User's Guide

Keysight M3201A/M3202A PXIe Arbitrary Waveform Generators (AWGs)

& M3300A/M3302A AWG & Digitizer Combos

KEYSIGHT TECHNOLOGIES

Notices

Copyright Notice

Manual Part Number

M3201-90001

Published By

Keysight Technologies
1400 Fountaingrove Parkway
Santa Rosa
CA 95403

Edition

Edition 2, March, 2020
Printed In USA

Regulatory Compliance

This product has been designed and tested in accordance with accepted industry standards, and has been supplied in a safe condition. To review the Declaration of Conformity, go to http://www.keysight.com/go/conformity.

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 OF 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 SHALL CONTROL.

KEYSIGHT TECHNOLOGIES DOES NOT WARRANT THIRD-PARTY SYSTEM-LEVEL (COMBINATION OF CHASSIS, CONTROLLERS, MODULES, ETC.) PERFORMANCE, SAFETY, OR REGULATORY COMPLIANCE, UNLESS SPECIFICALLY STATED.

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.

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.

Safety Notices

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.

The following safety precautions should be observed before using this product and any associated instrumentation.

This product is intended for use by qualified personnel who recognize

shock hazards and are familiar with the safety precautions required to avoid possible injury. Read and follow all installation, operation, and maintenance information carefully before using the product.

WARNING

If this product is not used as specified, the protection provided by the equipment could be impaired. This product must be used in a normal condition (in which all means for protection are intact) only.

The types of product users are:

- Responsible body is the individual or group responsible for the use and maintenance of equipment, for ensuring that the equipment is operated within its specifications and operating limits, and for ensuring operators are adequately trained.
- Operators use the product for its intended function. They must be trained in electrical safety procedures and proper use of the instrument. They must be protected from electric shock and contact with hazardous live circuits.
- Maintenance personnel perform routine procedures on the product to keep it operating properly (for example, setting the line voltage or replacing consumable materials). Maintenance procedures are described in the user documentation. The procedures explicitly state if the operator may perform them. Otherwise, they should be performed only by service personnel.
- Service personnel are trained to work on live circuits, perform safe installations, and repair products. Only properly trained service personnel may perform installation and service procedures.

WARNING

Operator is responsible to maintain safe operating conditions. To ensure safe operating conditions, modules should not be operated beyond the full temperature range specified in the Environmental and physical specification. Exceeding safe operating conditions can result in shorter lifespans, improper module performance and user safety issues. When the modules are in use and operation within the specified full temperature range is not maintained, module surface temperatures may exceed safe handling conditions which can cause discomfort or burns if touched. In the event of a module exceeding the full temperature range, always allow the module to cool before touching or removing modules from chassis.

Keysight products are designed for use with electrical signals that are rated Measurement Category I and Measurement Category II, as described in the International Electrotechnical Commission (IEC) Standard IEC 60664. Most measurement, control, and data I/O signals are Measurement Category I and must not be directly connected to mains voltage or to voltage sources with high transient over-voltages. Measurement Category II connections require protection for high transient over-voltages often associated with local AC mains connections. Assume all measurement, control, and data I/O connections are for connection to Category I sources unless otherwise marked or described in the user documentation.

Exercise extreme caution when a shock hazard is present. Lethal voltage may be present on cable connector jacks or test fixtures. The American National Standards Institute (ANSI) states that a shock hazard exists when voltage levels greater than 30V RMS, 42.4V peak, or 60VDC are present. A good safety practice is to expect that hazardous voltage is present in any unknown circuit before measuring.

Operators of this product must be protected from electric shock at all times. The responsible body must ensure that operators are prevented access and/or insulated from every connection point. In some cases, connections must be exposed to potential human contact. Product operators in these circumstances must be trained to protect themselves from the risk of electric shock. If the circuit is capable of operating at or above 1000V, no conductive part of the circuit may be exposed.

Do not connect switching cards directly to unlimited power circuits. They are intended to be used with impedance-limited sources. NEVER connect switching cards directly to AC mains. When connecting sources to switching cards, install protective devices to limit fault current and voltage to the card.

Before operating an instrument, ensure that the line cord is connected to a properly-grounded power receptacle. Inspect the connecting cables, test leads, and jumpers for possible wear, cracks, or breaks before each use.

When installing equipment where access to the main power cord is restricted, such as rack mounting, a separate main input power disconnect device must be provided in close proximity to the equipment and within easy reach of the operator.

For maximum safety, do not touch the product, test cables, or any other instruments while power is applied to the circuit under test. ALWAYS remove power from the entire test system and discharge any capacitors before: connecting or disconnecting cables or jumpers, installing or removing switching cards, or making internal changes, such as installing or removing jumpers.

Do not touch any object that could provide a current path to the common side of the circuit under test or power line (earth) ground. Always make measurements with dry hands while standing on a dry, insulated surface capable of withstanding the voltage being measured.

The instrument and accessories must be used in accordance with its specifications and operating instructions, or the safety of the equipment may be impaired.

Do not exceed the maximum signal levels of the instruments and accessories, as defined in the specifications and operating information, and as shown on the instrument or test fixture panels, or switching card.

When fuses are used in a product, replace with the same type and rating for continued protection against fire hazard.

Chassis connections must only be used as shield connections for measuring circuits, NOT as safety earth ground connections.

If you are using a test fixture, keep the lid closed while power is applied to the device under test. Safe operation requires the use of a lid interlock.

Instrumentation and accessories shall not be connected to humans.

Before performing any maintenance, disconnect the line cord and all test cables.

To maintain protection from electric shock and fire, replacement components in mains circuits – including the power transformer, test leads, and input jacks – must be purchased from Keysight. Standard fuses with applicable national safety approvals may be used if the rating and type are the same. Other components that are not safety-related may be purchased from other suppliers as long as they are equivalent to the original component (note that selected parts should be purchased only through Keysight to maintain accuracy and functionality of the product). If you are unsure about the applicability of a replacement component, call an Keysight office for information.

WARNING

No operator serviceable parts inside. Refer servicing to qualified personnel. To prevent electrical shock do not remove covers. For continued protection against fire hazard, replace fuse with same type and rating.

PRODUCT MARKINGS:

The CE mark is a registered trademark of the European Community.

Australian Communication and Media Authority mark to indicate regulatory compliance as a registered supplier.

ICES/NMB-001
ISM GRP.1 CLASS A

This symbol indicates product compliance with the Canadian Interference-Causing Equipment Standard (ICES-001). It also identifies the product is an Industrial Scientific and Medical Group 1 Class A product (CISPR 11, Clause 4).

South Korean Class A EMC Declaration. This equipment is Class A suitable for professional use and is for use in electromagnetic environments outside of the home. A 급 기기 (업무용 방송통신 기자재) 이 기기는 업무용 (A 급) 전자파적 합기기로서 판 매자 또는 사용자는 이 점을 주 의하시기 바라 며 , 가정외의 지역에서 사 용하는 것을 목적으 로 합니다.

This product complies with the WEEE Directive marketing requirement. The affixed product label (above) indicates that you must not discard this electrical/electronic product in domestic household waste. **Product Category**: With reference to the equipment types in the WEEE directive Annex 1, this product is classified as "Monitoring and Control instrumentation" product. Do not dispose in domestic household waste. To return unwanted products, contact your local Keysight office, or for more information see http://about.keysight.com/en/companyinfo/environment/takeback.shtml.

This symbol indicates the instrument is sensitive to electrostatic discharge (ESD). ESD can damage the highly sensitive components in your instrument. ESD damage is most likely to occur as the module is being installed or when cables are connected or disconnected. Protect the circuits from ESD damage by wearing a grounding strap that provides a high resistance path to ground. Alternatively, ground yourself to discharge any built-up static charge by touching the outer shell of any grounded instrument chassis before touching the port connectors.

This symbol on an instrument means caution, risk of danger. You should refer to the operating instructions located in the user documentation in all cases where the symbol is marked on the instrument.

This symbol indicates the time period during which no hazardous or toxic substance elements are expected to leak or deteriorate during normal use. Forty years is the expected useful life of the product.

Contents

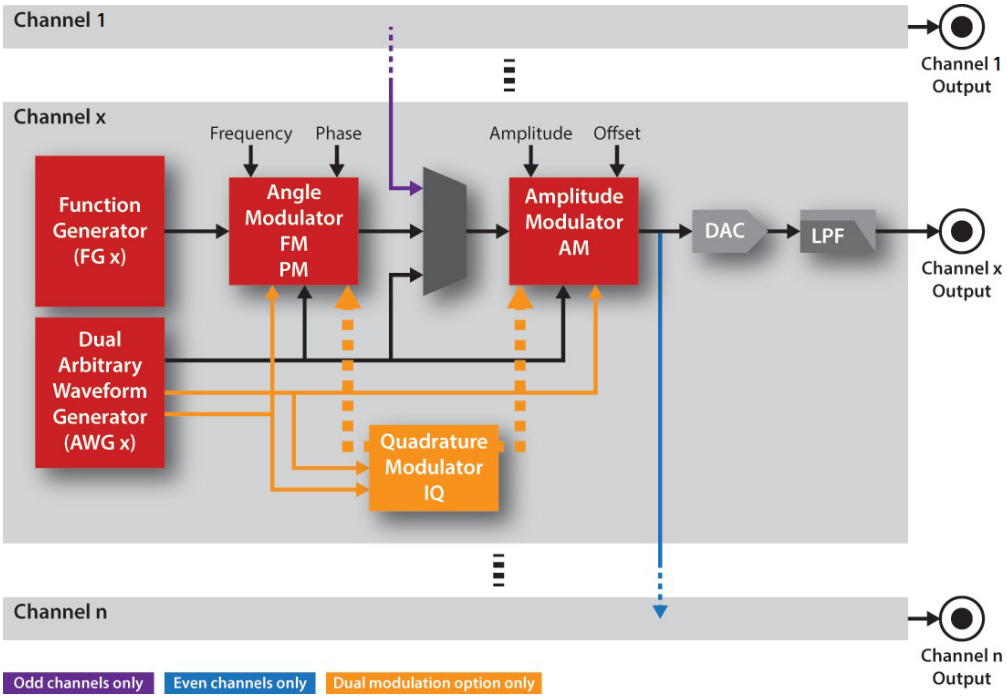
1 Overview of Keysight M320xA PXIe AWGs and Theory

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.

This chapter describes the following topics:

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.



This section describes the following topics:

1. 1. 1 Channel Numbering and Compatibility Mode

Compatibility mode, can be changed by open on page 72, is available to support legacy modules and allows the channel numbering (channel enumeration) to start with either
CH0 or CH1.

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

NOTE **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.

- Signal Generation with the Function Generator on page 7
- Signal Generation with the Arbitrary Waveform Generator on page 8

NOTE **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, frequency, and phase 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.

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 on page 21)	AOU_AWG	6
Partner Channel	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 models

NOTE AOU_PARTNER (8) is used for channels 1 and 3; the signal comes from AWG mode and not the Function Generator.

NOTE To produce differential I/Q signals for channels 1 and 3:

1. Set waveShape to AOU_PARTNER (8).
2. Set the amplitude in Channel 1 to (Amplitude 1).
   For example, Amplitude 1 could be set to 1 V.
3. Set the amplitude in Channel 3 to (Amplitude 3).
   For example, Amplitude 3 could be set to 1 V.

NOTE

To produce differential I/Q signals for channels 2 and 4:

1. Set waveShape to AOU_AWG (6)
2. Set the amplitude in Channel 2 to (–Amplitude 1).
3. Set the amplitude in Channel 4 to (–Amplitude 3).

Channel 2 amplitude must be equal to and opposite to Channel 1 amplitude.

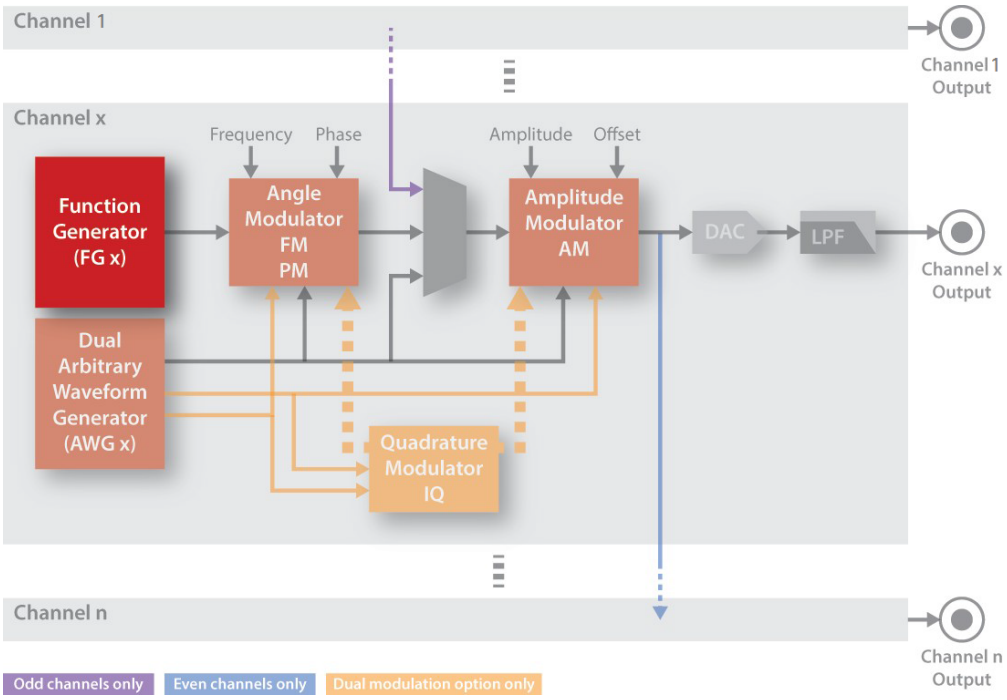
Channel 4 amplitude must be equal to and opposite to Channel 3 amplitude.

Programming Information

Function Name	Comments	Details
channelWaveShape	Sets the channel waveshape type	channelWaveShape on page 82

1. 1. 2. 1 Signal Generation with the Function Generator

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

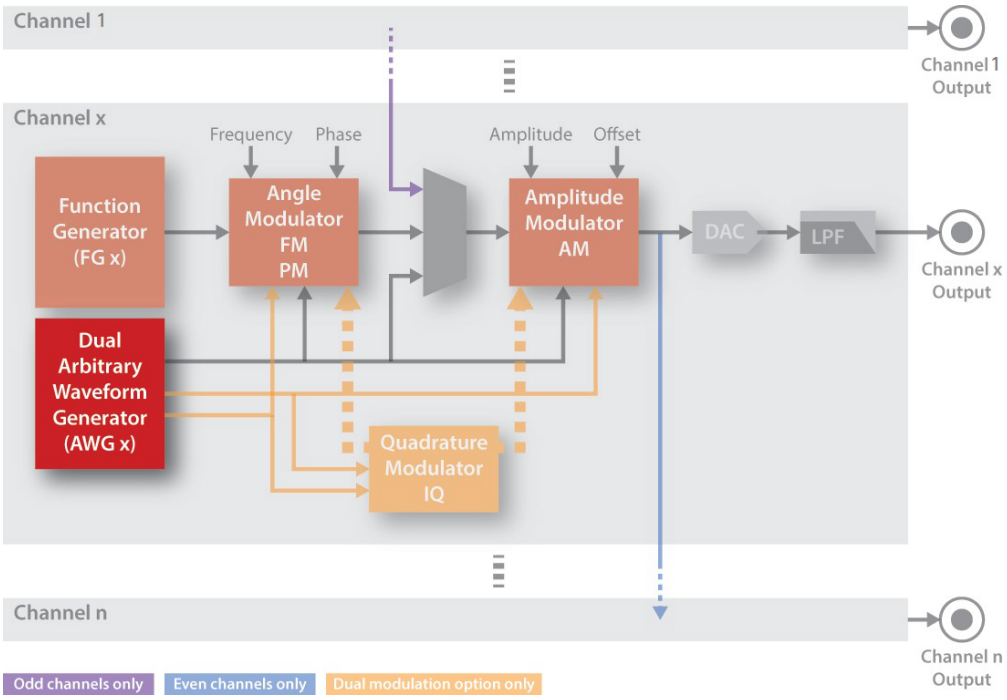


NOTE **Waveform harmonics:** Non-sinusoidal wave shapes (triangular, square, etc.) have high frequency components that may fall outside the bandwidth of the reconstruction filter if the fundamental frequency is too high. In this situation, the output analog signal may suffer some distortion due to the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

1. 1. 2. 2 Signal Generation with the Arbitrary Waveform Generator

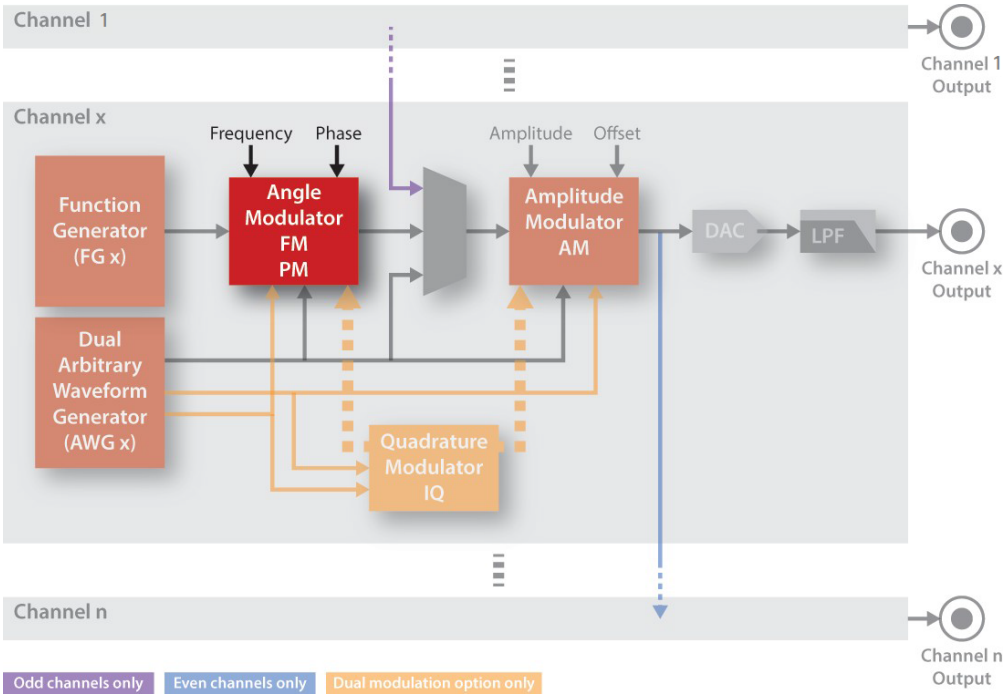
Each channel has an Arbitrary Waveform Generator (AWG) block that generates arbitrary waveforms that can be sent directly to each output channel or they can be used as a modulating signal for the frequency, phase, amplitude, or IQ modulators.

See Working with AWG Waveforms on page 11.



1. 1. 3 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 22.



NOTE **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 on page 86. In addition, in HVI operation (when using the Keysight M3601A Hard Virtual Instrument (HVI) Design Environment Software on page 38) the execution time is deterministic, which allows the user to calculate the new phase and adjust it after any frequency change.

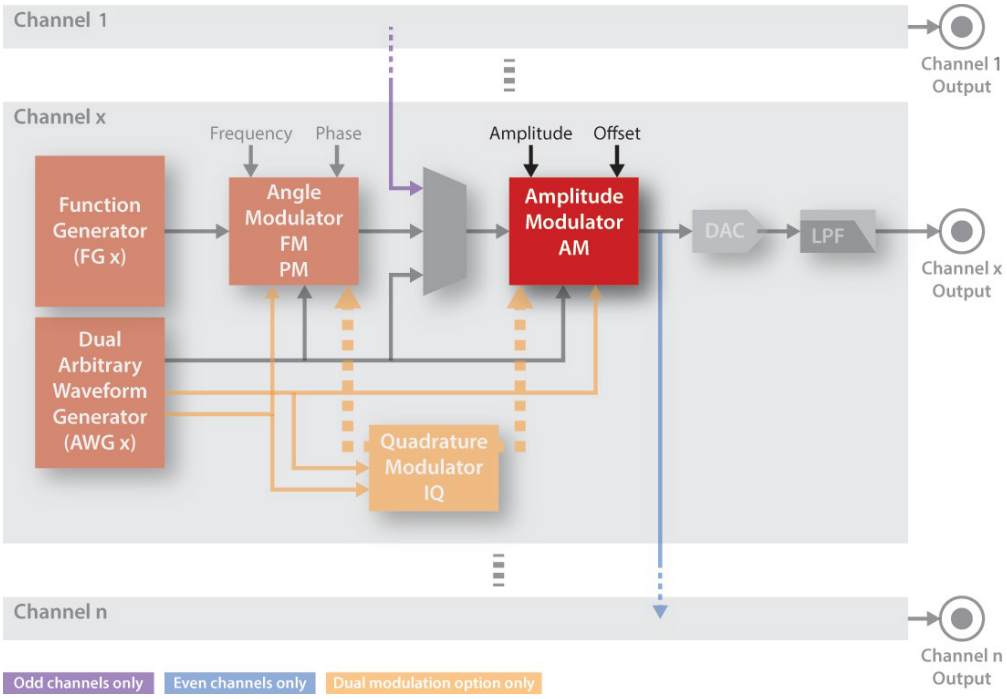
Programming Information

Function Name	Comments	Details
channelFrequency	Sets the frequency of the FG	channelFrequency on page 84
channelPhase	Sets the phase of the FG	channelPhase on page 85
channelPhaseReset	Resets the accumulated phase	channelPhaseReset on page 86

1. 1. 4 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 Frequency and Phase Modulation (Angle Modulator Block) on page 22.

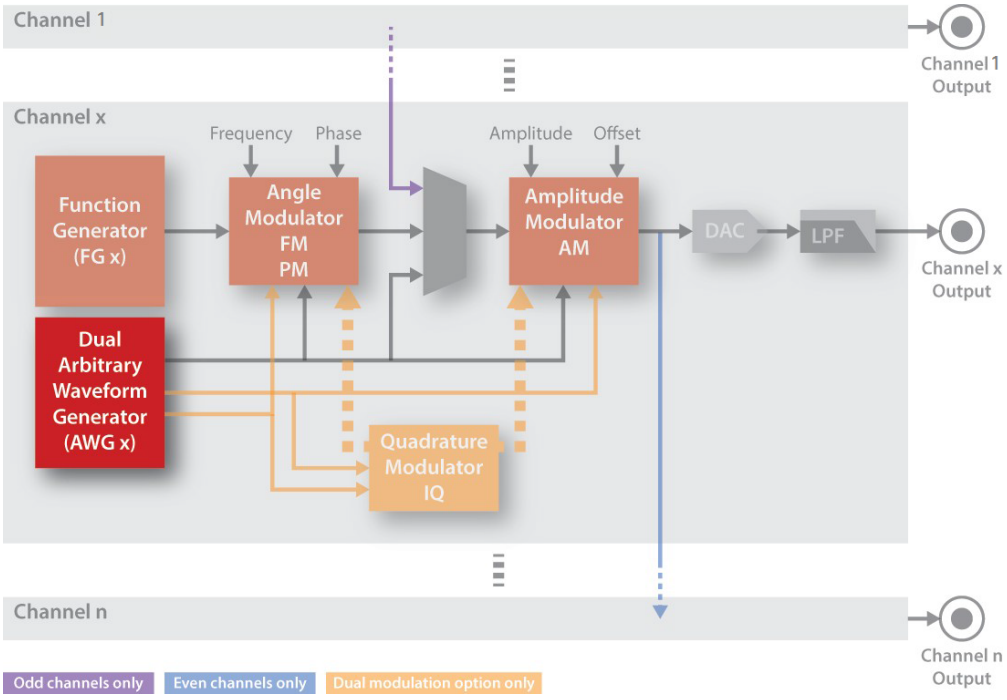


Programming Information

Function Name	Comments	Details
channelAmplitude	Sets the output amplitude	channelAmplitude on page 88
channelOffset	Sets the output DC offset	channelOffset on page 89

1. 2 Working with AWG Waveforms

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



This section describes the following topics:

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

AWG on page 101 provides a one-step solution to load, queue, and run a single waveform directly from a file or from an array in the PC.

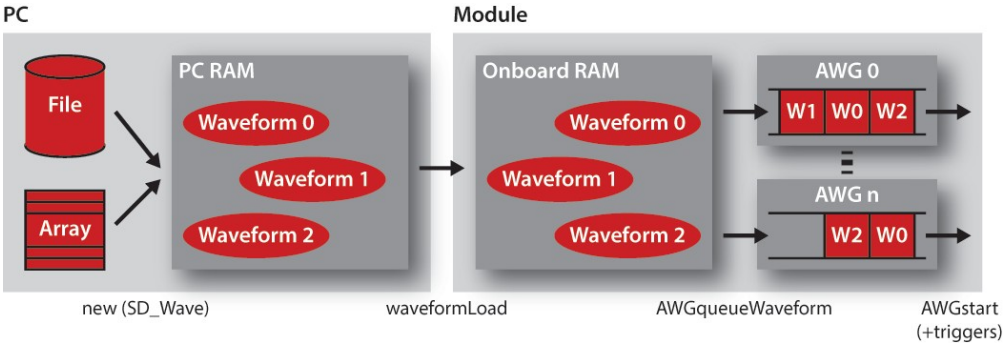
This function simplifies the generation of a single waveform, but it does not allow control over the following aspects of waveform generation:

- 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 on page 140; 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 on page 96.
3. **Queue waveforms in an AWG** with AWGqueueWaveform on page 104 to create the desired generation sequence.
4. **Select the sync mode of the queue** with AWGqueueSyncMode on page 139.
5. **Start the generation** with AWGstart on page 107 and provide triggers if required.
   See AWG Waveform Queue System on page 13.



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, etc.) 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 right 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: 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 on page 135.
  - For Cyclic mode, the minimum play time per cycle should be 1 µs for M3201A/M330xA and 2 µs 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 15.

1. 2. 2. 1 AWG Waveform Queue System Examples

Queue One shot

Waveform	Trigger	Delay	Cycles	Prescaler
A	Ext	0	1	0
B	Ext	0	2	0
C	SW/HVI	0	1	0
D	Auto	100	1	0

Queue One shot

Waveform	Trigger	Delay	Cycles	Prescaler
A	Ext	0	1	0
B	Ext (cycle)	0	2	0
C	SW/HVI	100	1	0
D	Auto	0	1	0

Queue One shot

Waveform	Trigger	Delay	Cycles	Prescaler
A	Ext	0	1	0
B	Auto	0	1	0
C	SW/HVI	0	1	1
D	Ext	0	0	0

Queue Cyclic

Waveform	Trigger	Delay	Cycles	Prescaler
A	Auto	0	1	0
B	Ext (cycle)	0	2	0
C	SW/HVI	100	1	0
D	Auto	0	1	0

Inter-waveform/inter-cycles discontinuities: The AWG Queue System allows the user 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.

NOTE **HVI Generation Sequences:** The queue system of the AWGs provide a way to create generation sequences. For more complex sequences, the best solution is the use of Keysight's Hard Virtual Instruments (HVIs), which are time deterministic sequences with nanosecond resolution, providing a hard real-time execution.

HVIs are programmed with Keysight M3601A Hard Virtual Instrument (HVI) Design Environment Software on page 38, a user-friendly flowchart-style design environment.

1. 2. 3 AWG Prescaler and Sampling Rate

The user 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 as follows:

M3201A

0 => fs=500 MS/s

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

M3202A

0 => fs=1 GS/s

1 => fs=200 MS/s

>1 => fs=100/n MS/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.
- 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.

NOTE **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 and therefore it generates aliasing inside the reconstruction filter bandwidth. For applications were full Upsampling is required, the user 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.

Option	Description	Name	Value
Auto	The waveform is launched automatically after AWGstart on page 107, or when the previous waveform in the queue finishes	AUTOTRIG	0
Software / HVI	Software trigger. The AWG is triggered by the AWGtrigger on page 123, provided that the AWG is running. AWGtrigger can be executed from the user application (VI) or from an HVI. (See HVI in Keysight M3601A Hard Virtual Instrument (HVI) Design Environment Software on page 38.)	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

1 VIHVITRIG is equivalent, but is considered obsolete

Programming Information

Function Name	Comments	Details
AWG	Provides a one-step method to load, queue, and start a single waveform	AWG on page 101
AWGqueueWaveform	Queues a waveform in the specified AWG	AWGqueueWaveform on page 104

If the queued waveforms are going to use any of the External Trigger modes, the source of this trigger must be configured using AWGtriggerExternalConfig on page 121. The available external trigger options.

NOTE **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, etc.) that need to be configured for proper operation.

1. 2. 5 AWG External Trigger Source

Option	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

Option	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 (AWGqueueMarkerConfig on page 137) in one of the module's AWGs.

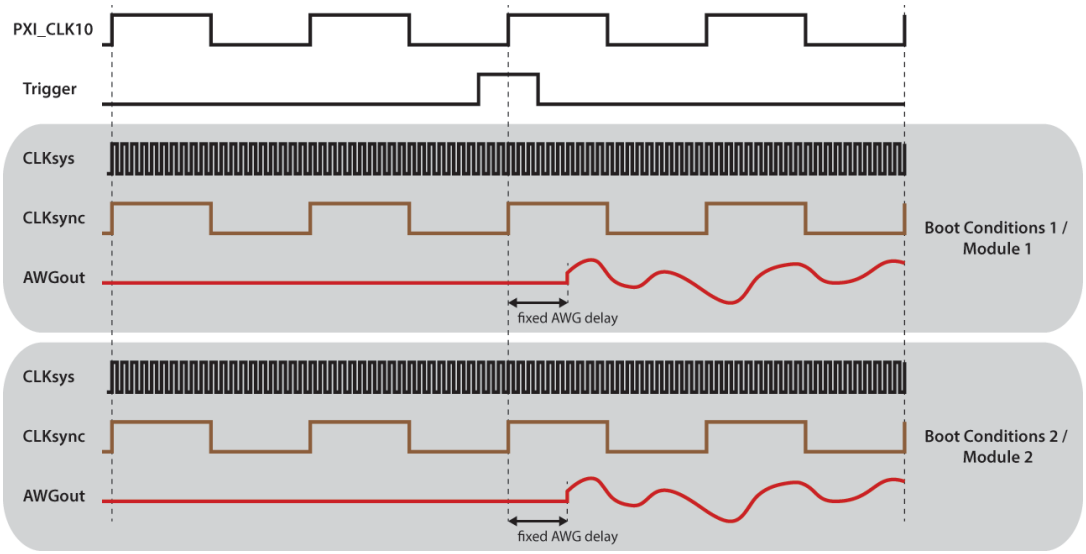
1. 2. 8 AWG FlexCLK Synchronization (models with variable sampling rate)

FlexCLK Technology (models w/ variable sampling rate) on page 33 shows the internal diagram and the operation of the M3201A/M3202A PXIe AWGFlexCLK system. 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 Technology (models w/ variable sampling rate) on page 33, resulting in the following scenarios:

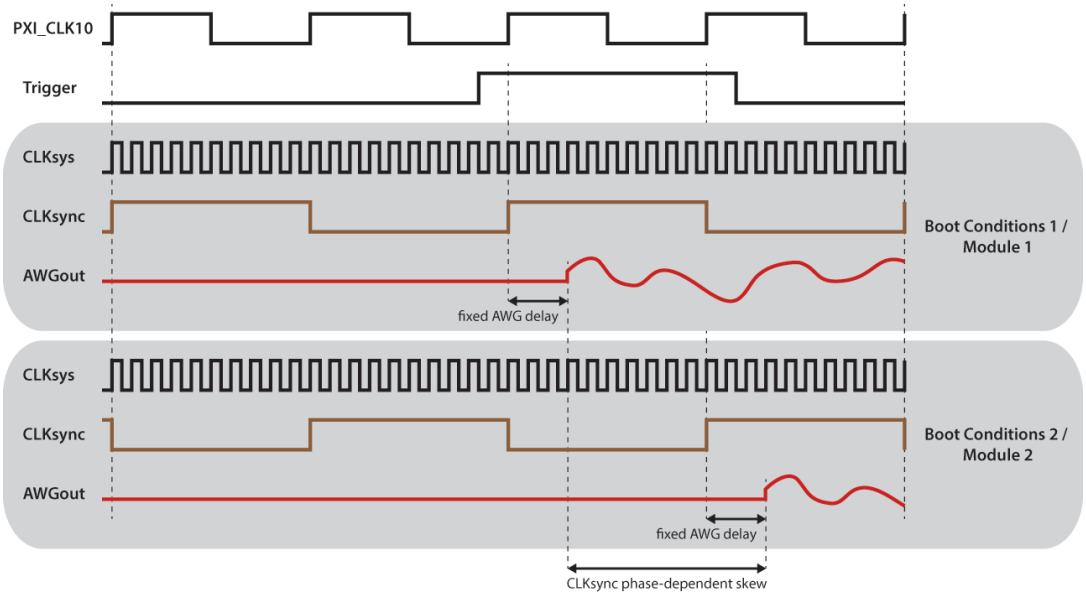
$f_{CLKsync} = f_{PXI\ CLK10}$

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.

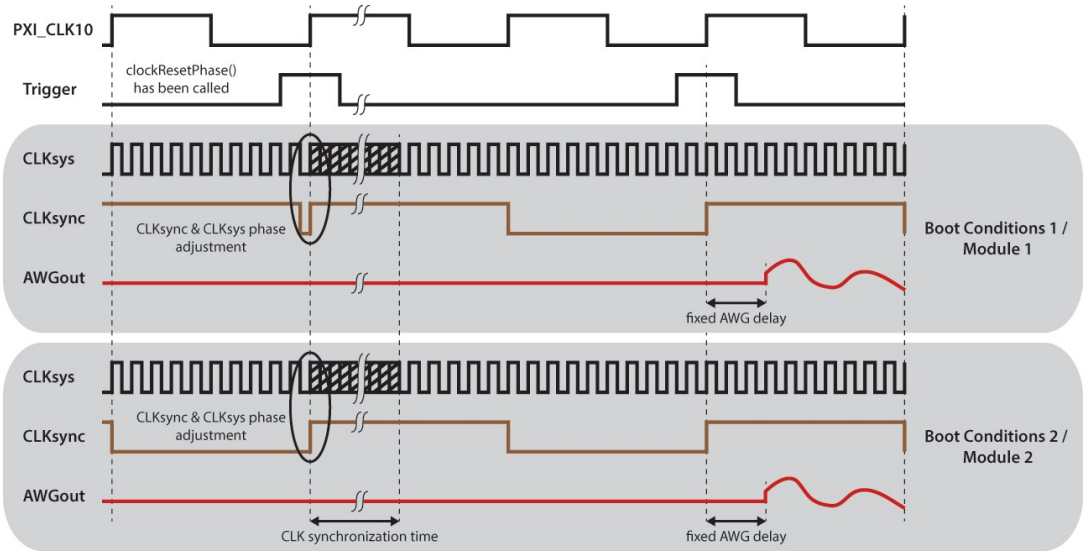


$f_{CLKsync!} = f_{P\ XI\ CLK10}$

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.



This phase uncertainty can be easily corrected using clockResetPhase on page 133. 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.



1. 2. 9 AWG Waveform Array and *.cvs 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

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 new on page 140.

Modulation Option	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 (Module and Phase)	WAVE_IQPOLAR	3
Digital	Digital waveforms defined with integers	WAVE_DIGITAL	5

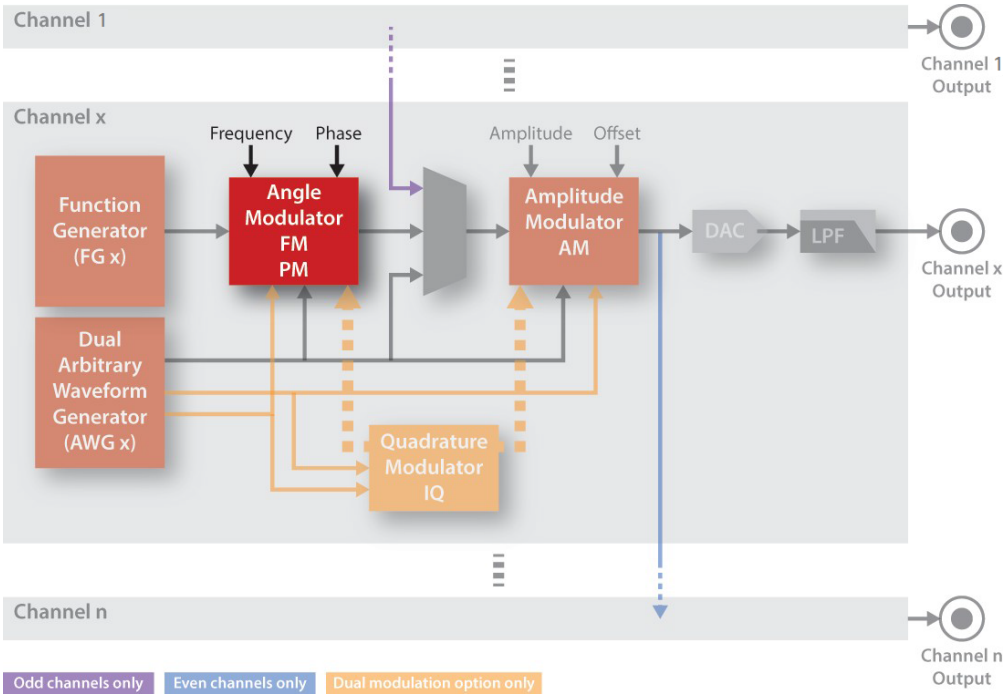
1. 3 Working with Signal Modulation

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

- Frequency and Phase Modulation (Angle Modulator Block) on page 22
- AM and DC Offset (Amplitude Modulator Block) on page 25
- IQ Modulation (Quadrature Modulator Block) on page 28

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.



Option	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 the user to create any analog/digital frequency or phase modulation, for example: FM, FSK, PM, PSK, DPSK, etc.

1. 3. 1. 1 The output signal of a channel using the frequency modulation is the following:

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

where:

- A is the channel amplitude (set by channelAmplitude on page 88)
- G is the deviation gain or peak frequency deviation (set by modulationAngleConfig on page 90). G is only used for 16-bit waveforms.
- AW G(t) is the normalized modulating signal generated by the AWGs
- $\cos(2\pi f_C t+\phi_C)$ 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 (G = h*maxf req[AW G(t)]).

1. 3. 1. 2 The output signal of a channel using the phase modulation is the following:

$$Output(t) = A \cdot \cos(2\pi f_c t + \phi_c + G \cdot AWG(t)) \quad (3)$$

where:

- A is the channel amplitude (set by channelAmplitude on page 88)
- G is the deviation gain or peak frequency deviation (set by modulationAngleConfig on page 90). G is only used for 16-bit waveforms.
- AW G(t) is the normalized modulating signal generated by the AWGs
- $\cos(2\pi f_C t+\phi_C)$ 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 1800, the settings must be A=0.8 and G=180 (G=h).

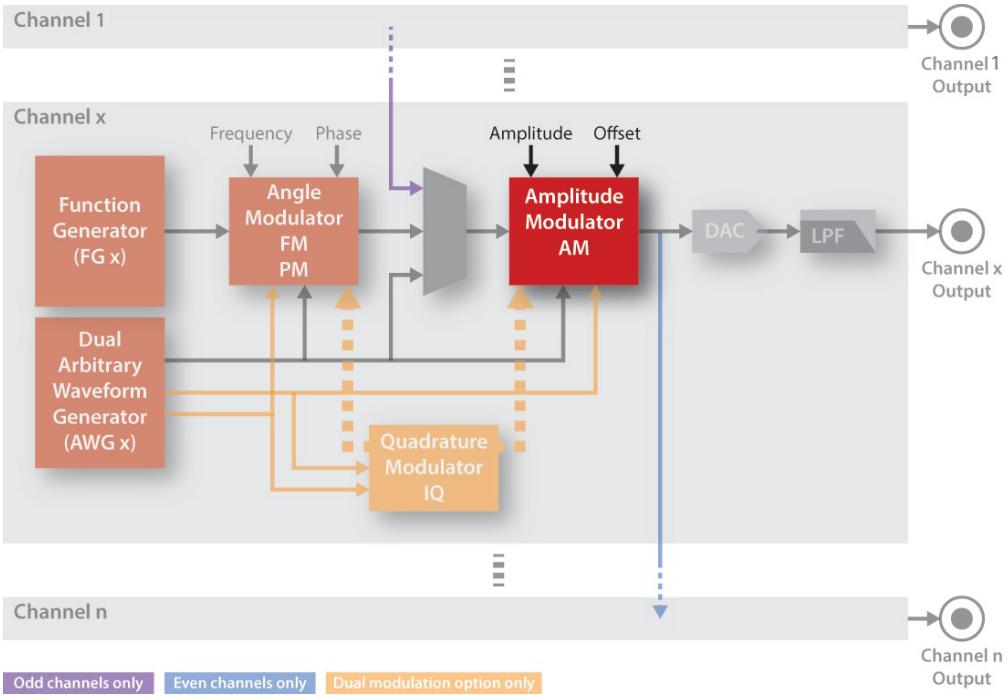
Programming Information

Function	Comments	Details
modulationAngleConfig	Configures the angle modulator	modulationAngleConfig on page 90
AWG functions	Control the modulating signal	Signal Generation with the Arbitrary Waveform Generator on page 8
FG functions	Control the carrier signal	Working with Signal Generation/Channel Structure on page 3

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.

NOTE **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.



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, etc.).

1. 3. 2. 1 The output signal of a channel using the Amplitude Modulator is the following:

where:

$$Output(t) = (A + G \cdot AWG(t)) \cdot \cos(2\pi f_c t + \phi_c) \quad (4)$$

- A is the channel amplitude (set by channelAmplitude on page 88).
- G is the deviation gain (set by modulationAmplitudeConfig on page 92). G is only used for 16-bit waveforms.
- AWG(t) is the normalized modulating signal generated by the AWG.
- $\cos(2\pi f_c t+\phi_c)$ 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 ($G = h \cdot A^2$).

1. 3. 2. 2 The output signal of a channel using the amplitude modulator to modulate the offset is the following:

$$Output(t) = A \cdot \cos(2\pi f_c t + \phi_c) + G \cdot AWG(t) \quad (5)$$

Options

NOTE **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 28, which uses the internal amplitude and the angle modulator).

Function	modulationType (const & value)		Description
No Modulation	AOU MOD OFF	0 (default)	Modulation is disabled. The channel amplitude and offset are only set by the main registers.
Amplitude Modulation	AOU MOD AM	1	The modulating signal is used to modulate the channel amplitude.
Offset Modulation	AOU MOD OFFSET	2	The modulating signal is used to modulate the channel offset.

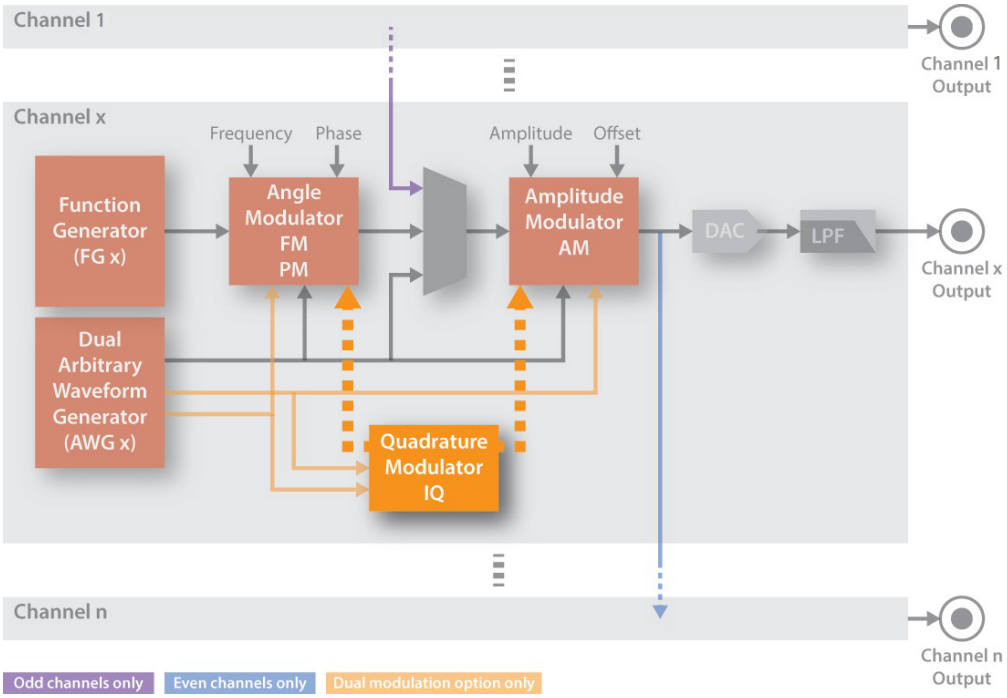
Programming Information

Function	Comments	Details
modulationAmplitudeConfig	Configures the amplitude modulator	modulationAmplitudeConfig on page 92
AWG functions	Controls the modulating signal	Signal Generation with the Arbitrary Waveform Generator on page 8
FG functions	Controls the carrier signal	Working with Signal Generation/Channel Structure on page 3

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.



The modulating signals are generated using the IQ Modulation (Quadrature Modulator Block) on page 28 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 18).

1. 3. 3. 1 The output signal of a channel using the IQ modulator is the following:

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

$$Output(t) = \frac{A}{\sqrt{2}} \cdot [AWG_I(t) \cdot \cos(2\pi f_c t + \phi_c) - AWG_Q(t) \cdot \sin(2\pi f_c t + \phi_c)] \quad (6)$$

where:

- A is the channel amplitude (set by channelAmplitude on page 88).
- $AWG_I(t)$ and $AWG_Q(t)$ are the normalized modulating signals generated by the AWGs.
- $\cos / \sin(2\pi f_c t + \phi_c)$ is the carrier signal, generated with the Function Generator. It is also possible to use the IQ modulator with amplitude and phase components:

$$Output(t) = A \cdot AWG_A(t) \cdot \cos(2\pi f_c t + \phi_c + \pi/2 \cdot AWG_\phi(t)) \quad (7)$$

where:

- A is the channel amplitude (set by channelAmplitude on page 88).
- $AWG_A(t)$ and $AWG_\phi(t)$ are the modulating signals generated by the AWGs.
- $\cos(2\pi f_c t + \phi_c)$ is the carrier signal, generated with the Function Generators.

Mathematical Background of IQ Modulation

$$A \cdot \cos(2\pi f_c t + \phi_c + \phi) = A \cdot \cos(2\pi f_c t + \phi_c) \cos(\phi) - A \cdot \sin(2\pi f_c t + \phi_c) \sin(\phi) \quad (8)$$

where the I and Q components are defined as follows:

$$I = A \cdot \cos(\phi) \quad (9)$$
$$Q = A \cdot \sin(\phi)$$

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

$$A \cdot \cos(2\pi f_c t + \phi_c + \phi) = I \cdot \cos(2\pi f_c t + \phi_c) - Q \cdot \sin(2\pi f_c t + \phi_c) \quad (10)$$

NOTE **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

Function	Comments	Details
modulationIQconfig	Configures the IQ modulation	modulationIQconfig on page 94
AWG functions	Control the modulating signal	Signal Generation with the Arbitrary Waveform Generator on page 8
FG functions	Control the carrier signal	Signal Generation with the Function Generator on page 7

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.

NOTE **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.**

Option	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

Option	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

1. 5 Working with the 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.

NOTE **Chassis Clock Replacement for High-Precision Applications:** For applications where clock stability and precision is crucial (for example: GPS, experimental physics, etc.), the user can replace the chassis clock with an external reference.

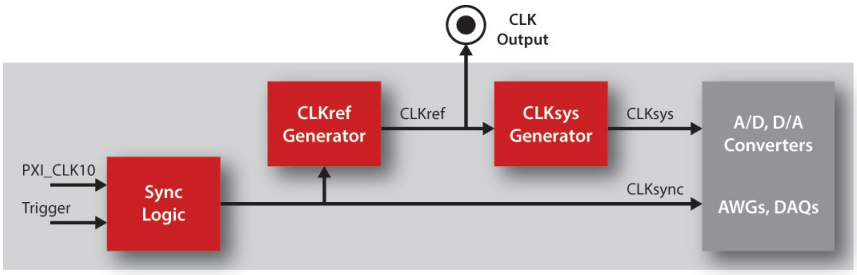
In the case of PXI/PXIe, this is possible via a chassis clock input connector or with a PXI/PXIe timing module. These options are not available in all chassis; see the corresponding chassis specifications.

1. 5. 1 CLK Output Options

Option	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.



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 datasheet of the corresponding product (clockSetFrequency (Requires Option CLV) on page 129). The CLKsync frequency changes with the CLKsys frequency as follows:

$$f_{CLK\,sync} = GreatestCommonDivisor\left(f_{PXI\_CLK10}, \frac{f_{CLK\,sys}}{5}\right) \quad (11)$$

The CLKsync frequency is returned by clockSetFrequency (Requires Option CLV) on page 129.

Option	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

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 Overview of Keysight Software and Programming Tools

This chapter contains an overview of the following software and programming tools:

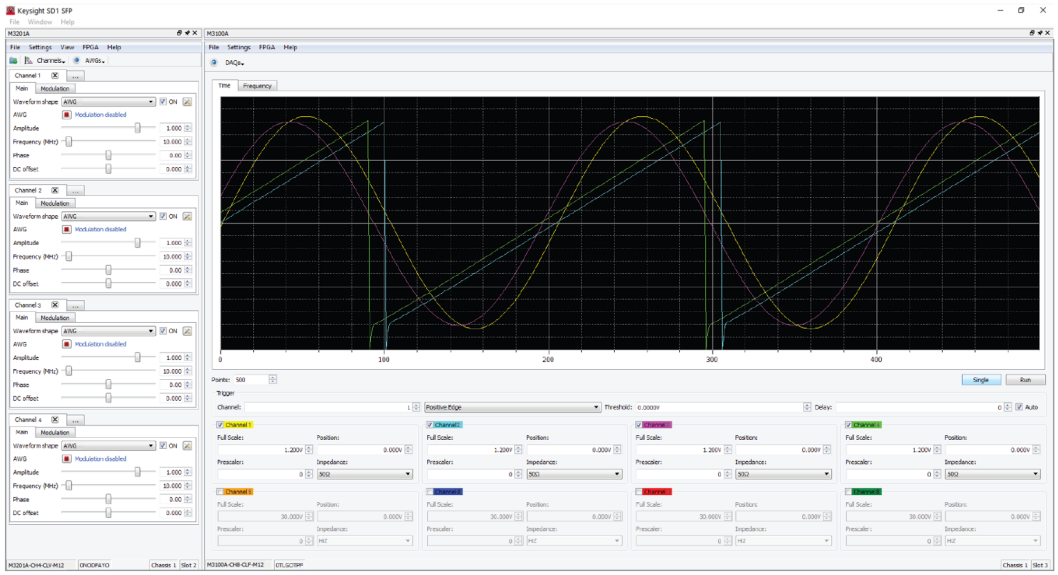
2. 1 Keysight SD1 SFP Software

2. 1. 1 Overview of 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.

When SD1 SFP is opened, 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.

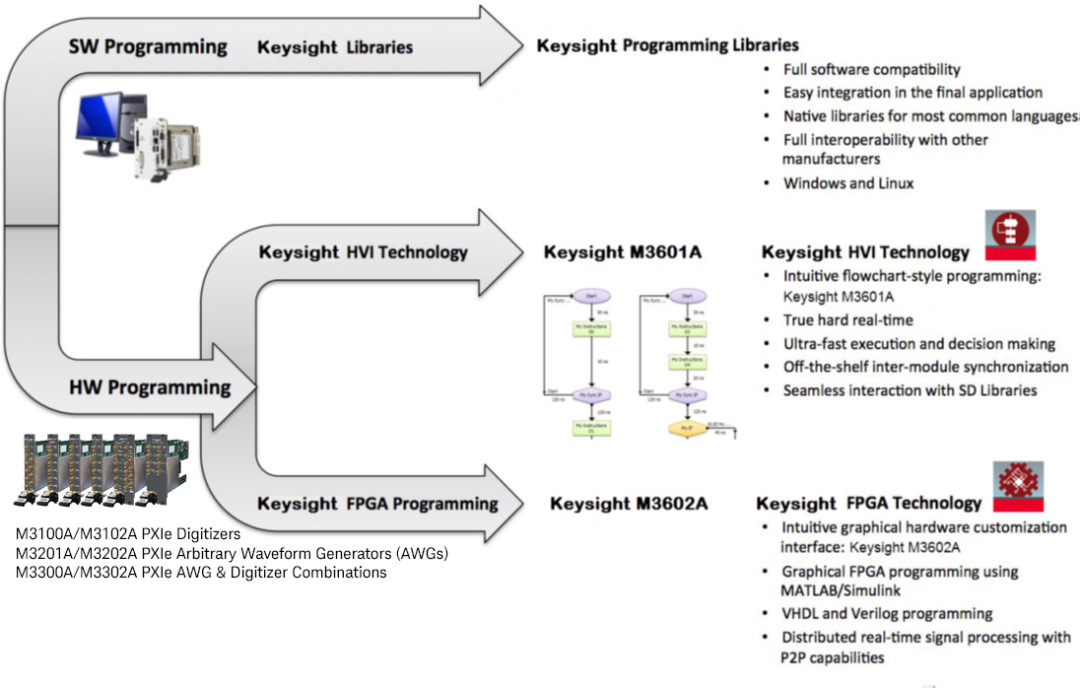
SD1 SFP



2. 2 Keysight Programming Tools

The following programming tools are available to control Keysight M3100A/M3102A PXIe Digitizers, M3201A/M3202A PXIe AWGs, and M3300A/M3302A PXIe AWG & Digitizer Combinations:

- Keysight SD1 Programming Libraries on page 37
- Keysight M3601A Hard Virtual Instrument (HVI) Design Environment Software on page 38
- Keysight M3602A FPGA Design Environment Software on page 43



2. 2. 1 Keysight SD1 Programming Libraries

Keysight supplies a comprehensive set of highly optimized software instructions that can control off-the-shelf functionalities of Keysight hardware. These software instructions are compiled into the Keysight SD1 Programming Libraries. Programs can be written with these libraries and run on an embedded controller or desktop computer.

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.

Keysight provides native programming libraries for a comprehensive set of programming languages, such as C, C++, Visual Studio (VC++, C#, VB), MATLAB, National Instruments LabVIEW, Python, etc., ensuring full software compatibility and seamless multivendor integration. Keysight also provides dynamic libraries, for example: DLLs, that can be used in virtually any programming language.

Keysight native programming libraries ensure full compatibility, providing effortless and seamless software integration user interaction, etc. The I/O modules run in parallel, completely synchronized, and exchange data and decisions in real-time. The result is a set of modules that behave like a single integrated real-time instrument.

For more information, refer to the following sections:

- Keysight Supplied Native Programming Libraries on page 67
- Support for Other Programming Languages on page 68
- Functions in SD1 Programming Libraries on page 69
  - SD_Module Functions on page 72
  - SD_AOU Functions on page 82
  - SD_Wave Functions (new and delete) on page 140
  - SD_Module Functions (M3601A HVI-related) on page 143
  - SD_Module Functions (M3602A FPGA-related) on page 147

2. 2. 2 Keysight M3601A Hard Virtual Instrument (HVI) Design Environment Software

NOTE **Keysight M3201A/M3202A PXIe AWGs and M3100A/M3102A PXIe digitizer must have Option HV1 to use Keysight M3601A software; Option HV1 is only available at time of purchase.**

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.

The following section is only an overview of the Keysight M3601A software; To learn how to use Keysight M3601A software, refer to the User's Guide for the [3] Keysight M3601A Hard Virtual Instrument (HVI) Design Environment Software.

2. 2. 2. 1 HVI Programming

Keysight's HVI technology provides the capability to create time-deterministic execution sequences that are executed by the Keysight M3201A/M3202A PXIe AWGs and M3100A/M3102A PXIe digitizers with Option HV1. HVIs are programmed with Keysight M3601A, an HVI design environment with a user-friendly flowchart-style interface.

2. 2. 2. 2 HVI Functions

Keysight's HVI Technology uses the same programming instructions that are available in the Keysight SD1 Programming Libraries, with the difference that in an HVI, those instructions are executed by the hardware modules in hard real-time, not by the embeded controller or desktop computer.

Virtual Instrumentation is the use of customizable software and modular hardware to create user-defined measurement systems, called Virtual Instruments (VIs). Thus, a Virtual Instrument is based on a software which is executed by a computer, and therefore its real-time performance (speed, latency, etc.) is limited by the computer and by its operating system. In many cases, this real-time performance might not be enough for the application, even with a real-time operating system. In addition, many modern applications require tight triggering and precise intermodule synchronization, making the development of final systems very complex and time consuming. For all these applications, Keysight has developed an exclusive technology called Hard Virtual Instrumentation. In a hard virtual instrument (HVI), the user application is executed by the hardware modules independently of the computer, which stays free for other VI tasks, like visualization.

NOTE **HVIs vs. VIs: Virtual Instrumentation is fully supported making use of the Keysight SD1 Programming Libraries. On the other hand, Keysight's exclusive Hard Virtual Instrumentation (HVI) technology provides the capability to create time-deterministic execution sequences which are executed by the hardware modules in parallel and with perfect intermodule synchronization. HVIs provide the same programming instructions available in the Keysight SD1 Programming Libraries.**

HVIs are programmed with Keysight M3601A Hard Virtual Instrument (HVI) Design Environment Software, with a user-friendly flowchart-style interface, compatible with Keysight M3201A/M3202A PXIe AWGs and M3100A/M3102A PXIe digitizers.

M3601A

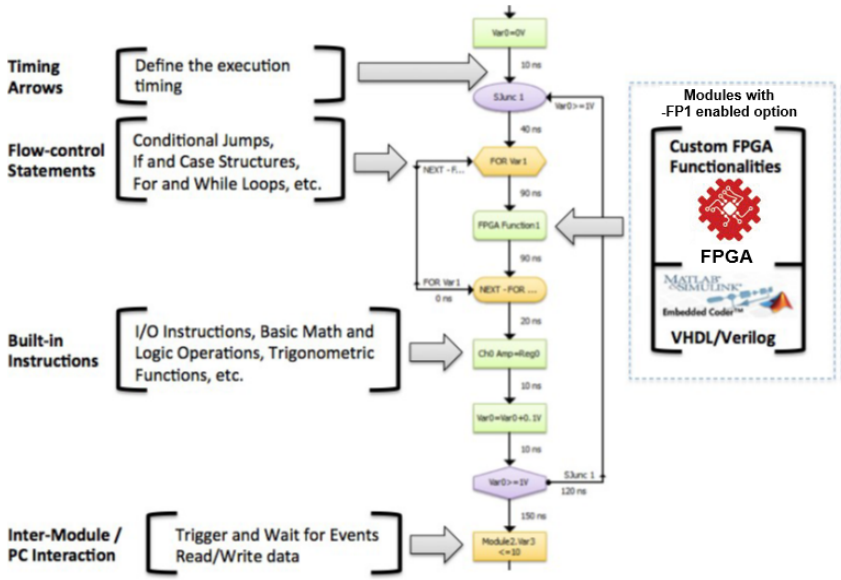
Keysight M3601A is based on flowchart programming, providing an easy-to-use environment to develop hard real-time applications.

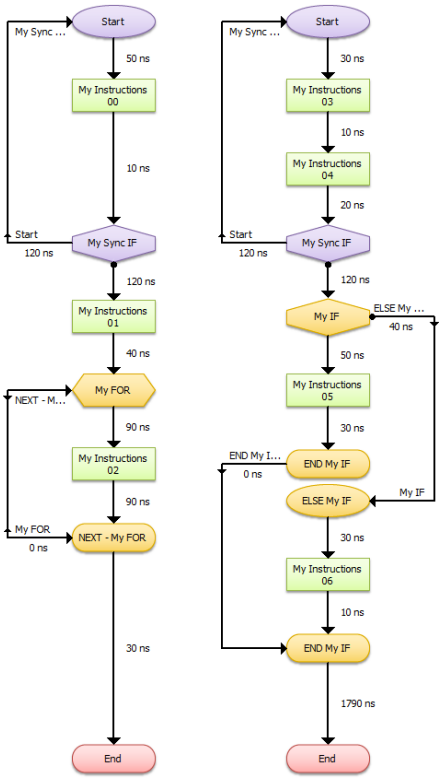
Keysight M3601A Hard Virtual Instrument (HVI) Design Environment Software provides:

- **Ultra-fast hard real time execution, processing, and decision making:** Execution is hardware-timed and can be as fast as 1 nanosecond, matching very high-performance FPGA-based systems and outperforming any real-time operating system.
- **User-friendly flowchart-style programming interface:** Keysight M3601A provides an intuitive flowchart-style programming environment that makes HVI programming extremely fast and easy. Using M3601A and its set of built-in instructions (the same instructions available for VIs), the user can program the hardware modules without any knowledge in FPGA technology, VHDL, etc.
- **Off-the-shelf intermodule synchronization and data exchange:** Each HVI is defined by a group of hardware modules which work perfectly synchronized, without the need of any external trigger or additional external hardware.
  In addition, Keysight modules exchange data and decisions for ultra-fast control algorithms.
- **Complete robustness:** Execution is performed by hardware, without operating system, and independently of the user PC.
- **Seamless integration with Keysight FPGA technology:** HVIs can interact with user-defined FPGA functions, making the real-time processing capabilities of HVIs unlimited.

- **Seamless integration with Keysight SD1 Programming Libraries:** In a complex control or test system, there are still some non- time-critical tasks that can only be performed by a VI, like for example: user interaction, visualization, or processing and decision tasks which are too complex to be implemented by hardware. Therefore, in a real application, the combination of VIs and HVIs is required. This task can be performed seamlessly with Keysight programming tools, for example, the user can have many HVIs and can control them from a VI using instructions like start, stop, pause, etc.

NOTE **Tip**: New hardware functionalities without FPGA programming: Keysight's HVI technology is the perfect tool to create new hardware functionalities with FPGA-like performance and without any FPGA programming knowledge. Users can create a repository of HVIs that can be launched from VIs using the Keysight SD1 Programming Libraries.





In an HVI, all Keysight modules run in parallel and completely synchronized, executing one flowchart per module. This results in simpler systems without the need of triggers.

2. 2. 3 Keysight M3602A FPGA Design Environment Software

Keysight FPGA programming technology is managed with Keysight M3602A FPGA Design Environment Software, an intuitive graphical FPGA programming environment.

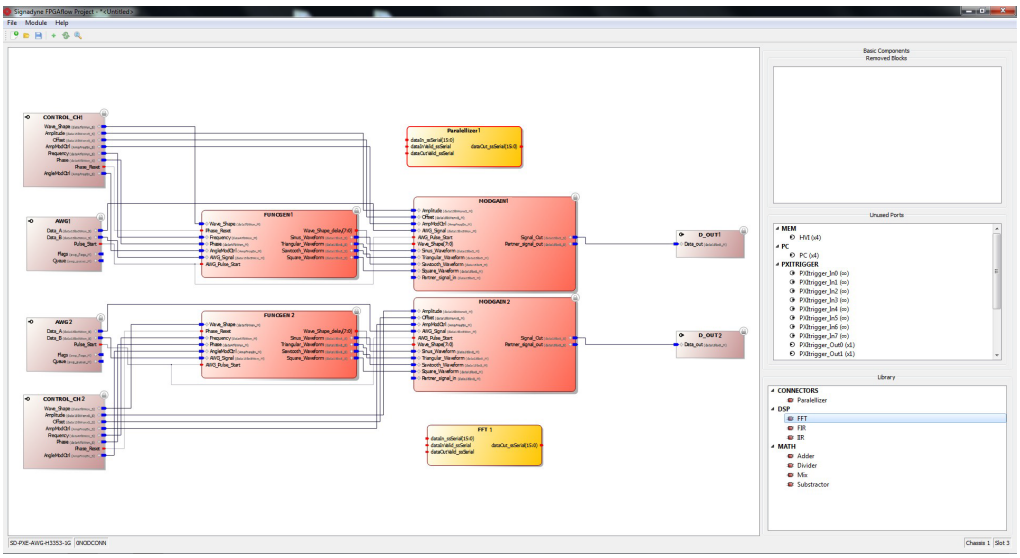
NOTE **Keysight M3201A/M3202A PXIe AWGs and M3100A/M3102A PXIe digitizers must have Option FP1 to use Keysight M3602A software; Option FP1 is only available at time of purchase.**

The following section is only an overview of the Keysight M3602A software; To learn how to use Keysight M3602A software, refer to the User's Guide for the Keysight M3602A FPGA Design Environment Software.

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.

All Keysight M3201A/M3202A PXIe AWGs and M3100A/M3102A PXIe digitizers can add Option FP1, which provide the same built-in functionalities of their standard counterparts, giving the users more time to focus on their specific functionalities. For example, using Option FP1 on a Keysight M3100A/M3102A PXIe digitizer, the user has all the off-the-shelf functionalities of the hardware (data capture, triggering, etc.), but custom real-time FPGA processing can be added in the data path, between the acquisition and the transmission of data to the computer.

M3602A



NOTE **FPGA programming made simple: Full language compatibility (including the graphical environment MATLAB/Simulink) and an easy-to-use FPGA graphical IDE, make Keysight FPGA programming extremely simple.**

An FPGA programming environment provides the following features:

Keysight M3602A is a complete FPGA programming environment that allows the user to customize Keysight M3201A/M3202A PXIe AWGs and M3100A/M3102A PXIe digitizers with Option FP1. Keysight M3602A provides the necessary tools to design, compile, and program the FPGA of the module.

User-friendly graphical FPGA programming environment:

- Complete platform, from design to FPGA programming: Keysight M3602A provides the necessary tools to design, compile, and program the FPGA of the module.

5x faster project development

Graphical environment without performance penalty

- **FPGA know-how requirement minimized:** The graphical environment provides a tool which does not require an extensive know-how in FPGA technology, improving the learning curve.

Streamlined design process:

- **Ready-to-use Keysight Block Library:** M3602A provides a continuously-growing library of blocks which reduces the need for custom FPGA-code development.
- **Include VHDL, Verilog, or Xilinx VIVADO/ISE projects:** Experienced FPGA users can squeeze the power of the onboard FPGA.

Include MATLAB/Simulink

Projects: MATLAB/Simulinkin conjunction with Xilinx

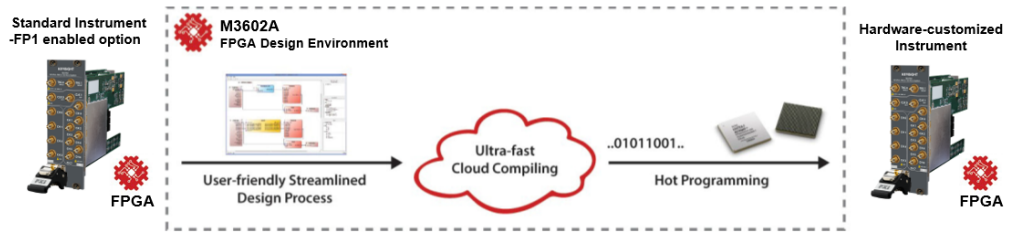
System Generator for DSP provides a powerful tool to implement digital signal processing. The user can go from the design/simulation power of MATLAB/Simulink to M3602A code in just a few clicks.

- **Include Xilinx CORE Generator IP cores:** Xilinx CORE Generator can be launched by M3602A to create IP cores that can be seamlessly included in the design.

Add and remove built-in resources to free up space: The user can remove unused built-in resources to free up more FPGA space.

One-click compiling and programming:

- **3x faster ultra-secure cloud FPGA compiling:** An ultra-fast cloud compiling system provides up to 3 times faster compiling. An ultra-secure TLS encrypted communication protects the IP of the user.
- **100x faster hot programming via PCI Express without rebooting:** Hardware can be reprogrammed without external cables and without rebooting the system.



Design Process: Customization vs. Complete Design

Keysight FPGA technology allows the user to customize Keysight M3201A/M3202A PXIe AWGs and M3100A/M3102A PXIe digitizers with Option FP1; these products are delivered with all the off-the-shelf functionalities of the standard products, and therefore the development time is dramatically reduced. The user can focus exclusively on expanding the functionality of the standard instrument, instead of developing a complete new one.

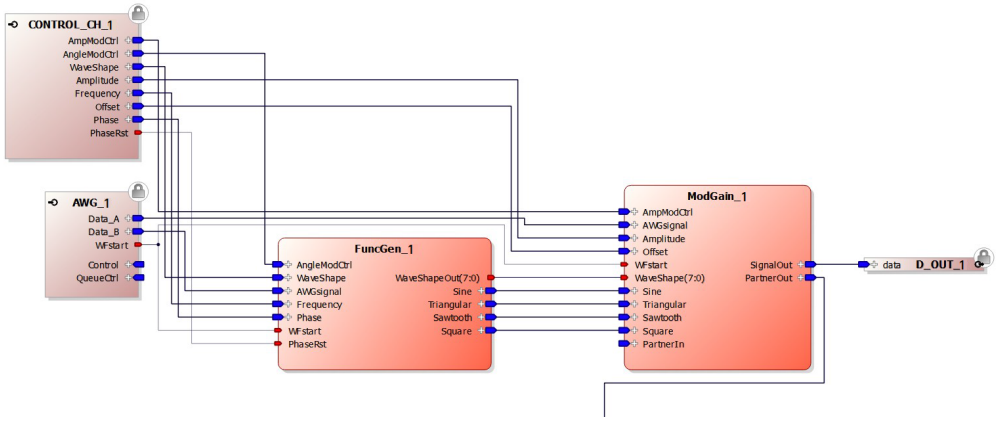
In Keysight M3602A, FPGA code is represented as boxes (called blocks) with IO ports. An empty project contains the "Default Product Blocks" (off-the-shelf functionalities), and the "Design IO Blocks" that provide the outer interface of the design. The user can add/remove blocks from the Keysight Block Library, External Blocks, or Xilinx IP cores.

2. 2. 3. 1 FPGA Programming Overview

NOTE **Keysight FPGA Block Library: Keysight M3602A provides a ready-to-use FPGA block library that reduces the requirement on FPGA know-how. Please check the M3602A User Guide to see a full description of the available FPGA blocks.**

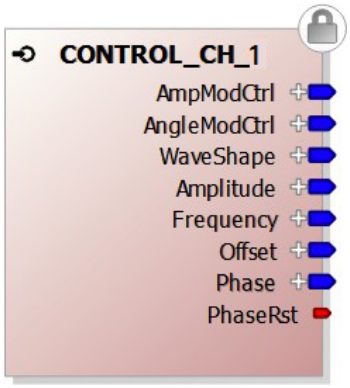
Keysight M3602A provides up to x3 faster FPGA compiling and hot programming without having to reboot the system.

M3602A Diagram Blocks



M3602A ControlCH Block

This block provides all the control parameters set by the user software using the Keysight SD1 Programming Libraries.

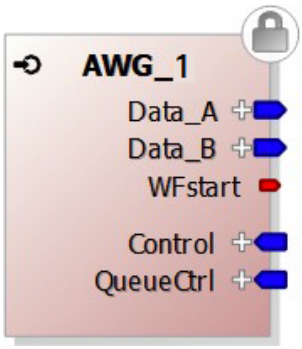


Parameters

Name	Description
Outputs	
AngleModCtrl	Angle modulation control (frequency or phase)
AmpModCtrl	Amplitude modulation control
WaveShape	Selects the output waveform
Amplitude	Signal amplitude value
Offset	DC offset value
Frequency	Signal frequency value
Phase	Signal phase value
PhaseRst	Signal to reset the phase of the function generator

M3602A AWG Block

This block is the Dual Arbitrary Waveform Generator.

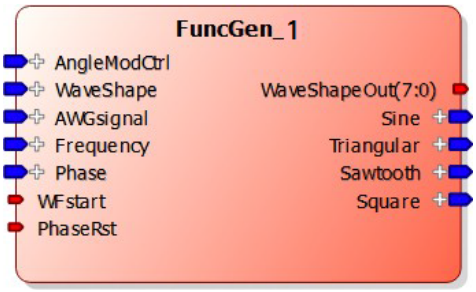


Parameters

Name	Description
Inputs	
Control	AWG operation control
QueueCtrl	AWG queue control
Outputs	
Data_A	Waveform A output (for dual waveforms), main waveform for single waveforms
Data_B	Waveform B output (for dual waveforms only)
WFstart	Signal that indicates when the AWG starts a waveform

M3602A FuncGen Block

This block is a function generator with angle modulation capabilities.



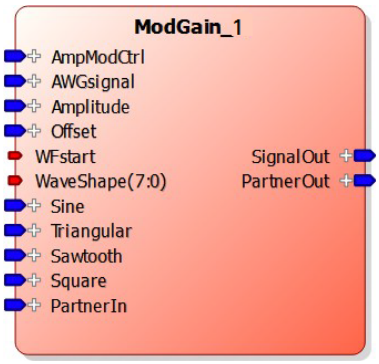
Parameters

Name	Description
Inputs	
AngleModCtrl	Configures the angle modulation (frequency or phase)
WaveShape	Selects the output waveform between Sine, Triangular or Square
AWGsignal	Arbitrary waveform coming from the AWG. It is used as the modulating signal
Frequency	Signal frequency value
Phase	Signal phase value
WFstart	Signal that indicates when the AWG signal starts a waveform
PhaseRst	Signal to reset the phase of the function generator
Outputs	
WaveShapeOut	Indicates which of the output signals is valid
Sine	Sinusoidal waveform
Triangular	Triangular waveform
Sawtooth	Not used
Square	Square waveform

M3602A ModGain Block

This block has the following functionalities:

- It selects the output waveform between Sine, Triangular, Sawtooth, Square, Partner Channel, or AWG
- It modulates the amplitude and the offset of the signal

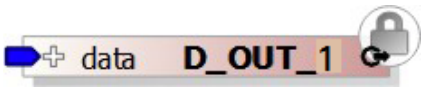


Parameters

Name	Description
Inputs	
AmpModCtrl	Configures the amplitude modulator
WaveShape	Selects the output waveform between: Sine, Triangular, Square, Partner Channel, or AWG
WFstart	Signal that indicates when the AWGsignal starts a waveform
Amplitude	Signal amplitude value
OffsetDC	offset value
Sine	Sinusoidal waveform coming from the Function Generator
Triangular	Triangular waveform coming from the Function Generator
Sawtooth	Not used
Square	Square waveform coming from the Function Generator
PartnerIn	Waveform coming from the Partner Channel. Used only in odd channels
AWGsignal	Arbitrary waveform coming from the AWG. It can be routed to SignalOut, or it can be used as the modulating signal
Outputs	
SignalOut	Output signal
PartnerOut	Copy of the output signal used for the even Partner Channel.

M3602A DOut Block

This block sends the data directly to the hardware analog output.



Parameters

Name	Description
Inputs	
data	data to be sent to the analog output channel

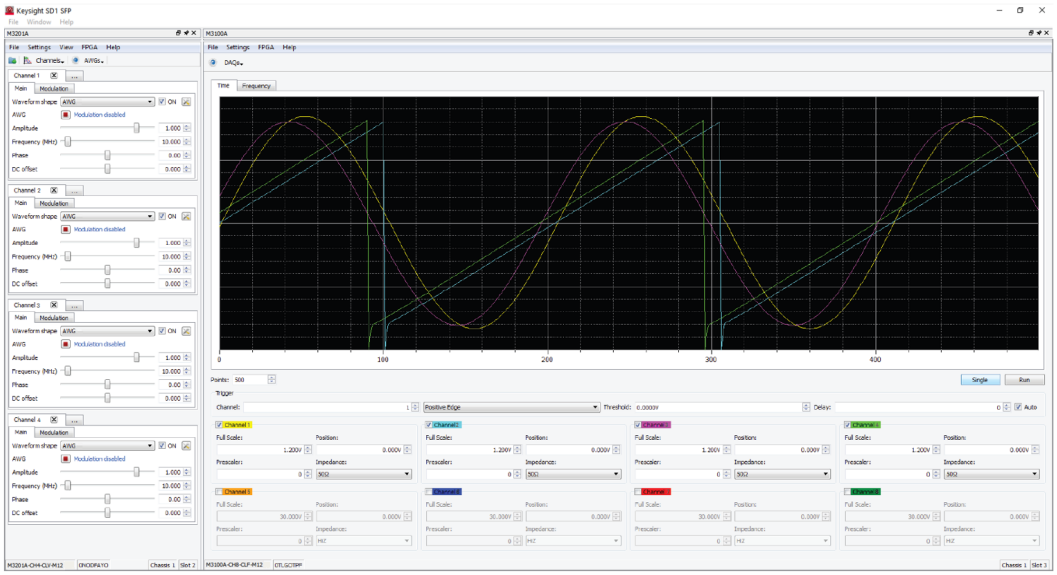
3 Using Keysight SD1 SFP Software

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.

When SD1 SFP is opened, 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.

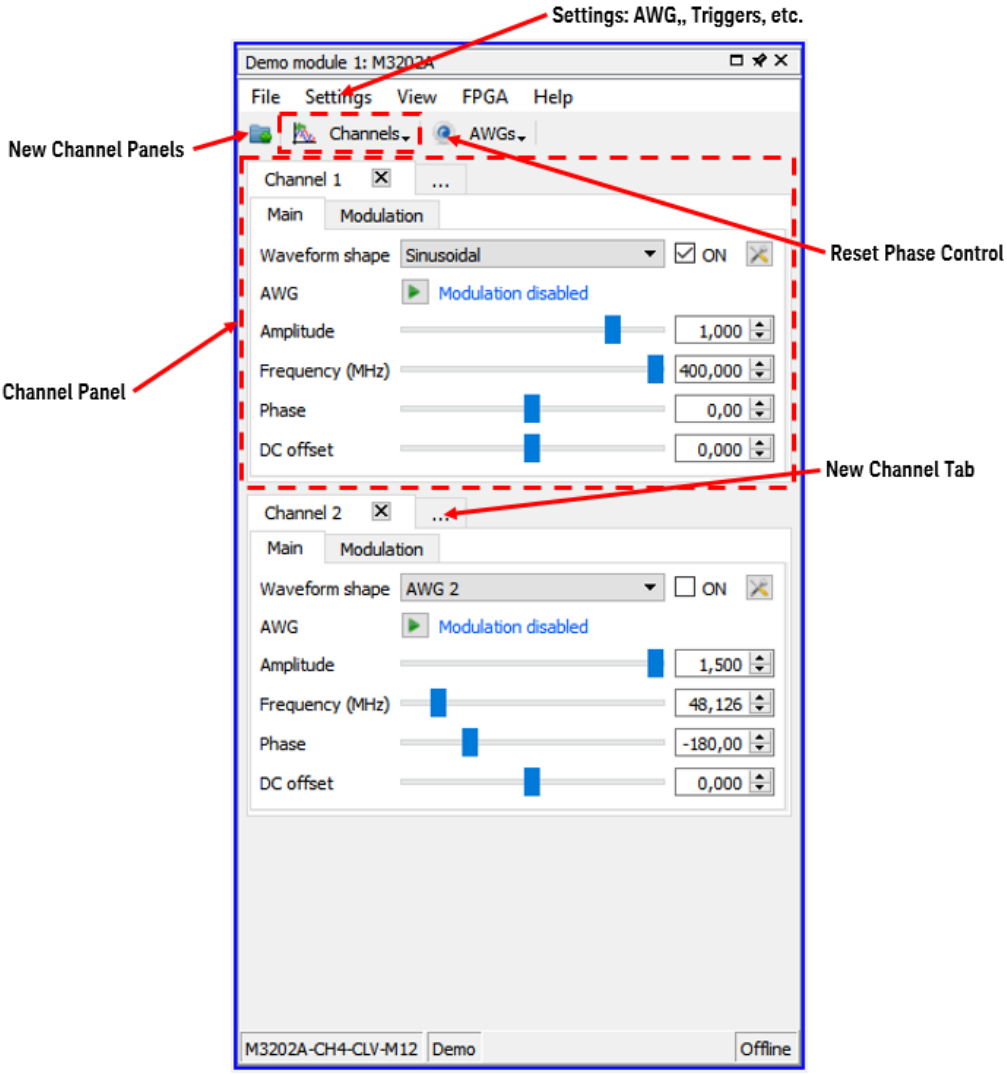
SD1 SFP



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.

3. 1 Main Soft Front Panel Controls

The M3201A/M3202A PXIe AWGs soft front panel appears automatically when SD1 SFP is launched and the module is connected to the chassis. If there are no modules available, SD1 SFP will launch "Demo Offline" modules.

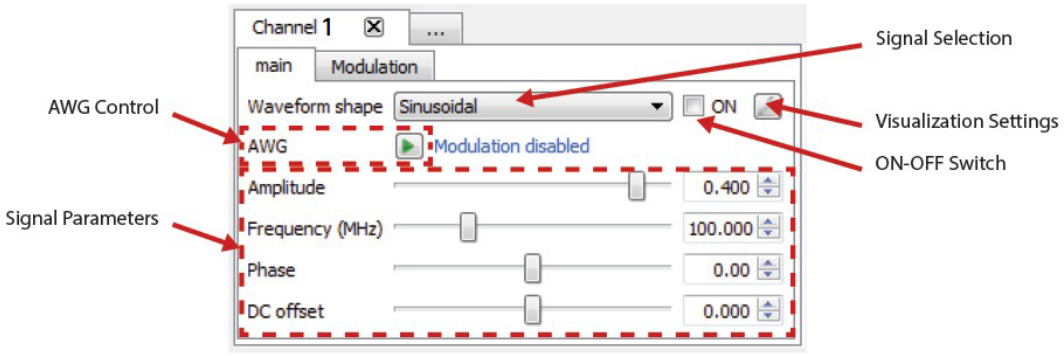


M3201A/M3202A PXIe AWGs Soft Front Panel Controls

When SD1 SFP is launched, the M3201A/M3202A PXIe AWGs soft front panel appears empty, waiting for the user to add ”Channels”, which are windows that control the channel operation. For maximum visualization flexibility, output channels can be added as new Panels or as Tabs within an existing Panel.

3. 2 Signal Generation Controls

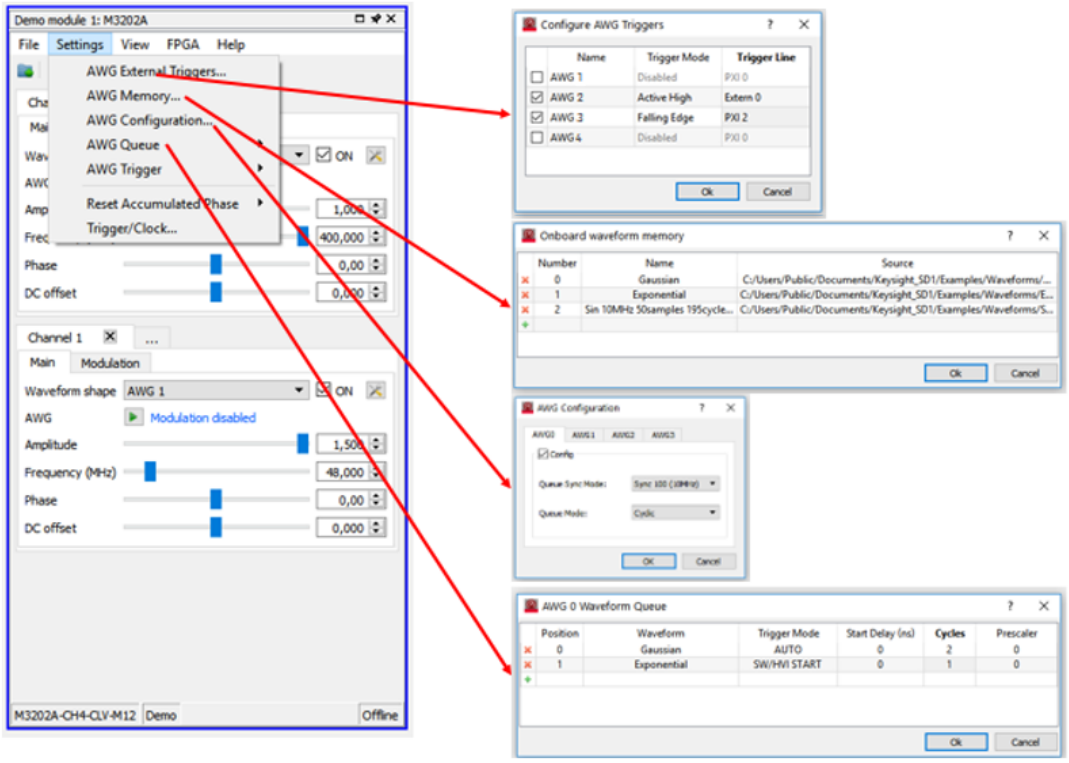
Signal generation controls are available for the M3201A/M3202A PXIe AWGs.



M3201A/M3202A PXIe AWGs signal generation control

3. 3 Arbitrary Waveform Generation Controls

AWG dialogs and the workflow to generate arbitrary waveforms is available using the AWGs.



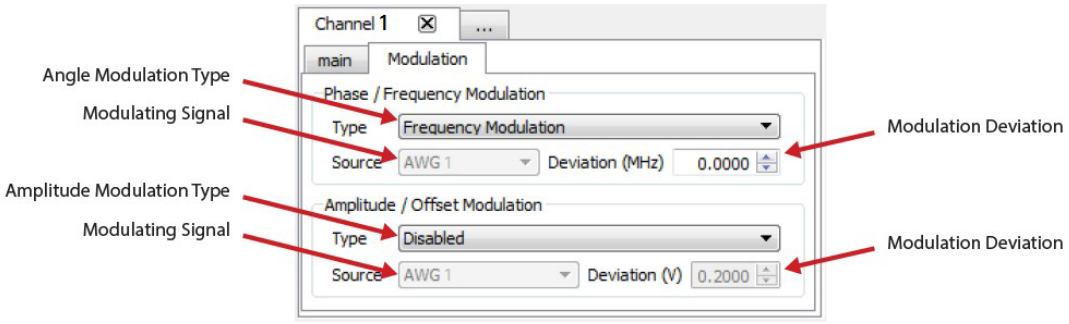
AWG Workflow:

1. (Optional) Configuration of the external AWG triggers: If external triggers are required, this dialog must be used to configure the source (external trigger connector, PXI trigger lines, etc.,) and the behavior (logic level high/low, rising/-falling edge, etc.).
2. Waveforms loading: Waveforms must be transferred to the onboard RAM in order to put them in the AWG queue. This dialog allows the user to select the waveform files and to assign custom waveform names. The waveforms are transferred to the onboard RAM when the dialog is closed.
3. AWG queue configuration: The waveforms loaded into the onboard RAM can be pushed into the corresponding AWG queue. This dialog allows the user to select the AWG queue options.

Once the waveforms are queued in the corresponding AWG, they can be launched using the AWG control button.

3. 4 Signal Modulation Controls

This tab configures the modulations.



M3201A/M3202A PXIe AWGs signal modulation control

As described in Working with Signal Modulation on page 22, the modulating signal for the modulators of one channel comes from the AWG corresponding to that particular channel. Therefore, the user must configure the AWG (See AWG workflow in SD1 SFP and the modulators). After this process, the user must run the AWG in the channel panel.

IQ modulators use both the amplitude modulator and the phase modulator. Therefore, the IQ modulator can be selected from any of the ”modulation type” combo boxes.

4 Using Keysight SD1 Programming Libraries

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

4. 1 Overall AWG Work Flow Using Python

In some languages, such as Python, **objects** (high-lighted in bold) have to be created.

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

4. 2 Example Programs Using Python

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

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

4. 2. 1 Example Program of Overall Work Flow for Python

This example program shows Python using the Keysight SD1 Programming Libraries.

```
# ----------
# Python - Sample Application, Overall Work Flow and Sequence of Commands
# ----------
# 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 key # Import SD1 library and AWG/Digitizer commands.

# ----------
# Specify values for variables related to the AWG and Chassis
AWG_PRODUCT = "M3202A" # Product's model number
CHASSIS = 1           # Chassis number holding product
AWG_SLOT = 4          # Slot number of product in chassis

# ----------
# Specify values for variables related to the AWG waveform
waveform_number = 1    # Numerical label of AWG waveform
cycles = 1             # Number of times to play a waveform from same channel
start_delay = 0        # Delay the start of the waveform playback
prescalar = 0          # How much to scale the outgoing waveform

# ----------
# Select settings and use specified variables
awg = key.SD_AOU()     # Create an AWG object

# ----------
# Open and connect to the specific AWG, using openWithSlot().
# If any errors occur, they are negative numbers and saved into aouID.
aouID = awg.openWithSlot(AWG_PRODUCT, CHASSIS, AWG_SLOT)
# Check aouID for errors and close connection if an error is found
if aouID < 0:
    # If aouID is a negative number, an error occurred so print it out
    print("ERROR")
    # Print error code so it can be looked up in the Keysight SD1 error list.
    print("aouID:", aouID)
    # Since there was an error, close the connection with the AWG.
    awg.close()
    # Print out a message that the connection is closed.
    print()
    print("AOU connection closed")

# If NO errors occur, flush/remove remaining old waveforms from AWG memory
awg.waveformFlush()

# ----------
# Set all four channels (1 to 4) of the AWG output mode
awg.channelWaveShape(1, key.SD_Waveshapes.AOU_AWG)
```

```
awg.channelWaveShape(2, key.SD_Waveshapes.AOU_AWG)
awg.channelWaveShape(3, key.SD_Waveshapes.AOU_AWG)
awg.channelWaveShape(4, key.SD_Waveshapes.AOU_AWG)

# ----------
# Create a new wave object
wave = key.SD_Wave()

# ----------
# Load a .csv file as the wave data
wave.newFromFile
("C:\\Users\Public\Documents\Keysight\SD1\Examples\Waveforms\Gaussian.csv")

# ----------
# Load the wave csv file into AWG memory,
# assigning it the arbitrary number set earlier in this program.
awg.waveformLoad(wave, waveform_number)

# ----------
# Queue everything that will be playing, with AWGqueueWaveform()
# AWGqueueWaveform(CHANNEL, number assigned to wave you want,
# trigger mode, delay before start (ns), number of times to play, prescaler)
awg.AWGqueueWaveform(1, waveform_number, key.SD_TriggerModes.SWHVITRIG,
start_delay, cycles, prescalar)
awg.AWGqueueWaveform(2, waveform_number, key.SD_TriggerModes.SWHVITRIG,
start_delay, cycles, prescalar)
awg.AWGqueueWaveform(3, waveform_number, key.SD_TriggerModes.SWHVITRIG,
start_delay, cycles, prescalar)
awg.AWGqueueWaveform(4, waveform_number, key.SD_TriggerModes.SWHVITRIG,
start_delay, cycles, prescalar)

# ----------
# Set the relative amplitudes for each channel.
# CSV waveforms are normalized between -1 and 1 * amplitude.
awg.channelAmplitude(1, 1.5)
awg.channelAmplitude(2, 1.5)
awg.channelAmplitude(3, 1.5)
awg.channelAmplitude(4, 1.5)

# ----------
# Start each channel's waveform - If trigger mode was set to AUTO, they would
start playing automatically but, since SD_TriggerModes.SWHVITRIG was
selected, a software trigger is required to play each channel's waveform
awg.AWGstart(1)
awg.AWGstart(2)
awg.AWGstart(3)
awg.AWGstart(4)

# ----------
# Trigger waveforms with software triggers to play the loaded waveforms
awg.AWGtrigger(1)
awg.AWGtrigger(2)
awg.AWGtrigger(3)
awg.AWGtrigger(4)
```

```
# ----------
# Close the connection with the AWG hardware.
awg.close()

# ----------
# © Keysight Technologies, 2018
# 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.
# ----------
```

4. 2. 2 Example Program Using Python to Produce a Sine Wave

This example program shows Python 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) Example: 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) Example: 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, 2018
# 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.
# ----------
```

4. 2. 3 Example Program Using Python to Produce a Sawtooth Wave from an Array

This example program shows Python 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, 2018
# 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.
# ----------
```

5 Keysight SD1 Command Reference

This chapter contains the following sections:

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.

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, C++, Visual Studio (VC++, C#, VB), MATLAB, National Instruments LabVIEW, Python, etc., ensuring full software compatibility and seamless multivendor integration. Ready-to-use native libraries are supplied for the following programming languages and compilers:

Language	Compiler	Library	Files
C	Microsoft Visual Studio . NET	.NET Library	*.dll
	MinGW (Qt), GCC	C Library	*.h, *.a
	Any C compiler	C Library	*.h, *.lib
C++	Microsoft Visual Studio . NET	.NET Library	*.dll
	MinGW (Qt), GCC	C++ Library	*.h, *.a
	C++ Builder / Turbo C++	C++ Library	*.h, *.lib
C#	Microsoft Visual Studio .NET	.NET Library	*.dll
MATLAB	MathWorks MATLAB	.NET Library	*.dll
Python	Any Python compiler	Python Library	*.py
Basic	Microsoft Visual Studio .NET	.NET Library	*.dll
LabVIEW	National Instruments LabVIEW	LabVIEW Library	*.vi

5. 2 Support for Other Programming Languages

Keysight provides dynamic libraries, e.g. 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.

Dynamic-link libraries available:

Exported Functions Language	Operating System	Files
C	Microsoft Windows	*.dll

NOTE **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++ or LabVIEW 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.

5. 3 Functions in SD1 Programming Libraries

The following functions are available in Keysight SD1 Programming Libraries.

SD_Module Functions on page 72

Function Name	Comments
open on page 72	Initializes a hardware module and must be called before using any other module-related function.
close on page 74	Releases all resources that were allocated for a module with open on page 72 and must always be called before exiting the application.
moduleCount on page 75	Returns the number of Keysight SD1 modules in the system.
getProductName on page 76	Returns the product name of the specified module.
getSerialNumber on page 77	Returns the serial number of the specified module.
getChassis on page 78	Returns the chassis number of where a module is located.
getSlot on page 79	Returns the slot number of where a module is located.
PXItriggerWrite on page 80	Sets the digital value of a PXI trigger in the PXI backplane. Only available in PXI/PXI Express form factors.
PXItriggerRead on page 81	Reads the digital value of a PXI trigger in the PXI backplane. Only available in PXI/PXI Express form factors.

SD_AOU Functions on page 82

Function Name	Comments
channelWaveShape on page 82	Sets the channel waveshape type.
channelFrequency on page 84	Sets the channel frequency for the periodic signal generated by the Function Generator.
channelPhase on page 85	Sets the channel phase for the periodic signals generated by the Function Generator.
channelPhaseReset on page 86	Resets the accumulated phase of the selected channel.
channelPhaseResetMultiple on page 87	Resets the accumulated phase of multiple selected channels, simultaneously.
channelAmplitude on page 88	Sets the amplitude of a channel.
channelOffset on page 89	Sets the DC offset of a channel.
modulationAngleConfig on page 90	Configures the modulation in frequency/phase for the selected channel.
modulationAmplitudeConfig on page 92	Configures the modulation in amplitude/offset for the selected channel.
modulationIQconfig on page 94	Sets the IQ modulation for the selected channel.
clockIOconfig on page 95	Configures the operation of the clock output connector.
waveformLoad on page 96	Loads a waveform into the module's onboard RAM.
waveformReLoad on page 98	Replaces a waveform located in the module's onboard RAM.
waveformFlush on page 100	Deletes all waveforms from the module's onboard RAM and flushes all AWG queues.
AWG on page 101	Provides a one-step method to load, queue, and start a

Function Name	Comments
	single waveform.
AWGqueueWaveform on page 104	Queues a waveform in the specified AWG.
AWGflush on page 106	Empties the queue of the AWG.
AWGstart on page 107	Runs the AWG starting from the beginning of its queue.
AWGstartMultiple on page 109	Runs the AWG starting from the beginning of its queue, but acting on more than one AWG at once.
AWGpause on page 110	Pauses the AWG.
AWGpauseMultiple on page 111	Pauses the AWG, but acting on more than one AWG at once.
AWGresume on page 112	Resumes the AWG.
AWGresumeMultiple on page 113	Resumes the AWG, but acting on more than one AWG at once.
AWGstop on page 114	Stops the AWG, resetting the queue to its initial position.
AWGstopMultiple on page 115	Stops the AWG, but acting on more than one AWG at once.
AWGreset on page 116	Resets the pointer variable that manages the AWG queue.
AWGjumpNextWaveform on page 117	Forces the AWG to jump to the next waveform in the queue.
AWGjumpNextWaveformMultiple on page 118	Forces the AWG to jump to the next waveform in the queue, but acting on more than one AWG at once.
AWGisRunning on page 119	Returns if the AWG is running or stopped.
AWGnWFplaying on page 120	Returns if the AWG is running or stopped, but acting on more than one AWG at once.
AWGtriggerExternalConfig on page 121	Configures the external triggers of the AWG.
AWGtrigger on page 123	Triggers the AWG.
AWGtriggerMultiple on page 124	Triggers the AWG, but acting on more than one AWG at once.
triggerIOconfig on page 125	Configures the trigger line direction.
triggerIOwrite on page 126	Sets the trigger output.
triggerIOread on page 128	Reads the trigger input.
clockSetFrequency (Requires Option CLV) on page 129	Sets the module clock frequency. This function is only usable for modules with the variable clock Option CLV.
clockGetFrequency on page 131	Returns the value in Hz of the module sample rate frequency.
clockGetSyncFrequency on page 132	Returns the frequency of the internal CLKsync signal in Hz.
clockResetPhase on page 133	Set modules in sync state, waiting for first trigger to reset the phase of the internal clocks CLKsync and CLKsys.
AWGqueueConfig on page 135	Configures the cyclic mode of the queue.
AWGqueueConfigRead on page 136	Reads the value of the cyclic mode of the queue.
AWGqueueMarkerConfig on page 137	Configures the Marker generation for each AWG.
AWGqueueSyncMode on page 139	Configures the sync mode of the queue.

SD_Wave Functions (new and delete) on page 140

Function Name	Comments
new on page 140	Creates a new waveform object in the PC RAM from a file or from an array.

Function Name	Comments
delete on page 142	Deletes the waveform object from the PC RAM.

SD_Module Functions (M3601A HVI-related) on page 143

Function Name	Comments
writeRegister on page 143	Writes a value in an HVI register of a hardware module (Option HV1 required).
readRegister on page 145	Reads a value from an HVI register of a hardware module (Option HV1 required).

SD_Module Functions (M3602A FPGA-related) on page 147

Function Name	Comments
FPGAwritePCport on page 147	Writes data at the PCport FPGA block (Option FP1 required).
FPGAreadPCport on page 149	Reads data at the PCport FPGA block (Option FP1 required).
FPGAload on page 151	Loads a bitstream file generated using M3602A software to FPGA (Option FP1 required).
FPGAreset on page 152	Sends a reset signal to FPGA (Option FP1 required).

5. 3. 1 SD_Module Functions

5. 3. 1. 1 open

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

Name	Description
Inputs	
productName	Module's product name (for example, "M3202A"). The product name can be found on the product or can be retrieved with getProductName on page 76.
serialNumber	Module's serial number (for example, "ES5641"). The serial number can be found on the product or can be retrieved with getSerialNumber on page 77.
chassis	Chassis number where the module is located. The chassis number can be found in Keysight SD1 software or can be retrieved with getChassis on page 78.
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 on page 79.
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 on page 4.
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
moduleID	(Non-object-oriented languages only) Module identifier or a negative number that indicates an error, see Error Codes on page 153.
errorOut	See Error Codes on page 153

C

```
int SD_Module_openWithSerialNumber(const char* productName, const char*
serialNumber);

int SD_Module_openWithSlot(const char* productName, int chassis, int slot);

int SD_Module_openWithSerialNumberCompatibility(const char* productName,
const char* serialNumber, int compatibility);

int SD_Module_openWithSlotCompatibility(const char* productName, int chassis,
int slot, int compatibility);
```

C++

```
int SD_Module::open(const char* productName, const char* serialNumber);

int SD_Module::open(const char* productName, int chassis, int slot);

int SD_Module::open(const char* productName, const char* serialNumber, int
compatibility);

int SD_Module::open(const char* productName, int chassis, int slot, int
compatibility);
```

Visual Studio .NET, MATLAB

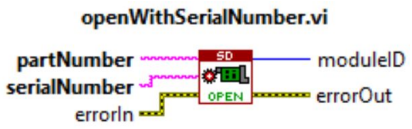
```
int SD_Module::open(string productName, string serialNumber);

int SD_Module::open(string productName, int chassis, int slot);

int SD_Module::open(string productName, string serialNumber, int
compatibility);

int SD_Module::open(string productName, int chassis, int slot, int
compatibility);
```

Python

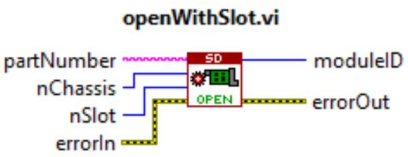
```
SD_Module.openWithSerialNumber(productName, serialNumber)

SD_Module.openWithSlot(productName, chassis, slot)

SD_Module.openWithSerialNumberCompatibility(productName, serialNumber,
compatibility)

SD_Module.openWithSlotCompatibility(productName, chassis, slot,
compatibility)
```

LabVIEW

`openWithSerialNumber.vi`



`openWithSlot.vi`



M3601A

`Available: No`

5. 3. 1. 2 close

Releases all resources that were allocated for a module with open on page 72 and must always be called before exiting the application.

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
errorOut	See Error Codes on page 153

C

```
int SD_Module_close(int moduleID);
```

C++

```
int SD_Module::close();
```

Visual Studio .NET, MATLAB

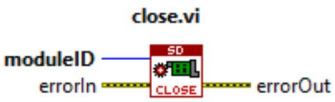
```
int SD_Module::close();
```

Python

```
SD_Module.close()
```

LabVIEW

```
close.vi
```



M3601A

```
Available: No
```

5. 3. 1. 3 moduleCount

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

NOTE **Static Function: (Object-oriented languages only)**
moduleCount is a static function

Parameters

Name	Description
Inputs	
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
nModules	Number of Keysight SD1 modules installed in the system. Negative numbers indicate an error, see Error Codes on page 153.
errorOut	(LabVIEW only) See Error Codes on page 153

C

```
int SD_Module_moduleCount();
```

C++

```
int SD_Module::moduleCount();
```

Visual Studio .NET, MATLAB

```
int SD_Module::moduleCount();
```

Python

```
SD_Module.moduleCount()
```

LabVIEW

```
Available: No
```

M3601A

```
Available: No
```

5. 3. 1. 4 getProductName

Returns the product name of the specified module.

NOTE **Static Function: (Object-oriented languages only) getProductName is a static function**

Parameters

Name	Description
Inputs	
index	Module index. It must be in the range (0 to nModules-1), where nModules is returned by moduleCount on page 75.
chassis	Chassis number where the module is located. The chassis number can be found in Keysight SD1 software or can be retrieved with getChassis on page 78.
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 on page 79.
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
productName	Product name of the specified module. This product name can be used in open on page 72.
errorOut	See Error Codes on page 153

C

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

C++

```
int SD_Module::getProductName(int index, char *productName);
int SD_Module::getProductName(int chassis, int slot, char* productName);
```

Visual Studio .NET, MATLAB

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

Python

```
SD_Module.getProductNameByIndex(index, productName)
SD_Module.getProductNameBySlot(chassis, slot, productName)
```

LabVIEW

```
Available: No
```

M3601A

```
Available: No
```

5. 3. 1. 5 getSerialNumber

Returns the serial number of the specified module.

NOTE **Static Function: (Object-oriented languages only)**
getSerialNumber is a static function

Parameters

Name	Description
Inputs	
index	Module index. It must be in the range (0 to nModules-1), where nModules is returned by moduleCount on page 75.
chassis	Chassis number where the module is located. The chassis number can be found in Keysight SD1 software or can be retrieved with getChassis on page 78.
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 on page 79.
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
serialNumber	Serial number of the specified module. This serial number can be used in open on page 72.
errorOut	See Error Codes on page 153

C

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

C++

```
int SD_Module::getSerialNumber(int index, char *serialNumber);
int SD_Module::getSerialNumber(int chassis, int slot, char* serialNumber);
```

Visual Studio .NET, MATLAB

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

Python

```
SD_Module.getSerialNumberByIndex(index, serialNumber)
SD_Module.getSerialNumberBySlot(chassis, slot, serialNumber)
```

LabVIEW

```
Available: No
```

M3601A

```
Available: No
```

5. 3. 1. 6 getChassis

Returns the chassis number of where a module is located.

NOTE **Static Function: (Object-oriented languages only) getChassis is a static function**

Parameters

Name	Description
Inputs	
index	Module index. It must be in the range (0 to nModules-1), where nModules is returned by moduleCount on page 75.
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
chassis	Chassis number of where a module is located. Negative numbers indicate an error, see Error Codes on page 153.
errorOut	(LabVIEW only) See Error Codes on page 153

C

```
int SD_Module_getChassis(int index);
```

C++

```
int SD_Module::getChassis(int index);
```

Visual Studio .NET, MATLAB

```
int SD_Module::getChassis(int index);
```

Python

```
SD_Module.getChassis()
SD_Module.getChassisByIndex(index)
```

LabVIEW

```
Available: No
```

M3601A

```
Available: No
```

5. 3. 1. 7 getSlot

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

NOTE **Static Function: (Object-oriented languages only)**
getSlot is a static function

Parameters

Name	Description
Inputs	
index	Module index. It must be in the range (0 to nModules-1), where nModules is returned by moduleCount on page 75.
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
slot	Slot number of where the module is located in the chassis. Negative numbers indicate an error, see Error Codes on page 153.
errorOut	(LabVIEW only) See Error Codes on page 153

C

```
int SD_Module_getSlot(int index);
```

C++

```
int SD_Module::getSlot(int index);
```

Visual Studio .NET, MATLAB

```
int SD_Module::getSlot(int index);
```

Python

```
SD_Module.getSlot()
SD_Module.getSlotByIndex(index)
```

LabVIEW

```
Available: No
```

M3601A

```
Available: No
```

5. 3. 1. 8 PXItriggerWrite

Sets the digital value of a PXI trigger in the PXI backplane.
This function is only available in PXI/PXI Express form factors.

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72.
nPXItrigger	PXI trigger number **Option** / **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. Selects between trigger lines on the backplane or the PXI chassis. / TRIG_PXI + Trigger No. / 4000 + Trigger No. See also, table after AWG External Trigger Source on page 17
value	Digital value with negated logic: 0 (ON) or 1 (OFF)
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
moduleIDOut	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes on page 153

C

```
int SD_Module_PXItriggerWrite(int moduleID, int nPXItrigger, int value);
```

C++

```
int SD_Module::PXItriggerWrite(int nPXItrigger, int value);
```

Visual Studio .NET, MATLAB

```
int SD_Module::PXItriggerWrite(int nPXItrigger, int value);
```

Python

```
SD_Module.PXItriggerWrite(nPXItrigger, value)
```

LabVIEW

```
Available: No
```

M3601A

```
Available: No
```

5. 3. 1. 9 PXItriggerRead

Reads the digital value of a PXI trigger in the PXI backplane.
This function is only available in PXI/PXI Express form factors.

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72.
nPXItrigger	PXI trigger number **Option** / **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 is synchronized to CLK10. — TRIG_PXI + Trigger No. — 4000 + Trigger No. See also, table after AWG External Trigger Source on page 17
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
moduleIDOut	(LabVIEW only) A copy of moduleID
value	Digital value with negated logic: 0 (ON) or 1 (OFF). Negative numbers indicate an error, see Error Codes on page 153.
errorOut	See Error Codes on page 153

C

```
int SD_Module_PXItriggerRead(int moduleID, int nPXItrigger);
```

C++

```
int SD_Module_PXItriggerRead(int moduleID, int nPXItrigger);
```

Visual Studio .NET, MATLAB

```
int SD_Module::PXItriggerRead(int nPXItrigger);
```

Python

```
SD_Module.PXItriggerRead(nPXItrigger, value)
```

LabVIEW

```
Available: No
```

M3601A

```
Available: No
(The value can be accessed using math operations: for example, MathAssign.)
```

5. 3. 2 SD_AOU Functions

5. 3. 2. 1 channelWaveShape

Sets the waveshape type for the selected channel.

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
nChannel	Channel number
waveShape	Channel waveshape type

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 on page 21.	AOU_AWG	6
Partner Channel	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

errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
moduleIDout	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes on page 153

C

```
int SD_AOU_channelWaveShape(int moduleID, int nChannel, int waveShape);
```

C++

```
int SD_AOU::channelWaveShape(int nChannel, int waveShape);
```

Visual Studio .NET, MATLAB

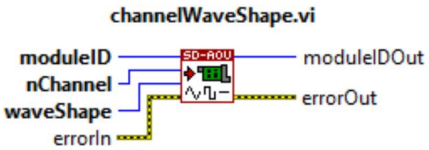
```
int SD_AOU::channelWaveShape(int nChannel, int waveShape);
```

Python

```
SD_AOU.channelWaveShape(nChannel, waveShape)
```

LabVIEW

```
channelWaveShape.vi
```



M3601A

```
Available: Yes
```

5. 3. 2. 2 channelFrequency

Sets the frequency, for the selected channel, of the periodic signals generated by the Function Generator block.

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
nChannel	Channel number
frequency	Frequency in Hz. (Refer to the product's datasheet for frequency specifications.)
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
moduleIDout	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes on page 153

C

```
int SD_AOU_channelFrequency(int moduleID, int nChannel, double frequency);
```

C++

```
int SD_AOU::channelFrequency(int nChannel, double frequency);
```

Visual Studio .NET, MATLAB

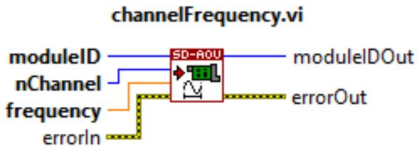
```
int SD_AOU::channelFrequency(int nChannel, double frequency);
```

Python

```
SD_AOU.channelFrequency(nChannel, frequency)
```

LabVIEW

```
channelFrequency.vi
```



M3601A

```
Available: Yes
```

5. 3. 2. 3 channelPhase

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

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
nChannel	Channel number
phase	Phase in degrees. (Refer to the product's datasheet for phase specifications.)
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
moduleIDout	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes on page 153

C

```
int SD_AOU_channelPhase(int moduleID, int nChannel, double phase);
```

C++

```
int SD_AOU::channelPhase(int nChannel, double phase);
```

Visual Studio .NET, MATLAB

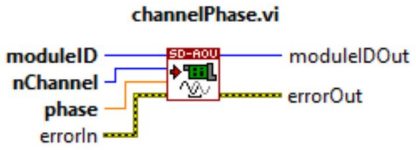
```
int SD_AOU::channelPhase(int nChannel, double phase);
```

Python

```
SD_AOU.channelPhase(nChannel, phase)
```

LabVIEW

```
channelPhase.vi
```



M3601A

```
Available: Yes
```

5. 3. 2. 4 channelPhaseReset

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

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
nChannel	Channel to reset
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
moduleIDout	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes on page 153

C

```
int SD_AOU_channelPhaseReset(int moduleID, int nChannel);
```

C++

```
int SD_AOU::channelPhaseReset(int nChannel);
```

Visual Studio .NET, MATLAB

```
int SD_AOU::channelPhaseReset(int nChannel);
```

Python

```
SD_AOU.channelPhaseReset(nChannel)
```

LabVIEW

```
channelPhaseReset.vi
```

M3601A

```
Available: Yes
```

5. 3. 2. 5 channelPhaseResetMultiple

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

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
channelMask	Mask to select the channels to reset (LSB is channel 1, bit 1 is channel 2, and so forth)
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
moduleIDout	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes on page 153

C

```
int SD_AOU_channelPhaseResetMultiple(int moduleID, int channelMask);
```

C++

```
int SD_AOU::channelPhaseResetMultiple(int channelMask);
```

Visual Studio .NET, MATLAB

```
int SD_AOU::channelPhaseResetMultiple(int channelMask);
```

Python

```
SD_AOU.channelPhaseResetMultiple(channelMask)
```

LabVIEW

```
channelPhaseResetMultiple.vi
```

M3601A

```
Available: No
(Multiple PhaseReset from different channels can be executed at once.)
```

5. 3. 2. 6 channelAmplitude

Sets the amplitude of a channel.
See Channel Amplitude and DC Offset on page 10.

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
nChannel	Channel number
amplitude	Amplitude in volts (–1.5 V to 1.5 V)
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
moduleIDout	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes on page 153

C

```
int SD_AOU_channelAmplitude(int moduleID, int nChannel, double amplitude);
```

C++

```
int SD_AOU::channelAmplitude(int nChannel, double amplitude);
```

Visual Studio .NET, MATLAB

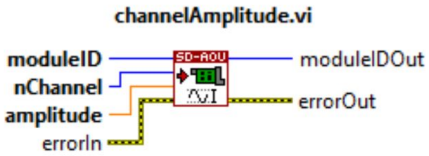
```
int SD_AOU::channelAmplitude(int nChannel, double amplitude);
```

Python

```
SD_AOU.channelAmplitude(nChannel, amplitude)
```

LabVIEW

```
channelAmplitude.vi
```



M3601A

```
Available: Yes
```

5. 3. 2. 7 channelOffset

Sets the DC offset of a channel.
See Channel Amplitude and DC Offset on page 10.

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
nChannel	Channel number
offset	DC offset in volts (–1.5 V to 1.5 V)
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
moduleIDout	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes on page 153

C

```
int SD_AOU_channelOffset(int moduleID, int nChannel, double offset);
```

C++

```
int SD_AOU::channelOffset(int nChannel, double offset);
```

Visual Studio .NET, MATLAB

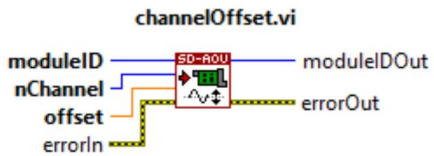
```
int SD_AOU::channelOffset(int nChannel, double offset);
```

Python

```
SD_AOU.channelOffset(nChannel, offset)
```

LabVIEW

```
channelOffset.vi
```



M3601A

```
Available: Yes
```

5. 3. 2. 8 modulationAngleConfig

Configures the modulation in frequency or phase for the selected channel.
See Channel Frequency and Phase on page 9.

Parameters

Name	Description			
Inputs				
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72			
nChannel	Channel number			
modulationType	Angle modulation options			
	Option	**Description**	**Name**	**Value**
	No Modulation	Modulation is disabled	AOU_MOD_OFF (default)	0
	Frequency Modulation	The AWG is used to modulate the channel frequency	AOU_MOD_FM	1
	Frequency Modulation (32 bits)	The AWG is used to modulate the channel frequency with 32 bit resolution	AOU_MOD_FM_32b	1
	Phase Modulation	The AWG is used to modulate the channel phase	AOU_MOD_PM	2
deviationGain	Gain for the normalized modulating signal			
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.			
Outputs				
moduleIDout	(LabVIEW only) A copy of moduleID			
errorOut	See Error Codes on page 153			

C

```
int SD_AOU_modulationAngleConfig(int moduleID, int nChannel, int
modulationType, int deviationGain);
```

C++

```
int SD_AOU::modulationAngleConfig(int nChannel, int modulationType, int
deviationGain);
```

Visual Studio .NET, MATLAB

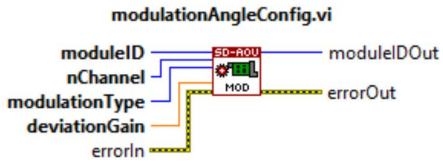
```
int SD_AOU::modulationAngleConfig(int nChannel, int modulationType, int
deviationGain);
```

Python

```
SD_AOU.modulationAngleConfig(nChannel, modulationType, deviationGain)
```

LabVIEW

modulationAngleConfig.vi



M3601A

Available: Yes

5. 3. 2. 9 modulationAmplitudeConfig

Configures the modulation in amplitude or offset for the selected channel.
See Channel Amplitude and DC Offset on page 10.

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
nChannel	Channel number
modulationType	Amplitude modulation options

Option	Description	Name	Value
No Modulation	Modulation is disabled. The channel amplitude and offset are only set by the main registers.	AOU_MOD_OFF (default)	0
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

deviationGain	Gain for the normalized modulating signal
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
moduleIDout	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes on page 153

C

```
int SD_AOU_modulationAmplitudeConfig(int moduleID, int nChannel, int modulationType, int deviationGain);
```

C++

```
int SD_AOU::modulationAmplitudeConfig(int nChannel, int modulationType, int deviationGain);
```

Visual Studio .NET, MATLAB

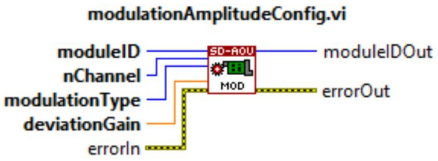
```
int SD_AOU::modulationAmplitudeConfig(int nChannel, int modulationType, int deviationGain);
```

Python

```
SD_AOU.modulationAmplitudeConfig(nChannel, modulationType, deviationGain)
```

LabVIEW

`modulationAmplitudeConfig.vi`



M3601A

`Available: Yes`

5. 3. 2. 10 modulationIQconfig

Configures the IQ modulation for the selected channel.
See IQ Modulation (Quadrature Modulator Block) on page 28.

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
nChannel	Channel number
enable	Enable (value 1) or Disable (value 0) the IQ modulation
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
moduleIDout	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes on page 153

C

```
int SD_AOU_modulationIQconfig(int moduleID, int nChannel, int enable);
```

C++

```
int SD_AOU::modulationIQconfig(int nChannel, int enable);
```

Visual Studio .NET, MATLAB

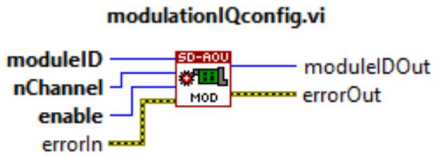
```
int SD-AOU::modulationIQconfig(int nChannel, int enable);
```

Python

```
SD_AOU.modulationIQconfig(nChannel, enable)
```

LabVIEW

```
modulationIQconfig.vi
```



M3601A

```
Available: No
```

5. 3. 2. 11 clockIOconfig

Configures the operation of the clock output connector.
See CLK Output Options on page 33.

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
clockConfig	Enable (value 1) or Disable (value 0) clock connector
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
moduleIDout	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes on page 153

Option	Description	Value
Disable	The CLK connector is disabled.	0
CLKref Output	The reference clock is available at the CLK connector.	1

C

```
int SD_AOU_clockIOconfig(int moduleID, int clockConfig);
```

C++

```
int SD_AOU::clockIOconfig(int clockConfig);
```

Visual Studio .NET, MATLAB

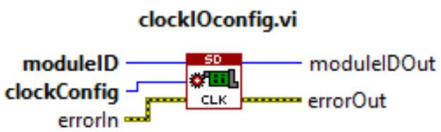
```
int SD_AOU::clockIOconfig(int clockConfig);
```

Python

```
SD_AOU.clockIOconfig(clockConfig)
```

LabVIEW

```
clockIOconfig.vi
```



M3601A

```
Available: No
```

5. 3. 2. 12 waveformLoad

Loads the specified waveform into the module's onboard RAM.
Waveforms must first be created with new on page 140.

Waveforms are loaded at different speeds depending on their length, longer waveforms will load at a faster rate. (e.g. 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

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
waveformID	Waveform identifier
waveformObject	Pointer to the waveform object
waveformType	Waveform type, used in the waveform file and in this function, defines the type of waveform to create; this parameter is used to internally configure the AWG.

Option	Description	Name	Value
Analog	Analog normalized waveforms (-1 to 1) defined with doubles	WAVE_ANALOG	0
Analog Dual	Analog normalized waveforms (-1 to 1) defined with doubles, with two components (A and B)	WAVE_ANALOG_DUAL	7
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.
See AWG Waveform Types on page 21.

waveformPoints	Number of points of the waveform, which must be a multiple of a certain number of points. See AWG specifications.
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).
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.

Name	Description
Outputs	
moduleIDOut	(LabVIEW only) A copy of moduleID
availableRAM	Available onboard RAM in waveform points, or negative numbers that indicate an error, see Error Codes on page 153.
errorOut	See Error Codes on page 153

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(SD_Wave* waveformObject, int waveformNumber, int paddingMode);

int SD_AOU::waveformLoad(int waveformType, int waveformPoints, short* waveformDataRaw, int waveformNumber, int paddingMode);
```

Visual Studio .NET, MATLAB

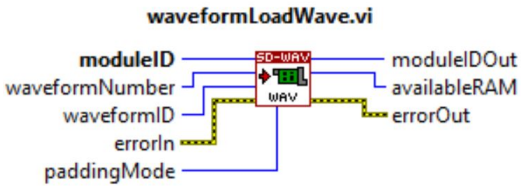
```
int SD_AOU::waveformLoad(SD_Wave waveformObject, int waveformNumber, int paddingMode);

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

Python

```
SD_AOU.waveformLoad(waveformObject, waveformNumber, paddingMode)

SD_AOU.waveformLoadInt16(waveformType, waveformDataRaw, waveformNumber, paddingMode)
```

LabVIEW

```
waveformLoadWave.vi (Note that LabVIEW uses a different function name.)
```



M3601A

```
Available: No
```

5. 3. 2. 13 waveformReLoad

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

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
waveformID	Waveform identifier, see SD_Wave Functions (new and delete) on page 140
waveformObject	Pointer to the waveform object, see SD_Wave Functions (new and delete) on page 140
waveformType	Waveform type, used in the waveform file and in this function, defines the type of waveform to create; this parameter is used to internally configure the AWG.

Option	Description	Name	Value
Analog	Analog normalized waveforms (-1 to 1) defined with doubles	WAVE_ANALOG	0
Analog Dual	Analog normalized waveforms (-1 to 1) defined with doubles, with two components (A and B)	WAVE_ANALOG_DUAL	7
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.
See AWG Waveform Types on page 21.

waveformPoints	Number of points of the waveform, which must be a multiple of a certain number of points. See AWG specifications.
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, 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).
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
moduleIDOut	(LabVIEW only) A copy of moduleID
availableRAM	Available onboard RAM in waveform points. Negative numbers indicate an error, see Error Codes on page 153.
errorOut	See Error Codes on page 153

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(SD_Wave* waveformObject, int waveformNumber, int
paddingMode);

int SD_AOU::waveformReLoad(int waveformType, int waveformPoints, short*
waveformDataRaw, int waveformNumber, int paddingMode);
```

Visual Studio .NET, MATLAB

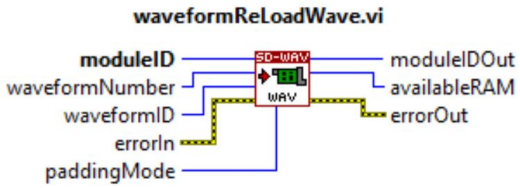
```
int SD_AOU::waveformReLoad(SD_Wave waveformObject, int waveformNumber, int
paddingMode);

int SD_AOU::waveformReLoad(int waveformType, short[] waveformDataRaw, int
waveformNumber, int paddingMode);
```

Python

```
SD_AOU.waveformReLoad(waveformObject, waveformNumber, paddingMode)

SD_AOU.waveformReLoadArrayInt16(waveformType, waveformDataRaw,
waveformNumber, paddingMode)
```

LabVIEW

```
waveformReLoadWave.vi (Note that LabVIEW uses a different function name.)
```



M3601A

```
Available: No
```

5. 3. 2. 14 waveformFlush

Deletes all waveforms from the module's onboard RAM and flushes all the AWG queues. See also AWGflush on page 106.

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
moduleIDOut	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes on page 153

C

```
int SD_AOU_waveformFlush(int moduleID);
```

C++

```
int SD_AOU::waveformFlush();
```

Visual Studio .NET, MATLAB

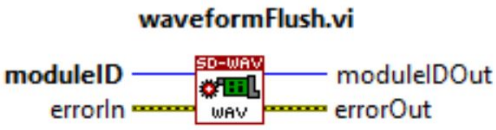
```
int SD_AOU::waveformFlush();
```

Python

```
SD_AOU.waveformFlush()
```

LabVIEW

```
waveformFlush.vi
```



M3601A

```
Available: No
```

5. 3. 2. 15 AWG

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.

NOTE **Step-by-Step Programming:** This AWG function is equivalent to:

1. creating a waveform with new on page 140
2. calling waveformLoad on page 96
3. followed by AWGqueueWaveform on page 104
4. and then calling AWGstart on page 107

Using these functions sequentially allows complete control of memory usage, data transfer times between the PC and the module, and the ability to create generation sequences and control the generation start time precisely.
See Working with AWG Waveforms on page 11.

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
nAWG	AWG channel number
triggerMode	Trigger method to launch the waveforms queued in an AWG.

Option	Description	Name	Value
Immediate (Auto)	The waveform is launched immediately after AWGstart on page 107, or when the previous waveform in the queue finishes.	AUTOTRIG	0
Software / HVI	Software trigger. The AWG is triggered by AWGtrigger on page 123 provided that the AWG is running. AWGtrigger can be executed from the user application (VI) or from an HVI. See Overview of Keysight Software and Programming Tools on page 35.	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 on page 121.)	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

startDelay	Defines the delay between the trigger and the waveform launch in tens of ns
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>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>waveformType</td><td>Waveform type, used in the waveform file and in this function, defines the type of waveform to create; this parameter is used to internally configure the AWG.
<table>
<tr><th>Option</th><th>Description</th><th>Name</th><th>Value</th></tr>
<tr><td>Analog</td><td>Analog normalized waveforms (-1 to 1) defined with doubles</td><td>WAVE_ANALOG</td><td>0</td></tr>
<tr><td>Analog Dual</td><td>Analog normalized waveforms (-1 to 1) defined with doubles, with two components (A and B)</td><td>WAVE_ANALOG_DUAL</td><td>7</td></tr>
<tr><td>IQ*</td><td>Analog normalized waveforms (-1 to 1) defined with doubles, with two components (I and Q)</td><td>WAVE_IQ</td><td>2</td></tr>
<tr><td>IQ Polar*</td><td>Analog waveforms (-1 to 1 module, -180 to +180 phase) defined with doubles, with two components (Magnitude and Phase)</td><td>WAVE_IQPOLAR</td><td>3</td></tr>
<tr><td>Digital</td><td>Digital waveforms defined with integers</td><td>WAVE_DIGITAL</td><td>5</td></tr>
</table>
* When using IQ or IQ Polar, each component will only play at a maximum rate of 500 MSa/s.
See AWG Waveform Types on page 21.</td></tr>
<tr><td>waveformPoints</td><td>Number of points of the waveform, which must be a multiple of a certain number of points. See AWG specifications.</td></tr>
<tr><td>waveformDataA</td><td>Array with waveform points. Analog waveforms are defined with floating point numbers, which correspond to a normalized amplitude (-1 to 1).</td></tr>
<tr><td>waveformDataB</td><td>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)</td></tr>
<tr><td>waveformFile</td><td>File containing the waveform points</td></tr>
<tr><td>errorIn</td><td>(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.</td></tr>
<tr><td>Outputs</td><td></td></tr>
<tr><td>moduleIDOut</td><td>(LabVIEW only) A copy of moduleID</td></tr>
<tr><td>availableRAM</td><td>Available onboard RAM in waveform points.
Negative numbers indicate an error, see Error Codes on page 153.</td></tr>
<tr><td>errorOut</td><td>See Error Codes on page 153</td></tr>
</table>

C

```
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 SD_AOU_AWGfromFile(int moduleID, int nAWG, char* waveformFile, int
triggerMode, int startDelay, int cycles, int prescaler);
```

C++

```
int SD_AOU::AWG(int nAWG, int triggerMode, int startDelay, int cycles, int
prescaler, int waveformType, int waveformPoints, double* waveformDataA,
double* waveformDataB=0);

int SD_AOU::AWG(int nAWG, char* waveformFile, int triggerMode, int
startDelay, int cycles, int prescaler);
```

Visual Studio .NET, MATLAB

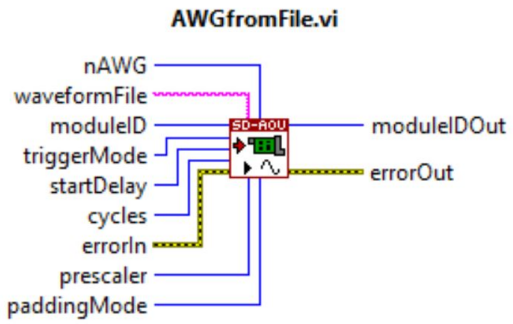
```
int SD_AOU::AWG(int nAWG, int triggerMode, int startDelay, int cycles, int
prescaler, int waveformType, double[] waveformDataA);

int SD_AOU::AWG(int nAWG, int triggerMode, int startDelay, int cycles, int
prescaler, int waveformType, double[] waveformDataA, double[] waveformDataB);

int SD_AOU::AWG(int nAWG, string waveformFile, int triggerMode, int
startDelay, int cycles, int prescaler);
```

Python

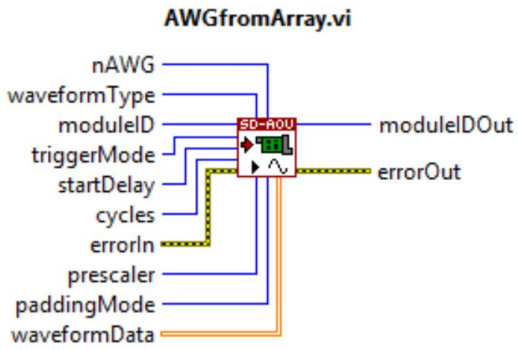
```
SD_AOU.AWGfromArray(nAWG, triggerMode, startDelay, cycles, prescaler,
waveformType, waveformDataA, waveformDataB=0)

SD_AOU.AWGfromFile(nAWG, waveformFile, triggerMode, startDelay, cycles,
prescaler)
```

LabVIEW

`AWGfromFile.vi`



`AWGfromArray.vi`



M3601A

```
Available: No
```

5. 3. 2. 16 AWGqueueWaveform

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 on page 96. The number of queued waveforms (regardless of cycles) is limited to 1024; AWGqueueWaveform can only be called 1024 times until the waveforms need to start being consumed by playing them.

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
nAWG	AWG channel number
waveformNumber	Waveform to be queued into the AWG. It must be already loaded using waveformLoad on page 96.
triggerMode	Trigger method to launch the waveforms queued in an AWG.

Option	Description	Name	Value
Immediate (Auto)	The waveform is launched immediately after AWGstart on page 107, or when the previous waveform in the queue finishes.	AUTOTRIG	0
Software / HVI	Software trigger. The AWG is triggered by AWGtrigger on page 123 provided that the AWG is running. AWGtrigger can be executed from the user application (VI) or from an HVI. See Overview of Keysight Software and Programming Tools on page 35.	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 on page 121.)	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

startDelay	Defines the delay between the trigger and the waveform launch in tens of ns. The startDelay parameter can be up to (2^{16}) - 1 cycles = (65535 * 10 ns) = 655.35 us.
cycles	Number of times the waveform is repeated once launched. (Zero specifies infinite cycles.)
prescaler	Waveform prescaler value, to reduce the effective sampling rate by prescaler x 5
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
moduleIDOut	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes on page 153

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);
```

Visual Studio .NET, MATLAB

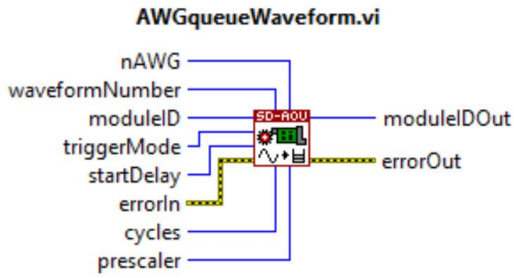
```
int SD_AOU::AWGqueueWaveform(int nAWG, int waveformNumber, int triggerMode, int startDelay, int cycles, int prescaler);
```

Python

```
SD_AOU.AWGqueueWaveform(nAWG, waveformNumber, triggerMode, startDelay, cycles, prescaler)
```

LabVIEW

```
AWGqueueWaveform.vi
```



M3601A

```
Available: Yes
```

5. 3. 2. 17 AWGflush

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

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
nAWG	AWG channel number
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
moduleIDOut	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes on page 153

C

```
int SD_AOU_AWGflush(int moduleID, int nAWG);
```

C++

```
int SD_AOU::AWGflush(int nAWG);
```

Visual Studio .NET, MATLAB

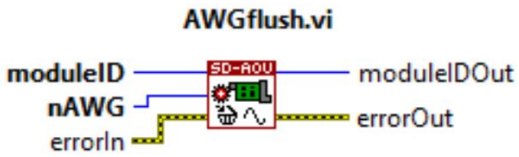
```
int SD_AOU::AWGflush(int nAWG);
```

Python

```
SD_AOU.AWGflush(nAWG)
```

LabVIEW

```
AWGflush.vi
```



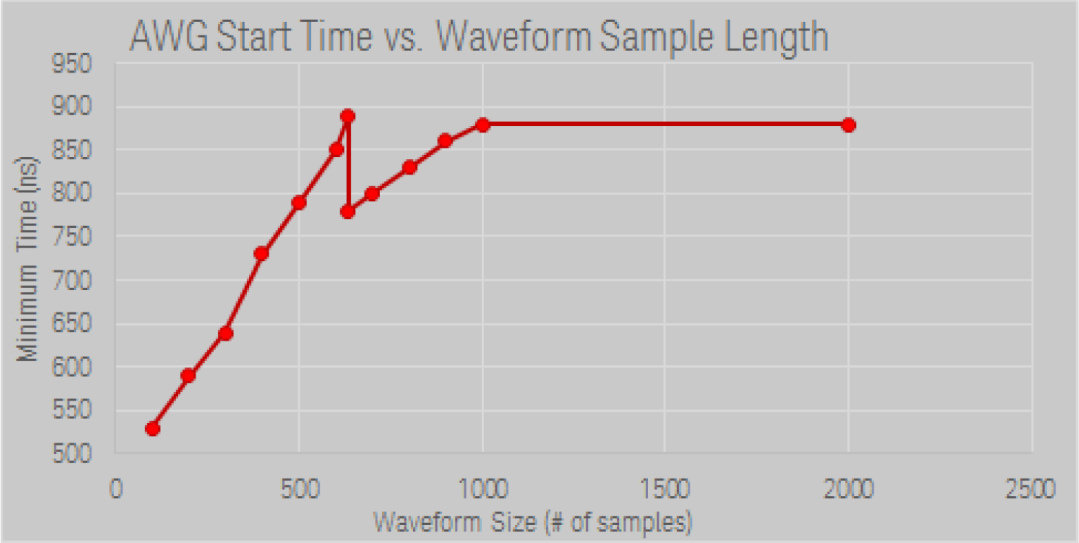
M3601A

```
Available: Yes
```

5. 3. 2. 18 AWGstart

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 on page 104 or AWG on page 101.

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.



Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
nAWG	AWG channel number
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
moduleIDOut	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes on page 153

C

```
int SD_AOU_AWGstart(int moduleID, int nAWG);
```

C++

```
int SD_AOU::AWGstart(int nAWG);
```

Visual Studio .NET, MATLAB

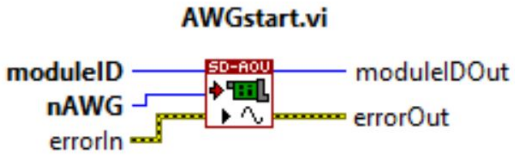
```
int SD_AOU::AWGstart(int nAWG);
```

Python

```
SD_AOU.AWGstart(nAWG)
```

LabVIEW

```
AWGstart.vi
```



M3601A

```
Available: Yes
```

5. 3. 2. 19 AWGstartMultiple

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 on page 104 or AWG on page 101.

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
AWGmask	Mask to select the AWGs to be started (LSB is AWG 0, bit 1 is AWG 1, and so forth)
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
moduleIDOut	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes on page 153

C

```
int SD_AOU_AWGstartMultiple(int moduleID, int AWGmask);
```

C++

```
int SD_AOU::AWGstartMultiple(int AWGmask);
```

Visual Studio .NET, MATLAB

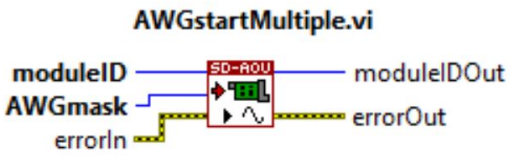
```
int SD_AOU::AWGstartMultiple(int AWGmask);
```

Python

```
SD_AOU.AWGstartMultiple(AWGmask)
```

LabVIEW

```
AWGstartMultiple.vi
```



M3601A

```
Available: No
(Multiple AWGstart from different channels can be executed at once.)
```

5. 3. 2. 20 AWGpause

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 on page 112.

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
nAWG	AWG channel number
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
moduleIDOut	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes on page 153

C

```
int SD_AOU_AWGpause(int moduleID, int nAWG);
```

C++

```
int SD_AOU::AWGpause(int nAWG);
```

Visual Studio .NET, MATLAB

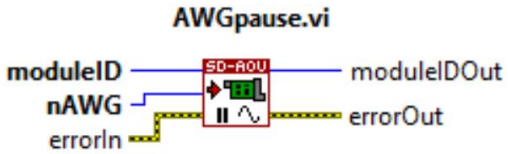
```
int SD_AOU::AWGpause(int nAWG);
```

Python

```
SD_AOU.AWGpause(nAWG)
```

LabVIEW

```
AWGpause.vi
```



M3601A

```
Available: Yes
```

5. 3. 2. 21 AWGpauseMultiple

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 on page 113.

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
AWGmask	Mask to select the AWG channels to be paused (LSB is Channel 1, bit 1 is Channel 2, and so forth)
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
moduleIDOut	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes on page 153

C

```
int SD_AOU_AWGpauseMultiple(int moduleID, int AWGmask);
```

C++

```
int SD_AOU::AWGpauseMultiple(int AWGmask);
```

Visual Studio .NET, MATLAB

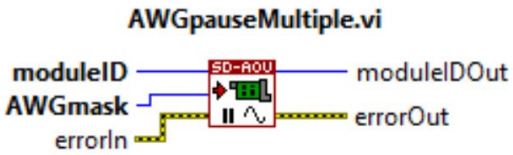
```
int SD_AOU::AWGpauseMultiple(int AWGmask);
```

Python

```
SD_AOU.AWGpauseMultiple(AWGmask)
```

LabVIEW

```
AWGpauseMultiple.vi
```



M3601A

```
Available: No
(Multiple AWGpause from different channels can be executed at once.)
```

5. 3. 2. 22 AWGresume

Resumes the operation of the selected AWG from the current position of the queue. The waveform generation can be paused by calling AWGpause on page 110.

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
nAWG	AWG channel number
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
moduleIDOut	(LabVIEW only) A copy of moduleID
errorOutS	See Error Codes on page 153

C

```
int SD_AOU_AWGresume(int moduleID, int nAWG);
```

C++

```
int SD_AOU::AWGresume(int nAWG);
```

Visual Studio .NET, MATLAB

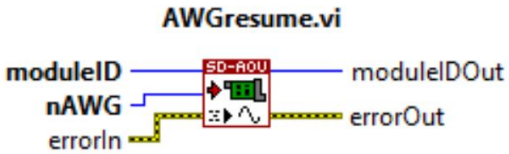
```
int SD_AOU::AWGresume(int nAWG);
```

Python

```
SD_AOU.AWGresume(nAWG)
```

LabVIEW

```
AWGresume.vi
```



M3601A

```
Available: Yes
```

5. 3. 2. 23 AWGresumeMultiple

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 on page 111.

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
AWGmask	Mask to select the AWG channels to be resumed (LSB is Channel 1, bit 1 is Channel 2, and so forth)
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
moduleIDOut	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes on page 153

C

```
int SD_AOU_AWGresumeMultiple(int moduleID, int AWGmask);
```

C++

```
int SD_AOU::AWGresumeMultiple(int AWGmask);
```

Visual Studio .NET, MATLAB

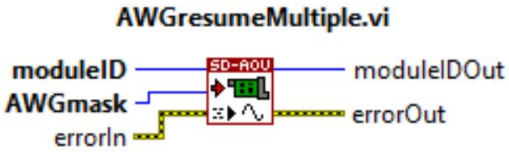
```
int SD_AOU::AWGresumeMultiple(int AWGmask);
```

Python

```
SD_AOU.AWGresumeMultiple(AWGmask)
```

LabVIEW

```
AWGresumeMultiple.vi
```



M3601A

```
Available: No
(Multiple AWGresume from different channels can be executed at once.)
```

5. 3. 2. 24 AWGstop

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

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
nAWG	AWG channel number
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
moduleIDOut	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes on page 153

C

```
int SD_AOU_AWGstop(int moduleID, int nAWG);
```

C++

```
int SD_AOU::AWGstop(int nAWG);
```

Visual Studio .NET, MATLAB

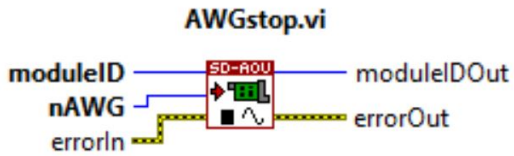
```
int SD_AOU::AWGstop(int nAWG);
```

Python

```
SD_AOU.AWGstop(nAWG)
```

LabVIEW

```
AWGstop.vi
```



M3601A

```
Available: Yes
```

5. 3. 2. 25 AWGstopMultiple

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

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
AWGmask	Mask to select the AWG channels to be stopped (LSB is Channel 1, bit 1 is Channel 2, and so forth)
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
moduleIDOut	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes on page 153

C

```
int SD_AOU_AWGstopMultiple(int moduleID, int AWGmask);
```

C++

```
int SD_AOU::AWGstopMultiple(int AWGmask);
```

Visual Studio .NET, MATLAB

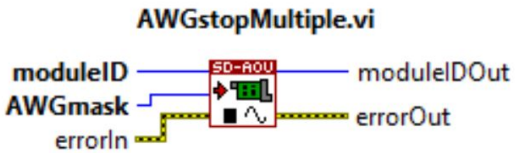
```
int SD_AOU::AWGstopMultiple(int AWGmask);
```

Python

```
SD_AOU.AWGstopMultiple(AWGmask)
```

LabVIEW

```
AWGstopMultiple.vi
```



M3601A

```
Available: No
(Multiple AWGstop from different channels can be executed at once.)
```

5. 3. 2. 26 AWGreset

Resets the pointer variable that manages the AWG queue. After a call to AWGreset, the pointer variable is pointing to the first queued waveform. Whenever an AWGreset is performed, it is necessary to call AWGresume on page 112 or AWGstart on page 107 for the waveform reproduction to start again.

NOTE AWGstart consists of a sequential call of AWGreset and AWGresume functions.

NOTE AWGreset is available in [3] Keysight M3601A Hard Virtual Instrument (HVI) Design Environment Software on page 157 only, it cannot be used in any programming language. In M3601A, AWGreset is available to allow the user to execute, at FPGA level, the reset of the queue pointer variable without an automatic AWGresume immediately afterwards.

C

```
Available: No
```

C++

```
Available: No
```

Visual Studio .NET, MATLAB

```
Available: No
```

Python

```
Available: No
```

LabVIEW

```
Available: No
```

M3601A

```
Available: Yes
```

5. 3. 2. 27 AWGjumpNextWaveform

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

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
nAWG	AWG channel number
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
moduleIDOut	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes on page 153

C

```
int SD_AOU_AWGjumpNextWaveform(int moduleID, int nAWG);
```

C++

```
int SD_AOU::AWGjumpNextWaveform(int nAWG);
```

Visual Studio .NET, MATLAB

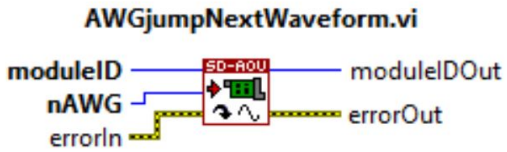
```
int SD_AOU::AWGjumpNextWaveform(int nAWG);
```

Python

```
SD_AOU.AWGjumpNextWaveform(nAWG)
```

LabVIEW

```
AWGjumpNextWaveform.vi
```



M3601A

```
Available: Yes
```

5. 3. 2. 28 AWGjumpNextWaveformMultiple

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

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
AWGmask	Mask to select the AWG channel numbers (LSB is Channel 1, bit 1 is Channel 2, and so forth)
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
moduleIDOut	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes on page 153

C

```
int SD_AOU_AWGjumpNextWaveformMultiple(int moduleID, int AWGmask);
```

C++

```
int SD_AOU::AWGjumpNextWaveformMultiple(int AWGmask);
```

Visual Studio .NET, MATLAB

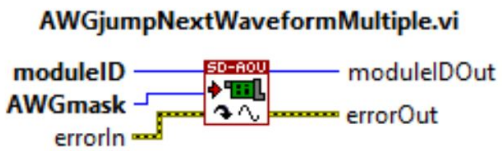
```
int SD_AOU::AWGjumpNextWaveformMultiple(int AWGmask);
```

Python

```
SD_AOU.AWGjumpNextWaveformMultiple(AWGmask)
```

LabVIEW

```
AWGjumpNextWaveformMultiple.vi
```



M3601A

```
Available: Yes
```

5. 3. 2. 29 AWGisRunning

Returns a value of 1 if the AWG is running or a value of 0 if it is stopped.

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
nAWG	AWG channel number
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
moduleIDOut	(LabVIEW only) A copy of moduleID
running	1 if the AWG is running, 0 if it is stopped
errorOut	See Error Codes on page 153

C

```
int SD_AOU_AWGisRunning(int moduleID, int nAWG);
```

C++

```
int SD_AOU::AWGisRunning(int nAWG);
```

Visual Studio .NET, MATLAB

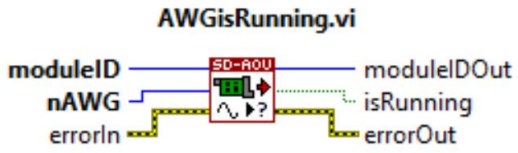
```
int SD_AOU::AWGisRunning(int nAWG);
```

Python

```
SD_AOU.AWGisRunning(nAWG)
```

LabVIEW

```
AWGisRunning.vi
```



M3601A

```
Available: No
```

5. 3. 2. 30 AWGnWFplaying

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

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
nAWG	AWG channel number
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
moduleIDOut	(LabVIEW only) A copy of moduleID
waveformNumber	Waveform identifier (see waveformLoad on page 96)
errorOut	See Error Codes on page 153

C

```
int SD_AOU_AWGnWFplaying(int moduleID, int nAWG);
```

C++

```
int SD_AOU::AWGnWFplaying(int nAWG);
```

Visual Studio .NET, MATLAB

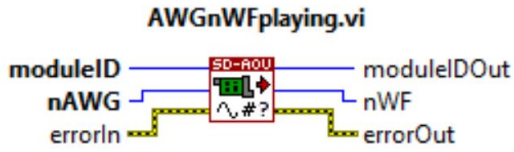
```
int SD_AOU::AWGnWFplaying(int nAWG);
```

Python

```
SD_AOU.AWGnWFplaying(nAWG)
```

LabVIEW

```
AWGnWFplaying.vi
```



M3601A

```
Available: No
```

5. 3. 2. 31 AWGtriggerExternalConfig

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 on page 104.

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
nAWG	AWG channel number
externalSource	AWG external trigger source

Option	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 is synchronized to CLK10. For example, to use PXI trigger line 1 on the PXI backplane, use `TRIG_PXI1 or 4001`.	TRIG_PXI + Trigger No.	4000 + Trigger No.

triggerBehavior — AWG external trigger behavior

Option	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

sync	0 for immediate trigger, 1 to synchronize with nearest CLK edge
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
moduleIDOut	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes on page 153

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);
```

Visual Studio .NET, MATLAB

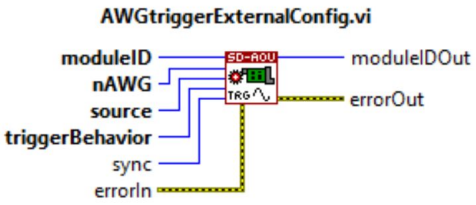
```
int SD_AOU::AWGtriggerExternalConfig(int nAWG, int externalSource, int
triggerBehavior, int sync);
```

Python

```
SD_AOU.AWGtriggerExternalConfig(nAWG, externalSource, triggerBehavior, sync)
```

LabVIEW

```
AWGtriggerExternalConfig.vi
```



M3601A

```
Available: No
```

5. 3. 2. 32 AWGtrigger

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

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
nAWG	AWG channel number
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
moduleIDOut	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes on page 153

C

```
int SD_AOU_AWGtrigger(int moduleID, int nAWG);
```

C++

```
int SD_AOU::AWGtrigger(int nAWG);
```

Visual Studio .NET, MATLAB

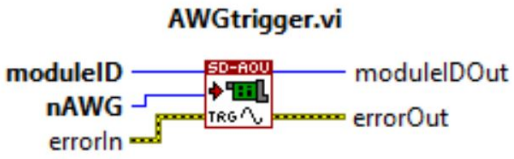
```
int SD_AOU::AWGtrigger(int nAWG);
```

Python

```
SD_AOU.AWGtrigger(nAWG)
```

LabVIEW

```
AWGtrigger.vi
```



M3601A

```
Available: Yes
```

5. 3. 2. 33 AWGtriggerMultiple

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

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
AWGmask	Mask to select the AWG channels to be triggered (LSB is Channel 1, bit 1 is Channel 2, and so forth)
AWG	AWG to be triggered
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
moduleIDOut	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes on page 153

C

```
int SD_AOU_AWGtriggerMultiple(int moduleID, int AWGmask);
```

C++

```
int SD_AOU::AWGtriggerMultiple(int AWGmask);
```

Visual Studio .NET, MATLAB

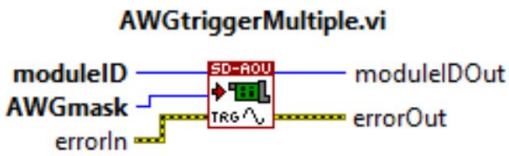
```
int SD_AOU::AWGtriggerMultiple(int AWGmask);
```

Python

```
SD_AOU.AWGtriggerMultiple(AWGmask)
```

LabVIEW

```
AWGtriggerMultiple.vi
```



M3601A

```
Available: No
(Multiple AWGtrigger from different channels can be executed at once.)
```

5. 3. 2. 34 triggerIOconfig

Configures the trigger line direction.
See Working with I/O Triggers on page 31.

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
direction	Output (0) or Input (1)

Option	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

errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
moduleIDout	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes on page 153

C

```
int SD_AOU_triggerIOconfig(int moduleID, int direction);
```

C++

```
int SD_AOU::triggerIOconfig(int direction);
```

Visual Studio .NET, MATLAB

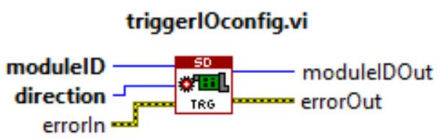
```
int SD_AOU::triggerIOconfig(int direction);
```

Python

```
SD_AOU.triggerIOconfig(direction)
```

LabVIEW

```
triggerIOconfig.vi
```



M3601A

```
Available: No
```

5. 3. 2. 35 triggerIOwrite

Sets the trigger output to be ON or OFF. The trigger must be configured as output using triggerIOconfig on page 125.

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
value	Trigger output value: 1 (ON), 0 (OFF)
syncMode	Sampling/synchronization mode 0 for immediate triggers, 1 to synchronize trigger to nearest CLK edge

Option	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 < 100 ns, varying depending on trigger arrival. If CLKsync is <10 MHz, the processing time will be >100 ns).

errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
moduleIDout	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes on page 153

C

```
int SD_AOU_triggerIOwrite(int moduleID, int value, int syncMode);
```

C++

```
int SD_AOU::triggerIOwrite(int value, int syncMode);
```

Visual Studio .NET, MATLAB

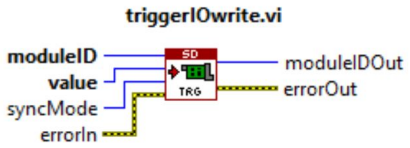
```
int SD_AOU::triggerIOwrite(int value, int syncMode);
```

Python

```
SD_AOU.triggerIOwrite(value, syncMode)
```

LabVIEW

`triggerIOwrite.vi`



M3601A

`Available: Yes`

5. 3. 2. 36 triggerIOread

Reads the trigger input.

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
moduleIDout	(LabVIEW only) A copy of moduleID
value	Trigger output value: 1 (ON), 0 (OFF). Negative numbers indicate an error, see Error Codes on page 153.
errorOut	See Error Codes on page 153

C

```
int SD_AOU_triggerIOread(int moduleID);
```

C++

```
int SD_AOU::triggerIOread();
```

Visual Studio .NET, MATLAB

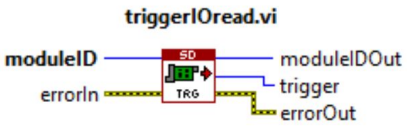
```
int SD_AOU::triggerIOread();
```

Python

```
SD_AOU.triggerIOread()
```

LabVIEW

```
triggerIOread.vi
```



M3601A

```
Available: No
(Can be accessed using math operations.)
```

5. 3. 2. 37 clockSetFrequency (Requires Option CLV)

Sets the module clock frequency. (See CLKsys in FlexCLK Technology (models w/ variable sampling rate) on page 33.)

NOTE **This clockSetFrequency function is only usable for modules with the variable clock Option CLV. This Option is no longer for sale as of November of 2019.**

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
frequency	Frequency in Hz. See datasheet for complete specifications.
mode	Operation mode of the variable clock system

Option	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 fastest tuning time, sacrificing jitter performance	CLK_FAST_TUNE	1

errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
moduleIDout	(LabVIEW only) A copy of moduleID
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 Error Codes on page 153.
errorOut	See Error Codes on page 153

*In Keysight Programming Libraries v.1.57.61 or older, clockSetFrequency returns CLKsyncFreq, the frequency of the internal CLKsync in Hz (Equation 11)

C

```
double SD_AOU_clockSetFrequency(int moduleID, double frequency, int mode);
```

C++

```
double SD_AOU::clockSetFrequency(double frequency, int mode);
```

Visual Studio .NET, MATLAB

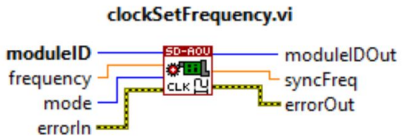
```
double SD_AOU::clockSetFrequency(double frequency, int mode);
```

Python

```
SD_AOU.clockSetFrequency(frequency, mode)
```

LabVIEW

clockSetFrequency.vi



M3601A

Available: No

5. 3. 2. 38 clockGetFrequency

Returns the value in Hz of the module sample rate frequency. (See CLKsys in FlexCLK Technology (models w/ variable sampling rate) on page 33.) It may differ from the frequency set with the clockSetFrequency (Requires Option CLV) on page 129 due to the hardware frequency resolution.

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
moduleIDout	(LabVIEW only) A copy of moduleID
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 Error Codes on page 153.
errorOut	See Error Codes on page 153

C

```
double SD_AOU_clockGetFrequency(int moduleID);
```

C++

```
double SD_AOU::clockGetFrequency();
```

Visual Studio .NET, MATLAB

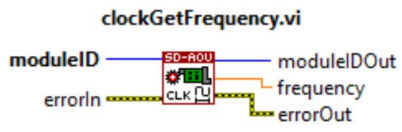
```
double SD_AOU::clockGetFrequency();
```

Python

```
SD_AOU.clockGetFrequency()
```

LabVIEW

```
clockGetFrequency.vi
```



M3601A

```
Available: No
```

5. 3. 2. 39 clockGetSyncFrequency

Returns the frequency of the internal CLKsync signal in Hz. (See CLKsync in FlexCLK Technology (models w/ variable sampling rate) on page 33.)

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
moduleIDout	(LabVIEW only) A copy of moduleID
CLKsyncFreq*	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 Error Codes on page 153.
errorOut	See Error Codes on page 153

C

```
int SD_AOU_clockGetSyncFrequency(int moduleID);
```

C++

```
int SD_AOU::clockGetSyncFrequency();
```

Visual Studio .NET, MATLAB

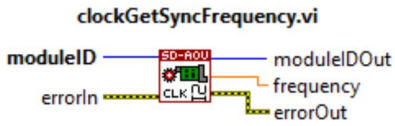
```
int SD_AOU::clockGetSyncFrequency();
```

Python

```
SD_AOU.clockGetSyncFrequency()
```

LabVIEW

```
clockGetSyncFrequency.vi
```



M3601A

```
Available: No
```

5. 3. 2. 40 clockResetPhase

Sets the module in a synchronous state, waiting for the first trigger to reset the phase of the internal clocks CLKsync and AWG on page 101. (See CLKsys in FlexCLK Technology (models w/ variable sampling rate) on page 33.)

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
triggerBehavior	AWG external trigger behavior

Option	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

PXItrigger — PXI trigger number

Option	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 is synchronized to CLK10.	TRIG_PXI + Trigger No.	4000 + Trigger No.

skew	Skew between PXI CLK10 and CLKsync in multiples of 10 ns
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
moduleIDout	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes on page 153

C

```
int SD_AOU_clockResetPhase(int moduleID, int triggerBehavior, int PXItrigger, double skew);
```

C++

```
int SD_AOU::clockResetPhase(int triggerBehavior, int PXItrigger, double skew);
```

Visual Studio .NET, MATLAB

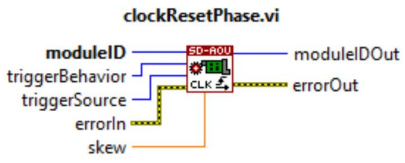
```
int SD_AOU::clockResetPhase(int triggerBehavior, int PXItrigger, double skew);
```

Python

```
SD_AOU.clockResetPhase(triggerBehavior, PXItrigger, skew)
```

LabVIEW

`clockResetPhase.vi`



M3601A

`Available: No`

5. 3. 2. 41 AWGqueueConfig

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 13.

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
nAWG	AWG channel number
mode	Operation mode of the queue: 0 One shot, 1 Cyclic. See AWG Waveform Queue System on page 13.
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
moduleIDout	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes on page 153

C

```
int SD_AOU_AWGqueueConfig(int moduleID, int nAWG, int mode);
```

C++

```
int SD_AOU::AWGqueueConfig(int nAWG, int mode);
```

Visual Studio .NET, MATLAB

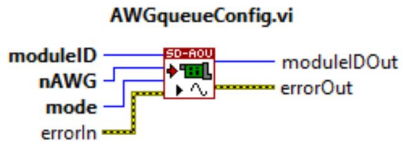
```
int SD_AOU::AWGqueueConfig(int nAWG, int mode);
```

Python

```
SD_AOU.AWGqueueConfig(nAWG, mode)
```

LabVIEW

```
AWGqueueConfig.vi
```



M3601A

```
Available: No
```

5. 3. 2. 42 AWGqueueConfigRead

Reads the value of the cyclic mode of the queue. All waveforms must be already queued (AWG Waveform Queue System on page 13) in one of the module's Arbitrary Waveform Generators.

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
nAWG	AWG channel number
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
moduleIDout	(LabVIEW only) A copy of moduleID
value	Cyclic mode value: 0 (OFF: One shot), 1 (ON). Negative numbers indicate an error, see Error Codes on page 153.
errorOut	See Error Codes on page 153

C

```
int SD_AOU_AWGqueueConfigRead(int moduleID, int nAWG);
```

C++

```
int SD_AOU::AWGqueueConfigRead(int nAWG);
```

Visual Studio .NET, MATLAB

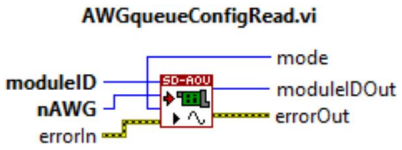
```
int SD_AOU::AWGqueueConfigRead(int nAWG);
```

Python

```
SD_AOU.AWGqueueConfigRead(nAWG)
```

LabVIEW

```
AWGqueueConfigRead.vi
```



M3601A

```
Available: No
```

5. 3. 2. 43 AWGqueueMarkerConfig

Configures the marker generation for each AWG. All waveforms must be already queued (AWG Waveform Queue System on page 13) 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

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
nAWG	AWG channel number to be configured
markerMode	Operation mode of the marker 0=Disabled, 1=On Start Event (when Start trigger is received) 2=On First Sample of Waveform (after WF startDelay), 3=On Every Cycle
trgPXImask	Mask to select PXI triggers to use (bit0->PXItrg0, bit1->PXItrg1, ...)
trgIOmask	Mask to select front-panel triggers to use (bit0->TriggerIO)
value	0=Low, 1=High (PXItrigger are active low signals, then 1 will generate a 0 pulse)
syncMode	0 is synchronized to CLKsys, 1 is synchronized to 10 MHz reference clock
length	Pulse length of the marker = length x T_{CLKsys} x 5 (length >1) (T_{CLKsys}= 1/CLKsys)
delaly	Delay to add before the marker pulse = delay x T_{CLKsys} x 5 (markerMode selects the start point of the marker, after which the delay is added)
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
moduleIDout	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes on page 153

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);
```

Visual Studio .NET, MATLAB

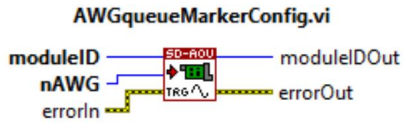
```
int SD_AOU::AWGqueueMarkerConfig(int nAWG, int markerMode, int trgPXImask, int trgIOmask, int value, int syncMode, int length, int delay);
```

Python

```
SD_AOU.AWGqueueMarkerConfig(nAWG, markerMode, trgPXImask, trgIOmask, value, syncMode, length, delay)
```

LabVIEW

AWGqueueMarkerConfig.vi



M3601A

Available: No

5. 3. 2. 44 AWGqueueSyncMode

Configures the synchronization mode of the queue.

All waveforms must be already queued (AWGqueueWaveform on page 104) in one of the module's AWGs.

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
nAWG	AWG channel number
syncMode	For 0 it is synchronized to CLKSYS, for 1 it is synchronized to the 10 MHz reference clock
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
moduleIDout	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes on page 153

C

```
int SD_AOU_AWGqueueSyncMode(int moduleID, int nAWG, int syncMode);
```

C++

```
int SD_AOU::AWGqueueSyncMode(int nAWG, int syncMode);
```

Visual Studio .NET, MATLAB

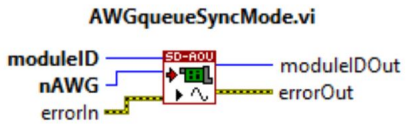
```
int SD_AOU::AWGqueueSyncMode(int nAWG, int syncMode);
```

Python

```
SD_AOU.AWGqueueSyncMode(nAWG, syncMode)
```

LabVIEW

```
AWGqueueSyncMode.vi
```



M3601A

```
Available: No
```

5. 3. 3 SD_Wave Functions (new and delete)

5. 3. 3. 1 new

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

ADVANCED

NOTE **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

Name	Description
Inputs	
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.

Option	Description	Name	Value
Analog	Analog normalized waveforms (-1 to 1) defined with doubles	WAVE_ANALOG	0
Analog Dual	Analog normalized waveforms (-1 to 1) defined with doubles, with two components (A and B)	WAVE_ANALOG_DUAL	7
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.
See AWG Waveform Types on page 21.

waveformPoints	Number of point 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.
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut.
Outputs	
waveformID	(Non-object-oriented languages only) Waveform identifier. Negative numbers indicate an error, see Error Codes on page 153.
errorOut	See Error Codes on page 153

C

```
int SD_Wave_newFromArrayDouble(int waveformType, int waveformPoints, double*
waveformDataA double* waveformDataB=0);

int SD_Wave_newFromFile(char* waveformFile);
```

C++

```
int SD_Wave SD AOU::SD AOU (int waveformType, int waveformPoints, double*
waveformDataA, double* waveformDataB=0);

int SD_Wave SD AOU::SD AOU (char* waveformFile);
```

Visual Studio .NET, MATLAB

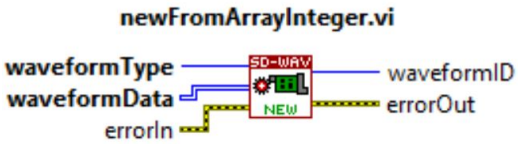
```
SD_Wave(int waveformType, double[] waveformDataA);

SD_Wave(int waveformType, double[] waveformDataA, double[] waveformDataB);

SD_Wave(string waveformFile);
```

Python

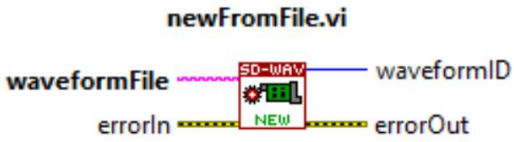
```
SD_Wave.newFromArrayDouble(waveformType, waveformDataA, waveformDataB=0)

SD_Wave.newFromFile(waveformFile)
```

LabVIEW

`newFromArrayInteger.vi`



`newFromFile.vi`



M3601A

```
Available: No
```

5. 3. 3. 2 delete

Removes a waveform created with the new function.

ADVANCED

NOTE **Onboard waveforms: Waveforms are removed from the PC RAM only, not from the module onboard RAM.**

Parameters

Name	Description
Inputs	
waveformID	Waveform identifier (returned by new on page 140)
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
errorOut	See Error Codes on page 153

C

```
int SD_Wave_delete(int waveformID);
```

C++

```
delete <SD_Wave object pointer>;
int SD_AOU::delete();
```

Visual Studio .NET, MATLAB

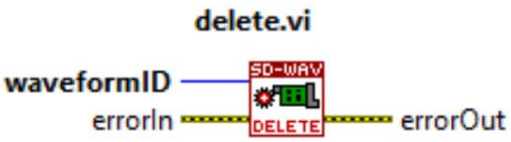
```
Automatically destroyed by the .NET garbage collector.
```

Python

```
Managed by Python.
```

LabVIEW

```
delete.vi
```



M3601A

```
Available: No
```

5. 3. 4 SD_Module Functions (M3601A HVI-related)

The **writeRegister** and **readRegister** functions are related to the [3] Keysight M3601A Hard Virtual Instrument (HVI) Design Environment Software on page 157.

5. 3. 4. 1 writeRegister

Writes a value in an HVI register of a hardware module (Option HV1 required).

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
regNumber	Register number
regName	Register name
regValue	Register value
unit	Unit of the register value
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
moduleIDout	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes on page 153

C

```
int SD_Module_writeRegister(int moduleID, int regNumber, int regValue);

int SD_Module_writeDoubleRegister(int moduleID, int regNumber, double
regValue, const char* unit);
```

C++

```
int SD_Module::writeRegister(int regNumber, int regValue);

int SD_Module::writeRegister(const char* regName, int regValue);

int SD_Module::writeRegister(int regNumber, double regValue, const char*
unit);

int SD_Module::writeRegister(const char* regName, double regValue, const
char* unit);
```

Visual Studio .NET, MATLAB

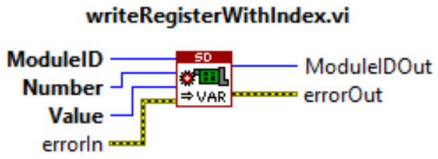
```
int SD_Module::writeRegister(int regNumber, int regValue);

int SD_Module::writeRegister(string regName, int regValue);

int SD_Module::writeRegister(int regNumber, double regValue, string unit);

int SD_Module::writeRegister(string regName, double regValue, string unit);
```

Python

```
SD_Module.writeRegisterByNumber(regNumber, varValue)

SD_Module.writeRegisterByName(regName, varValue)

SD_Module.writeRegisterDoubleByNumber(regNumber, value, unit)

SD_Module.writeRegisterDoubleByName(regName, value, unit)
```

LabVIEW

```
writeRegisterWithIndex.vi
```



M3601A

```
Available: No
(The value can be accessed using math operations: for example, MathAssign.)
```

5. 3. 4. 2 readRegister

Reads a value from an HVI register of a hardware module (Option HV1 required).

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
regNumber	Register number
regName	Register name
unit	Unit of the register value
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
regValue	Register value
moduleIDout	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes on page 153

C

```
int SD_Module_readRegister(int moduleID, int regNumber, int regValue);

double SD_Module_readDoubleRegister(int moduleID, int regNumber, const char*
unit, int& errorOut);
```

C++

```
int SD_Module::readRegister(int regNumber, int regValue);

int SD_Module::readRegister(const char* regName, int regValue);

double SD Module::readRegister(int regNumber, const char* unit, int&
errorOut);

double SD_Module::readRegister(const char* regName, const char* unit, int&
errorOut);
```

Visual Studio .NET, MATLAB

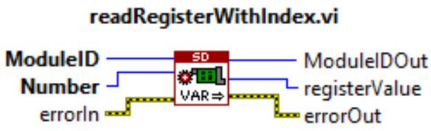
```
int SD_Module::readRegister(int regNumber, int regValue);

int SD_Module::readRegister(string regName, int regValue);

int SD_Module::readRegister(int regNumber, string unit, int errorOut);

int SD_Module::readRegister(string regName, string unit, int errorOut);
```

Python

```
SD_Module.readRegisterByNumber(regNumber)

SD_Module.readRegisterByName(regName)

SD_Module.readRegisterDoubleByNumber(regNumber, unit)

SD_Module.readRegisterDoubleByName(regName, unit)
```

LabVIEW

```
readRegisterWithIndex.vi
```



M3601A

```
Available: No
(The value can be accessed using math operations: for example, MathAssign.)
```

5. 3. 5 SD_Module Functions (M3602A FPGA-related)

The **FPGAwritePCport**, **FPGAreadPCport**, **FPGAload**, and **FPGAreset** functions are related to the [4] Keysight M3602A FPGA Design Environment Software on page 157.

5. 3. 5. 1 FPGAwritePCport

Writes data at the PCport FPGA block (Option FP1 required).

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
nPCport	PC port number
data	Data buffer to write through PC port to FPGA
dataSize	Number of 32-bit words to write (maximum is 128 words)
address	Address that appears in the PCport interface
addressMode	Selects between the two address modes shown below: **addressMode** / **Description** / **Name** / **Value** Auto Increment / Initial address is incremented after each access / ADDRESSING_AUTOINCREMENT / 0 Fixed / Initial address is used for the whole access / ADDRESSING_FIXED / 1
accessMode	Selects between the two memory access modes shown below: **accessMode** / **Description** / **Name** / **Value** Non-DMA / Memory access is split into multiple accesses / NONDMA / 0 DMA / Memory access is done with a DMA transaction / DMA / 1
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
moduleIDout	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes on page 153

C

```
int SD_Module_FPGAwritePCport(int moduleID, int nPCport, int* data, int
dataSize, int address, int addressMode, int accessMode);
```

C++

```
int SD_Module::FPGAwritePCport(int nPCport, int* data, int dataSize, int
address, SD_AddressingMode addressMode, SD_AccessMode accessMode);
```

Visual Studio .NET, MATLAB

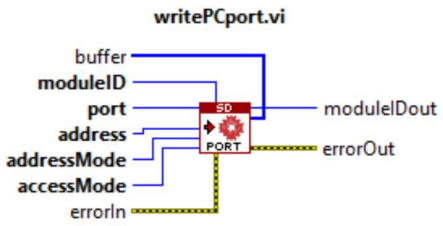
```
int SD_Module::FPGAwritePCport(int nPCport, int [] data, int address, SD_
AddressingMode addressMode, SD_AccessMode accessMode);
```

Python

```
SD_Module.FPGAwritePCport(nPCport, data, address, addressMode, accessMode)
```

LabVIEW

```
writePCport.vi
```



M3601A

```
Available: No
```

5. 3. 5. 2 FPGAreadPCport

Reads data at the PCport FPGA block (Option FP1 required).

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
nPCport	PCport number (as if appears in the M3602A software)
address	Address that appears in the PCport interface
dataSize	Number of 32-bit words to read (maximum is 128 words)
addressMode	Selects between the two address modes shown below:

addressMode	Description	Name	Value
Auto Increment	Initial address is incremented after each access	ADDRESSING_AUTOINCREMENT	0
Fixed	Initial address is used for the whole access	ADDRESSING_FIXED	1

accessMode	Selects between the two memory access modes shown below:

accessMode	Description	Name	Value
Non-DMA	Memory access is split into multiple accesses	NONDMA	0
DMA	Memory access is done with a DMA transaction	DMA	1

errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
data	Rx data buffer
moduleIDout	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes on page 153

C

```
int SD_Module_FPGAreadPCport(int moduleID, int nPCport, int* data, int
dataSize, int address, int addressMode, int accessMode);
```

C++

```
int SD_Module::FPGAreadPCport(int nPCport, int* data, int dataSize, int
address, SD_AddressingMode addressMode, SD_AccessMode accessMode);
```

Visual Studio .NET, MATLAB

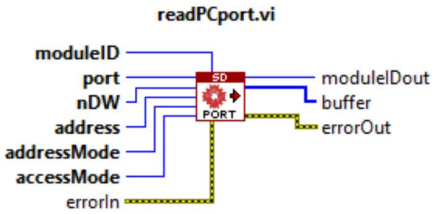
```
int SD_Module::FPGAreadPCport(int nPCport, int address, int[] data, SD_
AddressingMode addressMode, SD_AccessMode accessMode);
```

Python

```
SD_Module.FPGAreadPCport(nPCport, dataSize, address, addressMode, accessMode)
```

LabVIEW

`readPCport.vi`



M3601A

`Available: No`

5. 3. 5. 3 FPGAload

Loads a bitstream file generated using [4] Keysight M3602A FPGA Design Environment Software on page 157 to FPGA. (Option FP1 required).

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
fileName	File to load
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
moduleIDout	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes on page 153

C

```
int SD_Module_FPGAload(int moduleID, const char *fileName);
```

C++

```
int SD_Module::FPGAload(const char *fileName);
```

Visual Studio .NET, MATLAB

```
int SD_Module::FPGAload(string fileName);
```

Python

```
SD_Module.FPGAload(fileName)
```

LabVIEW

```
load.vi
```

M3601A

```
Available: No
```

5. 3. 5. 4 FPGAreset

Sends a reset signal to FPGA (Option FP1 required).

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by open on page 72
mode	Reset mode desired:

Option	Description	Programming Definitions Name	Value
Low	Low active reset	RESET_LOW	0
High	High active reset	RESET_HIGH	1
Pulse	Pulse reset	RESET_PULSE	2

errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
moduleIDout	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes on page 153

C

```
int SD_Module_FPGAreset(int moduleID, int mode);
```

C++

```
int SD_Module::FPGAreset(SD_ResetMode::SD_ResetMode mode);
```

Visual Studio .NET, MATLAB

```
int SD_Module::FPGAreset(SD_ResetMode mode);
```

Python

```
SD_Module.FPGAreset(mode)
```

LabVIEW

```
reset.vi
```

M3601A

```
Available: No
```

6 Error Codes

SD1 Error	Error No	Refer to keysightSD1.py for Definition.
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, see SD1 commands:
		`assignHardwareWithIndexAndSerialNumber()`
		`assignHardwareWithIndexAndSlot()`
		`assignHardwareWithUserNameAndSerialNumber()`
		`assignHardwareWithUserNameAndSlot()`
		`assignHardwareWithUserNameAndModuleID()`
		`compile()`
		`compilationErrorMessage()`
		`getNumberOfModules()`
		`getModuleIndex()`
		`getModuleByIndex()`
		`getModuleByName()`
		`load()`
		`pause()`
		`readIntegerConstantWithIndex()`
		`readIntegerConstantWithUserName()`
		`readDoubleConstantWithIndex()`
		`readDoubleConstantWithUserName()`
		`reset()`
		`resume()`
		`start()`
		`stop()`
		`writeIntegerConstantWithIndex()`
		`writeIntegerConstantWithUserName()`
		`writeDoubleConstantWithIndex()`
		`writeDoubleConstantWithUserName()`

SD_Error.INVALID_OBJECTID	-8008	Invalid objectID
SD_Error.INVALID_MODULEID	-8009	Invalid moduleID, see `getType()`.
SD_Error.INVALID_MODULEUSERNAME	-8010	Invalid moduleUsername
SD_Error.INVALID_HVIID	-8011	Invalid HVIID
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, see `newFromArrayDouble()`.

SD1 Error	Error No	Refer to keysightSD1.py for Definition.
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, see FFT().
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
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

SD1 Error	Error No	Refer to keysightSD1.py for Definition.
SD_Error.INVALID_END	-8061	Invalid end
SD_Error.INVALID_CYCLES	-8062	Invalid cycles
SD_Error.HVI_INVALID_NUMBER_ MODULES	-8063	Invalid number of modules on HVI
SD_Error.DAQ_P2P_ALREADY_RUNNING	-8064	DAQ P2P is already running
SD_Error.OPEN_DRAIN_NOT_SUPPORTED	-8065	Open drain not supported
SD_Error.CHASSIS_PORTS_NOT_ SUPPORTED	-8066	Chassis port not supported
SD_Error.CHASSIS_SETUP_NOT_ SUPPORTED	-8067	Chassis setup not supported
SD_Error.OPEN_DRAIN_FAILED	-8068	Open drain failed
SD_Error.CHASSIS_SETUP_FAILED	-8069	Chassis setup failed
SD_Error.INVALID_PART	-8070	Invalid part
SD_Error.INVALID_SIZE	-8071	Invalid size
SD_Error.INVALID_HANDLE	-8072	Invalid handle

7 References

Software

[1] Keysight SD1 SFP [Soft Front Panels] Software

[2] Keysight SD1 Programming Libraries

[3] Keysight M3601A Hard Virtual Instrument (HVI) Design Environment Software

[4] Keysight M3602A FPGA Design Environment Software

Hardware

[5] Keysight M3201A PXIe Arbitrary Waveform Generator, 500 MSa/s, 16 bit, 200 MHz

[6] Keysight M3202A PXIe Arbitrary Waveform Generator, 1 GSa/s, 14 bit, 400 MHz

[7] Keysight M3100A PXIe Digitizer: 100 MSa/s, 14 bit, 100 MHz

[8] Keysight M3102A PXIe Digitizer: 500 MSa/s, 14 bit, 200 MHz

[9] Keysight M3300A PXIe AWG and Digitizer Combination,
500 MSa/s, 16 bit and 100 MSa/s, 14 bit

[10] Keysight M3302A PXIe AWG and Digitizer Combination
500 MSa/s, 16 bit, and 500 MSa/s, 14 bit

Tested PCs

[11] Tested PC and PXI/AXIe Chassis Configurations

Edition 2, March, 2020
Printed In USA

M3201-90001
www.keysight.com

KEYSIGHT TECHNOLOGIES