syntax for triggerIOconfig (on M33xxA 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

5.5.36: triggerIOwrite

Description

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

Parameters

Table 224 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 39 Sync mode options and corresponding values.
port	Applicable for M33xxA COMBO cards only.

Table 225 Output parameters for triggerIOwrite

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

Syntax

Table 226 API syntax for triggerIOwrite (on M32xxA AWG modules)

Language	Syntax
C	`int SD_AOU_triggerIOwrite(int moduleID, int value, int syncMode);`
C++	`int SD_AOU::triggerIOwrite(int value, int syncMode);`
.NET	`int SD_AOU::triggerIOwrite(int value, int syncMode);`
Python	`SD_AOU.triggerIOwrite(value, syncMode)`
HVI	Not available

Table 227 API syntax for triggerIOwrite (on M33xxA 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)`
HVI	Not available

5.5.37: triggerIOread

Description Reads the trigger input.

Parameters

Table 228 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 229 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 230 API syntax for triggerIOread (on M32xxA AWG modules)

Language	Syntax
C	`int SD_AOU_triggerIOread(int moduleID);`
C++	`int SD_AOU::triggerIOread();`
.NET	`int SD_AOU::triggerIOread();`
Python	`SD_AOU.triggerIOread()`
HVI	Not available

Table 231 API syntax for triggerIOread (on M33xxA 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

5.5.38: clockSetFrequency

Description

Sets the module clock frequency. (See CLKsys in "FlexCLK Technology (models w/ variable sampling rate)" on page 36).

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 M320xA/M330xA modules.

Parameters

Table 232 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 40 Variable clock system operation mode options and corresponding values.
port	Applicable for M33xxA COMBO cards only.

Table 233 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 234 API syntax for clockSetFrequency (on M32xxA AWG modules)

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

Table 235 API syntax for clockSetFrequency (on M33xxA 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

5.5.39: clockGetFrequency

Description

Returns the value in Hz of the module sample rate frequency. (See CLKsys in "FlexCLK Technology (models w/ variable sampling rate)" on page 36). It may differ from the frequency set with the clockSetFrequency() due to the hardware frequency resolution.

Parameters

Table 236 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 237 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 238 API syntax for clockGetFrequency (on M32xxA AWG modules)

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

Table 239 API syntax for clockGetFrequency (on M33xxA 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

5.5.40: clockGetSyncFrequency

Description Returns the frequency of the internal CLKsync signal in Hz. (See CLKsys in "FlexCLK Technology (models w/ variable sampling rate)" on page 36).

Parameters

Table 240 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 241 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 242 API syntax for clockGetSyncFrequency (on M32xxA AWG modules)

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

Table 243 API syntax for clockGetSyncFrequency (on M33xxA 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(int port)`
HVI	Not available

5.5.41: clockResetPhase

Description

Sets the module in a synchronous state, waiting for the first trigger to reset the phase of the internal clocks CLKsync and AWG. (See CLKsys in "FlexCLK Technology (models w/ variable sampling rate)" on page 36).

Parameters

Table 244 Input parameters for clockResetPhase

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
triggerBehavior	AWG external trigger behavior. Refer to Table 37 External Trigger behavior and corresponding values.
PXItrigger	PXI trigger number. Refer to Table 36 External Trigger options and corresponding values.
skew	Skew between PXI CLK10 and CLKsync in multiples of 10 ns.
port	Applicable for M33xxA COMBO cards only.

Table 245 Output parameters for clockResetPhase

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

Syntax

Table 246 API syntax for clockResetPhase (on M32xxA AWG modules)

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

Table 247 API syntax for clockResetPhase (on M33xxA 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

5.5.42: AWGqueueConfig

Description

Configures the cyclic mode of the queue. All waveforms must be already queued in one of the AWGs of the module. See “AWG Waveform Queue System” on page 20.

Parameters

Table 248 Input parameters for AWGqueueConfig

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
nAWG	AWG channel number
mode	Operation mode of the queue: ‘0’ indicates ‘One shot’ and ‘1’ indicates ‘Cyclic’. See AWG Waveform Queue System Examples.

Table 249 Output parameters for AWGqueueConfig

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

Syntax

Table 250 API syntax for AWGqueueConfig

Language	Syntax
C	`int SD_AOU_AWGqueueConfig(int moduleID, int nAWG, int mode);`
C++	`int SD_AOU::AWGqueueConfig(int nAWG, int mode);`
.NET	`int SD_AOU::AWGqueueConfig(int nAWG, int mode);`
Python	`SD_AOU.AWGqueueConfig(nAWG, mode)`
HVI	Not available

5.5.43: AWGqueueConfigRead

Description

Reads the value of the cyclic mode of the queue. All waveforms must be already queued in one of the module's Arbitrary Waveform Generators. (see “AWG Waveform Queue System” on page 20)

Parameters

Table 251 Input parameters for AWGqueueConfigRead

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

Table 252 Output parameters for AWGqueueConfigRead

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

Syntax

Table 253 API syntax for AWGqueueConfigRead

Language	Syntax
C	`int SD_AOU_AWGqueueConfigRead(int moduleID, int nAWG);`
C++	`int SD_AOU::AWGqueueConfigRead(int nAWG);`
.NET	`int SD_AOU::AWGqueueConfigRead(int nAWG);`
Python	`SD_AOU.AWGqueueConfigRead(nAWG)`
HVI	Not available

5.5.44: AWGqueueMarkerConfig

Description

Configures the marker generation for each AWG. All waveforms must be already queued (AWG Queue System) in one of the module's Arbitrary Waveform Generators. For this function to operate correctly, the markers must be configured before the waveforms start to play. Each AWG channel can be configured to output a marker on the PXI backplane or the front panel trigger.

Parameters

Table 254 Input parameters for AWGqueueMarkerConfig

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
nAWG	AWG Channel number
markerMode	Operation mode of the marker '0' indicates 'Disabled', '1' indicates 'On Start Event (when Start trigger is received)', '2' indicates 'On First Sample of Waveform (after WF startDelay)', '3' indicates 'On Every Cycle'.
trgPXImask	Mask to select PXI triggers to use. bit0 indicates PXItrg0, bit1 indicates PXItrg1 and so on.
trgIOmask	Mask to select front-panel triggers to use. bit0 indicates TriggerIO and so on.
value	'0' indicates 'Low', '1' indicates 'High'. Note that PXItrigger are active low signals, '1' will generate a '0' pulse.
syncMode	'0' is synchronized to CLKsys and '1' is synchronized to 10 MHz reference clock.
length	Pulse length of the marker, in units of nanoseconds. Enter the value of length as a multiple of 10. For example, if you enter 30, it is considered as 30ns.
delay	Delay to add before the marker pulse. (markerMode selects the start point of the marker, after which the delay is added).

Table 255 Output parameters for AWGqueueMarkerConfig

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

Syntax

Table 256 API syntax for AWGqueueMarkerConfig

Language	Syntax
C	`int SD_AOU_AWGqueueMarkerConfig(int moduleID, int nAWG, int markerMode, int trgPXImask, int trgIOmask, int value, int syncMode, int length, int delay);`
C++	`int SD_AOU::AWGqueueMarkerConfig(int nAWG, int markerMode, int trgPXImask, int trgIOmask, int value, int syncMode, int length, int delay);`
.NET	`int SD_AOU::AWGqueueMarkerConfig(int nAWG, int markerMode, int trgPXImask, int trgIOmask, int value, int syncMode, int length, int delay);`
Python	`SD_AOU.AWGqueueMarkerConfig(nAWG, markerMode, trgPXImask, trgIOmask, value, syncMode, length, delay)`
HVI	Refer to Table 257 HVI Event syntax for AWGqueueMarkerConfig for events.

Table 257 HVI Event syntax for AWGqueueMarkerConfig

Language	Action Syntax	Parameters	Description
Python	`module.hvi.events.awg<x>_queue_marker`	None	Configure this event to occur on the first point played every cycle of a waveform request, only in the first cycle, with a delay and/or synchronized with a common 10 MHz clock.
.NET	`module.Hvi.Events.Awg<x>QueueMarker`		

where, x = Channel number

5.5.45: AWGqueueSyncMode

Description Configures the synchronization mode of the queue. All waveforms must be already queued "AWGqueueWaveform()" in one of the module's AWGs.

Parameters

Table 258 Input parameters for AWGqueueSyncMode

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
nAWG	AWG channel number
syncMode	For '0', it is synchronized to CLKSYS. For '1', it is synchronized to the 10 MHz reference clock.

Table 259 Output parameters for AWGqueueSyncMode

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

Syntax

Table 260 API syntax for AWGqueueSyncMode

Language	Syntax
C	`int SD_AOU_AWGqueueSyncMode(int moduleID, int nAWG, int syncMode);`
C++	`int SD_AOU::AWGqueueSyncMode(int nAWG, int syncMode);`
.NET	`int SD_AOU::AWGqueueSyncMode(int nAWG, int syncMode);`
Python	`SD_AOU.AWGqueueSyncMode(nAWG, syncMode)`
HVI	Not available

5.5.46: AWGqueueIsEmpty

Description Returns '1' if AWG queue is empty, else '0'.

Parameters

Table 261 Input parameters for AWGqueueIsEmpty

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

Table 262 Output parameters for AWGqueueIsEmpty

Output Parameter name	Description
ValOut	'1' if AWG queue is empty, else '0'.
errorOut	See Description of SD1 Error IDs.

Syntax

Table 263 API syntax for AWGqueueIsEmpty

Language	Syntax
C	`int SD_AOU_AWGqueueIsEmpty(int moduleID, int nAWG);`
C++	`int SD_AOU::AWGqueueIsEmpty(int nAWG);`
.NET	`int SD_AOU::AWGqueueIsEmpty(int nAWG);`
Python	`SD_AOU.AWGqueueIsEmpty(nAWG)`
HVI	Refer to Table 264 HVI Event syntax for AWGqueueIsEmpty for events.

Table 264 HVI Event syntax for AWGqueueIsEmpty

Language	Action Syntax	Parameters	Description
Python	`module.hvi.events.awg<x>_queue_empty`	None	Raises the event when the queue becomes empty for Channel <x>.
.NET	`module.Hvi.Events.Awg<x>QueueEmpty`		

where, x = Channel number

5.5.47: AWGqueueIsFull

Description Returns '1' if AWGqueue is full, else '0'.

Parameters

Table 265 Input parameters for AWGqueueIsFull

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

Table 266 Output parameters for AWGqueueIsFull

Output Parameter name	Description
isFullOut	'1' if AWG queue is full, else '0'.
errorOut	See Description of SD1 Error IDs.

Syntax

Table 267 API syntax for AWGqueueIsFull

Language	Syntax
C	`int SD_AOU_AWGqueueIsFull(int moduleID, int nAWG);`
C++	`int SD_AOU::AWGqueueIsFull(int nAWG);`
.NET	`int SD_AOU::AWGqueueIsFull(int nAWG);`
Python	`SD_AOU.int AWGqueueIsFull(nAWG)`
HVI	Refer to Table 268 HVI Event syntax for AWGqueueIsFull for events.

Table 268 HVI Event syntax for AWGqueueIsFull

Language	Action Syntax	Parameters	Description
Python	`module.hvi.events.awg<x>_queue_full`	None	Raises the event when the queue becomes full for channel <x>.
.NET	`module.Hvi.Events.Awg<x>QueueFull`		

where, x = Channel number

5.5.48: AWGqueueRemaining

Description Returns the size remaining on queue.

Parameters

Table 269 Input parameters for AWGqueueRemaining

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

Table 270 Output parameters for AWGqueueRemaining

Output Parameter name	Description
sizeOut	Size available on queue.
errorOut	See Description of SD1 Error IDs.

Syntax

Table 271 API syntax for AWGqueueRemaining

Language	Syntax
C	`int SD_AOU_AWGqueueRemaining(int moduleID, int nAWG);`
C++	`int SD_AOU::AWGqueueRemaining(int nAWG);`
.NET	`int SD_AOU::AWGqueueRemaining(int nAWG);`
Python	`SD_AOU.AWGqueueRemaining(nAWG)`
HVI	Not available

5.5.49: freqToInt

Description Helper method to convert frequency from 'double' to 'integer'. This conversion method is intended to be used with HVI native instructions only, when assigning literals to the registers for frequency.

Parameters

Table 272 Input parameters for freqToInt

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

Table 273 Output parameters for freqToInt

Output Parameter name	Description
freqOut	Converted frequency in integer
errorOut	See Description of SD1 Error IDs.

Syntax

Table 274 API syntax for freqToInt

Language	Syntax
C	`long long SD_AOU_freqToInt(int moduleID, double freq);`
C++	`long long SD_AOU::freqToInt(double freq);`
.NET	`long SD_AOU::freqToInt(double freq);`
Python	`SD_AOU.freqToInt(freq)`
HVI	Not available

5.5.50: freqGainToInt

Description Helper method to convert deviation gain (in frequency modulation) from 'double' to 'integer'. This conversion method is intended to be used with HVI native instructions only, when assigning literals to the registers for deviation gain (in frequency modulation).

Parameters

Table 275 Input parameters for freqGainToInt

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
freqGain	Input deviation gain in frequency modulation.

Table 276 Output parameters for freqGainToInt

Output Parameter name	Description
freqGainOut	Converted deviation gain in integer
errorOut	See Description of SD1 Error IDs.

Syntax

Table 277 API syntax for freqGainToInt

Language	Syntax
C	`int SD_AOU_freqGainToInt(int moduleID, double freqGain);`
C++	`int SD_AOU::freqGainToInt(double freqGain);`
.NET	`int SD_AOU::freqGainToInt(double freqGain);`
Python	`SD_AOU.freqGainToInt(freqGain)`
HVI	Not available

5.5.51: phaseToInt

Description

Helper method to convert phase from 'double' to 'integer'. This conversion method is intended to be used with HVI native instructions only, when assigning literals to the registers for phase.

Parameters

Table 278 Input parameters for phaseToInt

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

Table 279 Output parameters for phaseToInt

Output Parameter name	Description
phaseOut	Converted phase in integer
errorOut	See Description of SD1 Error IDs.

Syntax

Table 280 API syntax for phaseToInt

Language	Syntax
C	`long long SD_AOU_phaseToInt(int moduleID, double phase);`
C++	`long long SD_AOU::phaseToInt(double phase);`
.NET	`long SD_AOU::phaseToInt(double phase);`
Python	`SD_AOU.phaseToInt(phase)`
HVI	Not available

5.5.52: phaseGainToInt

Description Helper method to convert deviation gain (in phase modulation) from 'double' to 'integer'. This conversion method is intended to be used with HVI native instructions only, when assigning literals to the registers for deviation gain (in phase modulation).

Parameters

Table 281 Input parameters for phaseGainToInt

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
phaseGain	Input deviation gain in phase modulation.

Table 282 Output parameters for phaseGainToInt

Output Parameter name	Description
phaseGainOut	Converted deviation gain in integer
errorOut	See Description of SD1 Error IDs.

Syntax

Table 283 API syntax for phaseGainToInt

Language	Syntax
C	`int SD_AOU_phaseGainToInt(int moduleID, double phaseGain);`
C++	`int SD_AOU::phaseGainToInt(double phaseGain);`
.NET	`int SD_AOU::phaseGainToInt(double phaseGain);`
Python	`SD_AOU.phaseGainToInt(phaseGain)`
HVI	Not available

5.5.53: setDigitalFilterMode

NOTE This function applies to M3202A PXIe AWGs (1 GSa/s) only. Furthermore, it is supported by the K32 and K41 FPGAs only.

Description

Sets the digital filter mode.

Parameters

Table 284 Input parameters for setDigitalFilterMode

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
mode	Possible values are: ▪ AOU_FILTER_OFF = 0 ▪ AOU_FILTER_FLATNESS = 1 ▪ AOU_FILTER_FIFTEEN_TAP = 3

Table 285 Output parameters for setDigitalFilterMode

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

Syntax

Table 286 API syntax for setDigitalFilterMode

Language	Syntax
C	`int SD_AOU_setDigitalFilterMode(int moduleID, int mode);`
C++	`int SD_AOU::setDigitalFilterMode(int mode);`
.NET	`int SD_AOU::setDigitalFilterMode(int mode);`
Python	`SD_AOU.setDigitalFilterMode(mode)`
HVI	Not available

5.5.54: voltsToInt

Description Helper method to convert voltage from 'double' to 'integer'. This conversion method is intended to be used with HVI native instructions only, when assigning literals to the registers for amplitude, offset and deviation gain (in amplitude modulation).

Parameters

Table 287 Input parameters for voltsToInt

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

Table 288 Output parameters for voltsToInt

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

Syntax

Table 289 API syntax for voltsToInt

Language	Syntax
C	`int SD_AOU_voltsToInt(int moduleID, double volts);`
C++	`int SD_AOU::voltsToInt(double volts);`
.NET	`int SD_AOU::voltsToInt(double volts);`
Python	`SD_AOU.voltsToInt(volts)`
HVI	Not available

5.5.55: waveformAddToList

Description

Adds the waveform to the list.

Parameters

Table 290 Input parameters for waveformAddToList

Input Parameter name	Description
moduleID	(Non-object-oriented languages only) Module identifier, returned by open.
waveformID	Waveform identifier. See SD_Wave functions.
waveformNumber	Waveform number to identify the waveform in subsequent related function calls. This value must be in the (0 to n) range, and in order to optimize the memory usage, it should be as low as possible.
paddingMode	If '0', waveform is loaded as is, and zeros are added at the end if the number of points is not a multiple of the number required by the AWG. If '1', waveform is loaded n times (using DMA) until the total number of points is a multiple of the number required by the AWG (only for waveforms with an even number of points).

Table 291 Output parameters for waveformAddToList

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

Syntax

Table 292 API syntax for waveformAddToList

Language	Syntax
C	`int SD_AOU_waveformAddToList(int moduleID, int waveformID, int waveformNumber, int paddingMode);`
C++	`int SD_AOU::waveformAddToList(SD_Wave waveformObject, int waveformNumber, int paddingMode);` `int SD_AOU::waveformAddToList(SD_Wave waveformObject, int waveformNumber);`
.NET	`int SD_AOU::waveformAddToList(SD_Wave waveformObject, int waveformNumber, int paddingMode);` `int SD_AOU::waveformAddToList(SD_Wave waveformObject, int waveformNumber);`
Python	`SD_AOU.waveformAddToList(self, waveformObject, waveformNumber, paddingMode = 0)`
HVI	Not available

5.5.56: waveformListLoad

Description Loads the list of waveform for the module.

Parameters

Table 293 Input parameters for waveformListLoad

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

Table 294 Output parameters for waveformListLoad

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

Syntax

Table 295 API syntax for waveformListLoad

Language	Syntax
C	`int SD_AOU_waveformListLoad(int moduleID);`
C++	`int SD_AOU::waveformListLoad();`
.NET	`int SD_AOU::waveformListLoad();`
Python	`SD_AOU.waveformListLoad()`
HVI	Not available

Section 5.6: SD_Wave functions

5.6.1: new

Description Creates a waveform object from data points contained in an array in memory or in a file.

NOTE (Advanced) Memory Usage: Waveforms created with 'new' are stored in the PC RAM, not in the module's onboard RAM. The limitation in the number of waveforms and their sizes is given by the amount of PC RAM.

Parameters

Table 296 Input parameters for new

Input Parameter name	Description
waveformType	Waveform type. Defines the type of waveform to create. This parameter is used to internally configure the AWG and selects the waveform type which matches the organizational structure of the waveform data file being loaded. Refer to Table 34 Waveform types and corresponding values. Also, see AWG Waveform Types.
waveformPoints	Number of points of the waveform, which must be a multiple of a certain number of points. See *AWG specifications* in the Data Sheet.
waveformDataA	Array with waveform points. Analog waveforms are defined with floating point numbers, which correspond to a normalized amplitude (-1 to 1).
waveformDataB	Array with waveform points, only for dual/IQ waveforms.
waveformFile	File containing the waveform points.

Table 297 Output parameters for new

Output Parameter name	Description
waveformID	(Non-object-oriented languages only) Waveform identifier. Negative numbers indicate an error. See Description of SD1 Error IDs.
errorOut	See Description of SD1 Error IDs.

Syntax

Table 298 API syntax for new

Language	Syntax

C	int SD_Wave_newFromArrayInteger(int waveformType, int waveformPoints, int *waveformDataA, int *waveformDataB);
	int SD_Wave_newFromArrayDouble(int waveformType, int waveformPoints, double *waveformDataA, double *waveformDataB);
	int SD_Wave_newFromFile(const char *waveformFile);

C++	SD_Wave(std::string waveformFile); SD_Wave(int waveformType, int waveformPoints, std::vector<double> waveformDataA, std::vector<double> waveformDataB); SD_Wave(int waveformType, std::vector<double> waveformDataA, std::vector<double> waveformDataB); SD_Wave(int waveformType, int waveformPoints, std::vector<double> waveformDataA); SD_Wave(int waveformType, std::vector<double> waveformDataA); SD_Wave(int waveformType, int waveformPoints, std::vector<int> waveformDataA, std::vector<int> waveformDataB); SD_Wave(int waveformType, std::vector<int> waveformDataA, std::vector<int> waveformDataB); SD_Wave(int waveformType, int waveformPoints, std::vector<int> waveformDataA); SD_Wave(int waveformType, std::vector<int> waveformDataA);
.NET	SD_Wave(string waveformFile); SD_Wave(int waveformType, double[] waveformDataA); SD_Wave(int waveformType, int waveformPoints, double[] waveformDataA); SD_Wave(int waveformType, double[] waveformDataA, double[] waveformDataB); SD_Wave(int waveformType, int waveformPoints, double[] waveformDataA, double[] waveformDataB); SD_Wave(int waveformType, int[] waveformDataA); SD_Wave(int waveformType, int waveformPoints, int[] waveformDataA); SD_Wave(int waveformType, int[] waveformDataA, int[] waveformDataB); SD_Wave(int waveformType, int waveformPoints, int[] waveformDataA, int[] waveformDataB);
Python	SD_Wave.newFromArrayDouble(self, waveformType, waveformDataA, waveformDataB) SD_Wave. newFromArrayDoubleNP(self, waveformType, waveformDataA, waveformDataB) SD_Wave.newFromArrayInteger(self, waveformType, waveformDataA, waveformDataB) SD_Wave.newFromFile(waveformFile)
HVI	Not available

5.6.2: delete

Description

Removes a waveform created with the new function.

NOTE (Advanced) Onboard waveforms: Waveforms are removed from the PC RAM only, not from the module's onboard RAM.

Parameters

Table 299 Input parameters for delete

Input Parameter name	Description
waveformID	Waveform identifier (returned by new).

Table 300 Output parameters for delete

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

Syntax

Table 301 API syntax for delete

Language	Syntax
C	`int SD_Wave_delete(int waveformID);`
C++	Handled in the destructor.
.NET	Automatically destroyed by the .NET garbage collector.
Python	Managed by Python.
HVI	Not available

5.6.3: getStatus

Description Returns the waveform loading status.

Parameters

Table 302 Input parameters for getStatus

Input Parameter name	Description
waveformID	Waveform type. Defines the type of waveform to create. This parameter is used to internally configure the AWG and selects the waveform type which matches the organizational structure of the waveform data file being loaded. Refer to Table 34 Waveform types and corresponding values. Also, see AWG Waveform Types.

Table 303 Output parameters for getStatus

Output Parameter name	Description
waveformID	(Non-object-oriented languages only) Waveform identifier. Negative numbers indicate an error. See Description of SD1 Error IDs.
errorOut	See Description of SD1 Error IDs.

Syntax

Table 304 API syntax for getStatus

Language	Syntax
C	`int SD_Wave_getStatus(int waveformID);`
C++	`int SD_Wave::getStatus();`
.NET	`int SD_Wave::getStatus();`
Python	`SD_Wave.getStatus()`
HVI	Not available.

5.6.4: getType

Description Returns the type of loaded waveform. Refer to “AWG Waveform Types” on page 28.

Parameters

Table 305 Input parameters for getType

Input Parameter name	Description
waveformID	Waveform identifier (returned by new).

Table 306 Output parameters for getType

Output Parameter name	Description
waveformType	(Non-object-oriented languages only) Waveform type. Negative numbers indicate an error. See Description of SD1 Error IDs.
errorOut	See Description of SD1 Error IDs.

Syntax

Table 307 API syntax for getType

Language	Syntax
C	`int SD_Wave_getType(int waveformID);`
C++	`int SD_Wave::getType();`
.NET	`int SD_Wave::getType();`
Python	`SD_Wave.getType()`
HVI	Not available

Section 5.7: SD_Module functions (specific to Pathwave FPGA)

5.7.1: FPGAgetSandBoxRegister

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

Parameters

Table 308 Input parameters for FPGAgetSandBoxRegister

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

Table 309 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 310 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

5.7.2: FPGAgetSandBoxRegisters

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

Parameters

Table 311 Input parameters for FPGAgetSandBoxRegisters

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

Table 312 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 313 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.

5.7.3: FPGAload

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

Parameters

Table 314 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 315 Output parameters for FPGAload

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

Syntax

Table 316 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.

5.7.4: FPGAreset

Description Sends a reset signal to the FPGA sandbox region.

Parameters

Table 317 Input parameters for FPGAreset

Input Parameter name	Description
moduleID	Module identifier, returned by open.
mode	SD_ResetMode values. Refer to Table 41 SD_ResetMode attribute values.

Table 318 Output parameters for FPGAreset

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

Syntax

Table 319 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

5.7.5: FPGATriggerConfig

Description

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

Parameters

Table 320 Input parameters for FPGATriggerConfig

Input Parameter name	Description
moduleID	Module identifier, returned by open.
externalSource	AWG external trigger source.
direction	Refer to Table 42 FPGATriggerDirection attribute values.
polarity	Refer to Table 43 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 321 Output parameters for FPGATriggerConfig

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

Syntax

Table 322 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

5.7.6: 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 323 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 324 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

5.7.7: 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 325 HVI Engine syntax for M3xxxA modules**

Language	Engine 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`		

5.7.8: 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 326 HVI Trigger syntax for M3xxxA modules

Language	Trigger Syntax	Values <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 5.8: SD_SandboxRegister functions

5.8.1: readRegisterBuffer

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

Parameters **Table 327 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 328 Output parameters for readRegisterBuffer

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

Syntax **Table 329 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

5.8.2: readRegisterInt32

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

Parameters

Table 330 Input parameters for readRegisterInt32

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

Table 331 Output parameters for readRegisterInt32

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

Syntax

Table 332 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

5.8.3: writeRegisterBuffer

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

Parameters

Table 333 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 334 Output parameters for writeRegisterBuffer

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

Syntax

Table 335 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

5.8.4: writeRegisterInt32

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

Parameters

Table 336 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 337 Output parameters for writeRegisterInt32

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

Syntax

Table 338 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

5.8.5: 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 AWG

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

1 (Optional) Create an AWG object with "SD_AOU functions()".
2 Open an AWG with "open()".
3 Flush waveforms with "waveformFlush()".
4 Select "channelWaveShape()".
5 (Optional) Create a wave object with "SD_Wave functions()".
6 Load an AWG waveform with "waveformLoad()".
7 Queue a waveform with "AWGqueueWaveform()".
8 Start a waveform with "AWGstart()".
9 Trigger a waveform with "AWGtrigger()".

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

The following sample programs, using Python, give you an understanding of the work flow for one or more use cases. While the work flow is the same for other programming languages, the syntax vary based on the programming language being used.

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

```
This sample program shows how Python uses the Keysight SD1 Programming Libraries.
# ----------
# Import required system components
import sys
sys.path.append('C:\Program Files (x86)\Keysight\SD1\Libraries\Python')

import keysightSD1

waveform_data_list = [0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1, 0.11,
                      0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19, 0.2, 0.21, 0.22,
                      0.23, 0.24, 0.25, 0.26, 0.27, 0.28, 0.29, 0.3, 0.31, 0.32, 0.33,
                      0.34, 0.35, 0.36, 0.37, 0.38, 0.39, 0.4, 0.41, 0.42, 0.43, 0.44,
                      0.45, 0.46, 0.47, 0.48, 0.49, 0.5, 0.51, 0.52, 0.53, 0.54, 0.55,
                      0.56, 0.57, 0.58, 0.59, 0.6, 0.61, 0.62, 0.63, 0.64, 0.65, 0.66,
                      0.67, 0.68, 0.69, 0.7, 0.71, 0.72, 0.73, 0.74, 0.75, 0.76, 0.77,
                      0.78, 0.79, 0.8, 0.81, 0.82, 0.83, 0.84, 0.85, 0.86, 0.87, 0.88,
                      0.89, 0.9, 0.91, 0.92, 0.93, 0.94, 0.95, 0.96, 0.97, 0.98, 0.99,
                      1]
# MODULE CONSTANTS
PRODUCT = ""
CHASSIS = 1
# change slot number to your value
SLOT = 5
CHANNEL = 1
AMPLITUDE = 1.0
# CREATE AND OPEN MODULE
module = keysightSD1.SD_AOU()
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()
    module.channelWaveShape(CHANNEL, keysightSD1.SD_Waveshapes.AOU_AWG)
    module.channelAmplitude(CHANNEL, AMPLITUDE)
    # WAVEFORM FROM FILE
    WAVE_NUMBER = 0
    # create, open from file, load to module RAM and queue for execution
    wave = keysightSD1.SD_Wave()
    # set pat to your file here
    wave.newFromFile("C:/Users/Public/Documents/keysightSD1/Examples/Waveforms/Gaussian.csv")
    module.waveformLoad(wave, WAVE_NUMBER)
    module.AWGqueueWaveform(CHANNEL, WAVE_NUMBER, 0, 0, 0, 0)
    error = module.AWGstart(CHANNEL)
    if error < 0:
        print("AWG from file error:", error)
    else:
        print("AWG from file started successfully")
    input("Press any key to stop AWG")
    module.AWGflush(CHANNEL)
    module.AWGstop(CHANNEL)
    input("Press any key to start AWG from array")
    # WAVEFORM FROM ARRAY/LIST
    # This function is equivalent to create a waveform with new,
    # and then to call waveformLoad, AWGqueueWaveform and AWGstart
    error = module.AWGfromArray(CHANNEL, 0, 0, 0, 0, 0, waveform_data_list)
    if error < 0:
        print("AWG from array error:", error)
    else:
        print("AWG from array started successfully")
    # exiting...
    input("Press any key to stop AWG")
    module.AWGstop(CHANNEL)
    module.close()
    print()
    print("AOU 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 Sine Wave generation using Python

This example program shows a Python script using the Keysight SD1 Programming Libraries to produce a Sine Wave out of Keysight M3202A PXIe AWG's Channel 1 at 1 MHz and 0.1 V.

Optional example routines follow this code that demonstrate how to change the amplitude and frequency specified by values from an array; these optional routines require the time library to be imported.

```
# ----------
# Python - Sample Application to set up the AWG to output a sine wave out of channel 1,
# at 1 MHz, and at 0.1 Vpp.
# ----------
# Import required system components
import sys
# ----------
# Append the system path to include the location of Keysight SD1 Programming Libraries
# then import the library
sys.path.append('C:\Program Files (x86)\Keysight\SD1\Libraries\Python')
import keysightSD1          # Import Python SD1 library and AWG/Digitizer commands.
# ----------
# Specify values for variables
product = 'M3202A'          # Product's model number
chassis = 1                 # Chassis number holding product
slot = 4                    # Slot number of product in chassis
channel = 1                 # Channel being used
amplitude = 0.1             # (Unit: Vpp) Amplitude of AWG output signal (0.1 Vpp)
frequency = 1e6             # (Unit: Hz ) Frequency of AWG output signal (1 MHz)
waveshape = keysightSD1.SD_Waveshapes.AOU_SINUSOIDAL      # Specify sine wave
# ----------
```

```
# Select settings and use specified variables
awg = keysightSD1.SD_AOU()                # Creates SD_AOU object called awg
awg.openWithSlot(product, chassis, slot)      # Connects awg object to module
awg.channelAmplitude(channel, amplitude)      # Sets output amplitude for awg
awg.channelFrequency(channel, frequency)      # Sets output frequency for awg
awg.channelWaveShape(channel, waveshape)    # Sets output signal type for awg
# ----------
# Start playing the sine wave on the specified channel
awg.AWGstart(channel)
# ----------
# (Optional) Sample: To vary the amplitude of the sine wave
import time
amps = [0.1, 0.2, 0.3, 0.4, 0.1]
def varyAmplitude():
    for amplitude in amps:
        time.sleep(3)                   # Add delays before amplitude changes
        awg.channelAmplitude(channel, amplitude)
# ----------
# (Optional) Sample: To vary the frequency of the sine wave
freqs = [1e6, 2e6, 3e6, 4e6, 1e6]
def varyFrequency():
    for frequency in freqs:
        time.sleep(3)                  # Add delays before  frequency changes
        awg.channelFrequency(channel, frequency)
varyAmplitude()
varyFrequency()
# ----------
# Close the connection between the AWG object and the physical AWG hardware
awg.close()
# ----------
# © 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 Sawtooth Wave generation from an Array

This example program shows a Python script using the Keysight SD1 Programming Libraries to produce a Sawtooth Wave out of Keysight M3202A PXIe AWG's Channel 1.

```
# ----------
# Python - Sample Application to set up the AWG to output an array that was created with numpy.
# ----------
# Import required system components
import sys
# ----------
# Append the system path to include the
# location of Keysight SD1 Programming Libraries then import the library
sys.path.append('C:\Program Files (x86)\Keysight\SD1\Libraries\Python')
import keysightSD1          # Import Python SD1 library and AWG/Digitizer commands.
import numpy                # Import numpy, which is used to make an array.
# ----------
# Specify values for variables
product = 'M3202A' # Product's model number
chassis = 1                 # Chassis number holding product
slot = 4                    # Slot number of product in chassis
channel = 1                 # Channel being used
amplitude = 0.1             # (Unit: Vpp) Amplitude of AWG output signal (0.1 Vpp)
waveshape = keysightSD1.SD_Waveshapes.AOU_AWG            # Specify AWG output
delay = 0                   # (Unit: ns) Delay after trigger before generating output.
cycles = 0                  # Number of cycles. Zero specifies infinite cycles.
                            # Otherwise, a new trigger is required to actuate each cycle.
prescaler = 0               # Integer division reduces high freq signals to lower frequency.

# ----------
# Select settings and use specified variables
awg = keysightSD1.SD_AOU()                 # Creates SD_AOU object called awg
awg.openWithSlot(product, chassis, slot)      # Connects awg object to module
awg.channelAmplitude(channel, amplitude)      # Sets output amplitude for awg
awg.channelWaveShape(channel, waveshape)    # Sets output signal type for awg
awg.waveformFlush()                                      # Cleans the queue
awg.AWGflush(channel)          # Stops signal from outputing out of channel 1
# Create an array that represents a sawtooth signal using "numpy"
array = numpy.zeros(1000)          # Create array of zeros with 1000 elements
```

```
array[0] = -0.5                                 # Initialize element 0 as -0.5
for i in range(1, len(array)):      # This for..loop will increment from -0.5
    array[i] = array[i-1] + .001          # Increment by .001 every iteration
wave = keysightSD1.SD_Wave()       # Create SD_Wave object and call it "wave"
                                       # (will place the array inside "wave")
error = wave.newFromArrayDouble(keysightSD1.SD_WaveformTypes.WAVE_ANALOG, array.tolist())
# Place the array into the "wave" object
waveID = 0       # This number is arbitrary and used to identify the waveform.
awg.waveformLoad(wave, waveID)    # Load the "wave" object and give it an ID.
awg.AWGqueueWaveform(channel, waveID, keysightSD1.SD_TriggerModes.SWHVITRIG, delay, cycles,
prescaler)
                                  # Queue waveform to prepare it to be output
# ----------
awg.AWGstart(channel)                                          # Start the AWG
awg.AWGtrigger(channel)                              # Trigger the AWG to begin
# ----------
# Close the connection between the AWG object and the physical AWG hardware.
awg.close()
# ----------
# © 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 using AWG Partner Channel as Differential

This example program shows a Python script using the Keysight SD1 Programming Libraries to generate a differential signal on the Partner Channel.

```
# ----------
# Python - Sample Application to set up the AWG to output an array that was created with numpy.
# ----------
# Import required system components
import sys
# ----------
# Append the system path to include the location of Keysight SD1 Programming Libraries
# then import the library
sys.path.append('C:\Program Files (x86)\Keysight\SD1\Libraries\Python')
import keysightSD1 as sdlib # Import Python SD1 library and AWG/Digitizer commands.
# ----------
# Specify values for variables
product = 'M3201A' # Product's model number.
chassis = 1
slot = 14
# Opening the M3202A module
awg = sdlib.SD_AOU()
awgId = awg.openWithSlotCompatibility(product, chassis, slot, 1)
print(awgId)
# Perform Waveform flush
error = awg.waveformFlush()
print(error)
# Setup channels for differential partner mode
error=awg.channelWaveShape(1, sdlib.SD_Waveshapes.AOU_AWG)
print(error)
error=awg.channelWaveShape(2, sdlib.SD_Waveshapes.AOU_PARTNER)
print(error)
error=awg.channelWaveShape(3, sdlib.SD_Waveshapes.AOU_AWG)
print(error)
error=awg.channelWaveShape(4, sdlib.SD_Waveshapes.AOU_PARTNER)
print(error)
# Set amplitude on the Differential Channel pairs
error=awg.channelAmplitude(1, 1)
print(error)
error=awg.channelAmplitude(2, -1)
print(error)
error=awg.channelAmplitude(3, 1)
print(error)
```

```
error=awg.channelAmplitude(4, -1)
print(error)
# Configure Waveform and Trigger Characteristics to play the waveform
zeroDelay = 0
infiniteIterations = 0;
preScaler = 0
wave = sdlib.SD_Wave()
# Loading the AWG waveform type
wfm0 = 0
gaussianFilepath = r"C:\Users\Public\Documents\Keysight\SD1\Examples\Waveforms\Gaussian.csv"
# "\Waveforms\Gaussian.csv"
error=wave.newFromFile(gaussianFilepath)
print(error)
error=awg.waveformLoad(wave, wfm0)
print(error)
# Setting Trigger modes for the Queued AWG Waveforms on each Channel
triggerMode = sdlib.SD_TriggerModes.SWHVITRIG
print("TriggerMode set to SWHVITRIG. ")
# Queuing AWG waveforms on each AWG Channel
error=awg.AWGqueueWaveform(1, wfm0, triggerMode, zeroDelay, infiniteIterations, preScaler)
print(error)
error=awg.AWGqueueWaveform(3, wfm0, triggerMode, zeroDelay, infiniteIterations, preScaler)
print(error)
# Setting mask and perfoming Start operation on multiple AWGs
AWGmask = 5 # Start and trigger channels 1 & 3. Channels 2 & 4 will follow suit.
error=awg.AWGstartMultiple(AWGmask)
print(error)
# Perfoming Trigger operation on multiple AWGs for playing Queued Waveforms
input("Press key to play waveform : ")
error = awg.AWGtriggerMultiple(AWGmask)
print(error)
# Perfoming Stop operation on multiple AWGs
input("Press any key to stop : ")
error=awg.AWGstopMultiple(AWGmask)
print(error)
# Closing module
input("Press any key to close : ")
error = awg.close()
print(error)
# ----------
# © 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.
# ----------
```

6.2.5: Sample Program for using 48-bit HVI registers

This example program shows a Python script using support for 48-bit HVI registers.

```
# ----------
# Import required system components


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

awg = keysightSD1.SD_AOU()
awg.openWithSlot("M3201A", 1, 10)

sys_def = kthvi.SystemDefinition("mySystem")
sys_def.chassis.add_auto_detect()
sys_def.engines.add(awg.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("Sequencer", sys_def)
sequence = sequencer.sync_sequence.add_sync_multi_sequence_block('exec_block',
0).sequences['SdEngine0']

instruction_set = sequence.instruction_set
amp_reg = sequencer.sync_sequence.scopes['SdEngine0'].registers.add("amplitude",
kthvi.RegisterSize.SHORT)
amp_reg.initial_value = awg.voltsToInt(1.0)

# LONG data type used for 48-bit HVI register support
freq_reg = sequencer.sync_sequence.scopes['SdEngine0'].registers.add("freq",
kthvi.RegisterSize.LONG)
freq_reg.initial_value =awg.freqToInt(1000000.0)

phase_reg = sequencer.sync_sequence.scopes['SdEngine0'].registers.add("phase",
kthvi.RegisterSize.LONG)
phase_reg.initial_value =awg.phaseToInt(180.0)

instruction0 = sequence.add_instruction("SetWaveShape", 20,
awg.hvi.instruction_set.set_waveshape.id)
instruction0.set_parameter(awg.hvi.instruction_set.set_waveshape.channel.id, 1)
instruction0.set_parameter(awg.hvi.instruction_set.set_waveshape.value.id, 1)
instruction0 = sequence.add_instruction("SetAmplitude", 20,
awg.hvi.instruction_set.set_amplitude.id)
instruction0.set_parameter(awg.hvi.instruction_set.set_amplitude.channel.id, 1)
```

```
instruction0.set_parameter(awg.hvi.instruction_set.set_amplitude.value.id, amp_reg)

instruction0 = sequence.add_instruction("SetFrequency", 20,
awg.hvi.instruction_set.set_frequency.id)
instruction0.set_parameter(awg.hvi.instruction_set.set_frequency.channel.id, 1)
instruction0.set_parameter(awg.hvi.instruction_set.set_frequency.value.id,freq_reg)

instruction0 = sequence.add_instruction("SetPhase", 20, awg.hvi.instruction_set.set_phase.id)
instruction0.set_parameter(awg.hvi.instruction_set.set_phase.channel.id, 1)
instruction0.set_parameter(awg.hvi.instruction_set.set_phase.value.id,phase_reg)

# 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...")

# ----------
# © 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 SD1 Error IDs

Table 339 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 339 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
SD_ERROR_DRIVER_RESOURCE_BUSY	-8027	Driver resource busy
SD_ERROR_DRIVER_RESOURCE_NOT_READY	-8028	Driver resource not ready

Error Define	Error No.	Error Description
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
SD_ERROR_INVALID_END	-8061	Invalid end
SD_ERROR_INVALID_CYCLES	-8062	Invalid cycles

Error Define	Error No.	Error Description
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.

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 AWG/Combos can be accessed via the **Help** menu of the AWG SFPs for the M320xA 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\M32_M33XX_AWG*.

1 Navigate to folder where the SD1 files are installed.

2 Click the file "*AWG User Guide.htm*", which is highlighted in Figure 83.

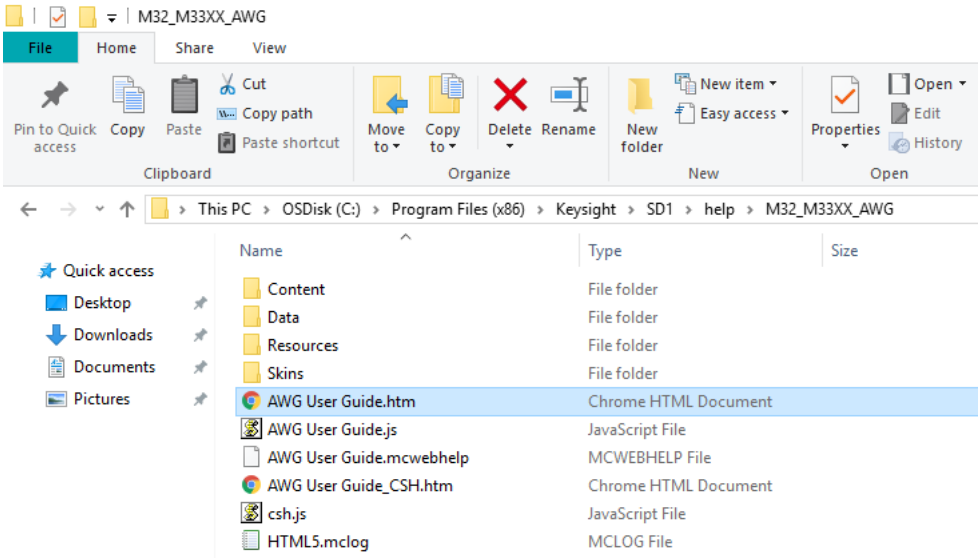


Figure 83 Accessing AWG Online Help on the local disk

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 location
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

KEYSIGHT
TECHNOLOGIES

M3xxx-90003

www.keysight.com