M5300A PXIe RF AWG Modules

Four CH, DC-16 GHz RF, 2 GHz IBW, 14-bits resolution

KEYSIGHT

USER GUIDE

Notices

Copyright Notice

Trademarks

Manual Part Number

M5300-91002

Edition

1.8, August 2025

Available in electronic format only

Published by

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

Technology Licenses

U.S. Government Rights

Warranty

Safety Information

CAUTION

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

WARNING

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

Keysight Cybersecurity

Product and Solution Cybersecurity

Keysight is dedicated to ensuring the cybersecurity of its products and solutions. For detailed information, visit Product and Solution Cybersecurity.

Report a Product Cybersecurity Issue

If you encounter a cybersecurity issue with a Keysight product, report it immediately at: Product Cybersecurity Issue Reporting.

Responsible Disclosure Program

Keysight encourages responsible disclosure of security vulnerabilities. For more details, visit Responsible Disclosure Program.

Product Software Updates

Keysight releases periodic software updates to fix known defects, incorporate product enhancements, and address cybersecurity vulnerabilities, if any. Ensure that your product software is always up to date. To search for software updates for your product, visit the Keysight Technical Support page at: https://www.keysight.com/find/M5300A.

User Documentation

User documentation includes comprehensive information on features that may impact the secure deployment, use, and decommissioning of our products. This includes details on exposed network ports, the use of cryptography and authentication, and firmware security settings, if necessary.

Safety Summary

The following general safety precautions must be observed during all phases of operation of this instrument. Failure to comply with these precautions or with specific warnings or operating instructions in the product manuals violates safety standards of design, manufacture, and intended use of the instrument. Keysight Technologies assumes no liability for the customer's failure to comply with these requirements. Product manuals are provided on the Web. Go to www.keysight.com and type in your product number in the Search field at the top of the page.

WARNING **Do not use the device if it is damaged. Contact your Keysight sales representative for replacement of device.**

WARNING **Verify that all safety precautions are taken. Make all connections to the unit before applying power. Note the external markings described under "Safety symbols & instrument markings".**

WARNING **Do not operate the device in an explosive atmosphere or wet environments. Do not operate the instrument around flammable gases or fumes, vapor, or wet environments.**

WARNING **Do not install substitute parts or perform any unauthorized modification to the product. Return the product to a Keysight Sales and Service Office to ensure that safety features are maintained.**

CAUTION If the device is used in a manner not specified by the manufacturer, the protection provided by the device may be impaired.

CAUTION Do not attempt to clean the device. If cleaning the card is absolutely necessary, Keysight recommends cleaning the card with a lightly dampened cloth while the module is in a de-energized condition. Do not re-energize the card until it is completely dry.

NOTE **The safety of any system incorporating the equipment is the responsibility of the assembler of the system.**

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.

CAUTION

The measuring terminals on this instrument are designed to be used with external signals described in Mains-isolated secondary circuits, but NOT with external signals described in Categories II, III, and IV. The input of this instrument cannot be connected to the mains.

Measurement Category	Description
II	Applicable to testing and measuring circuits connected directly to utilization points (socket outlets and similar points) of the low-voltage mains installation. Example: Measurements on MAINS CIRCUITS of household appliances, portable tools and similar equipment and on the consumer side only of socket-outlets in the fixed installation.
III	Applicable to test and measuring circuits connection to the distribution part of the building's low-voltage mains installation. To avoid risks caused by the hazards arising from these higher short-circuit currents, additional insulation and other provisions are required. Example: Measurements on distribution boards (including secondary meters), photovoltaic panels, circuit breakers, wiring, including cables, bus-bars, junction boxes, switches, socket-outlets in the fixed installation, equipment for industrial use and some other equipment such as stationary motors with permanent connection to the fixed installation. NOTE: For equipment that is part of a fixed installation, the fuse or circuit breaker of the installation can be considered to provide adequate protection against short-circuit currents.
IV	Applicable to test and measuring circuits connected at the source of the building's low-voltage mains installation. Due to these high short-circuit currents, which can be followed by a high energy level, measurements made within these locations are extremely dangerous. Great precautions shall be made to avoid any chance of a short circuit. Example: Measurements on devices installed before the main fuse or circuit breaker in the building installation.

WARNING

To ensure mandatory safety requirements are being met, the module must be installed in a chassis which has been certified and marked by a Nationally Recognized Testing Lab (such as CSA, UL, TUV, ETL, and so on) in which all the means of protection are properly implemented. Note that CE marking alone is not adequate.

WARNING

Servicing of the modules must be performed by qualified personnel only. To avoid electrical shock, do not perform any servicing manually. Return the module to Keysight Service Center.

WARNING **For safety reasons, only Keysight approved equipment and accessories should be used with the module.**

NOTE **Position chassis to ensure easy access to remove the modules.**

Environmental Conditions

This instrument is intended for “indoor use” only.

The following table shows the environmental requirements and the corresponding characteristics for the product.

Environmental Requirements	General Characteristics
Temperature	Operating condition: 0°C to 45°C Storage condition: -40°C to 70°C
Maximum Relative Humidity (non-condensing)	Type tested, 95% RH up to 40°C, decreases linearly to 40% RH at 45°C
Altitude	Operating condition: Up to 10,000 ft (3048m) Storage condition: Up to 15,000 ft (4572m)
Pollution degree*	Pollution Degree 2

* See table below for Pollution Degree definitions

Pollution Degree	Description
1	No pollution or only dry, non-conductive pollution occurs. The pollution has no influence. Example: A clean room or climate-controlled office environment.
2	Normally only dry non-conductive pollution occurs. Occasionally a temporary conductivity caused by condensation may occur. Example: General indoor environment.
3	Conductive pollution occurs, or dry, non-conductive pollution occurs which becomes conductive due to condensation which is expected. Example: Sheltered outdoor environment.

Safety symbols & instrument markings

Safety Symbol / Instrument Marking	Description
⚠	Hot surface. The metallic panels may get warm after powering on the equipment.
⚠	The instruction manual symbol. The product is marked with this warning symbol when it is necessary for the user to refer to the instructions in the manual.
ICES/NMB-001 CE ISM GRP 1-A	The CE mark is a registered trademark of the European Community.
UK CA	The UK mark is a registered trademark of the European Community.
	The RCM mark is a registered trademark of the Australian Communications and Media Authority.
	The KC mark is the Korean certification mark. This equipment is Class A suitable for professional use and is for use in electromagnetic environments outside of the home.
	Electro Static Discharge. Attach ESD protective wrist strap to avoid damage by direct contact with the equipment.
40	China Restricted Substance Product Label. The EPUP (environmental protection use period) number in the center indicates the time period during which no hazardous or toxic substances or elements are expected to leak or deteriorate during normal use and generally reflects the expected useful life of the product.
ccr.keysight@keysight.com	This is the Keysight email address required by EU directives applicable to our product.

Compliance and Environmental Information

Table 1 Compliance and Environmental Information

Safety Symbol	Description
	The crossed out wheeled bin symbol indicates that separate collection for waste electric and electronic equipment (WEEE) is required, as obligated by the EU DIRECTIVE and other National legislation. Refer to keysight.com/go/takeback to understand your Trade in options with Keysight in addition to product takeback instructions.

Declaration of Conformity

Declarations of Conformity for this product and for the Keysight products may be downloaded from the Web. Go to http://www.keysight.com/go/conformity.

You can then search by product number to find the latest Declaration of Conformity.

References to other documents

Document name	Location on installed machine
M5300x C++ API Help	<root-dir>\Program Files\Keysight\M5300x\Help\html\index.html
M5300x HVI Add-on Help	<root-dir>\Program Files\Keysight\M5300x\Help\Keysight_KtM5300xHvi_Fx45
M5300x IVI.Net Help	<root-dir>\Program Files\Keysight\M5300x\Help\Keysight_KtM5300x_Fx45
M5300x IVI-C Help	<root-dir>\Program Files\Keysight\M5300x\Help\KtM5300x
M5300x Python API Help	<root-dir>\Program Files\Keysight\M5300x\python\html\index.html
M5300A PXIe Hi-Speed AWGs BSP Help	Accessible within the KF9000B PathWave FPGA environment

Document name	K.com URL
M5300A PXIe RF AWG Modules Startup Guide	https://www.keysight.com/find/M5300A-TechSupport/
M5200A PXIe Digitizer Modules User Guide	https://www.keysight.com/find/M5200A-TechSupport/
M5201A PXIe Downconverter Modules User Guide	https://www.keysight.com/find/M5201A-TechSupport/
KS2201A PathWave Test Sync Executive User Manual	https://www.keysight.com/in/en/support/KS2201A/pathwave-test-sync-executive.html
KF9000B PathWave FPGA User Documentation	https://www.keysight.com/in/en/support/KF9000B/pathwave-fpga.html

Contents

1. Overview on M5300A High-Speed AWG Modules

Section 1.1: About M5300A AWG modules

The M5300A RF AWG module supports four directly synthesized RF channels in 2 PXIe slots with up to 18 GHz of RF and 2 GHz of instantaneous bandwidth. The channels are fully coherent across the module and across the entire system for applications in quantum computing and aerospace applications. The module supports the ability to load data into the module directly over the PCIe backplane. You can also create custom IP and processing algorithms in the FPGA.The module supports the PathWave FPGA and PathWave Test Sync Executive platforms for developing and configuring IP blocks and synchronous system event control.

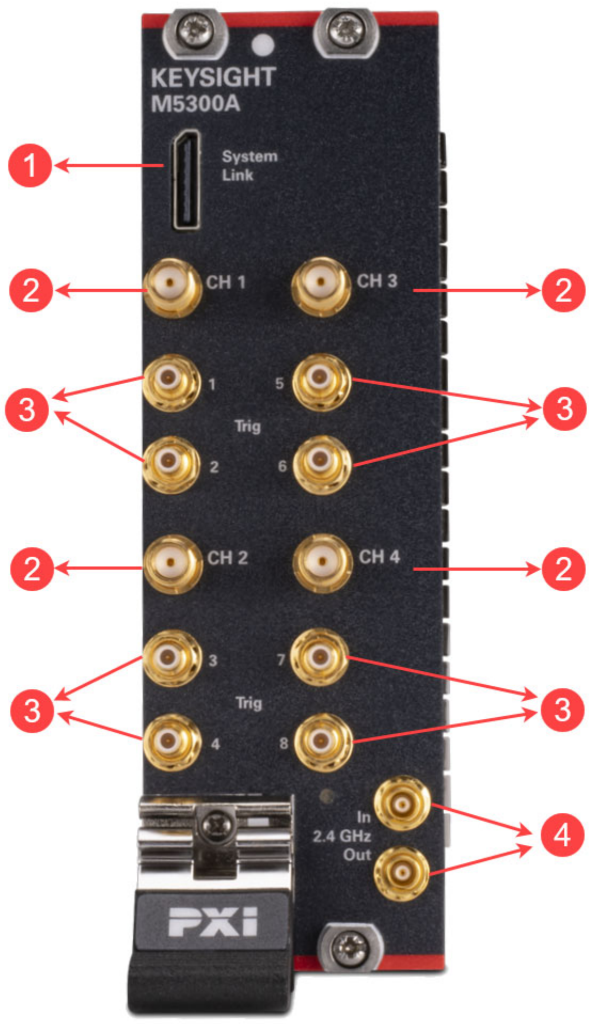


Figure 1 Front Panel view of the M5300A High Speed AWG module

The M5300A is a two-slot PXIe module with four high-speed (0 to 18 GHz) AWG outputs along with clock input-output, and eight triggers. Figure 1 shows the front panel of an M5300A High Speed AWG module, which comprises of the following connectors:

1 SystemLink Connector (1)

The top connector is a x4 System Link connection. It has eight bi-directional LVDS pairs (1.2 Gbps), unused RX and TX SERDES pairs (6 Gbps), four single-ended control lines, and a power supply (+5 V at 0.5 A). System Link also has two handshaking signals called TX_DISABLE_OUT and TX_DISABLE_IN. If these signals are not properly driven, the data signals in connector are forced to be inputs (tri-state). This connector can be used to connect to a system timing module, other cards, or to a breakout board.

2 AWG outputs

The analog outputs are created with a DAC clocked in GHz, with a special double-data rate mechanism inside the DAC doubles the effective sampling rate, thereby providing a Nyquist bandwidth of 19.2 GHz. The 19.2 GHz sample clock is derived from either the 100 MHz PXI clock, DSTARA, or an external 2.4 GHz clock using onboard clock multipliers.

3 Bi-directional SMB Triggers

The module allows you to control eight single-ended signals on the SMB connectors. These signals swing 0-3.3 V and can drive 50-ohm loads (dividing the voltage in half). The signals toggle up to 100 MHz with 3.3 ns edge alignment.

4 2.4 GHz Analog Reference Clock (In & Out)

There are two connectors for 2.4 GHz clkout and 2.4 GHz clkin - these are used for sharing a coherent frequency reference across multiple cards in one or more chassis.

The block diagram shown in Figure 2 depicts the various elements in the M5300A modules that help in generating high-speed RF waveforms.

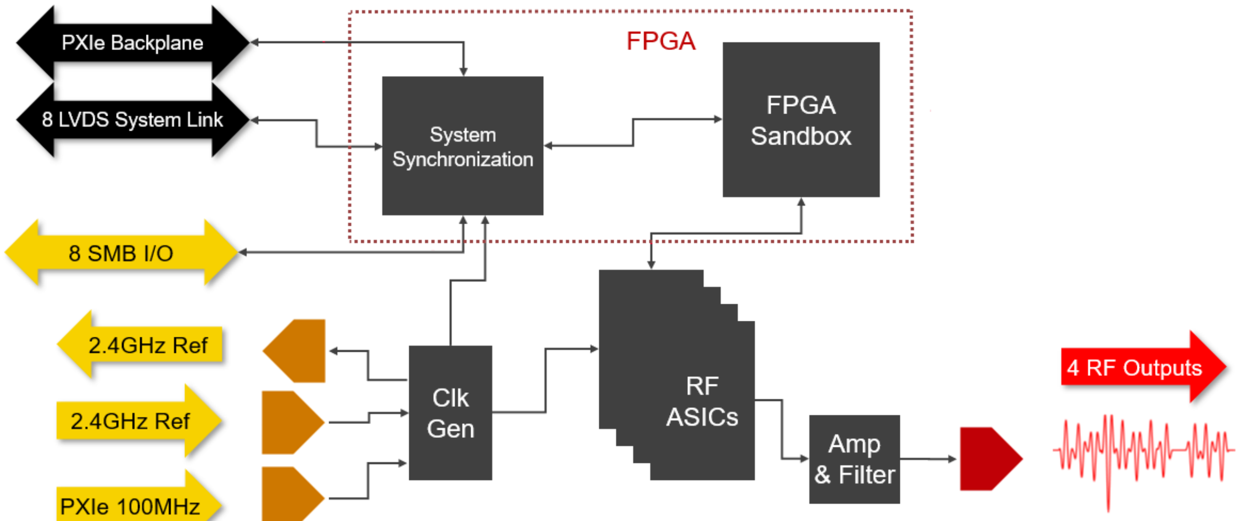


Figure 2 Block diagram representation of the M5300A High Speed AWG module

The M5300A generates three output waveform types, namely Standard, Baseband and Modulated. Figure 3 shows an example of how the M5300A RF AWG can use IQ modulation to create a 2.4 GHz signal anywhere within the frequency range of 0 to 18 GHz.

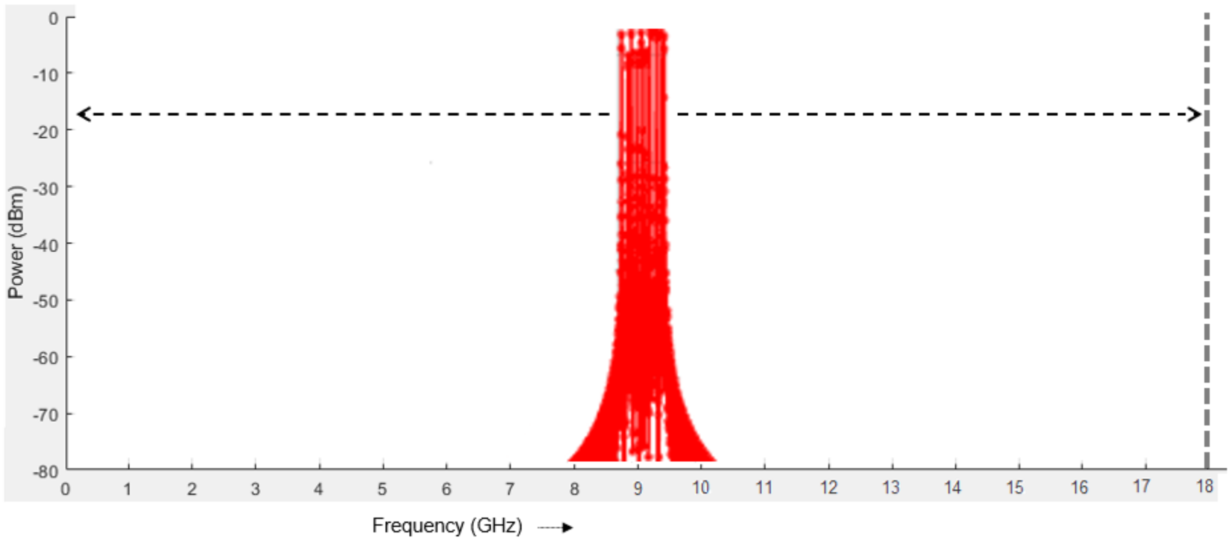


Figure 3 Example of using IQ modulation in M5300A RF AWG module

While the Front Panel connectors have been described above, the M5300A plugs into two slots of the High-performance PXIe chassis (M9046A) with two connectors (per slot). The PXI backplane provides eight bus trigger lines (PXI_TRIG[7:0]) and one point-point trigger line (PXI_STAR). These lines allow communication between cards. The PXI_TRIG lines operate at low MHz with open-drain outputs pulled up to 5 V. These lines can be controlled using either the SFP or the APIs. See "Configuring the 'PXI IO' bank" on page 130 for more information.

Other than level-shifting to the PXI triggers and generating/distributing clocks, the FPGA handles all communication to onboard chips and to the PXIe backplane. The main FPGA interfaces are x8 PCIe to the backplane, SERDES and LVDS links to the four DACs, controls for the clocks, and many small peripherals (power sequencers, temp sensors, and so on).

The Digital to Analog Converters within the M5300A modules allows 2 GHz of IF (signal) bandwidth upconverted to a frequency between 0-18 GHz. Each upconverter includes an amplifier with differential input and differential output to help level the output frequency response to 16 GHz.

Table 2 lists certain specifications for the M5300A module.

Table 2 Specifications for the M5300A module

Characteristic	Value
Operating Voltage	+3.3 V and +12 V
Operating current (max.)	6.3 A (3.3 V), 2.4 A (12 V)
Total Power dissipation	50 W
Measurement Category	Non-CAT

CAUTION The measuring terminals on this instrument are designed to be used with external signals described in Mains-isolated secondary circuits, but NOT with external signals described in Categories II, III, and IV. The input of this instrument cannot be connected to the mains.

Table 3 **Description of Measurement Category II, III and IV**

Measurement Category	Description
II	Applicable to testing and measuring circuits connected directly to utilization points (socket outlets and similar points) of the low-voltage mains installation. Example: Measurements on MAINS CIRCUITS of household appliances, portable tools and similar equipment and on the consumer side only of socket-outlets in the fixed installation.
III	Applicable to test and measuring circuits connection to the distribution part of the building's low-voltage mains installation. To avoid risks caused by the hazards arising from these higher short-circuit currents, additional insulation and other provisions are required. Example: Measurements on distribution boards (including secondary meters), photovoltaic panels, circuit breakers, wiring, including cables, bus-bars, junction boxes, switches, socket-outlets in the fixed installation, equipment for industrial use and some other equipment such as stationary motors with permanent connection to the fixed installation. NOTE: For equipment that is part of a fixed installation, the fuse or circuit breaker of the installation can be considered to provide adequate protection against short-circuit currents.
IV	Applicable to test and measuring circuits connected at the source of the building's low-voltage mains installation. Due to these high short-circuit currents, which can be followed by a high energy level, measurements made within these locations are extremely dangerous. Great precautions shall be made to avoid any chance of a short circuit. Example: Measurements on devices installed before the main fuse or circuit breaker in the building installation.

To know about the functionality of the M5300A SFP, refer to "Using M5300A High-Speed AWG Modules SFP" on page 99.

1.1.1: About M5300A module's timebase

The M5300A module generates an independent internal LO for each of the four channels. The time reference for the LO is selectable and can be sourced from either an internal 2.4 GHz Analog Reference Clock derived from the PXI chassis 100 MHz System Reference Clock or from an external 2.4 GHz Analog Reference Clock. The internal Analog Clock is easier to use than the external Analog Clock because it does not require external clock generation and distribution. However, the best absolute phase noise and coherency between modules is achieved when using the external Analog Clock.

System Clock Architecture

The M5300A modules are used in Keysight PXI-based systems that employ a variety of clock configuration options that trade off phase coherency and system complexity. Those different clock configurations are described in "Clock Configuration" on page 22.

Regardless of the system clock configuration used, the clock generation and distribution must be configured and aligned. Some of this configuration and alignment takes place within the PXI chassis and System Synchronization Modules (M9032A) and some of it takes place within each peripheral module (such as the M3202A AWG 1G module). In general, all clock configuration and alignment procedures must be performed at the system level before they are performed at the peripheral module level.

NOTE

Re-configuring the system-level clocks can cause disruptions and glitches in the both the 100 MHz System Clock and 2.4 GHz Analog Clock. Disruptions in these system-level clocks while a peripheral module is performing certain operations can cause the module to become unstable or unresponsive. System clock re-configuration should only be performed immediately after system power on or after all peripheral modules have been reset. Modules can be reset through the SFP or API.

For description on instructions for resetting modules, refer to “Resetting to Default Factory Settings” on page 131.

When either the SFP or API connects to a module driver for the first time after a power cycle, the module’s Analog Clock source is defaulted to internal, and all the module clocks are aligned. Subsequent disconnections and re-connections to the module driver do not interfere with the module’s clock configuration or alignments unless explicitly forced to do so using the “Reset” driver connection option.

Procedure for System Clock Initialization

The system-level clock initialization procedure comprises configuring and aligning the system clocks, which are external to the M5300A module. Depending on which clock configuration option you use, this procedure can range from very simple to extremely complex. Except for the simplest configurations, system clock initialization is performed using automated procedures provided by the PathWave Test Synchronization Executive. You can find more detail about using those automated system clock initialization procedures in the *KS2201A PathWave Test Sync Executive User Manual*.

Procedure for Module Clock Initialization

The module-level clock initialization procedure comprises configuring and aligning the clocks which are internal to the M5300A module. The module initialization procedure you need to perform depends on; 1) which system clock configuration option you use and 2) whether you care about the precise relative phase alignment between the M5300A module you are initializing and other peripheral modules.

The module clock initialization procedures are relatively simple and can be performed either through the module SFP interface, automated by the PathWave Test Synchronization Executive or automated yourself using the module API commands.

Clock Configuration

The various clock configuration options configure the module to use either the internal 2.4 GHz Analog Clock or the external 2.4 GHz Analog Clock. The procedure to select the Analog Clock source using the SFP interface is described in “Configuring Clock source and alignment settings” on page 116. The details to select the Analog Clock source using the PathWave Test Synchronization Executive is described in the *KS2201A PathWave Test Sync Executive User Manual*. The module API command to select the Analog Clock source using the API interface is described in the IVI-Help files and M5300x Python Help file.

NOTE When configuring the Analog Clock source using either the GUI or API for multi-module systems that daisy-chain the external Analog Clock through one of the modules, the order in which you configure each module is important. Upstream modules in daisy-chained configuration must be configured before downstream modules to prevent disruption or glitching of downstream clocks.

Inter-Module Phase Alignment

Single-module applications that do not require precise inter-module phase alignment do not require use of HVI synchronization or any additional clock alignment actions beyond simply selecting the desired Analog Clock source. However, multi-module applications that depend on synchronized LO phase between modules require both HVI and additional clock alignment and de-skew procedures.

Every time the M5300A powers on, it needs to perform a module clock alignment. The first time the module clock alignment is performed (an Initial Clock Alignment), the LO's residual random skew error (phase error) can be up to ±52 ps relative to the System Time Reference provided by the PathWave Test Sync Executive. You may correct this skew error as part of the normal User De-skew procedure, which corrects for your application's other cable and fixture skew errors.

Each time the module performs an Initial Clock Alignment, it could result in a residual skew error of ±52 ps, which would then need to be corrected with a User Skew adjustment.

You could choose to perform an Initial Clock Alignment after each power cycle, but that can be a time-consuming process. Alternatively, you can perform a Clock Re-Alignment, provided that you have performed an Initial Clock Alignment at least once before with the same exact system hardware and clock configuration. Performing a Clock Re-Alignment does not result in a random residual skew error and does not require an adjustment to the User Skew settings.

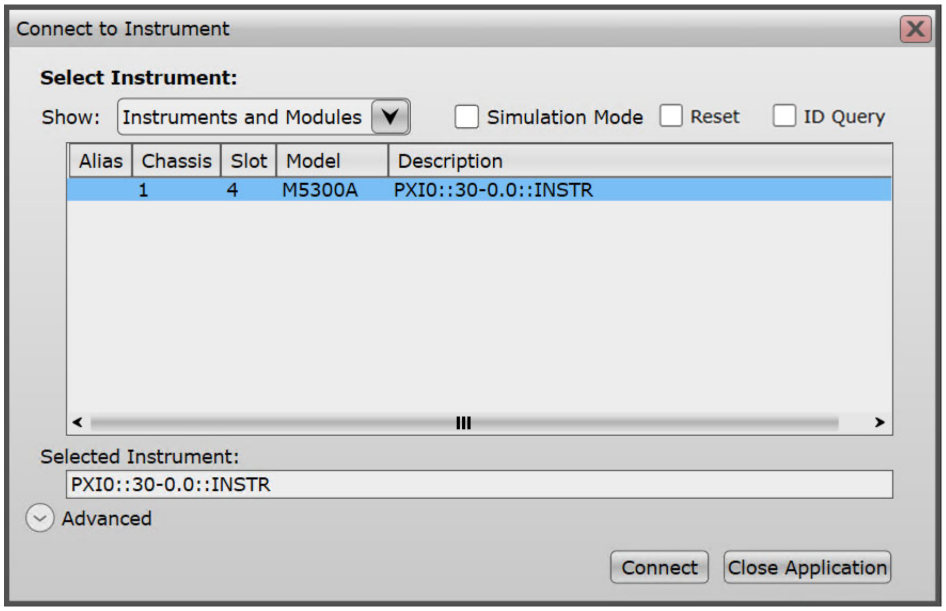
NOTE The Clock Re-Alignment procedure is very sensitive to the propagation delays of all the system and module clocks. Any change in the hardware or clock configuration will require a new Initial Clock Alignment. Simply replacing a defective MCX Analog Clock cable or moving a module to different chassis slot will require a new Initial Clock Alignment, as would a significant change to the system's ambient temperature.

Module clock alignment can be performed using the PathWave Test Synchronization Executive, the module's API or module's SFP. Module alignment of large systems is only practical using the PathWave Test Synchronization Executive software. For small systems of only a few modules, following procedures can be used.

Procedure for Initial Clock Alignment

Must be performed after any hardware or clock configuration change.

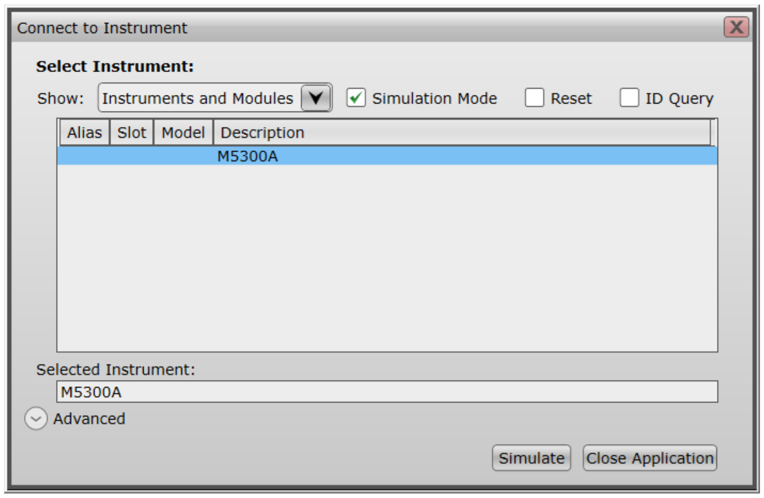
1 Open all module GUI or API driver connections.
   - GUI controls



Alias	Chassis	Slot	Model	Description
	1	4	M5300A	PXI0::30-0.0::INSTR

Selected Instrument: PXI0::30-0.0::INSTR

Figure 4 M5300x SFP software Connect to Instrument window (with one or more modules connected)



Alias	Slot	Model	Description
			M5300A

Selected Instrument: M5300A

Figure 5 M5300x SFP software Connect to Instrument window (Simulation Mode)

- API commands

```
driver = keysight_ktm5300x.KtM5300x(resource, query, reset, options)
```

2 Configure module to use the desired Analog Clock.

- GUI controls

  i From the main window of the M5300x SFP, select a reference clock source from the **Analog Clock in** drop-down field:

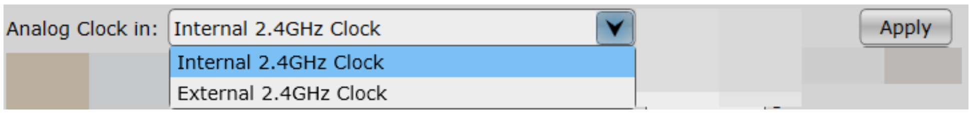


Figure 6 Configuring the desired analog clock in the M5300x SFP

- **Internal 2.4GHz Clock** (default option)
- **External 2.4GHz Clock**

ii Click **Apply**.

- API commands

```
driver.clock.clock_source = keysight_ktm5300x.ClockSource.INTERNAL
driver.clock.clock_source = keysight_ktm5300x.ClockSource.EXTERNAL
```

Modules in Daisy-chained clock configurations must be configured in a specific order to prevent disruption of downstream clocks.

3 Wait for system temperature to stabilize.

This could take up 30 minutes, depending on the complexity of the system and how recent the power distribution of the system has changed. You may skip this waiting step if you can tolerate the potential 26 ps inter-channel skew errors.

4 Perform an Initial Clock Alignment.

- GUI controls

i From the main window of the M5300x SFP, select **Initialize** from the **Set Clock Alignment** drop-down options to perform clock alignment configuration.

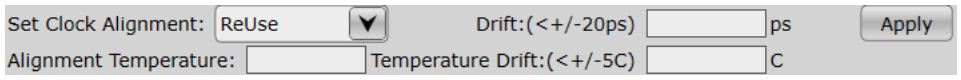


Figure 7 Configuring the desired clock alignment configuration in the M5300x SFP

ii Click **Apply**.

NOTE Select **Initialize** if you have not previously performed an alignment using this option for the current hardware & clock configuration. This option may introduce small random skew errors, where you must perform de-skew manually.

Notice that the **Drift**, **Alignment Temperature** and **Temperature Drift** values are auto-populated when you click **Apply** to set the Clock Alignment option. For more information about these properties, see "Configuring Clock source and alignment settings" on page 116.

- API commands

```
driver.clock.invalidate_current_alignment()
```

The `driver.clock.invalidate_current_alignment()` method must be run to either reset the clock alignment settings or if the Clock Alignment is being set to **Initialize** using the SFP controls. This way, a new alignment point is set, which could be different than the last alignment point. Consequently, you are required to perform once again the deskew of the outputs and inputs of the system.

```
driver.clock.align()
```

The `driver.clock.align()` method performs the clock alignment. In this case, since the current alignment is made invalid using the `driver.clock.invalidate_current_alignment()` method or has never been done, this method automatically performs an "initialize" alignment.

```
driver.clock.alignment_temperature(float)
```

`driver.clock.drift_temperature(float)`

`driver.clock.drift_time(str)`

Note that the order in which the modules are aligned is important. Modules should be deskewed in an order where the modules that are receiving an external clock from another module should be deskewed after the module that is sourcing the 2.4 GHz analog clock. That is, all sources of the 2.4 GHz analog clock should be deskewed before receivers of the 2.4 GHz analog clock.

5 De-skew residual skew error from and Initial Clock Alignment and application's unique cable and fixture skew errors.

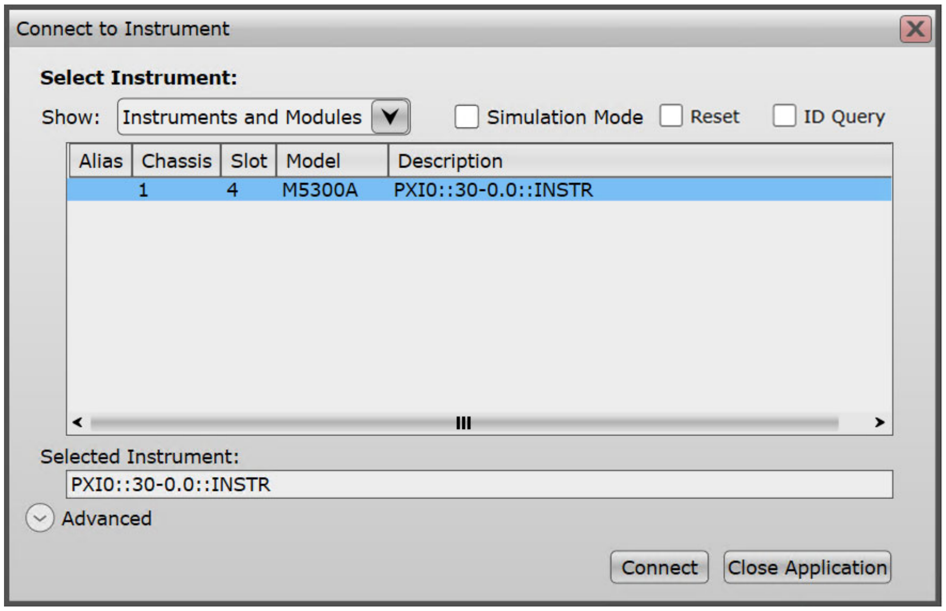
6 Record User Skew values for future reference.

Procedure for Clock Re-Alignment

May be performed after any power cycle for which now hardware or clock configuration changes have occurred since the previous Initial Clock Alignment was performed.

1 Open all module GUI or API driver connections.

- GUI controls



Alias	Chassis	Slot	Model	Description
	1	4	M5300A	PXI0::30-0.0::INSTR

Selected Instrument: PXI0::30-0.0::INSTR

Figure 8 M5300x SFP software Connect to Instrument window (with one or more modules connected)

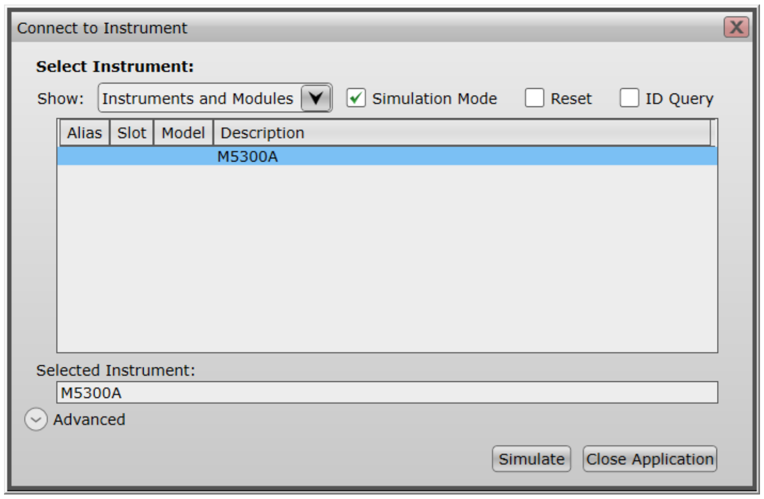


Figure 9 M5300x SFP software Connect to Instrument window (Simulation Mode)

- API commands

```
driver = keysight_ktm5300x.KtM5300x(resource, query, reset, options)
```

2 Configure module to use the desired Analog Clock.

- GUI controls

  i From the main window of the M5300x SFP, select a reference clock source from the **Analog Clock in** drop-down field:

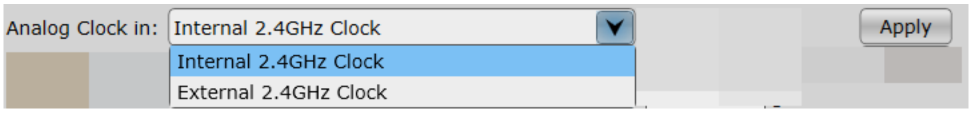


Figure 10 Configuring the desired analog clock in the M5300x SFP

- **Internal 2.4GHz Clock** (default option)
- **External 2.4GHz Clock**

  ii Click **Apply**.

- API commands

```
driver.clock.clock_source = keysight_ktm5300x.ClockSource.INTERNAL

driver.clock.clock_source = keysight_ktm5300x.ClockSource.EXTERNAL
```

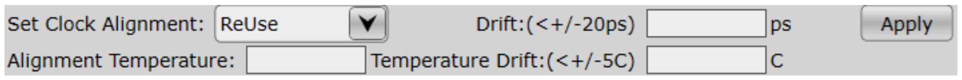
Modules in Daisy-chained clock configurations must be configured in a specific order to prevent disruption of downstream clocks.;

3 Wait for system temperature to stabilize.

This could take up 30 minutes, depending on the complexity of the system and how recent the power distribution of the system has changed. You may skip this waiting step if you can tolerate the potential 52 ps inter-channel skew errors.

4 Perform an Initial Clock Alignment.

- GUI controls
  - i From the main window of the M5300x SFP, select **ReUse** from the **Set Clock Alignment** drop-down options to perform clock alignment configuration.



Set Clock Alignment: ReUse Drift:(<+/-20ps) ps Apply
Alignment Temperature: Temperature Drift:(<+/-5C) C

Figure 11 Configuring the desired clock alignment configuration in the M5300x SFP

  - ii Click **Apply**.

NOTE Select this option if you previously performed an alignment using the **Initialize** option for the current hardware and clock configuration. This option preserves the inter-channel skews set by the prior alignment where the **Initialize** option was used.

Notice that the **Drift**, **Alignment Temperature** and **Temperature Drift** values are auto-populated when you click **Apply** to set the Clock Alignment option. For more information about these properties, see "Configuring Clock source and alignment settings" on page 116.

- API commands

`driver.clock.align()`

The `driver.clock.align()` method performs the clock alignment wherein, the previous alignment setting is used and by default, a "Reuse" alignment is performed. In this scenario, do not use the `driver.clock.invalidate_current_alignment()` method to perform clock re-alignment.

`driver.clock.alignment_temperature(float)`

`driver.clock.drift_temperature(float)`

`driver.clock.drift_time(str)`

Note that the order in which the modules are aligned is important. Modules should be deskewed in an order where the modules that are receiving an external clock from another module should be deskewed after the module that is sourcing the 2.4 GHz analog clock. That is, all sources of the 2.4 GHz analog clock should be deskewed before receivers of the 2.4 GHz analog clock.

5 Apply User Skew values that were recorded after the previous Initial Clock Alignment was performed.

1.1.2: List of products required / supported with M5300A modules

You require the following (recommended) Keysight products to achieve precision and control when performing measurements. Visit the corresponding product pages on www.keysight.com to procure and refer to the respective documentation to understand how to use these modules/products.

Table 4 List of supported equipment

Required equipment	Recommended Model / Part number(s)
PXIe Digitizer modules	M5200A
PXIe Downconverter modules	M5201A
PXIe Baseband AWG modules	M5301A
PXIe LVDS Digital IO modules	M5302A
System Sync modules	M9032A / M9033A
PXIe Chassis: High power, 18 slots, 24 GB/s, requires option -QS2	M9046A
Controller options (use one of the following options)	
PXIe High-Performance Embedded Controller	M9037A
External Controller*	Refer to http://www.keysight.com/find/PXIAXIePCBIOSandWindowsSettings.
Other accessories	
PXIe 5-channel Source / Measure Unit, 100 pA	M9614A
PXIe 5-channel Precision Source/Measure Unit, 500 kSa/s, 10 pA, 30 V, 500 mA	M9615A
PXIe System Modules and Cable Interface	M902xA
PCIe High-Performance Host Adapter	M9049A
Streamline Vector Network Analyzer, 9 kHz to 4.5 GHz, 2-port	P5000B
Infiniium UXR-Series Oscilloscope: 16 GHz, 4 Channels	UXR0164A

* You may use any External Controller of your choice. Keysight recommends using the HP Z8 G4 Workstation to scale up your system for multi-chassis operations.

NOTE For information about the recommended BIOS and Windows system settings in the External Controller being used along with one or more M5000-series modules, refer to http://www.keysight.com/find/PXIAXIePCBIOSandWindowsSettings.

To connect the HP Z8 G4 workstation as an external controller to the M9046A PXIe chassis, you require the following additional products:

- M9049A PCIe High Performance Host Adapter
- M9023A PXIe High Performance System Module

- Y1201A PCIe cable

The M5000-series modules require specific Keysight cables that have been customized for connectivity with the M9032x SSM modules and M9046A PXIe high-performance chassis. Table 5 lists the cables that you may procure for the configuration of a single/multi-chassis multi-system setup.

Table 5 Customized cables for use with M5300A PXIe RF AWGs

Cable type	Specification	Model number
System Sync/Link	x4-x4, 0.5M	Y1320A
	x4-x4, 1.0M	Y1321A
	x8-2, x4, 0.5M	Y1323A
	x8-2, x4, 1.0M	Y1324A
	x8-x8, 0.5M	Y1326A
	x8-x8, 1.0M	Y1327A
	x8-x8, 2.0M	Y1329A
MCX(m) to MCX(m)	phase stable, 0.3m	Y1330A
	phase stable, 1.0m	Y1331A
	phase stable, 2.0m	Y1332A
SMA(m)-SMP(f)	0.3m	Y1333A
Cat-5	RJ-45 connectors (to interconnect chassis)	

NOTE

Optimum phase coherency and stability between analog channels is achieved by distributing the external Analog Clock over the MCX to MCX cables and power dividers in a balanced star configuration. You may use cables with different lengths, but the total propagation delay of the Analog clock from its common source (usually the Leader chassis) to each module Ext. Ref. clock input should be equal.

The complexity of your quantum computing system shall determine the quantity of modules and cables that should be procured.

For more information about the cables that you wish to procure, contact Keysight Support.

Section 1.2: About M5300x software features

The Keysight M5300x software, which comprises of drivers, programming libraries and Software Front Panel for the M5300A module, provides a comprehensive platform to perform the basic operations pertaining to the High-Speed AWG modules. The M5300x software also supports the Hardware FPGA reprogramming, which is done using the PathWave FPGA Board Support Package. Moreover, the M5300x software API is powered by the KS2201A PathWave Test Sync Executive software for performing real-time operations and HVI sequencing.

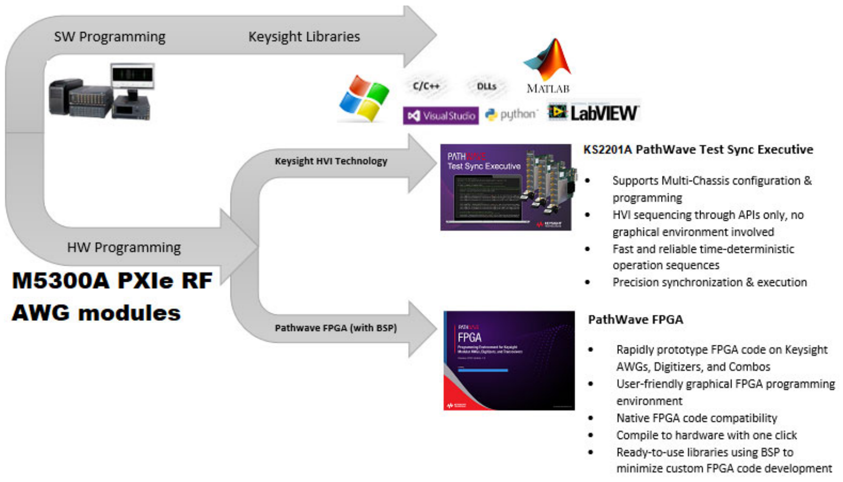

SW Programming
Keysight Libraries
C/C++
DLLs
MATLAB
Visual Studio
python
LabVIEW
Keysight HVI Technology
KS2201A PathWave Test Sync Executive
Supports Multi-Chassis configuration & programming
HVI sequencing through APIs only, no graphical environment involved
Fast and reliable time-deterministic operation sequences
Precision synchronization & execution
HW Programming
M5300A PXIe RF AWG modules
Pathwave FPGA (with BSP)
PathWave FPGA
Rapidly prototype FPGA code on Keysight AWGs, Digitizers, and Combos
User-friendly graphical FPGA programming environment
Native FPGA code compatibility
Compile to hardware with one click
Ready-to-use libraries using BSP to minimize custom FPGA code development

Figure 12 Model depicting the M5300x software capabilities

1.2.1: Waveform formats in the M5300x SW

You may load waveform files using the M5300x SFP or API.

To load waveform files using the SFP, see “Configuring Waveform Memory and Queue” on page 121.

The `createFromFile (file_path, waveform_id, type)` method in the M5300x SW API for loading a file from the disk is shown below, along with the `waveformType` enumeration that tells the API how to handle the waveform.

C#

```
ArbitraryWaveform.createFromFile(string filePath, int waveformId, WaveformFileTypeEnum
waveformType)
```

Python

```
arbitrary_waveform.create_from_file(filePath, waveformId, waveformType)
```

Table 6 Enum value to handle data

Value for waveformType enum	Data handling
`Analog`	The file is treated as real value only csv or binary file.
`IQ`	The file is treated as having IQ data in csv, binary or wfm format.
`Digital`	The file is treated as having integer IQ data in csv or binary format.

NOTE

The Signal Studio waveform file (WFM format) only supports the IQ waveform type. The waveform file is generated using the following software licenses - N7631APPC and N7631RAWC.

The M5300A module requires that data be presented in 16 sample increments. If the waveform presented to the driver is not an integer multiple of 16 samples, the driver 'pads 0s' to the waveform and returns a warning in the error queue that the waveform has been 'zero padded'.

Binary Floating Point Data

Values are stored in IEEE 754 double precision (64 bit) floating point values normalized between -1 and 1. On loading these values are converted to the nearest sample value. The floating point values are laid out in an array. For IQ waveforms the first value in the pair is the I value followed by the Q value.

Real values (ANALOG)

```
Point 0
Point 1
Point 2
...
Point N-1
```

IQ Data (IQ)

```
Point 0 I
Point 0 Q
Point 1 I
Point 1 Q
Point 2 I
Point 2 Q
...
Point N-1 I
Point N-1 Q
```

Binary Digital Data

This format contains the IQ pairs of data to be sent to the DAC. The data is interpreted as being stored as 16 bit signed integers (ranging from -32768 to 32767).

Integer Data (DIGITAL)

```
Point 0 I
Point 0 Q
Point 1 I
Point 1 Q
Point 2 I
Point 2 Q
```

```
...
Point N-1 I
Point N-1 Q
```

Invalid Values

The following values are considered invalid and if encountered, the driver will throw an exception.

- Infinity
- NaN
- Values out of the range (-1, 1)

CSV Data (CSV Floating Point Data)

For real valued waveforms, the format is simply a list of floating point values between -1 and 1.

Real Values (ANALOG)

```
Point 0
Point 1
Point 2
...
Point N - 1
```

For IQ waveforms, the format is:

IQ Data (IQ)

```
Point 0 I, Point 0 Q
Point 1 I, Point 1 Q
Point 2 I, Point 2 Q
...
Point N-1 I, Point N-1 Q
```

Note that the CSV format has the same rules about invalid values as the binary format.

CSV Digital Data

The CSV digital format follows the same rules as the floating point data except that data must be in the form of 16 bit signed integers (ranging from -32768 to 32767). Values that are above or below the possible values of samples will cause an exception to be thrown.

Integer Data (DIGITAL)

```
Point 0 I, Point 0 Q
Point 1 I, Point 1 Q
Point 2 I, Point 2 Q
...
Point N-1 I, Point N-1 Q
```

NOTE

The CSV formats support single line comments in starting with '!', '%' or '#' and ignoring any characters after that character until the end of the line. Valid CSV data can occur prior to the comment character and will be parsed normally by the SW.

1.2.2: Creating Waveform files in M5300x SW drivers

Table 7 lists the conditions to create a waveform file that is recognized and processed by the M5300x SW drivers.

Table 7 Conditions for creating the Waveform types in M5300x SW

Waveform type	Waveform creation method using...	Value range in the array / file	M5300x API to use
Analog	With numpy array (one dimensional)	between -1.0 to 1.0	`create()`
	With .bin File (single column)		`create_from_file()`
	With .csv File (single column)		`create_from_file()`
Digital	With numpy array (one dimensional)	between -32768 to 32767	`create_int16()`
	With .bin File (single column)		`create_from_file()`
	With .csv file (two columns)		`create_from_file()`

NOTE

When you load an analog waveform using the *create_from_file()* API, the original values from the .bin file are inserted at even rows, while zeroes are padded at odd rows - effectively doubling the data size to convert into IQ format.

The doubling of data size doesn't occur in case of a digital waveform.

1.2.3: Data transfer speed in M5300x SW

Normal APIs

Table 8 Average time and data transfer rate measured per waveform using the create() API in Python

Number of Points (per Waveform)	Average Time (ms)	Average Data Transfer Rate (MB/s)
64	0.18	0.6
500	0.2	4.9
1000	0.2	9.69
5000	0.31	31.57
10000	0.41	48.23
100000	2.59	73.5
500000	15.9	59.8
1000000	35.8	53.2

Remote APIs

Table 9 Average time and data transfer rate measured per waveform using the create() API in Python

Number of Points (per Waveform)	Average Time (ms)	Average Data Transfer Rate (MB/s)
64	0.39	0.32
500	0.42	2.35
1000	0.47	4.4
5000	0.8	12.39
10000	1.24	16.1
100000	10.18	18.7
500000	61.2	15.5
1000000	122.3	15.58

1.2.4: Measured values in M5300x SW

Table 10 Minimum and maximum values of parameters used in M5300x SW

Parameter	Minimum value	Typical value	Maximum value	Description
Waveform Size per channel	64 Samples (I+Q)	-	1G Samples (I+Q) = 1,073,741,824 Samples (I+Q)	Min/Max number of samples in a waveform.
Waveform length in time	64 Samples / 4.8 GSa/s = 13.33 ns	-	1,073,741,824 Samples / 4.8 GSa/s = 223.69 ms	Min and Max waveform length in time.
Waveform Granularity	16 Samples (I+Q)	-	-	Number of samples in a waveform should be integral multiple of 16 (I+Q).
start_delay	0	0	((2^24) - 1)*3.33 ns = 55868.125 µs	Amount of time a waveform start can be additionally delayed.
Skew value	-24	0	+24	Skew adjustment range in number of Samples (I+Q).
Number of Waveforms per channel	1	-	4096	Min/Max number of different waveforms that can be programmed in memory per channel. Note that total space in memory is 1G Samples per channel.
Number of entries in request queue	1	-	4096	Number of Waveform Requests that can be programmed per channel.
Number of cycles per request queue entry	1	1	((2^16) - 1) = 65535	Number of times a waveform request should be played. 0 = infinite cycles.
Minimum sample size for gapless cyclic play with cycle = 1	12,000 samples	-	-	To play continuous waveform in cyclic mode, our estimate is minimum 12000 sample sized waveform with cycle = 1. If you have are playing more than 1 cycle, then the minimum sample size requirement for gapless play will be less, because cycles will allow more time to buffer the next waveform.

Section 1.3: About PathWave FPGA and BSP

1.3.1: Using PathWave FPGA

The M5300x software supports the KF9000B PathWave FPGA Programming Environment (commonly known as PathWave FPGA) to program custom logic into the instrument FPGA.

PathWave FPGA provides a complete FPGA design flow from design creation to simulation to GateWare deployment to Hardware/Gateware verification. This environment provides an easy-to-use GUI, where you may create bitstream images targeted to a sandbox in the Keysight FPGA.

Refer to "Downloading required software" on page 42 for information about Keysight's landing page for PathWave FPGA software installer. For more information regarding the PathWave FPGA software, refer to the embedded help file or the KF9000B PathWave FPGA Programming Environment Document Library.

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

1.3.2: Using BSP with PathWave FPGA

PathWave FPGA, by itself, does not provide access to any of the controls, which are associated with the Keysight M5300A PXIe modules. You must install the Board Support Package (BSP) to leverage the features within the PathWave FPGA software for your design.

The Board Support Package (BSP) comprises of two parts—an FPGA Support Package (FSP) and a Runtime Support Package (RSP). These are installed separately from PathWave FPGA.

The FSP is that portion of the BSP that allows you to build a bit file for the target FPGA. It is consumed by PathWave FPGA to support design creation and sandbox compilation; everything that is performed without the physical hardware.

The RSP is that portion of the BSP that allows you to control your target FPGA. It provides an API that you can use to download and verify your FPGA bit image. You may use the RSP to load design images onto the hardware and perform simple register and streaming accesses to one or more sandboxes.

The FPGA design consists of two regions: the static region and the sandbox region. The static region for each supported module is defined within BSP and cannot be modified. This region defines the implementation of the FPGA interfaces to external resources, and defines the interfaces to the sandbox. A static region implementation can define one or more sandbox regions in an FPGA design. The sandbox region contains the user specific FPGA design. The interface of the sandbox depends on the static region implementation. A specific design flow is promoted by PathWave FPGA, called Partial Reconfiguration (PR). In a PR flow, a full FPGA reconfiguration is only necessary once for a given static region version. The sandboxes can be reconfigured anytime, without a full reconfiguration, and without stopping the current operation of the FPGA.

To perform FPGA designing, the BSP must be installed on the same machine as the PathWave FPGA software. Both PathWave FPGA software and the BSP function together and cannot be used individually.

To control the front panel IO on the M5300A modules, the M5300x API allows you to control the hardware.

For installation instructions regarding the PathWave FPGA Board Support Package, see “Installing required software” on page 43.

For more information regarding the PathWave FPGA interface available for the Board Support Package corresponding to the M5300A module, refer to the respective help file embedded in the PathWave FPGA software.

Section 1.4: About KS2201A PathWave Test Sync Executive software

The KS2201A PathWave Test Sync Executive software is a programming environment based on Keysight's Hard Virtual Instrument (HVI) technology, that enables you to develop and execute synchronous real-time operations across multiple instruments. The real-time sequencing and synchronization capabilities of the PathWave Test Sync Executive software make it a powerful tool for *Multi-Input Multi-Output (MIMO)* applications that require tight synchronization and real-time control and feedback in areas such as Quantum Computing.

1.4.1: About HVI Technology

HVI technology enables you to program one or multiple instruments to execute time-deterministic sequences of operations and execute them with precise synchronization. It achieves this by deploying an executable code into each instrument's HW to be executed by the HVI engine or processor integrated into the instrument. The code executes on these engines in parallel, across multiple instruments. The new user-defined HW operation of the group of instruments is called a Hard Virtual Instrument or just HVI. The sequences of operations or instructions executed by the HVI engines are called HVI sequences. On top of the advantages inherent of the new use model, several other features have been added, such as extended multi-chassis capabilities and expanded product support.

When creating an HVI, you can include any instrument, similar to Keysight's M5300A PXIe High-Speed AWG modules, that have HVI support.

1.4.2: M5300A Firmware version requirements for HVI

Refer to the *M5300A Release Notes* document to verify the M5300x software and firmware versions that are compatible for HVI programming with the most recent version of the KS2201A software. Firmware upgrade/downgrade and M5300x software upgrade/downgrade can be performed manually. There is no need to return the module to Keysight.

1.4.3: About HVI Application Programming Interface

The HVI Application Programming Interface (API) is the set of programming classes and methods that allows the user to create and program an HVI instance. Refer to the *KS2201A PathWave Test Sync Executive User Guide* to know more about the HVI Python API.

NOTE **PathWave Test Sync Executive programming supports Python versions 3.7, 3.8, 3.9, 3.10, 3.11 and 3.12 along with the .NET languages.**

HVI core functionality is extended by each instrument with an instrument specific API. The core API is common to all products and only the instrument specific HVI API will change (instrument instructions, actions, events), depending on the products. It is important to differentiate between the core HVI features and the instrument specific extensions, which allow a heterogeneous array of instruments and resources to coexist on a common framework.

The HVI Core API exposes all HVI functions and is a common API for all products. It defines the base interfaces and classes that are used to create an HVI, control the hardware execution flow, and operate with data, triggers, events and actions, but it alone does not include the ability to control instrument specific operations. The core API defines the hardware virtual instrumentation framework, and it is the job of the product-specific HVI instrument extensions to enable instrument functionalities in an HVI.

When Keysight M5300x is installed on a PXI system, it installs the drivers required to interact with the M5300A modules. Additionally, the graphical environment for the Keysight M5300x SFP provides a visual representation of most of the M5300x API's features.

The M5300x API classes contains HVI add-on API interfaces provided as an extension of the instrument. These add-on interfaces provide access to instrument specific HVI features such as toggling of the IO channel banks, and so on.

For installation and usage instructions regarding the KS2201A PathWave Test Sync Executive software, refer to the *KS2201A PathWave Test Sync Executive User Guide*.

2. Using the KS2201A PathWave Test Sync Executive software

This chapter provides an introduction to the KS2201A PathWave Test Sync Executive Software and describes its implementation in the M5300x API. For detailed information about the elements in KS2201A PathWave Test Sync Executive Software and its API, refer to the *KS2201A PathWave Test Sync Executive User Guide*.

Section 2.1: Licensing for KS2201A PathWave Test Sync Executive software

Each M5300A card is shipped with the newest version of firmware, which supports the KS2201A PathWave Test Sync Executive software. For more information regarding the supported firmware and software versions, refer to the *M5300A Software Startup Guide*.

Software license option for the KS2201A software

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

Section 2.2: Understanding operation flow in KS2201A HVI API

Table 11 summarizes the main operations necessary in an HVI design as performed in the KS2201A HVI API use model.

Table 11 Operation flow in KS2201A HVI API

Operations	KS2201A HVI API Use Model
HVI Design Flow	Application code must import the “keysight_pathwave_hvi” library to use the HVI API. HVI sequences can be created using programs directly into the application code without importing external files.
HVI Sequence	KtHviSequence class enables you to create a Local HVI sequence using programs that run “locally” on a specific HVI engine in a specific instrument. Local Sequences are accessed using ‘SyncMultiSequenceBlock’ statement placed in a SyncSequence (KtHviSyncSequence). The HVI top sequence is a SyncSequence that contains SyncStatements.
HVI SyncSequence	KtHviSyncSequence class enables you to add synchronized operations (Sync Statements) common to all HVI engines within the HVI instance. The HVI top sequence is a SyncSequence that contains SyncStatements. Local instructions are added and executed within Local Sequences that can be accessed by adding a ‘SyncMultiSequenceBlock’ in a SyncSequence.
HVI Resources (Chassis, Triggers, M9031A modules, and so on)	HVI resources can be configured using “KtHviPlatform” class and all the classes inside it.
Program HVI Sequences	You may program both HVI SyncSequences and HVI (Local) Sequences with the API methods add_XXX(), where ‘XXX’ is the statement name.
HVI Compile, Load, Run	API SW methods can compile the sequence using (hvi.compile()), load it to hardware using (hvi.load_to_hw()), run the sequence using (hvi.run()).

Section 2.3: Working with KS2201A PathWave Test Sync Executive software

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

2.3.1: Overview on HVI technology

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

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

The KS2201A PathWave Test Sync Executive software provides you with the capability of designing HVI sequences using an Application Programming Interface (API) available in Python coding language. The HVI Application Programming Interface (API) is the set of programming classes and methods that allows you to create and program an HVI instance. The HVI core functionality is extended by the PXIe M5300A modules using the M5300x API. The core HVI features and the API extensions that are specific to M5300A, allow a heterogeneous array of instruments and resources to coexist on a common framework.

All the PXIe M5300A modules support HVI technology. When Keysight M5300x software is installed on a PXI system, it installs the drivers required to interact with the M5300A modules. The classes in the Keysight M5300x API Library contain HVI add-on interfaces provided as an extension of the instrument.

The primary HVI elements defined for the M5300x API and the corresponding API functions are described in the following sections.

Section 2.4: Understanding the HVI elements used in M5300x API

The HVI Core API exposes all HVI functions and defines base interfaces and classes, which are used to create an HVI, control the hardware execution flow, and operate with data, triggers, events and actions, but it alone does not include the ability to control operations specific to the M5300x product family. It is the HVI instrument extensions specific to M5300A modules that enable instrument functionalities in an HVI. Such functions are exposed by the module specific add-on HVI definitions. The M5300x API describes the instrument specific resources and operations that can be executed or used within HVI sequences. See "Sample program to create an HVI sequence for an SMB trigger" on page 160 for an example, where HVI is implemented in M5300x API.

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

2.4.1: Description of various HVI elements

HVI Engine

For HVI to control an instrument, the instrument requires one or more HVI Engines. An HVI Engine is an *Intellectual Property (IP)* block that controls the functions of the instrument and the timing of operations. The HVI Engine is included directly in the instrument hardware or it can be programmed into a *Field Programmable Gate Array (FPGA)* in the instrument.

HVI works by deploying a binary executable to each hardware instrument to be executed by the HVI Engine. Different binaries execute on the different HVI Engines in parallel, across multiple instruments.

When you write an application that includes an HVI, you create HVI sequences. These are sequences of HVI Statements, and are operations that control the instrument. The HVI Sequences are compiled into the binary executables that the HVI Engine executes.

HVI Actions

HVI actions are digital electronic pulsed or level signals that are sent from the HVI engine to control instrument operations outside of the HVI Engine.

You use actions in HVI sequences to initiate operations. Typically, actions initiate instrument-specific operations. For example, in a digitizer instrument, a `StartAcquisition` action sends a digital pulse to start an acquisition operation.

HVI Events

HVI events are digital electronic pulsed or level signals that are sent to the HVI Engine and used as notifications when instrument operations have occurred outside of the HVI Engine.

You use HVI Events in HVI sequences as notification events that the execution has to wait for. Typically, events indicate instrument-specific operations have occurred. For example, in an AWG, it will send a digital pulse through the `WaveformDone` event when a waveform execution has been completed.

HVI Trigger

HVI Triggers are electronic signals that HVI engines can send or receive.

HVI Triggers are used to send signals and share data between instruments. You can use these to initiate operations, communicate states, or share information. There are multiple types of triggers depending on how they are connected, for example: front panel triggers (usually a SMA connector on the module’s front panel), PXIe triggers (connected to the PXIe backplane of the chassis), general purpose digital IO (LVDS connector in the module's front panel).

HVI Registers

HVI registers are similar to variables in a programming language. They hold values that can be modified at runtime and can be used as parameters for instructions and statements. Physically, HVI registers are small hardware memories located in HVI engines. Their contents can be shared between HVI Engines by using specific instructions.

The maximum number of available HVI registers in a sequence depends on the size of the registers used in the sequence.

Example:

Maximum number of 32 bit registers (HVIREG_COUNT) = 16

Maximum register size (HVIREG_MAXSIZE) = 48

Maximum number of 32 bit registers = 16

Maximum number of 48 bit register = 8

Maximum number of available HVI registers = 4x 48 bit registers + 8x 32 bit registers

HVI Instructions

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

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

FPGA Sandbox registers and Memory maps

Some instrument FPGAs provide a user-configurable region in the instrument FPGA known as an FPGA sandbox. This enables you to program the instrument with logic that implements your own custom functionality. HVI Registers and Memory Blocks are components in the FPGA sandbox you can use as resources in your HVI sequences. For more information see *Chapter 5: HVI integration with PathWaveFPGA* in the *KS2201A PathWave Test Sync Executive User Manual*.

For the instruments that support an FPGA sandbox, HVI can support the sharing of data between the sandbox and the HVI engine of a given instrument or between sandboxes of different instruments. This functionality depends of the availability of specific interfaces inside the FPGA Sandbox. To take advantage of these features, you must use *PathWave-FPGA* to create your design in the sandbox.

HVI is capable of addressing a number of 32 bit elements defined by the HVIPORT_AWIDTH parameter.

Example:

HVIPORT_AWIDTH = 24

Addressable 32 bit elements (memory arrays/registers): 2^24 = 16,777,216 (range: 0 - 16,777,215)

Resources available in PathWave Test Sync Executive

The following resources are available for use in the PathWave Test Sync Executive software:

Table 12 Available HVI Actions in M5300x SW

Action Number	Name	Description
0	AWG1Start	Start Waveform Playback for Channel 1
1	AWG2Start	Start Waveform Playback for Channel 2
2	AWG3Start	Start Waveform Playback for Channel 3
3	AWG4Start	Start Waveform Playback for Channel 4
4	AWG1Stop	Stop Waveform Playback for Channel 1, Will play out remaining data in data buffer with cycle = 1
5	AWG2Stop	Stop Waveform Playback for Channel 2, Will play out remaining data in data buffer with cycle = 1
6	AWG3Stop	Stop Waveform Playback for Channel 3, Will play out remaining data in data buffer with cycle = 1
7	AWG4Stop	Stop Waveform Playback for Channel 4, Will play out remaining data in data buffer with cycle = 1
8	AWG1Pause	Pause Waveform Playback for Channel 1
9	AWG2Pause	Pause Waveform Playback for Channel 2
10	AWG3Pause	Pause Waveform Playback for Channel 3
11	AWG4Pause	Pause Waveform Playback for Channel 4
12	AWG1Resume	Resume Waveform Playback for Channel 1
13	AWG2Resume	Resume Waveform Playback for Channel 2
14	AWG3Resume	Resume Waveform Playback for Channel 3
15	AWG4Resume	Resume Waveform Playback for Channel 4
16	AWG1Trigger	Trigger Waveform Playback for Channel 1
17	AWG2Trigger	Trigger Waveform Playback for Channel 2
18	AWG3Trigger	Trigger Waveform Playback for Channel 3
19	AWG4Trigger	Trigger Waveform Playback for Channel 4
20	AWG1JumpNextWaveform	Jump to Next Waveform for Channel 1
21	AWG2JumpNextWaveform	Jump to Next Waveform for Channel 2
22	AWG3JumpNextWaveform	Jump to Next Waveform for Channel 3
23	AWG4JumpNextWaveform	Jump to Next Waveform for Channel 4
24	AWG1QueueFlush	Queue Flush for Channel 1, Will abruptly stop waveform playback if no memory read is happening.
25	AWG2QueueFlush	Queue Flush for Channel 2, Will abruptly stop waveform playback if no memory read is happening.
26	AWG3QueueFlush	Queue Flush for Channel 3, Will abruptly stop waveform playback if no memory read is happening.
27	AWG4QueueFlush	Queue Flush for Channel 4, Will abruptly stop waveform playback if no memory read is happening.
28	UserFpga0	User Actions available in Sandbox
29	UserFpga1	User Actions available in Sandbox
30	UserFpga2	User Actions available in Sandbox

Action Number	Name	Description
31	UserFpga3	User Actions available in Sandbox
32	UserFpga4	User Actions available in Sandbox
33	UserFpga5	User Actions available in Sandbox
34	UserFpga6	User Actions available in Sandbox
35	UserFpga7	User Actions available in Sandbox
36	ApplyInstructions	Instructions with the delayedApply flag will wait for this action to be executed in sync
42	SwTrigger1	Used for loopback to Event [50]
43	SwTrigger2	Used for loopback to Event [51]
44	SwTrigger3	Used for loopback to Event [52]
45	SwTrigger4	Used for loopback to Event [53]

Table 13 Available HVI Triggers in M5300x SW

Action Number	Name	Description
0	PxiTrigger0	PXI Trigger 0
1	PxiTrigger1	PXI Trigger 1
2	PxiTrigger2	PXI Trigger 2
3	PxiTrigger3	PXI Trigger 3
4	PxiTrigger4	PXI Trigger 4
5	PxiTrigger5	PxiTrigger5
6	PxiTrigger6	PXI Trigger 6
7	PxiTrigger7	PXI Trigger 7
8	SmbTrigger1	SMB Front Panel Trigger 1
9	SmbTrigger2	SMB Front Panel Trigger 2
10	SmbTrigger3	SMB Front Panel Trigger 3
11	SmbTrigger4	SMB Front Panel Trigger 4
12	SmbTrigger5	SMB Front Panel Trigger 5
13	SmbTrigger6	SMB Front Panel Trigger 6
14	SmbTrigger7	SMB Front Panel Trigger 7
15	SmbTrigger8	SMB Front Panel Trigger 8
16	PxiStarTrigger	PXI STAR Trigger

Table 14 Available instrument-specific instructions in M5300x SW

Custom Instruction Name	Description	Parameters

`setAmplitude`	Configure Amplitude	`Channel number`
		`Amplitude Value`

`setPhaseMode`	Configure Phase Mode to either Absolute or Relative	`Channel number`
		`Phase Mode (ABSOLUTE or RELATIVE)`

`setPhase`	Configure Phase Value	`Channel number`
		`Phase Value`

`AwgQueueWaveform`	Queue Waveforms	`Channel number`
		`Waveform ID`
		`Cycles`
		`Trigger Mode`
		`Start Delay`

Section 2.5: Module initialization and execution procedures with HVI

2.5.1: Start up and Calibration operations

This operation must be performed when one or more of the following tasks are performed:

- the system is powered on for the first time or after any changes
- any hardware is changed
- (almost) any of the clock configuration is changed

Depending on whether the hardware and/or clock configuration has changed, Initialization Steps 6 and 7 below will change in the following way:

- If the System Calibration and User Deskew are already available for the current HW config and operating temperature:
  - Call `SystemDefinition::Initialize()` that will reuse existing calibration and apply existing User Deskew (an error will be generated if there is no System Calibration for the current system config/temp condition).
- If no System Calibration is available for the current HW config and operating temperature:
  - Call `SystemDefinition::Initialize(AlignmentOptions::ResetCalibration)` option in order to generate new System Calibration factors.
    - The RF ASIC DAC clock alignment may result in residual 26 ps channel skew errors that need to be further corrected using a User Deskew procedure.
    - This requires you to always re-calculate User Deskew when System Cal is executed (new System Cal factors).

Initialization steps

1 Physically assemble system and make all system connections (chassis, M5000-series modules, System Sync modules, external ref clock, and so on).

This includes changes on existing HW setup.

2 Power on all chassis in system as necessary.

3 Power on external controller if applicable.

4 Run a Python or Labber script that configures the system and aligns all the clocks. The script comprises:

a Connect to / Open modules.

```
driver = keysight_ktm5300x.KtM5300x(resource, query, reset, options)
```

b Create SystemDefintion (specify chassis, Cellini, modules, System Sync connections, clock configuration, and so on).

Specify in the `SystemDefinition.Clocking` interface to enable the 2.4 GHz reference.

```
system_def = keysight_hvi.SystemDefinition("MyHviSystem")

system_def.sync_resources = [ keysight_hvi.TriggerResourceId.PXI_TRIGGER2,
keysight_hvi.TriggerResourceId.PXI_TRIGGER3 ]

ref_chassis=system_def.chassis.add(2)

system_def.interconnects.add_sync_module('PXI0::CHASSIS2::SLOT10::INDEX0::INSTR', '')

clock_output_2_4GHz = ref_chassis.clock_source.outputs["FP2.4GHzOut"]

clock_output_2_4GHz.set_enabled(True)

clock_source = system_def.chassis[2].high_performance_clock_source

clock_source.set_mode(keysight_hvi.ClockingReferenceMode.INTERNAL)

system_def.clocking.reference_source = clock_source
```

`system_def.clocking.enable_external_analog_clocks([2400000000])`

c Add each module HVI Engine in the PathWave Test Sync Executive `SystemDefinition`.

```
awg_engine_name = "controlled module main engine"
engine0 = system_def.engines.add(ktm5300xhvi.engines.main_engine, "controlled module
main engine")
```

d Execute `SystemDefinition::Initialize(AlignmentMode::Full | AlignmentModes:PreCalibration)`

Note that the `AlignmentModes:PreCalibration` option indicates that there is no need to apply existing System Calibration Factors now because we're just preparing to warm-up the system.

Some modules may have channel skew errors.

5 Wait for at least 30 minutes for the system temperature to stabilize.

Note that the wait duration depends on the previous state of the system or the Hardware / Clock configuration that changed, or both.

6 Run a Python or Labber script that performs System Deskew calibration with Reset to generate Calibration data for the M5000-series modules.

a Create a new SystemDefinition or use the same as in step #4.

i Note that the SystemDefinition can change from step #4 here, but it cannot:

- change the system clocking definition
- add modules that require an analog clock alignment that is sensitive to temperature (M5000-series modules)

ii The KS2201A PWTSE software will change the behavior of the clocking reference selection. If you do not specify a “reference_source”, the PWTSE will use the existing clocking configuration if possible (instead of resetting to default configuration), thereby, automatically setting the clock_reference when creating system definitions after the initialization has previously been performed.

c Recalculate or use existing calibration, if available.

i If the System Calibration is available for this configuration and temperature profile, execute `SystemDefinition::Initialize()`.

The System Calibration is only available if this initialization procedure has been previously performed on this specific HW+clock configuration and this specific temperature profile. In general, this can only be known by the user and cannot be fully discovered by any SW.

All channel skews will be re-aligned to what they were after the prior analog clock alignment using `SystemDefinition::Initialize(AlignmentMode::ResetCalibration)`.

ii If the System Calibration is not available, execute `SystemDefinition::Initialize(AlignmentMode::ResetCalibration)`.

Aligning the M5000-series module’s clocks may introduce random channel skew errors, and will require the re-running of User De-skew calibrations.

d No actions after this step can be allowed to cause any HW or clock reconfigurations or another RF ASIC DAC clock alignment, even temporarily.

7 Perform and store new, or apply existing User Deskew calibrations.

Depending on step 6.b:

- If step 6.b.i, apply existing Deskew values
- If step 6.b.ii, Perform and Store new Deskew values

Note that this process may take a long duration.

2.5.2: Test Execution operations

This operation can be formed repeatedly after the system has been initialized as described in the previous section.

Execution steps

1 Connect to / Open modules, if not already open.

If you are required to open modules, do so without changing the clock configuration or anything requiring an analog clock re-alignment.

2 Apply User Deskew, if not already applied.

3 Perform User Experiments.

a Create a new **`SystemDefinition`** or use an existing system definition.

i Note that the physical system definition can change from that of the initialization step or previous system definitions, but it cannot:

- change the system clocking definition
- add modules that require an analog clock alignment that is sensitive to temperature (M5000-series modules)

ii The KS2201A PWTSE software will change the behavior of the clocking reference selection. If you do not specify a “reference_source”, the PWTSE will use the existing clocking configuration if possible (instead of resetting to default configuration), thereby, automatically setting the clock_reference when creating system definitions after the initialization has previously been performed.

b Create a Sequencer from the **`SystemDefinition`** and define/create the HVI sequence to run.

```
hvi_sequencer = keysight_hvi.Sequencer("myName", system_def)
```

Note that **`systemDefinition::Initialize()`** is called automatically if needed when the sequencer is created, no need to call it explicitly.

c Compile Sequencer to obtain the Hvi instance.

d Load HVI using **`hvi.load_to_hw()`**.

Note that **`systemDefinition::Initialize()`** is called automatically, if needed, on the **`load_to_Hw()`**, no need to call it explicitly.

e Run HVI using **`hvi.run()`**.

You can execute same HVI as many times as desired between Load and Release HW.

f Release the hardware using **`hvi.releaseHw()`**.

For more information and complete definition of HVI programming using the KS2201A PWTSE software, refer to the *KS2201A PathWave Test Sync Executive User Manual*.

Section 2.6: Timing to be added to HVI commands

This section defines the duration required to run HVI commands, which are required to program the M5300A modules. However, the duration corresponding to a command should be added to the following (or the next) command in your logic, to avoid conflicts in executing HVI commands.

You may run some commands simultaneously, but the wait time must be accounted for, at the end.

NOTE This is not the true latency of when the output from each command will respond.

Table 15 **Duration and Execution time for HVI commands in M5300A modules**

HVI command (Actions / Events / Instructions)	Duration of command (in nanoseconds)	Latency / Time from execution to signal appearing on output (in nanoseconds)	Can be run simultaneously for each Channel?
ktm5300xhvi.instruction_set.awg_queue_waveform	MAX [(2000 ns) or (no. of samples*100/1024)]	Max [(2000 ns) or (no. of samples*100/1024)]	Yes, but $16.6\overline{6}$ ns must be added between commands for each Channel
ktm5300xhvi.actions.awg1_queue_flush	2000	2000	Yes, all four Channels in one instruction
ktm5300xhvi.instruction_set.set_output_type	200	257.5	Yes, but $3.3\overline{3}$ ns must be added between commands for each Channel
ktm5300xhvi.instruction_set.set_phase_mode	200	257.5	Yes, but $3.3\overline{3}$ ns must be added between commands for each Channel
ktm5300xhvi.instruction_set.set_phase	200	257.5	Yes, but $3.3\overline{3}$ ns must be added between commands for each Channel
ktm5300xhvi.instruction_set.set_amplitude	200	257.5	Yes, but $3.3\overline{3}$ ns must be added between commands for each Channel
ktm5300xhvi.actions.awg1_start	10	100	Yes, all four Channels in one instruction
ktm5300xhvi.actions.awg1_trigger	10	20	Yes, all four Channels in one instruction
ktm5300xhvi.actions.apply_instructions	10	54.5	Yes, with trigger actions
ktm5300xhvi.actions.awg1_stop	10	100	Yes, all four Channels in one instruction
ktm5300xhvi.instruction_set.trigger_write (Write SMB/PXI trigger)	$3.3\overline{3}$	10	-

Section 2.7: HVI timings in M5300A modules

This section displays the HVI actions, events and instructions along with the corresponding latency measured for M5300A modules. For the description of each term listed in the tables below, see the chapter HVI Time Management and Latency in the *KS2201A PathWave Test Sync Executive User Manual*.

2.7.1: HVI-Engine Clock

The HVI Engine clock period is a key parameter for the sequence execution timing inside a module. All HVI statement timings are documented in clock cycles but the actual execution timing in each instrument depends on the HVI Engine clock period value.

For M5300A, the HVI engine clock is:

Table 16 HVI Engine clock frequency / period

HVI Engine Clock Frequency (MHz)	Period (ns)
300	$3.3\overline{3}$

2.7.2: HVI Sync Resources Timing

To calculate latency, the timing of the some HVI statements include the parameter *Instrument_SyncResources_Latency*, which is specific for each instrument.

Table 17 Instrument SyncResources Latency value

Resource name	Timing value (ns)
Instrument_SyncResources_Latency	$3.3\overline{3}$

2.7.3: HVI Sync and Flow control statement timing

For information on the Sync and Flow control statements, refer to *Chapter 8* in the *KS2201A PathWave Test Sync Executive User Manual*. The timing tables make use of product specific parameters like *Instrument_SyncResources_Latency*, HVI Engine clock period, and so on.

2.7.4: HVI Triggers Timing

Trigger groups

M5300A Trigger IOs are organized in groups, the organization of the groups has an impact on the instruction timing when multiple groups are used:

Table 18 Trigger groups and description

TriggerIO Groups	Signals	Description
Group 1 (Trig0 ... Trig7)	PxiTrigger0, PxiTrigger1, PxiTrigger2, PxiTrigger3, PxiTrigger4, PxiTrigger5, PxiTrigger6, PxiTrigger7	PXI Triggers [7:0]
Group 1 (Trig8 ... Trig15)	SmbTrigger1 - SmbTrigger8	8 SMB Front Panel Triggers
Group 2 (Trig16)	PxiStarTrigger	PXI STAR Trigger

TriggerWrite instruction timing

TriggerWrite instruction timing values for M5300A:

TriggerWriteGroups=1 for M5300A modules

Table 19 TriggerWrite instructions and timing values

HVI TriggerWrite Instructions	Execution time (ns)	Fetch time (ns)
trigger_write to PXI Star	21.89	$3.3\overline{3}$
trigger_write to PXI trigger	21.89	$3.3\overline{3}$
trigger_write to SMB output	42.68	$3.3\overline{3}$

#### Trigger Write Groups

Any number of TriggerIOs can be written at the same time, but the timing is impacted by the number of TriggerWriteGroups. The number of TriggerWriteGroups is determined by the Trigger IO groups and the combination of ON or OFF values.

#TriggerWriteGroups = ceil[(TriggerIOGroupsON + TriggerIOGroupsOFF)/2]

where,

#TriggerIOGroupsON is the number of TriggerIOGroups that contain values set to ON.

#TriggerIOGroupsOFF is the number of TriggerIOGroups that contain values set to OFF.

Table 20 provides some examples for M5300A:

Table 20 Example of Trigger Write groups for M5300A modules

Example	Triggers ON	Triggers OFF	#TriggerIOGroupsON	#TriggerIOGroupsOFF	#TriggerWrite Groups	Execution time (ns)	Fetch time (ns)
1	SMB1, SMB2		1	0	1	42.68	$3.3\overline{3}$
2	SMB1	SMB2	1	1	1	42.68	$3.3\overline{3}$
3	PXI0	PXI2	1	1	1	21.89	$3.3\overline{3}$

2.7.5: HVI Actions Timing

The HVI instruction ActionExecute synchronously runs a list of HVI actions that you specify.

HVI actions are organized in groups of up to 16 actions. Any number of HVI actions can be executed synchronously, regardless of the group to which each action belongs. However, the number of action groups (#ActionGroups) included in an instruction impacts the instruction timing.

Action groups for M5300A modules

Table 21 Action groups and description

Action Groups	Signals	Description
Group 1 (Action0 ... Action15)	AWG1Start, AWG2Start, AWG3Start, AWG4Start, AWG1Stop, AWG2Stop, AWG3Stop, AWG4Stop, AWG1Pause, AWG2Pause, AWG3Pause, AWG4Pause, AWG1Resume, AWG2Resume, AWG3Resume, AWG4Resume	Group1 Actions
Group 2 (Action16 ... Action31)	AWG1Trigger, AWG2Trigger, AWG3Trigger, AWG4Trigger, AWG1JumpNextWaveform, AWG2JumpNextWaveform, AWG3JumpNextWaveform, AWG4JumpNextWaveform, AWG1QueueFlush, AWG2QueueFlush, AWG3QueueFlush, AWG4QueueFlush, UserFpga0, UserFpga1, UserFpga2, UserFpga3	Group2 Actions
Group 3 (Action32 ... Action47)	UserFpga4, UserFpga5, UserFpga6, UserFpga7, ApplyInstructions	Group3 Actions

ActionExecute instruction timing

For each type of action, the *Xn* value is the delay in ns from the instant the ActionExecute is executed with only that action until it is seen in the FP or Backplane, or the action takes effect internally if it has no impact on external signals.

Tclk is the period of the clock connected to the HviEngine.

Table 22 Formula to calculate ActionExecute instruction execution and fetch times

HVI ActionExecute Instruction	Execution time (ns)	Fetch time (ns)
Action0, Action1, ...	*X1* + *Tclk* x INT[(#ActionGroups-1) / 2]	*Tclk* + *Tclk* x INT[(#ActionGroups -1) / 2]
Action8, ...	*X2* + *Tclk* x INT[(#ActionGroups-1) / 2]	

where, INT is the integer part of a decimal number, for instance INT(1.0)=INT(1.5)=1.

Table 23 **ActionExecute instruction execution and fetch times in nanoseconds**

HVI ActionExecute Instruction	Execution time (ns)	Fetch time (ns)
AWG1Start, AWG2Start, AWG3Start, AWG4Start	198.19	$3.3\overline{3}$
AWG1Stop, AWG2Stop, AWG3Stop, AWG4Stop	198.21 Note1	$3.3\overline{3}$
AWG1Pause, AWG2Pause, AWG3Pause, AWG4Pause	194.83	$3.3\overline{3}$
AWG1Resume, AWG2Resume, AWG3Resume, AWG4Resume	194.86	$3.3\overline{3}$
AWG1Trigger, AWG2Trigger, AWG3Trigger, AWG4Trigger	194.87	$3.3\overline{3}$
AWG1JumpNextWaveform, AWG2JumpNextWaveform, AWG3JumpNextWaveform, AWG4JumpNextWaveform	198.21 Note2	$3.3\overline{3}$
AWG1QueueFlush, AWG2QueueFlush, AWG3QueueFlush, AWG4QueueFlush	188.19 Note3	$3.3\overline{3}$
UserFpga0, UserFpga1, UserFpga2, UserFpga3, UserFpga4, UserFpga5, UserFpga6, UserFpga7	31.327	$3.3\overline{3}$
ApplyInstructions	85.478	$3.3\overline{3}$

Note1: AwgStop latency shown is the minimum latency playing minimum waveform size of 64 samples. The actual latency of AwgStop will depend on the pre-fetched waveform in internal buffer, since AWG will empty out the waveform in the internal buffer with cycle = 1 before stopping.

Note2: AwgJumpNextWaveform latency shown is the minimum latency playing minimum waveform size of 64 samples, the latency of when AwgJumpNextWaveform will take effect depends on the size of the current waveform, and when the FPGA gets the AwgJumpNext-Waveform action.

Note3: If the FPGA is not retrieving memory, the latency shown will be the latency for AwgQueueFlush. However, if AwgQueueFlush is asserted during memory read, it will wait for the current memory read to finish and then Flush.

Table 24 provides some examples for M5300A:

Table 24 **Example of Action groups for M5300A modules**

Example	Actions executed	#ActionGroups	Execution time (ns)	Fetch time (ns)
1	AWG1Start, AWG2Start	1	198.19	$3.3\overline{3}$
2	AWG1Trigger	2	194.87	$3.3\overline{3}$
	UserFPGA4		31.327	
3	AWG1Start	3	201.52	$6.6\overline{6}$
	AWG2JumpNextWaveform		201.54	
	UserFpga7		34.657	

2.7.6: HVI Event Timing

The Instrument_Event_Latency is the delay from the event source until the event state is available inside the HVI Engine. Events can be used in event conditions and HVI statements in general, like the WaitForEvent statement. See the *KS2201A PathWave Test Sync Executive User Manual* for details on WaitForEvent and other statement timing using events. Table 25 lists the event latency for the M5300A events.

Table 25 Event latency in M5300A modules

HVI Event		Instrument_Event_Latency (Cycles)
External Events	PXI0, PXI1, PXI2, PXI3, PXI4, PXI5, PXI6, PXI7, PXIstar	0
	SMB1, SMB2, SMB3, SMB4, SMB5, SMB6, SMB7, SMB8	3

For the M5300A the TriggerIO additional condition evaluation latency is:

Table 26 TriggerIO Condition Latency Cycles

Name	Timing value (Cycles)
Instrument_TriggerIO_Condition_Latency	2

For the M5300A the Event condition evaluation latency is:

Table 27 Event Condition Latency Cycles

Name	Timing value (Cycles)
Instrument_Event_Condition_Latency	2

Local Wait-for-event Statement timing

A Local Wait-for-event statement blocks HVI execution in a Local sequence until an event occurs. The timing of this statement depends on the actual event used, as listed in the Table 25.

Events sources can be any of the Trigger IOs described earlier.

Table 28 displays the values for Local Wait-for-event statement timings.

Table 28 Execution and fetch time (in nanoseconds) for events in M5300A modules

HVI Event		Execution time (ns)	Fetch time (ns)
Trigger IO		MAX[Event_Arrival_Time (1) + (Instrument_Event_Latency (2) + Instrument_Event_Condition_Latency (3))*Tclk, Fetch_Time] + 1	(1 + Instrument_Event_Condition_Latency) *Tclk
	PXI0, PXI1, PXI2, PXI3, PXI4, PXI5, PXI6, PXI7, PXIStar	MAX([Event_Arrival_Time (1) + 6.66), 10] + 1	10
	SMB1, SMB2, SMB3, SMB4, SMB5, SMB6, SMB7, SMB8	MAX[(Event_Arrival_Time (1) + 16.65), 10] + 1	10

(1) *Event_arrival_time* comprises of:

- Internal Events
  - *Event_Arrival_Time = Internal_Event_Generation_Time - WaitForEvent_Start_Time*
- External Events
  - *Event_Arrival_Time = Event_At_Module_Connector_Time – WaitForEvent_Start_Time*
  - The event time can be measured at the front panel or PXIe backplane connector depending on the event.

(2) *Instrument_Event_Latency* is the delay from the event source until the event state is available inside the HVI Engine. Events sources can be the Trigger IOs, or internal to the product (including FPGA User Sandbox Events). It is an instrument and event specific value.

(3) *Instrument_Event_Condition_Latency* is the time needed for the condition evaluation to be executed once the event has settled inside the HVI Engine. It is an instrument specific value.

NOTE The Event_Arrival_Time can be a negative value if the event enters the module before the Wait-For-Event instruction Start Time. A number of scenarios are shown in the diagrams below.

Figure 13 and Figure 14 show a number of scenarios where the execution time of a Wait-For-Event statement can vary:

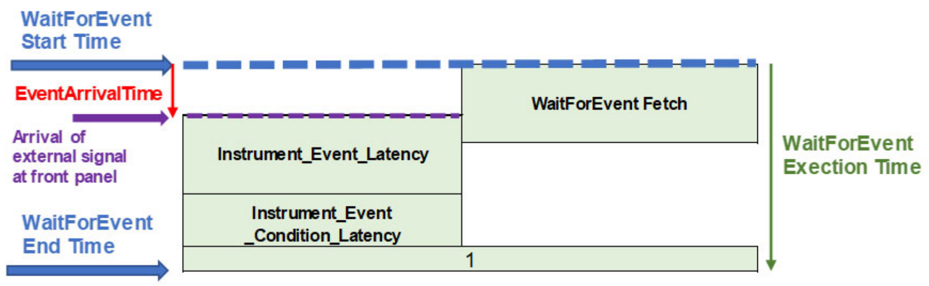


Figure 13 Block diagram representation of Event arrival after Fetch Time

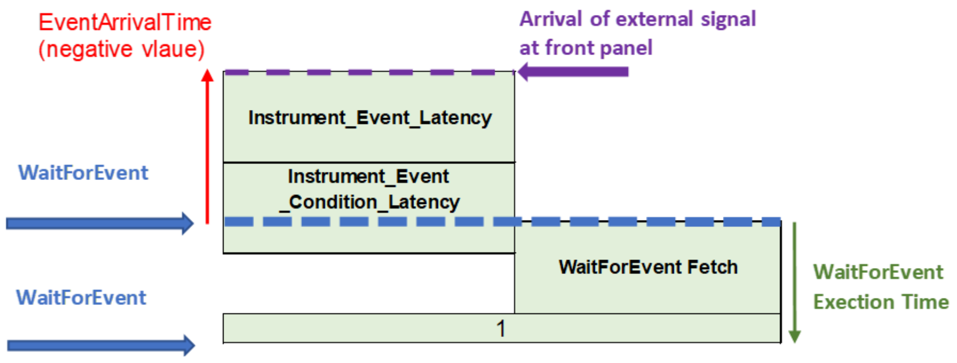

EventArrivalTime (negative vlaue)
Arrival of external signal at front panel
Instrument_Event_Latency
WaitForEvent
Instrument_Event _Condition_Latency
WaitForEvent Fetch
WaitForEvent
1
WaitForEvent Exection Time

Figure 14 Block diagram representation of Event arrival before Fetch Time

Table 29 **Local Wait-for-event latency values**

Parameter	Time (cycles)	Time (ns)
Start-Latency	0	0
End-Latency	1	$3.3\overline{3}$

Tclk is the period of the clock connected to the HviEngine. *EventArrivalTime* is the time from the start of the wait-for-event until the event is present at the instrument input, or for internal events when generated. If the Event happens before the wait-for-event start it can be negative down to *Instrument_Event_Latency* or (*Instrument_Event_Latency* + *Instrument_TriggerIO_Condition_Latency*) in the case of TriggerIOs.

2.7.7: HVI FPGA User Sandbox Instructions Timing

Instruction-specific parameter

The timing for the FPGA registers and Memory map access documented in the KS2201A PathWave Test Sync Executive User Manual includes a product specific parameter *Instrument_HVI_FPGA_Latency*.

Table 30 **Instrument HVI FPGA latency values**

Parameter	Time (ns)
Instrument_HVI_FPGA_Latency	$6.6\overline{6}$

Table 31 **Instrument HVI FPGA Instruction Execution and Fetch time values**

Instruction	Execution time (cycles)	Fetch time (cycles)
FpgaArrayRead	2 * Instrument_HVI_FPGA_Latency + 4	1
FpgaArrayRead (Address from HviRegister)	2 * Instrument_HVI_FPGA_Latency + 6	1
FpgaArrayWrite	Instrument_HVI_FPGA_Latency + 2	1
FpgaArrayWrite (Address or data from HviRegister)	Instrument_HVI_FPGA_Latency + 4	1
FpgaRegisterRead	2 * Instrument_HVI_FPGA_Latency + 4	1
FpgaRegisterWrite	Instrument_HVI_FPGA_Latency + 2	1
FpgaRegisterWrite (Address or data from HviRegister)	Instrument_HVI_FPGA_Latency + 4	1

Table 32 summarizes the latency for all HVI FPGA User Sandbox instructions for M5300A:

Table 32 **Instrument HVI FPGA Instruction Execution and Fetch time values**

Instruction	Execution time (ns)	Fetch time (ns)
FpgaArrayRead	8*3.33=26.64	$3.3\overline{3}$
FpgaArrayRead (Address from HviRegister)	10*3.33=33.33	$3.3\overline{3}$
FpgaArrayWrite	4*3.33=13.32	$3.3\overline{3}$

Instruction	Execution time (ns)	Fetch time (ns)
FpgaArrayWrite (Address or data from HviRegister)	6*3.33=19.98	$3.3\overline{3}$
FpgaRegisterRead	8*3.33=26.64	$3.3\overline{3}$
FpgaRegisterWrite	4*3.33= 13.32	$3.3\overline{3}$
FpgaRegisterWrite (Address or data from HviRegister)	6*3.33= 19.98	$3.3\overline{3}$

2.7.8: HVI Local Instructions Timing

The following tables show the instruction timing for HVI instructions defined by M5300A. Refer to the *KS2201A PathWave Test Sync Executive User Manual* to consult the HVI Native instruction timing and obtain further details about the HVI instruction timing and operation.

Table 33 ALU instructions basic timing values

ALU Instructions	Execution time (cycles)	Fetch time (cycles)
Add	8	1
Subtract	8	1
Assign	5	1

Arithmetic Logic Unit (ALU) instructions are the register add, subtract or assign operations that are available in the HVI-native instruction set.

Table 34 ALU instructions basic timing values in nanoseconds

ALU Instructions	Execution time (ns)	Fetch time (ns)
Add	26.64	$3.3\overline{3}$
Subtract	26.64	$3.3\overline{3}$
Assign	16.65	$3.3\overline{3}$

2.7.9: Other HVI timings

FPGA-Instruction Statement Timing

FPGA-Instruction statement latency depends on a number of factors:

- Instruction fetch time.
- Time to fetch data from any HVI registers it uses.
- Instrument specific delays.

Apart from fetch time and the first two execution cycles spent inside the HVI engine, the rest of the latency is defined by the instrument, this is condensed into the single parameter *Instrument_FpgaInstruction_Latency*.

Table 35 **Instrument_FpgaInstruction_Latency value**

Instruction	Execution Time (cycles)	Fetch Time (cycles)	Execution Time (ns)	Fetch Time (ns)
FPGA-Instruction	[Instrument_FpgaInstruction_Latency] + 2	1	[Instrument_FpgaInstruction_Latency] + 2*Tclk = [34.667 + (2*3.33)] + 2*3.33 = 47.987	$3.3\overline{3}$

M5300A-specific HVI Instructions

Instrument-specific local instruction statement latency depends on a number of factors:

- Instruction fetch time.
- Time to fetch data from any HVI registers it uses.
- Instrument specific delays.

Table 36 **Custom M5300A instructions timing values in nanoseconds**

M5300A specific Instructions	Apply	Execution time (ns)	Fetch time (ns)
`setAmplitude`	0	85.448	$3.3\overline{3}$
	1	292.168	$3.3\overline{3}$
`setPhaseMode`		292.188	$3.3\overline{3}$
`setPhase`	0	85.538	$3.3\overline{3}$
	1	292.188	$3.3\overline{3}$
`setOutputType`	0	84.578	$3.3\overline{3}$
	1	291.138	$3.3\overline{3}$

Sync FPGA Data sharing

For information about Sync FPGA Data Sharing statement, refer to the *KS2201A PathWave Test Sync Executive User Manual*.

Table 37 **Values for factors used in equations for Sync FPGA Data Sharing statement**

Parameter	Cycles
Instrument Tx latency	2
Instrument Rx latency	2
SSM latency (SSM specific) Note: SSM is a Sync module.	NA
Link latency (if this includes an instrument dependent factor)	NA

Table 38 **Latency values for Sync FPGA data-sharing statement**

Parameter	Description	Cycles
Start-Latency	Minimum start-delay for statement	0
End-Latency	Minimum start-delay for the next statement	1

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

This chapter provides an introduction to the PathWave FPGA Board Support Package (BSP) and describes its implementation in the M5300x API. For detailed information about using the *PathWave FPGA* software along with BSP for supported modules, refer to the *PathWave FPGA Customer Documentation* and the *BSP guides* for the respective modules.

NOTE The references to the version of the KF9000B PathWave FPGA environment in this chapter may be older than the current available version. Visit the PathWave FPGA Software download page to obtain the latest version.

Section 3.1: Licensing for PathWave FPGA BSP support

All instruments that can be programmed with PathWave FPGA BSP will enable programming.

These modules may also be purchased to include the following Xilinx FPGA:

1 xcvu35p-fsvh2104-2l-e

The following table outlines the Sandbox FPGA resources, which are available for custom logic for each module.

Table 39 Available FPGA Resources (Sandbox) per module

Resource Type	Resource Information
CLB LUTs	244936
CLB Registers	489872
DSP Blocks	1872
Block RAM Tiles	360
Ultra RAM	240

Section 3.2: Features in PathWave FPGA

The primary features available in the KF9000B PathWave FPGA environment are:

- Support for larger number of Keysight developed boards
- Uses Xilinx Vivado Design Suite locally on Host Machine to generate the output bitstream
- Generates “k7z” file, which is loaded on the FPGA of the new modules
- IPs can be defined using IP-XACT file. This gives you information about the IP name, version, interfaces, category, and so on.
- A repository of IPs that are using IP-XACT can be loaded concurrently.
- Block designs are now called sub-modules.
- Multiple boards are supported using the BSP technology.

Section 3.3: Working with PathWave FPGA software

The KF9000B PathWave FPGA Programming Environment (commonly known as PathWave FPGA), powered by *Xilinx Vivado Design Suite*, is used for FPGA designing on supported Keysight PXIe modules.

Some applications require the use of custom on-board real-time processing, which might not be covered by the comprehensive off-the-shelf functionalities of standard hardware products. With PathWave FPGA, development time is dramatically reduced and you can focus exclusively on expanding the functionality of the standard instrument, instead of developing a complete new one.

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

NOTE

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

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

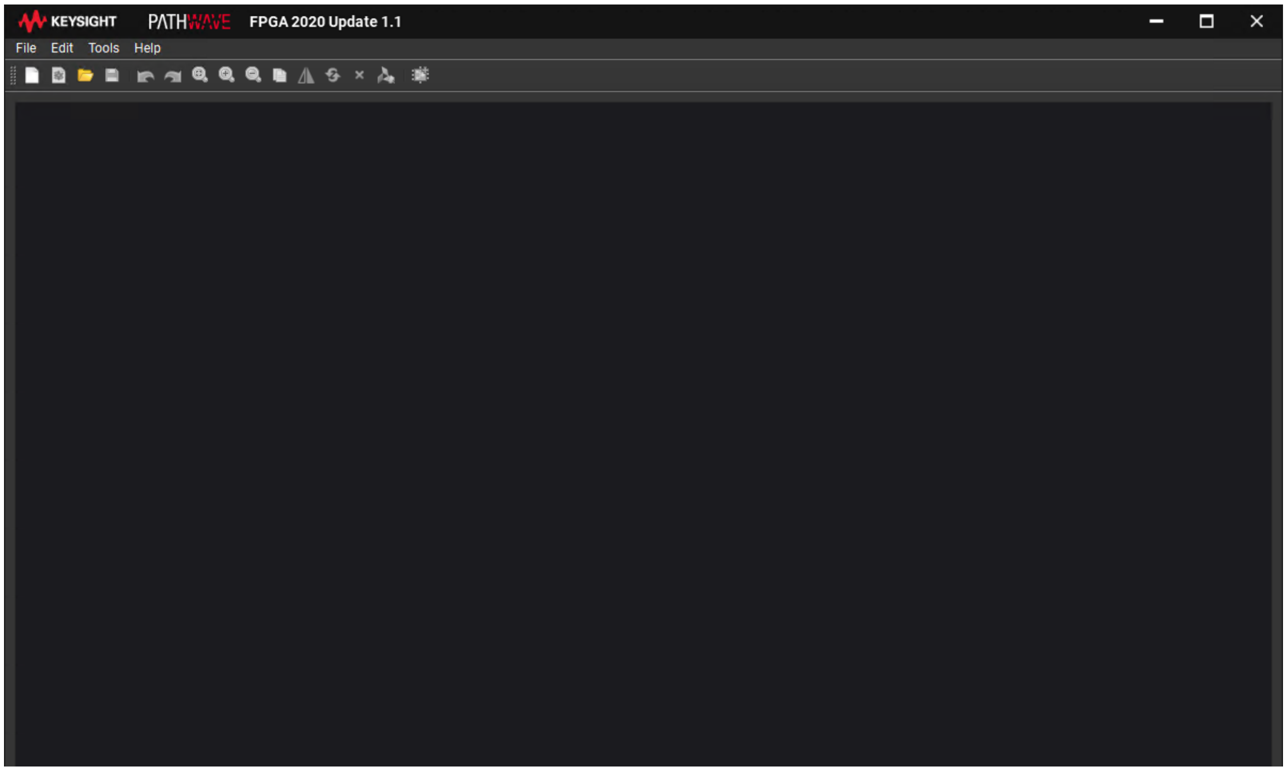


Figure 15 Default window for PathWave FPGA software

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

To know about the required prerequisite software, system requirements and licensing information for the PathWave FPGA software, refer to the "*Getting Started*" section of the *PathWave FPGA Customer Documentation*. To view the installation procedure, refer to the "*Installing PathWave FPGA software*" section and to view how to launch the software, refer to the "*Launching the PathWave FPGA BSP*" section in the *M5300A PXIe RF AWG Modules Startup Guide*.

3.3.1: Understanding Partial Configuration (PR)

The FPGA configurable region utilizes Xilinx FPGA partial reconfiguration technology to allow you to design and configure a defined section of the supported M5300A FPGA without the need to power down or reboot the module or host computer, respectively.

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

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

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

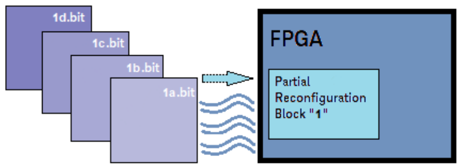


Figure 16 Block diagram depicting partial reconfiguration using bitstream files

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

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

The design part in Keysight FPGA consists of two regions: the static region and the sandbox region. The static region for each supported module is defined within BSP and cannot be modified. This region defines the implementation of the FPGA interfaces to external resources, and defines the interfaces to the sandbox. A static region implementation can define one or more sandbox regions in an FPGA design. The sandbox region contains the user specific FPGA design. The interface of the sandbox depends on the static region implementation.

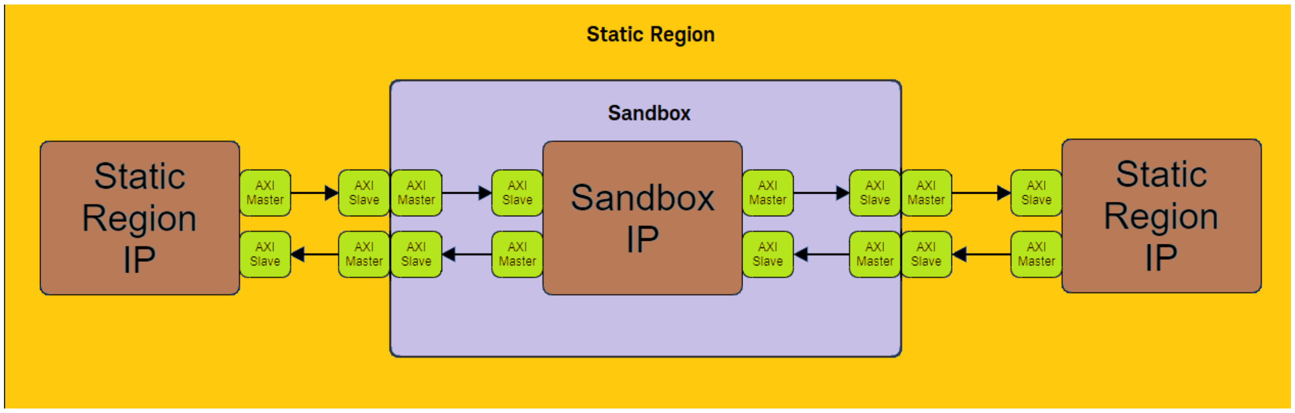


Figure 17 Block diagram representing layout of Static and Sandbox regions

The block diagram shown in Figure 18 depicts a simple view of M5300A IOs, sandbox ports, HVI connections, and data paths to and from the PCIe interface and DDR4 memory.

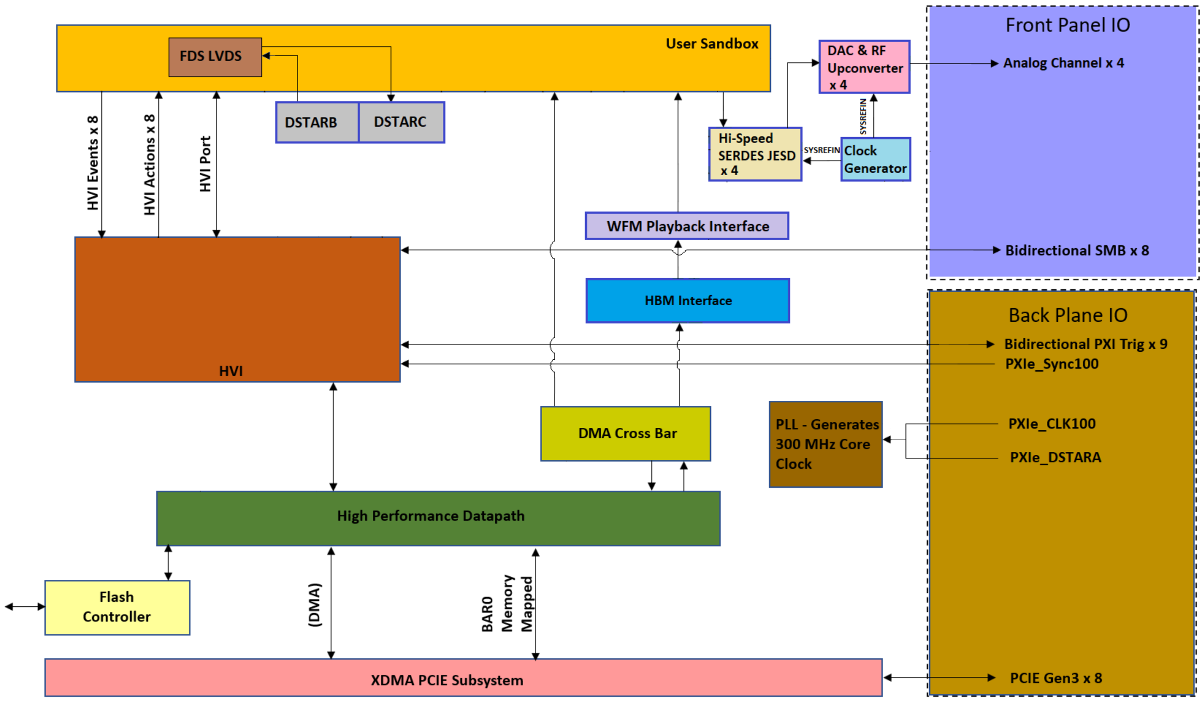


Figure 18 Block diagram depicting implementation of various ports in M5300A

Section 3.4: Using BSP with PathWave FPGA software

3.4.1: Understanding BSP composition

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

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

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

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

The block diagram shown in Figure 19 indicates how the bitstream file is generated using both the PathWave FPGA software and BSP together for any specific supported PXIe module.

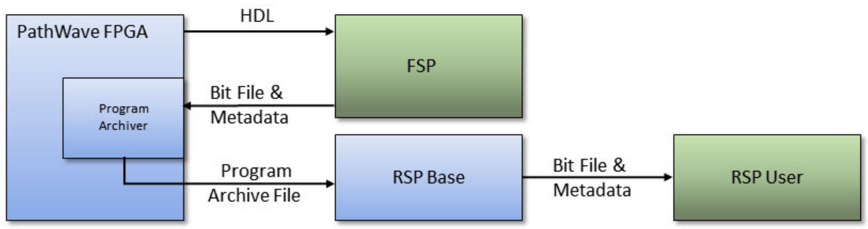


Figure 19 Bitstream file compilation flow using PathWave FPGA and BSP

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

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

The output from a PathWave FPGA design compilation is saved in a Keysight PathWave FPGA program archive file, with a *k7z* extension. To understand how to generate the k7z file onto a module, see "Generating a k7z file using PathWave FPGA BSP" on page 73.

Hereafter, you may load the k7z file onto the required hardware using the corresponding M5300x API functions. To understand how to load the k7z file onto a module, see "Loading k7z file onto modules" on page 80.

Section 3.5: Generating a k7z file using PathWave FPGA BSP

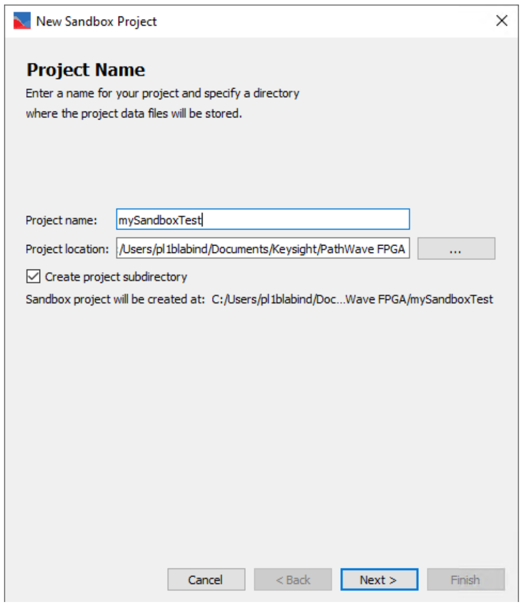
After you install BSP for one or more modules, you can proceed with creating a new project to design the sandbox region for the corresponding modules.

Following steps give you a quick glance into creating a new sandbox project using PathWave FPGA software for FPGA designing followed by generating a bitstream file for one or more supported PXIe modules.

You can also perform most of the steps shown below using command line arguments. Refer to the "*Advanced Features*" section of the *PathWave FPGA Customer Documentation*.

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

A new sandbox project dialog appears.



Project Name

Enter a name for your project and specify a directory where the project data files will be stored.

Project name: mySandboxTest

Project location: /Users/pl1blabind/Documents/Keysight/PathWave FPGA ...

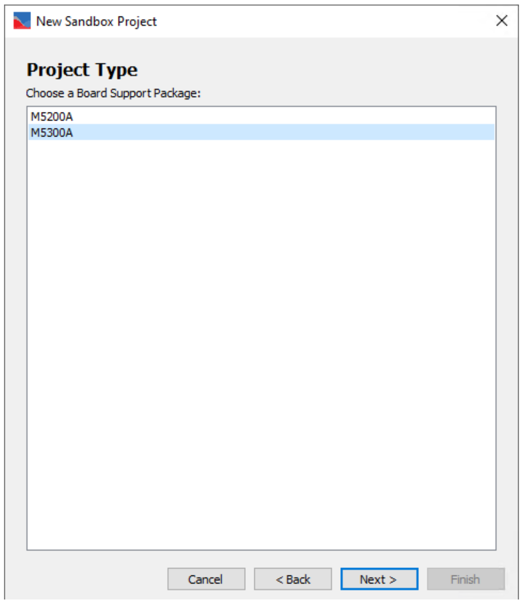
☑ Create project subdirectory

Sandbox project will be created at: C:/Users/pl1blabind/Doc...Wave FPGA/mySandboxTest

Figure 20 Creating a new Sandbox project

2 Click **Next >**.

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



Project Type

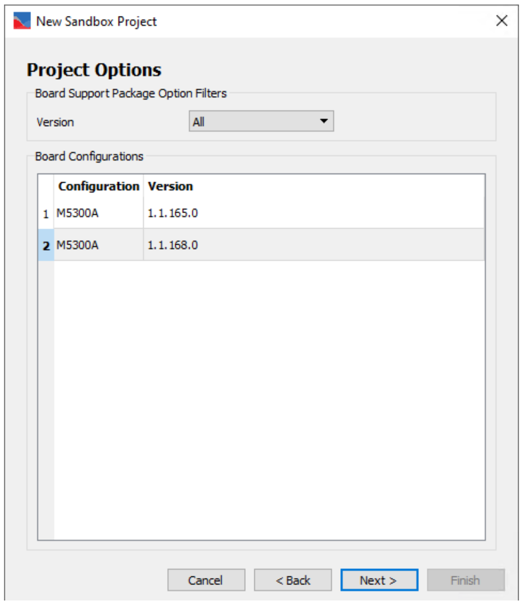
Choose a Board Support Package:

M5200A
M5300A

Figure 21 Selecting the module-specific BSP

4 Click **Next >**.

5 For the Project Options, choose the firmware version where the BSP changes must be updated.



Project Options

Board Support Package Option Filters

Version: All

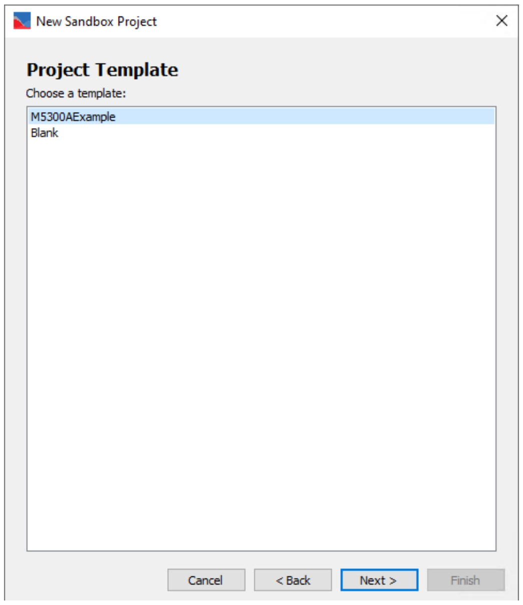
Board Configurations

	Configuration	Version
1	M5300A	1.1.165.0
2	M5300A	1.1.168.0

Figure 22 Selecting the firmware version to load BSP

6 Click **Next >**.

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



Project Template

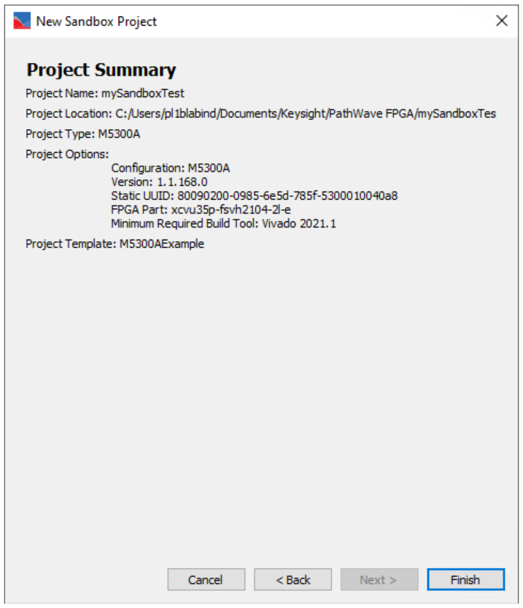
Choose a template:

M5300AExample
Blank

Figure 23 Selecting a project template for the selected BSP

8 Click **Next >**.

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



Project Summary

Project Name: mySandboxTest
Project Location: C:/Users/pl1blabind/Documents/Keysight/PathWave FPGA/mySandboxTes
Project Type: M5300A
Project Options:
Configuration: M5300A
Version: 1.1.168.0
Static UUID: 80090200-0985-6e5d-785f-5300010040a8
FPGA Part: xcvu35p-fsvh2104-2l-e
Minimum Required Build Tool: Vivado 2021.1
Project Template: M5300AExample

Figure 24 Viewing the project summary information

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

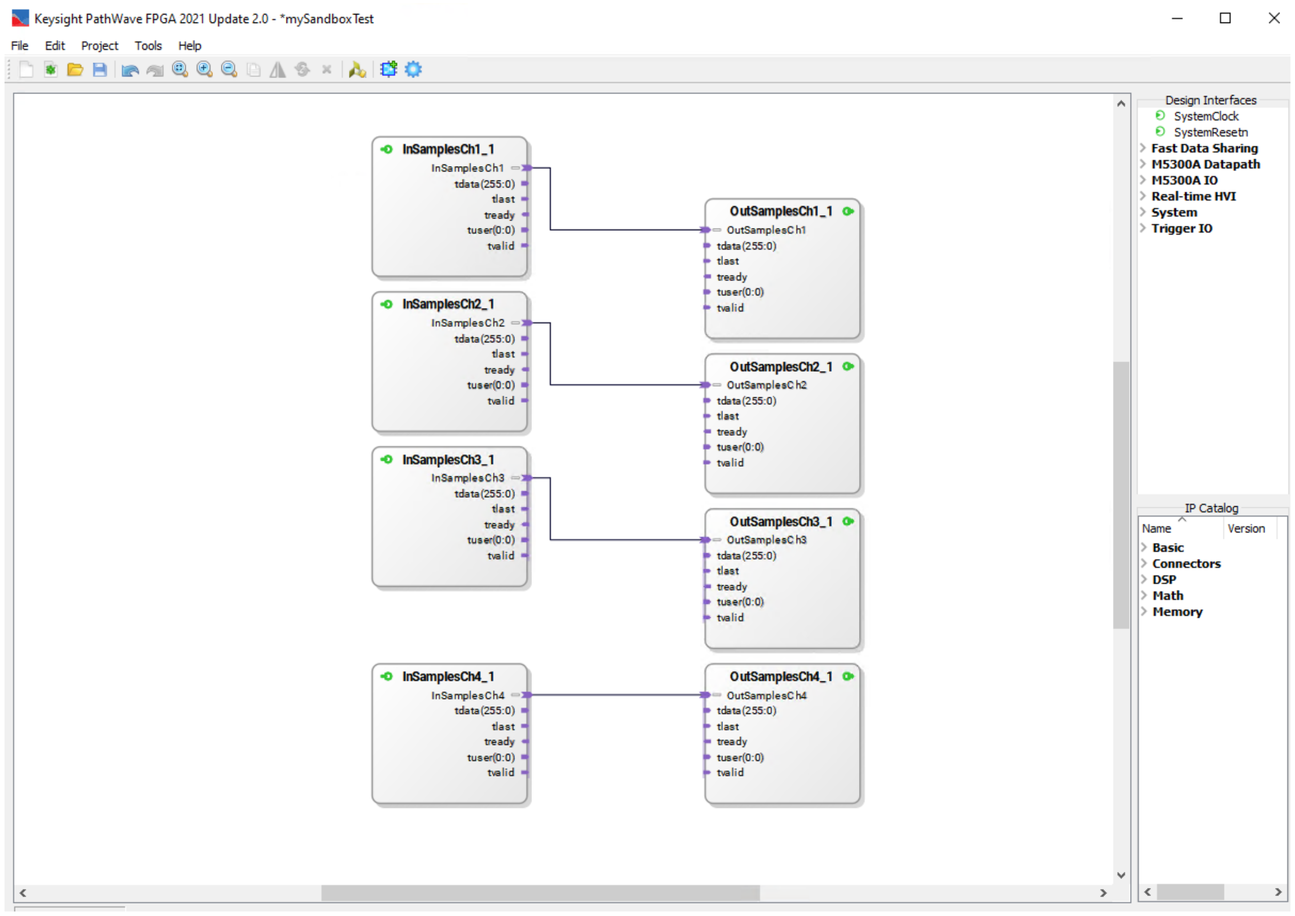


Figure 25 Viewing FPGA design blocks in the sample template

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

For more information regarding the configurable region of the M5300A FPGA interfaces and BSP IP Repository, refer to the BSP User Guide for the corresponding modules, which can be accessed via **Help** > **BSPs Help** menu options in the PathWave FPGA software.

11 After you have finished designing your logic, you can proceed with the k7z file generation. Click the **Generate Bit File...** icon as shown in Figure 26.

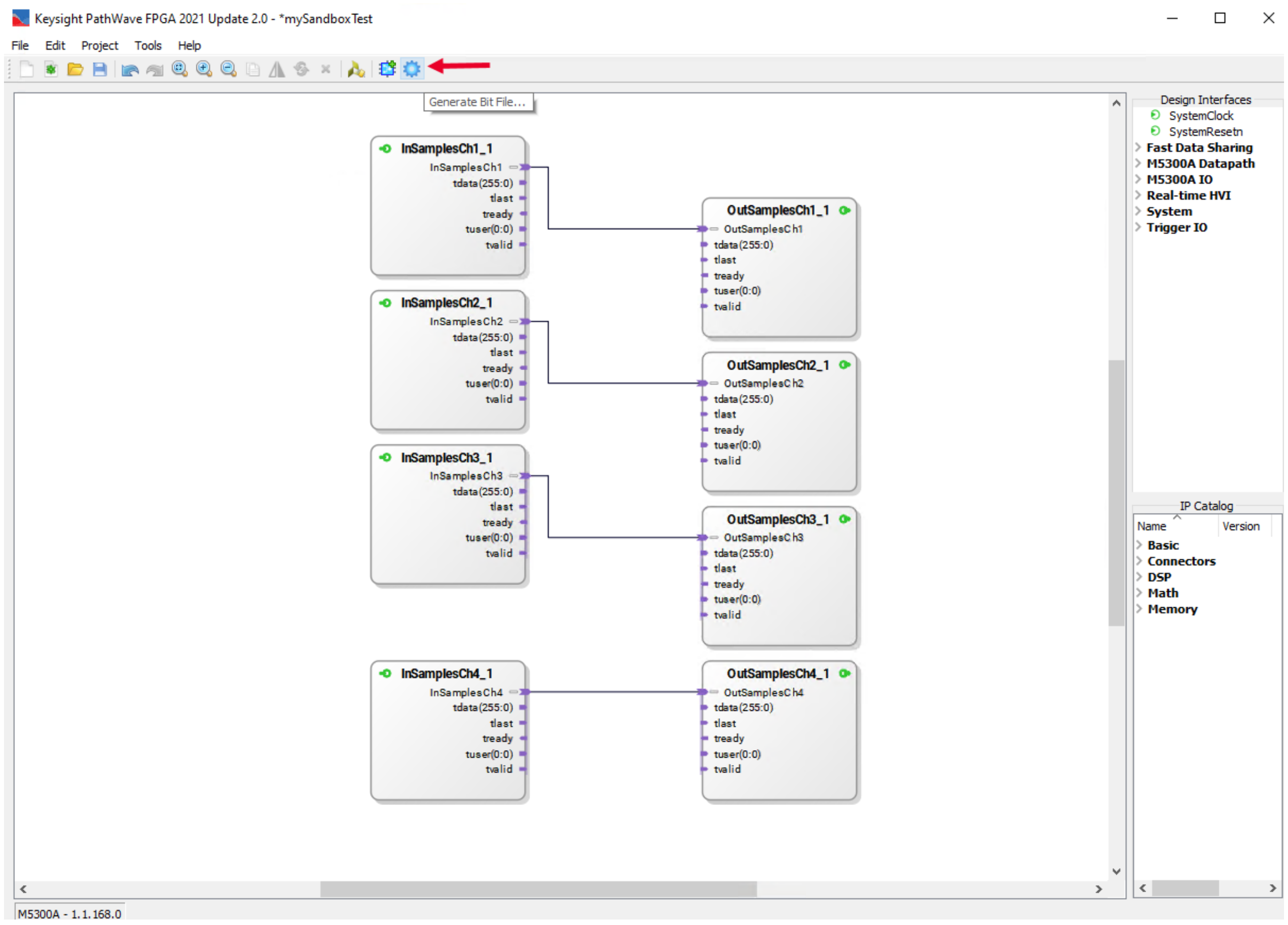
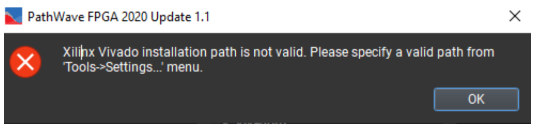


Figure 26 Initiating Bit File generation for the selected module

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

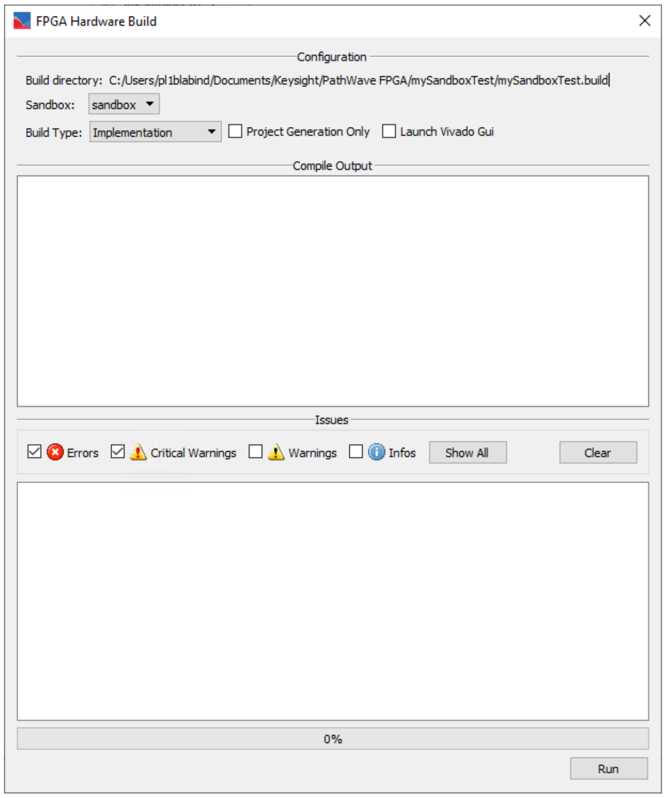


PathWave FPGA 2020 Update 1.1

Xilinx Vivado installation path is not valid. Please specify a valid path from 'Tools->Settings...' menu.

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

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



Build directory: C:/Users/pl1blabind/Documents/Keysight/PathWave FPGA/mySandboxTest/mySandboxTest.build

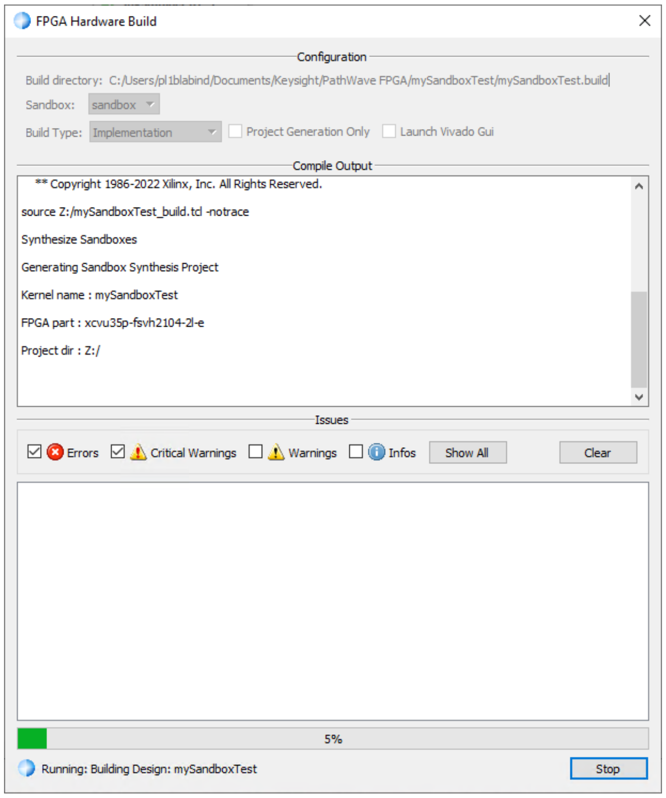
Sandbox: sandbox

Build Type: Implementation ☐ Project Generation Only ☐ Launch Vivado Gui

☑ Errors ☑ Critical Warnings ☐ Warnings ☐ Infos

Figure 27 Generating FPGA Hardware Build

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


FPGA Hardware Build
Configuration
Build directory: C:/Users/pl1blabind/Documents/Keysight/PathWave FPGA/mySandboxTest/mySandboxTest.build
Sandbox: sandbox
Build Type: Implementation
Project Generation Only
Launch Vivado Gui
Compile Output
Issues
Errors
Critical Warnings
Warnings
Infos
Show All
Clear
5%
Running: Building Design: mySandboxTest
Stop

```
** Copyright 1986-2022 Xilinx, Inc. All Rights Reserved.
source Z:/mySandboxTest_build.tcl -notrace
Synthesize Sandboxes
Generating Sandbox Synthesis Project
Kernel name : mySandboxTest
FPGA part : xcvu35p-fsvh2104-2l-e
Project dir : Z:/
```

Figure 28 Progress status for the k7z file generation

Once the Bitstream file (which is the *k7z* file) is generated, your custom FPGA logic is ready to be loaded on the selected module using the M5300x API.

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

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

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

3.5.1: Loading k7z file onto modules

After a bitstream file (k7z) is generated using the PathWave FPGA software, use the M5300x API functions to load the FPGA on the corresponding module and also, verify design compatibility on that module.

Prior to loading the bitstream file (k7z), make sure that the **Output Enable** feature in the M5300x SFP is set to **Disabled** so as to avoid unwanted data to be added at the output on the Channel. Refer to "Configuring Output Enable" on page 128 for more information.

To access the FPGA user design in the sandbox using M5300x API,

1 The generated *k7z* file may be loaded with the "configure_from_k7z" function

2 Retrieve the registers added to PathWave FPGA design using "get_register_entry" function.

  This API returns a tuple containing the address, length and access type.

3 (Optional) Write / Read to a 32-bit register using the Peek / Poke and writeRegisterBuffer / readRegisterBuffer functions, respectively.

See "Sample program to load a Sandbox design onto an M5300A module" on page 157 to understand the workflow of loading a sandbox design onto an M5300A module.

NOTE

If no data acquisition happens after the k7z file is loaded, it is likely that the issue may have stemmed from an incompatible or incorrect design-specific FPGA configuration. Try loading the default design located at *C:\Program Files\Keysight\M5300x\fpga\GtzFpga_R1.0_1.5.335.bin*. This should restore expected data acquisition functionality via the SFP.

M5300A PXIe RF AWG Modules

User Guide

4. Using Keysight Distributed Infrastructure (KDI)

KEYSIGHT TECHNOLOGIES

Section 4.1: Overview

Keysight Distributed Infrastructure (KDI) provides network infrastructure services and simplify deployment of a distributed environment.

KDI is Keysight's solution for tracking, monitoring, and managing various devices and services in a distributed environment. KDI provides a framework that enables Keysight software to discover and securely connect devices and services on the network for sharing/exchanging information and pushing updates and firmware. A subnet, where distributed nodes connect to a central server using KDI framework components, is referred to as KDI Fabric.

In a KDI Fabric, there are several "leaf" nodes that securely connect to a "root" node.

A root node (central server) is a device on which a KDI component called KDI Authentication Service (KDIS) is installed and is configured to run as a root node.

A leaf node could be any device, on which a KDI component called KDI Client (KDIC) is installed, be it an instrument, a test station, or any other system. You register services on these leaf nodes and configure them to connect to a root node in the KDI Fabric.

The central server is responsible for the following operations:

- registering new nodes to the KDI Fabric
- authenticating and authorizing secure connections within the nodes
- providing an interface for the KDI clients
  - to discover and securely connect to the members (devices/services) of the KDI Fabric.
  - enabling the creation of out-of-band/separate communication channels between a client and a member of the KDI Fabric.

Keysight software powered with KDI technology can seamlessly connect to devices and services registered in a KDI Fabric.

With KDI, you can:

- obtain a single access point to locate and securely communicate and control instruments and test stations.
- discover, start, stop, and send instant messages to services registered in a KDI fabric.
- obtain a single access point to update software modules across the KDI fabric.

You can also use KDI to avail TSE services for single and multi-host system tests. In single-host KDI remote control, you can access the remote HVI engines, but cannot control the chassis and system timing modules. That is where TSE service is used in Multi-host configuration. TSE Services makes use of KDI internally to open and initialize instruments at boot-up and get them ready to be used by the user application (such as M5200x SW, M5201x SW, M5300x SW, M5301x SW, M5302x SW and M9032x SW) later. For more information, refer to the *KS2201A PathWave Test Sync Executive User Manual*.

4.1.1: About KDI Authentication Service (KDIS)

KDIS is a standalone, cross-platform, and lightweight gateway service that authenticates connections. The authentication service enables secure communication within the KDI Fabric.

It includes the KDIS Admin UI that enables you to:

- Add, delete, and update users.
- Register devices (test stations and instruments).
- Register processes with credentials.

On receiving an authentication request, KDIS has the capability of performing the following actions:

- Validate the credentials against the database.
- Validate previously issued tokens and authorize a connection.
- Issue a public key for more efficient validation.

NOTE **Keysight recommends that only one KDIS instance is installed in a system. In this case, any host installing a KDI Client automatically detects the KDIS service for authentication. If for some reason more that one KDIS must be installed, then it is the responsibility of the user to direct each KDI client to the correct KDIS host.**

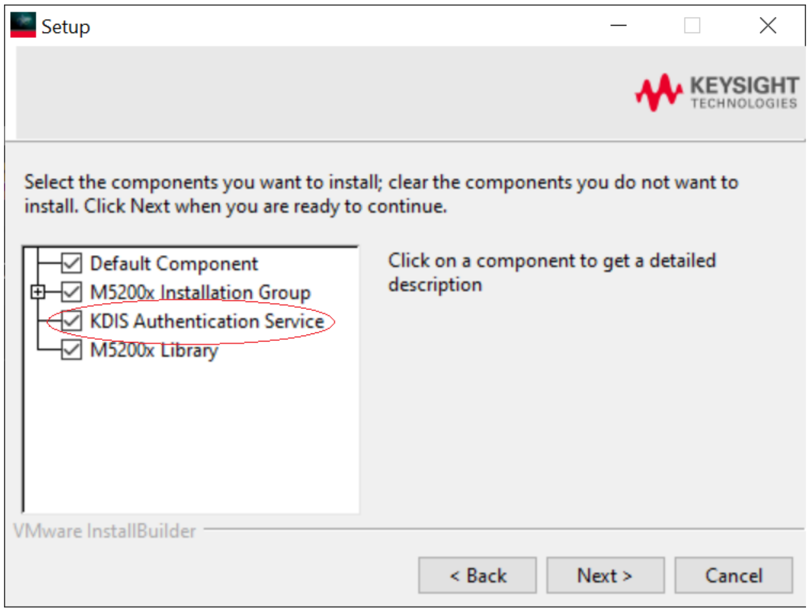
4.1.2: About KDI Clients (KDIC)

The KDI Client is a service that must run in any host that needs access to the KDI fabric and services: such as securely access information, status, and control features for nodes in the KDI Fabric.

Section 4.2: Installing KDI on the root node

During the installation of the SW drivers for one or more M5000-series modules, PathWave Test Sync Executive (PWTSE) 2023 or M9032x System Sync Module (SSM) drivers, you may select the 'KDI Authentication Service' component to be installed on the machine, which has been identified to be the root node.

For instance, if you install the M5200x SW drivers first on the root node with the 'KDI Authentication Service' component selected, as shown in Figure 29, the installers for the rest of the drivers shall ignore this component even if you select 'KDI Authentication Service'.



Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

- ☑ Default Component
- ☑ M5200x Installation Group
- ☑ KDIS Authentication Service
- ☑ M5200x Library

Click on a component to get a detailed description

Figure 29 Selecting KDI Authentication Service for installation on the root node

After installing one of more SW drivers on the root node, you must ensure the following on one or more leaf nodes:

- The same version of one or more M5000-series SW drivers / PWTSE 2023 / M9032x SW drivers must be installed on each leaf node. The order of installing the drivers does not matter.
- Do not select the 'KDI Authentication Service' component during installation of any of the drivers.

For instructions on how to install the SW drivers and the KDI component, refer to the *Startup Guide* for the respective M5000-series module / PWTSE 2023 / M9032x SW drivers.

Section 4.3: Configuring KDI and KDIS on root node

KDI infrastructure requires KDI Authentication Service (KDIS) to operate. The host with the KDIS installed is called the root-node. The KDIS instance must be the same for all hosts in a lab or setup that are intended to work together, basically the group of hosts that work together must all have the same root-node. It is recommended that KDI in all hosts are configured explicitly to point to the correct root-node or KDIS.

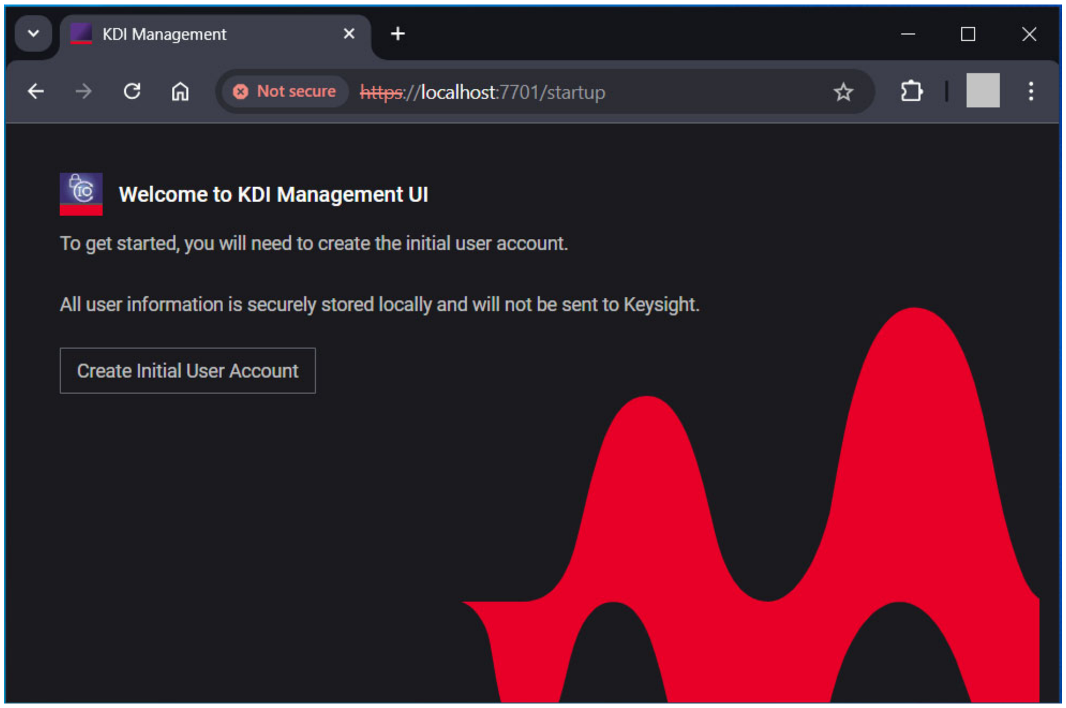
There are a number of steps to configure the KDI Infrastructure:

- Configure KDI Authentication Service (KDIS) - the root-node:
  - Configure admin credentials.
  - Create an initial user account.
  - Add one or more Users.
- Configure KDI Clients - this applies also to the root-node
- Restart the KDI Services.
- Accept KDI Clients in KDIS.

4.3.1: Configuring admin credentials in KDIS

1 In the root host, open the KDI Management UI at *https://localhost:7701/login*.
   - Ignore any security warnings to proceed to open this page.

   The **KDI Management UI** screen appears, as shown in Figure 30.


KDI Management
Not secure
~~https~~://localhost:7701/startup
Create Initial User Account

Welcome to KDI Management UI

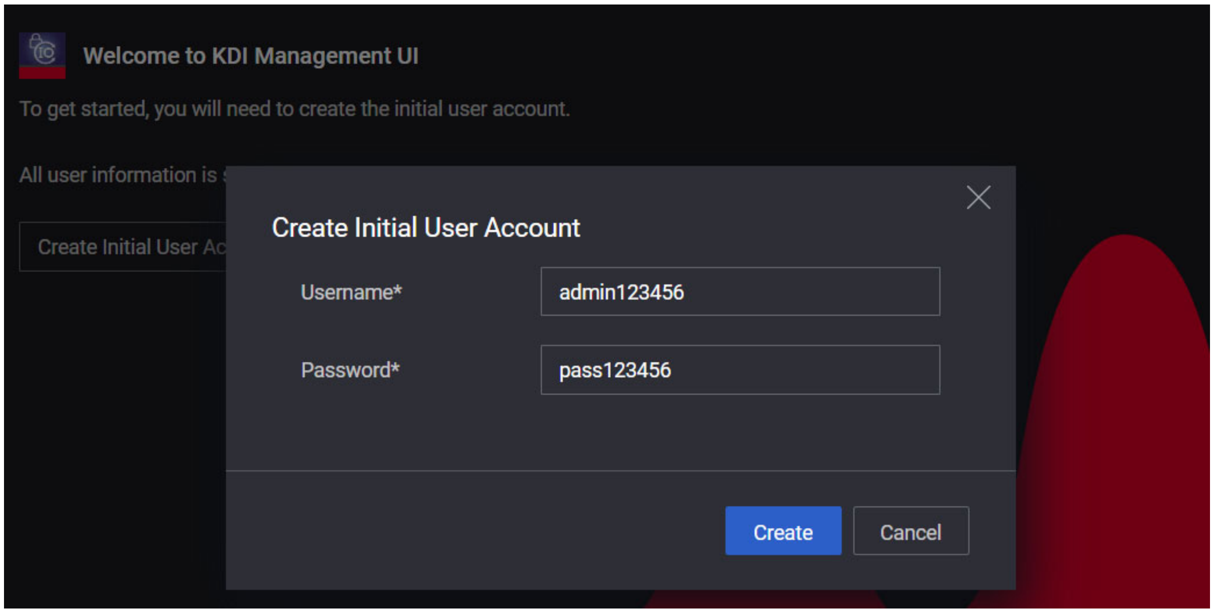
To get started, you will need to create the initial user account.

All user information is securely stored locally and will not be sent to Keysight.

Figure 30 Welcome to KDIS screen displayed on browser

You must first create an admin account, which enables you to access the KDIS User Interface and administer the system.

2 Click the **Create Initial User Account** button.
3 On the **Create Initial User Account** dialog box that appears, as shown in Figure 31:
   i Enter the (administrator) Username of your choice. For example, admin123456
   ii Enter the Password of your choice. For example, pass123456
4 Click **Create**.


Welcome to KDI Management UI
To get started, you will need to create the initial user account.
All user information is
Create Initial User Ac
Create Initial User Account
Create
Cancel

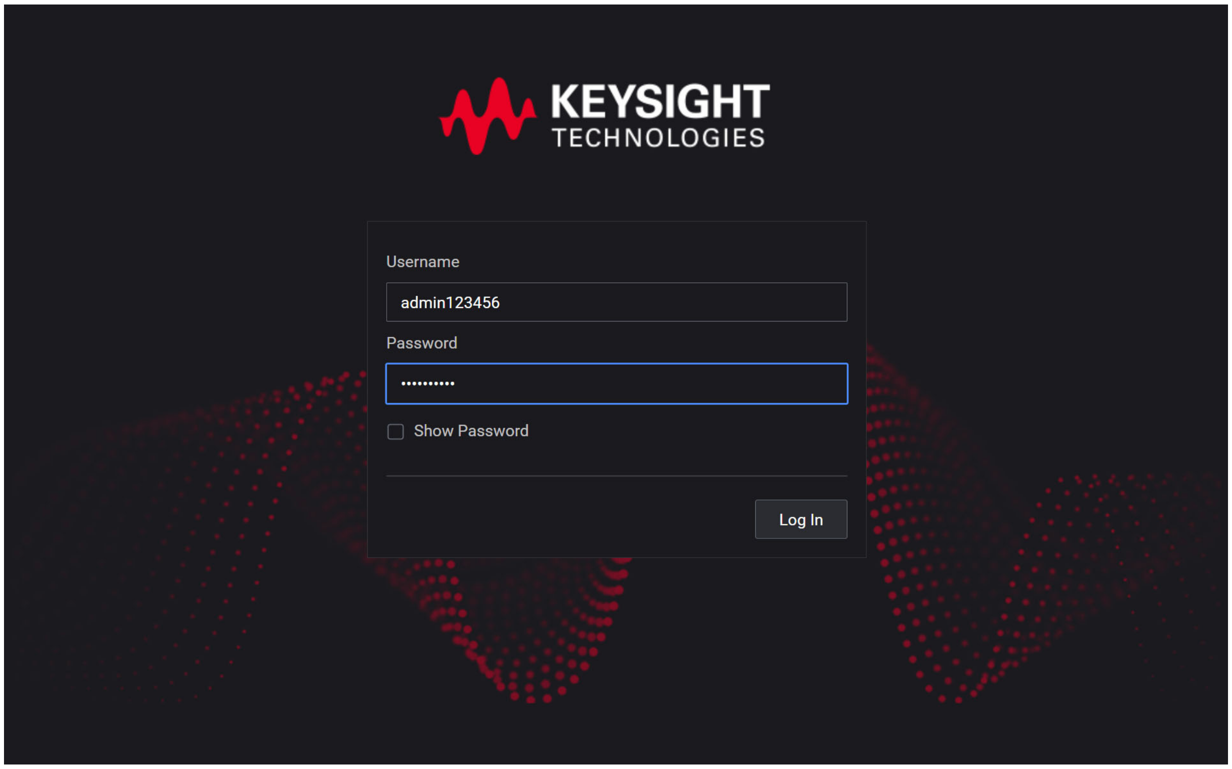
Username* admin123456
Password* pass123456

Figure 31 Creating credentials for the Admin user account

A login window appears.

5 Enter the admin credentials created in the previous step.

6 Click **Log In**.



admin123456

••••••••••

Figure 32 Logging in to the KDI Management UI with admin credentials

The KDI Management UI appears, as shown in Figure 33, which comprises of:

- the **Dashboard**
- the **Fabric Users**
- the **Fabric Devices**
- the **Remote Users**
- the **Manage Fabric**

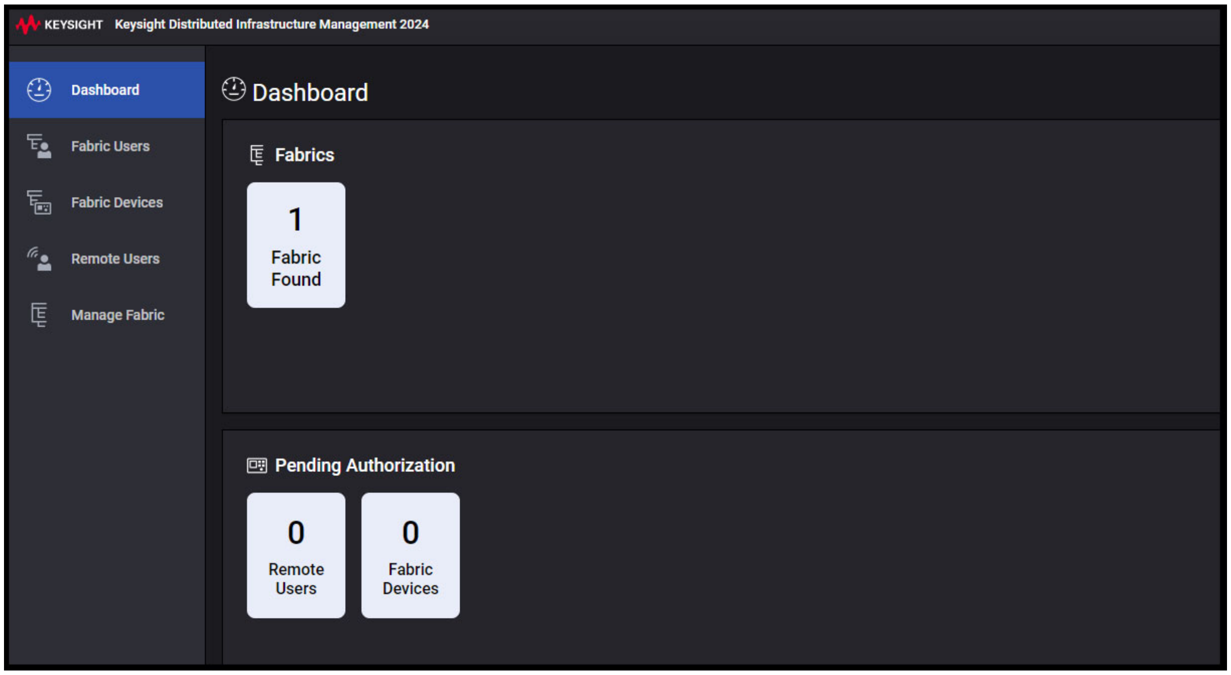
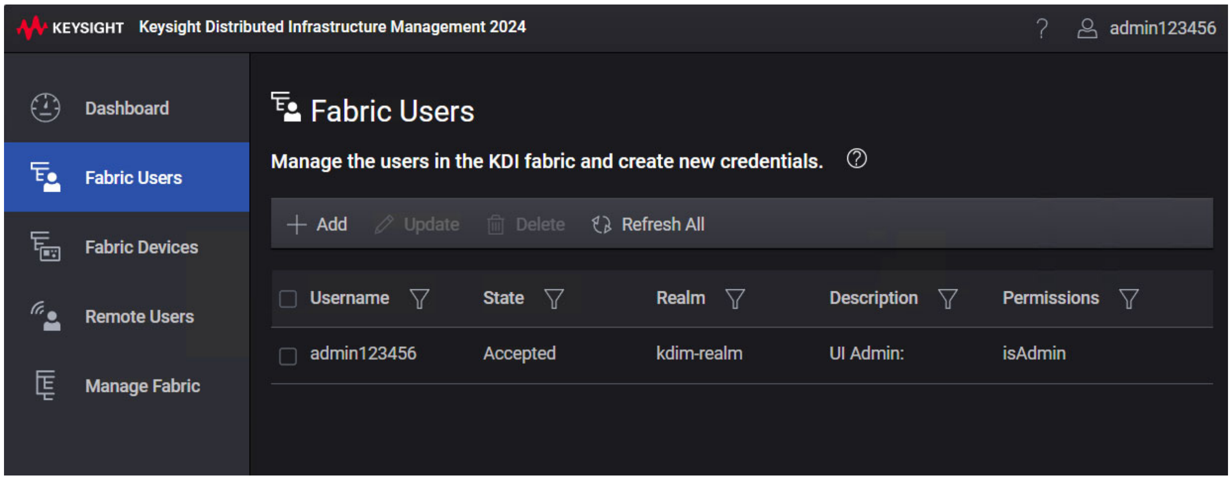


Figure 33 Viewing the KDI Management UI

The **Fabric Users** tab, as shown in Figure 34, comprises of:

- the **Username**, which displays
  - all admin and non-admin credentials created within the KDI Management UI
  - auto-generated IDs assigned to Root and Leaf KDIC nodes
- the **State**, which displays
  - *Accepted*, indicating that the entry, whether credentials or root/leaf KDICs, have been accepted by the Keysight Distributed Infrastructure System.
  - *Provisional*, indicating that the entry must be manually accepted or denied within the KDI Management UI.
- the **Realm**
- the **Description**, which displays the characteristic of the corresponding client Id.
- the **Permissions**, which displays the hostname of the leaf node that establishes connection to the root node.


KEYSIGHT Keysight Distributed Infrastructure Management 2024
admin123456
Dashboard
Fabric Users
Fabric Devices
Remote Users
Manage Fabric
Fabric Users
Manage the users in the KDI fabric and create new credentials.
Add
Update
Delete
Refresh All

Username	State	Realm	Description	Permissions
admin123456	Accepted	kdim-realm	UI Admin:	isAdmin

Figure 34 Viewing the default components in Fabric Users

4.3.2: Adding additional users in KDIS

To enable users to connect to an instrument with KDI requires a user (also called clients) registered in KDIS with a username (Client Id) and password (Secret).

NOTE The KdiUser and KdiPassword required when opening instruments in the user application refer to the Client Id and Secret registered in KDIS.

On the KDI Management UI, in the **Fabric Users** window:

1 Click **Add**.

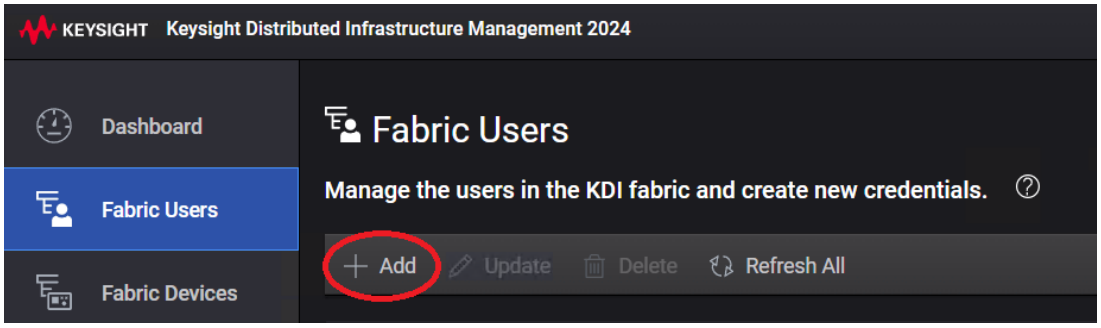
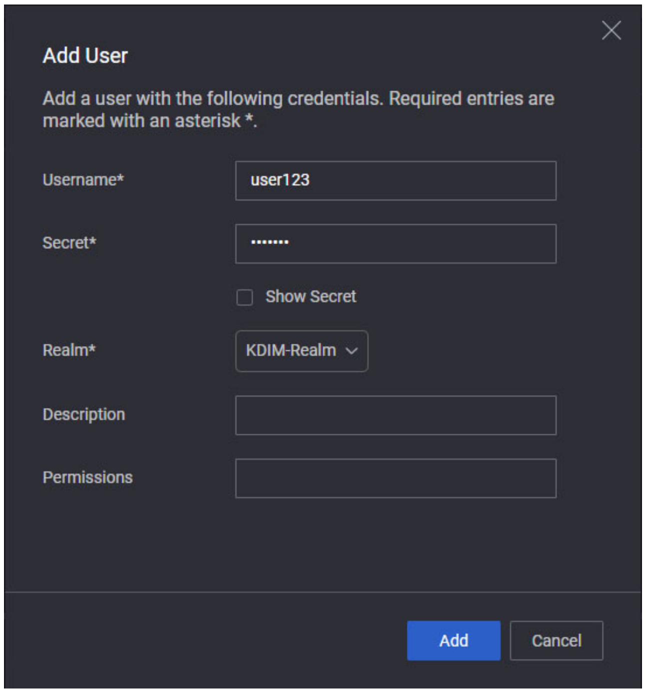

KEYSIGHT Keysight Distributed Infrastructure Management 2024
Dashboard
Fabric Users
Fabric Devices
Fabric Users
Manage the users in the KDI fabric and create new credentials.
Add
Update
Delete
Refresh All

Figure 35 Adding additional users in Fabric Users

The **Add User** dialog box appears, as shown in Figure 36.



Add User

Add a user with the following credentials. Required entries are marked with an asterisk *.

Username*	user123
Secret*	•••••••
	Show Secret
Realm*	KDIM-Realm
Description	
Permissions	

Figure 36 Adding client user credentials and realm

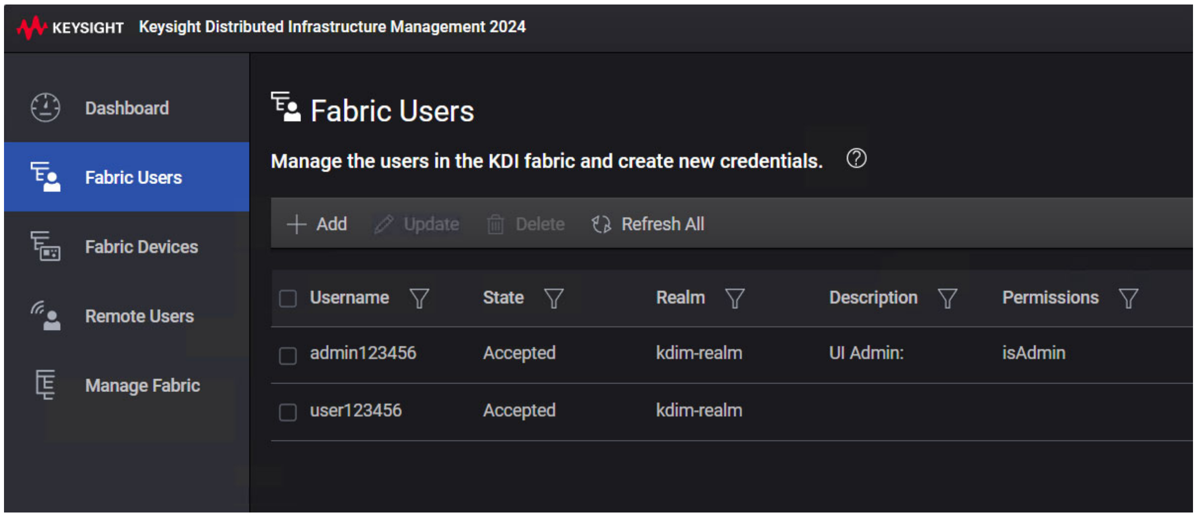
2 Enter new values for Username and Secret, as per your preference.

3 Among the additional options that appear, make sure that the **Realm** field is populated with the value *KDIM-Realm*.

4 Click **Add**.

5 Repeat the previous steps to create more users, as needed.

All new user entries appear on the Fabric Users window, as shown below. Note that the State displays ‘Accepted’, by default.



Fabric Users

Manage the users in the KDI fabric and create new credentials.

Username	State	Realm	Description	Permissions
admin123456	Accepted	kdim-realm	UI Admin:	isAdmin
user123456	Accepted	kdim-realm		

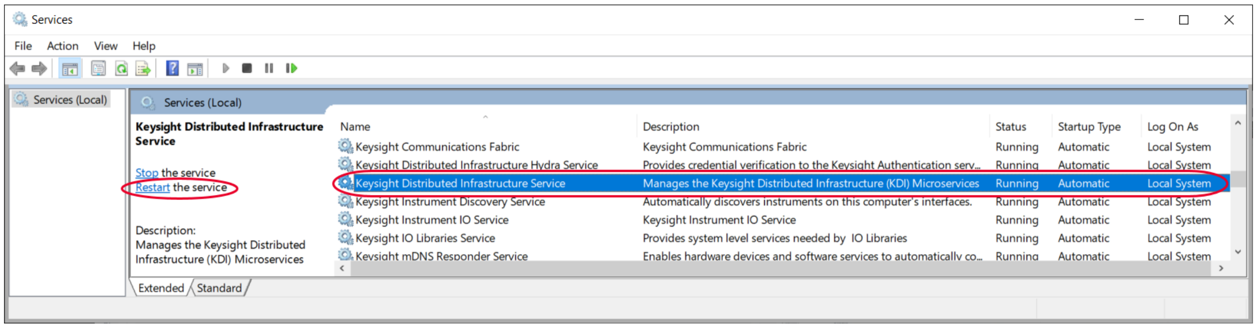
Figure 37 Viewing all newly added users in Fabric Users

6 After initial set up or each time you update the KDIS configuration in the root node, the KDI service in Windows must be restarted. Perform one of the following steps:

- Go to Task Manager > Services > highlight Keysight Distributed Infrastructure Service > click Restart.

  OR

- Use the following command: `C:\Program Files\Keysight\Distributed Infrastructure\DistributedInfrastructureService.exe -service restart`.



Keysight Distributed Infrastructure Service

Stop the service
Restart the service

Description:
Manages the Keysight Distributed Infrastructure (KDI) Microservices

Name	Description	Status	Startup Type	Log On As
Keysight Communications Fabric	Keysight Communications Fabric	Running	Automatic	Local System
Keysight Distributed Infrastructure Hydra Service	Provides credential verification to the Keysight Authentication serv...	Running	Automatic	Local System
Keysight Distributed Infrastructure Service	Manages the Keysight Distributed Infrastructure (KDI) Microservices	Running	Automatic	Local System
Keysight Instrument Discovery Service	Automatically discovers instruments on this computer's interfaces.	Running	Automatic	Local System
Keysight Instrument IO Service	Keysight Instrument IO Service	Running	Automatic	Local System
Keysight IO Libraries Service	Provides system level services needed by IO Libraries	Running	Automatic	Local System
Keysight mDNS Responder Service	Enables hardware devices and software services to automatically co...	Running	Automatic	Local System

Figure 38 Restarting the Keysight Distributed Infrastructure services in Windows

4.3.3: Configuring KDI Clients to find KDIS

Every host in the KDI infrastructure must have a root node (KDIS host) assigned. It is important for all hosts (leaf nodes) that must work together to have the same root node (KDIS) assigned. Note that the root node and all associated leaf nodes must have the same DNS configuration.

On each machine, identified as the leaf node,

7 Go to **Start** menu and run the **Set KDIM URL** application to define the hostname of the root node. You can find this application in the Start menu folder for any of the M5000-series module SW drivers, M9032x SW drivers or PWTSE 2023.

   For example, the Set KDIM URL is available in the M5200x SW folder.

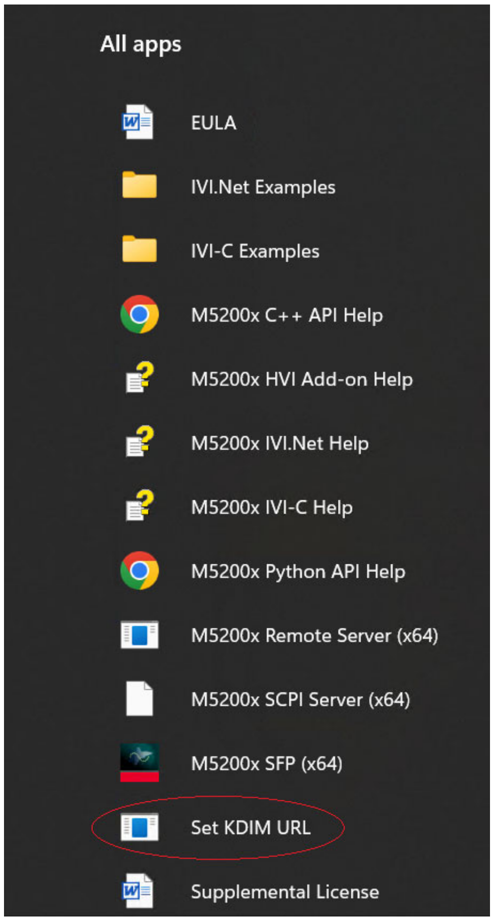
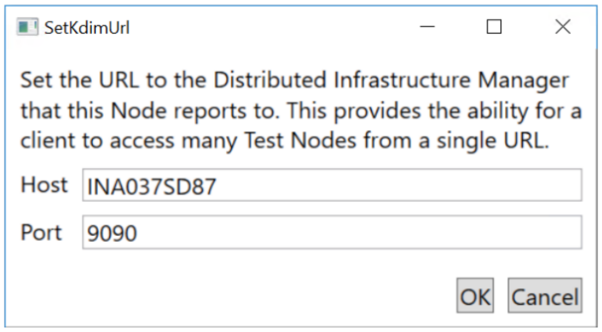


Figure 39 Accessing the Set KDIM URL application from one of the installed SW drivers

8 On the SetKdimUrl dialog box that appears,

- In the **Host** field, enter the hostname of the machine that was identified as the root node. If you wish to set the root node as leaf node, enter *localhost* (default entry). However, if the leaf node is different from the root node, enter the hostname / IP address of the root node.
- In the **Port** field, leave the default entry of *9090*.
- Click **OK**.


SetKdimUrl

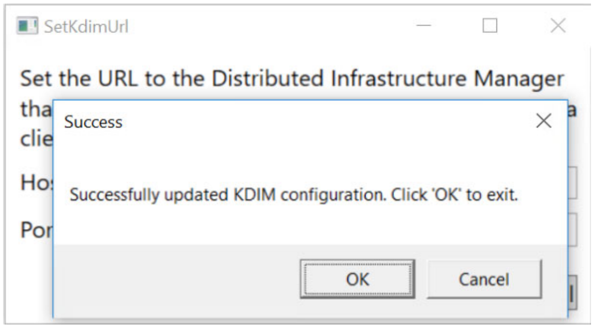
Set the URL to the Distributed Infrastructure Manager that this Node reports to. This provides the ability for a client to access many Test Nodes from a single URL.

Host INA037SD87

Port 9090

Figure 40 Configuring the SetKdimUrl on the leaf node to access the root node

9 Click **OK** on the Success prompt that is displayed.

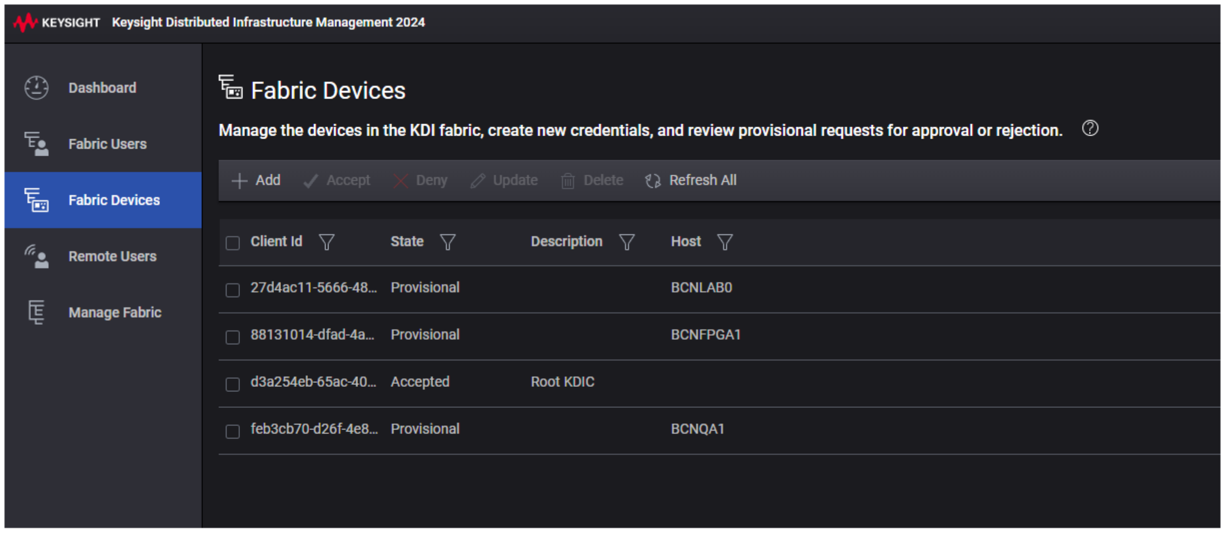

Success

Successfully updated KDIM configuration. Click 'OK' to exit.

Figure 41 Success prompt after successful KDIM configuration between leaf and root nodes

4.3.4: Accepting leaf nodes (KDI clients) in KDIS

In KDIS, all hosts (leaf nodes) are referred as clients. Once the KDI instances in the non-root (leaf nodes) hosts have been properly configured, as explained in the previous section and restarted, they will attempt to connect to the authentication server. When you open the KDI Management UI at *https://localhost:7701/login* and log in to KDIS using the admin credentials, you shall notice that on the first connection attempt, KDIS will accept the client as “Provisional” and until it is manually accepted, this leaf node will not be registered with the root node as a part of the larger KDI Fabric (network).



Fabric Devices

Manage the devices in the KDI fabric, create new credentials, and review provisional requests for approval or rejection.

Client Id	State	Description	Host
27d4ac11-5666-48...	Provisional		BCNLAB0
88131014-dfad-4a...	Provisional		BCNFPGA1
d3a254eb-65ac-40...	Accepted	Root KDIC	
feb3cb70-d26f-4e8...	Provisional		BCNQA1

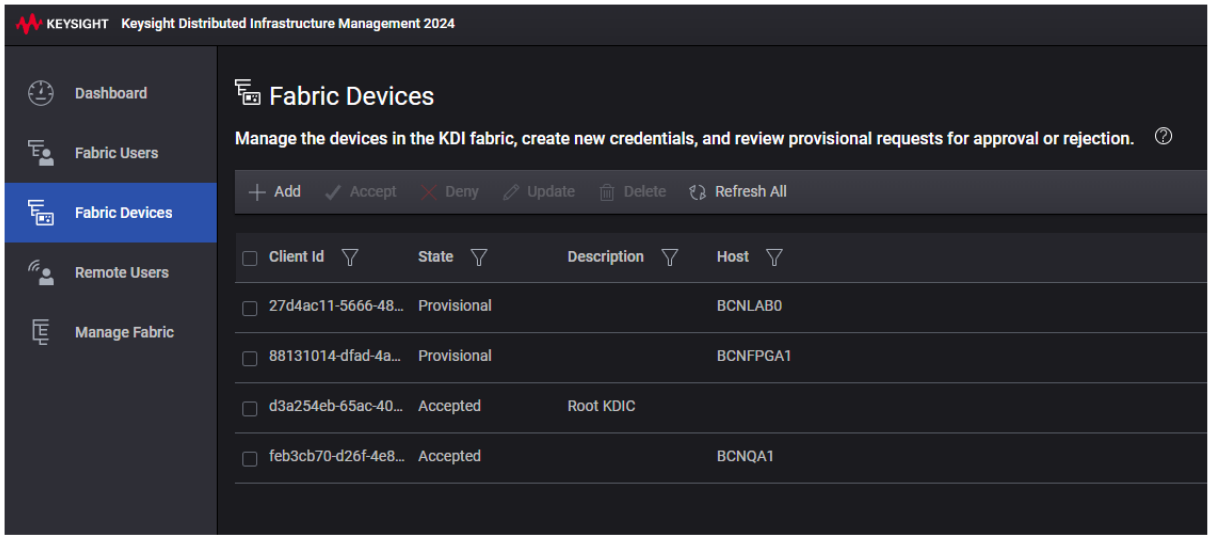
Figure 42 Provisional state of the leaf node on the first connection attempt to the root node

To accept a leaf-node into KDIS, in the **Fabric Devices** window:

1 Select each non-root host (client) with Provisional state. For example, BCNQA1 as shown in Figure 42.

2 Click **Accept**, if it is a valid host. The leaf-node BCNQA1 is now in Accepted state as shown in Figure 43.

The root-node where KDIS is installed also appear in the client list, but it is accepted by default.



Fabric Devices

Manage the devices in the KDI fabric, create new credentials, and review provisional requests for approval or rejection.

Client Id	State	Description	Host
27d4ac11-5666-48...	Provisional		BCNLAB0
88131014-dfad-4a...	Provisional		BCNFPGA1
d3a254eb-65ac-40...	Accepted	Root KDIC	
feb3cb70-d26f-4e8...	Accepted		BCNQA1

Figure 43 Accepted state of the leaf node

Section 4.4: Using KDI on root node

After you have configured the root and leaf nodes, use one of the following methods, that is, either the SW API or the SFP, to include the KDI elements to remotely access modules on one or more hosts.

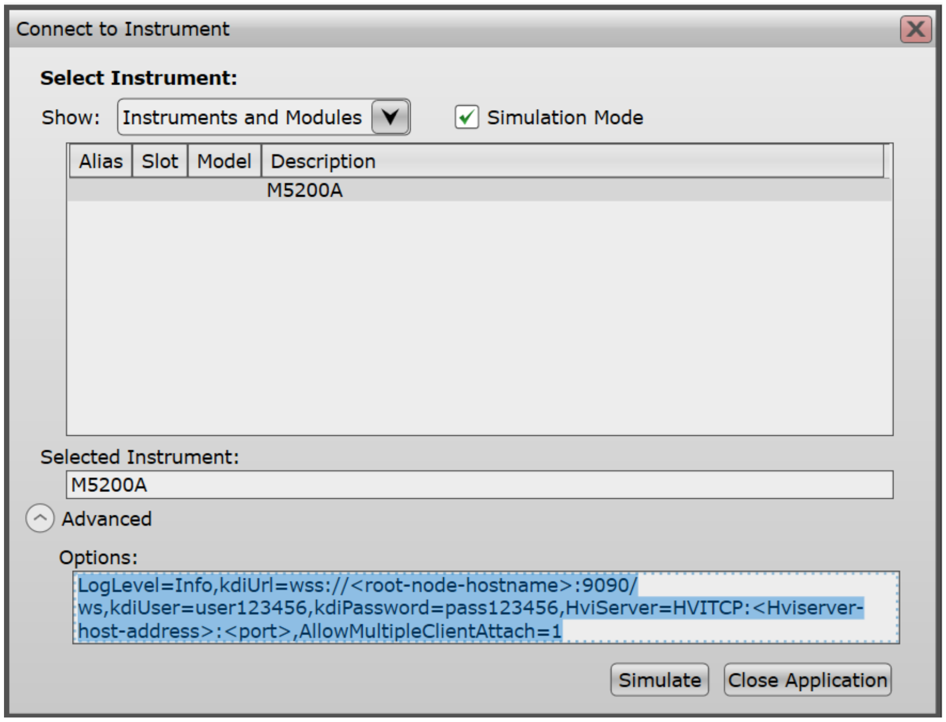
Method 1: Including KDI in SW API scripts

The following snippet from a Python script shows how to define the elements of KDI in the `resource` and `options` parameters.

```
# Open module on the leaf node using KDI on the root node
resource = 'kdi://<leaf-node-hostname>/PXI0::CHASSIS1::<slot-number>::INDEX0::INSTR'
query = False
reset = False
options =
'Simulate=0,LogLevel=Info,kdiUrl=wss://<root-node-hostname>:9090/ws,kdiUser=user123456,kdiPass
word=pass123456,HviServer=HVITCP:<Hviserver-host-address>:<port>,AllowMultipleClientAttach=1'
module = keysight_ktm5200x.KtM5200x(resource, query, reset, options)
```

Method 2: Including KDI in M5200x SFP

Figure 44 shows the **Options** area in the **Connect to Instrument** window of the M5200x SFP, where the KDI elements can be included.



LogLevel=Info,kdiUrl=wss://<root-node-hostname>:9090/ws,kdiUser=user123456,kdiPassword=pass123456,HviServer=HVITCP:<Hviserver-host-address>:<port>,AllowMultipleClientAttach=1

Figure 44 Configuring KDI elements in the SFP options

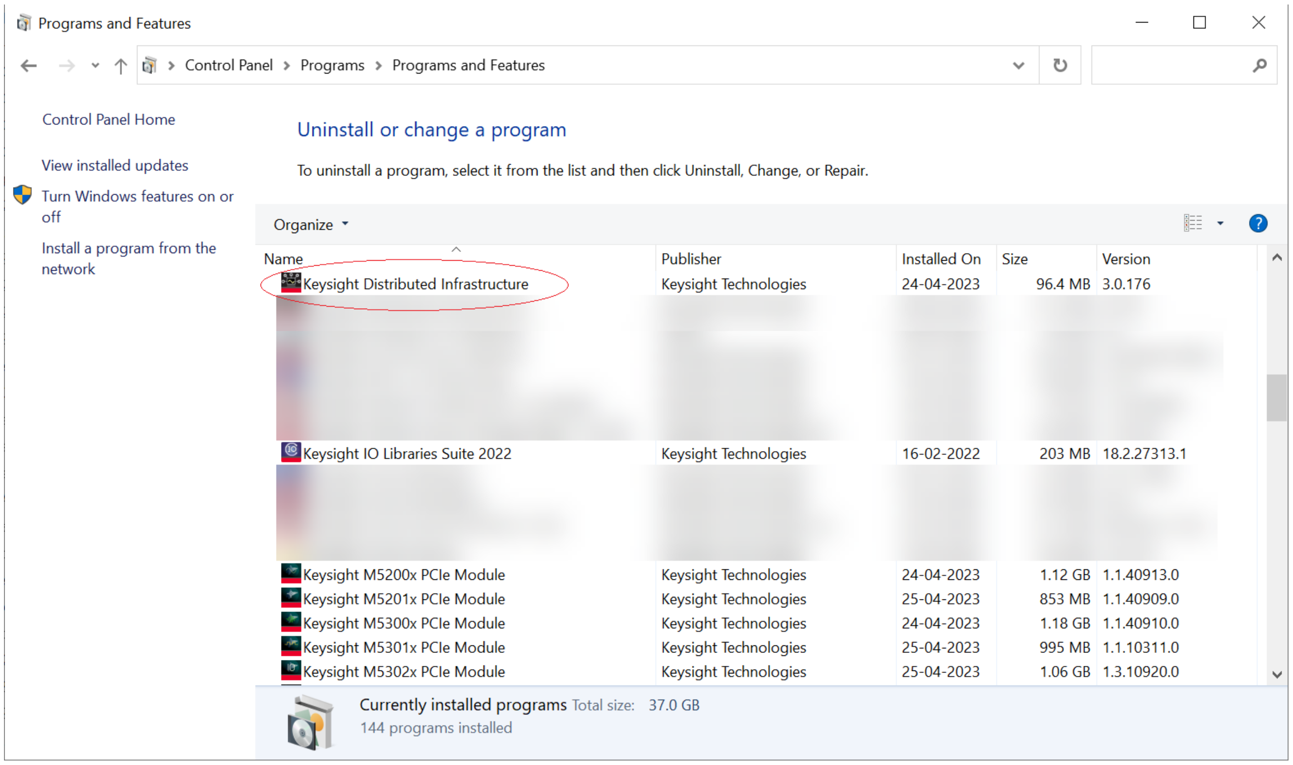
Table 40 describes some of the important variables from the snippet above.

Table 40 Description of important variables in snippet to configure KDI

Key elements to configure KDI	Description
`<leaf-node-hostname>`	Host name or IP address of the leaf node, which is connected to the root node.
`<slot-number>`	Slot number in the PXIe chassis, where the specific module is inserted.
`kdiUrl`	URL to access to the KDI Infrastructure. By default, it is wss://<root-node-hostname>:9090/ws.
`<root-node-hostname>`	Host name or IP address of the root node, which hosts the KDI Authentication Service. By default, it is set to *localhost* as the leaf node is controlled remotely by the root node in the KDI fabric.
`kdiUser`	Username to authenticate to KDI.
`kdiPassword`	Password to authenticate to KDI.
`<Hviserver-host-address>`	Address to open an HVI server
`<port>`	Port number corresponding to the HVI server
`AllowMultipleClientAttach`	Regardless of the setting of AllowMultipleClientAttach, a single connection to a single instrument is allowed. However, when AllowMultipleClientAttach is set to ‘1’, multiple connections to a single instrument are established, if all connections have been declared.

Section 4.5: Removing KDI from the root node

To remove the Keysight Distributed Infrastructure from the machine identified as root node, you must 'uninstall' the application from the Control Panel, as shown in Figure 45. Perform the steps described in the previous sections to configure another machine as the root node and connect other machines to the root as leaf nodes.


Programs and Features
Control Panel > Programs > Programs and Features
Control Panel Home
View installed updates
Turn Windows features on or off
Install a program from the network
Organize

Uninstall or change a program

To uninstall a program, select it from the list and then click Uninstall, Change, or Repair.

Name	Publisher	Installed On	Size	Version
Keysight Distributed Infrastructure	Keysight Technologies	24-04-2023	96.4 MB	3.0.176
Keysight IO Libraries Suite 2022	Keysight Technologies	16-02-2022	203 MB	18.2.27313.1
Keysight M5200x PCIe Module	Keysight Technologies	24-04-2023	1.12 GB	1.1.40913.0
Keysight M5201x PCIe Module	Keysight Technologies	25-04-2023	853 MB	1.1.40909.0
Keysight M5300x PCIe Module	Keysight Technologies	24-04-2023	1.18 GB	1.1.40910.0
Keysight M5301x PCIe Module	Keysight Technologies	25-04-2023	995 MB	1.1.10311.0
Keysight M5302x PCIe Module	Keysight Technologies	25-04-2023	1.06 GB	1.3.10920.0

Currently installed programs Total size: 37.0 GB
144 programs installed

Figure 45 Removing Keysight Distributed Infrastructure from the root node

5. Using M5300A High-Speed AWG Modules SFP

This chapter describes how to use the Keysight M5300A PXIe Module Soft Front Panel (SFP) software. Note that you may find one or more API / property associated with some of the SFP features. For a comprehensive list of the APIs pertaining to the programming language of your interest, refer to the Online Help files listed in “References to other documents” on page 9.

Section 5.1: Launching the M5300A PXIe Module SFP software

After the driver is installed, launch the SFP from the **Start** menu. Click **Start** > **Keysight M5300x PCIe Module** > **M5300x SFP (x64)**.

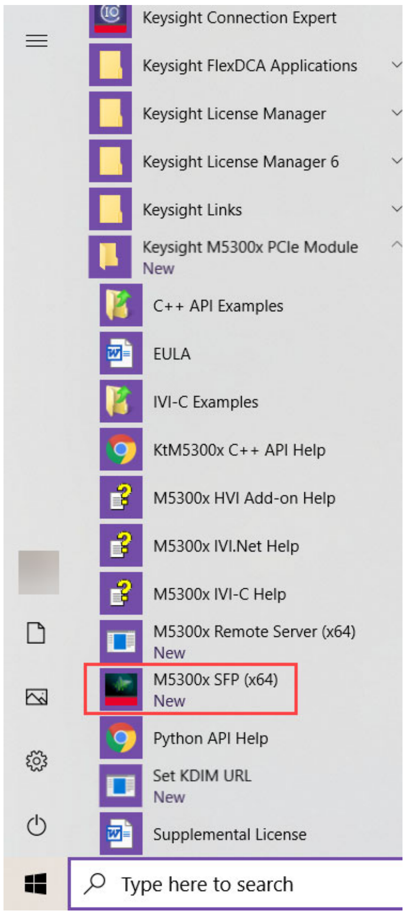
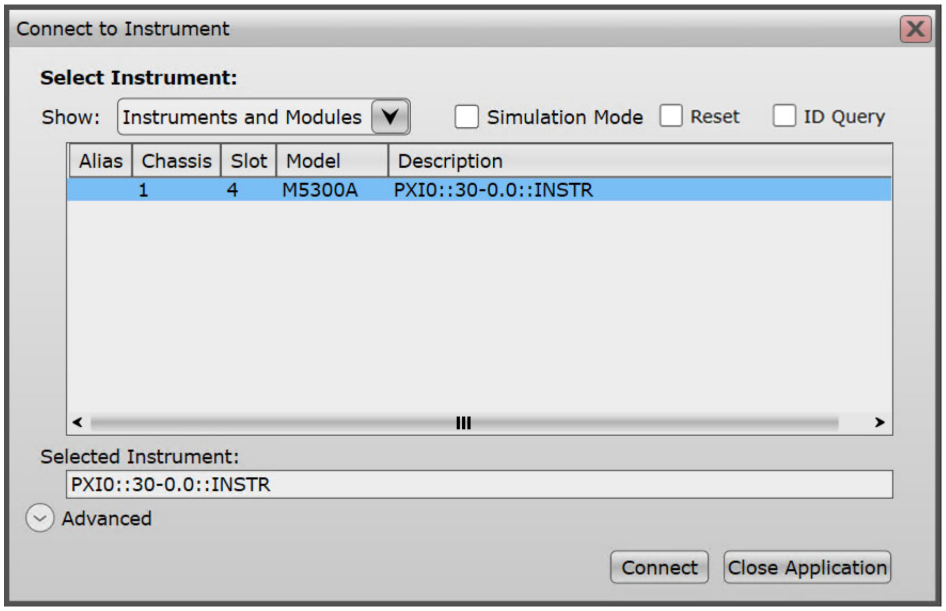


Figure 46 Launching M5300x SFP software from the Start menu

The Connect to Instrument window in the M5300A AWG SFP software is displayed, as shown in Figure 47 and Figure 48. Refer to “Connecting to Modules” on page 110 for more information about establishing connection in the ‘online mode’ and ‘simulation mode’.

Related Python APIs

```
keysight_ktm5300x.system.sfp.open()
```


Connect to Instrument

Select Instrument:

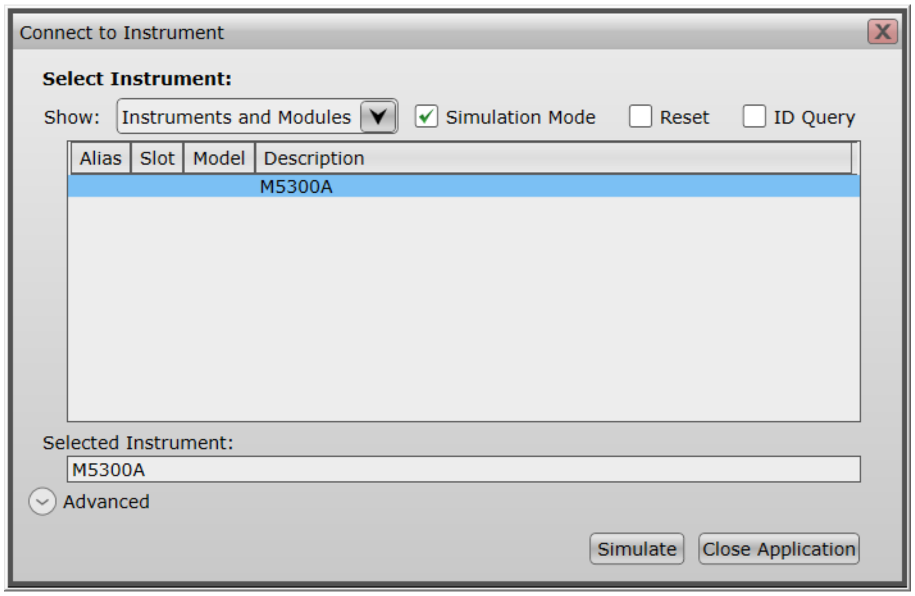
Show: Instruments and Modules ☐ Simulation Mode ☐ Reset ☐ ID Query

Alias	Chassis	Slot	Model	Description
	1	4	M5300A	PXI0::30-0.0::INSTR

Selected Instrument:

PXI0::30-0.0::INSTR

Figure 47 M5300x SFP software Connect to Instrument window (with one or more modules connected)


Connect to Instrument

Select Instrument:

Show: Instruments and Modules ✓ Simulation Mode ☐ Reset ☐ ID Query

Alias	Slot	Model	Description
			M5300A

Selected Instrument:

M5300A

Figure 48 M5300x SFP software Connect to Instrument window (Simulation Mode)

Related Python APIs

```
keysight_ktm5300x.KtM5300x(resource, query, reset, options)
```

Section 5.2: Familiarizing with the SFP software

The Soft Front Panel (SFP) interface for the M5300A RF AWG module provides a simple tool to help you accomplish one or more of the following tasks:

- Check the module's connection status
- Save and, if required, delete (saved) module connections
- Control single-ended and differential IO signals
- Perform firmware updates and hardware upgrades
- Monitor Driver calls

The SFP has the following elements in its user interface:

- Menus
- Status Indicators

5.2.1: Menus

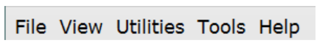
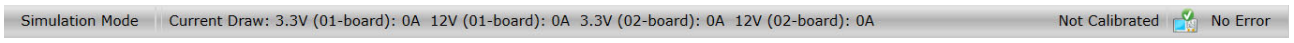


Figure 49 Menu options in the SFP

To understand the features that appear in the Menu options shown in Figure 49, see:

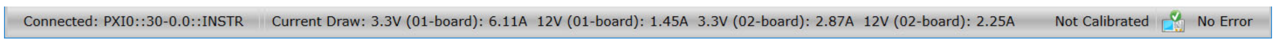
- File menu options
- View menu options
- Utilities menu options
- Tools menu options
- Help menu options

5.2.2: Status Indicators



Simulation Mode | Current Draw: 3.3V (01-board): 0A 12V (01-board): 0A 3.3V (02-board): 0A 12V (02-board): 0A | Not Calibrated | No Error

Figure 50 Status indicator tab in the SFP (simulation mode)



Connected: PXI0::30-0.0::INSTR | Current Draw: 3.3V (01-board): 6.11A 12V (01-board): 1.45A 3.3V (02-board): 2.87A 12V (02-board): 2.25A | Not Calibrated | No Error

Figure 51 Status indicator tab in the SFP (online mode)

To understand the status information that appear on this tab, as shown in Figure 50 and Figure 51, see Checking Instrument Status.

Section 5.3: Understanding Menu options

5.3.1: File menu options

The **File** menu provides the following options.

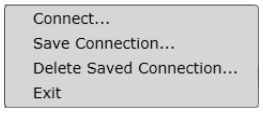
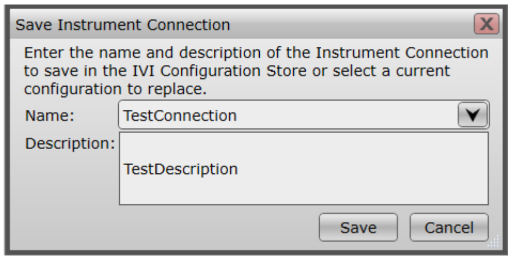


Figure 52 Options in the File menu

- **Connect...**: This option is used to connect the soft front panel (SFP) to a module in the chassis or to run in simulation mode. See Connecting to Modules.
- **Save Connection...**: This option is used to save an instrument configuration in the IVI Configuration Store. This process may also be referred to as Creating a Connection Alias.
- **Delete Saved Connection...**: This option is used to delete an instrument configuration from the IVI Configuration Store.
- **Exit**: This option is used to close the SFP session.

To save a connection:

1 Select **File** > **Save Connection...**.


Save Instrument Connection

Enter the name and description of the Instrument Connection to save in the IVI Configuration Store or select a current configuration to replace.

Name: TestConnection

Description: TestDescription

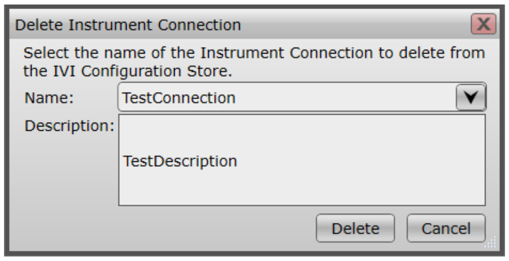
Figure 53 Saving instrument connection

2 In the **Name** field, specify a name for the connection to be saved. Optionally, you may add an appropriate description for the connection in the **Description** field.

3 Click **Save**. If the name specified for the connection (in the **Save Instrument Connection** dialog box) exists for another pre-saved connection, a prompt is displayed before the connection is replaced (overwritten) with the new connection.

To delete a connection:

1 Select **File** > **Delete Saved Connection...**.



Select the name of the Instrument Connection to delete from the IVI Configuration Store.

Name: TestConnection

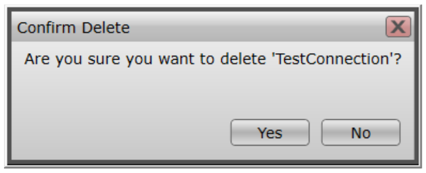
Description: TestDescription

Figure 54 Deleting instrument connection

2 From the **Name** drop-down list on the **Delete Instrument Connection** dialog box, select the name of the connection, which is to be deleted.

3 Click **Delete**.

4 On the **Confirm Delete** prompt, click “Yes” to proceed with deleting the connection.



Are you sure you want to delete 'TestConnection'?

Figure 55 Prompt to confirm deletion

5.3.2: View menu options

The **View** menu provides the following options.

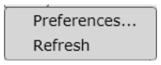
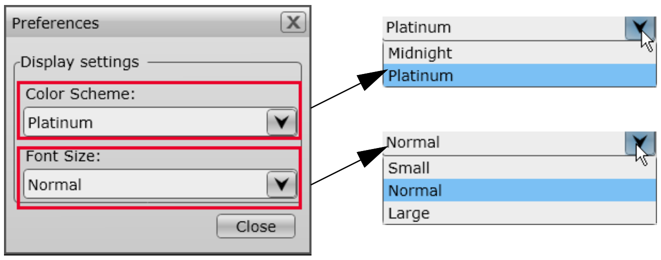


Figure 56 Options in the View menu

- **Preferences...**: This option opens the **Preferences** window, which is used to adjust the color scheme and the current font size.

  These preference values can be changed anytime between SFP sessions, so you do not have to set them each time the SFP is opened.



Color Scheme: Platinum

Font Size: Normal

Figure 57 Options in the Preferences sub-menu

- **Refresh**: Use this option to manually refresh the module state.

  In addition to overall status, the **Refresh** control synchronizes the SFP with the current hardware / driver settings.

Related Python APIs

`keysight_ktm5300x.system.sfp.refresh(wait_until_done: bool)`

5.3.3: Utilities menu options

The **Utilities** menu provides the following options.

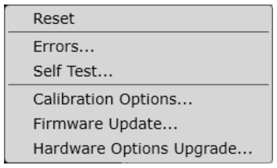


Figure 58 Options in the Utilities menu

- **Reset**: Resets the module to its default settings.

Related Python APIs

`keysight_ktm5300x.utility.reset()`

`keysight_ktm5300x.utility.reset_with_defaults()`

For more information, refer to Resetting to Default Factory Settings.

- **Errors...**: Lists the errors from the error queue of the M5300x SW.

Related Python APIs

`<tuple[int, str]> keysight_ktm5300x.utility.error_query()`

For more information, refer to Viewing Errors log.

- **Self Test...**: Displays the **Self Test** window, where you can run a Self Test on the module.

Related Python APIs

`<tuple[int, str]> keysight_ktm5300x.utility.self_test()`

For more information, refer to Conducting a Self Test.

- **Calibration Options...**: Displays the **Calibration Information and Preferences** window, where you can modify the calibration schedule, which allows you to manage and set the calibration intervals and warnings.

  Related Python APIs

  `<str> keysight_ktm5300x.calibration.adjustment_information`

  or

  `<str> keysight_ktm5300x.modules[<name_or_index>].calibration.adjustment_information`

  `<str> keysight_ktm5300x.calibration.due_date`

  or

  `<str> keysight_ktm5300x.modules[<name_or_index>].calibration.due_date`

  `<str> keysight_ktm5300x.calibration.instrument_identifier`

  `<str> keysight_ktm5300x.calibration.verification_information`

  or

  `<str> keysight_ktm5300x.modules[<name_or_index>].calibration.verification_information`

  `keysight_ktm5300x.calibration.status`

  or

  `keysight_ktm5300x.modules[<name_or_index>].calibration.status` returns a member from the enumeration `CalibrationStatus`:

`DUE`

`EXPIRED`

`INSTRUMENT_CALIBRATED`

`MODULES_CALIBRATED`

`NOT_CALIBRATED`

`NOT_SUBJECT_TO_CALIBRATION`

For more information, refer to Setting Calibration Options.

- **Firmware Update...**: Displays the **Firmware Update** window, where you can perform firmware update on the M5300A module.

Related Python APIs

`keysight_ktm5300x.system.update_firmware(wait_for_update: bool)`

returns a member from the enumeration `UpdateStatus:`

`IN_PROGRESS`

`SUCCESS`

`RESTART_REQUIRED`

`POWER_CYCLE_REQUIRED`

`FAILED`

and

`keysight_ktm5300x.system.wait_for_update_firmware_complete(timeout_milliseconds: int)`

returns a `<tuple[UpdateStatus, int]>,` where a member from the enumeration `UpdateStatus` as defined above is returned and `<int>` represents the progress in percent.

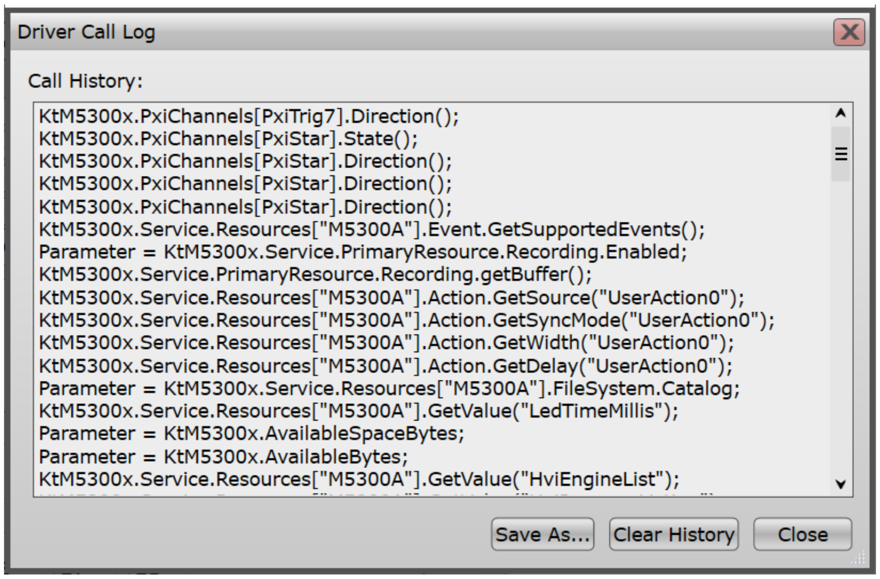
For more information, refer to Updating Firmware.

5.3.4: Tools menu options

The **Tools** menu provides the following options.

Figure 59 Options in the Tools menu

- **Monitor Driver Calls**: This option opens the **Driver Call Log** window, where a list of the driver calls are displayed in the **Call History** area. These driver calls correspond to each action you perform on the SFP.



```
KtM5300x.PxiChannels[PxiTrig7].Direction();
KtM5300x.PxiChannels[PxiStar].State();
KtM5300x.PxiChannels[PxiStar].Direction();
KtM5300x.PxiChannels[PxiStar].Direction();
KtM5300x.PxiChannels[PxiStar].Direction();
KtM5300x.Service.Resources["M5300A"].Event.GetSupportedEvents();
Parameter = KtM5300x.Service.PrimaryResource.Recording.Enabled;
KtM5300x.Service.PrimaryResource.Recording.getBuffer();
KtM5300x.Service.Resources["M5300A"].Action.GetSource("UserAction0");
KtM5300x.Service.Resources["M5300A"].Action.GetSyncMode("UserAction0");
KtM5300x.Service.Resources["M5300A"].Action.GetWidth("UserAction0");
KtM5300x.Service.Resources["M5300A"].Action.GetDelay("UserAction0");
Parameter = KtM5300x.Service.Resources["M5300A"].FileSystem.Catalog;
KtM5300x.Service.Resources["M5300A"].GetValue("LedTimeMillis");
Parameter = KtM5300x.AvailableSpaceBytes;
Parameter = KtM5300x.AvailableBytes;
KtM5300x.Service.Resources["M5300A"].GetValue("HviEngineList");
```

Figure 60 Messages as they appear on the Driver Call Log window

To learn more about the IVI commands associated with the driver calls listed in the driver call log,

- For IVI-C Driver Help, go to the folder *C:\Program Files\IVI Foundation\IVI\Drivers\KtM5300x*.
- For IVI-.NET Driver Help, go to the folder *C:\Program Files\IVI Foundation\IVI\Microsoft.NET\Framework64\v4.5.50709\Keysight.KtM5300x <build-version>\Help*.

To clear the driver call logs:

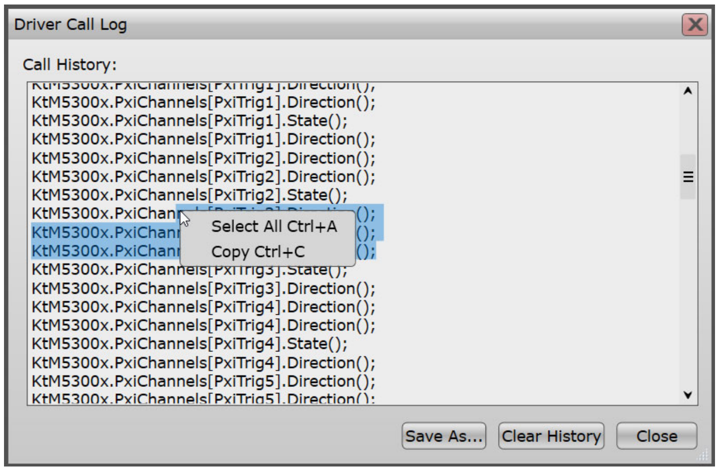
1 On the **Driver Call Log** window, click **Clear History**.

This clears the displayed driver call logs and refreshes the **Driver Call Log** window.

To copy and paste call history from the driver call log:

1 In the **Driver Call Log** window, highlight the call history to be copied. You may also use the right-click menu option **Select All Ctrl+A** to select the entire log displayed in the active instance.

2 Right-click in the text area and click **Copy Ctrl+C**.



```
KtM5300x.PxiChannels[PxiTrig1].Direction();
KtM5300x.PxiChannels[PxiTrig1].State();
KtM5300x.PxiChannels[PxiTrig1].Direction();
KtM5300x.PxiChannels[PxiTrig2].Direction();
KtM5300x.PxiChannels[PxiTrig2].Direction();
KtM5300x.PxiChannels[PxiTrig2].State();
KtM5300x.PxiChan
KtM5300x.PxiChan
KtM5300x.PxiChan
KtM5300x.PxiChannels[PxiTrig3].State();
KtM5300x.PxiChannels[PxiTrig3].Direction();
KtM5300x.PxiChannels[PxiTrig4].Direction();
KtM5300x.PxiChannels[PxiTrig4].Direction();
KtM5300x.PxiChannels[PxiTrig4].State();
KtM5300x.PxiChannels[PxiTrig4].Direction();
KtM5300x.PxiChannels[PxiTrig5].Direction();
KtM5300x.PxiChannels[PxiTrig5].Direction();
```

Figure 61 Right-click menu options in the Driver Call Log window

3 Paste the call log entries from the system's memory on to any editor, such as Notepad, and so on.

To save a copy of the Driver Call Log to a file:

1 Click **Save As...**.

A file system browser opens, which enables you to navigate any folder and save the Driver Call Log.

2 Navigate to a folder, specify a file name, and click **Save**.

5.3.5: Help menu options

The **Help** menu provides the following options.

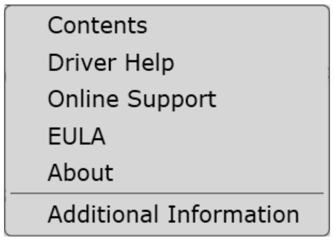
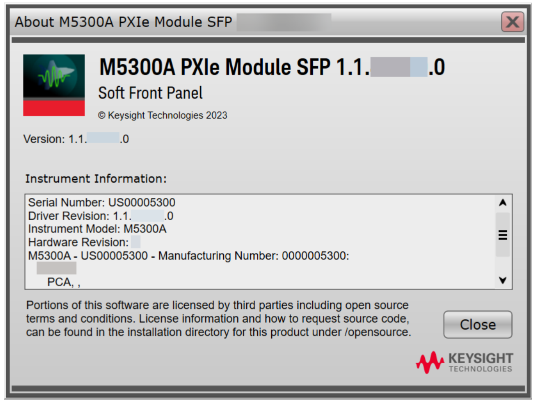


Figure 62 Options in the Help menu

- **Contents**: Opens the Online Help system related to SFP.
- **Driver Help**: Opens the Online Help system related to IVI.NET driver.
- **Online Support**: Opens a browser web page to Keysight Technologies support.
- **EULA**: Opens the Keysight software's end user license agreement, associated with the SFP software.
- **About**: Opens a window that displays the driver version information and module information, such as the serial number, hardware revision, and license options.


About M5300A PXIe Module SFP [redacted]

M5300A PXIe Module SFP 1.1.[redacted].0

Soft Front Panel

© Keysight Technologies 2023

Version: 1.1.[redacted].0

Instrument Information:

```
Serial Number: US00005300
Driver Revision: 1.1.[redacted].0
Instrument Model: M5300A
Hardware Revision: [redacted]
M5300A - US00005300 - Manufacturing Number: 0000005300:
  [redacted]
    PCA, ,
```

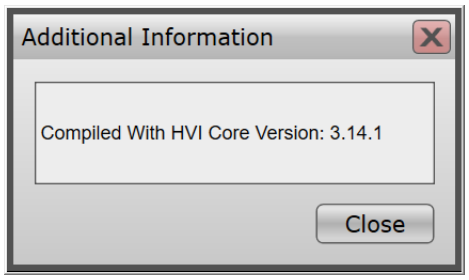
Portions of this software are licensed by third parties including open source terms and conditions. License information and how to request source code, can be found in the installation directory for this product under /opensource.

Figure 63 About window in SFP

Related Python APIs

`<str> keysight_ktm5300x.system.about`

- **Additional Information**: Opens a window that displays the HVI Core version that the M5300x SW drivers are compiled with.



Compiled With HVI Core Version: 3.14.1

Figure 64 Additional Information window in SFP

Section 5.4: Performing specific tasks in the M5300x SFP Software

This section provides how to information of various features in M5300x SFP. These features are explained in details in the following topics:

- Connecting to Modules
- Managing Module Run Logs
- Checking Instrument Status
- Configuring the SFP features
- Resetting to Default Factory Settings
- Viewing Errors log
- Conducting a Self Test
- Setting Calibration Options
- Updating Firmware

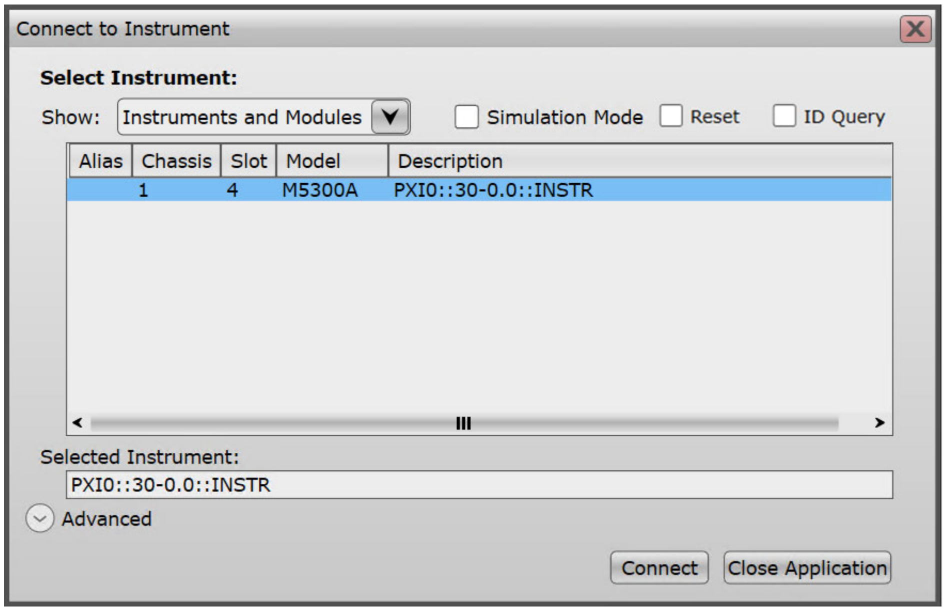
As you use the SFP to perform various actions, Keysight recommends to use **Tools** > **Monitor Driver Calls** to bring up the **Driver Call Log** window. This enables you to view the methods and properties that are used to program the module.

5.4.1: Connecting to Modules

The **Connect to Instrument** dialog enables you to select an M5300A module connected to a specific slot with a unique VISA address.

The selected configuration performs either of the following operations:

- Connects to the SFP of the physical hardware in a chassis
- Simulates the SFP session but not with an actual hardware connection



Alias	Chassis	Slot	Model	Description
	1	4	M5300A	PXI0::30-0.0::INSTR

Selected Instrument: PXI0::30-0.0::INSTR

Figure 65 Connect to Instrument window in the M5300x SFP

Connecting via PXI VISA address

To connect to a Module or Saved Instruments:

- Select one of the following options from the **Show** drop-down list. The displayed modules / configuration is filtered by the option you select in the **Show** drop-down list.
  - **Modules** – This element shows modules found through VISA discovery.
  - Saved Instruments – This element shows the known instruments, which are defined by saving a configuration into the **IVI Configuration Store** after the SFP is up and running.
- The selected item is highlighted and displayed in the **Selected Instrument** entry box. An IVI logical instrument, a module, or Saved Instrument may also be selected by typing its name directly into the **Selected Instrument** entry box.

In this connection mode, you may optionally:

- select the **Reset** check box (disabled by default) to reset the module's properties to the default values/settings. See "Resetting to Default Factory Settings" on page 131.
- select the **ID Query** check box to specify whether to verify the ID of the selected module.

Connecting via Simulation mode

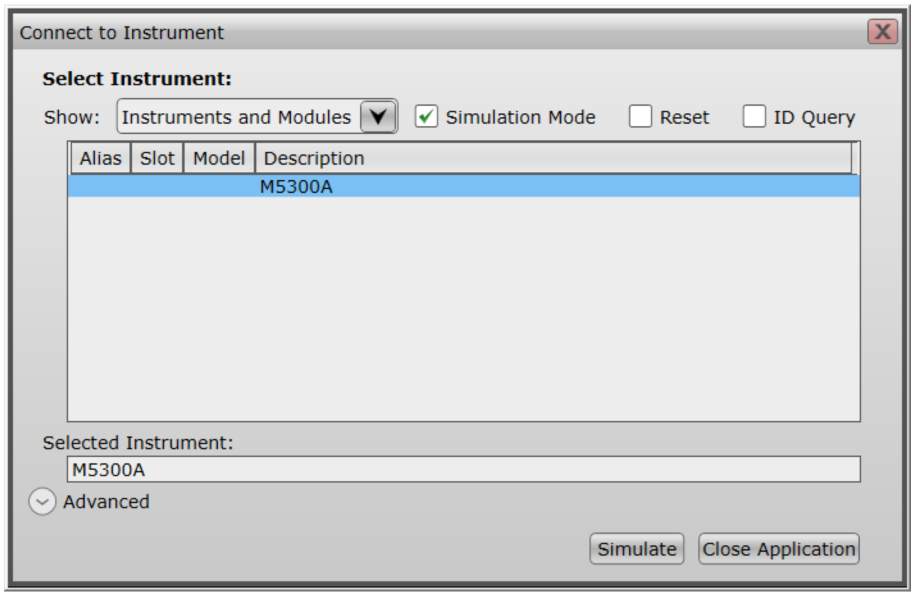
- **Simulation Mode**: When enabled, you can select the type of hardware which is being simulated. This works in conjunction with the Simulate button.
  - **Connect** / **Simulate** – When this button is labeled as **Connect**, it initiates the connection to the selected instrument or hardware module. If **Simulation Mode** is checked, this button is labeled as **Simulate**, as shown in Figure 66. Click the button to initiate a session with only the driver(s) for the selected instrument or module (not an actual hardware connection).

Related Python APIs

`<str> keysight_ktm5300x.driver_operation.simulate`

To set simulation mode, assign a `<bool>` value.

If you select the **Simulation Mode** check box and click the **Simulate** button after selecting a module / configuration, the underlying IVI driver (that the SFP is using) is opened in simulation mode. In this case, the selected resources may be ignored.



Connect to Instrument

Select Instrument:

Show: Instruments and Modules [✓ Simulation Mode] Reset ID Query

Alias	Slot	Model	Description
			M5300A

Selected Instrument: M5300A

Figure 66 Simulation Mode in the Connect to Instrument window

NOTE An IVI driver can operate without the presence of an actual instrument by using simulation mode. When simulation is enabled, the driver does not communicate with the instrument but attempts to generate results, which allow client applications to run reasonably.

Simulation is designed to allow the driver to be used without an instrument or any I/O software or hardware. However, simulation is not intended to fully emulate the instrument. There is often no parameter range checking or coercion unless done in the driver. Most drivers depend on the instrument to perform range checking and coercion of parameter values. In simulation mode, this does not occur.

- **Advanced** – Select this control to view the **Options**: dialog.

Related Python APIs

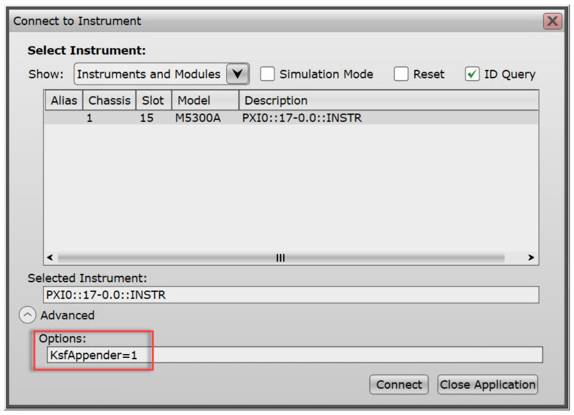
```
<str> keysight_ktm5300x.modules[<name_or_index>].options
<str> keysight_ktm5300x.system.options
```

5.4.2: Managing Module Run Logs

Beginning with the 1.3 release of the M5300x software, you can manage the runtime logs for the M5300A module that you are connecting to, using the M5300x SFP. Not only can you enable/disable module run logs in the M5300x SW, but also configure the level of detail that must be included, if and when logging is enabled.

By default, the "KsfAppender" flag has been preset to '0', by which logging during the runtime of the modules is disabled (set to OFF) for the M5300A RF AWGs.

To enable (set to ON) module run logs, you must manually pass the "KsfAppender=1" flag as one of the 'Advanced Options' while opening a M5300A module. See Figure 67.



Alias	Chassis	Slot	Model	Description
	1	15	M5300A	PXI0::17-0.0::INSTR

Selected Instrument: PXI0::17-0.0::INSTR

Advanced

Options: KsfAppender=1

Figure 67 Setting Advanced Option 'KsfAppender=1' to enable module run logs

Once the driver session is terminated for a specific module, the M5300A module's run logs are saved in a TEXT file named M5300A-<*connection-mode*>-Log.txt in the folder *C:\TEMP\M5300xLogging\ModuleRunLogs*. Here, <connection-mode> is "Simulate" if the module is opened in the Simulation mode, else it displays the VISA address of that module.

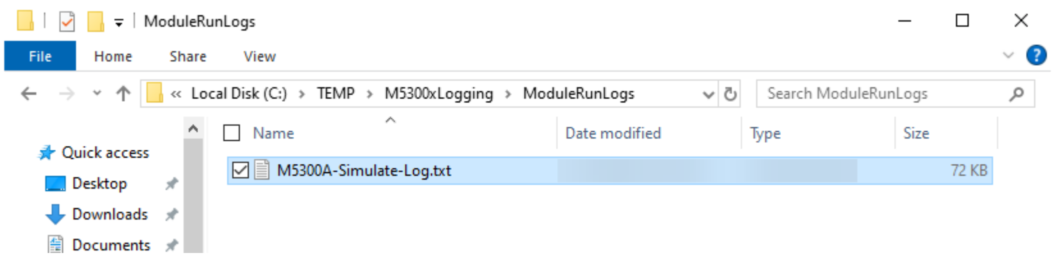
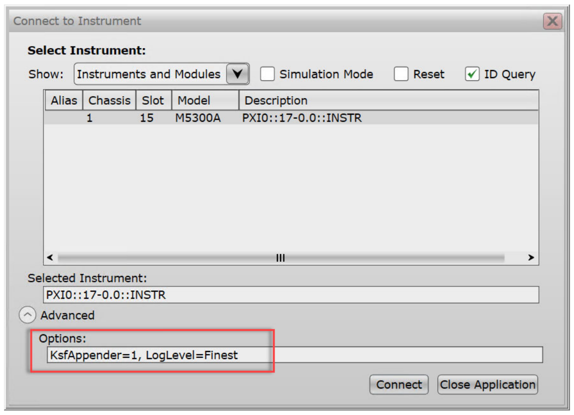


Figure 68 Module log files saved at the end of the driver session of an M5300A module in Simulate mode

NOTE The Log file is overwritten for each driver session (that is, each time you open) for a particular module, where logging is enabled. The log file for each specific driver instance is not appended.

As mentioned earlier, you can set a specific level of logging detail along with or after you enable logging. It is set by passing "LogLevel" flag as one of the 'Advanced Options', while opening the module. See Figure 69.



Alias	Chassis	Slot	Model	Description
	1	15	M5300A	PXI0::17-0.0::INSTR

Selected Instrument: PXI0::17-0.0::INSTR

Options: KsfAppender=1, LogLevel=Finest

Figure 69 Setting Advanced Option 'LogLevel' for managing details when logging

Table 41 lists the supported log level options that you may set as a value to the LogLevel flag. Note that "LogLevel=Info" is the default option if you do not specify this flag.

Table 41 Supported log level options

LogLevel option	Description
Finest	logs finest, fine, debug, info, warning, error, fatal
Fine	logs fine, debug, info, warning, error, fatal
Debug	logs debug, info, warning, error, fatal
Info (DEFAULT)	logs info, warning, error, fatal
Warning	logs warning, error, fatal
Error	logs error, fatal
Fatal	logs fatal

5.4.3: Checking Instrument Status

The status tab, which is displayed at the bottom of the SFP window, shows the status of the module in your Keysight instrument. You may also click **Refresh** to update the status (**View** > **Refresh**).

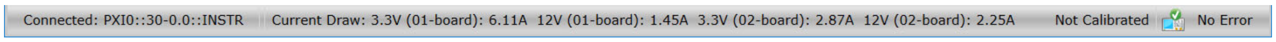

Connected: PXI0::30-0.0::INSTR | Current Draw: 3.3V (01-board): 6.11A 12V (01-board): 1.45A 3.3V (02-board): 2.87A 12V (02-board): 2.25A | Not Calibrated | No Error

Figure 70 Status indicator when module is connected in the M5300x SFP

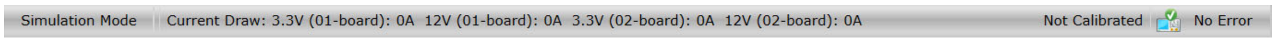

Simulation Mode | Current Draw: 3.3V (01-board): 0A 12V (01-board): 0A 3.3V (02-board): 0A 12V (02-board): 0A | Not Calibrated | No Error

Figure 71 Status indicator in Simulation Mode in the M5300x SFP

The bottom-left corner of the SFP displays the following operational mode:

- **Simulation Mode**: This mode is displayed when the SFP is being used in “Simulation Mode” and no real hardware is being communicated with or controlled.
- **Not Connected**: This mode is displayed when no connections have been established with the module. This is typically displayed when the dialog box first opens instead of the **Connect to Instrument** window.
- **Connected**: This mode is displayed when the SFP has established a connection with a module.
  - If a single module is connected, the bottom-left corner shows “Connected” along with the VISA address.
  - If an alias is selected as the instrument to connect to, the bottom-left corner shows the alias.
  - The current drawn at Voltage values of 3.3 V and 12 V are also displayed.

Related Python APIs

```
<int> keysight_ktm5300x.modules[<name_or_index>].current_draws.count
<str> keysight_ktm5300x.modules[<name_or_index>].current_draws[<name_or_index>].name
<float> keysight_ktm5300x.modules[<name_or_index>].current_draws[<name_or_index>].value
```

- If you perform a factory reset on the connected module, the status tab displays “Reset complete” upon successful reset of the hardware settings. See Resetting to Default Factory Settings.

The bottom-right corner of the SFP displays the current status of the Calibration and Error Log:

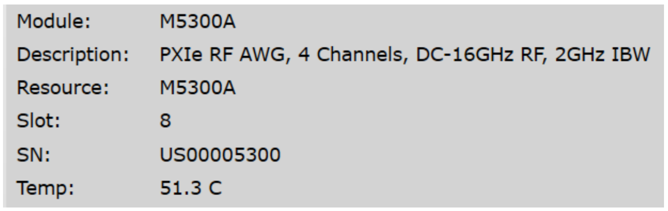
- **Calibration status** – Displays the calibration type to which the module is subjected to.

NOTE **Calibration is not supported in the current version of the M5300x SFP. Therefore, the status shall display as “Not Subject to Periodic Calibration”.**

- **No Error** – Displayed when there are no errors to report and the “Error Log” is empty.
- **Error** – Displayed when errors have occurred. You may click “Error” on the status bar or click **Utilities** > **Errors** to view the **Errors** window. Click **Get Instrument Errors** to view any errors encountered on the instrument.

5.4.4: Configuring the SFP features

This section describes the features/functionalities that are displayed on the Soft Front Panel (SFP).



Module:	M5300A
Description:	PXIe RF AWG, 4 Channels, DC-16GHz RF, 2GHz IBW
Resource:	M5300A
Slot:	8
SN:	US00005300
Temp:	51.3 C

Figure 72 Read-only information about the module on the SFP

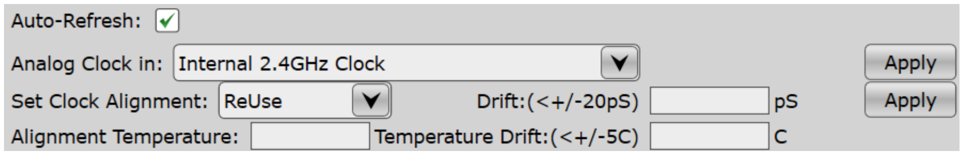
Read-only data

The SFP for M5300A modules displays certain read-only data:

- **Module** — displays the name/model number of the module.
- **Description** — displays any information associated with the module.
- **Resource** — displays the resource information.
- **Slot** — displays the slot in the PXIe chassis where the physical card has been inserted.
- **SN** — displays the serial number of the module.
- **Temp** — displays the operating temperature of the module.

Related Python APIs

```
<str> keysight_ktm5300x.modules[<name_or_index>].name
<str> keysight_ktm5300x.modules[<name_or_index>].additional_information
<str> keysight_ktm5300x.modules[<name_or_index>].serial_number
<int> keysight_ktm5300x.modules[<name_or_index>].slot
<float> keysight_ktm5300x.modules[<name_or_index>].temperature
```



Auto-Refresh: ✓
Analog Clock in: Internal 2.4GHz Clock
Set Clock Alignment: ReUse
Drift:(<+/-20pS) pS
Alignment Temperature:
Temperature Drift:(<+/-5C) C

Figure 73 Auto-Refresh and Clock settings on the SFP

Auto-Refresh

The **Auto-Refresh** check box is selected by default. If this function is enabled, the SFP automatically modifies the 'read-only' data pertaining to the module. For example, the 'Temp' field shall display the real-time operating temperature, based on its variations on the module.

Related Python APIs

```
<bool> keysight_ktm5300x.system.sfp.auto_refresh.enabled
<float> keysight_ktm5300x.system.sfp.auto_refresh.period
```

Configuring Clock source and alignment settings

Analog Clock is the precision 2.4 GHz clock from which all analog sample clocks are derived and determines parametric performance of analog channels.

1 Select a reference clock source from the **Analog Clock in** drop-down field:
   - **Internal 2.4GHz Clock** – Selected by default as the reference clock source. The M5300A module defaults to using PXIe_CLK100 for reference clock at power-on. The analog clock is thereby generated internally from the module's 100 MHz clock.
   - **External 2.4GHz Clock** – If selected, the module uses a common analog clock distributed externally to the system.

NOTE The M5300x software is designed to repeat the previous Analog Clock configuration in each new connection to driver unless the driver itself undergoes a power-cycle or if a failure (such as external clock unavailable) is detected. after the AWG starts running, the Analog Clock setting cannot be changed until the AWG is stopped.

2 Click **Apply** for the changes, if any, to take effect.

Related Python APIs

```
driver.clock.clock_source = keysight_ktm5300x.ClockSource.INTERNAL
```
or
```
driver.clock.clock_source = keysight_ktm5300x.ClockSource.EXTERNAL
```

Clock alignment is performed when one or more of the following cases occur:

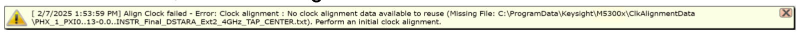
- the system is powered on
- any hardware is changed
- any clock configuration is changed

The features associated with configuring clock alignment are:

1 **Set Clock Alignment**: From the drop-down options, select one of the following options:
   - **ReUse**–(default) Configures the next clock alignment operation to reuse the calibration factors that were determined during the prior clock alignment performed using **Initialize** mode. Select this option if you previously performed an alignment using the **Initialize** option for the current hardware and clock configuration. This option preserves the inter-channel skews set by the prior alignment that used the **Initialize** option.
   - **Initialize**–Configures the next clock alignment operation to align the clocks and store calibration factors that will be used for subsequent alignments performed using the **ReUse** mode. Select this option if you have not previously performed an alignment using the **Initialize** option for the current hardware and clock configuration. This option may introduce small random skew errors, where you must perform de-skew manually.

   Note that Analog Clock should only be aligned if it is determined that some internal parameter of the module changed and re-alignment is necessary.

NOTE If you set Clock Alignment as "ReUse", the M5300x SW checks for any existing calibration data for the clock. If no calibration data is available for the SW to reuse, the following error is returned.



```
[ 2/7/2025 1:53:59 PM] Align Clock failed - Error: Clock alignment : No clock alignment data available to reuse (Missing File: C:\ProgramData\Keysight\M5300x\ClkAlignmentData
\PHX_1_PXI0..13-0.0..INSTR_Final_DSTARA_Ext2_4GHz_TAP_CENTER.txt). Perform an initial clock alignment.
```

In such scenarios, set Clock Alignment to "Initialize" to perform clock calibration.

Related Python APIs

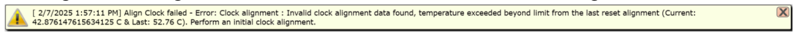
`keysight_ktm5300x.clock.align()`

2 Click **Apply** corresponding to the Clock Alignment setting so that the following 'read-only' information is displayed.

- **Alignment Temperature**—Displays the temperature in degree Celsius that is attained by the module when clock alignment setting is configured or changed.
- **Drift**—Displays the drift in picoseconds on the signal that results from the configuration or change in the clock alignment setting.
- **Temperature Drift**—Displays the drift in the temperature in degree Celsius that results from the configuration or change in the clock alignment setting.

NOTE

If the M5300x SW detects that the alignment temperature has varied by more than 10 degrees Celsius from the last time reset alignment was done, setting Clock Alignment as "ReUse" returns the following error:



```
[ 2/7/2025 1:57:11 PM] Align Clock failed - Error: Clock alignment : Invalid clock alignment data found, temperature exceeded beyond limit from the last reset alignment (Current:
42.876147615634125 C & Last: 52.76 C). Perform an initial clock alignment.
```

In such scenarios, set Clock Alignment to "Initialize" to perform clock calibration.

Related Python APIs

`<alignment_status> keysight_ktm5300x.clock.alignment_status`

returns one of the members from the enumeration `AlignmentStatus:`

`NOT_ALIGNED`

`TEMPORARY`

`ALIGNED`

`INVALID_ALIGNMENT`

`<float> keysight_ktm5300x.clock.alignment_temperature()`

`<float> keysight_ktm5300x.clock.drift_temperature()`

`<str> keysight_ktm5300x.clock.drift_time()`

`keysight_ktm5300x.clock.invalidate_current_alignment()`

NOTE

The Clock Alignment properties (Alignment Temperature, Drift and Temperature Drift) are reset whenever the "Analog Clock" configuration is changed or even when the existing setting is applied again.

For more information about Analog Clocks in the M5300A modules, see "About M5300A module's timebase" on page 21.

Configuring Waveform External Trigger Setup

1 On the SFP window, click the button corresponding to "**Waveform External Trigger Setup**".

The **Waveform External Trigger Setup** dialog box appears. This dialog box allows you to configure the external trigger to play waveform on the selected Channel.

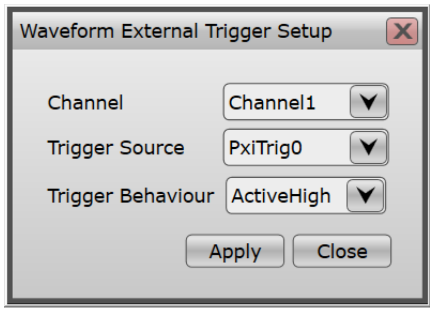


Figure 74 Default view of the Waveform External Trigger Setup dialog box

2 From the **Channel** drop-down field, select the Channel where the external trigger must be applied. **Channel1** is the default option.

3 From the **Trigger Source** drop-down field, select either one of the eight PXI Triggers, PXI Star or one of the eight SMB Triggers to send the trigger signal. **PXITrig0** is selected by default.

4 From the **Trigger Behaviour** drop-down field, select one of the following options:

- **ActiveHigh**—(default) indicates that the trigger signal must be sent when active high voltage on the signal is attained.
- **ActiveLow**—indicates that the trigger signal must be sent when active low voltage on the signal is attained.
- **RisingEdge**—indicates that the trigger signal must be sent when the rising edge on the signal is attained.
- **FallingEdge**—indicates that the trigger signal must be sent when the falling edge on the signal is attained.

5 Click **Apply** for the changes, if any, to take effect.

6 Click **Close** to return to the main window of the SFP.

Related Python ktM5300x APIs

`keysight_ktm5300x.channels[<name_or_index>].external_trigger_config(source, mode)`

replace `source` with one of the members from the enumeration `TriggerSource`:

```
PXI_TRIG0
PXI_TRIG1
PXI_TRIG2
PXI_TRIG3
PXI_TRIG4
PXI_TRIG5
PXI_TRIG6
PXI_TRIG7
PXI_STAR
SMB_TRIG1
SMB_TRIG2
SMB_TRIG3
SMB_TRIG4
SMB_TRIG5
SMB_TRIG6
SMB_TRIG7
SMB_TRIG8
```

replace `mode` with one of the members from the enumeration `ExternalTriggerMode`:

```
ACTIVE_HIGH
ACTIVE_LOW
RISING_EDGE
```

```
FALLING_EDGE
```

Related Python ktM5300xHvi APIs

```
<int> ktm5300xhvi.triggers.front_panel1
<int> ktm5300xhvi.triggers.pxi0
<int> ktm5300xhvi.triggers.pxi1
<int> ktm5300xhvi.triggers.pxi2
<int> ktm5300xhvi.triggers.pxi3
<int> ktm5300xhvi.triggers.pxi4
<int> ktm5300xhvi.triggers.pxi5
<int> ktm5300xhvi.triggers.pxi6
<int> ktm5300xhvi.triggers.pxi7
<int> ktm5300xhvi.triggers.smb1
<int> ktm5300xhvi.triggers.smb2
<int> ktm5300xhvi.triggers.smb3
<int> ktm5300xhvi.triggers.smb4
<int> ktm5300xhvi.triggers.smb5
<int> ktm5300xhvi.triggers.smb6
<int> ktm5300xhvi.triggers.smb7
<int> ktm5300xhvi.triggers.smb8
```

Configuring SW/HVI Trigger Control

1 On the SFP window, click the button corresponding to “**Waveform SW/HVI Trigger Control**”.

The **Waveform SW HVI Trigger Control** dialog box appears. This dialog box allows you to control the SW/HVI trigger to play waveform on one or more Channels.

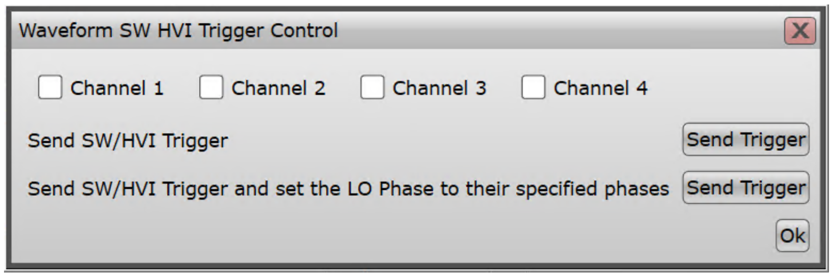

Waveform SW HVI Trigger Control
Channel 1
Channel 2
Channel 3
Channel 4
Send SW/HVI Trigger
Send Trigger
Send SW/HVI Trigger and set the LO Phase to their specified phases
Send Trigger
Ok

Figure 75 Default view of the Waveform SW HVI Trigger Control dialog box

1 Select one or more Channels, where you wish to send the trigger. No Channel is selected by default.

2 To send a triggering signal on one or more selected Channels,

- Click the **Send Trigger** button corresponding to **Send SW/HVI Trigger** to trigger waveform play using any triggering software or using an HVI function.
- Click the **Send Trigger** button corresponding to **Send SW/HVI Trigger and set the LO Phase to their specified phases** to trigger waveform play using any triggering software or using an HVI function. In this case, the M5300x SW synchronizes the phase set by the SW / HVI function with that of the Local Oscillator in the M5300A module’s DAC.

NOTE

If you set the value of LO Frequency in multiples of 600 MHz (which is the fundamental frequency of M5300A modules), the M5300x SW automatically synchronizes the LO phase. However, for all other LO Frequency values, you must click the **Send Trigger** button corresponding to **Send SW/HVI Trigger and set the LO Phase to their specified phases** to reset the LO phase.

3 Click **OK** to save changes and exit the dialog box.

Related Python ktM5300x APIs

`keysight_ktm5300x.awg_multiple.trigger(channel_mask, apply_phase)`

assign an `<int>` value to `channel_mask` and a `<bool>` value to `apply_phase`.

`keysight_ktm5300x.channels[<name_or_index>].awg_trigger()`

`keysight_ktm5300x.channels[<name_or_index>].phase`

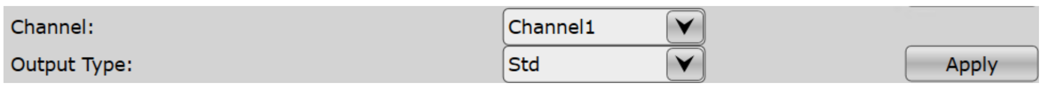
assign a `<float>` value to `phase`.

Related Python ktM5300xHvi APIs

```
<int> ktm5300xhvi.actions.awg1_trigger
<int> ktm5300xhvi.actions.awg2_trigger
<int> ktm5300xhvi.actions.awg3_trigger
<int> ktm5300xhvi.actions.awg4_trigger
ktm5300xhvi.instruction_set.set_phase.apply
ktm5300xhvi.instruction_set.set_phase.channel
<int> ktm5300xhvi.instruction_set.set_phase.id
<str> ktm5300xhvi.instruction_set.set_phase.name
ktm5300xhvi.instruction_set.set_phase.value
```

Configuring Channel and Output

One or more channels must be selected where the AWG waveforms of a specific type can be played.



Channel: Channel1

Output Type: Std

Figure 76 Default view of the Channel and Output Type options

1 From the **Channel** drop-down options, select either one of **Channelx**, (where x = 1 to 4) or **All Channels**, to play waveforms on the selected channel or all channels simultaneously.

Related Python APIs

`<bool> keysight_ktm5300x.channels[<name_or_index>].enabled`

To set a specific Channel to `enabled`, assign a `<bool>` value.

The M5300A modules include an embedded Function Generator (FG), Arbitrary Waveform Generator (AWG), and modulator blocks; together forming 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.

2 From the **Output Type** drop-down options, select one of the waveform output type options:

- **Std**—generates standard waveforms using the internal Function Generator.
- **Baseband**—generates arbitrary waveforms using the Arbitrary Waveform Generator at the baseband frequency.
- **Modulated**—generates modulated waveforms.

Related Python ktM5300x APIs

`<output_waveform> driver.channels[<name_or_index>].output_type`

To set `output_type`, assign a member from the enumeration `OutputWaveform`:

```
STD
BASEBAND
MODULATED
```

and

`keysight_ktm5300x.awg_multiple.output_type(channel_mask, value)`

assign an `<int>` value to `channel_mask` and replace `value` with a member from the enumeration `OutputWaveform`:

`STD`

`BASEBAND`

`MODULATED`

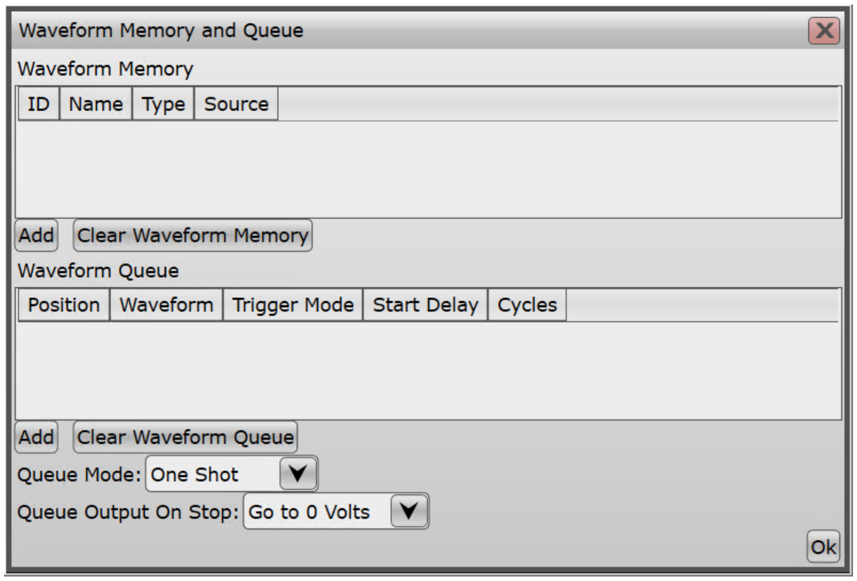
Related Python ktM5300xHvi APIs

```
ktm5300xhvi.instruction_set.set_output_type.apply
ktm5300xhvi.instruction_set.set_output_type.channel
<int> ktm5300xhvi.instruction_set.set_output_type.id
<str> ktm5300xhvi.instruction_set.set_output_type.name
ktm5300xhvi.instruction_set.set_output_type.value
```

Configuring Waveform Memory and Queue

1 On the SFP window, click the button corresponding to "**Waveform Memory and Queue**".

The **Waveform Memory ad Queue** dialog box appears. This dialog box allows you to manage loading waveforms into the AWG Memory and queuing of multiple waveforms, which you wish to play in a specific sequence.



Queue Mode: One Shot

Queue Output On Stop: Go to 0 Volts

Figure 77 Default view of the Waveform Memory And Queue dialog box

To load one or more waveforms in the AWG memory,

2 Click **Add** corresponding to the **Waveform Memory** area.

The **Add Waveform** dialog box is displayed.

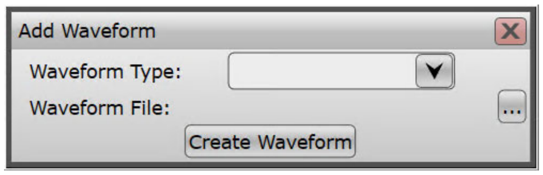


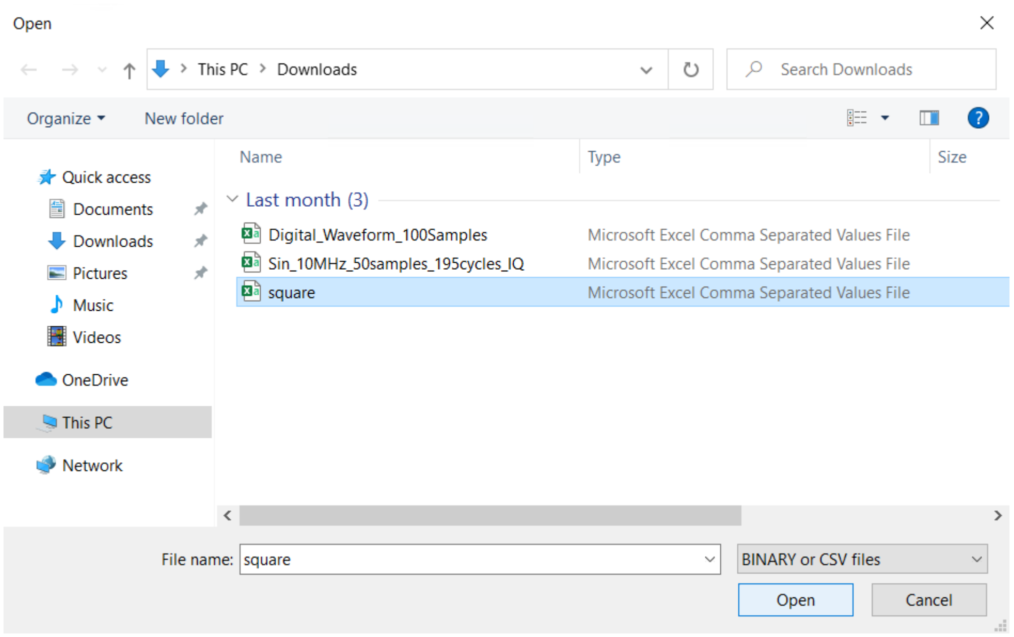
Figure 78 Default view of the Add Waveform dialog box

a From the drop-down options for **Waveform Type**, which are 'Analog', 'Digital' and 'IQ'; select a type of waveform you wish to load into the waveform memory.

b A waveform file in CSV, WFM or binary format can be loaded into the AWG memory. Click the button corresponding to **Waveform File**.

NOTE The Signal Studio waveform file (WFM format) only supports the IQ waveform type. The waveform file is generated using the following software licenses - N7631APPC and N7631RAWC.

c From the **Open** window that is displayed, navigate to the desired folder, select the desired file and click Open.


Open

This PC > Downloads

Search Downloads

Organize ▾ New folder

Name	Type	Size
Last month (3)		
Digital_Waveform_100Samples	Microsoft Excel Comma Separated Values File	
Sin_10MHz_50samples_195cycles_IQ	Microsoft Excel Comma Separated Values File	
square	Microsoft Excel Comma Separated Values File	

Quick access, Documents, Downloads, Pictures, Music, Videos, OneDrive, This PC, Network

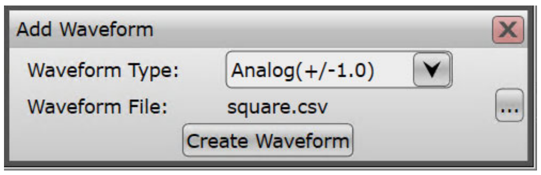
File name: square

BINARY or CSV files

Open Cancel

Figure 79 Selecting a file to load into waveform memory

The selected waveform type and waveform file name are displayed.


Add Waveform

Waveform Type: Analog(+/-1.0)

Waveform File: square.csv

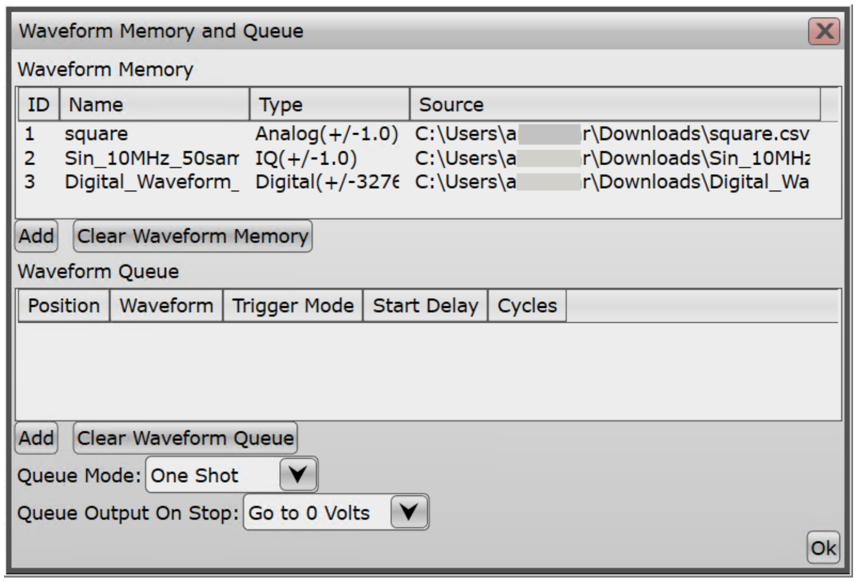
Create Waveform

Figure 80 Modified Add Waveform dialog box

d Click **Create Waveform** to load the file contents into the AWG memory.

This causes the **Add Waveform** dialog box to close and the waveform entry is displayed on the **Waveform Memory and Queue** dialog box.

e To load more files into the waveform memory, repeat step 2 and the corresponding sub-steps. The entries are displayed in the manner shown in Figure 81.



Waveform Memory and Queue

Waveform Memory

ID	Name	Type	Source
1	square	Analog(+/-1.0)	C:\Users\a[redacted]r\Downloads\square.csv
2	Sin_10MHz_50sam	IQ(+/-1.0)	C:\Users\a[redacted]r\Downloads\Sin_10MHz
3	Digital_Waveform_	Digital(+/-3276	C:\Users\a[redacted]r\Downloads\Digital_Wa

Add | Clear Waveform Memory

Waveform Queue

Position	Waveform	Trigger Mode	Start Delay	Cycles

Add | Clear Waveform Queue

Queue Mode: One Shot

Queue Output On Stop: Go to 0 Volts

Ok

Figure 81 Modified Waveform Memory and Queue dialog box

Notice the various columns in the **Waveform Memory** area shown in Figure 81,

- **ID** displays the numerical sequence starting at *1*, which is auto-assigned by the M5300x SFP, of loading waveforms into the memory.
- **Name** displays the loaded waveform file name only.
- **Type** displays the waveform type that was selected for the corresponding waveform file in the **Add Waveform** dialog box.
- **Source** displays the file path on the machine where the waveform files are located.

Related Python APIs

`keysight_ktm5300x.channels[<name_or_index>].arbitrary_waveform.create_from_file(file_path, waveform_id, type)`

assign an `<str>` value to `file_path`, an `<int>` value to `waveform_id` and `type` with one of the members of the enumeration `WaveformFileType`:

`ANALOG`

`IQ`

`DIGITAL`

3 If you wish to remove all entries from the **Waveform Memory** area, click **Clear Waveform Memory**.

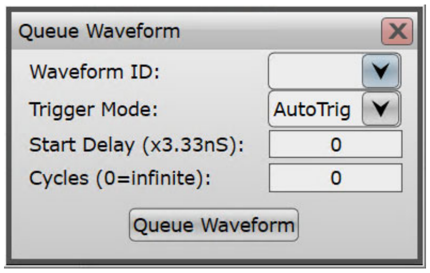
Related Python APIs

`keysight_ktm5300x.channels[<name_or_index>].arbitrary_waveform.clear_waveform_memory()`

4 To play the waveforms that have been loaded into the waveform memory, you must queue them. Note that the sequence of queuing can be different from that by which they were loaded into the memory. To queue the waveform files in the **Waveform Queue** area,

a Click **Add** corresponding to the **Waveform Queue** area.

The **Queue Waveform** dialog box is displayed.


Queue Waveform

Waveform ID:

Trigger Mode: AutoTrig

Start Delay (x3.33nS): 0

Cycles (0=infinite): 0

Figure 82 Default view of the Queue Waveform dialog box

b **Waveform ID**—In the drop down text box, select an ID number from the list that was created in the **Waveform Memory** area.

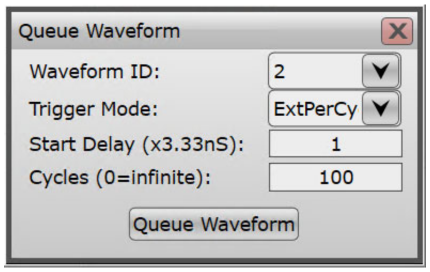
c **Trigger Mode**—Select one of the trigger modes that meets the corresponding condition:

- **AutoTrig**—The waveform is triggered immediately after **Waveform Play** is set to **Start**, or when the previous waveform in the queue finishes.
- **SWHviTrig**—Software / HVI trigger. The AWG is triggered using the **Waveform SW/HVI Trigger Control** in the SFP provided that the AWG is running.
- **ExtTrig**—Hardware trigger. The AWG waits for an external trigger source configured for the AWGs.
- **SWHviPerCycle**—Software / HVI trigger. Identical to the **SWHviTrig** option, but a trigger is required for each waveform cycle.
- **ExtPerCycleTrig**—Hardware trigger. Identical to the **ExtTrig** option, but a trigger is required for each waveform cycle.

d **Start Delay**—Defines the delay between the trigger and the waveform launch. Enter the value in the corresponding text box, which is multiplied by a factor of 3.33 nS, and the final value derived in units of nanoseconds. Also, note that this delay would be applied only in the first cycle of the queued WF.

e **Cycles**—Defines the number of times the waveform is played once launched. (Zero specifies infinite cycles).

f Click **Queue Waveform** after making the required modifications.


Queue Waveform

Waveform ID: 2

Trigger Mode: ExtPerCy

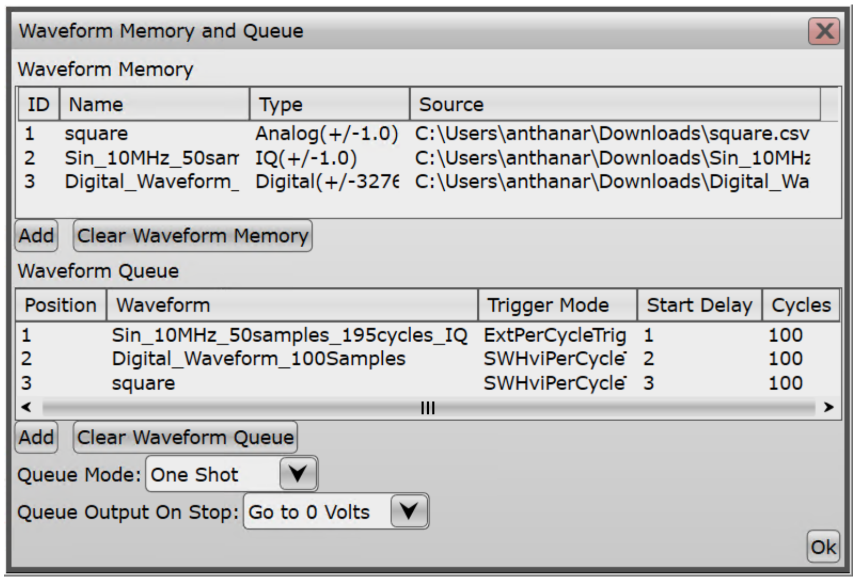
Start Delay (x3.33nS): 1

Cycles (0=infinite): 100

Figure 83 Queue Waveform dialog box after modifications

The display returns to the **Waveform Memory and Queue** dialog box and the selected Waveform ID entry can be seen.

5 Repeat the previous step to queue more waveforms. After adding two or more waveform IDs, the **Waveform Memory and Queue** dialog box should appear similar to that shown in Figure 84.


Waveform Memory and Queue

Waveform Memory

ID	Name	Type	Source
1	square	Analog(+/-1.0)	C:\Users\anthanar\Downloads\square.csv
2	Sin_10MHz_50sam	IQ(+/-1.0)	C:\Users\anthanar\Downloads\Sin_10MHz
3	Digital_Waveform_	Digital(+/-3276	C:\Users\anthanar\Downloads\Digital_Wa

Waveform Queue

Position	Waveform	Trigger Mode	Start Delay	Cycles
1	Sin_10MHz_50samples_195cycles_IQ	ExtPerCycleTrig	1	100
2	Digital_Waveform_100Samples	SWHviPerCycle	2	100
3	square	SWHviPerCycle	3	100

Queue Mode: One Shot

Queue Output On Stop: Go to 0 Volts

Figure 84 Waveform Memory and Queue dialog box after modifications

Notice in the previous image that the numbering in the **Position** tab of the **Waveform Queue** area is automatically assigned in the order you queue waveforms. The rest of the tabs display information corresponding to the fields you modified in the **Queue Waveform** dialog box for each waveform, as shown in Figure 83.

Related Python ktM5300x APIs

`keysight_ktm5300x.channels[<name_or_index>].awg_queue_waveform(trigger_mode, start_delay, cycles, waveform_id)`

assign `trigger_mode` as one of the members of the enumeration `TriggerMode`:

`AUTO_TRIG`

`SW_HVI_TRIG`

`EXT_TRIG`

`SW_HVI_PER_CYCLE_TRIG`

`EXT_PER_CYCLE_TRIG`

assign respective `<int>` values to `start_delay`, `cycles` and `waveform_id`.

Related Python ktM5300xHvi APIs

```
ktm5300xhvi.instruction_set.awg_queue_waveform.channel
ktm5300xhvi.instruction_set.awg_queue_waveform.cycles
<int> ktm5300xhvi.instruction_set.awg_queue_waveform.id
<str> ktm5300xhvi.instruction_set.awg_queue_waveform.name
ktm5300xhvi.instruction_set.awg_queue_waveform.phase_id
ktm5300xhvi.instruction_set.awg_queue_waveform.prescaler
ktm5300xhvi.instruction_set.awg_queue_waveform.start_delay
ktm5300xhvi.instruction_set.awg_queue_waveform.trigger_mode
ktm5300xhvi.instruction_set.awg_queue_waveform.waveform_id
```

6 If you wish to remove all entries from the **Waveform Queue**, click **Clear Waveform Queue**.

Related Python ktM5300x APIs

`keysight_ktm5300x.awg_multiple.queue_flush(channel_mask)`

assign an `<int>` value to `channel_mask`.

`keysight_ktm5300x.channels[<name_or_index>].awg_queue_flush()`

Related Python ktM5300xHvi APIs

`<int> ktm5300xhvi.actions.awg1_queue_flush`

```
<int> ktm5300xhvi.actions.awg2_queue_flush
<int> ktm5300xhvi.actions.awg3_queue_flush
<int> ktm5300xhvi.actions.awg4_queue_flush
```

7 **Queue Mode**—Configures the repetition cycles for the complete queue. From the drop-down options, select:

- **One Shot**—(default) Set this option for the complete queue to be reproduced one time only.
- **Cyclic**—Set this option for the complete queue to be reproduced for more than one cycles.

Related Python APIs

```
driver.channels[<name_or_index>].arbitrary_waveform.playback_mode =
keysight_ktm5300x.PlaybackMode.CYCLIC
```

or

```
driver.channels[<name_or_index>].arbitrary_waveform.playback_mode =
keysight_ktm5300x.PlaybackMode.ONE_SHOT
```

8 **Queue Output On Stop**—Configures the value that the waveform queue must be set to when **Waveform Play** is set to **Stop**, or when the waveform cycles in the queue come to an end.

- **Go to 0 Volts**—Waveform queue stops at the 0 V marker.
- **Last Value**—Waveform queue stops at the last value attained.

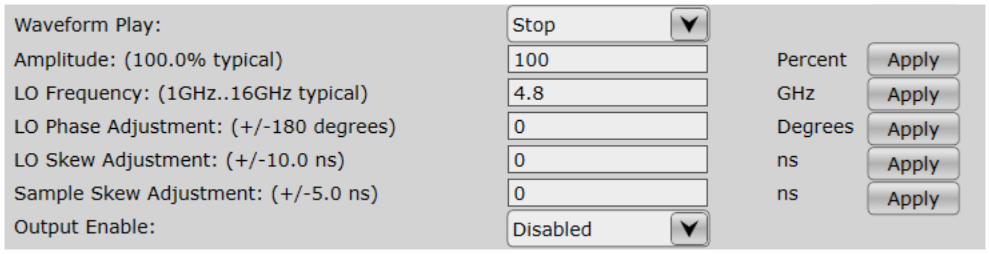
Related Python APIs

```
driver.channels[<name_or_index>].arbitrary_waveform.queue_output_on_stop =
keysight_ktm5300x.QueueOutputOnStop.LAST_VALUE
```

or

```
driver.channels[<name_or_index>].arbitrary_waveform.queue_output_on_stop =
keysight_ktm5300x.QueueOutputOnStop.GO_TO0_V
```

Configuring Waveform Play



Waveform Play:	Stop		
Amplitude: (100.0% typical)	100	Percent	Apply
LO Frequency: (1GHz..16GHz typical)	4.8	GHz	Apply
LO Phase Adjustment: (+/-180 degrees)	0	Degrees	Apply
LO Skew Adjustment: (+/-10.0 ns)	0	ns	Apply
Sample Skew Adjustment: (+/-5.0 ns)	0	ns	Apply
Output Enable:	Disabled		

Figure 85 Default view of the Waveform Play, LO and Output Enable options

From the **Waveform Play** drop-down options, select:

- **Stop**—(default) stops the AWG from playing any waveform on one or more selected Channels.
- **Start**—plays the selected waveform type on one or more selected Channels.

Related Python ktM5300x APIs

```
keysight_ktm5300x.channels[<name_or_index>].awg_start()
keysight_ktm5300x.channels[<name_or_index>].awg_stop()
keysight_ktm5300x.awg_multiple.start(channel_mask, apply_phase)
```

assign an `<int>` value to `channel_mask` and a `<bool>` value to `apply_phase`.

```
keysight_ktm5300x.awg_multiple.stop(channel_mask)
```

assign an `<int>` value to `channel_mask`.

Related Python ktM5300xHvi APIs

```
<int> ktm5300xhvi.actions.awg1_start
<int> ktm5300xhvi.actions.awg2_start
<int> ktm5300xhvi.actions.awg3_start
```

```
<int> ktm5300xhvi.actions.awg4_start
<int> ktm5300xhvi.actions.awg1_stop
<int> ktm5300xhvi.actions.awg2_stop
<int> ktm5300xhvi.actions.awg3_stop
<int> ktm5300xhvi.actions.awg4_stop
```

Configuring the Amplitude & Local Oscillator (LO) parameters

- **Amplitude**—Set a percentage, typically 100%, of the amplitude for the waveform being played.

Related Python ktM5300x APIs

`keysight_ktm5300x.awg_multiple.amplitude(channel_mask, value)`

assign an `<int>` value to `channel_mask` and a `<float>` value to `value`.

`<float> keysight_ktm5300x.channels[<name_or_index>].amplitude`

To set `amplitude`, assign a `<float>` value.

Related Python ktM5300xHvi APIs

```
ktm5300xhvi.instruction_set.set_amplitude
ktm5300xhvi.instruction_set.set_amplitude.apply
ktm5300xhvi.instruction_set.set_amplitude.channel
<int> ktm5300xhvi.instruction_set.set_amplitude.id
<str> ktm5300xhvi.instruction_set.set_amplitude.name
ktm5300xhvi.instruction_set.set_amplitude.value
```

The following LO settings can be configured in the M5300x SFP:

1 **LO Frequency** – Set the frequency on the LO, in units of GigaHertz. Typical range is 1 GHz to 16 GHz.

- Click **Apply** for the changes, if any, to take effect.

Related Python APIs

`<float> keysight_ktm5300x.channels[<name_or_index>].upconverter.local_oscillator_frequency`

To set `local_oscillator_frequency`, assign a `<float>` value.

NOTE

If you set the value of LO Frequency in multiples of 600 MHz (which is the fundamental frequency of M5300A modules), you do not have to reset the LO Phase. For all other LO Frequency values, resetting phase using the Waveform SW HVI Trigger Control (as described above) is imperative.

NOTE

You may procure the M5300A module with the HW option '012', which allows you to configure the LO frequency in the range 1, ..., 12 GHz only. To avail the entire frequency range of up to 18 GHz, do not procure this HW option. The 'About' window in the M5300x SW displays the HW option you have procured.

2 **LO Phase Adjustment** – Set the phase adjustment on the LO, in units of Degrees. Range is -180 degrees to +180 degrees.

- Click **Apply** for the changes, if any, to take effect.

Related Python ktM5300x APIs

`keysight_ktm5300x.awg_multiple.phase(channel_mask, value)`

assign an `<int>` value to `channel_mask` and a `<float>` value to `value`.

`<float> keysight_ktm5300x.channels[<name_or_index>].phase`

To set `phase`, assign a `<float>` value.

`driver.channels[<name_or_index>].phase_mode = keysight_ktm5300x.PhaseMode.RELATIVE`

or

`driver.channels[<name_or_index>].phase_mode = keysight_ktm5300x.PhaseMode.ABSOLUTE`

Related Python ktM5300xHvi APIs

`ktm5300xhvi.instruction_set.awg_queue_waveform.phase_id`
`ktm5300xhvi.instruction_set.set_phase.apply`
`ktm5300xhvi.instruction_set.set_phase.channel`
`<int> ktm5300xhvi.instruction_set.set_phase.id`
`<str> ktm5300xhvi.instruction_set.set_phase.name`
`ktm5300xhvi.instruction_set.set_phase.value`
`ktm5300xhvi.instruction_set.set_phase_mode.channel`
`<int> ktm5300xhvi.instruction_set.set_phase_mode.id`
`ktm5300xhvi.instruction_set.set_phase_mode.mode`
`<str> ktm5300xhvi.instruction_set.set_phase_mode.name`

3 **LO Skew Adjustment** – Set the skew adjustment on the LO, in units of nanoseconds. Range is -10 ns to +10 ns.

- Click **Apply** for the changes, if any, to take effect.

Related Python APIs

`<float> keysight_ktm5300x.channels[<name_or_index>].lo_skew`

To set `lo_skew`, assign a `<float>` value.

4 **Sample Skew Adjustment** – Set the skew adjustment on the waveform output for the “Baseband” and “Modulated” types by a resolution of 0.416 ns from -5 ns to +5 ns. This is a coarse skew adjustment of the waveform output which is equivalent to ±12 samples. For the “Modulated” output type, you can also configure the “LO Skew Adjustment” for fine skew adjustment. These skew adjustments help you to precisely control the timing of the signal arrival at a target point.

- Click **Apply** for the changes, if any, to take effect.

Related Python APIs

`<float> keysight_ktm5300x.channels[<name_or_index>].skew`

To set `skew`, assign a `<float>` value.

Configuring Output Enable

From the drop-down options, select:

- **Disabled**–(default option) indicates that the Channel (selected previously) is disabled for playing waveform output.
- **Enabled**–indicates that the Channel (selected previously) is enabled for playing waveform output.

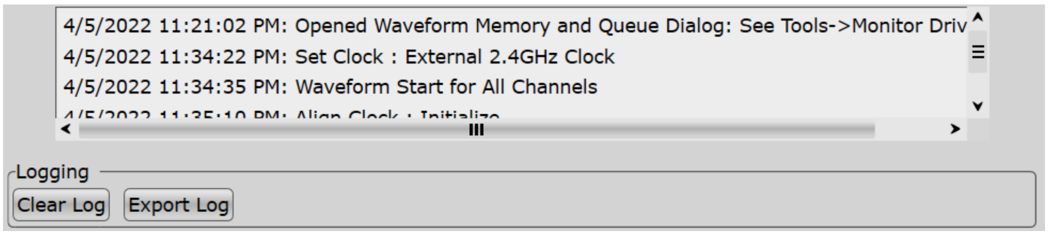
Related Python APIs

`<bool> keysight_ktm5300x.channels[<name_or_index>].enabled`

To set a specific Channel to `enabled`, assign a `<bool>` value.

Configuring Logs

The Keysight M5300A PXIe Module SFP displays a log entry for each feature where a selection is made or the associated data is modified.



```
4/5/2022 11:21:02 PM: Opened Waveform Memory and Queue Dialog: See Tools->Monitor Driv
4/5/2022 11:34:22 PM: Set Clock : External 2.4GHz Clock
4/5/2022 11:34:35 PM: Waveform Start for All Channels
```

Figure 86 Viewing and modifying logs in the SFP

Clear Log — Click this button to remove any logs that are displayed in the text area.

Export Log — Click this button to save the logs in a TXT file format.

Configuring the ‘SMB IO’ bank

The M5300A module comprises of four connectors for single-ended IO signals, which are denoted as *TRIG 1* through *8*.

The SMB IO bank on the SFP lets you control the input-output single-ended signal state on one or more connectors. Each connector TRIG*n* is represented by S*n* on the SFP, where n = 1 to 8. By default, the connectors are set to drive an input (I) signal in a low (0) state. You can manually switch to output (O) and toggle the signal state (low-to-high or vice-versa) for either the input or the output line. Figure 87 shows the default setting on the SMB IO Trigger connection.

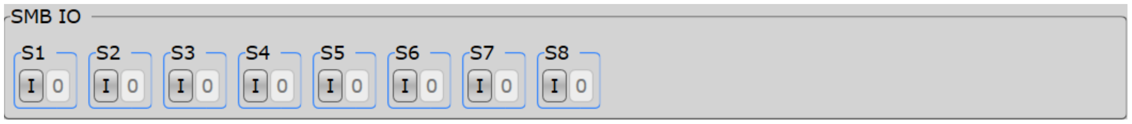


Figure 87 Default setting of the SMB IO Trigger connection

To toggle the state for a specific pin / connector / line,

1 Click the button displaying ‘I’ (Input) so that it displays ‘O’ (Output).

The button for the low (0) and high (1) states are enabled.

2 Click the button to toggle between ‘0’ and ‘1’, as required.

Note that the band around the configured SMB line changes from blue to green.

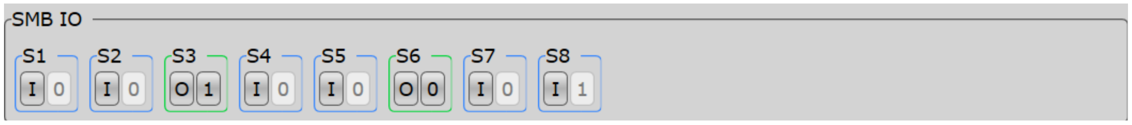


Figure 88 SMB IO Trigger settings after configuration of one or more lines

3 Repeat steps 1 and 2 for another pin / connector / line.

Related Python ktM5300x APIs

`<int> keysight_ktm5300x.smb_channels.count`

`<channel_direction> keysight_ktm5300x.smb_channels[<name_or_index>].direction`

To set `direction`, assign one of the members `IN` or `OUT` from the enumeration `ChannelDirection`

`<float> keysight_ktm5300x.smb_channels[<name_or_index>].input_skew`

To set `input_skew`, assign a `<float>` value.

`<str> keysight_ktm5300x.smb_channels[<name_or_index>].name`

`<float> keysight_ktm5300x.smb_channels[<name_or_index>].output_skew`

To set `output_skew`, assign a `<float>` value.

`<channel_state> keysight_ktm5300x.smb_channels[<name_or_index>]state`

To set **`state`**, assign one of the members **`LOGIC_LOW`** or **`LOGIC_HIGH`** from the enumeration **`ChannelState`**

Related Python ktM5300xHvi APIs

`<int> ktm5300xhvi.triggers.smb1`

`<int> ktm5300xhvi.triggers.smb2`

`<int> ktm5300xhvi.triggers.smb3`

`<int> ktm5300xhvi.triggers.smb4`

`<int> ktm5300xhvi.triggers.smb5`

`<int> ktm5300xhvi.triggers.smb6`

`<int> ktm5300xhvi.triggers.smb7`

`<int> ktm5300xhvi.triggers.smb8`

Configuring the ‘PXI IO’ bank

By default, the state of the input signal is set to ‘0’ (low). You can toggle states for output signal only. However, you may set the state to ‘1’ (high) and switch the signal from output to input.

The following image shows the default setting on the PXI IO Trigger connection.

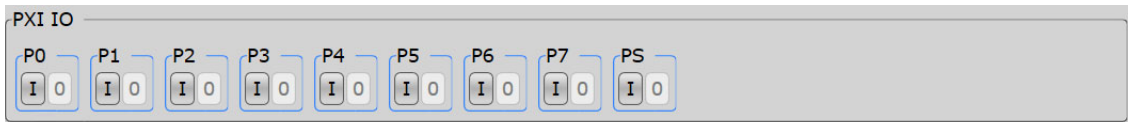


Figure 89 Default setting of the PXI IO Trigger connection

The PXI backplane of the M5300A module provides eight bus-trigger lines (PXI_TRIG[7:0]) and one point-point trigger line (PXI_STAR). These lines allow communication between cards (for example, one card communicates to another card to perform an action, when there is a rising edge on a trigger line).

NOTE The PXI_STAR line is currently not supported in the M9032A / M9033A System Sync modules. This feature may be enabled in a future release of the M9032x drivers.

The PXI IO bank on the SFP lets you control the states on these PXI trigger lines. Each line (PXI_TRIG[7:0]) is represented by Pn on the SFP, where n = 0 to 7. By default, the lines are set to drive an input (I) signal in a low (0) state. You can manually switch to output (O) and toggle the signal state (low-to-high or vice-versa).

To toggle the state for a specific pin / connector / line, follow the steps described in “Configuring the ‘SMB IO’ bank” on page 129.

Related Python ktM5300x APIs

`<int> keysight_ktm5300x.pxi_channels.count`

`<channel_direction> keysight_ktm5300x.pxi_channels[<name_or_index>].direction`

To set **`direction`**, assign one of the members **`IN`** or **`OUT`** from the enumeration **`ChannelDirection`**

`<str> keysight_ktm5300x.pxi_channels[<name_or_index>].name`

`<channel_state> keysight_ktm5300x.pxi_channels[<name_or_index>].state`

To set **`state`**, assign one of the members **`LOGIC_LOW`** or **`LOGIC_HIGH`** from the enumeration **`ChannelState`**

Related Python ktM5300xHvi APIs

```
<int> ktm5300xhvi.triggers.pxi0
<int> ktm5300xhvi.triggers.pxi1
<int> ktm5300xhvi.triggers.pxi2
<int> ktm5300xhvi.triggers.pxi3
<int> ktm5300xhvi.triggers.pxi4
<int> ktm5300xhvi.triggers.pxi5
<int> ktm5300xhvi.triggers.pxi6
<int> ktm5300xhvi.triggers.pxi7
```

Update / Auto-Update

Once you have finished defining the required IO configuration, click **Update** for the changes to take effect on the M5300A module.

Optionally, you may select the **Auto-Update** check box, so that the SFP automatically modifies the configuration on the module.

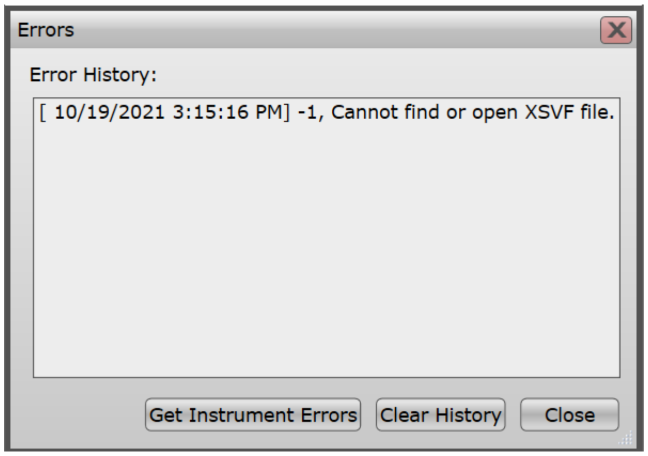
5.4.5: Resetting to Default Factory Settings

- From the **Utilities** menu, select **Reset**.

The module's default factory settings are restored. Performing this action resets the Reference Clock source and clock alignment settings to the default values as well.

5.4.6: Viewing Errors log

This section provides the details of the reported errors.



```
[ 10/19/2021 3:15:16 PM] -1, Cannot find or open XSVF file.
```

Figure 90 Viewing Errors log in M5300x SFP

To view the error log:

1. Select **Utilities** > **Errors**.
2. Click **Get Instrument Errors**.
   All errors are displayed in the **Errors** dialog window.

NOTE

If errors occur, the bottom-right corner of the SFP displays the word “Error”. Clicking the word “Error” opens the **Errors** dialog box. To learn more about instrument status, refer to Managing Module Run Logs.

To clear the error log history:

- Click **Clear History**.

5.4.7: Conducting a Self Test

This section provides the details of conducting a Self Test.

The M5300x SW performs the following self-tests internally and all these tests must meet the pass criteria for the overall Self Test to pass.

- HBM self test—This self test runs two tests. Both these sub-tests must pass for the HBM self test to pass.
  - i Checks for stuck data lines, where a data pattern is generated, peek/poke is performed and the SW reads back the data. If the SW reads the data correctly, this test passes.
  - ii Checks for stuck address lines. If the SW reads back the address correctly, this test passes.
- EEPROM self test—Data is generated and written to the start address of the last block and reads back the data. If the read data matches with that which was written, this test passes.
- FPGA loaded self test—The SW reads the boot status register (BOOTSTS) for Ultrascale. If the register value matches that from the specification for the Xilinx Ultrascale FPGA, this test passes.

NOTE

To obtain accurate results, perform a power cycle of the module prior to running the self-test using the M5300x SFP. Once the unit is up, make sure that the following SMB Channels have been paired for the digital IO self-test to "pass":

- **SMB1 to SMB5**
- **SMB2 to SMB6**
- **SMB3 to SMB7**
- **SMB4 to SMB8**

Performing Self Test using the Soft Front Panel

To initiate Self Test from the M5300x SFP,

1 Navigate to the main menu option **Utilities** and click **Self Test...**.

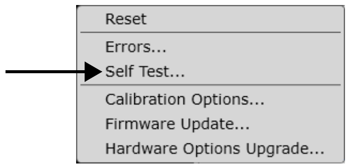
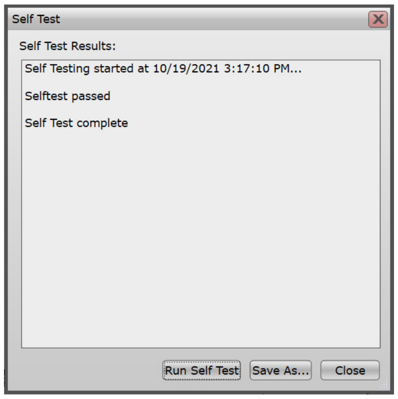


Figure 91 Selecting "Self Test" option from the Utilities menu

2 On the **Self Test** window that appears, click **Run Self Test**.

All Self Test errors are displayed in the **Self Test Results** area. If your M5300A AWG module passes its Self Test, the message "Selftest passed" is displayed.



Self Test Results:

```
Self Testing started at 10/19/2021 3:17:10 PM...
Selftest passed
Self Test complete
```

Figure 92 Running Self Test in M5300x SFP

NOTE Performing a Self Test may take several minutes.

To clear the self-test results window:

1. Close the **Self Test** window.
2. Re-open it by clicking **Utilities** > **Self Test...**.

   This clears the Self Test results log, and refreshes the **Self Test Results** area.

To save Self Test results:

1. Click **Save As...**.
2. On the **Save As** dialog that appears, save the results of your Self Test as a text (.txt) file in a folder of your choice. If multiple files are saved in the same folder, the latest file is at the bottom of the stack.

Performing Self Test using IVI Drivers

The programmatic execution of Self Test is performed in a similar manner to the SFP implementation of Self Test. In fact, the process by which the SFP reads the Self Test results is the same process that would be followed by an application program.

Following is a sample program (in Python) that uses the `driver.utility.self_test()` API function in the M5300x SW to perform Self Test:

```
import keysight_ktm5300x as m5300x


def main():
    chassis = 1
    slot = 14
    resource_name = f"PXI0::CHASSIS{chassis}::slot{slot}::INSTR"

    # Edit initialization options as needed
    id_query = False
    reset = False
    options = "Simulate=False"

    try:
        # Call driver constructor with options
        global driver
        driver = None
        driver = m5300x.KtM5300x(resource_name, id_query, reset, options)
        print("Driver Initialized")

        # Run driver self-test
        self_test_result = driver.utility.self_test()
        print(f"Self test error code : {self_test_result[0]}")
        print(f"Self test error message : {self_test_result[1]}")

        # Check instrument for errors
        print()
        while True:
            outVal = ()
            outVal = driver.utility.error_query()
            print("  error_query: code:", outVal[0], " message:", outVal[1])
            if(outVal[0] == 0):
                break

    except Exception as e:
            print("\n  Exception:", e.__class__.__name__, e.args)

    finally:
        if driver is not None: # Skip close() if constructor failed
            driver.close()
        input("\nDone - Press Enter to Exit")

if __name__ == "__main__":
    main()
```

Self Test Errors

The errors pertaining to self test messages are listed below. From the main menu of the SFP, click **Utilities** > **Errors...** or run the `driver.utility.error_query()` API to view the errors from the error queue of the software.

Table 42 List of Exceptions / Errors / Warnings pertaining to Self Tests only

Exceptions/Errors/Warnings in the Error queue	Condition for error	Possible action for eradication of error
Module M53xxA in slot nn: Digital I/O Self Test finished: Failed due to incorrect SMB Port connections.	Failure in Digital I/O self test run on a module (from the M5000-series) in a specific slot ‘nn’ due to incorrect SMB cable connections, as indicated.	Check the SMB cable connections to the module in the specified slot number.

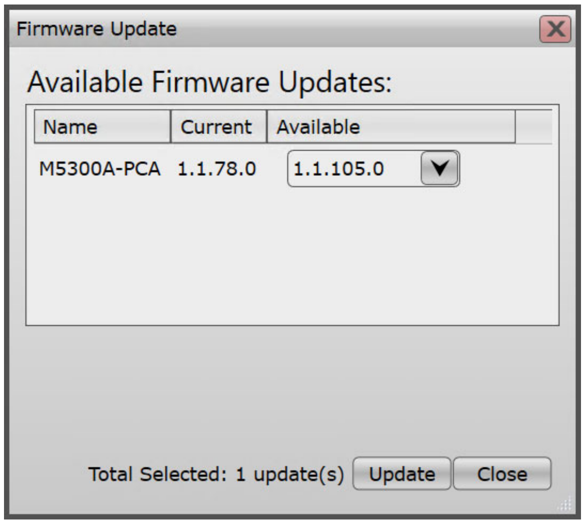
5.4.8: Setting Calibration Options

NOTE **Calibration is not supported in the current version of the M5300x SFP. Therefore, the status shall display as "Not Subject to Periodic Calibration".**

Your Keysight M5300A AWG module comes fully calibrated from the factory.

5.4.9: Updating Firmware

This section provides the details of updating the firmware.



Name	Current	Available
M5300A-PCA	1.1.78.0	1.1.105.0

Figure 93 Performing Firmware update in M5300x SFP

Use the **Firmware Update...** option in the **Utilities** menu to update the firmware on a module.

To Update the Firmware in a Module:

1 Select a version of firmware for the module using the drop-down.

2 Click **Update** to install the selected Firmware version.

NOTE **Updating the Firmware takes several minutes to complete. You can also select and install a previous version of the firmware.**

6. Configuring waveforms using the M5300x SFP

Section 6.1: Mechanism of waveform generation

Figure 94 shows a basic illustration of the waveform generation in the M5300A High Speed RF AWG modules.

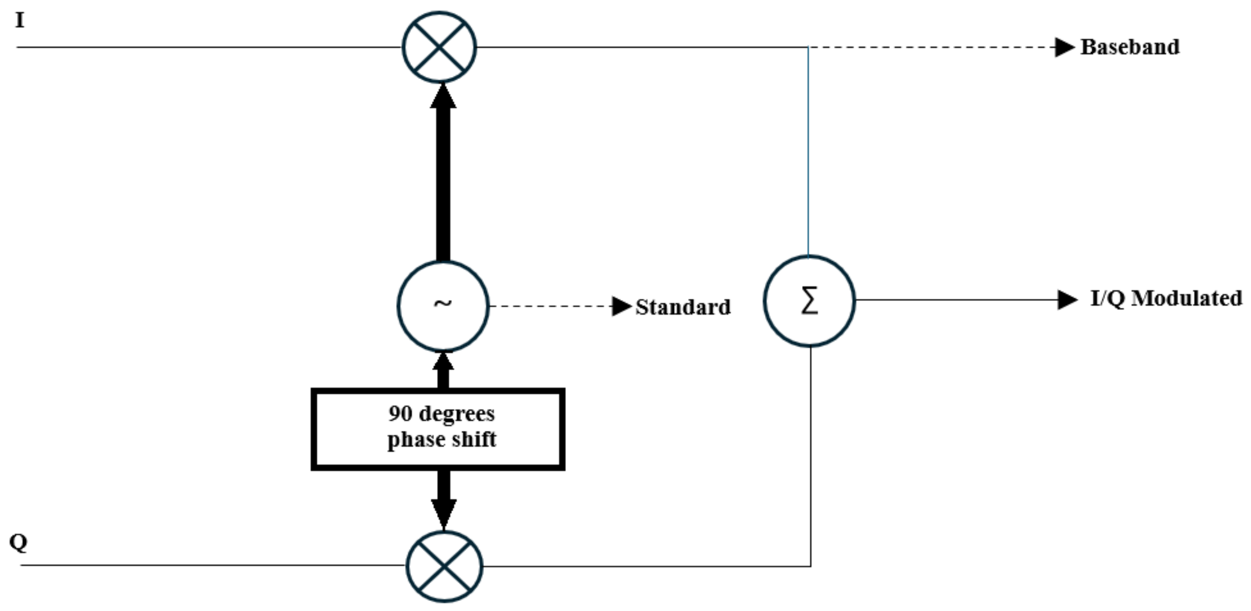


Figure 94 Block diagram of the waveform generation mechanism

Section 6.2: Understanding Waveform formats

6.2.1: Waveform formats in the M5300x SW

To understand about the different waveform formats used in the M5300x SW, see “Waveform formats in the M5300x SW” on page 31.

6.2.2: Conditions to create Waveform files in M5300x SW drivers

“Conditions to create Waveform files in M5300x SW drivers” on page 35 lists the conditions to create a waveform file.

6.2.3: Data transfer speed in M5300x SW

To know about the data transfer speeds for normal and remote APIs, see “Data transfer speed in M5300x SW” on page 36.

6.2.4: Measured values in M5300x SW

For the minimum and maximum values of parameters used in M5300x SW, see “Measured values in M5300x SW” on page 37.

Section 6.3: Configuring M5300x SFP for waveform generation

This section describes the various steps to configure the M5300x SFP for generating waveforms.

1. To configure channel and output, see “Configuring Channel and Output” on page 120.
2. To configure waveform memory and queue, see “Configuring Waveform Memory and Queue” on page 121.
3. To configure waveform play, see “Configuring Waveform Play” on page 126.
4. To configure the amplitude and Local Oscillator (LO) parameters, see “Configuring the Amplitude & Local Oscillator (LO) parameters” on page 127.
5. To configure output enable, see “Configuring Output Enable” on page 128.
6. To configure logs, see “Configuring Logs” on page 128.
7. To configure the ‘SMB IO’ bank, see “Configuring the ‘SMB IO’ bank” on page 129.
8. To configure the ‘PXI IO’ bank, see “Configuring the ‘PXI IO’ bank” on page 130.
9. After you finish defining the required IO configuration, see “Update / Auto-Update” on page 131 to update or auto-update the changes.

7. Using Peer-to-Peer Streaming for data transfers

This chapter describes the Peer-to-Peer (P2P) Streaming (of data) feature and explains its functionality in the M5200A PXIe Digitizer & M5300A PXIe RF AWG modules.

Section 7.1: P2P Streaming - Overview

Streaming means the transfer of a data stream, possibly of indefinite length, to support gap-free production or consumption of sampled data from real-time hardware.

Peer-to-peer (P2P) Streaming Transfers move a continuous (gap-free) stream of data between modules. These transfers involve the use of flow-controlled FIFOs rather than memory windows.

Peer-to-Peer (P2P) streaming in the M5000-series modules refers to the transfer of data between PXIe modules installed within specific segments of a single chassis (typically, the M9046A PXIe 18-slot chassis). This feature is currently enabled by the M5200x / M5300x software (for the respective modules) and the associated FPGA elements, which in turn, enable the streaming of data without any involvement of the host PC.

P2P Streaming is part of the PCIe Data Transfer (PDTI) technology, and involves streaming achieved through the use of FPGA-based DMAs within PXIe modules. This concept is explained in the following sections through the use of both the M5200A Digitizers (functioning as Data Transmitters) and the M5300A RF AWGs (functioning as Data Receivers), which are fixed roles. You may configure the hardware to operate in several datapath modes.

7.1.1: About PDTI

PCIe Data Transfer Interface (PDTI) components provide efficient data transfer over a PCI Express bus, between a module and either host computer or a peer module.

Currently, data transfer between peer modules only is supported over the PCI Express bus in the High-Performance Chassis.

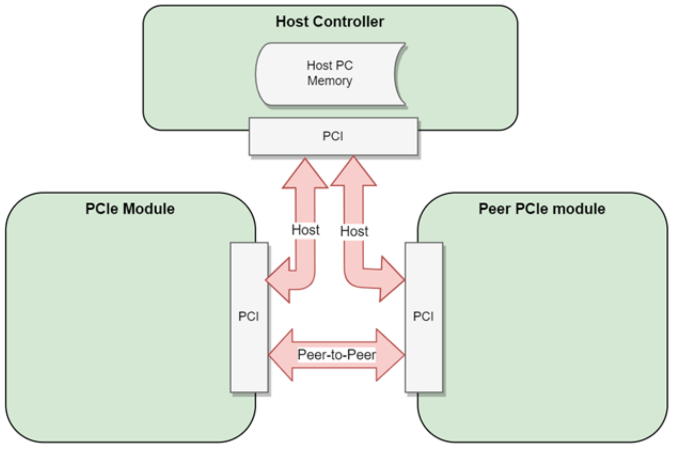


Figure 95 Components of PDTI

7.1.2: Common terminologies in P2P streaming

- Host or Peer to Peer (P2P): Host data transfers move data between the module and host memory. P2P data transfers move data between the module and another module.
- Block or Continuous: Block refers to a finite length transfer, typically to/from module memory. Continuous transfers typically move a stream of gap-free samples to/from an ADC or DAC or signal processing chain in the module.

- DMA, or Direct Memory Access, refers to dedicated GateWare IP used to move data. Once software configures and starts a DMA engine, it can run independently. Although host computers also have DMA engines, PDTI always uses the module's DMA even when moving data to/from host memory.
- I/O Requester or I/O Completer: The module with the DMA is the I/O Requester and initiates writes and read requests. The I/O Completer responds by returning or accepting data.
- Data Producer or Consumer: The direction of data flow. A data Producer can be implemented by an I/O Requester module writing data to host or peer, or by an I/O Completer module making data available to be read by the host or a peer.

Figure 96 shows a block diagram depicting the flow of data in the Peer-to-Peer Streaming model.

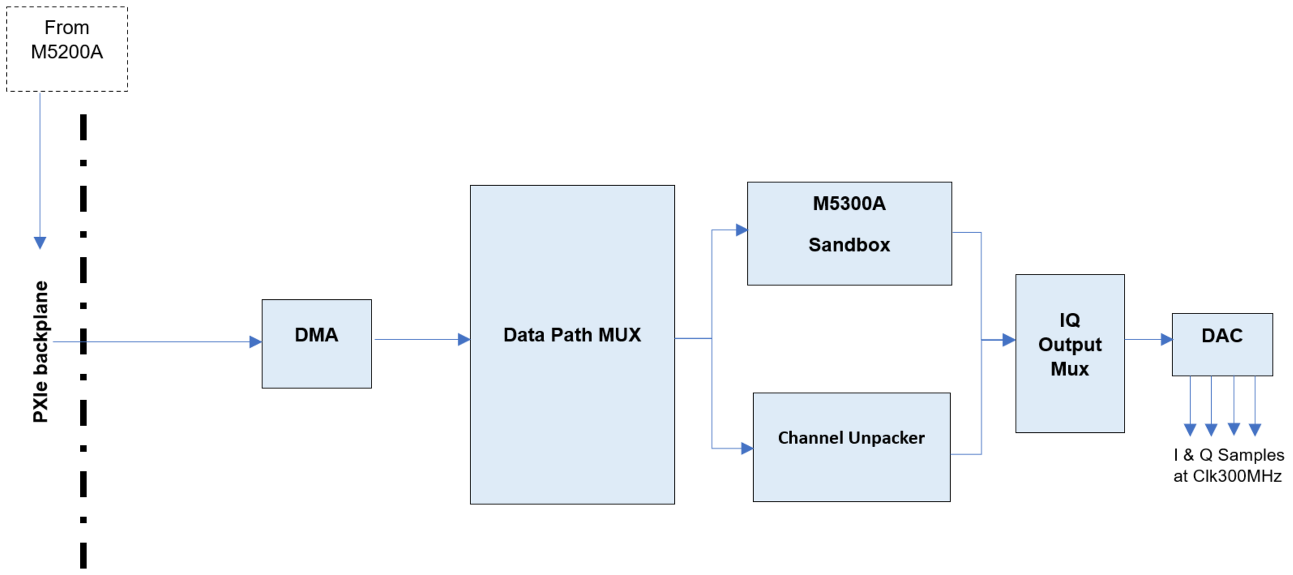


Figure 96 Block diagram showing P2P streaming from Requester Producer to Completer Consumer

For the purpose of understanding how to perform successful Peer-to-Peer transfer of data using the PXIe modules, the M5200A PXIe Digitizer is the Requester Producer and the M5300A PXIe RF AWG is the Completer Consumer in this model.

The M5200A Digitizer, as the Requester with DMA engine, streams out data to the M5300A AWG, which is the completer.

See "Performing P2P data streaming between modules" on page 150 for the steps you must perform to transfer data using this model.

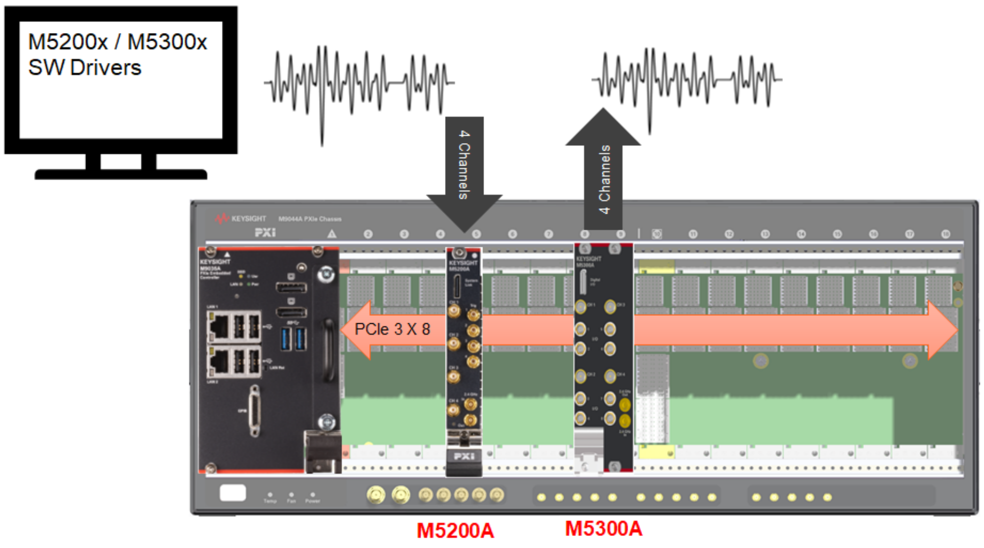


Figure 97 Typical setup for P2P streaming between M5200A and M5300A modules

7.1.3: Data flow in P2P streaming

In a typical setup (refer to Figure 97) where peer-to-peer streaming is used as the data transfer mechanism, an external RF Analog signal source feeds input data to the M5200A Digitizer (Requester Producer). This input data is digitized into 12-bit real data samples. The digitized signal is processed via DDC and Half band Decimation and packetized into 16-bit I & Q samples. Using the M5200x and M5300x SW API drivers, the P2P streaming method transfers data to the M5300A RF AWG (Completer Consumer) over the PXIe backplane.

The data, in the form of 16-bit I & Q samples (8 I and 8 Q samples) is received by the M5300A RF AWG module over the PXIe backplane and based on the Channel configuration, it is processed to the DAC either directly or via the Sandbox.

7.1.4: Data Format & Transfer in P2P streaming

As mentioned earlier, 16bit I and 16bit Q samples are transmitted through the streaming port. P2P data streaming supports data transfer in words of size 32-bit, 64-bit, 128-bit, or 256-bit (4, 8, 16, or 32 byte). Data sets that are not a multiple of the negotiated P2P transfer word size are padded up with additional bytes by the producer.

Bandwidth and Channel Supported: Three Bandwidth Options (based upon number of channels) are available with associated Sample Rates & Decimation (at M5200A Processing) & DAC configuration (at M5300A). Refer to Table 43.

Table 43 Bandwidth and Channel supported for P2P streaming of data

Software requirements

Channel number used	Bandwidth	Sample Rate	M5200A Decimation factor	M5300A DAC interpolation
1 (CH1)	±480 MHz (~1 GHz)	1.2 GSa/s for I 1.2 GSa/s for Q (2.4 GSa/s)	4 for I 4 for Q	16
2 (CH1 and CH2)	±240 MHz (~500 MHz)	600 GSa/s for I 600 GSa/s for Q (1.2 GSa/s)	8 for I 8 for Q	32
4 (CH1, CH2, CH3 & CH4)	±120 MHz (~250 MHz)	300 GSa/s for I 300 GSa/s for Q (600 MSa/s)	16 for I 16 for Q	64

Section 7.2: Prerequisites to P2P Streaming

Peer-to-Peer Streaming is a licensed feature and requires certain prerequisites you must fulfill, as described in this section.

7.2.1: Hardware prerequisites

P2P streaming of data requires the assembling of the following instruments:

1 M9046A PXIe High-Performance Chassis
2 M5200A PXIe Digitizer module
3 M5300A PXIe RF AWG module

Positioning of modules in the PXIe Chassis

To achieve successful data transmission from an M5200A PXIe Digitizer to an M5300A PXIe RF AWG module using the P2P Streaming feature, it is imperative that such Digitizer - AWG module pair (being used for P2P Streaming) be inserted into proper slots, as depicted in Figure 98.

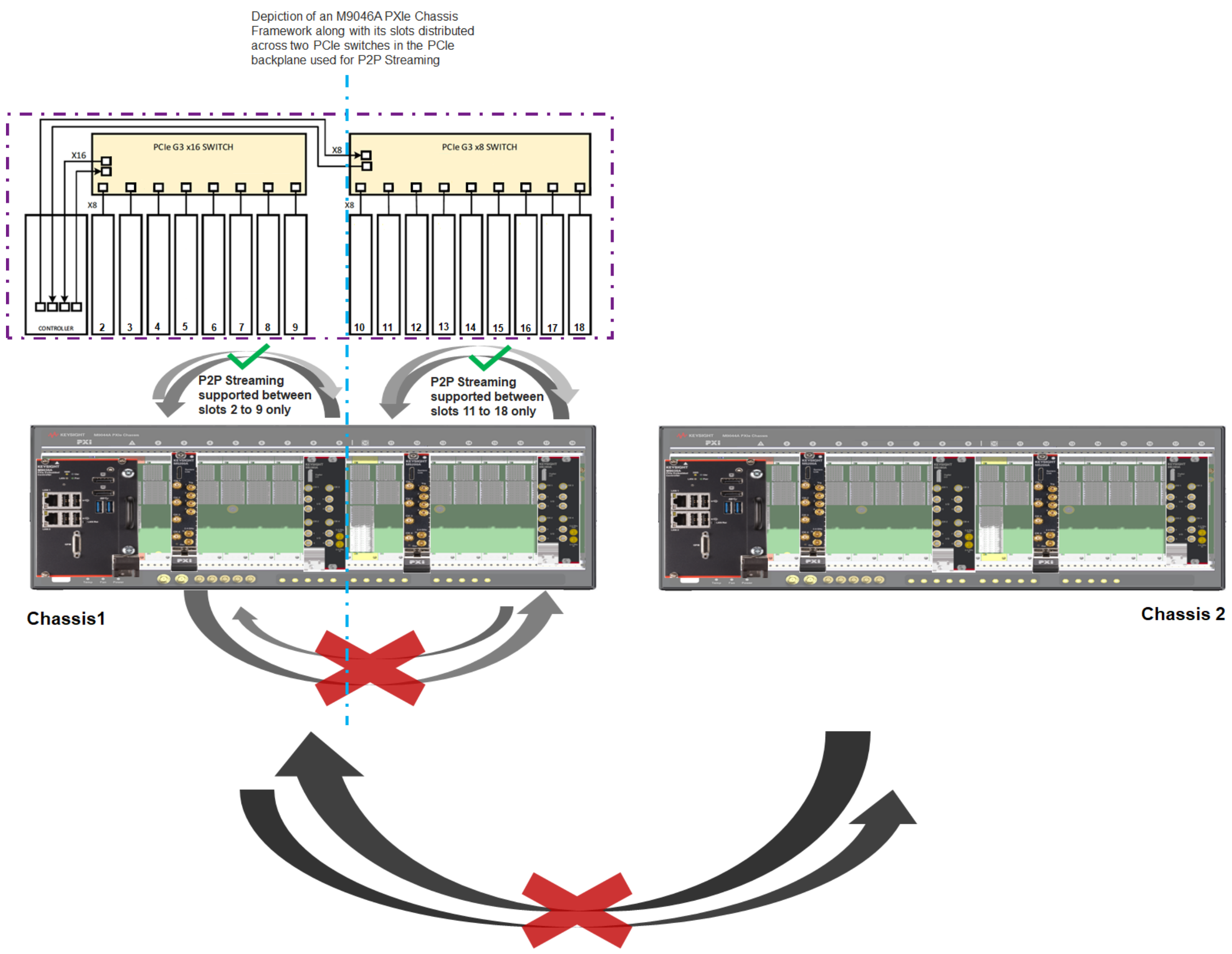

Depiction of an M9046A PXIe Chassis Framework along with its slots distributed across two PCIe switches in the PCIe backplane used for P2P Streaming
PCIe G3 x16 SWITCH
PCIe G3 x8 SWITCH
X16
X8
X8
X8
CONTROLLER
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
P2P Streaming supported between slots 2 to 9 only
P2P Streaming supported between slots 11 to 18 only
Chassis1
Chassis 2

Figure 98 Supported slot positions for M5200A & M5300A modules in an M9046A Chassis for P2P Streaming

In this image, the PXIe Chassis framework shows that slots 2 to 9 of the M9046A PXIe Chassis are connected to the PCIe x16 switch on the PCIe backplane, whereas slots 10 to 18 are connected to the PCIe x8 switch. Typically, slot 10 is reserved for M9032A/M9033A System Sync Modules, especially in a multi-chassis setup.

P2P data streaming is supported when modules are connected to the same switch [either PCIe x16 (in any of the slots numbered 2 to 9) or PCIe x8 (in any of the slots numbered 11 t o18)] on the PCIe backplane.

P2P Streaming is NOT supported when:

- One module is connected in any of the slots numbered 2 to 9 and the other is connected in a slot numbered 11 to 18 (that is, modules are connected to different switches on the PCIe backplane within a chassis)
- Modules are connected across different chassis

For more information about the M9046A High-Performance PXIe chassis, refer to the M9046A PXIe Chassis Technical Support page.

7.2.2: Software requirements

- M5200x SW Drivers version 1.4.x and above
- M5300x SW Drivers version 1.4.x and above
- PXI Chassis Family Drivers version 1.7.1032.1 and above

7.2.3: Licensing requirements

- M5000AU-001: Upgrade- Backplane streaming option for RF AWG, node-locked perpetual license
- M5000AU-002: Upgrade- Backplane streaming option for RF Digitizer, node-locked perpetual license

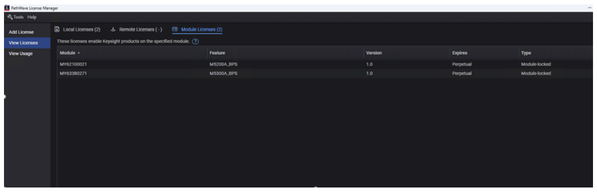
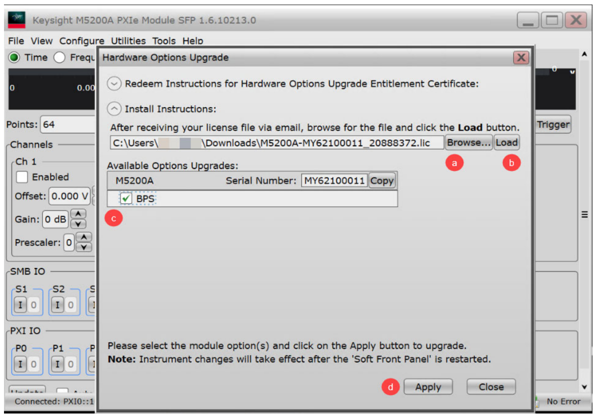
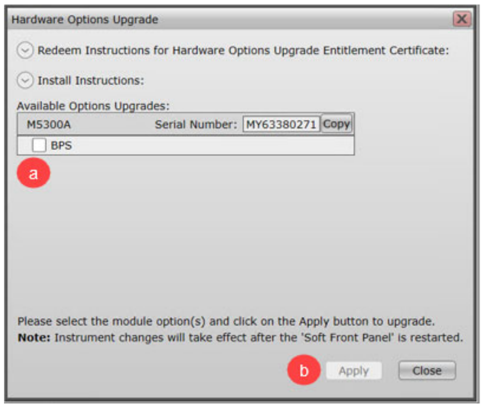
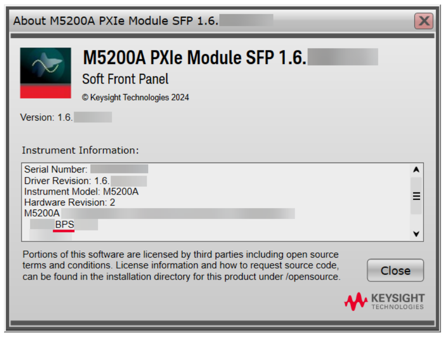
Navigate to the M5200A PXIe Digitizer Options and M5300A PXIe RF AWG Options pages, to procure and download the respective licenses.

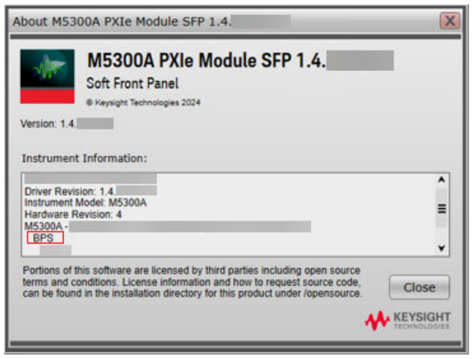
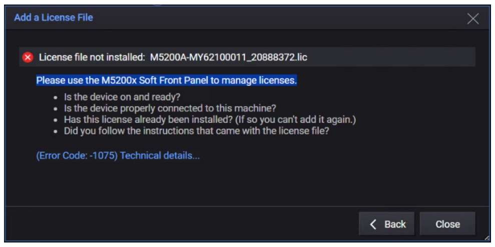
Also, see “Description of messages returned by the SW” on page 184 for the errors prompted by the M5300x SW if you try to perform the P2P streaming process without installing the required licenses.

NOTE **The M5300x ver. 1.6.x (or higher) SW drivers do not require PathWave License Manager to install licenses that enable P2P streaming, as compared to M5300x ver. 1.4.x SW drivers. Refer to the table below to follow the steps to install licenses for the required SW driver version.**

Table 44 Steps to install P2P license for various M5300x SW driver versions

#	Installing licenses for P2P (M5300x SW drivers ver. 1.4.x only)	Installing licenses for P2P (M5300x SW drivers ver. 1.6.x or higher)
1	Save the license file to the machine (computer or instrument) where you plan to use the M5200x and M5300x software drivers	Save the license file to the machine (computer or instrument) where you plan to use the M5200x and M5300x software drivers
2	Double-click the installer and follow the prompts to install the licensed software on your machine	Launch the SFP corresponding to the module where you desire to enable P2P streaming

#	Installing licenses for P2P (M5300x SW drivers ver. 1.4.x only)	Installing licenses for P2P (M5300x SW drivers ver. 1.6.x or higher)
3	From the Start menu, launch Keysight PathWave License Manager. Alternatively, launch M5300x SW SFP and from the main menu, click **Help** > **Technical Support** > **License Manager**	From the SFP main menu, click **Utilities** > **Hardware Options Upgrade**
4	In PathWave License Manager, **a** Click **Add a License File** **b** Follow the prompts to install the license file The licenses appear in the License Manager:  PathWave License Manager Tools Help Add License / View Licenses / View Usage Local Licenses (2) Remote Licenses (-) Module Licenses (2) These licenses enable Keysight products on the specified module. Module | Feature | Version | Expires | Type MY62100021 | M5200A_BPS | 1.0 | Perpetual | Module-locked MY63380271 | M5300A_BPS | 1.0 | Perpetual | Module-locked **Note**: If you have redeemed licenses for multiple host machines, the licenses may be consolidated into a single license file. In that case, install the same license file on each host. Open the attached license file and check the comments in the file to see if that is the case. If so, repeat the instructions above for each included host.	From the Install instructions drop-down, **a** **Browse** the license file **b** Click **Load** **c** Select the check box for one or more **BPS** options that appear **d** Click **Apply**  Keysight M5200A PXIe Module SFP 1.6.10213.0 Hardware Options Upgrade Redeem Instructions for Hardware Options Upgrade Entitlement Certificate: Install Instructions: After receiving your license file via email, browse for the file and click the **Load** button. C:\Users\ [redacted] \Downloads\M5200A-MY62100011_20888372.lic Available Options Upgrades: M5200A Serial Number: MY62100011 ✓ BPS Please select the module option(s) and click on the Apply button to upgrade. **Note:** Instrument changes will take effect after the 'Soft Front Panel' is restarted.
5	From the SFP main menu, click **Utilities** > **Hardware Options Upgrade** From the Install instructions drop-down, **a** select the check box for one or more **BPS** options that appear **b** click **Apply**  Hardware Options Upgrade Redeem Instructions for Hardware Options Upgrade Entitlement Certificate: Install Instructions: Available Options Upgrades: M5300A Serial Number: MY63380271 BPS Please select the module option(s) and click on the Apply button to upgrade. **Note:** Instrument changes will take effect after the 'Soft Front Panel' is restarted.	From the SFP main menu, click **Help** > **About** to verify that BPS is displayed as one of the installed license options  About M5200A PXIe Module SFP 1.6. [redacted] M5200A PXIe Module SFP 1.6. [redacted] Soft Front Panel © Keysight Technologies 2024 Version: 1.6. [redacted] Instrument Information: Serial Number: [redacted] Driver Revision: 1.6. [redacted] Instrument Model: M5200A Hardware Revision: 2 M5200A [redacted] BPS Portions of this software are licensed by third parties including open source terms and conditions. License information and how to request source code, can be found in the installation directory for this product under /opensource.

#	Installing licenses for P2P (M5300x SW drivers ver. 1.4.x only)	Installing licenses for P2P (M5300x SW drivers ver. 1.6.x or higher)
6	From the SFP main menu, click **Help** > **About** to verify that BPS is displayed as one of the installed license options  About M5300A PXIe Module SFP 1.4.[redacted] M5300A PXIe Module SFP 1.4.[redacted] Soft Front Panel © Keysight Technologies 2024 Version: 1.4.[redacted] Instrument Information: [redacted] Driver Revision: 1.4.[redacted] Instrument Model: M5300A Hardware Revision: 4 M5300A - [redacted] BPS [redacted] Portions of this software are licensed by third parties including open source terms and conditions. License information and how to request source code, can be found in the installation directory for this product under /opensource. 	If you try to install the BPS license using PathWave License Manager, the following error is displayed  License file not installed: M5200A-MY62100011_20888372.lic Please use the M5200x Soft Front Panel to manage licenses. • Is the device on and ready? • Is the device properly connected to this machine? • Has this license already been installed? (If so you can't add it again.) • Did you follow the instructions that came with the license file? (Error Code: -1075) Technical details...

Section 7.3: Performing P2P data streaming between modules

This section describes the basic steps that you must perform to transfer data using the Peer-to-Peer Streaming model, as shown in Figure 96. See "Supported APIs" on page 150 for a list of APIs that enable the peer-to-peer streaming operation. For more details, refer to the *M5300x Python API Help*.

1 On both the primary/requester (M5200A Digitizer) and secondary/completer (M5300A RF AWG) instruments, select a FIFO peer to peer port and set the 'Role' property.

2 Call the PrepareStreamingRequester() method on the primary/requester. The FlowControlAddress is returned in a parameter.

3 On the secondary, call PrepareStreamingCompleter() to prepare the secondary and specify FIFO Channel. Pass in the primary's FlowControlAddress. The secondary address Space and Offset are returned in parameters.

CAUTION PeerToPeer Streaming is a memory exhaustive process. When programming with KS2201A PathWave Test Sync Executive, any HVI API, such as load_to_hw(), that require access to module's memory must be called either before you start or after you stop the P2P streaming of data, else the operation enters into a non-responsive state because all connector calls run in a blocking mode to avoid race condition issues.

4 On the primary, call StartAsync() to start transfer, passing in the secondary address Space and Offset.

5 Start the data consuming and data producing operations for the application.

6 When the transfer is complete, call Stop() on both the primary and secondary. This is required to release the resources used for streaming.

Also, see "Description of messages returned by the SW" on page 184 in case the M5300x SW prompts specific errors during the P2P streaming process, along with the possible causes and fixes.

7.3.1: Supported APIs

- `KtM5300xPeerToPeerPort`
  - `KtM5300xPeerToPeerPort.bit_width`
  - `KtM5300xPeerToPeerPort.is_fifo`
  - `KtM5300xPeerToPeerPort.job_id`
  - `KtM5300xPeerToPeerPort.name`
  - `KtM5300xPeerToPeerPort.prepare_streaming_completer()`
  - `KtM5300xPeerToPeerPort.prepare_streaming_requester()`
  - `KtM5300xPeerToPeerPort.role`
  - `KtM5300xPeerToPeerPort.start_async()`
  - `KtM5300xPeerToPeerPort.stop()`
  - `KtM5300xPeerToPeerPort.visa_session_id`
  - `KtM5300xPeerToPeerPort.wait()`
- `keysight_ktm5300x.KtM5300xSystem.timeout`
- `KtM5300xPeerToPeerStreaming`
  - `KtM5300xPeerToPeerStreaming.set_streaming_source()`
  - `KtM5300xPeerToPeerStreaming.set_channel()`
  - `KtM5300xPeerToPeerStreaming.enable_data_transfer()`
  - `KtM5300xPeerToPeerStreaming.flush_p2p()`

See “Sample program to perform P2P streaming” on page 169 and “Sample program using HVI to perform P2P streaming” on page 175 for sample programs using the SW driver APIs and HVI APIs, respectively, that can be run to perform data transfer using the Peer-to-Peer Streaming method.

8. Using M5300x API functions in sample programs

This chapter describes how to use the Keysight M5300x Programming Libraries with Python.

Section 8.1: Implementing M5300x API functions — Sample Programs

The M5300x Programming Libraries can be located in *C:\Program Files\Keysight\M5300x*.

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

8.1.1: Sample program to perform & check toggling on SMB IO Channels

```
"""
keysight_ktM5300x Python API Example Program
Creates a driver object, toggles a SMB output channel, and checks for the toggle
on an SMB input channel.
Requires Python 3.7 or newer and keysight_ktM5300x Python extension source module.
"""


import sys
import os
import keysight_ktm5300x
import numpy as np # For keysight_ktM5300x arrays


def main():
    """
   Edit resource_name and options as needed.
   resource_name is ignored if option Simulate=true
    For this example, resource_name may be a VISA address(e.g.
"PXI0::CHASSISchassis::SLOTslot::INSTR")
    or a VISA alias.
   For more information on using VISA aliases, refer to the Keysight IO Libraries Connection
Expert documentation.
    """
    resource_name = "VISA alias"


    #  Edit the initialization options as needed
    idQuery = False
    reset   = False
    options = "Simulate=True"


    try:
        print("\n  keysight_ktM5300x Python API Example1\n")


        # Call driver constructor with options
        global driver # May be used in other functions
        driver = None
        driver = keysight_ktm5300x.KtM5300x(resource_name, idQuery, reset, options)
        print("Driver Initialized")

        #  Print a few identity properties
        print('  model:      ', driver.identity.instrument_model)
        print('  resource:   ', driver.driver_operation.io_resource_descriptor)
        print('  options:    ', driver.driver_operation.driver_setup)
        print('  simulate:   ', driver.driver_operation.simulate)

       # Toggle SMB output and validate on SMB input.
       # Assumes SMB 1 is connected to SMB 2.
        print("\nSMB channels count: ", driver.smb_channels.count)
        source  = 0
        destination = 1
        print("  Source Name: ", driver.smb_channels[source].name)
        print("  Destination Name: ", driver.smb_channels[destination].name)

        driver.smb_channels[source].direction = keysight_ktm5300x.ChannelDirection.OUT
        print("  Source Direction: ", driver.smb_channels[source].direction)

        driver.smb_channels[destination].direction = keysight_ktm5300x.ChannelDirection.IN
        print("  Destination Direction: ", driver.smb_channels[destination].direction)

        driver.smb_channels[source].state = keysight_ktm5300x.ChannelState.LOGIC_LOW
        print("  Destination State: ", driver.smb_channels[destination].state)

        driver.smb_channels[source].state = keysight_ktm5300x.ChannelState.LOGIC_HIGH
        print("  Destination State: ", driver.smb_channels[destination].state)
        print()

        # Check instrument for errors
        print()
        while True:
            outVal = ()
            outVal = driver.utility.error_query()
            print("  error_query: code:", outVal[0], " message:", outVal[1])
            if(outVal[0] == 0):
                break

    except Exception as e:
        print("\n  Exception:", e.__class__.__name__, e.args)

    finally:
        if driver is not None: # Skip close() if constructor failed
            driver.close()
        input("\nDone - Press Enter to Exit")

if __name__ == "__main__":
    main()

# © Keysight Technologies, 2021
# 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.
# ----------
```

Output

```
keysight_ktM5300x Python API Example1

Driver Initialized
  model:       M5300A
  resource:    VISA-ALIAS
  options:
  simulate:    True

SMB channels count:  8
  Source Name:  SmbIO1
  Destination Name:  SmbIO2
  Source Direction:  ChannelDirection.OUT
  Destination Direction:  ChannelDirection.IN
  Destination State:  ChannelState.LOGIC_LOW
  Destination State:  ChannelState.LOGIC_LOW
  error_query: code: 0  message: No error.

Done - Press Enter to Exit
```

8.1.2: Sample program to load a Sandbox design onto an M5300A module

```
"""
keysight_ktM5300x Python BSP Example Program
Creates a driver object, loads a sandbox design, and performs read/write operations.
Requires Python 3.7 or newer and keysight_ktM5300x Python extension source module.
"""

import sys
import os
import keysight_ktm5300x
import numpy as np # For keysight_ktM5300x arrays

def main():
    """
    Edit resource_name and options as needed.  resource_name is ignored if option Simulate=true
    For this example, resource_name may be a VISA address(e.g.
"PXI0::CHASSISchassis::SLOTslot::INSTR")
    or a VISA alias.  For more information on using VISA aliases, refer to the Keysight IO
    Libraries Connection Expert documentation.
    """
    resource_name = "VISA-alias"

    #  Edit the initialization options as needed
    idQuery = False
    reset   = False
    options = "Simulate=True"

    try:
        print("\n  keysight_ktM5300x Python BSP Example1\n")

        # Call driver constructor with options
        global driver # May be used in other functions
        driver = None
        driver = keysight_ktm5300x.KtM5300x(resource_name, idQuery, reset, options)
        print("Driver Initialized")

        # Print some FPGA properties
```

```
sandbox = driver.fpga_sandboxes['Application']
print("\nDevice UUID : ", sandbox.static_region_id) # Unique ID
# of the currently loaded static region, 128 bits, formatted as a string
# of 32 hex characters.
print("Sandbox index : ", sandbox.sandbox_index)
print("Sandbox name : ", sandbox.name)
print("Kernel id : ", sandbox.kernel_id) # Unique ID
# of the currently loaded sandbox content identifying the algorithm.
print("FPGA name : ", sandbox.fpga_name)
print("FPGA version : ", sandbox.fpga_version)

# Load FPGA sandbox with default example k7z file
k7z_file = "example.k7z"
sandbox.configure_from_k7z(k7z_file)

# Given a register name, return address and other information.
name = 'AxilRegisterBank_AxilReg_0' # modify to register name in your bsp design.
access_type = 'RW' # or 'Memory Map'
register_entry = ()
register_entry = sandbox.get_register_entry(name, access_type)
address = register_entry[0]
length =  register_entry[1]
access_type = register_entry[2]
print("Register address : ", address)
print("Register length : ", length)
print("Register access type : ", access_type)

# Poke / Peek a 32-bit register in the sandbox
print("\nPEEK/POKE test...")
address_offset = 0
data = 100
print("Poke : ", data)
sandbox.poke(address, data)
ret_val = sandbox.peek(address + address_offset)
print("Peek : ", ret_val)

# Check instrument for errors
print()
while True:
    outVal = ()
    outVal = driver.utility.error_query()
    print("  error_query: code:", outVal[0], " message:", outVal[1])
```

```
            if(outVal[0] == 0):
                break

    except Exception as e:
        print("\n  Exception:", e.__class__.__name__, e.args)

    finally:
        if driver is not None: # Skip close() if constructor failed
            driver.close()
        input("\nDone - Press Enter to Exit")

if __name__ == "__main__":
    main()

# © Keysight Technologies, 2021
# 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.
# ----------
```

Output

```
keysight_ktM5300x Python BSP Example1

Driver Initialized

Device UUID :  80090200-09856e5d-785f-5300-010083e7
Sandbox index :  0
Sandbox name :  Application
Kernel id :  00000000-0000-0000-0000-000000000000
FPGA name :  M5300A_PCA
FPGA version :  1.2.999.0
```

```
  Exception: RuntimeError ('File does not exist: example.k7z',)

Done - Press Enter to Exit
```

8.1.3: Sample program to create an HVI sequence for an SMB trigger

```
"""
keysight_ktM5300x Python API Example Program
Creates an HVI sequence that waits for smb trigger, and sets the HVI register to true
when trigger condition is met.
Requires Python 3.7 or newer and keysight_ktM5300x Python extension source module
installed using Package Installer for Python (pip install <fileName>.tar.gz).
"""

import sys
import os
import keysight_hvi
import keysight_ktm5300x as my_module
from datetime import timedelta
import time

def main():
    try:
        #Change Simulate to True to use dummy module
        #simulate = True
        # Open Keysight module with HVI capability
        resource = 'VISA-Alias1'
        resource1 = 'VISA-Alias2'
        query = False
        reset = False
        options = 'Simulate=1, LogLevel=Info'
        module = my_module.KtM5300x(resource, query, reset, options)
        module1 = my_module.KtM5300x(resource1, query, reset, options)

        # create System Definition
        system_def = keysight_hvi.SystemDefinition("MyHviSystem")

        # add clocks
        system_def.non_hvi_system_clocks = [100.0e6]

        # add the module engine to the System Definition
        engine_name = 'My Engine'
        cmi = module.get_instance()
        ktM5300xhvi = my_module.KtM5300xHvi(cmi)

        print("HVI Engine ID in slot 2:  ", ktM5300xhvi.engines.main_engine)

        system_def.engines.add(ktM5300xhvi.engines.main_engine, engine_name)
        if len(system_def.engines) == 0:
            raise Exception('System Definition has no engines')

        def_engine = system_def.engines[0]
    # Configure an smb trigger
    # that follows the pxi event (PXI event->SMB6 toggle)
        smb3_trigger_id = ktM5300xhvi.triggers.smb2
        smb3_trigger = def_engine.triggers.add(smb3_trigger_id, 'smb3')
        smb3_trigger.config.sync_mode = keysight_hvi.SyncMode.IMMEDIATE
        smb3_trigger.config.direction = keysight_hvi.Direction.INPUT

        # Make HVI sequencer
        hvi_sequencer = keysight_hvi.Sequencer("myName", system_def)

        # Create sequence
        sync_sequence = hvi_sequencer.sync_sequence # Obtain main sequence
    # from the created HVI instrument
        sequence = sync_sequence.add_sync_multi_sequence_block('SyncExec',
10).sequences[engine_name]
        hvi_done = sync_sequence.scopes[engine_name].registers.add("hvi_done",
keysight_hvi.RegisterSize.SHORT)

        wait_condition = keysight_hvi.Condition.trigger(smb3_trigger)
        wait_event = sequence.add_wait('wait external_trigger', 10, wait_condition)
        wait_event.set_mode(keysight_hvi.WaitMode.TRANSITION,keysight_hvi.SyncMode.IMMEDIATE)

        instruction = sequence.add_instruction("hvi_done = 1", 10,
sequence.instruction_set.assign.id)
        instruction.set_parameter(sequence.instruction_set.assign.destination.id, hvi_done)
        instruction.set_parameter(sequence.instruction_set.assign.source.id, 1)

        print("Compiling and running Sequence!")

        # Compile and run the HVI sequence
        hvi = hvi_sequencer.compile()       # Compile the instrument sequence(s)
        hvi.load_to_hw()    # Load the instrument sequence(s) to HW
        hvi.run(hvi.no_wait)       # Execute the instrument sequence(s)

        print("Waiting for external trigger")

        while True:
            # Wait for HVI sequence loop to finish
            time.sleep(.5)

            if hvi.sync_sequence.scopes[engine_name].registers['hvi_done'].read() == 1:
                print("Trigger received")
                break

            print("Press any key to send smb trigger")
            input()

            #send smb trigger
            module1.smb_channels[2].direction = my_module.ChannelDirection.OUT
            module1.smb_channels[2].state = my_module.ChannelState.LOGIC_HIGH
            module1.smb_channels[2].state = my_module.ChannelState.LOGIC_LOW
            module1.smb_channels[2].state = my_module.ChannelState.LOGIC_HIGH

        hvi.release_hw()
        module.close()
        print("Test complete!")

    except Exception as ex:
        print(ex)
        print('Exiting')

if __name__ == '__main__':
    main()

# © Keysight Technologies, 2021
# 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.
# ----------
```

Output

```
HVI Engine ID in slot 2:   #M5300xHviEngine@PXI0::CHASSIS1::SLOT8::INSTR
Compiling and running Sequence!
Waiting for external trigger
Press any key to send smb trigger
```

8.1.4: Sample program to configure waveform settings on the module

```
"""
keysight_ktM5300x Python API Example Program
Requires Python 3.7 or newer and keysight_ktM5300x Python extension source module
installed using Package Installer for Python (pip install <fileName>.tar.gz).
"""

import sys
import os
import keysight_ktm5300x as m5300x

def main():
    try:
        # Open Keysight module, use True to simulate
        hardware_simulated = True
        query = False
        reset = False

        if hardware_simulated:
            options = 'Simulate=1'
        else:
            options = 'Simulate=0, DriverSetup= log=1'

        # Adjust the line below to the name of your module
        module = m5300x.KtM5300x("VISA_ALIAS", query, reset, options)

        # Print a few driver identity properties
        print('Identifier:  {0}'.format(module.identity.identifier))
        print('Revision:    {0}'.format(module.identity.revision))
        print('Description: {0}'.format(module.identity.description))
        print('Model:       {0}'.format(module.identity.instrument_model))

        # Set the analog clock to be INTERNAL or EXTERNAL
        module.clock.clock_source = m5300x.ClockSource.INTERNAL

        # Set the LO Frequency in Hertz 1.0e9 to 16.0e9 typical
        module.channels['Channel1'].upconverter.local_oscillator_frequency = 5.0e9

        # Set the amplitude in percentage
        module.channels['Channel1'].amplitude = 20.0

        # Set the PhaseMode to ABSOLUTE or RELATIVE, RELATIVE is relative to last phase,
ABSOLUTE is to a fixed phase at current time
        module.channels['Channel1'].phase_mode = m5300x.PhaseMode.ABSOLUTE

        # Set the Phase from -180 to +180 degrees
        module.channels['Channel1'].phase = 90.0

        # Set the skew from -10ns to +10ns
        module.channels['Channel1'].skew = -1.0e-9

        module.close()

        print("Test complete!")

    except Exception as ex:
        print(ex)
        print('Exiting')

if __name__ == '__main__':
    main()

# © Keysight Technologies, 2021
# 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.
# ----------
```

Output

```
Identifier:  KtM5300x
Revision:    1.0.42321
Description: PythonAPI for KtM5300x
Model:       M5300A
Test complete!
```

8.1.5: Sample program to adjust phase using HVI

```
"""
keysight_ktM5300x Python API Example Program
Requires Python 3.7 or newer and keysight_ktM5300x Python extension source module
installed using Package Installer for Python (pip install <fileName>.tar.gz).
"""

import sys
import os
import keysight_hvi
import keysight_ktm5300x as m5300x

def main():
    try:
        # Open Keysight module with HVI capability, use True to simulate
        hardware_simulated = True
        query = False
        reset = False



        # create System Definition
        system_def = keysight_hvi.SystemDefinition("MyHviSystem")

        if hardware_simulated:
            options = 'Simulate=1'
            system_def.chassis.add_with_options(1,
'Simulate=True,DriverSetup=model=M9018B,NoDriver=True')
```

```
        else:
            options = 'Simulate=0, DriverSetup= log=1'

        # Adjust the line below to the name of your module
        module = m5300x.KtM5300x("VISA_ALIAS", query, reset, options)

        # Print a few driver identity properties
        print('Identifier:  {0}'.format(module.identity.identifier))
        print('Revision:    {0}'.format(module.identity.revision))
        print('Description: {0}'.format(module.identity.description))
        print('Model:       {0}'.format(module.identity.instrument_model))

                # We are doing a phase adjustment below
        # Set the LO Frequency in Hertz 1.0e9 to 16.0e9 typical
        module.channels['Channel1'].upconverter.local_oscillator_frequency = 5.0e9

        # Set the PhaseMode to ABSOLUTE or RELATIVE, RELATIVE is relative to last phase,
ABSOLUTE is to a fixed phase at current time
        module.channels['Channel1'].phase_mode = m5300x.PhaseMode.ABSOLUTE

        # Set the Phase from -180 to +180 degrees
        module.channels['Channel1'].phase = 52.0

        # Reset the phase to 0 (Absolute)
        module.channels['Channel1'].phase = 0.0

        # Now we repeat the same phase adjustment using HVI commands
        # HVI commands allow precise time alignment of the application of the commands across
multiple modules
        print("Initialize for HVI commands")
        # add clocks
        system_def.non_hvi_system_clocks = [100.0e6]

        # add the module engine to the System Definition
        engine_name = 'My Engine'

        # Start the HVI object
        cmi = module.get_instance()
        ktm5300xhvi = m5300x.KtM5300xHvi(cmi)

        # Add the HVI engine
        system_def.engines.add(ktm5300xhvi.engines.main_engine, engine_name)
        if len(system_def.engines) == 0:
            raise Exception('System Definition has no engines')

        # Add the Apply instructon
        def_engine = system_def.engines[0]
        apply_instructions_action =
def_engine.actions.add(ktm5300xhvi.actions.apply_instructions, "Apply")

        # Make HVI sequencer
        hvi_sequencer = keysight_hvi.Sequencer("mySequencer", system_def)

        # Create the sequence:  Setting the phase to 52 degrees Absolute
        sync_sequence = hvi_sequencer.sync_sequence    # Obtain main sequence from the created
HVI instrument
        sequence = sync_sequence.add_sync_multi_sequence_block('SyncExec',
10).sequences[engine_name]

        # First instruction must start at a time of 3.33ns or greater.  It cannot be 0.0ns
        instruction1 = sequence.add_instruction('Set Phase Mode', 3.33,
ktm5300xhvi.instruction_set.set_phase_mode.id)
        instruction1.set_parameter(ktm5300xhvi.instruction_set.set_phase_mode.mode.id,
m5300x.PhaseMode.ABSOLUTE)

        # Set the phase but do not apply it
        # Note: Need to allow the preceeding Set Phase Mode command 280ns to complete, before
we can send in this command.
        # This is what the 280.0 parameter is doing in this command.
        instruction2 = sequence.add_instruction('Set Phase', 280.0,
ktm5300xhvi.instruction_set.set_phase.id)
        instruction2.set_parameter(ktm5300xhvi.instruction_set.set_phase.value.id, 52.0)
        instruction2.set_parameter(ktm5300xhvi.instruction_set.set_phase.apply.id, 0)

        # Now apply the phase at a very precise time in response to a signal
        # Note: Need to allow the preceeding Set Phase command 800ns to complete, before we can
send in this command.
        # This is what the 800.0 parameter is doing in this command.
        instruction3 = sequence.add_instruction('Execute apply instructions', 800.0,
sequence.instruction_set.action_execute.id)
        instruction3.set_parameter(sequence.instruction_set.action_execute.action.id,
apply_instructions_action)

        # Compile and run the HVI sequence
        print("Compiling and running Sequence!")
        hvi = hvi_sequencer.compile()
        hvi.load_to_hw()           # Load the instrument sequence(s) to HW
        hvi.run(hvi.no_wait)       # Execute the instrument sequence(s)

        # Release the HVI hardware
        hvi.release_hw()
        module.close()

        print("Test complete!")

    except Exception as ex:
        print(ex)
        print('Exiting')

if __name__ == '__main__':
    main()

# © Keysight Technologies, 2021
# 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.
# ----------
```

Output

```
Identifier:  KtM5300x
Revision:    1.0.42321
Description: PythonAPI for KtM5300x
Model:       M5300A
Initialize for HVI commands
Compiling and running Sequence!
Test complete!
```

8.1.6: Sample program to perform P2P streaming

```
"""
keysight_ktM5300x Python API Example Program
Requires Python 3.7 or newer and keysight_ktM5300x Python extension source module
installed using Package Installer for Python (pip install <fileName>.tar.gz).
"""

import time
import keysight_ktm5300x
import keysight_ktm5200x
query = False
reset = False

options_Consumer = 'CompleterMinCredit=8192,RequesterMinCredit=8192'
options_Producer = 'CompleterMinCredit=8192,RequesterMinCredit=8192'

m5200xProd_resource = "PXI0::29-0.0::INSTR"
m5300xCons_resource = "PXI0::33-0.0::INSTR"

channel_config = int(input("Enter channel configuration."))

# Considering M5200A is Producer, M5300A is Consumer
global driver_m5200x_Prod, driver_m5300x_Cons

# Connecting with Module Drivers
print("adding Digitizer...")
driver_m5200x_Prod = keysight_ktm5200x.KtM5200x(m5200xProd_resource, query, reset,
options_Producer)
print("Digitizer added")

print("adding AWG...")
driver_m5300x_Cons = keysight_ktm5300x.KtM5300x(m5300xCons_resource, query, reset,
options_Consumer)
print("AWG added")

# Functions for static region configurations

def configure_mixer_static_region():
        if (channel_config == 1):
          mixer_freq = int(input("Enter mixer freq for Mixer 1 in MHz for 1Mhz enter 1   -> "))
```

```
driver_m5200x_Prod.peer_to_peer_streaming.set_center_frequency(keysight_ktm5200x.keysight_ktm5
200x.ChannelNumber.ONE, float(mixer_freq*1000000))
        elif (channel_config == 2):
          mixer_freq = int(input("Enter mixer freq for Mixer 1 in MHz for 1Mhz enter 1   -> "))
          mixer_freq1 = int(input("Enter mixer freq for Mixer 2 in MHz for 1Mhz enter 1   -> "))

driver_m5200x_Prod.peer_to_peer_streaming.set_center_frequency(keysight_ktm5200x.keysight_ktm5
200x.ChannelNumber.ONE, float(mixer_freq*1000000))

driver_m5200x_Prod.peer_to_peer_streaming.set_center_frequency(keysight_ktm5200x.keysight_ktm5
200x.ChannelNumber.TWO, float(mixer_freq1*1000000))
        elif(channel_config == 4):
          mixer_freq = int(input("Enter mixer freq for Mixer 1 in MHz for 1Mhz enter 1   -> "))
          mixer_freq1 = int(input("Enter mixer freq for Mixer 2 in MHz for 1Mhz enter 1   -> "))
          mixer_freq2 = int(input("Enter mixer freq for Mixer 3 in MHz for 1Mhz enter 1   -> "))
          mixer_freq3 = int(input("Enter mixer freq for Mixer 4 in MHz for 1Mhz enter 1   -> "))

driver_m5200x_Prod.peer_to_peer_streaming.set_center_frequency(keysight_ktm5200x.keysight_ktm5
200x.ChannelNumber.ONE, float(mixer_freq*1000000))

driver_m5200x_Prod.peer_to_peer_streaming.set_center_frequency(keysight_ktm5200x.keysight_ktm5
200x.ChannelNumber.TWO, float(mixer_freq1*1000000))

driver_m5200x_Prod.peer_to_peer_streaming.set_center_frequency(keysight_ktm5200x.keysight_ktm5
200x.ChannelNumber.THREE, float(mixer_freq2*1000000))

driver_m5200x_Prod.peer_to_peer_streaming.set_center_frequency(keysight_ktm5200x.keysight_ktm5
200x.ChannelNumber.FOUR, float(mixer_freq3*1000000))


def path_sel():

driver_m5200x_Prod.peer_to_peer_streaming.set_streaming_source(keysight_ktm5200x.keysight_ktm5
200x.PeerToPeerStreamingSource.PROCESSED_DATA)

driver_m5300x_Cons.peer_to_peer_streaming.set_streaming_source(keysight_ktm5300x.keysight_ktm5
300x.PeerToPeerStreamingSource.PROCESSED_DATA)



def program_channel_configuration():
            if (channel_config == 1):

driver_m5300x_Cons.peer_to_peer_streaming.set_channel(keysight_ktm5300x.keysight_ktm5300x.Peer
ToPeerStreamingChannel.ONE)

driver_m5200x_Prod.peer_to_peer_streaming.set_channel(keysight_ktm5200x.PeerToPeerStreamingCha
nnel.ONE)
            elif (channel_config == 2):
```

```
driver_m5300x_Cons.peer_to_peer_streaming.set_channel(keysight_ktm5300x.keysight_ktm5300x.Peer
ToPeerStreamingChannel.TWO)

driver_m5200x_Prod.peer_to_peer_streaming.set_channel(keysight_ktm5200x.PeerToPeerStreamingCha
nnel.TWO)
            elif(channel_config == 4):

driver_m5300x_Cons.peer_to_peer_streaming.set_channel(keysight_ktm5300x.keysight_ktm5300x.Peer
ToPeerStreamingChannel.FOUR)

driver_m5200x_Prod.peer_to_peer_streaming.set_channel(keysight_ktm5200x.PeerToPeerStreamingCha
nnel.FOUR)


def peer_to_peer_configurtion():
            driver_m5200x_Prod.system.timeout = -1
            driver_m5300x_Cons.system.timeout = -1


            # Setting p2p in continuous streaming mode
            p2p_block_transfer_length = -1


            # Preparing streaming port for a peer-to-peer continuous data transfer as the bus
requester.
            driver_m5200x_Prod.peer_to_peer_ports["Port1"].role =
keysight_ktm5200x.PeerToPeerPortRole.STREAMING_REQUESTER_PRODUCER
            [length,dataFormat,flow_conrol_addr] =
driver_m5200x_Prod.peer_to_peer_ports["Port1"].prepare_streaming_requester(0,p2p_block_transfe
r_length,keysight_ktm5200x.PeerToPeerDataFormat.I32,256); # fifo channel , Length



            # Preparing streaming port for a peer-to-peer continuous data transfer as the bus
completer.
            driver_m5300x_Cons.peer_to_peer_ports["Port2"].role =
keysight_ktm5300x.PeerToPeerPortRole.STREAMING_COMPLETER_CONSUMER
            [length,dataformat,visa_addr,visa_offset] =
driver_m5300x_Cons.peer_to_peer_ports["Port2"].prepare_streaming_completer(0,length,keysight_k
tm5300x.PeerToPeerDataFormat.I32,flow_conrol_addr);
            visa_session_id = driver_m5300x_Cons.peer_to_peer_ports["Port2"].visa_session_id



driver_m5200x_Prod.peer_to_peer_ports["Port1"].start_async(visa_session_id,visa_addr,visa_offs
et,length)


def flush_p2p_fifo():
            print("Clearing both streaming FIFOs")
            driver_m5200x_Prod.peer_to_peer_streaming.flush_p2p()
            driver_m5300x_Cons.peer_to_peer_streaming.flush_p2p()
```

```
def set_dac_output_mode():
            print("\nset to Baseband waveform output...")


            driver_m5300x_Cons.channels[0].output_type =
keysight_ktm5300x.OutputWaveform.BASEBAND
            driver_m5300x_Cons.channels[0].amplitude = 100.0
            driver_m5300x_Cons.channels[0].enabled = True



            driver_m5300x_Cons.channels[1].output_type =
keysight_ktm5300x.OutputWaveform.BASEBAND
            driver_m5300x_Cons.channels[1].amplitude = 100.0
            driver_m5300x_Cons.channels[1].enabled = True



            driver_m5300x_Cons.channels[2].output_type =
keysight_ktm5300x.OutputWaveform.BASEBAND
            driver_m5300x_Cons.channels[2].amplitude = 100.0
            driver_m5300x_Cons.channels[2].enabled = True



            driver_m5300x_Cons.channels[3].output_type =
keysight_ktm5300x.OutputWaveform.BASEBAND
            driver_m5300x_Cons.channels[3].amplitude = 100.0
            driver_m5300x_Cons.channels[3].enabled = True


def close_p2p():
            driver_m5300x_Cons.peer_to_peer_streaming.enable_data_transfer(False)
            driver_m5200x_Prod.peer_to_peer_streaming.enable_data_transfer(False)
            driver_m5200x_Prod.peer_to_peer_ports["Port1"].stop()
            driver_m5300x_Cons.peer_to_peer_ports["Port2"].stop()


def main():
    try:


        program_channel_configuration()


        set_dac_output_mode()


        driver_m5300x_Cons.peer_to_peer_streaming.enable_data_transfer(False)
        driver_m5200x_Prod.peer_to_peer_streaming.enable_data_transfer(False)
        peer_to_peer_configurtion()

        configure_mixer_static_region()

        # Clearing both streaming FIFOs
        flush_p2p_fifo()

        path_sel()

        # Waiting for produer and consumer to get configured
        time.sleep(1)



        driver_m5200x_Prod.peer_to_peer_streaming.ddc_phase_reset()

        driver_m5300x_Cons.peer_to_peer_streaming.enable_data_transfer(True)
        driver_m5200x_Prod.peer_to_peer_streaming.enable_data_transfer(True)



       # waiting in infinite while loop unless asked to quit be pressing "q" and Can change the
LO Frequency
        while 1:
           var = input("Enter q to quit or press any key to continue ->  ")
           if(var == "q" ):
             break
          freq_switch = int(input("Enter 1 to change the mixer frequency and 0 to continue with
current mixer frequency  ->  "))
           if freq_switch == 1:
               # Stop p2p transfer. Also releases hardware resources used by FIFO streaming.
               close_p2p()
               configure_mixer_static_region()
               peer_to_peer_configurtion()
               # Clearing both streaming FIFOs
               flush_p2p_fifo()

               time.sleep(1)
               driver_m5200x_Prod.peer_to_peer_streaming.ddc_phase_reset()
               driver_m5300x_Cons.peer_to_peer_streaming.enable_data_transfer(True)
               driver_m5200x_Prod.peer_to_peer_streaming.enable_data_transfer(True)
        close_p2p()
        driver_m5200x_Prod.close()
        driver_m5300x_Cons.close()
        print('Done!')

    except Exception as ex:
        print(ex)
        print('Exiting')

if __name__ == '__main__':
    main()

# © Keysight Technologies, 2021
# 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.
# ----------
```

8.1.7: Sample program using HVI to perform P2P streaming

```
"""
keysight_ktM5300x Python API Example Program
Requires Python 3.7 or newer and keysight_ktM5300x Python extension source module
installed using Package Installer for Python (pip install <fileName>.tar.gz).
"""
import time
import keysight_ktm5300xex as keysight_ktm5300x
import keysight_ktm5200xex as keysight_ktm5200x
import keysight_tse
query = False
reset = True

options_Rx = 'CompleterMinCredit=8192,RequesterMinCredit=8192'
options_Tx = 'CompleterMinCredit=8192,RequesterMinCredit=8192'

m5200aTx_resource = "PXI0::13-0.0::INSTR"
m5300aRx_resource = "PXI0::16-0.0::INSTR"

channel_config = int(input("Enter channel configuration............"))

# Considering m5300a is Tx, m5300a is Rx
global driver_m5200a_Tx, driver_m5300a_Rx

# Connecting with Module Drivers
print("adding m5200a...")
driver_m5200a_Tx = keysight_ktm5200x.KtM5200x(m5200aTx_resource, query, reset, options_Tx)
print("m5200a added")

print("adding m5300a...")
driver_m5300a_Rx = keysight_ktm5300x.KtM5300x(m5300aRx_resource, query, reset, options_Rx)
print("m5300a added")

driver_m5200a_Tx_ex = keysight_ktm5200x.KtM5200xEx(driver_m5200a_Tx)
resource_m5200aTx = driver_m5200a_Tx_ex.service.resources[0]
io = resource_m5200aTx.io

driver_m5300a_Rx_ex = keysight_ktm5300x.KtM5300xEx(driver_m5300a_Rx)
resource_m5300aRx = driver_m5300a_Rx_ex.service.resources[0]
io_rx = resource_m5300aRx.io
# Functions for static region configurations
```

```
def configure_mixer_static_region():
        if (channel_config == 1):
          mixer_freq = int(input("Enter mixer freq for Mixer 1 in MHz for 1Mhz enter 1   -> "))

driver_m5200a_Tx.peer_to_peer_streaming.set_center_frequency(keysight_ktm5200x.keysight_ktm520
0x.ChannelNumber.ONE, float(mixer_freq*1000000))
        elif (channel_config == 2):
          mixer_freq = int(input("Enter mixer freq for Mixer 1 in MHz for 1Mhz enter 1   -> "))
          mixer_freq1 = int(input("Enter mixer freq for Mixer 2 in MHz for 1Mhz enter 1   -> "))

driver_m5200a_Tx.peer_to_peer_streaming.set_center_frequency(keysight_ktm5200x.keysight_ktm520
0x.ChannelNumber.ONE, float(mixer_freq*1000000))

driver_m5200a_Tx.peer_to_peer_streaming.set_center_frequency(keysight_ktm5200x.keysight_ktm520
0x.ChannelNumber.TWO, float(mixer_freq1*1000000))
        elif(channel_config == 4):
          mixer_freq = int(input("Enter mixer freq for Mixer 1 in MHz for 1Mhz enter 1   -> "))
          mixer_freq1 = int(input("Enter mixer freq for Mixer 2 in MHz for 1Mhz enter 1   -> "))
          mixer_freq2 = int(input("Enter mixer freq for Mixer 3 in MHz for 1Mhz enter 1   -> "))
          mixer_freq3 = int(input("Enter mixer freq for Mixer 4 in MHz for 1Mhz enter 1   -> "))

driver_m5200a_Tx.peer_to_peer_streaming.set_center_frequency(keysight_ktm5200x.keysight_ktm520
0x.ChannelNumber.ONE, float(mixer_freq*1000000))

driver_m5200a_Tx.peer_to_peer_streaming.set_center_frequency(keysight_ktm5200x.keysight_ktm520
0x.ChannelNumber.TWO, float(mixer_freq1*1000000))

driver_m5200a_Tx.peer_to_peer_streaming.set_center_frequency(keysight_ktm5200x.keysight_ktm520
0x.ChannelNumber.THREE, float(mixer_freq2*1000000))

driver_m5200a_Tx.peer_to_peer_streaming.set_center_frequency(keysight_ktm5200x.keysight_ktm520
0x.ChannelNumber.FOUR, float(mixer_freq3*1000000))


def path_sel():

driver_m5200a_Tx.peer_to_peer_streaming.set_streaming_source(keysight_ktm5200x.keysight_ktm520
0x.m5300aStreamingSource.PROCESSED_DATA)

driver_m5300a_Rx.peer_to_peer_streaming.set_streaming_source(keysight_ktm5300x.keysight_ktm530
0x.m5300aStreamingSource.PROCESSED_DATA)



def program_channel_configuration():
            if (channel_config == 1):

driver_m5300a_Rx.peer_to_peer_streaming.set_channel(keysight_ktm5300x.keysight_ktm5300x.m5300a
StreamingChannel.ONE)
```

```
driver_m5200a_Tx.peer_to_peer_streaming.set_channel(keysight_ktm5200x.m5300aStreamingChannel.O
NE)
            elif (channel_config == 2):

driver_m5300a_Rx.peer_to_peer_streaming.set_channel(keysight_ktm5300x.keysight_ktm5300x.m5300a
StreamingChannel.TWO)

driver_m5200a_Tx.peer_to_peer_streaming.set_channel(keysight_ktm5200x.m5300aStreamingChannel.T
WO)
            elif(channel_config == 4):

driver_m5300a_Rx.peer_to_peer_streaming.set_channel(keysight_ktm5300x.keysight_ktm5300x.m5300a
StreamingChannel.FOUR)

driver_m5200a_Tx.peer_to_peer_streaming.set_channel(keysight_ktm5200x.m5300aStreamingChannel.F
OUR)


def peer_to_peer_configurtion():
            driver_m5200a_Tx.system.timeout = -1
            driver_m5300a_Rx.system.timeout = -1


            # Setting p2p in continuous streaming mode
            p2p_block_transfer_length = -1


            # Preparing streaming port for a peer-to-peer continuous data transfer as the bus
requester.
            driver_m5200a_Tx.peer_to_peer_ports["Port1"].role =
keysight_ktm5200x.m5300aPortRole.STREAMING_REQUESTER_PRODUCER
            [length,dataFormat,flow_conrol_addr] =
driver_m5200a_Tx.peer_to_peer_ports["Port1"].prepare_streaming_requester(0,p2p_block_transfer_
length,keysight_ktm5200x.m5300aDataFormat.I32,256); # fifo channel , Length



            # Preparing streaming port for a peer-to-peer continuous data transfer as the bus
completer.
            driver_m5300a_Rx.peer_to_peer_ports["Port2"].role =
keysight_ktm5300x.m5300aPortRole.STREAMING_COMPLETER_CONSUMER
            [length,dataformat,visa_addr,visa_offset] =
driver_m5300a_Rx.peer_to_peer_ports["Port2"].prepare_streaming_completer(0,length,keysight_ktm
5300x.m5300aDataFormat.I32,flow_conrol_addr);
            visa_session_id = driver_m5300a_Rx.peer_to_peer_ports["Port2"].visa_session_id



driver_m5200a_Tx.peer_to_peer_ports["Port1"].start_async(visa_session_id,visa_addr,visa_offset
,length)


def flush_p2p_fifo():
```

```
            print("Clearing both streaming FIFOs")
            driver_m5200a_Tx.peer_to_peer_streaming.flush_p2p()
            driver_m5300a_Rx.peer_to_peer_streaming.flush_p2p()




def set_phoenix_output_mode():
            print("\nset to Baseband waveform output...")

            driver_m5300a_Rx.channels[0].output_type =
keysight_ktm5300x.OutputWaveform.BASEBAND
            driver_m5300a_Rx.channels[0].amplitude = 100.0
            driver_m5300a_Rx.channels[0].enabled = True



            driver_m5300a_Rx.channels[1].output_type =
keysight_ktm5300x.OutputWaveform.BASEBAND
            driver_m5300a_Rx.channels[1].amplitude = 100.0
            driver_m5300a_Rx.channels[1].enabled = True



            driver_m5300a_Rx.channels[2].output_type =
keysight_ktm5300x.OutputWaveform.BASEBAND
            driver_m5300a_Rx.channels[2].amplitude = 100.0
            driver_m5300a_Rx.channels[2].enabled = True



            driver_m5300a_Rx.channels[3].output_type =
keysight_ktm5300x.OutputWaveform.BASEBAND
            driver_m5300a_Rx.channels[3].amplitude = 100.0
            driver_m5300a_Rx.channels[3].enabled = True


def close_p2p():
            driver_m5300a_Rx.peer_to_peer_streaming.enable_data_transfer(False)
            driver_m5200a_Tx.peer_to_peer_streaming.enable_data_transfer(False)

            driver_m5200a_Tx.peer_to_peer_ports["Port1"].stop()
            driver_m5300a_Rx.peer_to_peer_ports["Port2"].stop()


def SetupHviLoReset():
    global hvi

    cmi = driver_m5200a_Tx.get_instance()
    M5200_hvi = keysight_ktm5200x.KtM5200xHvi(cmi)
    my_system = keysight_tse.SystemDefinition("PeerToPeer Setup")
    M5200_hvi_engine_name = "M5200_HVI_Engine"
    M5200_hvi_engine = my_system.engines.add(M5200_hvi.engines.main_engine,
M5200_hvi_engine_name)

    M5200_reset_lo_actions = []

M5200_reset_lo_actions.append(M5200_hvi_engine.actions.add(M5200_hvi.actions.p2p_mixer_reset,
"ResetLo"))

    hvi_sequencer = keysight_tse.Sequencer("Sequencer", my_system)
    sync_sequence = hvi_sequencer.sync_sequence
    sequence = sync_sequence.add_sync_multi_sequence_block('SyncExec',
100).sequences[M5200_hvi_engine_name]
    action_execute = sequence.instruction_set.action_execute
    engine = hvi_sequencer.sync_sequence.engines[M5200_hvi_engine_name]

    M5200_reset_lo_actions = []
    M5200_reset_lo_actions.append(engine.actions['ResetLo'])

    M5200_reset_lo_instruction = sequence.add_instruction('ResetM5200Lo', 1000,
action_execute.id)
    M5200_reset_lo_instruction.set_parameter(action_execute.action.id, M5200_reset_lo_actions)


    hvi = hvi_sequencer.compile()
    fn = "hvi_seq.txt"
    seq_output = hvi_sequencer.sync_sequence.to_string(keysight_tse.OutputFormat.DEBUG)
    with open(fn, 'w') as f:
        f.write(seq_output)
    f.close()
    print("HVI Module Engine ID: ", M5200_hvi.engines.main_engine)
    print("load_to_hw")
    hvi.load_to_hw()


def HviLoReset():
    print("run")
    hvi.run(hvi.no_timeout)

def main():
    try:
        program_channel_configuration()

        set_phoenix_output_mode()

        ##################################
        ##  HVI Action for Mixer reset  ##
        ##################################
        SetupHviLoReset()

        driver_m5300a_Rx.peer_to_peer_streaming.enable_data_transfer(False)
        driver_m5200a_Tx.peer_to_peer_streaming.enable_data_transfer(False)

        configure_mixer_static_region()

        peer_to_peer_configurtion()


        # Clearing both streaming FIFOs
        flush_p2p_fifo()

        path_sel()

        # Waiting for produer and consumer to get configured
        time.sleep(1)

        driver_m5300a_Rx.peer_to_peer_streaming.enable_data_transfer(True)
        driver_m5200a_Tx.peer_to_peer_streaming.enable_data_transfer(True)


        HviLoReset()

       # waiting in infinite while loop unless asked to quit be pressing "q" and Can change the
LO Frequency
        while 1:
           var = input("Enter q to quit or press any key to continue ->  ")
           if(var == "q" ):
             break
          freq_switch = int(input("Enter 1 to change the mixer frequency and 0 to continue with
current mixer frequency  ->  "))
           if freq_switch == 1:
               # Stop p2p transfer. Also releases hardware resources used by FIFO streaming.
```

```
                close_p2p()

                configure_mixer_static_region()

                peer_to_peer_configurtion()

                # Clearing both streaming FIFOs
                flush_p2p_fifo()

                time.sleep(1)
                driver_m5200a_Tx.peer_to_peer_streaming.ddc_phase_reset()
                driver_m5300a_Rx.peer_to_peer_streaming.enable_data_transfer(True)
                driver_m5200a_Tx.peer_to_peer_streaming.enable_data_transfer(True)


        close_p2p()
        hvi.release_hw()
        driver_m5200a_Tx.close()
        driver_m5300a_Rx.close()
        input()
        print('Done!')

    except Exception as ex:
        print(ex)
        print('Exiting')

if __name__ == '__main__':
    main()
# © Keysight Technologies, 2021
# 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.
# ----------
```

9. Exceptions/Errors/Warnings

KEYSIGHT
TECHNOLOGIES

Section 9.1: Description of messages returned by the SW

Table 45 lists the exceptions/errors/warnings returned by the software, along with the erroneous conditions, as a result of which the software returns such messages. It also informs you the necessary action to be taken to eradicate / avoid such errors.

From the main menu of the SFP, click **Utilities** > **Errors...** or run the `driver.utility.error_query()` API to view the errors from the error queue of the software.

Table 45 List of Exceptions / Errors / Warnings

Exceptions/Errors/Warnings in the Error queue	Condition for error	Possible action for eradication of error
Interrupt caught- external 2.4GHz clock is not present.	Unavailability of external analog 2.4 GHz clock or noise in the external analog clock.	Check the external analog clock source and connection.
External 2.4GHz clock not present, configuring module with internal 2.4GHz clock.	During module initialization or while switching to external analog 2.4 GHz clock, if external clock is not available.	Check the external analog clock source and connection.
Clock alignment failed!	Alignment failure occurs when there is a problem with the analog clock.	Make sure 100 MHz and 2.4 GHz clocks are locked to one another. Check analog clock source and hardware connections.
Align Clock failed - Error: Clock alignment: No clock alignment data available to reuse (Missing File: <*missing-file-location*>). Perform an initial clock alignment.	No calibration data is available for the SW to reuse, when Set Clock Alignment is selected as “ReUse”.	Set Clock Alignment to “Initialize” to perform clock calibration.
Align Clock failed - Error: Clock alignment: Invalid clock alignment data found, temperature exceeded beyond limit from the last reset alignment (Current: <*current-temp-logged-in-celsius*> & Last: <*last-temp-logged-in-celsius*>). Perform an initial clock alignment.	Set Clock Alignment is selected as “ReUse” and SW detects that alignment temperature has varied by more than 10 degrees Celsius from the last time reset alignment was done.	Set Clock Alignment to “Initialize” to perform clock calibration.
SMB Skew Initialization halted.	SMB calibration failure.	Try opening the module with **reset=1** option. If error persists, it is likely a hardware failure. Contact Keysight Support.
Module SPI calibration failed.	-	Try a power cycle. If error persists, it is likely a hardware failure. Contact Keysight Support.
Unsupported version of FPGA version Found : x.x.x. Please update the FPGA to y.y.y.y	Incompatible software and FPGA combination used.	Perform a software upgrade. Check Release Notes document for compatible FPGA version information. Note that ‘x.x.x.x’ represents the currently installed version and ‘y.y.y.y’ indicates the required version of the FPGA.
No kernel found matches deviceId: xxxx. Supported deviceId: xxxx Include ‘FpgaVersion=<FPGA-Version>’ flag in the ‘options’ string.	Occurs if the FPGAVersion flag is not defined in the ‘options’ string in the simulation mode (Simulate=true).	In the simulation mode, for the k7z file to be loaded successfully using the configure_from_k7z() function, ensure that the FPGA version is defined as "FPGAVersion=x.x.x" (replace x.x.x with the FPGA version for which k7z is generated) in the ‘options’ attribute in your script. For example, options = "Simulate=true, FPGAVersion=1.1.414".

Exceptions/Errors/Warnings in the Error queue	Condition for error	Possible action for eradication of error
Module Initialization halted with message: Delay control register not ready.	Improper installation of the module drivers cause Python drivers to not be updated.	Perform a removal and installation of the module drivers. If error persists, it is likely a hardware failure. Contact Keysight Support.
Visa error 0xbfff0011. VI_ERROR_RSRC_NFOUND: No matching resources found / no more matches.	Improper installation of the module drivers.	Perform a clean installation of the module drivers.
The HVI engine version (X) does not match the expected version (X).	Incompatible versions of the software, KF9000B PathWave FPGA, and KS2201A PathWave Test Sync Executive.	Perform a software upgrade. Check Release Notes document for compatible versions of KF9000B PathWave FPGA and KS2201A PathWave Test Sync Executive.
operation "LinkAlignment" failed: Could not align link...	FDS link alignment failed between M9032A/M9033A modules and one or more M5000-series modules.	Check Release Notes document for compatible versions of the respective M5000-series module driver, M9032A/M9033A module drivers and KS2201A PathWave Test Sync Executive.
Module Initialization halted with Message: Could not program skew	Most likely a case of overdriving, as a result of incompatible SW and FPGA combinations or improper installation of module drivers.	Perform a removal and installation of the module drivers. Check Release Notes document for compatible FPGA version information. If error persists, it is likely a hardware failure. Contact Keysight Support.
No Calibration file found.	Missing calibration file from flash memory, which indicates that the module is not calibrated.	Module requires calibration.
Unable to open instrument. Diver initialization failed. Invalid argument: memory 'UserSandboxDma' does not exist.	Missing one of the following elements: ▪ stage 2 image ▪ chassis driver ▪ system module driver	▪ Try opening the module with **initialize=0** option to display the fpga image version the SW is looking for. Upgrade the module drivers. This fixes the issue with stage 2 image. ▪ Upgrade the chassis drivers.
Module Initialization halted with message: Error : Clock initialization failed: Module SPI communication failed. Please try cycling power.	-	Try to restart the module or perform a power cycle. If error persists, it is likely a hardware failure. Contact Keysight Support.
Module Initialization halted with message: Unable to select 2.4GHz internal.	Possibly a hardware issue, where module is unable to complete calibration.	Try to perform a power cycle and open the module with **loglevel=finest** option for additional information.
Tx Ready asserted but Valid is not asserted.	The TX_Ready signal on the FDS line is 'HIGH'. However, the TX_Valid signal is 'LOW' and irrespective of whether or not valid data available to transmit.	The TX_Ready and the TX_Valid signals on the FDS line must be 'HIGH'.
Tx Valid asserted but Ready is not asserted.	The TX_Valid signal on the FDS line is 'HIGH' and valid data available to transmit. However, the TX_Ready signal is 'LOW'.	The TX_Ready and the TX_Valid signals on the FDS line must be 'HIGH'.
Rx Ready asserted but Valid is not asserted.	The RX_Ready signal on the FDS line is 'HIGH'. However, the RX_Valid signal is 'LOW'.	The RX_Ready and the RX_Valid signals on the FDS line must be 'HIGH'.
Rx Valid asserted but Ready is not asserted.	The RX_Valid signal on the FDS line is 'HIGH' However, the RX_Ready signal is 'LOW'.	The RX_Valid and the RX_Ready signals on the FDS line must be 'HIGH'.
Frequency out of range. Module supports LO frequencies in the range (0, 1...12GHz) only.	Input LO frequency is greater than 12 GHz. However, module includes the '012' option only.	Input LO frequency value should be in the specified range. For frequencies greater than 12 GHz, procure the '012' option.

Exceptions/Errors/Warnings in the Error queue	Condition for error	Possible action for eradication of error
Module M53xxA in slot nn: Digital I/O Self Test finished: Failed due to incorrect SMB Port connections.	Failure in Digital I/O self test run on a module (from the M5000-series) in a specific slot 'nn' due to incorrect cable connections, as indicated.	Check the cable connections to the module in the specified slot number.
Error: CRITICAL WARNING: [Vivado xx-yy] set_clock_groups:No valid object(s) found for '-group <[group_name]>'. Resolution: Check if the specified object(s) exists in the current design. If it does, ensure that the correct design hierarchy was specified for the object. If you are working with clocks, make sure create_clock was used to create the clock object before it is referenced. CRITICAL WARNING: [Timing mm-nn] Multiple clocks arrive at pins <pin_names>. A single clock must arrive at pin <pin_names> and it must have the same master clock as a single clock arriving at pin <pin_names>, with the latter only being phase shifted by 0/90/180/270 degrees. Any auto-derived clock on pin <pin_names> will be created with 0 phase. CRITICAL WARNING: [Vivado xx-yy] Please provide a list of objects. CRITICAL WARNING: [Timing mm-nn] The design failed to meet the timing requirements. Please see the timing summary report for details on the timing violations.	When the abstract shell is created in the sandbox, the clock routes get disconnected. So, the timing constraint is not applied properly within the abstract shell logic, which is driving the static logic.	Ignore any such critical warnings.
Prompts pertaining to P2P Streaming		
Error: Peer To Peer Streaming requires M5000AU-001 license (Backplane streaming option for RF AWG)	M5000AU-001 license, which is required for P2P streaming, is missing for the corresponding M5300A RF AWG module.	Install the require license to enable Peer To Peer Streaming on the M5300A RF AWG module.
Error: Peer To Peer Streaming requires M5000AU-002 license (Backplane streaming option for RF Digitizer)	M5000AU-002 license, which is required for P2P streaming, is missing for the M5200A RF Digitizer module.	Install the require license to enable Peer To Peer Streaming on the M5200A RF Digitizer module.
Interrupt caught - peer To peer streaming encountered with an error	Prompted when an error occurs during an ongoing P2P streaming transaction. Possible causes for this error may be: ▪ Modules do not meet the slot positioning requirement for P2P streaming ▪ Chassis has achieved Lower Speed linkups, such as G1 or G2, instead of the required G3 speeds ▪ Attempted transfer size is larger than the available PCIe throughput, especially when custom sandbox design may have been loaded ▪ Unstable external reference clock source	Perform one or more of the following actions: ▪ Make sure that both AWG and Digitizer module pair being used for streaming is installed in the correct slot range ▪ Chassis is operating at G3 speeds ▪ Streaming throughput is less than or equal to that of the PCIe ▪ Fix/replace the external clock source
Failed to enable streaming data transfer, PCIe link is down	Prompted when you try to enable P2P streaming transfer but the PCIe link is down. This condition occurs when the PXIe Chassis is malfunctioning or is in an unhealthy condition & has an unstable PCIe link.	Perform a power cycle on the Chassis. If error persists, it is likely a hardware failure. Contact Keysight Support.

Index

M5300-91002
www.keysight.com