

PathWave Test Sync Executive Transitioning from M3601A to KS2201A

PATHWAVE

This transition guide explains how to translate an existing project designed with Keysight M3601A HVI (Hard Virtual Instrument) Design Environment into an equivalent implementation based on the HVI Python API (Application Programming Interface) provided by Keysight KS2201A PathWave Test Sync Executive. An example project, showcasing all the possible M3601A functionalities, is used to provide transition guidelines. Every HVI statement contained in the example project is translated using the HVI API, hence providing to the user the Python code snippets necessary to translate every block of an M3601A flowchart into an equivalent implementation based on the HVI Python API. By following the provided guidelines, users can translate their own M3601A project and obtain an equivalent implementation compatible with Keysight KS2201A PathWave Test Sync Executive

Table of Contents

Transitioning from M3601A HVI Programming Environment to KS2201A PathWave Test Sync Executive	4
Introduction	4
Software and Firmware Versions	5
M3601A HVI (Hard Virtual Instrument) Design Environment Project Example	5
Guidelines to Transition from an M3601A HVI Project to a KS2201A PathWave Test Sync Executive Implementation	7
KS2201A PathWave Test Sync Executive Use Model	8
System Definition	10
Define HVI Constants	10
Define Platform Resources: Chassis, PXI triggers	11
Define HVI engines	12
Define HVI actions, events, triggers	12
Program HVI Sequence	13
Define HVI Registers	14
Start (a)	15
Synchronized While	15
Synchronized Multi-Sequence Block (SMSB) (a)	16
Wait Statement (b)	17
IF-ELSEIF-ELSE Statement (c)	18
HVI Instrument-Specific Instruction: Change AWG Amplitude using an FPGA Register	19
HVI Instrument-Specific Instruction: Change AWG Frequency using an HVI Constant	20
Sync Register Sharing	20
HVI Instrument-Specific Instruction: Queue Waveform (i)	21
Action Execute: AWG Trigger (j)	23
Wait Time (l)	24
End of Sequence (n)	25
Compile, Load, Execute the HVI	25
Compile HVI	26
Load HVI to Hardware	27
Execute	27
Release Hardware	27
Further HVI API Explanations	28
Comparison of M3601A HVI GUI and KS2201A HVI API Use Models	28
Instrument Compatibility	30

Conclusions31

Transitioning from M3601A HVI Programming Environment to KS2201A PathWave Test Sync Executive

This transition guide explains how to translate an existing application designed with Keysight M3601A HVI (Hard Virtual Instrument) Design Environment into an equivalent implementation based on the HVI Python API (Application Programming Interface) provided by Keysight KS2201A PathWave Test Sync Executive. An example project, showcasing all the possible M3601A functionalities, is used to provide transition guidelines. Every HVI statement contained in the example project is translated using the HVI API, hence providing to the user the Python code snippets necessary to translate every block of an M3601A flowchart into an equivalent implementation based on the HVI Python API. By following the provided guidelines, users can translate their own M3601A project and obtain an equivalent implementation compatible with Keysight KS2201A PathWave Test Sync Executive.

NOTE PathWave Test Sync Executive (KS2201A) is based on the next generation of HVI technology and it is **NOT compatible** with the previous version M3601A. If you are using M3601A, you must migrate your applications based on the M3601A GUI to the completely new HVI API. To be able to use KS2201A, you must also update your instrument's firmware and software drivers to newer versions with support for KS2201A. Please check the recommended M3xxxA firmware and software versions in the *SD1 3.x Software Startup Guide* available on www.keysight.com.

Introduction

KS2201A PathWave Test Sync Executive Development Software is a new API based environment for developing and running programs with a new generation of Keysight's Hard Virtual Instrument (HVI) technology. KS2201A enables programmatic development and execution of synchronous real-time operations across multiple instruments. This initial release is only programmable via API. Graphical programming capability is planned, but specific availability timelines are yet to be defined. The new generation of HVI technology is only programmable by KS2201A PathWave Test Sync Executive and is not backward compatible with the previous generation of HVI technology.

The previous generation of HVI technology is only programmable by M3601A Hard Virtual Instrument (HVI) Design Environment and is not forward compatible with the new generation of HVI technology or PathWave Test Sync Executive.

This document is for existing users of M3601A. It is intended to help them transition to KS2201A and translate their existing M3601A projects to KS2201A projects.

The original generation of HVI (Hard Virtual Instrument) technology is programmable by the M3601A HVI Design Environment for development of HVI instances. The M3601A is graphical design environment employing a Graphical User Interface (GUI) to develop HVI instances using flowcharts. The M3601A does not have a programmatic interface - API - for program development. It requires users to load the HVI design files into their

code with no ability to integrate the whole application in a single code script. The M3601A GUI has limited capability to automatically deploy the same HVI sequence or sub-sequence on multiple HVI engines and instruments.

KS2201A PathWave Test Sync Executive implements the new generation of HVI technology and delivers the HVI Application Programming Interface (API). This was created to address strong user demand for a robust API programming paradigm. Using the HVI API, users can deploy HVI applications directly in their code and easily access any HVI resource by using code Variables. Design of HVI sequences over multiple modules can be carried out using FOR loops or other types of code loops. In addition to the API development model, several other features have been added, such as extended multi-chassis capabilities and expanded product support.

When transitioning from M3601A to KS2201A, it is important to note that the same fundamental design steps are involved, but how these steps are performed has changed. In both use models, you must list and define HVI resources (registers, trigger resources, etc.), define HVI execution sequences, compile the HVI, load them to HW, and execute them.

Software and Firmware Versions

The versions of software (SW) and FPGA firmware (FW) that were used for the M3601A HVI GUI example described in this document are listed below.

1. Software versions used:

- Keysight SD1 Drivers, Libraries and SFP (v2.01.50)
- Keysight M3601A Hard Virtual Instrument (HVI) Design Environment (v2.01.46)

2. M3xxxA with -HVx HW option and following FPGA firmware versions:

- M3202A AWG FPGA firmware (v3.73.00)
- M3201A AWG FPGA firmware (v3.69.00)

To translate the M3601A HVI GUI example presented in this document into an implementation based on KS2201A HVI API, a different version of both Keysight SD1 SFP software must be installed. Once done FPGA FW of M3xxxA PXI instruments can be instead programmed using the "Hardware Manager" window of SD1 Software Front Panel (SFP). Please review the "Instrument Compatibility" section of this document to get the exact instrument SW and FPGA FW versions necessary to move to implementations based on KS2201A HVI API. KS2201A PathWave Test Sync Executive installer and documentation are available at www.keysight.com/find/KS2201A-downloads.

M3601A HVI (Hard Virtual Instrument) Design Environment Project Example

The figure below represents the project example that is the focus of this transition guide. The instruments used in the example are two M320xA AWG (Arbitrary Waveform Generators) of Keysight M3xxxA family of PXI instruments. The example showcases the following functionalities:

- Off-shelf inter-instrument synchronization capabilities.
- Synchronized While Statement to run synchronized loops over multiple instruments.
- Wait Event statement.
- Wait Time statement.
- Example of Local Flow Control functionality: IF statement.
- Sync Register Sharing functionality to real-time share register information through different instruments.
- HVI native instruction examples: increment FPGA (Field Programmable Gate Array) registers, execute AWG trigger actions.
- HVI product-specific instruction examples: change AWG frequency using an HVI constant, change AWG amplitude using an FPGA register.

The rest of this transition guide explains how these functionalities can be equivalently implemented using KS2201A PathWave Test Sync Executive. For further details on the functionalities please consult the M3601A HVI Design Environment User Guide and the KS2201A PathWave Test Sync Executive User Manual, available at www.keysight.com/find/KS2201A-downloads.

The image displays a Keysight M3601A PathWave Test Sync Executive implementation. The central part of the image shows a flowchart with two main modules, Module 0 (Main) and Module 1 (Main). The flowchart starts with a 'Start' node, followed by a 'Wait FP trigger high' block (10 ns), then a 'Wait FP trigger low' block (10 ns). A decision diamond 'IF' (40 ns) checks 'ChAmplitude = RegA'. If true, it goes to 'ChFrequency = FreqA' (10 ns), then 'END IF' (20 ns), then 'Share W...' (110 ns), 'Queue with WfNum' (2 us), 'AWG Trigger' (10 ns), 'cycleCnt++' (10 ns), and 'Loop Delay' (100 ns + [?]). If false, it goes to 'ChAmplitude = RegB' (10 ns), 'ChFrequency = FreqB' (10 ns), 'END IF' (40 ns + [?]), 'Share W...' (110 ns), 'Queue with WfNum' (2 us), 'AWG Trigger' (10 ns), 'cycleCnt++' (10 ns), and 'Loop Delay' (10 ns + [?]). Both paths lead to a 'Jump' block (200 ns) and finally 'End 1' (120 ns).

Surrounding the flowchart are several property windows, each with a red border and a letter label (a-n):

- a.** Properties for 'Jump 0'. Name: Start, Previous: --, Next: Wait PXI trigger high, Time: 0 ns.
- b.** Properties for 'Wait FP trigger high'. Name: Wait FP trigger high, Previous: Start, Next: Wait FP trigger low, Time: 10 ns.
- c.** Properties for 'IF'. Name: IF, Previous: Wait PXI trigger low, Next: ChAmplitude = RegA, Time: 30 ns to ?.
- d.** Properties for 'ChAmplitude = RegA'. Name: ChAmplitude = RegA, Previous: IF, Next: ChFrequency = FreqA, Time: 70 ns to ?.
- e.** Properties for 'ChFrequency = FreqA'. Name: ChFrequency = FreqA, Previous: ChAmplitude = RegA, Next: END IF, Time: 80 ns to ?.
- f.** Properties for 'ChAmplitude = RegB'. Name: ChAmplitude = RegB, Previous: ELSE IF, Next: ChFrequency = FreqB, Time: 80 ns to ?.
- g.** Properties for 'ChFrequency = FreqB'. Name: ChFrequency = FreqB, Previous: END IF, Next: END IF, Time: 90 ns to ?.
- h.** Properties for 'Share WfNum'. Name: Share WfNum, Previous: END IF, Next: Queue with WfNum, Time: 140 ns to ?.
- i.** Properties for 'Queue with WfNum'. Name: Queue with WfNum, Previous: Share WfNum, Next: AWG Trigger, Time: 2.11 us.
- j.** Properties for 'AWG Trigger'. Name: AWG Trigger, Previous: Queue with WfNum, Next: cycleCnt++, Time: 2.13 us.
- k.** Properties for 'cycleCnt++'. Name: cycleCnt++, Previous: AWG Trigger, Next: Loop Delay, Time: 2.23 us to ?.
- l.** Properties for 'Loop Delay'. Name: Loop Delay, Previous: cycleCnt++, Next: Jump, Time: 2.13 us.
- m.** Properties for 'Jump'. Name: Jump, Previous: Loop Delay, Next: End 1, Time: 2.23 us to ?.
- n.** Properties for 'End 1'. Name: End 1, Previous: Jump, Next: --, Time: 120 ns.

Guidelines to Transition from an M3601A HVI Project to a KS2201A PathWave Test Sync Executive Implementation

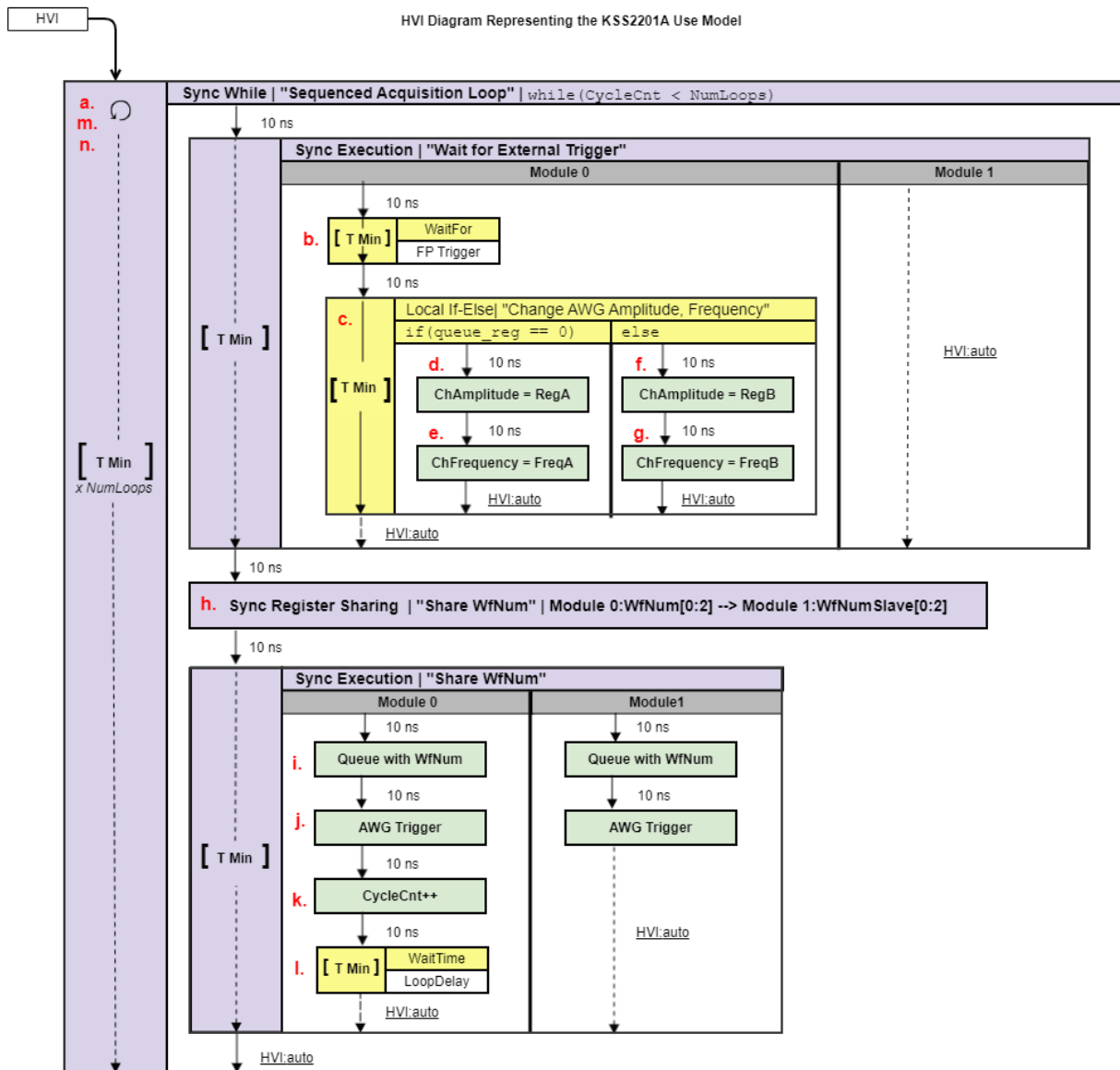
This section explains how each of the measurement actions of this application can be implemented using the HVI Application Programming Interface (API) provided by PathWave Test Sync Executive. Every measurement action is presented below with a letter referencing to the equivalent piece in the M3601A flow-chart, to facilitate user transition between the two different use models.

KS2201A PathWave Test Sync Executive Use Model

Differently than M3601A, KS2201A PathWave Test Sync Executive does not have a GUI in its 2020 release. However, there are graphical representation conventions that are used to represent by means of an HVI diagram the HVI sequences programmed using the HVI API. More details on these graphical conventions are explained in the "HVI Diagrams" section of the **KS2201A PathWave Test Sync Executive User Manual** available at www.keysight.com/find/KS2201A-downloads.

The HVI sequences that can be programmed using the KS2201A HVI API are based on the new concepts of sync sequences and sync statements that were not defined in M3601A. M3601A blocks like the "Start" and "Share WfNum" Sync Junctions and the "Jump" Synchronized Conditionals are absent in the KS2201A use model. Their functionality is replaced by two types of sync statements: Sync While and Sync Multi-Sequence Block (SMSB). The correspondence is highlighted by means of letters in the HVI diagram below, to match the M3601A blocks with the corresponding KS2201A statements that have the same functionalities. For further details about sync sequences, sync statements and the KS2201A use model please consult the "HVI Elements" section of the **KS2201A PathWave Test Sync Executive User Manual**.

Finally please note that the timing constraints have also changed between M3601A and KS2201A. KS2201A timing constraints are explained in the section "HVI time management and latency" of the of the KS2201A PathWave Test Sync Executive User Manual. In the HVI diagram below the label "HVI auto" is used whenever HVI automatically calculates the time necessary to match the execution of different local sequences executed by different instruments in parallel within the same SMSB. The minimum start delay between consecutive statements is given by the instrument FPGA clock period, which corresponds to 10 ns Keysight M3xxxA instruments. For further details please consult the "HVI Timing" section of the KS2201A PathWave Test Sync Executive User Manual.



NOTE: 10 ns is the FPGA clock period for M3xxxA instruments

When transitioning from M3601A to the new KS2201A API use model, one of the differences is that the API programming requires to follow some steps in a well-defined order. In particular, three fundamental steps shall be followed:

1. System definition: defines all the necessary HVI resources, including platform resources, engines, triggers, registers, actions, events, etc.
2. Program HVI sequences: defines all the statements to be executed within each HVI sequence
3. Execute HVI: compiles, loads to HW and executes HVI

The following sub-sections describe in details how the M3601A example presented earlier can be translated following each step.

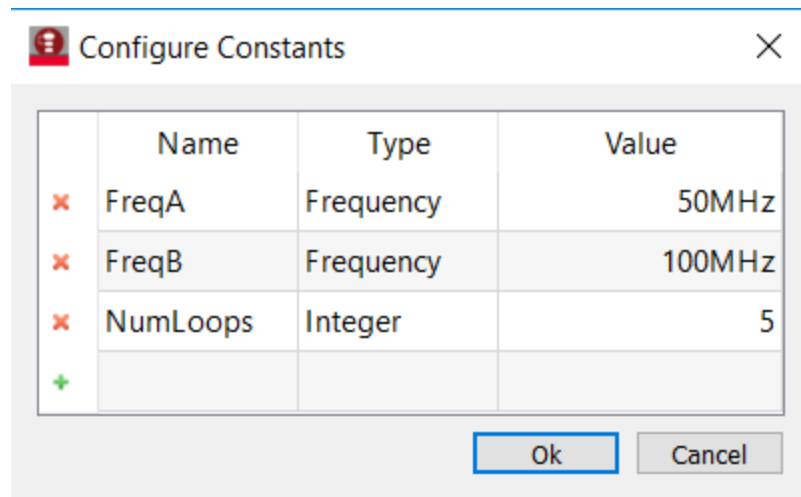
System Definition

The definition of HVI resources is the first step of a creating an HVI instance. HVI resources include all the platform resources, engines, triggers, registers, actions, events, etc. that the HVI sequences are going to use and execute. Users need to declare them upfront and add them to the corresponding collections. In M3601A the HVI resources can be defined through different windows of the GUI, as illustrated in the rest of this section. In KS2201A API, the SystemDefinition class must be used to define all the HVI resources before users can move on to program and execute the HVI sequences.

Define HVI Constants

In M3601A HVI GUI a "Configure Constants" window allows to define constant parameters to be used as Variable within the M3601A flow-chart of each specific HVI engine. It is important to note that in the KS2201A HVI API the HVI constants do not exist any more because their function is replaced by user-defined Python code Variables that can for example be collected into an application parameters class as in the code snippet below.

M3601A API



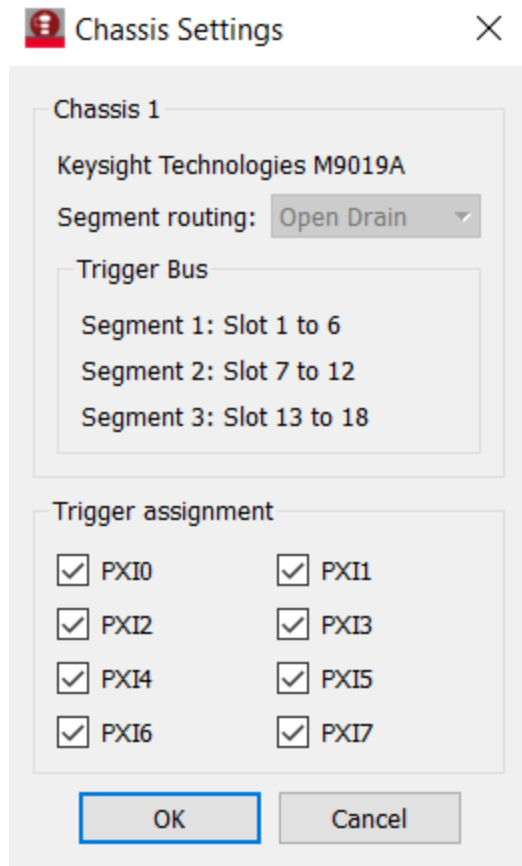
Python

```
class ApplicationParameters:  
    # Configures the application parameters  
    def __init__(self):  
        self.FreqA = 50e6 # [Hz]  
        self.FreqB = 100e6 # [Hz]  
        self.NumLoops = 5
```

Define Platform Resources: Chassis, PXI triggers

In M3601A HVI GUI the "Chassis Settings" window allows to define both the chassis and PXI trigger resources to be used by HVI. In KS2201A HVI API all HVI instances need to define the chassis and eventual chassis interconnections using the SystemDefinition class. PXI trigger lines to be reserved by HVI for its execution can be assigned using the sync_resources interface of the SystemDefinition class. See M3601A screenshot below and KS2201A API code snippet below for comparison of the two implementations.

M3601A GUI



KS2201A API

```
# Create system definition object
sys_def = kthvi.SystemDefinition("MyMultiChassisSystemDefinition")

# Add chassis resources
sys_def.chassis.add_with_options(1, 'Simulate=True,DriverSetup=model=M9018B,NoDriver=True')
# or sys_def.platform.chassis.add(chassis_number)
# or sys_def.platform.chassis.add_auto_detect()
```

```
# Assign triggers to HVI object to be used for synchronization, data sharing, etc
# NOTE: In a multi-chassis setup ALL the PXI lines listed below need to be shared
# among each M9031 board pair by means of SMB cable connections
pxi_sync_trigger_resources = [
    kthvi.TriggerResourceId.PXI_TRIGGER0,
    kthvi.TriggerResourceId.PXI_TRIGGER1,
    kthvi.TriggerResourceId.PXI_TRIGGER2,
    kthvi.TriggerResourceId.PXI_TRIGGER3]

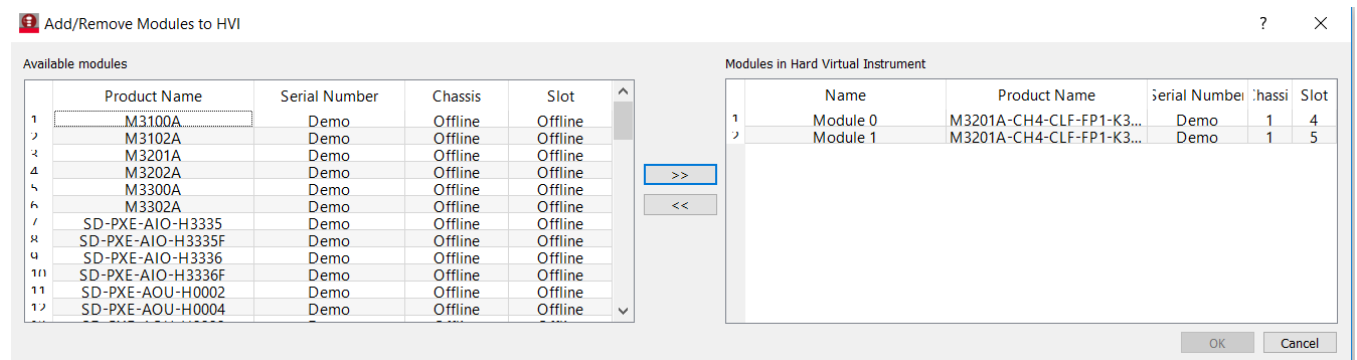
# Assign the defined PXI trigger resources
sys_def.sync_resources = pxi_sync_trigger_resources
```

Define HVI engines

The window "Add/Remove Modules to HVI" allows to add to the HVI instance the necessary modules and their HVI engines.

In KS2201A HVI API all HVI Engines included in the application need to be registered into the *EngineCollection* class instance. The HVI resource definitions are summarized in the code snippets below.

M3601A GUI



KS2201A API

```
# Create system definition object
sys_def = kthvi.SystemDefinition("MyMultiChassisSystemDefinition")

# For each instrument to be used in the HVI application add its HVI Engine to the HVI
# Engine Collection
engine_Names = []
for engine_index in range(len(module_list)):
    engine_Names.append(f'AwgEngine{engine_index}')
    sys_def.engines.add(module_list[engine_index].hvi.engines.main_engine, engine_Names
[engine_index])
```

Define HVI actions, events, triggers

The HVI API implementation of the example presented in this transition guide requires also to register AWG trigger actions in the HVI actions collection and the Front Panel trigger into the HVI trigger collection, as shown in the last two code snippets presented below. In M3601A no equivalent GUI exists. For triggers, some of the settings can be set using the properties of statements that use triggers, e.g. the Wait Statement (block b. in the example that is the focus of this document).

M3601A GUI

- No equivalent GUI exists -

KS2201A API

```
# Create system definition object
sys_def = kthvi.SystemDefinition("MyMultiChassisSystemDefinition")

# For each AWG, define the list of HVI Actions to be executed and add such list to its own
HVI Action Collection
for engine_Name, module in zip(engine_Names, module_list):
    for ch_index in range(1, module.num_channels + 1):
        # Actions need to be added to the engine's action list so that they can be executed
        action_Name = "My AWG Trigger {}".format(ch_index) # arbitrary user-defined Name
        instrument_action = "awg{}_trigger".format(ch_index) # Name decided by instrument
API
        action_id = getattr(module.instrument.hvi.actions, instrument_action)
        sys_def.engines[engine_Name].actions.add(action_id, action_Name)

# Add to the HVI Trigger Collection of each HVI Engine the FP Trigger object of that same
instrument to be used as event for the Wait statement
for engine_Name, module in zip(engine_Names, module_list):
    fp_trigger_id = module.instrument.hvi.triggers.front_panel_1
    fp_trigger = sys_def.engines[engine_Name].triggers.add(fp_trigger_id, "My FP Trigger")
    # Configure FP trigger in each sys_def.engines[index]
    fp_trigger.config.direction = kthvi.Direction.OUTPUT
    fp_trigger.config.polarity = kthvi.Polarity.ACTIVE_HIGH
    fp_trigger.config.sync_mode = kthvi.SyncMode.IMMEDIATE
    fp_trigger.config.hw_routing_delay = 0
    fp_trigger.config.trigger_mode = kthvi.TriggerMode.LEVEL
    #NOTE: FP trigger pulse length is defined by the HVI Statements that control FP Trigger
ON/OFF
```

Program HVI Sequence

Once the HVI resources are defined, users can program the HVI sequence of measurement actions to be executed by each HVI engine. In KS2201A PathWave Test Sync Executive HVI sequences can be programmed using the *Sequencer* class. In KS2201A sequences are structured within a global sequence (defined by the *SyncSequence* class) that takes care of synchronizing and encapsulating the local sequences corresponding to each HVI engine included in the application. In M3601A there was no Sync Sequence concept and every flow-chart was representing the local sequence to be executed by the HVI engine of a specific instrument.

KS2201A API

```
# Create sequencer object from previously define SystemDefinition object
sequencer = kthvi.Sequencer("mySequencer", sys_def)
```

M3601A GUI

- No equivalent GUI exists -

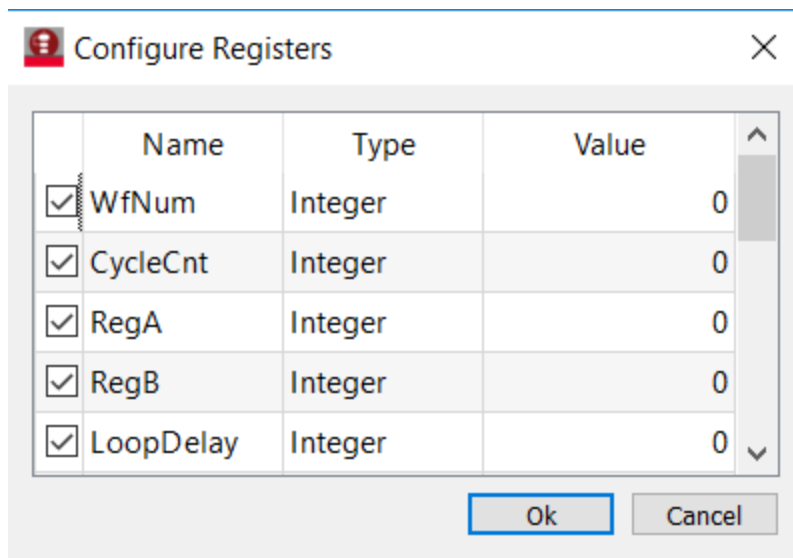
Define HVI Registers

HVI registers correspond to very fast access physical memory registers in the HVI Engine located in the instrument HW (e.g. FPGA or ASIC). HVI Registers can be used as parameters for operations and modified during the sequence execution (same as Variables in any programming language). The number and size of registers is defined by each instrument. The registers that users want to use in the HVI sequences need to be defined beforehand into the register collection within the scope of the corresponding HVI Sequence.

In M3601A HVI GUI each module allows opening a "Register Settings" window from where registers belonging to the HVI engine of that same module can be defined, reNamed and initialized.

In KS2201A HVI API this can be done using the *RegisterCollection* class that is within the Scope object corresponding to each sequence. HVI Registers belong to a specific HVI Engine because they refer to HW registers of that specific instrument. Register from one HVI Engine cannot be used by other engines or outside of their scope. Note that currently, registers can only be added to the HVI top SyncSequence scopes, which means that only global registers visible in all child sequences can be added. HVI registers are defined in this application by the code snippet below

M3601A GUI



KS2201A API

```
# Define Primary AWG registers
WfNum = hvi.sync_sequence.scopes[awg_engine].registers.add("WfNum",
kthvi.RegisterSize.SHORT)
WfNum.initial_value = 0
CycleCnt = hvi.sync_sequence.scopes[awg_engine].registers.add("CycleCnt",
kthvi.RegisterSize.SHORT)
CycleCnt.initial_value = 0
RegA = hvi.sync_sequence.scopes[awg_engine].registers.add("RegA", kthvi.RegisterSize.SHORT)
RegA.initial_value = 0
RegB = hvi.sync_sequence.scopes[awg_engine].registers.add("RegB", kthvi.RegisterSize.SHORT)
RegB.initial_value = 0
LoopDelay = hvi.sync_sequence.scopes[awg_engine].registers.add("LoopDelay",
kthvi.RegisterSize.SHORT)
LoopDelay.initial_value = 0
```

Start (a)

In HVI API the first synchronized junction called “Start” is always implicit and no code needs to be written to add it to the HVI sequences of the instruments involved in an HVI execution. All instruments start as synchronized within the HVI execution. In M3601A the instrument corresponding to the first HVI sequence that is created is the instrument that has the “master” role in the “Start” junction. This role was reNamed to "primary" in KS2201A. The primary/master modules send the signal to the other modules to synchronously start a sequence of measurement actions.

M3601A API

Property	Value
General parameters	
Name:	Start
Previous:	...
Next	Wait PXI trigger high
Time:	0 ns
HVI role	Slave
Input connections	
Jump 0:	Jump

Synchronized While

The loop implemented in the M3601A flow-chart by means of a combination of synchronized junctions (a. block) and synchronized conditional (m. block) is implemented in the HVI API using a Synchronized While (Sync While) statement. Sync While statements belongs to the set of HVI Sync Statements and are defined by the API class *SyncWhile*. A Sync While allows you to synchronously execute multiple local HVI sequences until a user-defined condition is met, that is, the sync while condition. For local sequences to be defined within the

Sync While, it is necessary to use a Synchronized Multi-Sequence Block (SMSB) that is explained in the details in the next sub-section.

M3601A GUI

Properties	
Property	Value
General parameters	
Name:	Jump
Previous:	Loop Delay
Next:	End 1
Time:	2.23 us to ?
Jump parameters	
True:	Start
False:	End 1
Decision role:	Master
Comparison CycleCnt < NumLoops	
Left operand [Re...]	CycleCnt
Left mask [Off]	
Comparison	<
Right operand [...]	NumLoops
Right mask [Off]	

KS2201A API

```
# Create sequencer object from previously define SystemDefinition object
sequencer = kthvi.Sequencer("mySequencer", sys_def)
```

```
# Define sync while condition
sync_while_condition = kthvi.Condition.register_comparison(iteration_counter,
kthvi.ComparisonOperator.LESS_THAN, rf_pulse_params.num_loops)
# Add Sync While Statement
sync_while = sequencer.sync_sequence.add_sync_while("Sequenced Acquisition Loop", 10, sync_
while_condition)
```

Synchronized Multi-Sequence Block (SMSB) (a)

SMSB is a type of HVI statements that was not present in the M3601A use model. Synchronized multi-sequence blocks are defined by the API class *SyncMultiSequenceBlock*. This type of sync statement synchronizes all the HVI engines that are part of the sync sequence. It allows you to program each HVI Engine to do specific operations by exposing a local sequence for each engine. By calling the API method *add_multi_sequence_block()* a synchronized multi-sequence block is added to the Sync (global) Sequence.

M3601A does not contain synchronized multi-sequence blocks because it is centered instead on a use model based on local sequences, each of them running locally on each HVI engine within each instrument. In M3601A local sequences are synchronized between each other by using synchronized junctions and conditionals. Instead, KS2201A provides a set of global sync statements, like the SMSB and the sync while, that encapsulate and synchronize pieces of local sequences executed by each instrument's HVI engine.

M3601A GUI

- No equivalent statement exists -

KS2201A API

```
# Add 1st Sync Multi-Sequence Block to the Sync While sequence
sync_block_1 = sync_while.sync_sequence.add_sync_multi_sequence_block("Wait for External Trigger", 10)
```

Wait Statement (b)

The wait statement is a local flow control statement that can be implemented using the API class *WaitStatement*. This sequence block sets an instrument to wait for a condition. The condition can be defined by a trigger, an event, or any combination of them through the usage of logical operators. In this application example, the wait is used to set the AWG to wait for a transition on the FP (Front Panel) trigger. The wait statement is set to wait for a trigger falling edge using the .wait mode *WaitMode.TRANSITION* combined with a trigger configuration as *ACTIVE_LOW*. The sync mode *SyncMode.IMMEDIATE* sets the wait event to let the execution continue immediately, i.e. as soon as the trigger event is received.

The difference in M3601A HVI GUI is that the implementation of this type of wait statement requires the combination of two flow-chart boxes whenever the user wants to wait for an event transition instead of an event value (high/low or active/inactive).

M3601A GUI

Properties	
Property	Value
<div style="border: 1px solid black; padding: 5px;"> <div style="display: flex; align-items: center;"> <div style="flex: 1;"> <p>▼ General parameters</p> <p>Name: Wait FP trigger high</p> <p>Previous: Start</p> <p>Next: Wait FP trigger low</p> <p>Time: 10 ns</p> </div> <div style="flex: 1;"> </div> </div> </div>	
<div style="border: 1px solid black; padding: 5px;"> <p>▼ Wait parameters (...)</p> <p>Trigger Source [...]: PXI Trigger 7</p> <p>Value: On</p> <p>Mode: Immediate</p> </div>	


KS2201A API

```
# Define the condition for the wait statement
wait_condition = kthvi.Condition.trigger(hvi.engines[awg_engine_Name].triggers["FP
Trigger"])
# Add a Wait For Event
primary_sequence = sync_block_1.sequences["AwgEngine0"]
wait_event = primary_sequence.add_wait("Wait for FP Trigger", 10, wait_condition)
wait_event.set_mode(kthvi.WaitMode.TRANSITION, kthvi.SyncMode.IMMEDIATE)
```

IF-ELSEIF-ELSE Statement (c)

IfStatement class allows you to add an IF-ELSEIF-ELSE loop within the main HVI sequence of any instrument engine. The IF-ELSEIF-ELSE loop contains one (or more) IF branches and an ELSE branch. The statements contained in each IF or ELSE branch are executed if the condition of each branch is met. The condition of each branch can be defined using the API class *ConditionalExpression*. Branch sub-sequence can be programmed using the same API methods and classes used to program the main HVI sequence, by means of the API classes *IfBranch* and *ElseBranch*. The IF-ELSEIF-ELSE statement in KS2201A is enhanced with a new additional functionality that allows to match the timing of each branch of the statement. For additional info on this functionality, please consult the section "HVI Timing" in the **KS2201A PathWave Test Sync Executive User Manual** available on www.keysight.com

M3601A GUI

Properties	
Property	Value
<div style="display: flex; align-items: center;"> C.  </div>	
General parameters	
Name:	IF
Previous:	Wait PXI trigger low
Next	ChAmplitude = RegA
Time:	30 ns to ?
Branch parameters	
Synchronization:	Force
Comparison	
Left operand [Re...]	WfNum
Left mask [Off]	
Comparison	==
Right operand [I...]	0
Right mask [Off]	

KS2201A API

```
# Configure IF condition
if_condition = kthvi.Condition.register_comparison(WfNum, kthvi.ComparisonOperator.EQUAL_
TO, 0)
# Set flag that enables to match the execution time of all the IF branches
enable_ifbranches_time_matching = True
```

```

# Add If statement
primary_sequence = sync_block_1.sequences["AwgEngine0"]
if_statement = primary_sequence.add_if("My If Statement", 10, if_condition, enable_
ifbranches_time_matching)
# Program IF branch
if_sequence = if_statement.if_branch.sequence
# Add statements in if-sequence
instruction = if_sequence.add_instruction("ChAmplitude = RegA", start_delay,
module.hvi.instructions.set_amplitude.id)
instruction.set_parameter(...)
...
# Else-branch
# Program Else branch
else_sequence = else_branch.sequence
# Add statements in Else-sequence
instruction = else_sequence.add_instruction("ChAmplitude = RegB", start_delay,
module.hvi.instructions.set_amplitude.id)
...

```

HVI Instrument-Specific Instruction: Change AWG Amplitude using an FPGA Register

Blocks d and F execute a product-specific HVI instruction. Product-specific HVI instruction can be added in M3601A using the green instruction blocks. In KS2201A API method *add_instruction()* allows you to add the wanted instruction within the HVI sequence. Instruction parameters are set using the API method *set_parameter()*. All HVI product-specific instructions and parameters are defined in the *hvi.instruction_set* interface of each product. Instructions, actions, events and in general all the HVI definitions specific of M3xxxA instruments can be found in the **M320xA PXI AWGs User Guide** available on www.keysight.com.

M3601A GUI

The screenshot shows the GUI for the M3601A instrument. On the left is a table of properties for an instruction. On the right is a diagram of a green instruction block labeled 'd.' with the text 'ChAmplitude = RegA' inside. Arrows indicate the flow of data from the block to the table and vice versa.

Properties	
Property	Value
General parameters	
Name:	ChAmplitude = RegA
Previous:	IF
Next:	ChFrequency = FreqA
Time:	70 ns to ?
Instructions	
ChannelAmplitude ch1=RegA	
Channel	1
Amplitude [Re...	RegA

KS2201A API

```
# Set CH1 amplitude to ON_value
instruction = if_sequence.add_instruction("ChAmplitude = RegA", 10,
module.hvi.instructions.set_amplitude.id) instruction.set_parameter
(module.hvi.instructions.set_amplitude.channel.id, channel_number)
instruction.set_parameter(module.hvi.instructions.set_amplitude.value.id, RegA)
```

HVI Instrument-Specific Instruction: Change AWG Frequency using an HVI Constant

Blocks e. and g. in the M3601A flow-chart execute product-specific HVI instructions. Both are AWG specific instructions that can change the frequency value of the specified AWG channel to the value specified by the HVI constants "FreqA" or "FreqB". HVI constants do not exist in the HVI API use model because the HVI API Python code is completely integrated into the rest of the application code and Python code Variables can be used to replace the HVI constants of M3601A.

M3601A GUI

The screenshot shows a GUI window with a table of properties and a flowchart block. The table is as follows:

Properties	
Property	Value
General parameters	
Name:	ChFrequency = FreqA
Previous:	ChAmplitude = RegA
Next	END IF
Time:	80 ns to ?
Instructions	
ChannelFrequency	ch1=FreqA
Channel	1
Frequency [Co...	FreqA

To the right of the table is a flowchart block labeled 'e.' containing the text 'ChFrequency = FreqA'. Arrows indicate flow into and out of the block.

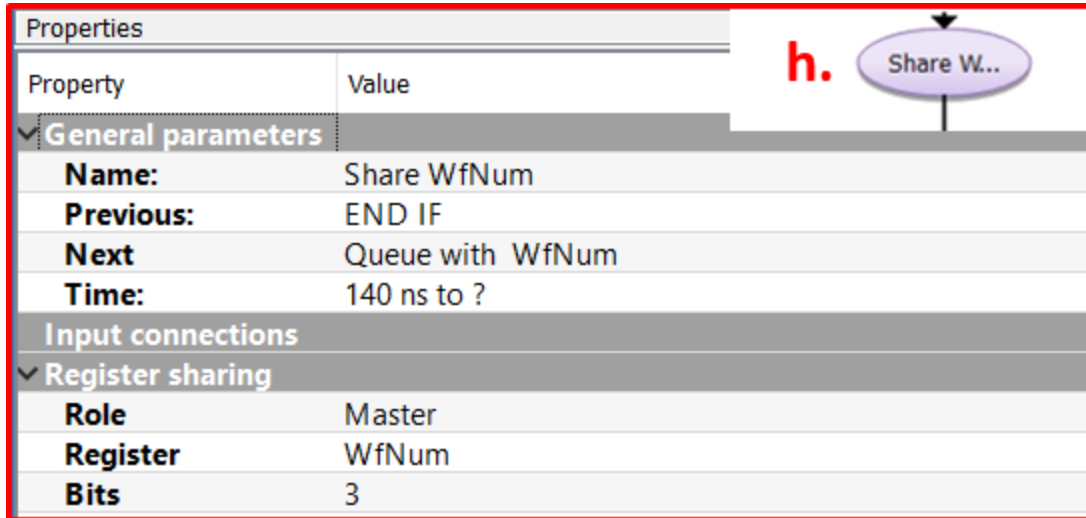
KS2201A API

```
# Set CH1 Frequency to FreqA
instruction = if_sequence.add_instruction("ChFrequency = FreqA", 10,
module.hvi.instructions.set_frequency.id)
instruction.set_parameter(module.hvi.instructions.set_frequency.channel.id, channel_number)
instruction.set_parameter(module.hvi.instructions.set_frequency.value.id, FreqA)
```

Sync Register Sharing

Register sharing (h) is a functionality that has an equivalent implementation in M3601A HVI GUI and KS2201A HVI API. In M3601A register sharing can be enabled within any Synchronized Junction block. In the KS2201A HVI API, it is defined and programmed using the *RegisterSharing* class. Register sharing allows to share the content of N adjacent bits of a source register and write the information to a destination register in any of the other HVI engines included in the HVI execution. In this application note this functionality is used to share the content of the primary/master AWG register *WfNum* and write into the slave AWG register *WfNumSlave* to use it to select real-time the waveform to be played by both AWGs at each experiment step.

M3601A GUI



Properties	
Property	Value
General parameters	
Name:	Share WfNum
Previous:	END IF
Next:	Queue with WfNum
Time:	140 ns to ?
Input connections	
Register sharing	
Role:	Master
Register:	WfNum
Bits:	3

KS2201A API

```
# AWG registers
WfNum = sequencer.sync_sequence.scopes["AwgEngine0"].registers["WfNum"]
WfNumSlave = sequencer.sync_sequence.scopes["AwgEngine1"].registers["WfNumSlave"]
```

```
# Add sync register sharing
bits_to_share = 3
sync_while_2.sync_sequence.add_sync_register_sharing("Share WfNum", 10, WfNum, WfNumSlave,
bits_to_share)
```

HVI Instrument-Specific Instruction: Queue Waveform (i)

The waveform ID shared from the primary AWG to the slave AWG using the register sharing functionality can be used to queue and play the corresponding waveform from both AWGs. All the necessary AWG parameters can be defined in a separated class as the class `AWG_parameters` reported below. For information about the definition and usage of M320xA AWG parameters please consult the [M320xA PXI AWGs User Guide](#) available on www.keysight.com.

Both in M3601A and in KS2201A the operation of queuing waveform is performed by using an instrument-specific instructions. HVI instrument specific instructions are provided in the KS2201A use model by the instrument-specific HVI definitions, documented in the instrument user guide. For more info on the M320xA AWG instrument-specific HVI definitions please consult the [M320xA PXI AWGs User Guide](#).

M3601A GUI

Properties	
Property	Value
<div style="display: flex; align-items: center;"> i. <div style="border: 1px solid black; padding: 2px; background-color: #e0f0e0; text-align: center;"> Queue with WfNum </div> </div>	
<div style="background-color: #cccccc; padding: 2px;"> General parameters </div>	
Name:	Queue with WfNum
Previous:	Share WfNum
Next:	AWG Trigger
Time:	110 ns
<div style="background-color: #cccccc; padding: 2px;"> Instructions </div>	
<div style="background-color: #cccccc; padding: 2px;"> queueWaveform 1 <= WFWfNum(Software trigger start, Dly=0s, # 1, P... </div>	
WG	1
Waveform nu...	WfNum
Trigger mode	Software trigger start
Start delay [Ti...	0s
Cycles [Integer]	1
Prescaler [Inte...	0

KS2201A API

```

class AWG_parameters:
    """ Configures AWG for waveform generation"""
    def __init__(self):
        self.all_ch_mask = 0xF # binary mask defining which channels to use
        # AWG settings for all channels
        self.sync_mode = keysightSD1.SD_SyncModes.SYNC_NONE
        self.queue_mode = keysightSD1.SD_QueueMode.ONE_SHOT
        self.awg_mode = keysightSD1.SD_Waveshapes.AOU_SINUSOIDAL
        self.start_delay = 0 # x10 [ns]
            self.awg_ch = 1
        self.prescaler = 0
        self.wfm_cycles = 2
        self.amplitude = 1 # [V]
        self.offset = 0 # [V]
        # Trigger settings
        self.trigger_mode = keysightSD1.SD_TriggerModes.SWHVITRIG_CYCLE

# AWG parameters
awg_params = AWG_parameters()
# Queue waveform to AWG CH1
instruction0 = awg_sequence.add_instruction("Queue Wfm with WfNum at CH1", 10,
module.hvi.instruction_set.queue_waveform.id)
#Set every parameter of AWGqueueWaveform(awg_ch, waveformNumber, triggerMode, startDelay,
cycles, prescaler)
instruction0.set_parameter(module.hvi.instruction_set.queue_waveform.waveform_number.id,
WfmNum)
instruction0.set_parameter(module.hvi.instruction_set.queue_waveform.channel.id, awg_
paramsawg_ch)
instruction0.set_parameter(module.hvi.instruction_set.queue_waveform.trigger_mode.id, awg_
params.trigger_mode)

```

```

instruction0.set_parameter(module.hvi.instruction_set.queue_waveform.start_delay.id, awg_
params.start_delay)
instruction0.set_parameter(module.hvi.instruction_set.queue_waveform.cycles.id, awg_
params.wfm_cycles)
instruction0.set_parameter(module.hvi.instruction_set.queue_waveform.prescaler.id, awg_
params.prescaler)

```

Action Execute: AWG Trigger (j)

In M3601A HVI GUI AWG trigger actions can be executed by adding the corresponding instruction "AWG trigger" from the AWG-specific instruction set.

In KS2201A HVI API the use model to execute instrument actions has changed with respect to M3601A HVI GUI. Actions to be used within an HVI sequence need to be added to the instrument HVI engine using the API "add()" method of the *ActionCollection* class. Once the wanted actions are added within the list of the instruments' HVI engine actions, an instruction to execute them can be added to the instrument's HVI sequence using the HVI API class *InstructionsActionExecute*. One or multiple actions can be executed at the same time within the same "Action Execute" instruction.

M3601A GUI

Properties	
Property	Value
<div style="display: flex; align-items: center;"> j. <div style="border: 1px solid black; padding: 5px; background-color: #d9ead3;">AWG Trigger</div> </div>	
<div style="background-color: #f2f2f2; padding: 2px;"> General parameters </div>	
Name:	AWG Trigger
Previous:	Queue with WfNum
Next:	cycleCnt++
Time:	2.11 us
<div style="background-color: #f2f2f2; padding: 2px;"> Instructions </div>	
<div style="background-color: #f2f2f2; padding: 2px;"> AWGtrigger </div>	
	Trigger(1)
WG	1

KS2201A API

```

# AWG local sequences can be accessed from within the Sync Multi-Sequence Block
primary_sequence = sync_block_1.sequences["AwgEngine0"]
# List of previously defined actions
awg_trigger_list = primary_sequence.engine.actions
# AWG trigger
inst_trigger = primary_sequence.add_instruction("AWG Trigger", 10, hvi.instruction_
set.action_execute.id)
inst_trigger.set_parameter(hvi.instruction_set.action_execute.action.id, awg_trigger_list)

```

HVI Native Instruction: Register Increment (k)

In M3601A HVI GUI registers can be incremented using the correspondent math instruction available within the HVI instruction set.

In KS2201A HVI API a register increment can be implemented within an HVI sequence using an instance of the API instruction class *InstructionsAdd*. The same instruction can be used to add registers and constant values (operands) and put the result in another register (result). The register to be incremented needs to be added previously to the scope of the corresponding HVI engine.

M3601A GUI

Properties	
Property	Value
<div style="display: flex; align-items: center;"> k. <div style="border: 1px solid green; padding: 5px; text-align: center;"> ↓ cycleCnt++ </div> </div>	
<div style="background-color: #f0f0f0; padding: 2px;"> ✓ General parameters </div>	
Name:	cycleCnt++
Previous:	AWG Trigger
Next	Loop Delay
Time:	2.12 us
<div style="background-color: #f0f0f0; padding: 2px;"> ✓ Instructions </div>	
<div style="background-color: #f0f0f0; padding: 2px;"> ✓ MathArithmetic... cycleCnt=cycleCnt + 1 </div>	
Result	cycleCnt
A [Register]	cycleCnt
Operation	+
B [Integer]	1

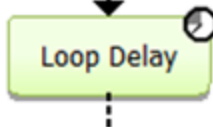
KS2201A API

```
# Previously defined register
cycle_cnt = sequencer.sync_sequence.scopes["AwgEngine0"].registers["CycleCnt"]
#
# AWG local sequences can be accessed from within the Sync Multi-Sequence Block
primary_sequence = sync_block_2.sequences["AwgEngine0"]
# Increment iteration counter
instruction = primary_sequence.add_instruction("CycleCnt++", 10, primary_
sequence.instruction_set.add.id)
instruction.set_parameter(primary_sequence.instruction_set.add.destination.id, cycle_cnt)
instruction.set_parameter(primary_sequence.instruction_set.add.left_operand.id, cycle_cnt)
instruction.set_parameter(primary_sequence.instruction_set.add.right_operand.id, 1)
```

Wait Time (l)

Inserting an instance of WaitTime instruction class causes an HVI sequence to wait for an amount of time specified by a register previously added to the same HVI sequence. The register used needs to be initialized before its usage. Time unit is expressed as integer multiple of the instrument clock cycle duration. For example, in M3xxxA PXI modules a cycle lasts 10 ns.

M3601A GUI

Properties	
Property	Value
<div style="display: flex; align-items: center;"> l.  </div>	
General parameters	
Name:	Loop Delay
Previous:	cycleCnt++
Next:	Jump
Time:	2.13 us
Wait parameters (...)	
Source:	LoopDelay

KS2201A API

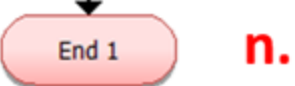
```
# AWG registers
loop_delay = sequencer.sync_sequence.scopes["AwgEngine0"].registers.add("LoopDelay",
kthvi.RegisterSize.SHORT)

# Wait Time
wait_time = awg_sequence.add_wait_time("LoopDelay", 10, loop_delay)
```

End of Sequence (n)

"End" synchronized junctions are used in M3601A to synchronously end all the HVI sequences included in the M3601A flow-chart. There is no equivalent HVI statement in the HVI Python API provided by PathWave Test Sync Executive. All sync statements start and end synchronously with no need for an equivalent "End" statement.

M3601A GUI

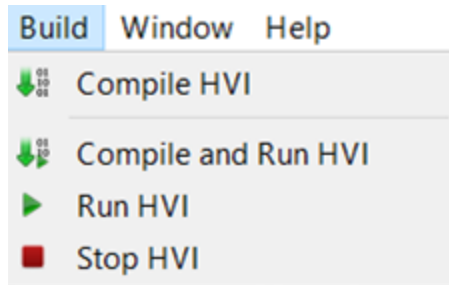
Properties	
Property	Value
<div style="display: flex; align-items: center;">  </div>	
General parameters	
Name:	End 1
Previous:	Jump
Next:	...
Time:	120 ns

Compile, Load, Execute the HVI

Once the HVI sequences are programmed by defining all the necessary HVI statements, users can compile, load and execute the HVI. These steps are equivalent between M3601A and KS2201A. In KS2201A API compile, load and run functionalities can be accessed from the Hvi class. In M3601A HVI GUI offers the

possibility to test the designed HVI sequences using the Build menu commands that allow to compile and execute the HVI, see details in the screenshot below.

M3601A



M3601A use model requires users to generate an .HVI binary file from the designed graphical project. The generated .HVI binary file can be compiled, loaded to HW and executed using a Python API very similar to KS2201A HVI API methods that performs the same equivalent functionalities, more details in the following sub-sections. On top of the methods described below, M3601A API also has a stop() method that can be used to kill the HVI execution. There is no equivalent method in the KS2201A HVI API.

Compile HVI

The compilation operation is performed by calling the compile() API method in both M3601A and KS2201A. This operation processes all the info related to the HVI application, including the necessary HVI resources and the HVI statements included in the HVI sequences. The compilation generates a binary compiled output that can be loaded to the hardware instruments for their HVI engine to execute it. In KS2201A, as an output, the compile() API method provides an object that can tell to the user how many PXI sync resources are necessary to be reserved to execute the HVI application. This is an additional functionality with respect to M3601A and it can be very useful during the HVI sequence design phase to assess the number of PXI trigger lines needed as a function of the design.

M3601A API

```
# Compile HVI
error = hvi.compile()
if error != 0:
    print("HVI Compile Failed!")
    print("Error - ", error, ": ", keysightSD1.SD_Error.getErrorMessage(error) )
    exit(-1)
```

KS2201A API

```
# Compile HVI sequences
hvi = sequencer.compile()
print(hvi.compile_status.to_string())
print("HVI Compiled")
print("This HVI application needs to reserve {} PXI trigger resources to execute".format
(len(hvi.compile_status.sync_resources))
```

Load HVI to Hardware

M3601A API method `load()` is replaced by the KS2201A API method `load_to_hw()`. The API method `load_to_hw()` loads to each HVI engine the binary output obtained from the HVI compilation so that the HVI engine programmed into their digital HW (FPGA or ASIC) can execute it.

M3601A API

```
# Load HVI
error = hvi.load()
if error != 0:
    print("HVI Load Failed!")
    print("Error - ", error, ": ", keysightSD1.SD_Error.getErrorMessage(error) )
    exit(-1)
```

KS2201A API

```
# Load HVI to HW: load sequences, configure actions/triggers/events, lock resources, etc.
hvi.load_to_hw()
```

Execute

HVI execution is controlled by the `start()` API method in M3601A and by the `run()` API method in KS2201A. In M3601A HVI always runs in a non-blocking mode. HVI can be run in a blocking or non-blocking mode. In this application example the non-blocking mode is used. By using this execution mode, SW execution can interact through registers read/write with the HVI sequence execution.

M3601A API

```
# Run HVI sequence
error = hvi.start()
if error != 0:
    print("HVI Run Failed!")
    print("Error - ", error, ": ", keysightSD1.SD_Error.getErrorMessage(error) )
    exit(-1)
else:
    print("HVI is Running!")
```

KS2201A API

```
# Execute HVI in non-blocking mode
# This mode allows SW execution to interact with HVI execution
hvi.run(hvi.no_wait)
print("HVI Running...")
```

Release Hardware

API method `release_hw()` shall be called once the HVI execution is finished to release all the HW resources that were reserved during the HVI execution, including the PXI trigger resources that had been locked by HVI for its execution.

M3601A API

```
# Release PXI trigger resources and close HVI
error = hvi.releaseHW() #releases PXI trigger resources that were reserved by HVI for its
execution
if error != 0:
    print("HVI ReleaseHW Failed!")
    print("Error - ", error, ": ", keysightSD1.SD_Error.getErrorMessage(error) )
    exit(-1)
print("HVI releaseHW successful")
```

KS2201A API

```
# Unlock and release HW resources
hvi.release_hw()
print("Releasing HW...")
```

Further HVI API Explanations

Detailed explanations of each class and functionality of the HVI API can be found in the section "HVI Core API" of the *KS2201A PathWave Test Sync Executive User Manual* or in the Python help file that is provided with the HVI installer, available at: C:\ProgramFiles\Keysight\HVI\api\python\doc\keysight_pathwave_hvi.htm.

Comparison of M3601A HVI GUI and KS2201A HVI API Use Models

The following table summarizes the main operations necessary in an HVI design as performed from the point of view of the M3601A HVI GUI use model and the KS2201A HVI API use model. This table of equivalence can be useful to the users transitioning from one use model to the next.

	M3601A HVI GUI Use Model	KS2201A HVI API Use Model
Operations		
HVI Design Flow	First, a .HVIprj project file must be created using M3601A GUI to design the wanted HVI sequences in form of flow-charts. A binary .HVI file is generated from the .HVIprj file once the HVI sequence design is final. The .HVI file must be open from code to integrate the HVI solution into the application code.	Application code needs to import the keysight_pathwave_hvi library to be able to use the HVI API. HVI sequences can be created programmatically directly into application code with no need to import external files.
HVI Sequence	It is implemented by means of a graphical flow-chart. Each HVI Engine in each Instrument has a single or main HVI Sequence associated where all statements, local and synchronized are added graphically.	Sequence class enables you to create a Local HVI sequence programmatically that run "locally" on a specific HVI engine in a specific instrument. Local Sequences are access by means of the SyncMultiSequenceBlock statement placed in a SyncSequence (SyncSequence). The HVI top sequence is a SyncSequence that contains SyncStatements.
HVI SyncSequence	The concept of HVI SyncSequences was not available in the M3601A flow-charts.	SyncSequence class enables you to add synchronized operations (Sync Statements) common to all HVI engines within the HVI instance. The HVI top sequence is a SyncSequence that contains SyncStatements. Local instructions are added and executed within Local Sequences that can be accessed by adding a SyncMultiSequenceBlock in a SyncSequence.
HVI Resources: Chassis, Triggers, M9031A modules, etc.	Connected chassis are automatically recognized. M9031A boards are transparent to the M3601A software. PXI trigger resources that can be allocated to the HVI solution are chosen from the "Chassis settings" window.	HVI resources can be configured using SystemDefinition class and all the classes inside it.
Program HVI Sequences	You program HVI sequences by adding flow-chart boxes using the M3601A GUI. Configure settings for statements in the Properties window of each flow-chart box.	You program both HVI SyncSequences and HVI (Local) Sequences with the API methods add_XXX(), where XXX is the statement Name.
HVI Compile, Load, Run	Once an .HVI file is open from a script, users can assign each sequence to an HW engine for it to be compiled, loaded to HW, and executed. Project .HVIprj	API SW methods can compile the sequence (hvi.compile()), load it to hardware (hvi.load_to_hw()), and run it (hvi.run()).

files can be also tested directly from the M3601A GUI using the "Compile and Run" function.

Instrument Compatibility

To move to an implementation based on KS2201A PathWave Test Sync Executive please install the latest **Keysight PathWave Test Sync Executive (v1.00.10 or later)**

Both KS2201A PathWave Test Sync Executive and M3601A Hard Virtual Instrument (HVI) Design Environment work with the M3xxxA series of PXIe products. However, KS2201A requires newer firmware while M3601A requires older firmware. The table below lists the firmware version requirements for each.

Instrument	Firmware Version Required by M3601A	Firmware Version Required by KS2201A
M3100A Digitizer	< 2.00	≥ 2.00
M3102A Digitizer	< 2.00	≥ 2.00
M3201A AWG	< 4.00	≥ 4.00
M3202A AWG	< 4.00	≥ 4.00
M3300A AWG & Digitizer Combination	< 4.00	≥ 4.00
M3302A AWG & Digitizer Combination	< 4.00	≥ 4.00

SD1 Software provides drivers, programming libraries and soft front panels for the M3xxxA series. As above, there are similar version requirements included in the table below.

Instrument	SD1 Version Required by M3601A	SD1 Version Required by KS2201A
All M3xxxA series modules	< 3.00	≥ 3.00

New orders of the above modules will be shipped with the newest versions of firmware and SD1 software. In order to use new modules with M3601A software users must downgrade the firmware and SD1 software to the versions listed above.

To use an older module with KS2201A PathWave Test Sync Executive the firmware and SD1 software need to be upgraded to the versions listed above. Firmware and SD1 software are available on [Keysight.com](https://www.keysight.com). They can be found on the Drivers, Firmware & Software tab on the technical support page for the specific instrument.

Conclusions

This transition guide explained what are the differences and equivalences between HVI (Hard Virtual Instrument) instances designed using the different use models of M3601A HVI GUI (Graphical User Interface) and KS2201A HVI API (Application Programming Interface). In this document the focus is on an HVI example designed in M3601A HVI GUI which contains all the key functionalities provided by HVI technology. Throughout the transition guide it is shown how the same example and functionalities can be implemented using the new KS2201A HVI API, with an highlight on the equivalences and differences between the two use models. Users can consume this guide to transition their own HVI instances by leveraging from the provided explanations.