User's Guide

Keysight M31XXA M33XXA Digitizers

KEYSIGHT TECHNOLOGIES

Notices

Copyright Notice

© Keysight Technologies 2013-2020

No part of this manual may be reproduced in any form or by any means (including electronic storage and retrieval or translation into a foreign language) without prior agreement and written consent from Keysight Technologies, Inc. as governed by United States and international copyright laws.

Manual Part Number

M3100-90002

Published By

Keysight Technologies
1400 Fountaingrove Parkway
Santa Rosa
CA 95405

Edition

Edition 2, February, 2020
Printed In USA

Regulatory Compliance

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

Warranty

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

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

Technology Licenses

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

U.S. Government Rights

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

Safety Notices

CAUTION

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

WARNING

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

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

This product is intended for use by qualified personnel who recognize

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

WARNING

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

The types of product users are:

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

WARNING

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Instrumentation and accessories shall not be connected to humans.

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

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

WARNING

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

PRODUCT MARKINGS:

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

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

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

South Korean Class A EMC Declaration. This equipment is Class A suitable for professional use and is for use in electromagnetic environments outside of the home. A 급 기기 (업무용 방송통신기자재) 이 기기는 업무용 (A 급) 전자파적합기기로서 판 매 자 또는 사용자는 이 점을 주 의하시기 바라 며 , 가정외의 지역에서 사용하는 것을 목적으로 합니다 .

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

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

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

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

Contents

1 Theory of Operation: M31/M33XX Digitizers

M31/M33XX digitizers are part of the new M3XXXA family of FPGA-programmable AWGs and digitizers. These high-performance digitizers with high channel density (up to 8 Ch in a single 3U PXIe slot) have an advanced data acquisition system (DAQ), includes easy-to-use programming libraries (section 3.1.1.1) and provides optional real-time sequencing and decision making capability (Hard Virtual Instrumentation or HVI: section 3.1.2.1) with precise timing and multi-module synchronization. Graphical FPGA programing (section 3.1.2.2) allows for FPGA customization without HDL programming expertise and performance penalty.

1. 1 Digitizer Operation

The M31/M33xx Digitizers has a flexible and powerful input structure to acquire signals (Figure 1).

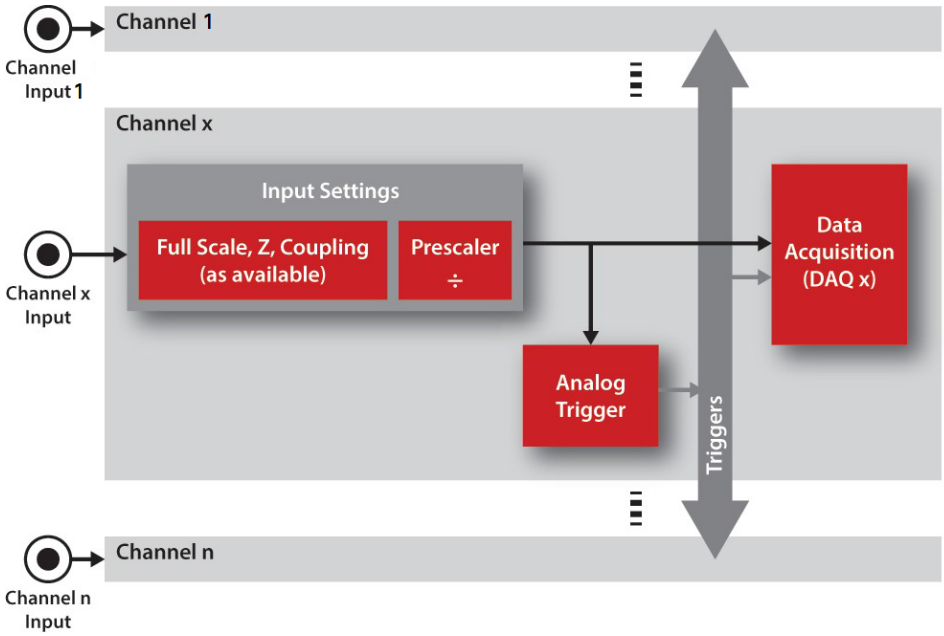


Figure 1: M31/M33XXA Digitizers input functional block diagram. All input channels have identical structure.

Figure 1 shows Keysight standards for the output labeling (channel enumeration starts with CH1), however function open (Section 2.4.2.1) provides a compatibility mode for legacy modules (channel enumeration starts with CH0).

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

The compatibility options are shown in Table 0.

Output Signal Selection

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

Table 0: Compatibility, Section 2.4.2.1)

1.1.1 Input Settings

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

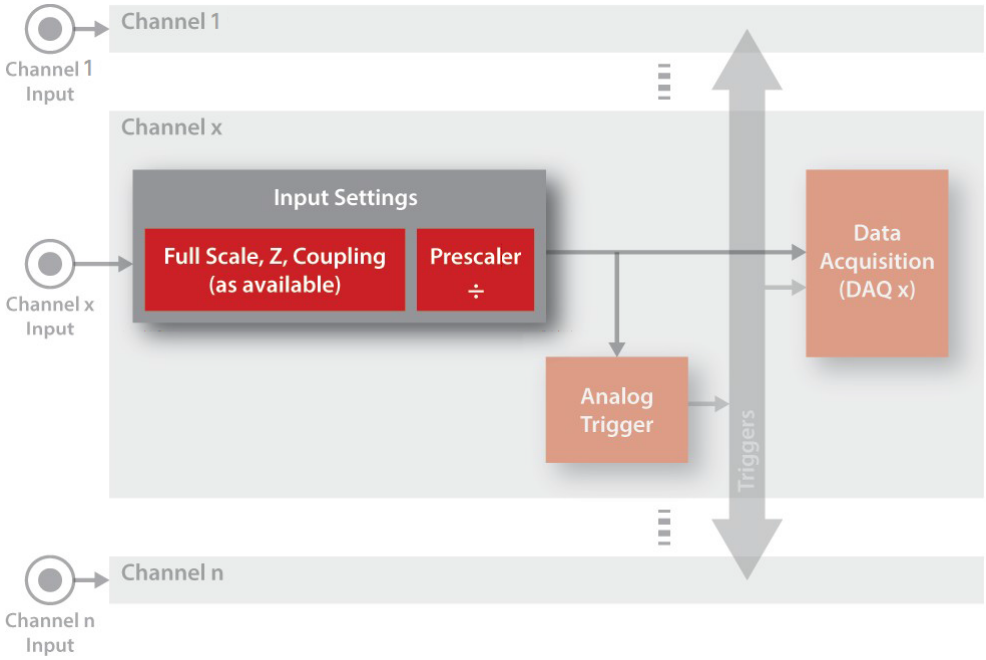


Figure 2: Input settings in the M31/M33XXA Digitizers functional block diagram

1.1.1.1 Full Scale, Impedance and Coupling

Depending on the product specifications the user can configure the input full scale value, the input impedance (Table 1) and the input coupling (Table 2). The full scale parameter adjusts automatically the input gain to maximize the input dynamic range.

Product-dependent Settings: This Section describes all the possible input settings, but in reality they are product-dependent. Please check the corresponding product datasheet to see if they are applicable.

Input Impedance

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

Table 1: Options for the input impedance setting (parameter impedance in function channelInputConfig (page 45).

Input Coupling

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

Table 2: Options for the input coupling setting (parameter coupling in function channelInputConfig (page 45).

Programming Information

Function Name	Comments	Details
channelInputConfigIt	configures the input full scale, impedance and coupling settings	Section 2.3.4.1 on page 28

Table 3: Programming functions related to the input full scale, impedance and coupling settings

1. 1. 1. 2 Prescaler

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

$$fs = fCLKsys/(1 + prescaler)$$

where:

- fs is final effective sampling frequency
- $fCLKsys$ is the Clock System
- $prescaler$ is an integer value (4 bits)

NOTE **Advanced**: Prescaler vs. Downsampling/Decimation: The prescaler cannot be considered as a full decimation or downsampling block, as it does not contain any filter and therefore it generates aliasing. For applications were full downsampling is required, the user must choose an IF Keysight Digitizer or Transceiver, which provide DDC (Digital Down Conversion).

Programming Information

Function Name	Comments	Details
channelPrescalerConfig	It configures the input prescaler	channelPrescalerConfig (page 46)

Table 4: Programming functions related to the input prescaler

1.1.2 Analog Trigger

The analog trigger block processes the input data and generates a digital trigger that can be used by any Data Acquisition Block (Data Acquisition (DAQs) (page 8)).

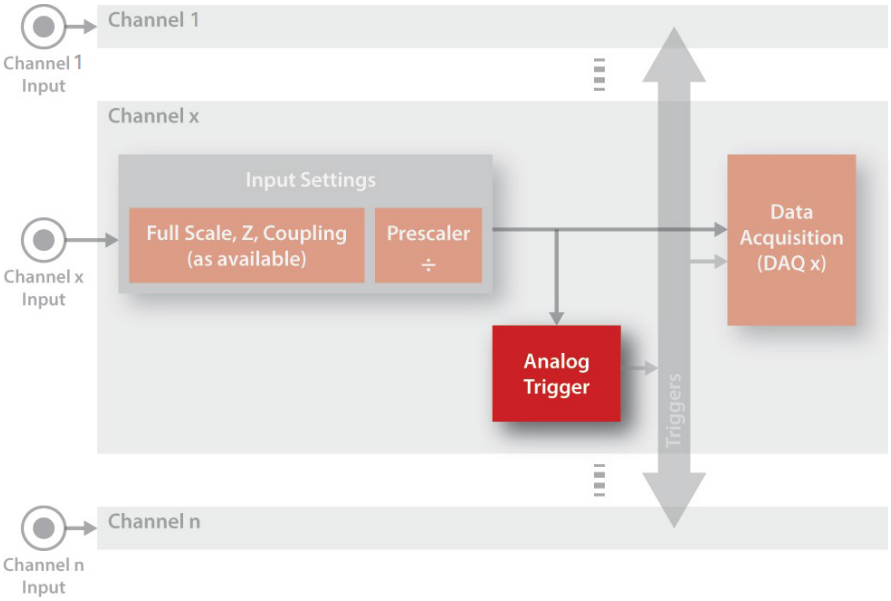


Figure 3: Analog trigger processor in the M31/M33XXA Digitizers functional block diagram

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

Analog Trigger Mode

Programming Definitions

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

Table 5: Options for the analog trigger (parameter analogTriggerMode in function channelTriggerConfig (page 47)

Programming Information

Function Name	Comments	Details
channelTriggerConfig	It configures the analog trigger for each channel	channelTriggerConfig (page 47)

Table 6: Programming functions related to the analog triggers

1. 1. 3 Data Acquisition (DAQs)

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

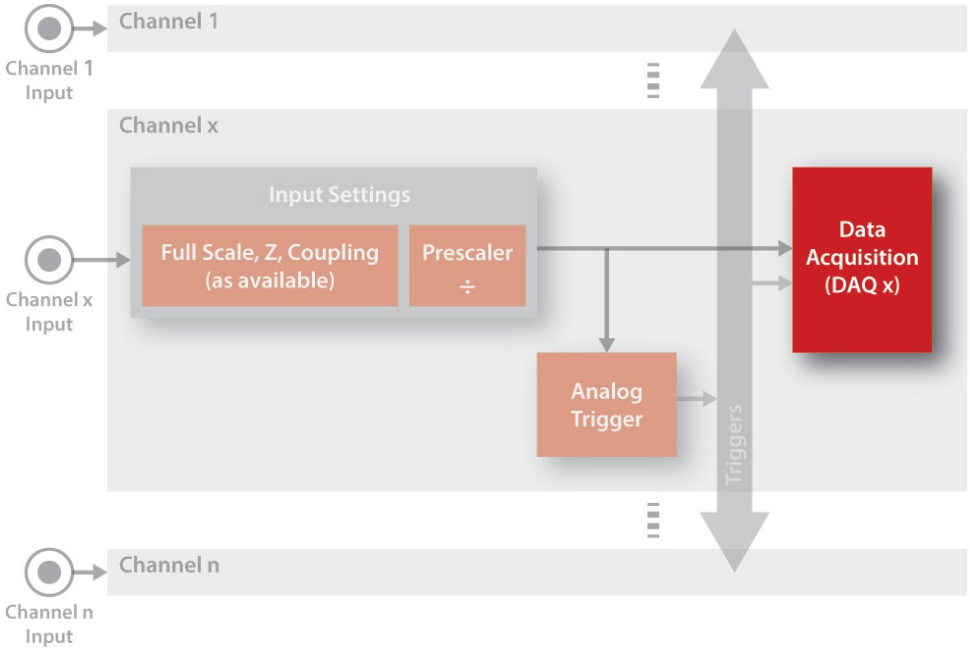


Figure 4: DAQ units in the M31/M33XXA Digitizers

1. 1. 3. 1 Operation

The words acquisition requires two easy steps:

1. Configuration: A call to the function DAQconfig (page 48) allows the user to configure, among others, the trigger method, the number of DAQ cycles to perform (number of triggers), the number of acquired words per cycle (DAQ-pointsPerCycle), and the number of words that must be acquired before interrupting the PC (DAQpoints). Figure 6 on the following page illustrates the flexibility of the DAQ block operation.
2. Data read:
   a. Using DAQread (page 51) : a blocking/non-blocking function that returns the array of acquired words (DAQ- data).
   b. Using a callback function: a user function that is automatically called when the configured amount of data (set with DAQpoints) is available.

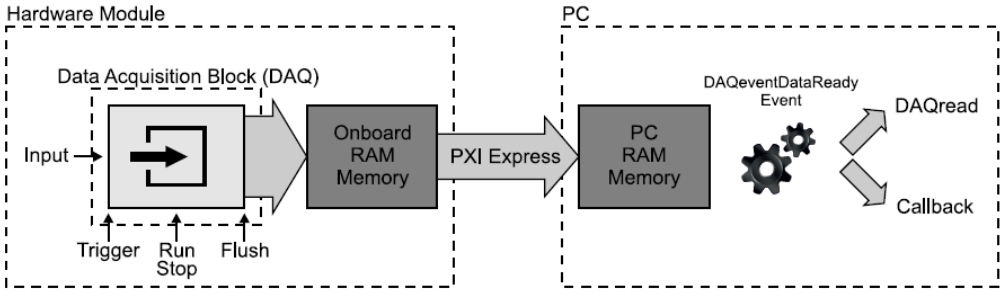


Figure 5: M31/M33XXA Digitizers words acquisition operation

NOTE **Tip:** Pausing and Resuming the DAQ: The function DAQstop (page 55) pauses the DAQ operation (triggers are discarded). The acquisition can be resumed calling DAQstart (page 53).
A DAQstop is performed automatically when the DAQ block reaches the specified number of acquisition cycles.

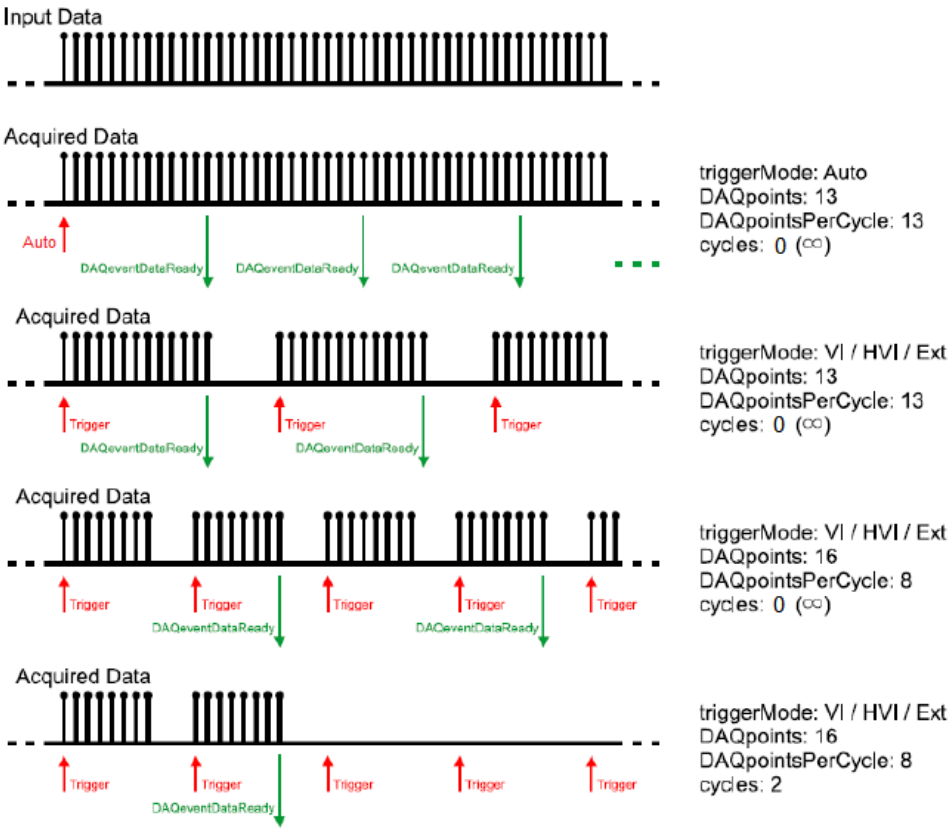


Figure 6: Examples of the DAQ operation (not all trigger methods are available in all modules)

NOTE **Advanced**: Onboard memory and DAQpoints selection: The acquired words are first stored in a DAQ buffer located in the module onboard RAM and then sent to the PC RAM via the ultra-fast PXI Express bus (Figure 5 on the previous page) using dedicated DMA channels. This DAQ buffer, and therefore the minimum amount of onboard RAM needed for an application, is directly the size of the data array the user wants to receive in each DAQread or callback function call. This data array, called DAQdata, contains DAQpoints words. Therefore, its size is DAQpoints x 2 bytes/word (in the case of the M31/M33XXA Digitizers). The size of this DAQ buffer must be chosen according to two criteria:

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

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

Prescaler: The DAQ block has an input prescaler which can be configured to discard words, reducing the effective acquisition data rate (function DAQconfig (page 48)).

DAQ counter: The DAQ block has a dedicated counter to store the number of acquired words since the last call to DAQconfig or DAQflush. This counter can be read with DAQcounterRead (page 65).

1. 1. 3. 2 DAQ Trigger

As previously explained, the user can configure the trigger for the acquisition. The available trigger modes for the DAQ are show in Table 7).

DAQ Trigger Mode

Option	Description	Programming Definitions	
		Name	Value
Auto (Immediate)	The acquisition starts automatically after a call to function DAQstart (page 53)	AUTOTRIG	0
Software / HVI	Software trigger. The acquisition is triggered by the function DAQtrigger, DAQtrigger (page 63) provided that the DAQ is running. DAQtrigger can be executed from the user application (VI) or from an HVI (see HVI details, HVI Programming Overview (page 24))	SWHVITRIG1	1
Hardware Digital Trigger	Hardware trigger. The DAQ waits for an external digital trigger	HWDIGTRIG	2
Hardware Analog Trigger	Hardware trigger. The DAQ waits for an external analog trigger (only products with analog inputs)	HWANATRIG	3

Table 7: DAQ trigger modes (parameter triggerMode in function DAQconfig, Section 2.4.5.4)

As show in Table 7, the user has the following options for hardware triggers:

- Hardware Digital Trigger: If the DAQ is set to use a digital hardware trigger, the user must configure it using the function DAQdigitalTriggerConfig (page 49). The available digital hardware trigger options are shown in Table 8 and Table 9. If external I/O trigger is selected in DAQdigitalTriggerConfig, the user must configure additional settings of this particular I/O line, such as input/output direction, sampling/synchronization options, etc. (I/O Triggers (page 15)).

- Hardware Analog Trigger: (Only products with analog inputs) If the DAQ is set to use an analog hardware trigger, the user must configure it using the function DAQanalogTriggerConfig (page 50). The Analog Trigger Block of the corresponding analog input channel must also be configured.

DAQ Hardware Digital Trigger Source

Option	Description	Name	Value
External I/O Trigger	The DAQ trigger is a TRG connector/line of the product (I/O Triggers (page 15)). PXI form factor only: this trigger can be synchronized to CLK10, see Table	TRIG_EXTERNAL	0
PXI Trigger	PXI form factor only. The DAQ trigger is a PXI trigger line and it is synchronized to CLK10	TRIG_PXI	1

Table 8: External trigger source for the DAQ (parameter externalSource in function DAQdigitalTriggerConfig (page 49)).

DAQ Hardware Digital Trigger Behaviour

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

Table 9: Hardware digital trigger behaviour for the DAQ (parameter triggerBehaviour in function DAQdigitalTriggerConfig (page 49))

1. 1. 3. 3 Programming Functions Summary

Programming Information

Function Name	Comments	Details
channelInputConfig	This function sets the input full scale, impedance and coupling	Section 2.4.5.1
channelPrescalerConfig	This function configures the input	Section 2.4.5.2
channelTriggerConfig	This function configures the analog trigger block for each channel	Section 2.4.5.3
DAQconfig	This function configures the acquisition of words Data Acquisition (DAQs)	Section 2.4.5.4
DAQdigitalTriggerConfig	This function configures the digital hardware triggers for the selected DAQ Trigger	Section 2.4.5.5
DAQanalogTriggerConfig	This function configures the analog hardware trigger for the selected DAQ Trigger	Section 2.4.5.6
DAQread	This function reads the words acquired with the selected DAQ Data Acquisition. It can be used only after calling the function DAQconfig and when a callback function is not configured	Section 2.4.5.7
DAQstart	This function starts acquisition on the selected DAQ	Section 2.4.5.8
DAQstartMultiple	This function starts acquisition on multiple DAQs	Section 2.4.5.9
DAQstop	This function stops the words acquisition Data Acquisition of a DAQ	Section 2.4.5.10
DAQstopMultiple	This function stops the words acquisition Data Acquisition of multiple DAQs	Section 2.4.5.11
DAQpause	This function pauses the words acquisition Data Acquisition of a DAQs. Acquisition can be resumed using DAQresume.	Section 2.4.5.12
DAQpauseMultiple	This function pauses the words acquisition Data Acquisition of multiple DAQs. Acquisition can be resumed using DAQre-	Section 2.4.5.13

Function Name	Comments	Details
	sumeMultiple.	
DAQresume	This function resumes acquisition on the selected DAQ Data Acquisition	Section 2.4.5.14
DAQresumeMultiple	This function resumes acquisition on multiple DAQs Data Acquisition	Section 2.4.5.15
DAQflush	This function flushes the acquisition buffers and resets the acquisition counter included in Data Acquisition block of a DAQ	Section 2.4.5.16
DAQflushMultiple	This function flushes the acquisition buffers and resets the acquisition counter included in multiple Data Acquisition blocks of the DAQs	Section 2.4.5.17
DAQtrigger	This function triggers the acquisition of words in the selected DAQ	Section 2.4.5.18
DAQtriggerMultiple	This function triggers the acquisition of words in multiple DAQs	Section 2.4.5.19
DAQcounterRead	This function reads the number of words acquired by the selected DAQ	Section 2.4.5.20
triggerIOconfig	This function configures the trigger connector/line direction and synchronization/sampling method	Section 2.4.5.21
triggerIOwrite	This function sets the trigger output. The trigger must be configured as output using function triggerIOconfig and I/O Triggers	Section 2.4.5.22
triggerIOread	This function reads the trigger input	Section 2.4.5.23
clockSetFrequency	This function sets the module clock frequency	Section 2.4.5.24
clockGetFrequency	This function returns the real hardware clock frequency	Section 2.4.5.25
clockGetSyncFrequency	This function returns the frequency of Clock System	Section 2.4.5.26
clockResetPhase	This function sets the module in a sync state, waiting for the first trigger to reset the phase of the internal clocks CLKsync and CLKsys	Section 2.4.5.27
DAQbufferPoolConfig	This function configures buffer pool that will be filled with the data of the channel to be transferred to PC.	Section 2.4.5.28
DAQbufferAdd	Adds an additional buffer to the channel's previously configured pool.	Section 2.4.5.29
DAQbufferGet	Gets a filled buffer from the channel buffer pool. User has to call DAQbufferAdd with this buffer to tell the pool that the buffer can be used again.	Section 2.4.5.30
DAQbufferPoolRelease	Releases the channel buffer pool and its resources. After this call, user has to call DAQbufferRemove consecutively to get all buffers back and release them.	Section 2.4.5.31
DAQbufferRemove	Ask for a buffer to be removed from the channel buffer pool. If NULL pointer is returned, no more buffers remains in buffer pool. Returned buffer is a previously added buffer from user and user has to release/delete it.	Section 2.4.5.32
FFT	Calculates the FFT of data captured by DAQread for the selected channel	Section 2.4.5.33

Table 10: Programming functions related to the digitizers

1. 2 I/O Triggers

The M31/M33XXA Digitizers has general purpose input/output triggers (TRG connectors/lines). A trigger can be used as general purpose digital IO or as a trigger input (Table 11), and can be sampled using the options shown in Table 12.

Trigger I/O Options

		Programming Definitions	
Option	Description	Name	Value
Trigger Output (readable)	TRG operates as a general purpose digital output signal, which can be written by the user software	AOU_TRG_OUT	0
Trigger Input	TRG operates as a trigger input, or as general purpose digital input signal, which can be read by the user software	AOU_TRG_IN	1

Table 11: M31/M33XXA Digitizers TRG connectors configuration (triggerIOconfig (page 66))

Trigger Synchronization/Sampling Options

		Programming Definitions	
Option	Description	Name	Value
Non-synchronized mode	The trigger is sampled with an internal 100 MHz clock	SYNC_NONE	0
Synchronized mode	(PXI form factor only) The trigger is sampled using CLK10	SYNC_CLK_0	1

Table 12: M31/M33XXA Digitizers TRG connectors configuration (triggerIOconfig (page 66))

1. 3 Clock System

The M31/M33XXA Digitizers uses an internally generated high-quality clock (CLKref) which is phase-locked to the chassis clock. Therefore, this clock is an extremely jitter-cleaned copy of the chassis clock. This implementation achieves a jitter and phase noise above 100 Hz which is independent of the chassis clock, depending on it only for the frequency absolute precision and long term stability. A copy of CLKref is available at the CLK connector (Table 13).

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

NOTE **Advanced**: Chassis Clock Replacement for High-Precision Applications: For applications where clock stability and precision is crucial (e.g. GPS, experimental physics, etc.), the user can replace the chassis clock with an external reference. In the case of PXI/PXIe, this is possible via a chassis clock input connector or with a PXI/PXIe timing module. These options are not available in all chassis, please check the corresponding chassis specifications.

CLK Output Options

		Programming Definitions	
Option	Description	Name	Value
Disable	The CLK connector is disable	n/a	0 (default)
CLKref Output	A copy of the reference clock is available at the CLK connector	n/a	1

Table 13: M31/M33XXA Digitizers output clock configuration

1. 3. 1 FlexCLK Technology (models with variable sampling rate)

The sampling frequency of the M31/M33XXA Digitizers (CLKsys frequency) can be changed using the advanced clocking system shown in Figure 7.

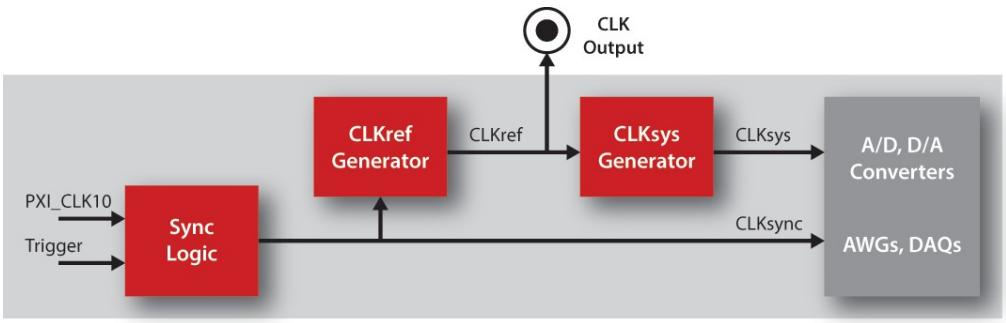


Figure 7: Functional diagram of the M31/M33XXA Digitizers FlexCLK system

where:

- CLKref is the internal reference clock, which is phase-locked to the chassis clock.
- CLKsys is the system clock, used to sample data.
- CLKsync is an internal clock used for the synchronization features of the M31/M33XXA Digitizers.
- PXI CLK10 is the 10 MHz clock of the PXI/PXIe backplane (PXI/PXIe only).

The CLKsys frequency can be changed within the range indicated in the datasheet of the corresponding product (function clockSet- Frequency, Section 2.3.4.17 on page 44). The CLKsync frequency changes with the CLKsys frequency as follows:

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

The CLKsync frequency is returned by the function clockSetFrequency (page 69)

CLK Set Frequency Mode

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

Table 14: Clock set frequency mode (function clockSetFrequency (page 69)

2 Software Tools: M31/M33XXA Digitizers

2. 1 Software Front Panel

2. 1. 1 Software Front Panel: Keysight SD1 SFP

2. 1. 1. 1 Keysight SD1 SFP (Software Front Panels) Overview

Keysight hardware products can be operated as classical benchtop instruments using Keysight SD1 SFP [2], a software front panel application. This chapter describes the M31/M33XXA digitizer SD1 SFP front panel. Consult Section 3.3.1.

2. 1. 1. 2 Main Front Panel

Figure7a shows the M32/M33XXA AWGs SD1 SFP front panel, which appears automatically when SD1 SFP is launched and the module is connected to the chassis. If there are no modules available, SD1 SFP will launch ”Demo Offline” modules

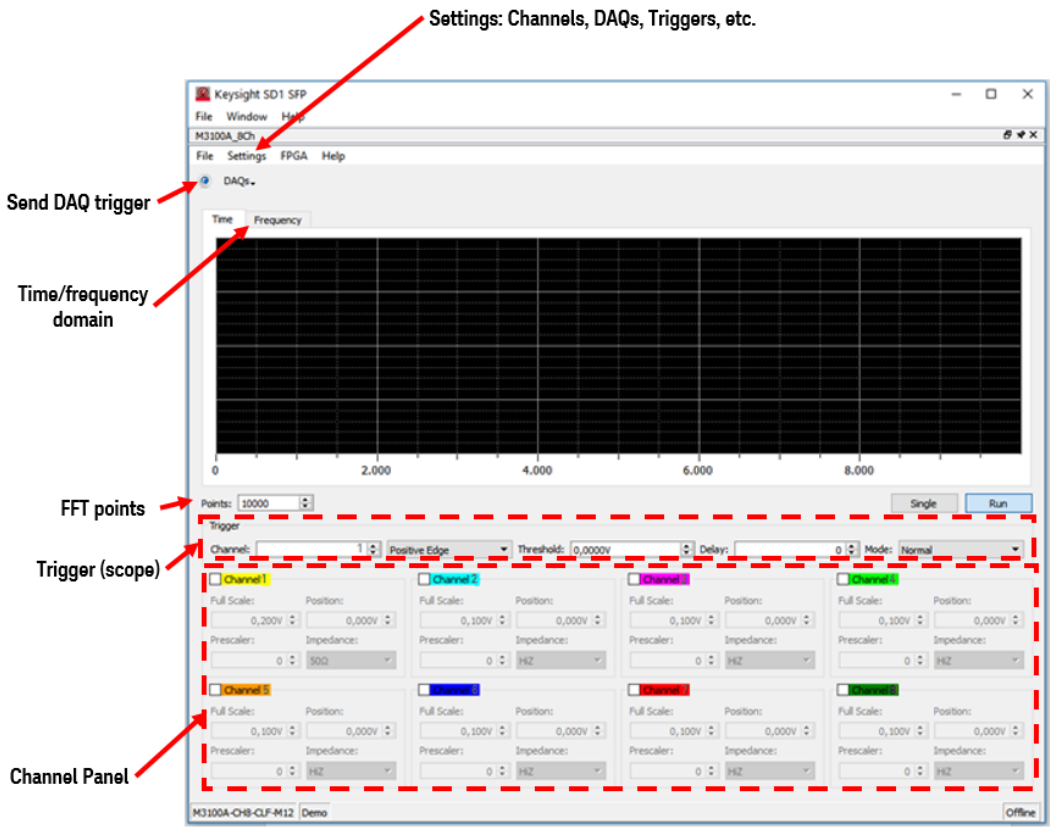


Figure: 7a

When SD1 SFP is launched, the M32/M33XXA digitizers front panel appears populated with all available channels, waiting for the user to configure the input ”Channels”. Digitizer front panel provide both time domain (scope like functionality) and frequency domain (spectrum analyzer like functionality).

2. 1. 1. 3 Input settings

Figure7b shows the input controls of the M32/M33XXA digitizers in SD1 SFP.

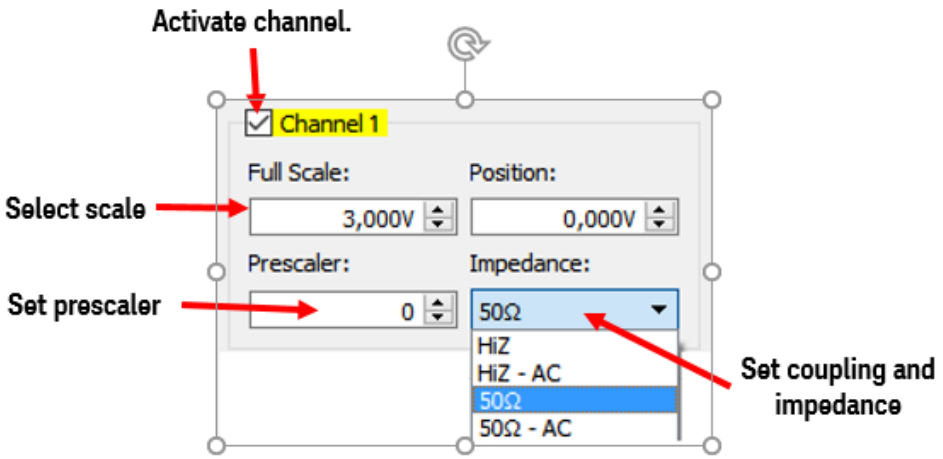


Figure7b: M31/M33XXA digitizer input control.

2. 1. 1. 4 Time domain (scope like operation)

Figure7c shows the dialogs and the workflow to use the SD1 SFP as a scope.

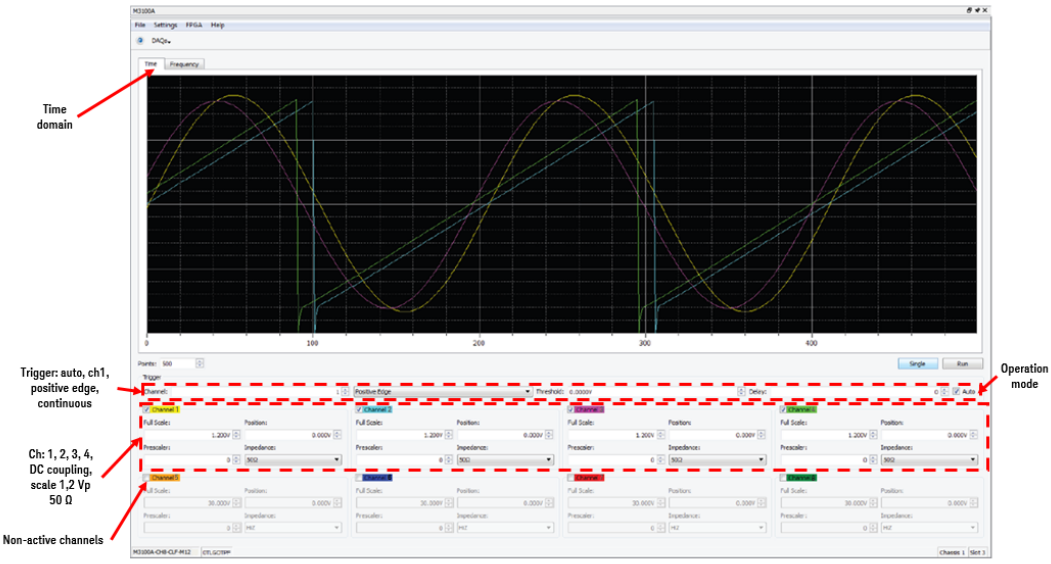


Figure7c: M31/M33XXA digitizer scope workflow.

Digitizer scope Workflow:

1. Select channels
2. Define acquisition type (single or run)
3. Set coupling and impedance

4. Define triggering:
- channel (any of the channels)
- edge (positive, negative, both)
- threshold (within fullscale settings of the channel)
- delay
- mode (normal, auto, slave)

2. 1. 1. 5 Frequency domain (Spectrum Analyzer)

Figure7d: shows the tab to configure the Spectrum Analyzer (FFT) options.

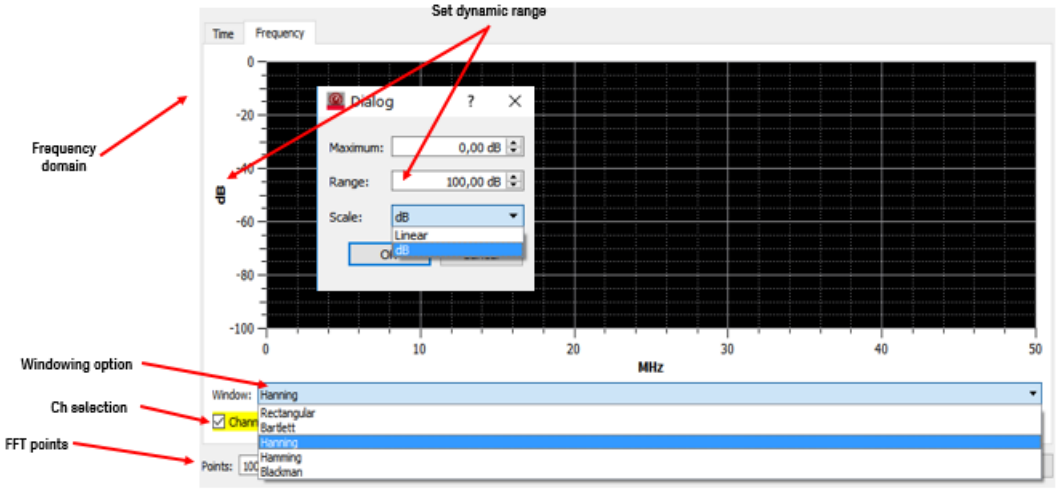


Figure7d: M31/M33XXA digitizer FFT (Spectrum analyzer) control.

Spectrum analyzer Workflow:
1. Select channel
2. Define number of FFT points (set resolution)
3. Set windowing option (see windows types below)
4. Set dynamic range:
- scale (Linear or dB)
- set max
- define range

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

Window types used in FFT function (section 2. 2. 7. 33)

Option	Description	Programming Definitions Name	Value
Rectangular	Simplest B-spine window	WINDOW_ RECTANGULAR	0 (default)
Bartlett	Hybrid window	WINDOW_BARTLETT	1
Hanning	Side-lobes roll off about 18 dB per octave	WINDOW_HANNING	2
Hamming	Optimized to minimize the maximum nearest side lobe	WINDOW_HAMMING	3
Blackman	Higher-order generalized cosine windows for applic-ations that require windowing by the convolution in the frequency-domain	WINDOW_BLACKMAN	4
Kaiser	Adjustable window maximizing energy concentration in the main lobe	WINDOW_KAISER	5
Gauss	Adjustable window (can be used for quadratic inter-polation in frequency estimation)	WINDOW_GAUSS	6

Table14a: M31/M33XXA digitizer FFT window types.

2. 2 FPGA Programming: Keysight M3602A Design Environment M3XXXA PXIe hardware with -FP1 option

2. 2. 1 FPGA Programming Overview

As described in FPGA Programming: Keysight M3602A (page 90), Keysight's FPGA design environment allows the user to customize the FPGA code of Keysight M3XXXA PXIe modules with -FP1 option enabled, providing the capability to perform custom Digital Signal Processing onboard in real time. Keysight's FPGA technology uses Keysight M3602A, an easy-to-use FPGA graphical design environment.

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

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

2. 2. 2 M3602A Diagram

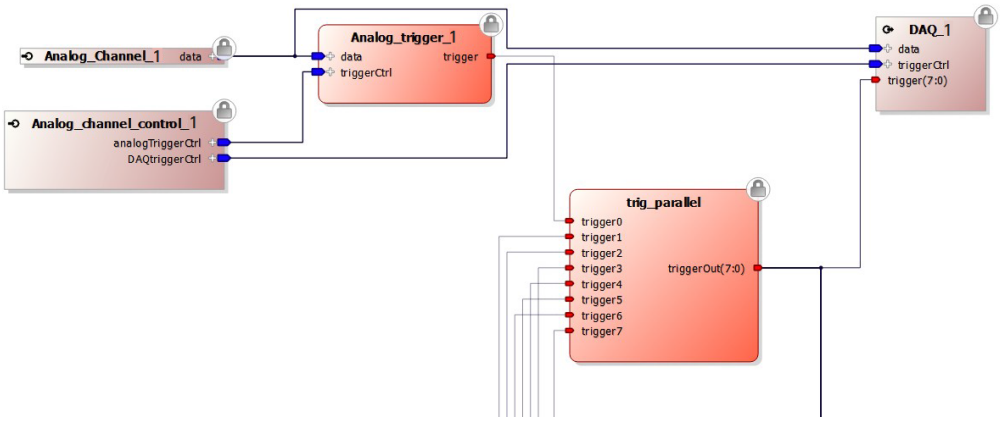


Figure 8: M3602A diagram of the M31/M33XXA Digitizers input channels (all input channels have the same structure)

2. 3 HVI Programming: Keysight M3601A

2. 3. 1 HVI Programming Overview

As described in Section HVI Technology: Keysight M3601A (page 86), Keysight's HVI technology provides the capability to create time-deterministic execution sequences that are executed by the Keysight M3XXXA PXIe hardware. HVIs are programmed with Keysight M3601A, an HVI design environment with a user-friendly flowchart-style interface.

2. 3. 2 HVI Functions

Keysight's HVI Technology uses the same programming instructions that are available in the Keysight SD1 Programming Libraries, with the difference that in a HVI those instructions are executed by the hardware modules in hard real-time, not by the computer. The programming instructions for the M31/M33XXA Digitizers are described next.

2. 4 SW Programming: Keysight SD1 Programming Libraries

2. 4. 1 SW Programming Overview

As described in Keysight SD1 Programming Libraries (page 86), Keysight provides highly-optimized programming librariesin the most common programming languages:

A. Native Languages

Ready-to-use native libraries are supplied for the following programming languages and compilers:

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

B. Other Languages

Dynamic-link libraries are compatible with any programming language that has a compiler capable of performing dynamic linking. Here are some case examples:

- Compilers not listed above.
- Other programming languages: Java, PHP, Perl, Fortran, Pascal, etc.
- Computer Algebra Systems (CAS): Wolfram Mathematica, Maplesoft Maple, etc. Dynamic-link libraries available:

Exported Functions LanguageOperating SystemFiles

CMicrosoft Windows*.dll

NOTE DLL function prototypes: The exported functions of the dynamic libraries have the same prototype as their counterparts of the static libraries

NOTE Function Parameters: Some of the parameters of the library functions are language dependent. The table of inputs and outputs parameters for each function is a conceptual description, therefore, the user must check the specific language function to see how to use it. One example are the ID parameters (moduleID, etc.), which identify objects in non-object-oriented languages. In object-oriented languages the objects are identified by their instances, and therefore the IDs are not present.

NOTE Function Names: Some programming languages like C++ or LabVIEW have a feature called function overloading or polymorphism, that allows creating several functions with the same name but with different input/output parameters. In languages without this feature, functions with different parameters must have different names.

2. 4. 2 SD_Module Functions

2. 4. 2. 1 open

This function initializes a hardware module therefore it must be called before using any other module-related function. A module can be opened using the serial number or the chassis and slot number. The first option ensures the same module is always opened regardless its chassis or slot location.

Parameters

Name	Description
Inputs	
productName	Complete product name (e.g."M3100A"). This name can be found on the product, or in nearly any Keysight software. It can also be retrieved with the function getProductName (page 30)
serialNumber	Module Serial Number (e.g."ES5641"). The serial number can be found on the product, or in nearly any Keysight software. It can also be retrieved with the function getSerialNumber (page 32)
chassis	Number where the device is located. The chassis number can be found in nearly any Keysight software. It can also be retrieved with the function getChassis (page 33)
slot	Slot number where the device is plugged in. This number can be found on the chassis, in nearly any Keysight software. It can also be retrieved with the functiongetSlot (page 34)
compatibility	Forces the channel facing numbers to be compatible with legacy models (Table 0).
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
moduleID	(Non-object-oriented languages only) Module identifier, or a negative number for Error Codes in Section 2.4.7
errorOut	See Error Codes

C

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

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

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

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

C++

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

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

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

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

Visual Studio .NET, MATLAB

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

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

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

int SD_ Module::open(string productName, int chassis, int slot, int compatibility);

Python

int SD_Module::openWithSerialNumber(string productName, string serialNumber);

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

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

int SD_ Module::openWithSlotCompatibility(string productName, int chassis, int slot, int compatibility);

LabVIEW

open.vi

M3601A

Available: No

2. 4. 2. 2 close

This function releases all the resources allocated for the module instance. It must be always called before exiting the application.

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by function open (page 27)
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be

Name	Description
	passed to errorOut

Outputs	
errorOut	Error Codes (page 81)

C

int SD_Module_close(int moduleID);

C++

int SD_Module::close();

Visual Studio .NET, MATLAB

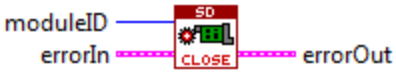
int SD_Module::close();

Python

int SD_Module::close();

LabVIEW

close.vi



M3601A

Available: No

2. 4. 2. 3 moduleCount

This function returns the number of Keysight modules installed in the system.

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

Parameters

Name	Description
Inputs	
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut

Outputs	
nModules	Number of Keysight modules installed in the system. Negative numbers for errors Error Codes (page 81)
errorOut	(LabVIEW only) Error Codes (page 81)

C

int SD_Module_moduleCount();

C++

int SD_Module::moduleCount();

Visual Studio .NET, MATLAB

int SD_Module::moduleCount();

Python

int SD_Module::moduleCount();

LabVIEW

moduleCount.vi

M3601A

Available: No

2. 4. 2. 4 getProductName

This function returns the product name of the specified device.

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

Parameters

Name	Description
Inputs	
index	Module index. It must be in the range 0..(nModules-1), where nModules is returned by function moduleCount (page 29)
chassis	Chassis number where the device is located. The chassis number can be found in nearly any Keysight software. It can also be retrieved with the function getChassis (page 33)
slot	Slot number where the device is plugged in. This number can be found on the chassis, in nearly any Keysight software. It can also be retrieved with the function getSlot (page 34)
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut

Outputs	
productName	Product name. It can be used in function open
errorOut	Error Codes (page 81)

C

int SD_Module_getProductNameByIndex(int index, char *productName);

int SD_Module_getProductNameBySlot(int chassis, int slot, char* productName);

C++

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

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

Visual Studio .NET, MATLAB

int SD_Module::getProductName(int index, string productName);

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

Python

int SD_Module::getProductNameByIndex(int index, string productName);

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

LabVIEW

getProductNameByIndex.vi

getProductNameBySlot.vi

M3601A

Available: No

2. 4. 2. 5 getSerialNumber

This function returns the serial number of the specified device.

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

Parameters

Name	Description
Inputs	
index	Module index. It must be in the range 0..(nModules-1), where nModules is returned by function moduleCount (page 29)
chassis	Chassis number where the device is located. The chassis number can be found in nearly any Keysight software. It can also be retrieved with the function getChassis (page 33)
slot	Slot number where the device is plugged in. This number can be found on the chassis, or in nearly any Keysight software. It can also be retrieved with the function getSlot (page 34)
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
serialNumber	Serial number of the product. It can be used in function open (page 27)
errorOut	Error Codes (page 81)

C

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

C++

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

Visual Studio .NET, MATLAB

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

Python

int SD_Module::getSerialNumberByIndex(int index, char *serialNumber);

int SD_Module::getSerialNumberBySlot(int chassis, int slot, char* serialNumber);

LabVIEW

getSerialNumberByIndex.vi

getSerialNumberBySlot.vi

M3601A

Available: No

2. 4. 2. 6 getChassis

This function returns the chassis number where the device is located.

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

Parameters

Name	Description
Inputs	
index	Module index. It must be in the range 0..(nModules-1), where nModules is returned by function moduleCount (page 29)
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
chassis	Chassis number where the device is located. Negative numbers for errors Error Codes (page 81)
errorOut	(LabVIEW only) Error Codes (page 81)

C

int SD_Module_getChassis(int index);

C++

int SD_Module::getChassis(int index);

Visual Studio .NET, MATLAB

int SD_Module::getChassis(int index);

Python

int SD_Module::getChassis(int index);

LabVIEW

getChassis.vi

M3601A

Available: No

2. 4. 2. 7 getSlot

This function returns the slot number where the device is located.

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

Parameters

Name	Description
Inputs	
index	Module index. It must be in the range 0..(nModules-1), where nModules is returned by function moduleCount (page 29)
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
slot	Slot number where the device is plugged in. Negative numbers for errors Error Codes (page 81)
errorOut	(LabVIEW only) Error Codes (page 81)

C

int SD_Module_getSlot(int index);

C++

int SD_Module::getSlot(int index);

Visual Studio .NET, MATLAB

int SD_Module::getSlot(int index);

Python

int SD_Module::getSlot(int index);

LabVIEW

getSlot.vi

M3601A

Available: No

2. 4. 2. 8 PXItriggerWrite

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

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by function open (page 27)
nPXItrigger	PXI trigger number
value	Digital value with negated logic, 0 (ON) or 1 (OFF)
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
moduleIDOut	(LabVIEW only) A copy of moduleID
errorOut	Error Codes (page 81)

C

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

C++

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

Visual Studio .NET, MATLAB

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

Python

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

LabVIEW

PXItriggerWrite.vi

M3601A

Available: Yes

2. 4. 2. 9 PXItriggerRead

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

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by function open (page 27)
nPXItrigger	PXI trigger number
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
moduleIDOut	(LabVIEW only) A copy of moduleID
value	Digital value with negated logic, 0 (ON) or 1 (OFF), or negative numbers for errors Error Codes (page 81)
errorOut	Error Codes (page 81)

C

int SD_Module_PXItriggerRead(int moduleID, int nPXItrigger);

C++

int SD_Module::PXItriggerRead(int nPXItrigger);

Visual Studio .NET, MATLAB

int SD_Module::PXItriggerRead(int nPXItrigger);

Python

int SD_Module::PXItriggerRead(int nPXItrigger);

LabVIEW

PXItriggerRead.vi

M3601A

Available: No (the value can be accessed using math operations: e.g. MathAssign)

2. 4. 3 SD_Module Functions (FPGA-related)

The following programming functions are related to Keysight's FPGA technology and Keysight M3602A programming environment. Please check the Keysight M3602A datasheet for more information [1].

2. 4. 3. 1 FPGAwritePCport

This function writes data at the PCport FPGA Block.

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by function open (page 27)
nPCport	PCport number
address	Address that will appear the PCport interface
data	Tx data buffer
dataSize	Number of 32-bit words to write (maximum is 128 words)
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
moduleIDout	(LabVIEW only) A copy of moduleID
errorOut	Error Codes (page 81)

C

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

C++

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

Visual Studio .NET, MATLAB

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

Python

int SD_Module::FPGAwritePCport(int nPCport, int [] data, int address, int addressMode, int accessMode);

LabVIEW

PCportWrite.vi

M3601A

Available: No

2. 4. 3. 2 FPGAreadPCport

This function reads data at the PCport FPGA Block.

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by function open (page 27)
nPCport	PCport number
address	Address that will appear the PCport interface
dataSize	Number of 32-bit words to read (maximum is 128 words)
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
data	Rx data buffer
moduleIDout	(LabVIEW only) A copy of moduleID
errorOut	Error Codes (page 81)

C

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

C++

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

Visual Studio .NET, MATLAB

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

Python

{int[], int} SD_Module::FPGAreadPCport(int nPCport, int dataSize, int address, int addressMode, int accessMode);

*Returned data array is a NumPy array

LabVIEW

FPGAwritePCport.vi

M3601A

Available: No

2. 4. 4 SD Module Functions (HVI-related)

The following programming functions are related to Keysight's HVI technology and Keysight M3601A Design Environment. Please, check M3601A User Guide for more information.

2. 4. 4. 1 writeRegister

This function writes a value in an HVI register of a hardware module.

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by function open in Section 2.4.2.1
regNumber	Register number
regName	Register name
regValue	Register value
unit	Unit of the register value
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
moduleIDout	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes

C

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

int SD_Module_writeRegisterWithName(int moduleID, const char* regName, int regValue);

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

int SD_Module_writeDoubleRegisterWithName(int moduleID, const char* regName, double regValue, const char* unit);

C++

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

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

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

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

Visual Studio .NET, MATLAB

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

Python

int SD_Module::writeRegisterByNumber(int regNumber, int regValue);
int SD_Module::writeRegisterWithName(string regName, int regValue);
int SD_Module::writeDoubleRegisterByNumber(int regNumber, double regValue, string unit);
int SD_Module::writeDoubleRegisterWithName(string regName, double regValue, string unit);

LabVIEW

writeRegister.vi

M3601A

Available: No (the value can be accessed using math operations: e.g. MathAssign)

2. 4. 4. 2 readRegister

This function reads a value from an HVI register of a hardware module.

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by function open in Section 2.4.2.1
regNumber	Register number
regName	Register name
regValue	Register value
unit	Unit of the register value
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut

Name	Description
Outputs	
regValue	Register value
moduleIDout	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes

C

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

int SD_Module_readRegisterWithName(int moduleID, const char* regName, int regValue);

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

double SD_Module_readDoubleRegisterWithName(int moduleID, const char* regName, const char* unit, int& errorOut);

C++

int SD_ Module::readRegister(int regNumber, int regValue);

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

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

double SD_ Module::readRegister(const char* regName, const char* unit, int& errorOut);

Visual Studio .NET, MATLAB

int SD_ Module::readRegister(int regNumber, int regValue);

int SD_ Module::readRegister(string regName, int regValue);

int SD_ Module::readRegister(int regNumber, string unit, out int error);

int SD_ Module::readRegister(string regName, string unit, out int error);

Python

int SD_Module::readRegisterByNumber(int regNumber, int regValue);

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

[int errorOut, double regValue] SD_Module::readDoubleRegisterByNumber(int regNumber, string unit;

[int errorOut, double regValue] SD_Module::readDoubleRegisterWithName(string regName, string unit);

LabVIEW

readRegister.vi

M3601A

Available: No (the value can be accessed using math operations: e.g. MathAssign)

2. 4. 5 SD_AIN Functions

2. 4. 5. 1 channelInputConfig

This function configures the input full scale, impedance and coupling as applicable according to the product Full Scale, Impedance and Coupling (page 4).

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by function open (page 27)
nChannel	Input channel number
fullScale	Input full scale in volts
impedance	Input impedance
coupling	Input coupling
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
moduleIDout	(LabVIEW only) A copy of moduleID
errorOut	Error Codes (page 81)

C

int SD_AIN_channelInputConfig(int moduleID, int nChannel, double fullScale, int coupling);

C++

int SD_AIN::channelInputConfig(int nChannel, double fullScale, int coupling);

Visual Studio .NET, MATLAB

int SD_AIN::channelInputConfig(int nChannel, double fullScale, int coupling);

Python

int SD_AIN::channelInputConfig(int nChannel, double fullScale, int coupling);

LabVIEW

channelInputConfig.vi

M3601A

Available: No

2. 4. 5. 2 channelPrescalerConfig

This function configures the input Theory of Operation: M31/M33XX Digitizers (page 1).

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by function open (page 27)
nChannel	Input channel number
prescaler	prescaler value).
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
moduleIDout	(LabVIEW only) A copy of moduleID
errorOut	Error Codes (page 81)

C

int SD_AIN_channelPrescalerConfig(int moduleID, int nChannel, int prescaler);

C++

int SD_AIN::channelPrescalerConfig(int nChannel, int prescaler);

Visual Studio .NET, MATLAB

int SD_AIN::channelPrescalerConfig(int nChannel, int prescaler);

Python

int SD_AIN::channelPrescalerConfig(int nChannel, int prescaler);

LabVIEW

channelPrescalerConfig.vi

M3601A

Available: Yes

2. 4. 5. 3 channelTriggerConfig

This function configures the analog trigger block for each channel Analog Trigger (page 6).

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by function open (page 27)
nChannel	Input channel number
analogTriggerMode	Trigger mode
threshold	Threshold in volts
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
moduleIDout	(LabVIEW only) A copy of moduleID
errorOut	Error Codes (page 81)

C

int SD_AIN_channelTriggerConfig(int moduleID, int nChannel, int analogTriggerMode, double threshold);

C++

int SD_AIN::channelTriggerConfig(int nChannel, int analogTriggerMode, double threshold);

Visual Studio .NET, MATLAB

int SD_AIN::channelTriggerConfig(int nChannel, int analogTriggerMode, double threshold);

Python

int SD_AIN::channelTriggerConfig(int nChannel, int analogTriggerMode, double threshold);

LabVIEW

channelTriggerConfig.vi

M3601A

Available: Yes

2. 4. 5. 4 DAQconfig

This function configures the acquisition of words Data Acquisition (DAQs) (page 8) in two possible reading modes:

- Blocking: Using the function DAQread (page 51) to read the words. DAQread is a blocking function that is released when the amount of words specified in DAQ-points is acquired or when timeout elapses. This mode is enabled when a call-back function is not specified (it is set to null).
- Non-blocking: The user specifies a callback function which is called whenever the DAQeventDataReady event is signaled or when

timeout elapses. In the latter condition, there may be words available, but less than the amount specified in DAQpoints.

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by function open (page 27)
nDAQ	DAQ to configure
triggerMode	Trigger mode
triggerDelay	(number of samples that trigger is delayed (or advanced if negative))
DAQpointsPerCycle	Number of words to acquire per trigger
cycles	Number of acquisition cycles. Each cycle requires a trigger specified by triggerMode. A negative number means continuous acquisition
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
moduleIDout	(LabVIEW only) A copy of moduleID
errorOut	Error Codes (page 81)

C

int SD_AIN_DAQconfig(int moduleID, int nDAQ, int DAQpointsPerCycle, int cycles, int triggerDelay, int triggerMode);

C++

int SD_AIN::DAQconfig(int nDAQ, int DAQpointsPerCycle, int cycles, int triggerDelay, int triggerMode;

Visual Studio .NET, MATLAB

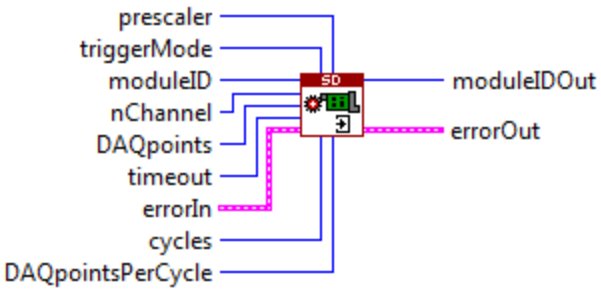
int SD_AIN::DAQconfig(int nDAQ, int DAQpointsPerCycle, int cycles, int triggerDelay, int triggerMode;

Python

int SD_AIN::DAQconfig(int nDAQ, int DAQpointsPerCycle, int cycles, int triggerDelay, int triggerMode;

LabVIEW

DAQconfig.vi



2. 4. 5. 5 DAQdigitalTriggerConfig

This function configures the digital hardware triggers for the selected DAQ Trigger (page 11).

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by function open (page 27)
nDAQ	DAQ number
triggerSource	HW digital trigger source
triggerNumber	PXI (PXI/PXIe only) trigger number or external I/O trigger number
triggerBehavior	Trigger behaviour (
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut

Name	Description
Outputs	
moduleIDOut	(LabVIEW only) A copy of moduleID
errorOut	Error Codes (page 81)

C

int SD_AIN_DAQdigitalTriggerConfig(int moduleID, int nDAQ, int triggerSource, int triggerNumber, int triggerBehaviour);

C++

int SD_AIN::DAQdigitalTriggerConfig(int nDAQ, int triggerSource, int triggerBehaviour);

Visual Studio .NET, MATLAB

int SD_AIN::DAQdigitalTriggerConfig(int nDAQ, int triggerSource, int triggerBehaviour);

Python

int SD_AIN::DAQdigitalTriggerConfig(int nDAQ, int triggerSource, int triggerBehaviour);

LabVIEW

DAQdigitalTriggerConfig.vi

M3601A

Available: Yes

2. 4. 5. 6 DAQanalogTriggerConfig

This function configures the analog hardware trigger for the selected DAQ Trigger (page 11).

NOTE **Analog DAQ: This feature is only available for Data Acquisition Blocks (DAQs) included in products with analog inputs**

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by function open (page 27)

Name	Description
nDAQ	DAQ number
triggerNumber	Analog trigger number
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut

Outputs	
moduleIDOut	(LabVIEW only) A copy of moduleID
errorOut	Error Codes (page 81)

C

int SD_AIN_DAQanalogTriggerConfig (int moduleID, int nDAQ, int triggerNumber);

C++

int SD_AIN::DAQanalogTriggerConfig (int nDAQ, int triggerNumber);

Visual Studio .NET, MATLAB

int SD_AIN::DAQanalogTriggerConfig (int nDAQ, int triggerNumber);

Python

int SD_AIN::DAQanalogTriggerConfig (int nDAQ, int triggerNumber);

LabVIEW

DAQanalogTriggerConfig .vi

M3601A

Available: No

2. 4. 5. 7 DAQread

This function reads the words acquired with the selected DAQ Data Acquisition (DAQs) (page 8). It can be used only after calling the function DAQconfig (page 48) and when a callback function is not configured. DAQread is a blocking function released when the configured amount of words is acquired, or when the configured timeout elapses (if timeout is set to ”0” , then DAQreadwaits until DAQpoints are acquired). In the timeout elapses, there may be words available, but less than the configured amount.

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by function open (page 27)
nDAQ	DAQ to be read
DAQdata	Array to be filled with acquired words
DAQpoints	Size (number of words) of DAQdata
timeout	Timeout in ms when waiting for the amount of words specified in DAQpoints. "0" means infinite
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
moduleIDout	(LabVIEW only) A copy of moduleID
DAQdata	Array with acquired words
DAQpoints	Number of acquired words
status	"1" if DAQpoints is equal to the amount of words configured with DAQconfig, "0" in case of timeout, or negative numbers for Error Codes (page 81)
errorOut	Error Codes (page 81)

C

int SD_AIN_DAQread(int moduleID, int nDAQ, short* DAQdata, int DAQpoints, int timeout);

C++

int SD_AIN::DAQread(int nDAQ, short* DAQdata, int DAQpoints, int timeout);

Visual Studio .NET, MATLAB

int SD_AIN::DAQread(int nDAQ, short[] DAQdata, int timeout);

Python

{short[], int} SD_AIN::DAQread(int nDAQ, int DAQpoints, int timeout);

*Returned data array is a NumPy array

LabVIEW

DAQread.vi

M3601A

Available: No

2. 4. 5. 8 DAQstart

This function starts acquisition on the selected DAQs Data Acquisition (DAQs) (page 8). Acquisition will start when a trigger is received.

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by function open (page 27)
nDAQ	DAQ to be started or resumed. DAQ n is connected to channel n
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
moduleIDOut	(LabVIEW only) A copy of moduleID
errorOut	Error Codes (page 81)

C

int SD_AIN_DAQstart(int moduleID, int nDAQ);

C++

int SD_AIN::DAQstart(int nDAQ);

Visual Studio .NET, MATLAB

int SD_AIN::DAQstart(int nDAQ);

Python

int SD_AIN::DAQstart(int nDAQ);

LabVIEW

DAQstart.vi

M3601A

Available: Yes

2. 4. 5. 9 DAQstartMultiple

This function starts acquisition on the selected DAQs Data Acquisition (DAQs) (page 8). Acquisition will start when a trigger is received.

Parameters

Name	Description
Inputs	
moduleID	(Non-object oriented languages only) Module identifier, returned by function open
DAQmask	Mask to select which DAQs are started or resumed (LSB is 0, bit 1 and so forth). DAQ n is connected to channel n
nDAQ	DAQ to be started or resumed. DAQ n is connected to channel n
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
moduleIDOut	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes

C

int SD_AIN_DAQstartMultiple(int moduleID, int DAQmask);

C++

int SD_AIN::DAQstartMultiple(int DAQmask);

Visual Studio .NET, MATLAB

int SD_AIN::DAQstartMultiple(int nDAQmask);

Python

int SD_AIN::DAQstartMultiple(int nDAQmask);

LabVIEW

DAQstartMultiple.vi

M3601A

Available: No (multiple DAQstart from different channels can be executed at once)

2. 4. 5. 10 DAQstop

This function stops the words acquisition Data Acquisition (DAQs) (page 8).

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by function open (page 27)
nDAQ	DAQ to be paused. DAQ n is connected to channel n
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
moduleIDOut	(LabVIEW only) A copy of moduleID
errorOut	Error Codes (page 81)

C

```
int SD_AIN_DAQstop(int moduleID, int nDAQ);
```

C++

```
int SD_AIN::DAQstop(int nDAQ);
```

Visual Studio .NET, MATLAB

```
int SD_AIN::DAQstop(int nDAQ);
```

Python

```
int SD_AIN::DAQstop(int nDAQ);
```

LabVIEW

DAQstop.vi

M3601A

Available: Yes

2. 4. 5. 11 DAQstopMultiple

This function pauses the words acquisition Data Acquisition (DAQs) (page 8).

Parameters

Name	Description
Inputs	
moduleID	(Non-object oriented languages only) Module identifier, returned by function open
DAQmask	Mask to select which DAQs are paused (LSB is 0, bit 1 and so forth). DAQ n is connected to channel n
nDAQ	DAQ is paused. DAQ n is connected to channel n
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
moduleIDOut	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes

C

```
int SD_AIN_DAQstopMultiple(int moduleID, int DAQmask);
```

C++

```
int SD_AIN::DAQstopMultiple(int DAQmask);
```

Visual Studio .NET, MATLAB

```
int SD_AIN::DAQstopMultiple(int nDAQ);
```

Python

```
int SD_AIN::DAQstopMultiple(int nDAQ);
```

LabVIEW

DAQstopMultiple.vi

M3601A

Available: No (multiple DAQstop from different channels can be executed at once)

2. 4. 5. 12 DAQpause

This function pauses the words acquisition Data Acquisition (DAQs) (page 8). Acquisition can be resumed using DAQresume.

Parameters

Name	Description
Inputs	
moduleID	(Non-object oriented languages only) Module identifier, returned by function open
nDAQ	DAQ to be stopped
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
moduleIDOut	(Non-object-oriented languages only) Module identifier, or a negative number for errors (see Error Codes)
errorOut	See Error Codes

C

```
int SD_AIN_DAQpause(int moduleID, int nDAQ);
```

C++

```
int SD_AIN::DAQpause(int nDAQ);
```

Visual Studio .NET, MATLAB

```
int SD_AIN::DAQpause(int nDAQ);
```

Python

```
int SD_AIN::DAQpause(int nDAQ);
```

LabVIEW

DAQpause.vi

M3601A

Available: Yes

2. 4. 5. 13 DAQpauseMultiple

This function pauses the words acquisition Data Acquisition (DAQs) (page 8). Acquisition can be resumed using DAQresume.

Parameters

Name	Description
Inputs	
moduleID	(Non-object oriented languages only) Module identifier, returned by function open
DAQmask	Mask to select which DAQs are paused (LSB is 0, bit 1 and so forth). DAQ n is connected to channel n
DAQ	DAQ is paused. DAQ n is connected to channel n
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
moduleIDOut	(LabVIEW only) A copy of noduleID
errorOut	See Error Codes

C

```
int SD_AIN_DAQpauseMultiple(int moduleID, int DAQmask);
```

C++

```
int SD_AIN::DAQpauseMultiple(int DAQmask);
```

Visual Studio .NET, MATLAB

```
int SD_AIN::DAQpauseMultiple(int DAQmask);
```

Python

```
int SD_AIN::DAQpauseMultiple(int DAQmask);
```

LabVIEW

DAQpauseMultiple.vi

M3601A

Available: No (multiple DAQpause from different channels can be executed at once)

2. 4. 5. 14 DAQresume

This function resumes acquisition on the selected DAQs Data Acquisition (DAQs) (page 8).

Parameters

Name	Description
Inputs	
moduleID	(Non-object oriented languages only) Module identifier, returned by function open (page 27)
nDAQ	DAQ to be resumed
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
moduleID	(Non-object-oriented languages only) Module identifier, or a negative number for errors (Error Codes (page 81))
errorOut	Error Codes (page 81)

C

```
int SD_AIN_DAQresume(int moduleID, int nDAQ);
```

C++

```
int SD_AIN::DAQresume(int nDAQ);
```

Visual Studio .NET, MATLAB

```
int SD_AIN::DAQresume(int nDAQ);
```

Python

```
int SD_AIN::DAQresume(int nDAQ);
```

LabVIEW

DAQresume.vi

M3601A

Available: Yes

2. 4. 5. 15 DAQresumeMultiple

This function resumes acquisition on the selected DAQs Data Acquisition (DAQs) (page 8)

Parameters

Name	Description
Inputs	
moduleID	(Non-object oriented languages only) Module identifier, returned by function open
DAQmask	Mask to select which DAQs are started or resumed (LSB is 0, bit 1 and so forth). DAQ n is connected to channel n
nDAQ	DAQ is started or resumed. DAQ n is connected to channel n
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
moduleIDOut	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes

C

int SD_AIN_DAQresumeMultiple(int moduleID, int DAQmask);

C++

int SD_AIN::DAQresumeMultiple(int DAQmask);

Visual Studio .NET, MATLAB

int SD_AIN::DAQresumeMultiple(int DAQmask);

Python

int SD_AIN::DAQresumeMultiple(int DAQmask);

LabVIEW

DAQresumeMultiple.vi

M3601A

Available: No (multiple DAQresume from different channels can be executed at once)

2. 4. 5. 16 DAQflush

This function flushes the acquisition buffers and resets the acquisition counter included in a Data Acquisition block (Data Acquisition (DAQs) (page 8)).

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by function open (page 27)
nDAQ	DAQ to be reset.
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
moduleIDOut	Non-object-oriented languages only) Module identifier, or a negative number for errors (see Error Codes (page 81))
errorOut	Error Codes (page 81)

C

int SD_AIN_DAQflush(int moduleID, int nDAQ);

C++

int SD_AIN::DAQflush(int nDAQ);

Visual Studio .NET, MATLAB

int SD_AIN::DAQflush(int nDAQ);

Python

int SD_AIN::DAQflush(int nDAQ);

LabVIEW

DAQflush.vi

M3601A

Available: Yes

2. 4. 5. 17 DAQflushMultiple

This function flushes the acquisition buffers and resets the acquisition counter included in a Data Acquisition block (Data Acquisition (DAQs) (page 8).

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by function open
DAQmask	Mask to select which DAQs are reset (LSB is DAQ 0, bit 1 is DAQ 1 and so forth). DAQ n is connected to channel n
DAQ	DAQ to be reset. DAQ n is connected to channel n
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
moduleIDOut	(LabVIEW only) A copy of moduleID
errorOut	Error Codes (page 81)

C

int SD_AIN_DAQflushMultiple(int moduleID, int DAQmask);

C++

int SD_AIN::DAQflushMultiple(int DAQmask);

Visual Studio .NET, MATLAB

int SD_AIN::DAQflushMultiple(int DAQmask);

Python

int SD_AIN::DAQflushMultiple(int DAQmask);

LabVIEW

DAQflushMultiple.vi

M3601A

Available: No (multiple DAQflush from different channels can be executed at once)

2. 4. 5. 18 DAQtrigger

This function triggers the acquisition of words in the selected DAQs (Data Acquisition (DAQs) (page 8)) provided that they are configured for VI/HVI Trigger.

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by function open (page 27)
nDAQ	DAQ to be triggered. DAQ n is connected to channel n
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
moduleID	(Non-object-oriented languages only) Module identifier, or a negative number for errors (see Error Codes (page 81)
errorOut	Error Codes (page 81)

C

int SD_AIN_DAQtrigger(int moduleID, int nDAQ);

C++

int SD_AIN::DAQtrigger(int nDAQ);

Visual Studio .NET, MATLAB

int SD_AIN::DAQtrigger(int nDAQ);

Python

int SD_AIN::DAQtrigger(int nDAQ);

LabVIEW

DAQtrigger.vi

M3601A

Available: Yes

2. 4. 5. 19 DAQtriggerMultiple

This function triggers the acquisition of words in the selected DAQs Data Acquisition (DAQs) (page 8) provided that they are configured for VI/HVI Trigger.

Parameters

Name	Description
Inputs	
moduleID	(Non-object oriented languages only) Module identifier, returned by function open
DAQmask	Mask to select which DAQs are triggered (LSB is 0, bit 1 and so forth). DAQ n is connected to channel n
DAQ	DAQ to be triggered. DAQ n is connected to channel n
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
moduleIDOut	(LabVIEW only) A copy of moduleID
errorOut	See Error Codes

C

int SD_AIN_DAQtriggerMultiple(int moduleID, int DAQmask);

C++

int SD_AIN::DAQtriggerMultiple(int DAQmask);

Visual Studio .NET, MATLAB

int SD_AIN::DAQtriggerMultiple(int DAQmask);

Python

int SD_AIN::DAQtriggerMultiple(int DAQmask);

LabVIEW

DAQtriggerMultiple.vi

M3601A

Available: No (multiple DAQtrigger from different channels can be executed at once)

2. 4. 5. 20 DAQcounterRead

This function reads the number of words acquired by the selected DAQ (Data Acquisition (DAQs) (page 8)) since the last call to DAQflush or DAQ.

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by function open (page 27)
DAQ	DAQ whose counter is read
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
moduleIDOut	(LabVIEW only) A copy of moduleID
counter	Value of the DAQ counter or a negative number for errors Error Codes (page 81)
errorOut	Error Codes (page 81)

C

```
int SD_AIN_DAQcounterRead(int moduleID, int DAQ);
```

C++

```
int SD_AIN::DAQcounterRead(int DAQ);
```

Visual Studio .NET, MATLAB

```
int SD_AIN::DAQcounterRead(int DAQ);
```

Python

```
int SD_AIN::DAQcounterRead(int DAQ);
```

LabVIEW

DAQcounterRead.vi

M3601A

Available: No

2. 4. 5. 21 triggerIOconfig

This function configures the trigger connector/line direction and synchronization/sampling method (I/O Triggers (page 15)).

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by function open (page 27)
direction	Input or output
syncMode	Sampling/synchronization mode
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
moduleIDout	(LabVIEW only) A copy of moduleID
errorOut	Error Codes (page 81)

C

```
int SD_AIN_triggerIOconfig(int moduleID, int direction, int syncMode);
```

C++

```
int SD_AIN::triggerIOconfig(int direction, int syncMode);
```

Visual Studio .NET, MATLAB

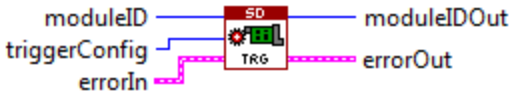
```
int SD_AIN::triggerIOconfig(int direction, int syncMode);
```

Python

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

LabVIEW

triggerIOconfig.vi



M3601A

Available: No

2. 4. 5. 22 triggerIOwrite

This function sets the trigger output. The trigger must be configured as output using function triggerIOconfig (page 66)) and I/O Triggers (page 15).

Parameters

Name	Description
	Inputs
moduleID	(Non-object-oriented languages only) Module identifier, returned by function open (page 27)
value	Trigger output value: 0 (OFF), 1 (ON)
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
moduleIDout	(LabVIEW only) A copy of moduleID
errorOut	Error Codes (page 81)

C

```
int SD_AIN_triggerIOwrite(int moduleID, int value);
```

C++

```
int SD_AIN::triggerIOwrite(int value);
```

Visual Studio .NET, MATLAB

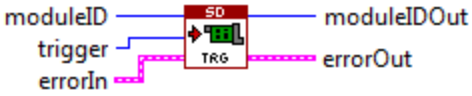
```
int SD_AIN::triggerIOwrite(int value);
```

Python

int SD_AIN::triggerIOwrite(int value);

LabVIEW

triggerIOwrite.vi



M3601A

Available: Yes

2. 4. 5. 23 triggerIOread

This function reads the trigger input (I/O Triggers (page 15))

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by functionopen (page 27)
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
moduleIDout	(LabVIEW only) A copy of moduleID
value	Trigger input value: 0 (OFF), 1 (ON). Negative numbers forError Codes (page 81)
errorOut	Error Codes (page 81)

C

int SD_AIN_triggerIOread(int moduleID);

C++

int SD_AIN::triggerIOread();

Visual Studio .NET, MATLAB

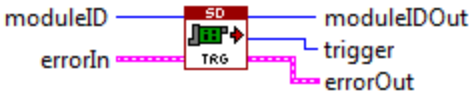
int SD_AIN::triggerIOread();

Python

int SD_AIN::triggerIOread();

LabVIEW

triggerIOread.vi



M3601A

Available: No (can be acceses usingmat operations)

2. 4. 5. 24 clockSetFrequency

This function sets the module clock frequency, see FlexCLK Technology (models with variable sampling rate) (page 16).

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by function open (page 27)
frequency	Frequency in Hz
mode	Operation mode of the variable Clock System (page 16)
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
moduleIDout	(LabVIEW only) A copy of moduleID
CLKsysFreq*	It returns the real frequency applied to the hardware in Hz. It may differ from the desired frequency due to the hardware frequency resolution. Negative numbers for Error Codes (page 81)
errorOut(LabVIEW only)	Error Codes (page 81)

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

C

double SD_AIN_clockSetFrequency(int moduleID, double frequency, int mode);

C++

double SD_AIN::clockSetFrequency(double frequency, int mode);

Visual Studio .NET, MATLAB

double SD_AIN::clockSetFrequency(double frequency, int mode);

Python

double SD_AIN::clockSetFrequency(double frequency, int mode);

LabVIEW

clockSetFrequency.vi

M3601A

Available: No

2. 4. 5. 25 clockGetFrequency

This function returns the real hardware clock frequency (Clock System (page 16)). It may differ from the frequency set with the function clockSetFrequency (page 69), due to the hardware frequency resolution.

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by function open (page 27)
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	

Name	Description
moduleIDout	(LabVIEW only) A copy of moduleID
CLKsysFreq	It returns the real hardware clock frequency in Hz (CLKsys). Negative numbers for Error Codes (page 81)
errorOut	(LabVIEW only) Error Codes (page 81)

C

double SD_AIN_clockGetFrequency(int moduleID);

C++

double SD_AIN::clockGetFrequency();

Visual Studio .NET, MATLAB

double SD_AIN::clockGetFrequency();

Python

double SD_AIN::clockGetFrequency();

LabVIEW

clockGetFrequency.vi

M3601A

Available: No

2. 4. 5. 26 clockGetSyncFrequency

This function returns the frequency of Clock System (page 16)

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by function open (page 27)
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut

Name	Description
Outputs	
moduleIDout	(LabVIEW only) A copy of moduleID
CLKsyncFreq	It returns the frequency of the internal CLKsync in Hz (Equation 2 on page 12). Negative numbers for Error Codes (page 81)
errorOut	(LabVIEW only) Error Codes (page 81)

C

int SD_AIN_clockGetSyncFrequency(int moduleID);

C++

int SD_AIN::clockGetSyncFrequency();

Visual Studio .NET, MATLAB

int SD_AIN::clockGetSyncFrequency();

Python

int SD_AIN::clockGetSyncFrequency();

LabVIEW

clockGetSyncFrequency.vi

M3601A

Available: No

2. 4. 5. 27 clockResetPhase

This function sets the module in a sync state, waiting for the first trigger to reset the phase of the internal clocks CLKsync and CLKsys (see Clock System (page 16)).

Parameters

Name	Description
Inputs	
moduleID	(Non-object-oriented languages only) Module identifier, returned by function open (page 27)
triggerBehavior	Trigger behaviour

Name	Description
PXItrigger	PXI trigger number
skew	Skew between PXI CLK10 and CLKsync in multiples of 10 ns
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut

Outputs	
moduleIDout	(LabVIEW only) A copy of moduleID
errorOut	Error Codes (page 81)

C

int SD_AIN_clockResetPhase(int moduleID, int triggerBehavior, int PXItrigger, double skew);

C++

int SD_AIN::clockResetPhase(int triggerBehavior, int PXItrigger, double skew);

Visual Studio .NET, MATLAB

int SD_AIN::clockResetPhase(int triggerBehavior, int PXItrigger, double skew);

Python

int SD_AIN::clockResetPhase(int triggerBehavior, int PXItrigger, double skew);

LabVIEW

clockResetPhase.vi

M3601A

Available: No

2. 4. 5. 28 DAQbufferPoolConfig

This function configures buffer pool that will be filled with the data of the channel to be transferred to PC.

Parameters

Name	Description

Name	Description
Inputs	
moduleID	(Non-object oriented languages only) Module identifier, returned by function open (page 27)
nDAQ	DAQ to configure
dataBuffer	Buffer to use in buffer pool. Has to be created and released by user
nPoints	Size of dataBuffer buffer
timeOut	Maximum time used to fill each buffer. If 0, timeout is not set and buffer will not be delivered to the user until it's full
callbackFunction	Callback that will be called each time a buffer is ready. It can be null. If callback is set, DAQbufferGet is useless
callbackUserObj	Pointer to user object that will be passed in the callback as parameter called user-Object. It can be null
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut

Outputs	
moduleID	(Non-object-oriented languages only) Module identifier, or a negative number for errors (see Error Codes (page 81)
errorOut	See Error Codes (page 81)

C

int SD_AIN_DAQbufferPoolConfig(int moduleID, int nDAQ, short* dataBuffer, int nPoints, int timeOut, callbackEventPtr callbackFunction, void *callbackUserObj);

C++

int SD_AIN::DAQbufferPoolConfig(int nDAQ, short* dataBuffer, int nPoints, int timeOut, callbackEventPtr callbackFunction, void *callbackUserObj);

Visual Studio .NET, MATLAB

int SD_AIN::DAQbufferPoolConfig(int nDAQ, short[] dataBuffer);

Python

int SD_AIN::DAQbufferPoolConfig(int nDAQ, int nPoints, int timeOut);

LabVIEW

DAQbuffer functions are not accessible

M3601A

Available: No

2. 4. 5. 29 DAQbufferAdd

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

Parameters

Name	Description
Inputs	
moduleID	(Non-object oriented languages only) Module identifier, returned by function open
nDAQ	DAQ to configure
dataBuffer	Buffer to use in buffer pool. Has to be created and released by user
nPoints	Size of dataBuffer buffer
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
moduleID	(Non-object-oriented languages only) Module identifier, or a negative number for errors (see Error Codes)
errorOut	See Error Codes

C

int SD_AIN_DAQbufferAdd(int moduleID, int nDAQ, short* dataBuffer, int nPoints);

C++

int SD_AIN::DAQbufferAdd(int nDAQ, short *dataBuffer, int nPoints);

Visual Studio .NET, MATLAB

int SD_AIN::DAQbufferAdd(int nDAQ, short[] dataBuffer);

Python

Not available

LabVIEW

DAQbuffer functions are not accessible

M3601A

Available: No

2. 4. 5. 30 DAQbufferGet

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

Parameters

Name	Description
Inputs	
moduleID	(Non-object oriented languages only) Module identifier, returned by function open
nDAQ	DAQ from where get the buffer
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
moduleID	(Non-object-oriented languages only) Module identifier, or a negative number for errors (see Error Codes)
errorOut	See Error Codes
buffer	Buffer obtained
readPointsOut	Number of points of the returned buffer

C

short* SD_AIN_DAQbufferGet(int moduleID, int nDAQ, int &readPointsOut, int &errorOut);

C++

short* SD_AIN:: DAQbufferGet(int nDAQ, int &readPointsOut, int &errorOut);

Visual Studio .NET, MATLAB

short[] SD_AIN::DAQbufferGet(int nDAQ, out int readPointsOut, out int errorOut);

Python

[short[], int} SD_AIN::DAQbufferGet(int nDAQ);

*Returned data array is a NumPY array

LabVIEW

DAQbuffer functions are not accessible

M3601A

Available: No

2. 4. 5. 31 DAQbufferPoolRelease

Releases the channel buffer pool and its resources. After this call, user has to call DAQbufferRemove consecutively to get all buffers back and release them.

Parameters

Name	Description
Inputs	
moduleID	(Non-object oriented languages only) Module identifier, returned by function open
nDAQ	DAQ from where use take out the buffer
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
moduleID	(Non-object-oriented languages only) Module identifier, or a negative number for errors (see Error Codes
errorOut	See Error Codes

C

int SD_AIN_DAQbufferRelease(int moduleID, int nDAQ);

C++

int SD_AIN::DAQbufferPoolRelease(int nDAQ);

Visual Studio .NET, MATLAB

int SD_AIN::DAQbufferPoolRelease(int nDAQ);

Python

SD_AIN::DAQbufferDAQbufferPoolRelease(int nDAQ);

LabVIEW

DAQbuffer functions are not accessible

M3601A

Available: No

2. 4. 5. 32 DAQbufferRemove

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

Parameters

Name	Description
Inputs	
moduleID	(Non-object oriented languages only) Module identifier, returned by function open
nDAQ	DAQ from where use take out the buffer
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
moduleID	(Non-object-oriented languages only) Module identifier, or a negative number for errors (see Error Codes (page 81)
buffer	Buffer obtained

C

short* SD_AIN_DAQbufferRemove(int moduleID, int nDAQ);

C++

short* SD_AIN::DAQbufferRemove(int nDAQ);

Visual Studio .NET, MATLAB

short[] SD_AIN::DAQbufferRemove(int nDAQ);

Python

Not available

LabVIEW

DAQbuffer functions are not accessible

M3601A

Available: No

2. 4. 5. 33 FFT

Calculates the FFT of data captured by DAQread for the selected channel.

Parameters

Name	Description
Inputs	
moduleID	(Non-object oriented languages only) Module identifier, returned by function open (page 27)
channel	Input channel number
data	data previously acquired by DAQread
size	Input data size
resultSize	Size of the output buffers (module and phase)
dB	Scale (dB or linear)
windowType	Windowing option (section 2.1.5)
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
moduleID	(Non-object-oriented languages only) Module identifier, or a negative number for errors (see Error Codes (page 70)
result	Module (magnitude) of the FFT
resultPhase	Phase of the FFT
errorOut	Error Codes

C

int SD_AIN_FFT(int moduleID, int channel, short *data, int size, double *result, int resultSize, double *resultPhase, bool dB, int windowType);

C++

int SD_AIN::FFT(int channel, short *data, int size, double *result, int resultSize, double *resultPhase, bool dB, int windowType);

Visual Studio .NET, MATLAB

int SD_AIN::FFT(int channel, short[] data, out double[] result, bool dB, int windowType);

int SD_AIN::FFT(int channel, short[] data, out double[] result, out double[] resultPhase, bool dB, int windowType);

Python

{double[][], int} SD_AIN::FFT(int channel, short[] data, bool dB, int windowType);

*Returned data array is a NumPy array

LabVIEW

Not available

M3601A

Available: No

2. 4. 6 Error Codes

Error Define	Error No	Error Description
SD_ERROR_OPENING_MODULE	-8000	Keysight Error: Opening module
SD_ERROR_CLOSING_MODULE	-8001	Keysight Error: Closing module
SD_ERROR_OPENING_HVI	-8002	Keysight Error: Opening HVI
SD_ERROR_CLOSING_HVI	-8003	Keysight Error: Closing HVI
SD_ERROR_MODULE_NOT_OPENED	-8004	Keysight Error: Module not opened
SD_ERROR_MODULE_NOT_OPENED_BY_USER	-8005	Keysight Error: Module not opened by user
SD_ERROR_MODULE_ALREADY_OPENED	-8006	Keysight Error: Module already opened
SD_ERROR_HVI_NOT_OPENED	-8007	Keysight Error: HVI not opened
SD_ERROR_INVALID_OBJECTID	-8008	Keysight Error: Invalid ObjectID
SD_ERROR_INVALID_MODULEID	-8009	Keysight Error: Invalid ModuleID
SD_ERROR_INVALID_MODULEUSERNAME	-8010	Keysight Error: Invalid Module User Name
SD_ERROR_INVALID_HVIID	-8011	Keysight Error: Invalid HVI
SD_ERROR_INVALID_OBJECT	-8012	Error: Invalid Object
SD_ERROR_INVALID_NCHANNEL	-8013	Keysight Error: Invalid channel number
SD_ERROR_BUS_DOES_NOT_EXIST	-8014	Keysight Error: Bus doesn't exist
SD_ERROR_BITMAP_ASSIGNED_DOES_NOT_EXIST	-8015	Keysight Error: Any input assigned to the bitMap does not exist
SD_ERROR_BUS_INVALID_SIZE	-8016	Keysight Error: Input size does not fit on this bus
SD_ERROR_BUS_INVALID_DATA	-8017	Keysight Error: Input data does not fit on this bus
SD_ERROR_INVALID_VALUE	-8018	Keysight Error: Invalid value
SD_ERROR_CREATING_WAVE	-8019	Keysight Error: Creating Waveform
SD_ERRO_NOT_VALID_PARAMETERS	-8020	Keysight Error: Invalid Parameters
SD_ERROR_AWG	-8021	Keysight Error: AWG function failed
SD_ERROR_DAQ_INVALID_FUNCTIONALITY	-8022	Keysight Error: Invalid DAQ functionality
SD_ERROR_DAQ_POOL_ALREADY_RUNNING	-8023	Keysight Error: DAQ buffer pool is already running
SD_ERROR_UNKNOWN	-8024	Keysight Error: Unknown error
SD_ERROR_INVALID_PARAMETERS	-8025	Keysight Error: Invalid parameter
SD_ERROR_MODULE_NOT_FOUND	-8026	Keysight Error: Module not found
SD_ERROR_DRIVER_RESOURCE_BUSY	-8027	Keysight Error: Driver resource busy
SD_ERROR_DRIVER_RESOURCE_NOT_READY	-8028	Keysight Error: Driver resource not ready
SD_ERROR_DRIVER_ALLOCATE_BUFFER	-8029	Keysight Error: Cannot allocate buffer in driver
SD_ERROR_ALLOCATE_BUFFER	-8030	Keysight Error: Cannot allocate buffer
SD_ERROR_RESOURCE_NOT_READY	-8031	Keysight Error: Resource not ready
SD_ERROR_HARDWARE	-8032	Keysight Error: Hardware error

Error Define	Error No	Error Description
SD_ERROR_INVALID_OPERATION	-8033	Keysight Error: Invalid Operation
SD_ERROR_NO_COMPILED_CODE	-8034	Keysight Error: No compiled code in the module
SD_ERROR_FW_VERIFICATION	-8035	Keysight Error: Firmware verification failed
SD_ERROR_COMPATIBILITY	-8036	Keysight Error: Compatibility error
SD_ERROR_INVALID_TYPE	-8037	Keysight Error: Invalid type
SD_ERROR_DEMO_MODULE	-8038	Keysight Error: Demo module
SD_ERROR_INVALID_BUFFER	-8039	Keysight Error: Invalid buffer
SD_ERROR_INVALID_INDEX	-8040	Keysight Error: Invalid index
SD_ERROR_INVALID_NHISTOGRAM	-8041	Keysight Error: Invalid histogram number
SD_ERROR_INVALID_NBINS	-8042	Keysight Error: Invalid number of bins
SD_ERROR_INVALID_MASK	-8043	Keysight Error: Invalid mask
SD_ERROR_INVALID_WAVEFORM	-8044	Keysight Error: Invalid waveform
SD_ERROR_INVALID_STROBE	-8045	Keysight Error: Invalid strobe
SD_ERROR_INVALID_STROBE_VALUE	-8046	Keysight Error: Invalid strobe value
SD_ERROR_INVALID_DEBOUNCING	-8047	Keysight Error: Invalid debouncing
SD_ERROR_INVALID_PRESCALER	-8048	Keysight Error: Invalid prescaler
SD_ERROR_INVALID_PORT	-8049	Keysight Error: Invalid port
SD_ERROR_INVALID_DIRECTION	-8050	Keysight Error: Invalid direction
SD_ERROR_INVALID_MODE	-8051	Keysight Error: Invalid mode
SD_ERROR_INVALID_FREQUENCY	-8052	Keysight Error: Invalid frequency
SD_ERROR_INVALID_IMPEDANCE	-8053	Keysight Error: Invalid impedance
SD_ERROR_INVALID_GAIN	-8054	Keysight Error: Invalid gain
SD_ERROR_INVALID_FULLSCALE	-8055	Keysight Error: Invalid fullscale
SD_ERROR_INVALID_FILE	-8056	Keysight Error: Invalid file
SD_ERROR_INVALID_SLOT	-8057	Keysight Error: Invalid slot
SD_ERROR_INVALID_NAME	-8058	Keysight Error: Invalid product name
SD_ERROR_INVALID_SERIAL	-8059	Keysight Error: Invalid serial number
SD_ERROR_INVALID_START	-8060	Keysight Error: Invalid start
SD_ERROR_INVALID_END	-8061	Keysight Error: Invalid end
SD_ERROR_INVALID_CYCLES	-8062	Keysight Error: Invalid number of cycles
SD_ERROR_HVI_INVALID_NUMBER_ MODULES	-8063	Keysight Error: Invalid number of modules on HVI
SD_ERROR_DAQ_P2P_ALREADY_RUNNING	-8064	Keysight Error: DAQ P2P is already running
SD_Error.OPEN_DRAIN_NOT_SUPPORTED	-8065	Open drain not supported
SD_Error.CHASSIS_PORTS_NOT_ SUPPORTED	-8066	Chassis port not supported
SD_Error.CHASSIS_SETUP_NOT_ SUPPORTED	-8067	Chassis setup not supported
SD_Error.OPEN_DRAIN_FAILED	-8068	Open drain failed
SD_Error.CHASSIS_SETUP_FAILED	-8069	Chassis setup failed
SD_Error.INVALID_PART	-8070	Invalid part
SD_Error.INVALID_SIZE	-8071	Invalid size

Error Define	Error No	Error Description
SD_Error.INVALID_HANDLE	-8072	Invalid handle

Table 15: Software error codes

3 Addendum: Keysight Technology and Software Overview

3. 1 Programming Tools

The diagram shown in Figure 9 summarizes the programming tools available to control any Keysight Hardware.

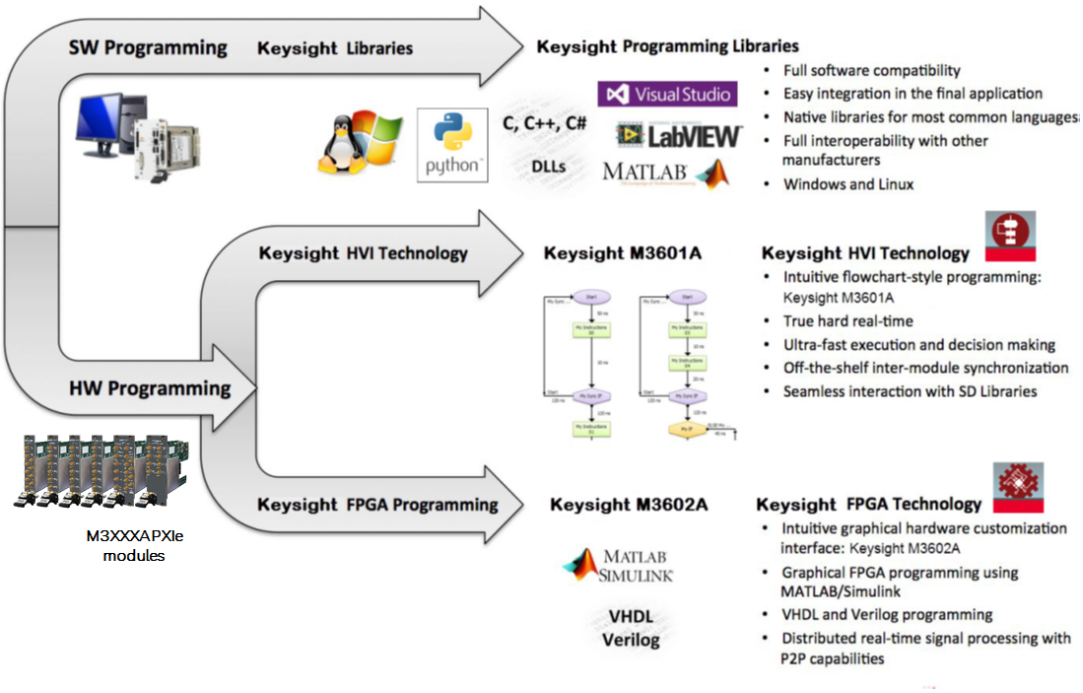


Figure 9: Programming tools for all Keysight hardware

3. 1. 1 SW Programming

A comprehensive set of highly optimized software instructions controls the off-the-shelf functionalities of the compatible Keysight hardware. These instructions are compiled into the Programming Libraries. The use of customizable software to create user-defined control, test and measurement systems is commonly referred as Virtual Instrumentation. In all Keysight documentation, the concept of a Virtual Instrument (or VI) describes user software that uses programming libraries and is executed by a computer.

3. 1. 1. 1 Keysight SD1 Programming Libraries

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

3. 1. 2 HW Programming

3. 1. 2. 1 HVI Technology: Keysight M3601A

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

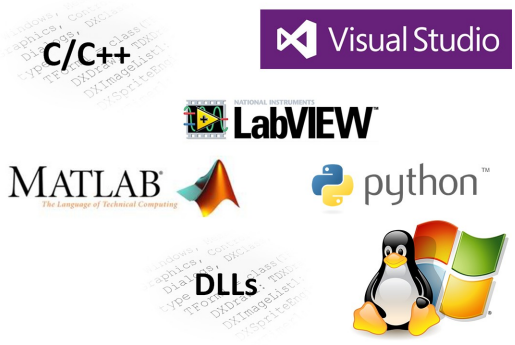


Figure 10: Keysight native programming libraries ensure full compatibility, providing effortless and seamless software integration and user interaction, etc. The I/O modules run in parallel, completely synchronized, and exchange data and decisions in real-time. The result is a set of modules that behave like a single integrated real-time instrument.

NOTE **HVIs vs VIs: Virtual Instrumentation is fully supported making use of the Keysight SD1 Programming Libraries. On the other hand,**

NOTE **Keysight's exclusive Hard Virtual Instrumentation (HVI) technology provides the capability to create time-deterministic execution sequences which are executed by the hardware modules in parallel and with perfect intermodule synchronization. HVIs provide the same programming instructions available in the Keysight SD1 Programming Libraries.**

HVIs are programmed with Keysight M3601A, an HVI design environment with a user-friendly flowchart-style interface, compatible with all M3XXXA Keysight PXIe hardware modules.

M3601A

Figure 11: Keysight M3601A, a user-friendly flowchart-style HVI programming environment

Keysight's Hard Virtual Instrumentation technology provides:

- Ultra-fast hard real time execution, processing and decision making: Execution is hardware-timed and can be as fast as 1 nanosecond, matching very high-performance FPGA-based systems and outperforming any real-time operating system.

- User-friendly flowchart-style programming interface: Keysight M3601A provides an intuitive flowchart-style program- ming environment that makes HVI programming extremely fast and easy (Figure 12 on the facing page). Using M3601A and its set of built-in instructions (the same instructions available for VIs), the user can program the hardware modules without any knowledge in FPGA technology, VHDL, etc.
- Off-the-shelf intermodule synchronization and data exchange: Each HVI is defined by a group of hardware modules which work perfectly synchronized, without the need of any external trigger or additional external hardware (Figure 13 on page 52). In addition, Keysight modules exchange data and decisions for ultra-fast control algorithms.
- Complete robustness: Execution is performed by hardware, without operating system, and independently of the user PC.
- Seamless integration with Keysight FPGA technology FPGA Programming: Keysight M3602A Design Environment M3XXXA PXIe hardware with -FP1 option (page 23):HVIs can interact with user-defined FPGA functions, making the real-time processing capabilities of HVIs unlimited.
- Seamless integration with Keysight SD1 Programming Libraries: In a complex control or test system, there are still some non- time-critical tasks that can only be performed by a VI, like for example: user interaction, visualization, or processing and decisions tasks which are too complex to be implemented by hardware. Therefore, in a real application, the combination of VIs and HVIs is required. This task can be performed seamlessly with the Keysight SD1 programming tools, e.g. the user can have many HVIs and can control them from a VI using instructions like start, stop, pause, etc.

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

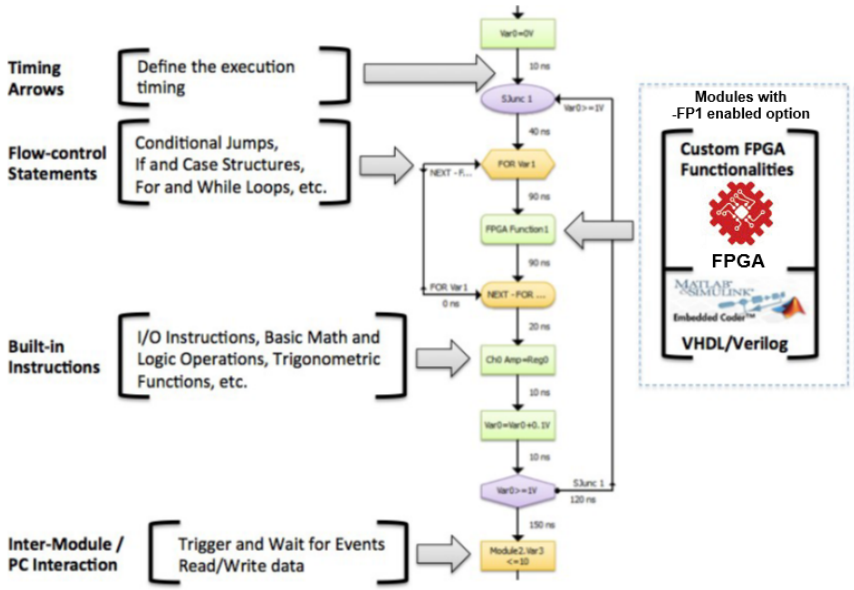


Figure 12: HVI flowchart elements. Keysight M3601A is based on flowchart programming, providing an easy-to-use environment to develop hard real-time applications

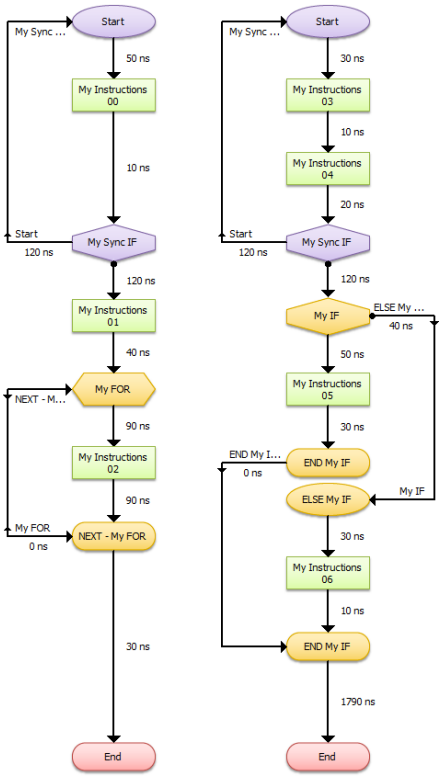


Figure 13: HVI example with two hardware modules. In an HVI, all Keysight modules run in parallel and completely synchronized, executing one flowchart per module. This results in simpler systems without the need of triggers.

3. 1. 2. 2 FPGA Programming: Keysight M3602A

Some applications require the use of custom onboard real-time processing which might not be covered by the comprehensive off-the- shelf functionalities of the standard hardware products. For these applications, Keysight M3XXXA PXIe models are supplied with -FP1 option, hardware products that provide the capability to program the onboard FPGA.

The Keysight M3100A, M3102A, M3201A, M3202A, M3300A and M3302A PXIe modular hardware family of products offers an optional -FP1 Enabled FPGA Programming capability with -K32 or -K41 logic. This capability provides the same built-in functionalities of their standard counterparts, giving the users more time to focus on their specific functionalities. For example, using the -FP1 enabled FPGA Programming with -K32 or -K41 logic version of a Keysight digitizer, the user has all the off-the-shelf functionalities of the hardware (data capture, triggering, etc.), but custom real-time FPGA processing can be added in the data path, between the acquisition and the transmission of data to the computer.

NOTE **Keysight FPGA-programmable Hardware: Keysight FPGA technology is available for M3XXXA hardware product with -FP1 option enabled providing the same built-in functionalities of their standard counterparts.**

Keysight FPGA programming technology is managed with Keysight M3602A [1], an intuitive graphical FPGA programming environment.

environment.

M3602A

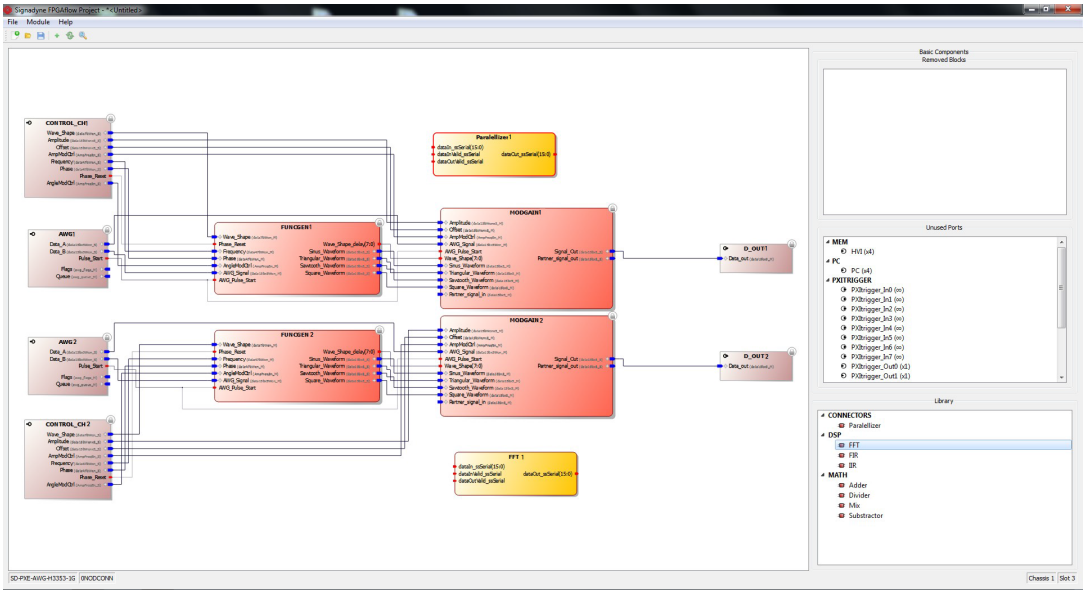


Figure 14: Keysight M3602A provides an intuitive graphical FPGA customization interface

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

Keysight M3602A: An FPGA Design Environment

Keysight M3602A is a complete FPGA design environment that allows the user to customize M3XXXA PXIe hardware products. M3602A provides the necessary tools to design, compile and program the FPGA of the module (Figure 15).

Keysight M3602A provides the following features:

- User-friendly graphical FPGA programming environment
- Complete platform, from design to FPGA programming: Keysight M3602A provides the necessary tools to design, compile and program the FPGA of the module (Figure 15)
- 5x faster project development
- Graphical environment without performance penalty
- FPGA know-how requirement minimized: The graphical environment provides a tool which does not require an extensive know how in FPGA technology, improving drastically the learning curve.

Streamlined design process

- Ready-to-use Keysight Block Library: M3602A provides a continuously-growing library of blocks which reduces the need for custom FPGA-code development.
- Include VHDL, Verilog, or Xilinx VIVADO/ISE projects: Experienced FPGA users can squeeze the power of the onboard FPGA.
- Include MATLAB/Simulink projects: MATLAB/Simulink in conjunction with Xilinx System Generator for DSP provides a powerful tool to implement Digital Signal Processing. The user can go from the design/simulation power of MATLAB/Simulink to M3602A code in just a few clicks.
- Include Xilinx CORE Generator IP cores: Xilinx CORE Generator can be launched by M3602A to create IP cores that can be seamlessly included in the design.
- Add and remove built-in resources to free up space: The user can remove unused built-in resources to free up more FPGA space.

One-click compiling and programming:

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

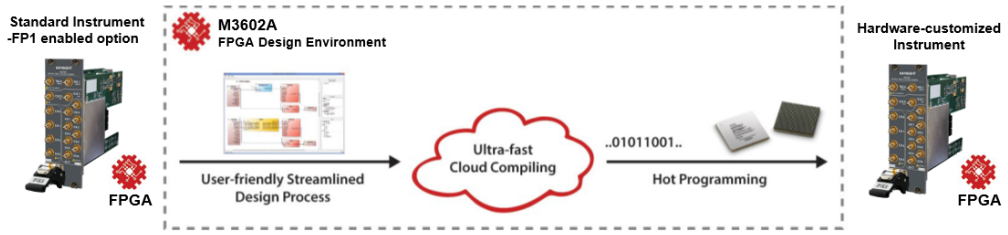


Figure 15: Keysight M3602A: a platform that provides the complete flow from design to FPGA programming

3. 2 Design Process: Customization vs. Complete Design

The M3602A FPGA Design Environment simplifies the development of custom processing functions for the following -FP1 enabled FPGA programming PXIe modules: M3100A, M3102A, M3201A, M3202A, M3300A, M3302A. These products are delivered with all the off-the-shelf functionalities of the standard products, and therefore the development time is dramatically

reduced. The user can focus exclusively on expanding the functionality of the standard instrument, instead of developing a complete new one.

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

3. 3 Application Software

3. 3. 1 Keysight SD1 SFP

All Keysight modules can be operated as classical workbench instruments using Keysight SD1 SFP [4], a ready-to-use soft- ware front panels for live operation. When SD1 SFP opens, it identifies all Keysight hardware connected to the computer, opening the corresponding front panels.

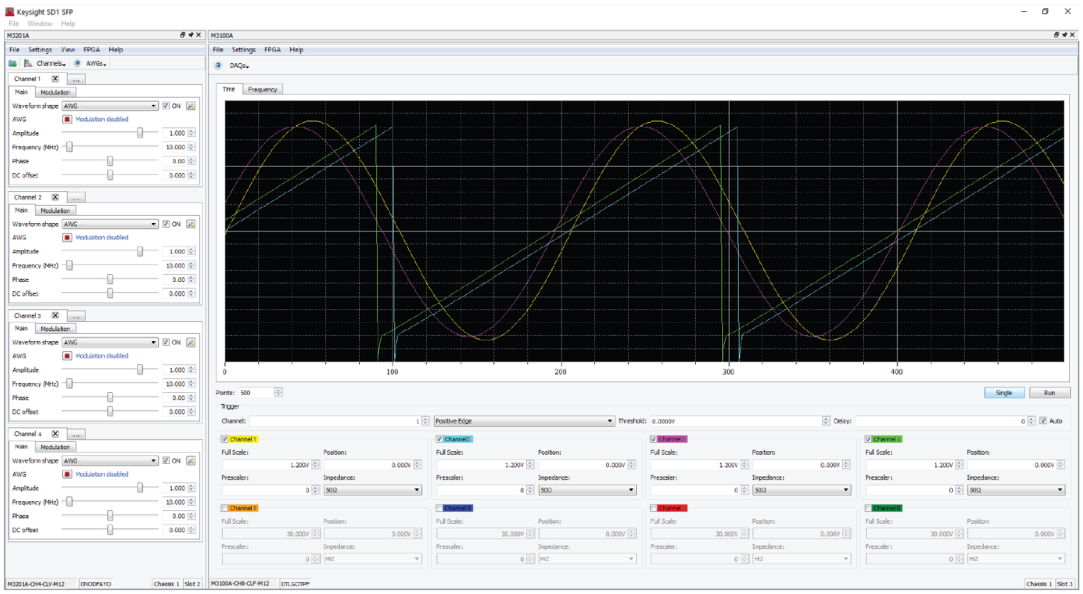


Figure 18: Keysight SD1 SFP provides software front panels, a fast and intuitive way of operating any Keysight hardware

This information is subject to change without notice.
© Keysight Technologies 2013-2020

Edition 2, February, 2020

M3100-90002

www.keysight.com