User's Guide

Keysight M3601A HVI Design Environment

KEYSIGHT TECHNOLOGIES

Notices

Copyright Notice

© Keysight Technologies 2013 - 2017

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

M3601-90001

Published By

Keysight Technologies
1400 Fountaingrove Parkway
Santa Rosa
CA 95405

Edition

Edition 1, May, 2021
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 매자 또는 사용자는 이 점을 주의하시기 바라 며 , 가정외의 지역에서 사용하는 것을 목적으 로 합니다.

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

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

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

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

Contents

1 Keysight HVI Design Environment 1
1. 1 Introduction: the HVI Design Environment 1
1. 2 M3601A HVI Design Environment Basics 1
1. 2. 1 Main Window 1
1. 2. 2 Creating an HVI Project 2
1. 2. 3 Creating HVI Instructions 3
1. 2. 4 HVI Compilation and Execution 3
1. 2. 5 Hardware Module Options 3
1. 2. 6 HVI Registers 3
1. 3 HVI Flowcharts 4
1. 3. 1 Timing Arrows 4
1. 3. 2 Synchronized Flow-Control Statements 4
1. 3. 2. 1 Start 5
1. 3. 2. 2 End 5
1. 3. 2. 3 Synchronized Junction 5
1. 3. 2. 4 Synchronized Conditional 6
1. 3. 2. 5 Local Flow-control Statements 7
1. 3. 2. 6 If Block 8
1. 3. 2. 7 For Block 9
1. 3. 2. 8 While Block 10
1. 3. 2. 9 Wait 10
1. 3. 3 Instruction Statements 12
1. 3. 3. 1 Built-in Instructions 12
1. 3. 3. 2 External Variable Access 14
2 Software Programming Guide: HVI-VI Interaction 15
2. 1 Programming Functions 15
2. 1. 1 SW Programming Overview Programming Libraries 15
2. 1. 2 SD_HVI Functions 16
2. 1. 2. 1 open 16
2. 1. 2. 2 close 18
2. 1. 2. 3 start 19
2. 1. 2. 4 pause 19
2. 1. 2. 5 resume 20
2. 1. 2. 6 stop 21
2. 1. 2. 7 reset 23
2. 1. 2. 8 compile 23
2. 1. 2. 9 compilationErrorMessage 24
2. 1. 2. 10 load 25
2. 1. 2. 11 assignHardware 26
2. 1. 2. 12 getNumberOfModules 28

2.1.2.13 getModuleName ... 29
2.1.2.14 getModuleIndex ... 30
2.1.2.15 getModule ... 31
2.1.2.16 writeConstant ... 32
2.1.2.17 readConstant ... 33
2.1.3 SD Module Functions (HVI-related) ... 35
2.1.3.1 writeRegister ... 35
2.1.3.2 readRegister ... 37
2.1.4 Error Codes ... 39
3 Addendum: Keysight Technology and Software Overview ... 41
3.1 Programming Tools ... 41
3.1.1 SW Programming ... 41
3.1.1.1 Keysight SD1 Programming Libraries ... 42
3.1.2 HW Programming ... 42
3.1.2.1 HVI Technology: Keysight M3601A ... 42
3.1.2.2 FPGA Programming: Keysight M3602A ... 46
3.1.2.3 Keysight M3602A: An FPGA Design Environment ... 47
3.2 Design Process: Customization vs. Complete Design ... 48
3.3 Application Software ... 49
3.3.1 Keysight SD1 SFP ... 49

1 Keysight HVI Design Environment

1.1 Introduction: the HVI Design Environment

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.

M3601A HVI Design Environment is supported by all M3XXXA AWG and Digitizer PXIe modules with -HV1 option enabled (this option should be selected with purchase). Additional information is available at HVI Technology: Keysight M3601A (page 42)).

1.2 M3601A HVI Design Environment Basics

Keysight M3601A allows the user to create or edit HVIs using an intuitive graphical flowchart interface. This section introduces the basics of this interface.

1.2.1 Main Window

M3601A main window is shown in Figure 1. The Properties window located on the left part of the screen is a multipurpose tool that shows the instructions and properties of all the objects which are activated by a mouse left-click. The Flowcharts of the two hardware modules comprising this HVI can be identified on the right side of the window. These flowcharts contain statements (different shaped boxes) and time arrows (Section Timing Arrows (page 4)) which define the execution of the HVI.

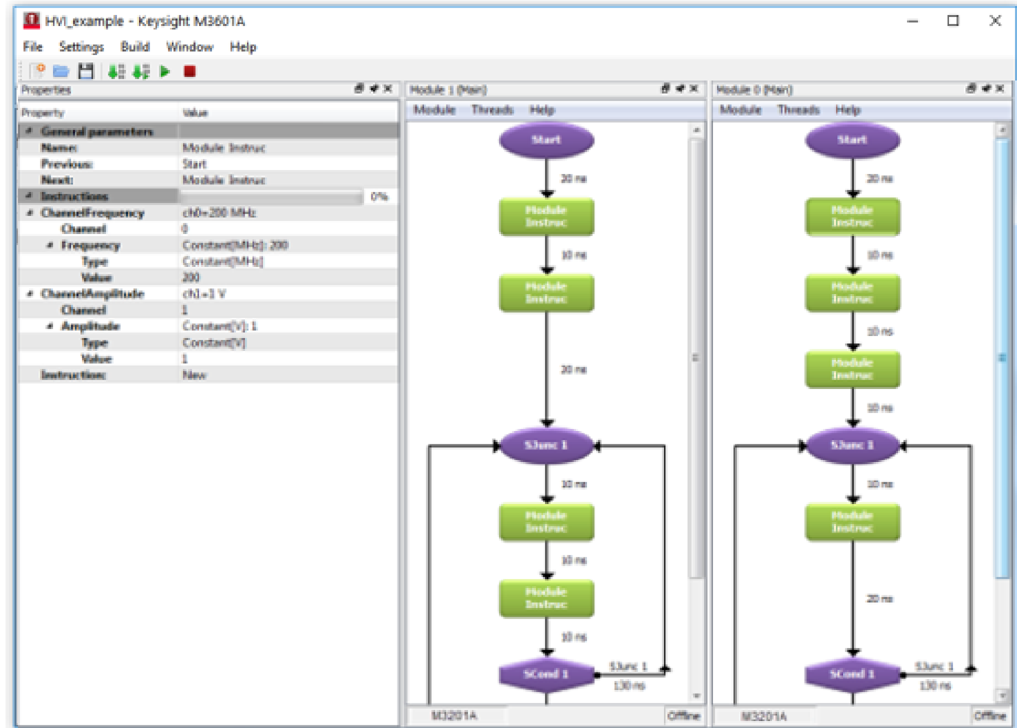


Figure 1: M3601A HVI Design Environment

NOTE Docking options: all windows of Keysight M3601A can be docked at will to adapt its appearance to the application and the screen size. These windows can also be undocked using the thumbtack. Take your time to familiarize with the flexible window system.

1. 2. 2 Creating an HVI Project

An HVI is defined by a group of hardware modules* that are fully synchronized, working as a single integrated hardware module. Therefore, the creation of a new HVI starts by selecting its modules (Figure 2), after clicking File ⇒ New HVI Project.

NOTE Reassigning hardware modules: (Module ⇒ Change) once the HVI was created, a hardware module can be reassigned to another hardware module of the same type. This is particularly useful when modules are changed from one physical slot to another, or when the HVI is created without the hardware connected (modules are opened offline) and must be assigned to a physical hardware module when available.

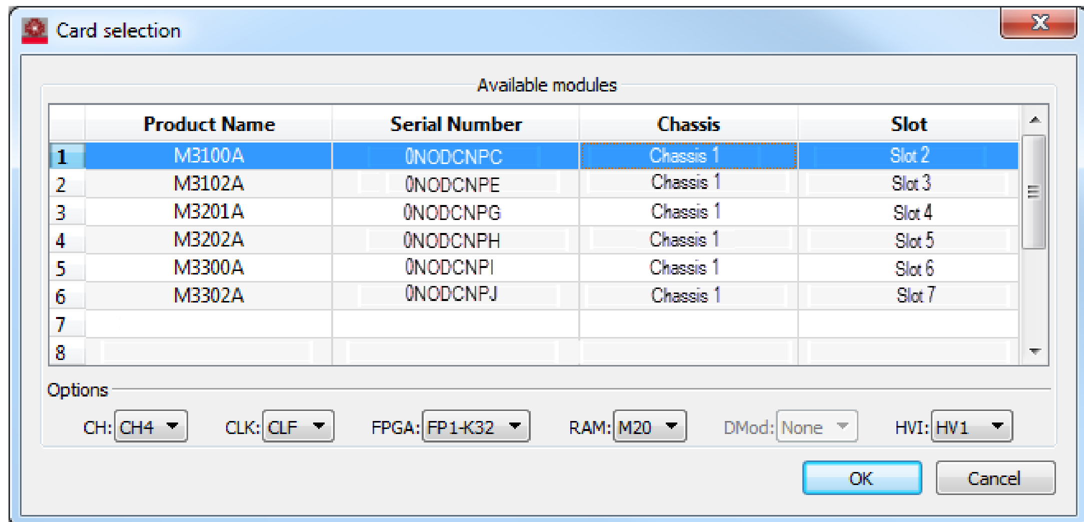


Figure 2: New Hard Virtual Instrument (HVI) project creation. Dialog to select the hardware modules that will become part of the HVI.

1. 2. 3 Creating HVI Instructions

An empty project shows all the hardware modules* with the Start (page 5). A mouse right-click over any statement (like the Start Statement) opens a menu to create and delete new flowchart elements (see HVI Flowcharts (page 4) to learn more about these elements).

1. 2. 4 HVI Compilation and Execution

HVI Projects can be compiled and executed directly within M3601A using the Build option on the main menu. An HVI Project can also be compiled into an HVI object (File ⇒ Generate HVI...). With this file, the HVI can be executed from a VI using the Keysight SD1 Programming Libraries, e.g. from C/C++, LabVIEW, Python, MATLAB, etc. See Software Programming Guide: HVI-VI Interaction (page 15) for more information about the interaction between VIs and HVIs.

1. 2. 5 Hardware Module Options

Each hardware module may have different options, e.g. an AWG may be able to define arbitrary waveforms, etc., which are accessible from the module menu in the Main thread window (Module ⇒ ...).

1. 2. 6 HVI Registers

Each hardware module has 16 local registers (32 bits integers) that can be used in the programming of the HVI. Most statements can write or read into/from these registers. Registers can be

read/written between modules, providing a fast tool to transfer small amounts of data (see how to access external variables External Variable Access (page 14)).

> **NOTE** PC-HVI Semaphores: HVI module registers are also useful for signaling purposes between the PC and the HVI. For example, the HVI can wait until the PC changes a register or viceversa, allowing the user to perform tasks in the PC and in the HVI simultaneously or alternatively.

1.3 HVI Flowcharts

Flowcharts represent a program or execution sequence by means of instructions and flow-control boxes (called statements), and timing arrows. Statements define HVI actions, while arrows define the execution flow and timing. All these elements are explained throughout this section.

1.3.1 Timing Arrows

Timing Arrows define and inform about the time between statements. Therefore, the user may thing as if statements do not take any time and all the timing information is given by the Timing Arrows. There are two kind of arrows:

- Solid Arrows: Normal arrows which connect statements. They can have a well defined time at programming time, or they can show a time uncertainty with a question mark (e.g. Figure 4).
- Dashed Arrows: The dashed arrows show a time uncertainty due to a synchronization recovery (the modules must wait for other modules). These arrows appear before a (Synchronized Flow-Control Statements (page 4)) when there has been a desynchronization event (e.g. after Wait Statements, loops with variable cycles, etc.). For dashed arrows, the user can set the minimum time, but the maximum its only defined at runtime.

> **NOTE** Synchronization: Dashed arrows indicate a desynchronization between the modules which is being recovered by introducing a Synchronized Flow-control Statement.

1.3.2 Synchronized Flow-Control Statements

Flow-control statements are elements that control the HVI execution flow. Synchronized Flow-Control Statements are particular Flow- control Statements that are executed in all modules at the same time.

NOTE Synchronization Recovery: Some statements like Wait, For Blocks or While Blocks may have an undefined execution time while programming (their timing is only defined during runtime). Therefore, these statements produce a desynchronization of the modules. Synchronized Flow-Control Statements can be used as synchronization elements after a desynchronization.

NOTE Synchronized and Local Flow-control Statements Colors: Synchronized Flow-Control Statements are show in the flowchart as violet elements, while Local Flow-Control Statements are show as yellow elements.

1. 3. 2. 1 Start

Every flowchart begins with a Start Statement (Figure 3), which is in fact a Synchronized Junction Statement (Figure 4), This statement ensures that all the hardware modules in an HVI start the execution fully synchronized.

NOTE Junction Statement: The Start Statement is really a Junction Statement (Synchronized Junction (page 5)), and therefore can be used as a destination for Conditional Statements (Figure 4).

1. 3. 2. 2 End

The End Statement indicates the end of the HVI execution, and must be placed by the user to terminate the HVI properly (Figure 3).

1. 3. 2. 3 Synchronized Junction

A Synchronized Junction Statement is an element that can be used as a destination for conditional jumps (Synchronized Conditional (page 6)).

NOTE Synchronization Recovery: In addition to operate as a destination for conditional jumps, the Synchronized Junction Statement is ideal to perform synchronization recovery (Figure 4).

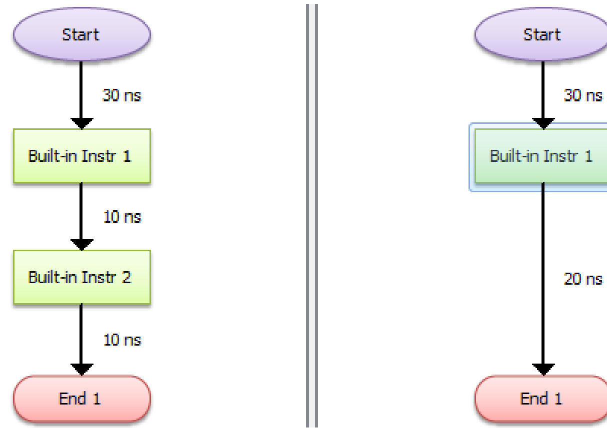


Figure 3: Flowcharts of two modules, showing the Start and the End Statements

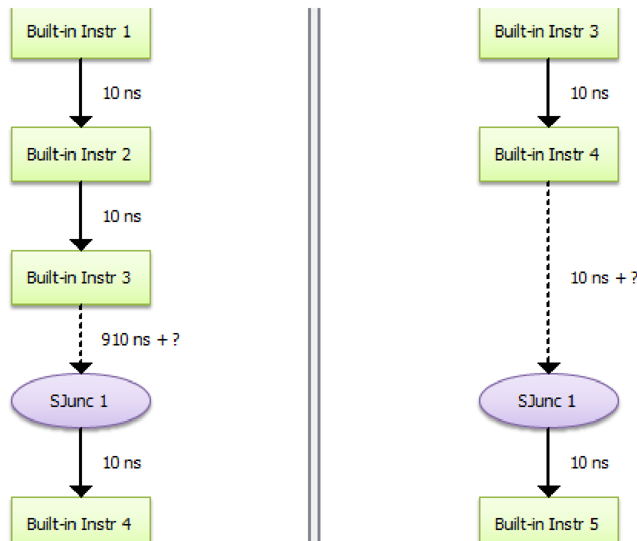


Figure 4: A Synchronized Junction Statement (ellipse-shaped element) can also be used to synchronize flowcharts

1. 3. 2. 4 Synchronized Conditional

M3601A allows the user to perform conditional jumps to control the HVI execution flow, enabling the implementation of ultra-fast decision making, loops, etc.

A synchronized jump in the execution flow requires two elements:

1. Synchronized Conditional Statement, which is the element that evaluates the condition and performs the jump
2. Destination element.

The latter can be the statement right below the conditional or a Junction Statement placed anywhere in the flowchart (Figure 5).

Leader and follower synchronized conditionals: In a Synchronized Conditional Statement, the decision is taken by only one of the modules (called Leader), and all the others (follower) will comply with the same conditional result.

NOTE

Synchronization Recovery: Like any other Synchronized Flow-control Statement, a Synchronized Conditional will synchronize all modules after a desynchronization.

1. 3. 2. 5 Local Flow-control Statements

Flow-control Statements are elements that control the HVI execution flow. Local Flow-Control Statements are particular Flow-control Statements that are executed only in one module, without affecting the execution flow of other modules.

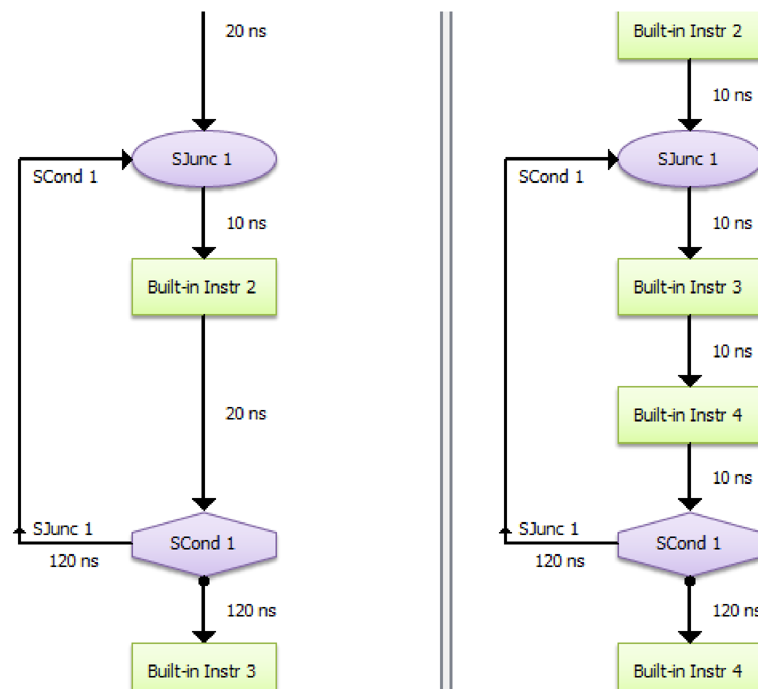


Figure 5: A Synchronized Conditional Statement (diamond-shaped element) jumps to a Junction Statement (ellipse-shaped element) in two modules exactly at the same time. The GUI maintains the synchronization by adjusting automatically the time of the last Timing Arrow. The solid dot in the arrow below the Conditional indicates the jump under a FALSE condition evaluation, while the side branch indicates the jump under a TRUE condition evaluation.

NOTE

Synchronized and Local Flow-control Statements Colors: Synchronized Flow-Control Statements are show in the flowchart

NOTE as violet elements, while Local Flow-Control Statements are show as yellow elements.

1. 3. 2. 6 If Block

The If Block Statement is the classical "if/else structure" available in any programming language. The If Block provides two operation options:

- Branch Synchronization Forced: the total time of the two branches (if and else) is forced to be equal, and therefore the execution time is well defined (does not depend on the branch). In this case, the If Block Statement does not desynchronize.
- Branch Synchronization Unforced: the total time of the two branches (if and else) can be different, and therefore the execution time may be defined only at runtime (when the branch is decided). If this is the case, the If Block Statement produces desynchronization.

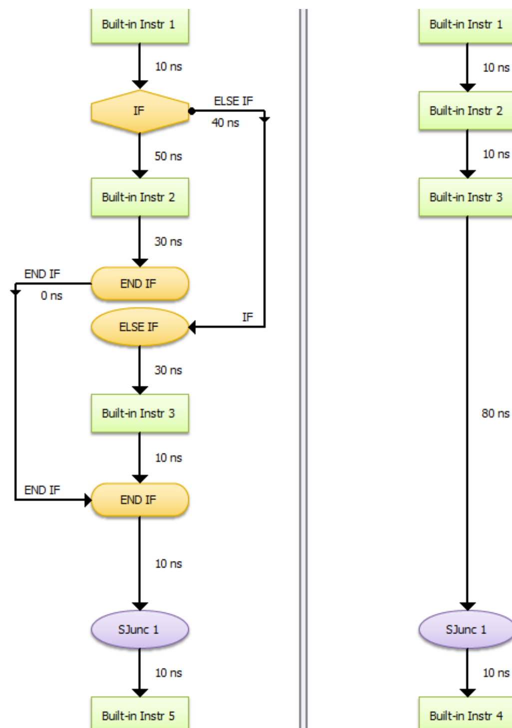


Figure 6: An If Block with equal timing in both branches does not desynchronize

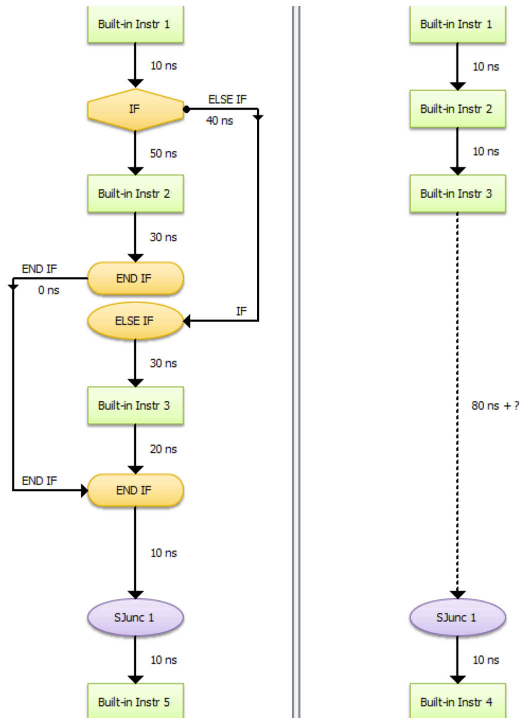


Figure 7: An If Block with different timing in both branches desynchronizes, as it it shown in the flowchart of the other modules (right)

1. 3. 2. 7 For Block

The For Block Statement is the classical ”for loop structure” of any programming language. The user can choose which local register (M3601A HVI Design Environment Basics (page 1)) is used for the loop index, and the exit value (number of loops).

NOTE Undefined number of loops: If the number of loops is not fixed at programming time (e.g. it is defined by a local register), the For Block Statement desynchronizes.

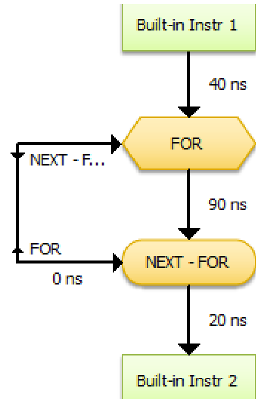


Figure 8: For Block Statement

1. 3. 2. 8 While Block

A While Block Statement is the classical "while loop" structure of any programming language. The execution loops while a condition is fulfilled.

NOTE Undefined number of loops: Like in the case of the For Block, if the exit condition depends on registers (which values are only defined at runtime), then the While Block desynchronizes.

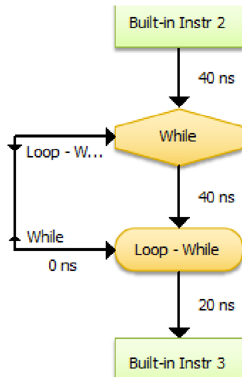


Figure 9: While Block Statement

1. 3. 2. 9 Wait

The Wait Statement can be configured to wait for two different events:

- Variable Time: It waits the amount of time defined in a local register. The time unit in the register is defined in tens of nanoseconds.
- PXI Trigger: It waits until the selected PXI trigger has the desired value (On or Off).

NOTE Undefined waiting time: The exit condition of the Wait Statement depends on registers (which values are only defined at runtime) or external signals (e.g. PXI triggers), therefore the execution time is not well defined at programming time, producing desynchronization. The Timing Arrow below the Wait Statement informs about the undefined execution time by adding a question mark to the specified time (Figure 10).

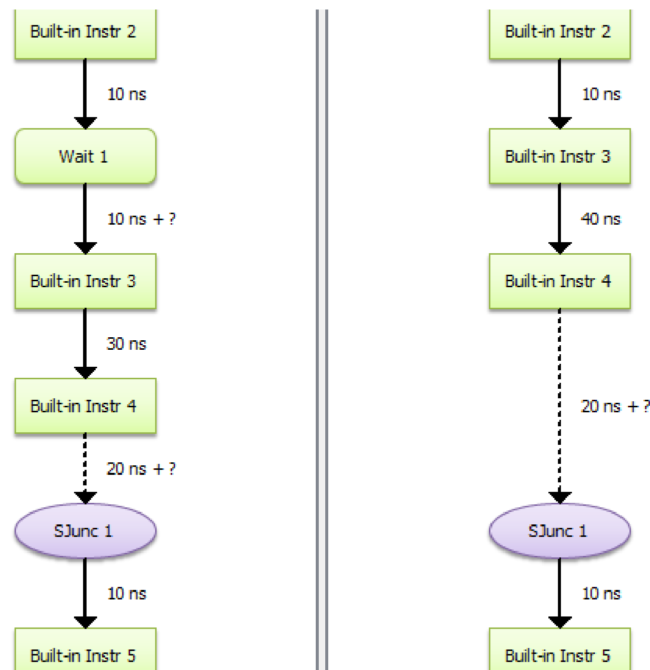


Figure 10: Wait Statement with an undefined execution time (left flowchart). This timing uncertainty can be seen in the arrow below the Wait. Please note that this also produces a desynchronization, which is shown with the dashed arrows above the Synchronized Junction. The latter performs a synchronization recovery in both modules (left and right).

NOTE PXI triggers: PXI triggers are dedicated lines available in the PXI backplane, and can be used at will to exchange signals between modules.

NOTE **ADVANCED: Converting a Local Wait into a Synchronized Wait : By inserting a Synchronized Junction after a Wait, all modules will be forced to wait for the module that takes more time to reach the junction, as the junction resynchronizes all flowcharts.**

1. 3. 3 Instruction Statements

Instruction statements describe the main actions of an HVI, without altering the execution flow directly.

1. 3. 3. 1 Built-in Instructions

Built-in instructions are the main execution statements of an HVI. There are two kind of instructions statements: module-specific, e.g. an instruction to change the amplitude of an Arbitrary Waveform Generator (AWG) (built-in module-specific instructions are described in the User Guide of the module), and universal instruction statements, like arithmetic operations, etc., which are available in all Keysight compatible modules with -HV1 programming enabled option.

Instruction Execution Latency: Instructions are triggered according to the statement timing within the flowchart, however each module-specific instruction has its own execution latency. Check the module datasheet for latency values.

1. 3. 3. 1. 1 Built-in Instructions: Universal

This chapter describes the universal instruction statements available in all modules.

1. 3. 3. 1. 1. 1 MathAssign (D=S)

This function copies the value of the source parameter (S) into the destination variable (D).

Parameters

Name	Description
Inputs	
Destination	Destination local register
Source	Source value to be assigned: · Constant in the specified format (Decimal, Binary, Hexadecimal, etc.) · Source local register · Module-specific digital values

1. 3. 3. 1. 1. 2 MathArithmetics (R=A[+-*/]B)

This function subtracts, adds, multiplies or divides the values of the operands A and B, writing the result in R.

Parameters

Name	Description
Inputs	
Result	Destination local register
A	First operand of the arithmetic operation: · Constant in the specified format (Decimal, Binary, Hexadecimal, etc.) · Source local register · Module-specific digital values
Operation	Arithmetic operation (+, -, *, /)
B	Second operand of the arithmetic operation: · Constant in the specified format (Decimal, Binary, Hexadecimal, etc.) · Source local register · Module-specific digital values

1. 3. 3. 1. 1. 3 MathLogic (R=A [op] B)

This function performs a logic operation with the values of the operands A and B, writing the result in R.

Parameters

Name	Description
Inputs	
Result	Destination local register
A	First operand of the operation: · Constant in the specified format (Decimal, Binary, Hexadecimal, etc.) · Source local register · Module-specific digital values
Operation	Logic operation (and, or, xor, etc.)
B	Second operand of the operation: · Constant in the specified format (Decimal, Binary, Hexadecimal, etc.) · Source local register · Module-specific digital values

1. 3. 3. 1. 1. 4 MathSQRT (R=sqrt(A))

This function performs the square of the source parameter (A), writing the result in R.

Parameters

Name	Description
Inputs	
Destination	Destination local register
A	Source value to be assigned: · Constant in the specified format (Decimal, Binary, Hexadecimal, etc.) · Source local register

Name	Description
	· Module-specific digital values

1. 3. 3. 1. 1. 5 HVIport

This function performs a read or write operation on port HVI available in M3602A.

Parameters

Name	Description
Inputs	
Port	Port index of HVI port
Operation	Defines if it's a read or write operation
Address	Offset from HVI port base address expressed in double words
Source	(Available only on write operation) Source value to be write on HVI port
Destination	(Available only on read operation) Destination local register

1. 3. 3. 1. 2 Built-in Instructions: Module-specific

Module-specific instructions are explained in the user guides of the corresponding hardware modules.

1. 3. 3. 2 External Variable Access

This statement writes or reads into/from a variable of another module (M3601A HVI Design Environment Basics (page 1)).

NOTE External Variable Access time: The execution time of the External Variable Access Statement is undefined, because it depends, among others, on the data bus bandwidth and congestion. For this reason, this statement produces a desynchronization and dashed Time Arrows will appear in the flowcharts (see Timing Arrows (page 4) for more details).

2 Software Programming Guide: HVI-VI Interaction

Thanks to the Keysight Programming Libraries, HVIs created with Keysight M3601A integrate seamlessly with the user application, commonly referred in Keysight documentation as VI (Virtual Instrument, Section Programming Tools (page 41)).

Within M3601A, an HVI Project can be compiled into an HVI binary file (File ⇒ Generate HVI...). With this file, the HVI can be executed and controlled from the user application, e.g. the HVI can be launched, paused, stopped, etc. VIs and HVIs can also exchange data and signals.

This Section describes the programming functions designed to control HVIs.

2.1 Programming Functions

2.1.1 SW Programming Overview Programming Libraries

Keysight provides highly optimized programming libraries to operate the Keysight M3601A HVI Design Environment which is supported by the following modules: M3100A, M3102A, M3201A, M3202A, M3300A and M3302A with -HV1 option enabled.

Native Programming 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

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 Language	Operating System	Files
C	Microsoft Windows	*.dll

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

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.
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. 1. 2 SD_HVI Functions

2. 1. 2. 1 open

This function loads a Hard Virtual Instrument (HVI) previously created with Keysight M3601A into the hardware modules (see Software Programming Guide: HVI-VI Interaction (page 15)).

Advanced

Memory usage: HVIs do not occupy RAM in the hardware modules, they have a dedicated HVI memory.

Parameters

Name	Description
Inputs	
HVIfile	The path of the HVI file (*.HVI) created with Keysight M3601A
errorIn	If it contains an error, the function will not be executed .and errorIn will be passed to errorOut
Outputs	
HVIID	HVI Identifier, or a negative number in case of error (see error codes in Table 1)
errorOut	See error codes in Table 1

C Function

int SD_HVI_open(char* HVIfile);

C++ Function

int SD_HVI::open(char* HVIfile);

Visual Studio .NET, MATLAB

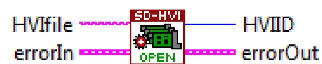
int SD_HVI::open(string HVIfile);

Python

int SD_HVI::open(string HVIfile);

LabVIEW

SD_HVI open.vi



2. 1. 2. 2 close

This function closes the HVI, removing it from the hardware modules.

Parameters

Name	Description
Inputs	
HVIID	HVI identifier (returned by Open)
errorIn	If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
errorOut	See error codes in Table 1

C Function

int SD_HVI_close(int HVIID);

C++ Function

int SD_HVI::close();

Visual Studio .NET, MATLAB

int SD_HVI::close();

Python

int SD_HVI::close();

LabVIEW

SD_HVI close.vi

2. 1. 2. 3 start

This function starts the HVI execution from the beginning.

Parameters

Name	Description
Inputs	
HVIID	HVI identifier (returned by Open)
errorIn	If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
HVIIDOut	A copy of HVIID
errorOut	See error codes in Table 1

C Function

int SD_HVI_start(int HVIID);

C++ Function

int SD_HVI::start();

Visual Studio .NET, MATLAB

int SD_HVI::start();

Python

int SD_HVI::start();

LabVIEW

SD_HVI start.vi

2. 1. 2. 4 pause

This function pauses the execution of an HVI.

Parameters

Name	Description
Inputs	
HVIID	HVI identifier (returned by Open)
errorIn	If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
HVIIDOut	A copy of HVIID
errorOut	See error codes in Table 1

C Function

int SD_HVI_pause(int HVIID);

C++ Function

int SD_HVI::pause();

Visual Studio .NET, MATLAB

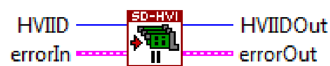
int SD_HVI::pause();

Python

int SD_HVI::pause();

LabVIEW

SD_HVI pause.vi



2. 1. 2. 5 resume

This function resumes the execution of an HVI paused with the function Pause.

Parameters

Name	Description
Inputs	
HVIID	HVI identifier (returned by Open)
errorIn	If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
HVIIDOut	A copy of HVIID
errorOut	See error codes in Table 1

C Function

int SD_HVI_resume(int HVIID);

C++ Function

int SD_HVI::resume();

Visual Studio .NET, MATLAB

int SD_HVI::resume();

Python

int SD_HVI::resume();

LabVIEW

SD_HVI resume.vi

2. 1. 2. 6 stop

This function finishes the HVI execution, releasing all the resources needed for the HVI.

Parameters

Name	Description
Inputs	
HVIID	HVI identifier (returned by Open)
errorIn	If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
HVIIDOut	A copy of HVIID
errorOut	See error codes in Table 1

C Function

int SD_HVI_stop(int HVIID);

C++ Function

int SD_HVI::stop();

Visual Studio .NET, MATLAB

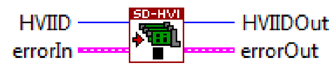
int SD_HVI::stop();

Python

int SD_HVI::stop();

LabVIEW

SD_HVI stop.vi



2. 1. 2. 7 reset

This function resets the HVI.

Parameters

Name	Description
Inputs	
HVIID	HVI identifier (returned by Open)
errorIn	If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
HVIIDOut	A copy of HVIID
errorOut	See error codes in Table 1

C Function

int SD_HVI_reset(int HVIID);

C++ Function

int SD_HVI::reset();

Visual Studio .NET, MATLAB

int SD_HVI::reset();

Python

int SD_HVI::reset();

LabVIEW

SD_HVI reset.vi

2. 1. 2. 8 compile

This function compiles the HVI.

Parameters

Name	Description
Inputs	
HVIID	HVI identifier (returned by Open)
errorIn	If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
HVIIDOut	A copy of HVIID
nErrors	Number of error found during HVI compilation. To read the error message call function compilationErrorMessage (Section 2.1.2.9)
errorOut	See error codes in Table 1

C Function

int SD_HVI_compile(int HVIID);

C++ Function

int SD_HVI::compile();

Visual Studio .NET, MATLAB

int SD_HVI::compile();

Python

int SD_HVI::compile();

LabVIEW

SD_HVI_compile.vi

2. 1. 2. 9 compilationErrorMessage

This function gets the indicated compilation error message.

Parameters

Name	Description

Name	Description
Inputs	
HVIID	HVI identifier (returned by Open)
errorIndex	Index of error (from 0 to nErrors -1: section 2.1.2.8)
maxSize	Size of message buffer
errorIn	If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
HVIIDOut	A copy of HVIID
message	Buffer to copy indicated error message
errorOut	See error codes in Table 1

C Function

int SD_HVI_compilationErrorMessage(int HVIID, int errorIndex, char *message, int maxSize);

C++ Function

int SD_HVI::compilationErrorMessage(int errorIndex, char *message, int maxSize);

Visual Studio .NET, MATLAB

int SD_HVI::compilationErrorMessage(int errorIndex, out string message);

Python

{int, string} SD_HVI::compilationErrorMessage(int errorIndex);

LabVIEW

SD_HVI_compilationErrorMessage.vi

2. 1. 2. 10 load

This function loads the HVI to the modules.

Parameters

Name	Description
Inputs	
HVIID	HVI identifier (returned by Open)
errorIn	If it contains an error, the function will not be executed and errorIn will be passed to errorOut

Name	Description
Outputs	
HVIIDOut	A copy of HVIID
errorOut	See error codes in Table 1

C Function

int SD_HVI_load(int HVIID);

C++ Function

int SD_HVI::load();

Visual Studio .NET, MATLAB

int SD_HVI::load();

Python

int SD_HVI::load();

LabVIEW

SD_HVI_load.vi

2. 1. 2. 11 assignHardware

This function substitutes one HVI module by another compatible module.

Parameters

Name	Description
Inputs	
HVIID	HVI identifier (returned by Open)
index	Module index inside the HVI
moduleUserName	Nick name of the module defined by the user within HVI
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. [1]
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. [1]

Name	Description
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. [1]
module	ID or module object to assign.
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. [1]
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 Table 1
errorOut	See Error Codes in Table 1

1 These are module related functions: please, consult the User Guide of your product

C

int SD_HVI_assignHardwareWithIndexAndSerialNumber(int HVIID, int index, const char *productName, const char *serialNumber);

int SD_HVI_assignHardwareWithIndexAndSlot(int HVIID, int index, int chassis, int slot);

int SD_HVI_assignHardwareWithUserNameAndSerialNumber(int HVIID, const char *moduleUserName, const char *productName, const char *serialNumber);

int SD_HVI_assignHardwareWithUserNameAndSlot(int HVIID, const char *moduleUserName, int chassis, int slot);

int SD_HVI_assignHardwareWithUserNameAndModuleID(int HVIID, const char *moduleUserName, int module);

C++

int SD_HVI::assignHardware(int index, const char *productName, const char *serialNumber);

int SD_HVI::assignHardware(int index, int chassis, int slot);

int SD_HVI::assignHardware(const char *moduleUserName, const char *productName, const char *serialNumber);

int SD_HVI::assignHardware(const char *moduleUserName, int chassis, int slot);

int SD_HVI::assignHardware(const char *moduleUserName, SD_Module *module);

Visual Studio .NET, MATLAB

int SD_HVI::assignHardware(int index, string productName, string serialNumber);

int SD_HVI::assignHardware(int index, int chassis, int slot);

int SD_HVI::assignHardware(string moduleUserName, string productName, string serialNumber);

int SD_HVI::assignHardware(string moduleUserName, int chassis, int slot);

int SD_HVI::assignHardware(string moduleUserName, SD_Module module);

Python

int SD_HVI::assignHardwareWithIndexAndSerialNumber(int index, string productName, string serialNumber);

int SD_HVI::assignHardwareWithIndexAndSlot(int index, int chassis, int slot);

int SD_HVI::assignHardwareWithUserNameAndSerialNumber(string moduleUserName, string productName, string serialNumber);

int SD_HVI::assignHardwareWithUserNameAndSlot(string moduleUserName, int chassis, int slot);

int SD_HVI::assignHardwareWithUserNameAndModuleID(string moduleUserName, SD_Module module);

LabVIEW

assignHardware.vi

2. 1. 2. 12 getNumberOfModules

This function returns the number of modules controlled by the HVI. '

Parameters

Name	Description
Inputs	
HVIID	HVI identifier (returned by Open)
errorIn	If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
HVIIDOut	A copy of HVIID
nModules	Number of modules controlled by the HVI
errorOut	See error codes in Table 1

C Function

int SD_HVI_getNumberOfModules(int HVIID);

C++ Function

int SD_HVI::getNumberOfModules();

Visual Studio .NET, MATLAB

int SD_HVI::getNumberOfModules();

Python

int SD_HVI::getNumberOfModules();

LabVIEW

SD_HVI_getNumberOfModules.vi

2. 1. 2. 13 getModuleName

This function returns the nick name of one of the modules controlled by the HVI.

Parameters

Name	Description
Inputs	
HVIID	HVI identifier (returned by Open)
index	Module index inside the HVI
size	Size of the buffer
errorIn	If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
HVIIDOut	A copy of HVIID
buffer	Nick name of the indicated module
errorOut	See error codes in Table 1

C Function

int SD_HVI_getModuleName(int HVIID, int index, char *buffer, int size);

C++ Function

int SD_HVI::getModuleName(int index, char *buffer, int size);

Visual Studio .NET, MATLAB

string SD_HVI::getModuleName(int index);

Python

string SD_HVI::getModuleName(int index);

LabVIEW

SD_HVI_getModuleName.vi

2. 1. 2. 14 getModuleIndex

This function returns the nick name of one of the modules controlled by the HVI.

Parameters

Name	Description
Inputs	
HVIID	HVI identifier (returned by Open)
moduleUserName	Nick name of the module defined by the user within HVI
errorIn	If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
HVIIDOut	A copy of HVIID
errorOut	See error codes in Table 1

C Function

int SD_HVI_getModuleIndex(int HVIID, const char *moduleUserName);

C++ Function

int SD_HVI::getModuleIndex(const char *moduleUserName);

Visual Studio .NET, MATLAB

int SD_HVI::getModuleIndex(string moduleUserName);

Python

int SD_HVI::getModuleIndex(string moduleUserName);

LabVIEW

SD_HVI_getModuleIndex.vi

2. 1. 2. 15 getModule

This function returns the selected module from the HVI.

Parameters

Name	Description
Inputs	
HVIID	HVI identifier (returned by Open)
index	Module index inside the HVI
moduleUserName	Nick name of the module defined by the user within HVI
errorIn	If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
HVIIDOut	A copy of HVIID
module	ID or module object
errorOut	See error codes in Table 1

C Function

int SD_HVI_getModuleIDwithIndex(int HVIID, int index);

int SD_HVI_getModuleIDwithUserName(int HVIID, const char *moduleUserName);

C++ Function

SD_Module* SD_HVI::getModule(int index);

SD_Module* SD_HVI::getModule(const char *moduleUserName);

Visual Studio .NET, MATLAB

SD_Module SD_HVI::getModule(int index);

SD_Module SD_HVI::getModule(string moduleUserName);

Python

SD_Module SD_HVI::getModuleByIndex(int index);

SD_Module SD_HVI::getModuleByName(string moduleUserName);

LabVIEW

SD_HVI_getModuleID.vi

2. 1. 2. 16 writeConstant

This function writes a value in an HVI constant of a module.

Parameters

Name	Description
Inputs	
HVIID	HVI identifier (returned by Open)
moduleIndex Mod-ule	index inside the HVI
moduleUserName	Nick name of the module defined by the user within HVI
constantName	Constant name
value	Constant value
unit	Unit of the constant
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut
Outputs	
HVIIDOut	A copy of HVIID
errorOut	See Error Codes in Table 1

C

int SD_HVI_writeIntegerConstantWithIndex(int HVIID, int moduleIndex, const char *constantName, int value);

int SD_HVI_writeIntegerConstantWithUserName(int HVIID, const char *moduleUserName, const char *constantName, int value);

int SD_HVI_writeDoubleConstantWithIndex(int HVIID, int moduleIndex, const char *constantName, double value, const char *unit);

int SD_HVI_writeDoubleConstantWithUserName(int HVIID, const char *moduleUserName, const char *constantName, double value, const char *unit);

C++

int SD_HVI::writeConstant(int moduleIndex, const char *constantName, int value);

int SD_HVI::writeConstant(const char *moduleUserName, const char *constantName, int value);

int SD_HVI::writeConstant(int moduleIndex, const char *constantName, double value, const char *unit);

int SD_HVI::writeConstant(const char *moduleUserName, const char *constantName, double value, const char *unit);

Visual Studio .NET, MATLAB

int SD_HVI::writeConstant(int moduleIndex, string constantName, int value);

int SD_HVI::writeConstant(string moduleUserName, string constantName, int value);

int SD_HVI::writeConstant(int moduleIndex, string constantName, double value, string unit);

int SD_HVI::writeConstant(string moduleUserName, string constantName, double value, string unit);

Python

int SD_HVI::writeIntegerConstantWithIndex(int moduleIndex, string constantName, int value);

int SD_HVI::writeIntegerConstantWithUserName(string moduleUserName, string constantName, int value);

int SD_HVI::writeDoubleConstantWithIndex(int moduleIndex, string constantName, double value, string unit);

int SD_HVI::writeDoubleConstantWithUserName(string moduleUserName, string constantName, double value, string unit);

LabVIEW

writeConstant.vi

2. 1. 2. 17 readConstant

This function reads the value of an HVI constant of a module.

Parameters

Name	Description
Inputs	
HVIID	HVI identifier (returned by Open)
moduleIndex	Module index inside the HVI
moduleUserName	Nick name of the module defined by the user within HVI
constantName	Constant name
errorIn	(LabVIEW only) If it contains an error, the function will not be executed and errorIn will be passed to errorOut

Name	Description
Outputs	
HVIIDOut	A copy of HVIID
value	Constant value
unit	Unit of the constant
errorOut	See Error Codes in Table 1

C

int SD_HVI_readIntegerConstantWithIndex(int HVIID, int moduleIndex, const char *constantName, int &value);

int SD_HVI_readIntegerConstantWithUserName(int HVIID, const char *moduleUserName, const char *constantName, int &value);

int SD_HVI_readDoubleConstantWithIndex(int HVIID, int moduleIndex, const char *constantName, double &value, const char * &unit);

int SD_HVI_readDoubleConstantWithUserName(int HVIID, const char *moduleUserName, const char *constantName, double &value, const char * &unit);

C++

int SD_HVI::readConstant(int moduleIndex, const char *constantName, int &value);

int SD_HVI::readConstant(const char *moduleUserName, const char *constantName, int &value);

int SD_HVI::readConstant(int moduleIndex, const char *constantName, double &value, const char *&unit);

int SD_HVI::readConstant(const char *moduleUserName, const char *constantName, double &value, const char *&unit);

Visual Studio .NET, MATLAB

int SD_HVI::readConstant(int moduleIndex, string constantName, out int value);

int SD_HVI::readConstant(string moduleUserName, string constantName, out int value);

int SD_HVI::readConstant(int moduleIndex, string constantName, out double value, out string unit);

int SD_HVI::readConstant(string moduleUserName, string constantName, out double value, out string unit);

Python

[int errorOut, int value] SD_HVI::readIntegerConstantWithIndex(int moduleIndex, string constantName);

[int errorOut, int value] SD_HVI::readIntegerConstantWithUserName(string moduleUserName, string constantName);

[int errorOut, double value, string unit] SD_HVI::readDoubleConstantWithIndex(int moduleIndex, string constantName);

[int errorOut, double value, string unit] SD_HVI::readDoubleConstantWithUserName(string moduleUserName, string constantName);

LabVIEW

readConstant.vi

2. 1. 3 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. 1. 3. 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 (page 16)
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 in Table 1

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. 1. 3. 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 (page 16)
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	
regValue	Register value
moduleIDout	(LabVIEW only) A copy of moduleID
errorOut	Error Codes (page 39)

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 SD_Module::readRegisterWithName(string regName);

[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. 1. 4 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

Table 1: Software error codes

3 Addendum: Keysight Technology and Software Overview

This is an overview of the M3100A, M3102A, M3201A, M3202A, M3300A and M3302A family of PXIe modules.

3.1 Programming Tools

The diagram shown in Figure 9 summarizes the programming tools available to control M3XXA Keysight PXIe Hardware.

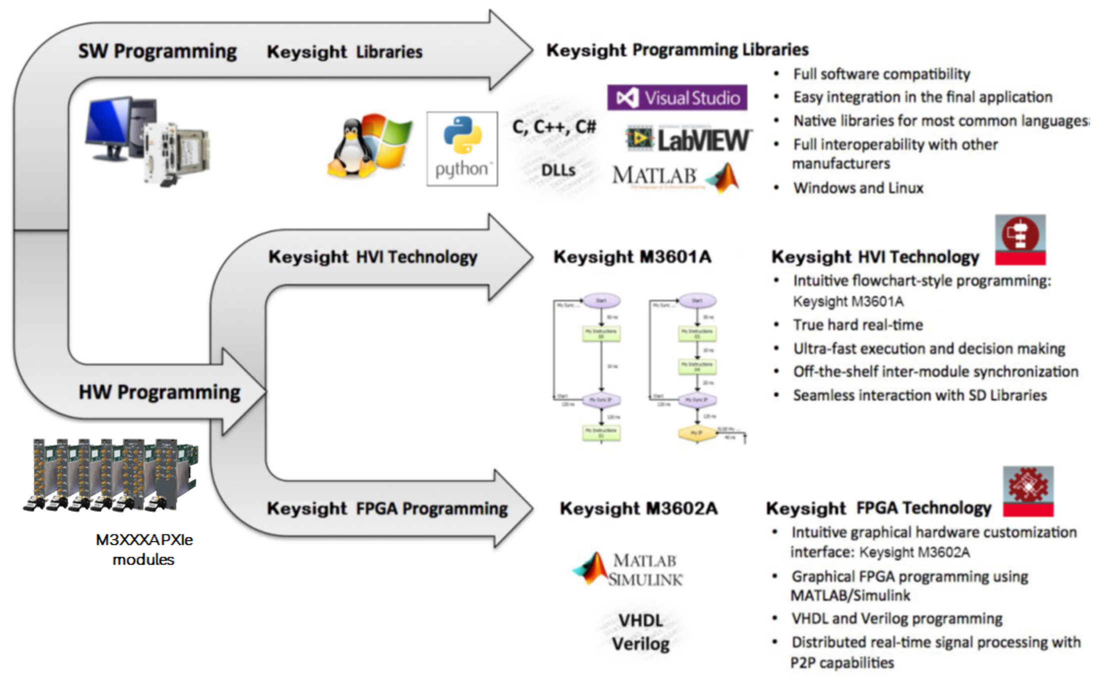


Figure 9: Programming tools for the following Keysight PXIe AWGs and digitizers: M3100A, M3102A, M3201A, M3202A, M3300A and M3302A

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 provides also 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, like visualization.

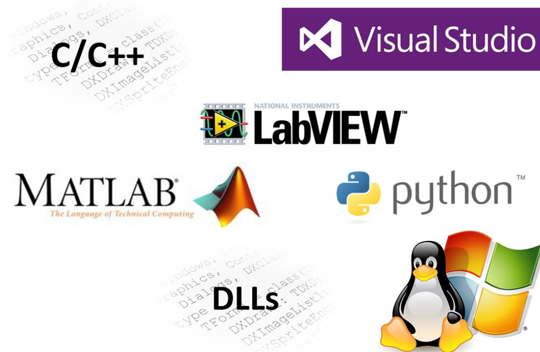


Figure 10: Keysight native programming libraries ensure full compatibility, providing effortless and seamless software integration and user interaction, etc. The I/O modules run in parallel, completely synchronized, and exchange data and decisions in real-time. The result is a set of modules that behave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

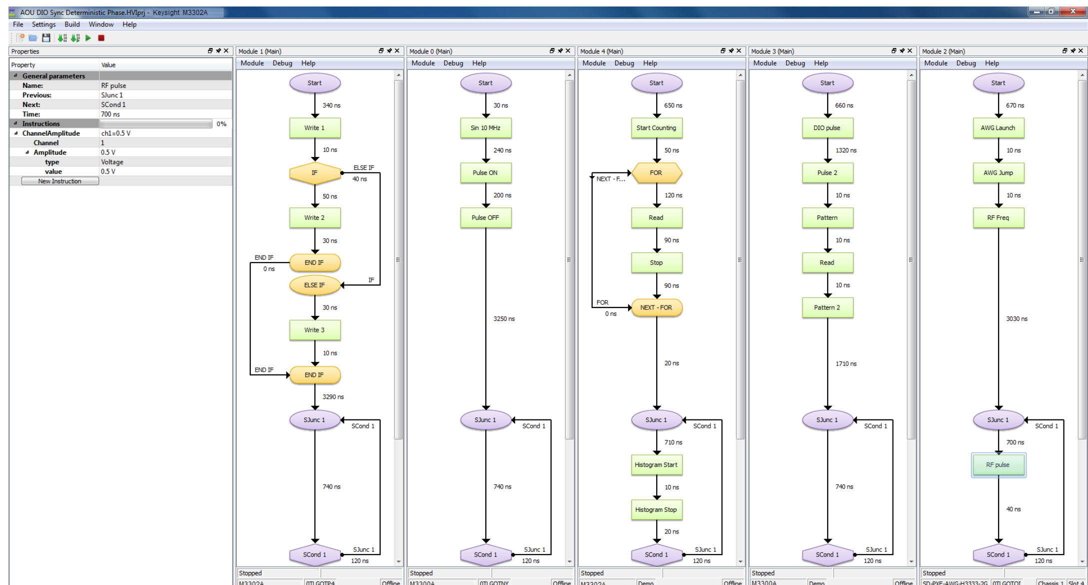


Figure 11: Keysight M3601A, a user-friendly flowchart-style HVI programming environment

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 programming environment that makes HVI programming extremely fast and easy (Figure 12). 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). 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 (see HVI Technology: Keysight M3601A (page 42))
- 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.

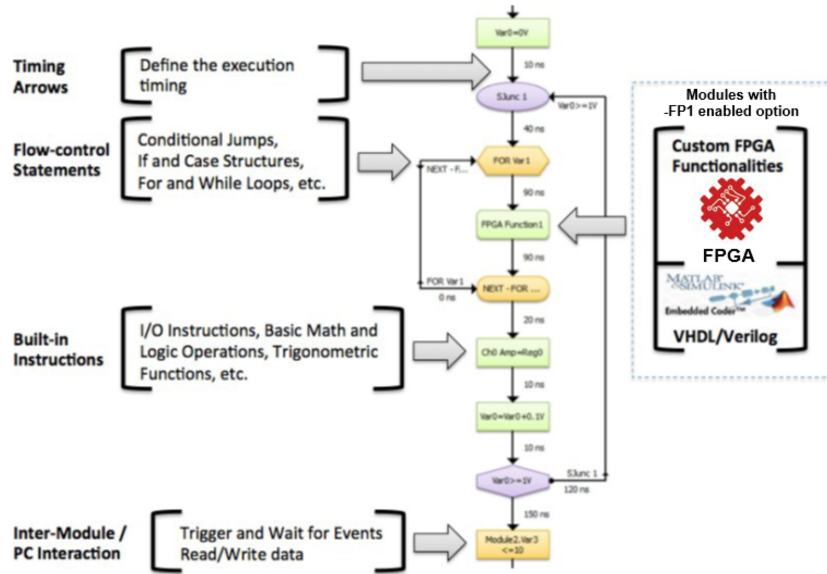


Figure 12: HVI flowchart elements. Keysight M3601A is based on flowchart programming, providing an easy-to-use environment to develop hard real-time applications

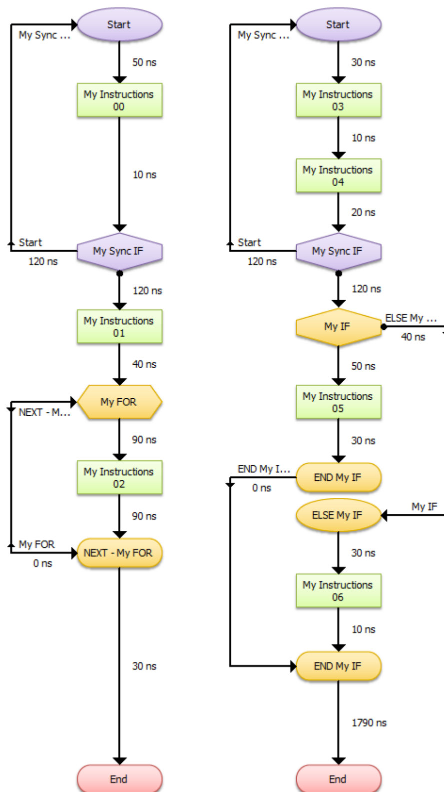


Figure 13: HVI example with two hardware modules. In an HVI, all Keysight modules run in parallel and completely synchronized, executing one flowchart per module. This results in simpler systems without the need of triggers.

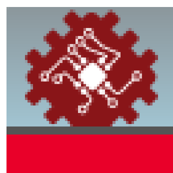
3. 1. 2. 2 FPGA Programming: Keysight M3602A

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.



M3602A

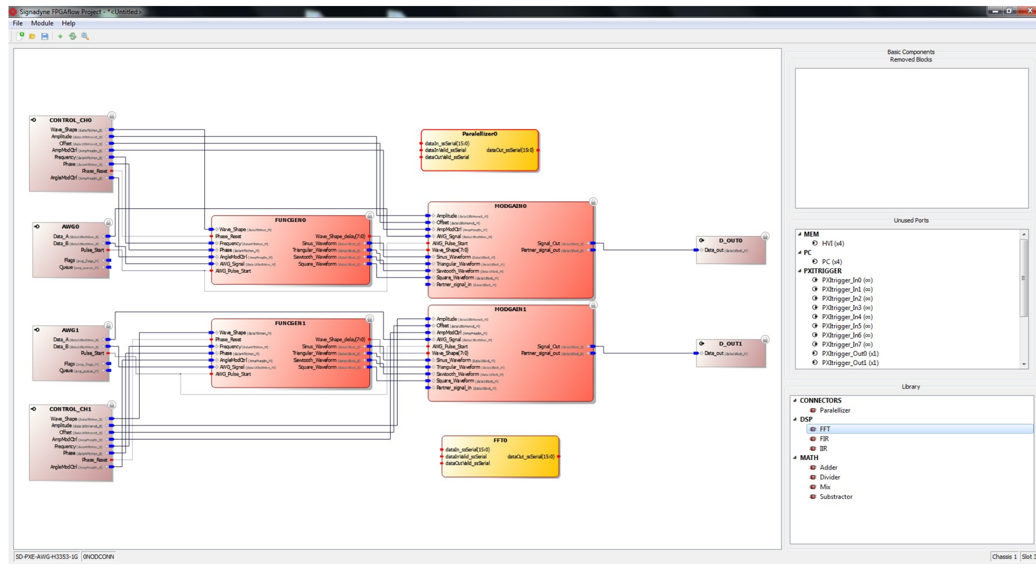


Figure 14: Keysight M3602A provides an intuitive graphical FPGA customization interface

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.

3. 1. 2. 3 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.

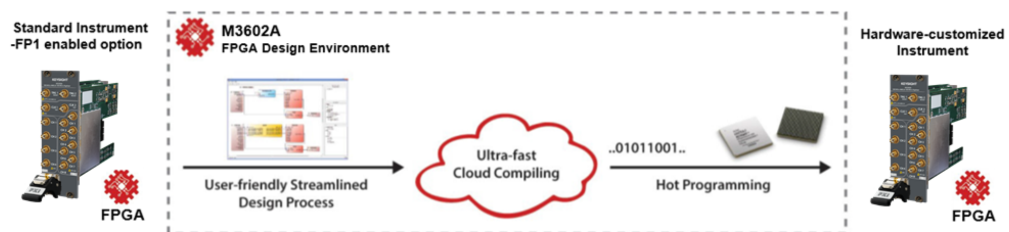


Figure 15: Keysight M3602A: a platform that provides the complete flow from design to FPGA programming

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 [3], a ready-to-use software front panels for live operation. When SD1 SFP opens, it identifies all Keysight hardware connected to the computer, opening the corresponding front panels.

SD1 SFP

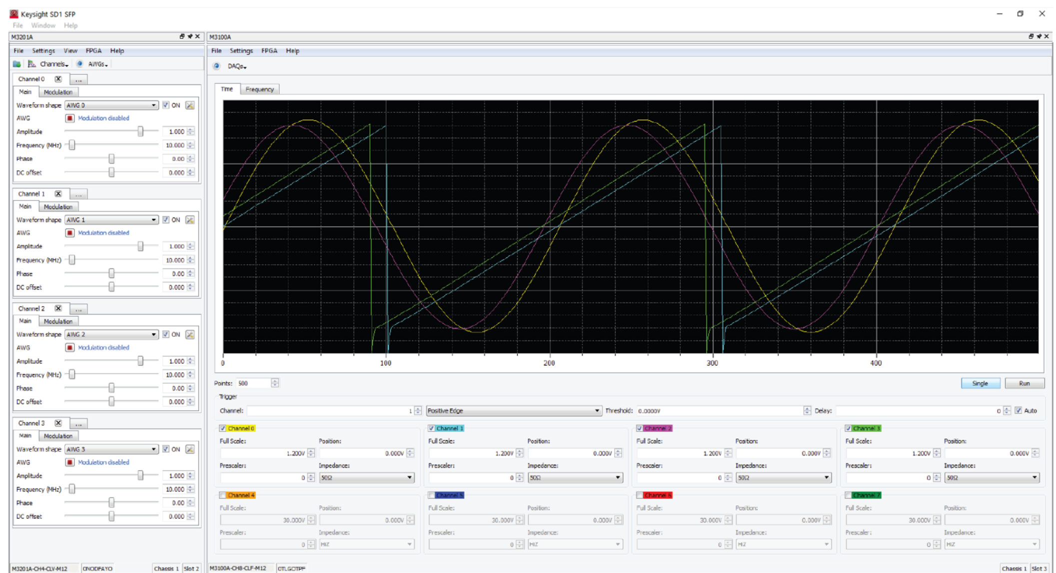


Figure 16: Keysight SD1 SFP provides software front panels, a fast and intuitive way of operating any Keysight hardware