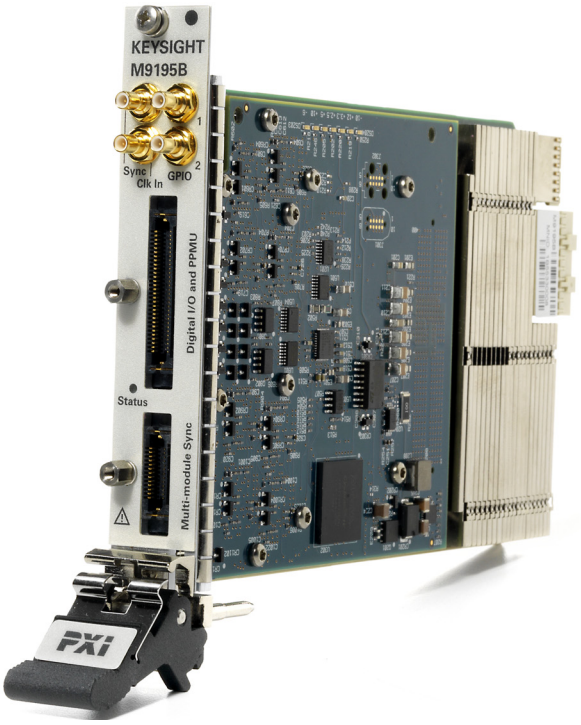
User Guide

For the User Guide, software and other important documentation, see the M9195A/B Software and Product Information CD.

Keysight M9195A/B PXIe Digital Stimulus/Response with PPMU: 250 MHz, 16 channel



KEYSIGHT TECHNOLOGIES

Notices

Manual Part Number

M9195-90005

Edition

Second Edition, March 2016

Published by

Keysight Technologies, Inc.
900 S. Taft Ave.
Loveland, CO 80537 USA

Trademarks

PICMG®, Compact PCI® are registered trademarks of the PCI Industrial Computer Manufacturers Group.

AdvancedTCA® and ATCA are registered trademarks of the PCI Industrial Computer Manufacturers Group.

PCI-SIG®, PCI Express®, and PCIe® are registered trademarks of PCI-SIG.

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.

Declaration of Conformity

Declarations of Conformity for this product and for other Keysight products may be downloaded from the Web. Go to http://keysight.com/go/conformity. You can then search by product number to find the latest Declaration of Conformity.

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.

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.

Safety Information

CAUTION

A CAUTION denotes a hazard. It calls attention to an operating procedure or practice 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 denotes a hazard. It calls attention to an operating procedure or practice, 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.

Contents

3 Using the Four DSR Site Types

4 Using STIL Files

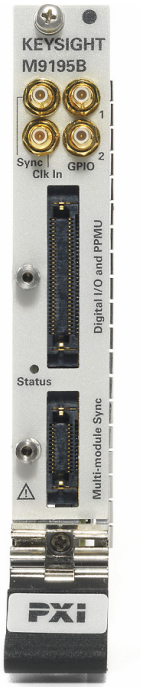
1 Introduction

The M9195A and M9195B modules are PXIe Digital Stimulus/Response (DSR) modules. Both offer:

- 16 bidirectional channels with per-pin programmable logic levels
- High speed pattern application and clock rates up to 250 MHz
- Flexible, per-bit timing control for fast and accurate waveform development
- Per-pin Parametric Measurement Unit (PPMU) for each channel
- Single site (16 channel) and multi-site (four, 4-channel) configurations
- Edit patterns on the fly, without recompiling and downloading the test
- Ability to execute patterns in arbitrary order
- Flexible allocation of deep pattern memory, per channel or per site
- 4 high voltage channels for usages such as flash programming or fuse test
- 4 open drain auxiliary output pins for usages such as fixture relays
- Channel delay adjustment to compensate for cable and fixture propagation delays
- Comprehensive software tool set for quick test development

M9195A

M9195B



M9195A versus M9195B

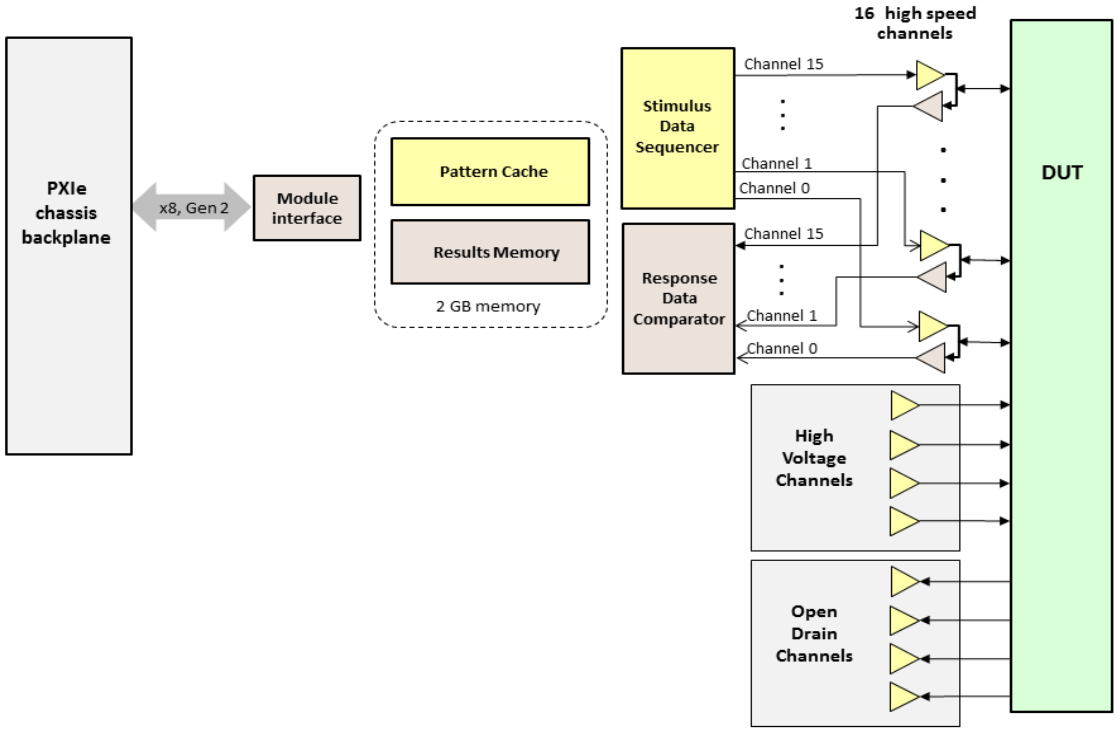
The main distinction is that the M9195B with the multi-module option can support up to 12 synchronized modules handling patterns of up to 192 channels.

Common M9195A/B Functionality

The Keysight M9195A and M9195B are ideal for digital integrated circuit design validation and production test environments. Both modules provide fundamental automated test environment features such as high speed pattern application rates up to 250 MHz, per-pin programming of voltage levels, real time compare, per-pin parametric measurement unit (PPMU), deep vector memory, and flexible pattern sequencing. The 16 channel, single-slot PXIe module introduces a high performance pattern sequencer which enables very powerful pattern creation, two drive edges per period, and supports up to four independent multi-sites to enable quick test development. The M9195A/B drivers provide automatic response delay cable compensation.

Functional Overview

At its most basic level, the M9195A/B DSR modules deliver test vectors to a DUT at up to 250 MHz and monitor responses. Each of 16 channels can be configured for PPMU operation. Four high voltage, and four open drain channels are also available.



How to Use this Guide

NOTE To see if a later version of this document is available, refer to the section “Obtaining the Latest Introduction and KtMDsr Driver” in the Introduction document, or go to the M9195A or M9195B page on the Keysight website and look for a newer release under the Document Library tab.

Scope of this Document

This guide aims to give an overview of the use of M9195A/B modules as digital test instruments. It also provides details on where to go for specific API information. This guide also includes detailed presentations on:

- STIL Programming of the M9195A/B
- Using Bulk Data Files
- Using OpenXML Spreadsheets

Document Conventions

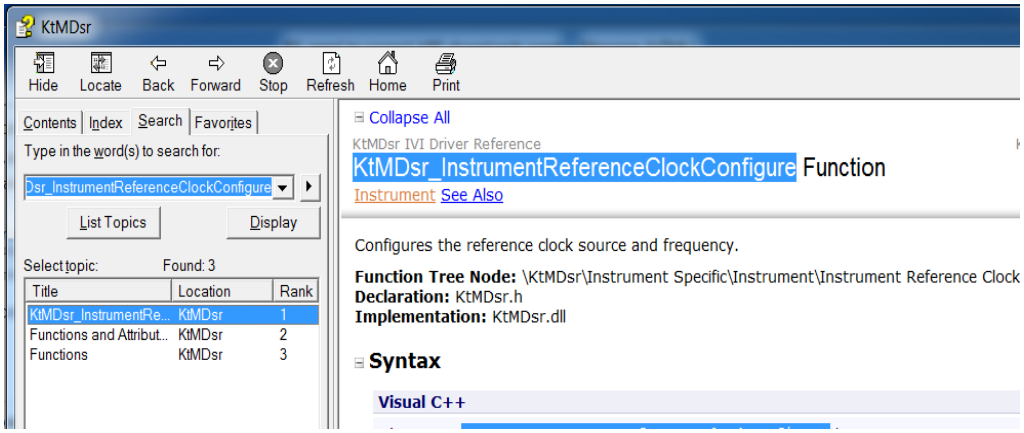
Automatic hypertext links in Keysight documents are highlighted in red.

This document provides manual links to detailed discussion of software topics and API-level details through key words that can be used in help system search boxes. These key words are in ***bold italic***.

For example, in a discussion of reference clock functionality it this guide, you may find:

- Programming Guide:
  “***KtMDsr_InstrumentReferenceClockConfigure”***

The bold italic wording can be copied and paster into the Search function on the KtMDsr Programming Guide .chm file to find the appropriate API detail.



Manuals and Help Systems

M9195A/B Documents and Help Systems

- **Introduction:** This document summarizes current information on the M9195A/B modules including:
  - -- current release details
  - -- hardware and software requirements

  This document is on the *Software and Product Information CD* that came with your system.

  To find the most recent Introduction on the website, go to the web page for either the M9195A or B, on Keysight.com, and select *Technical Support*. Under *Technical Support*, select the Drivers, Firmware & Software tab and then click on the drivers for download. On the drivers download page, there is a link to the *Introduction*.

  To see if a later version of this document is available refer to the section in it entitled: *Obtaining the latest Introduction and KtMDsr Driver*.
- **Startup Guide:** This document describes installation of the module.
- **Programming Guide :** This help system provides overview material for programmers as well as detailed reference material on APIs for the M9195A/B.
- **Soft Front Panel (SFP) Help file:** This help system supports the use of the DSR Software Front Panel. It is inter-linked with the Programming Guide.
- **LabVIEW Driver Help :** Supports LabVIEW driver for M9195A/B.
- **Specification Guide:** Contains technical specifications for all manufacturing versions of the M9195A/B PXIe Digital Stimulus/Response module. Specifications published in the data sheet only apply to the current manufacturing version of the module.
- **Security Guide:** This document details the internal memory locations of the M9195A/B PXIe Digital Stimulus/Response module. It describes instrument security features and the steps necessary to declassify the products through memory sanitization or removal.

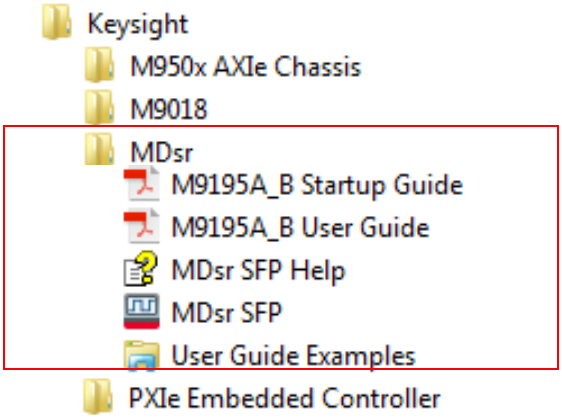
Other Keysight PXI and PXIe Modular Systems

www.keysight.com/find/pxi

Where to Find the Documentation

At the Windows Start Button

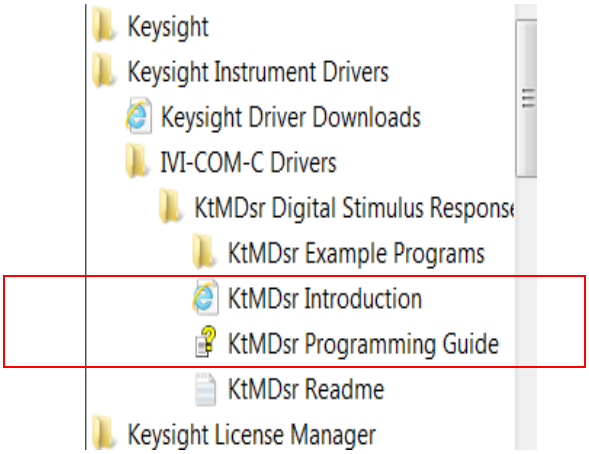
Click through **Start>All Programs>Keysight>MDsr**. The display will appear:



Click on the appropriate icon for:

-- Startup Guide
-- User Guide
-- Software Front Panel (SFP) help system

For addition documentation, go further down the Start button listing to **Start>All Programs>Keysight Instrument Drivers>KtMDsr Digital Stimulus Response.** The display will appear something like:



Here click on the icons for:

-- KtMDsr Programming Guide
-- KtMDSr Introduction

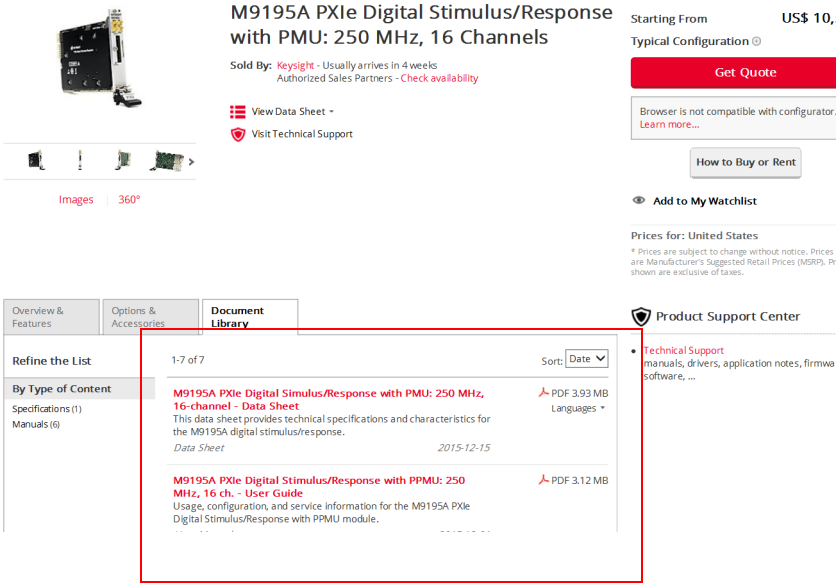
At the Keysight website

Go to the M9195 page on the web:

www.keysight.com/find/M9195A (go to **Document Library**).

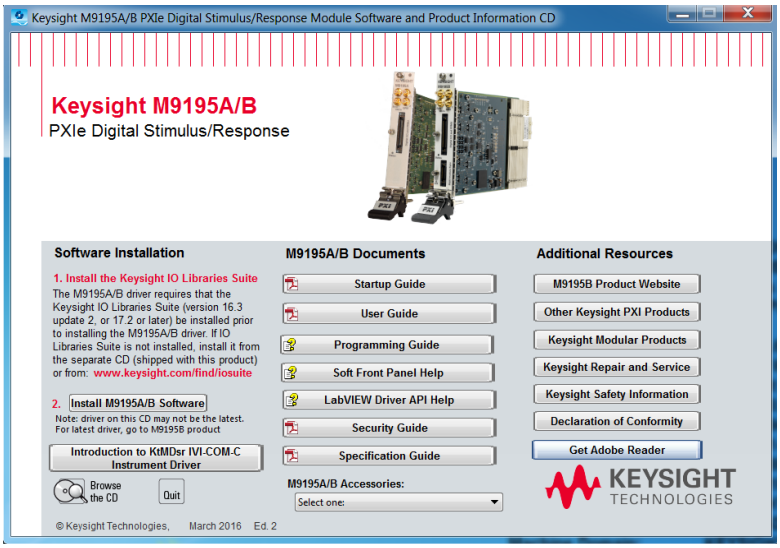
www.keysight.com/find/M9195B (go to **Document Library**).

All released M9195A/B help systems and documents are there, including the Introduction.



On the *Software and Product Information CD*

Bring up the CD main menu.



All M9195A/B documents and help systems are here, including accessory guides.

System Requirements

For up-to-date details on M9195A/B hardware and software requirements, see the *Introduction* document section: *Hardware and software supported with this driver*. For details on finding the *Introduction*, see "Manuals and Help Systems" on page 4.

Hardware Upgrade Option Licenses

Hardware upgrade options extend M9195B capabilities. For details, refer to the *M9195A/B Startup Guide* (see "Manuals and Help Systems" on page 4).

In the Startup Guide, *Step 7: Check M9195B Licensing (Optional)* describes available licensing options.

The Startup Guide section, *Installing Hardware Upgrade License Options*, details the upgrade process. This section shows how to check which M9195B hardware upgrade options are already present.

Firmware Updates/Downgrades

As new driver releases are installed, M9195A/B users will usually have to upgrade the firmware on the modules. For the procedure, refer to the *MDsr Soft Front Panel Help* and search for ***Firmware Update***.

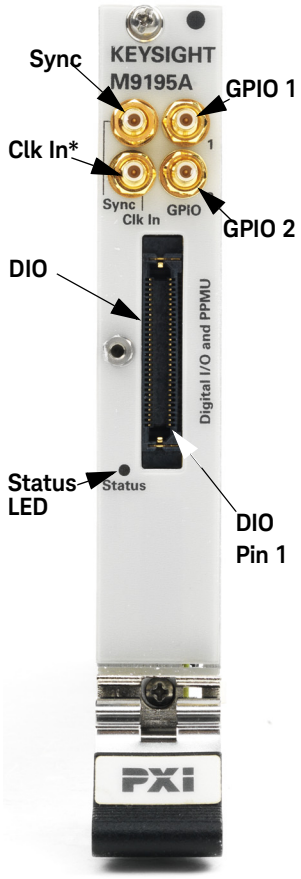
In general, updating to the latest driver release is the recommended process.

At some point, M9195A/B users may find that a new driver is not compatible with their application. Therefore, they may wish to roll back to an earlier driver. This will make downgrading firmware necessary. Another possibility is that a new additional module may be added to an existing test system with an older driver and older firmware. If there is no desire to update the driver, it becomes necessary to roll back the firmware on the new module. For details on this procedure, go to the *MDsr Soft Front Panel Help* and search for ***Firmware Downgrade.***

NOTE Automatic Corrections must be performed after both Firmware Updates and Downgrades.

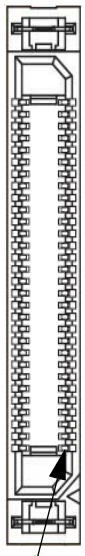
M9195A/B Front Panel Features

M9195A

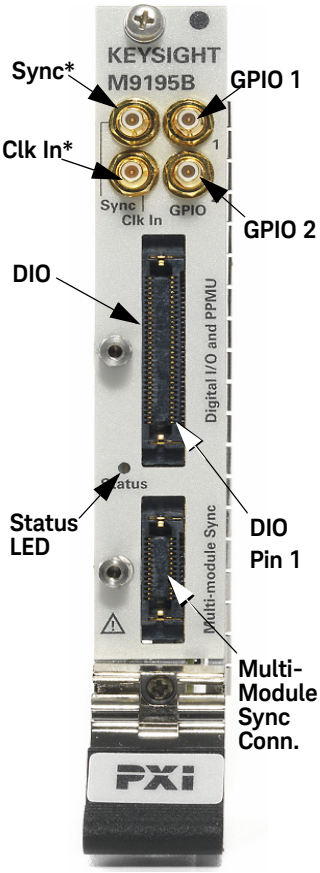


Digital I/O (DIO) Connector Pinout

Signal	Pin #		Pin #	Signal
Channel 03	60		59	GND
GND	58		57	Channel 00
Channel 01	56		55	GND
GND	54		53	Channel 02
Channel 07	52		51	GND
GND	50		49	Channel 06
Channel 05	48		47	GND
GND	46		45	Channel 04
HV22	44		43	GND
GND	42		41	HV23
HV21	40		39	GND
GND	38		37	HV20
Channel 11	36		35	GND
GND	34		33	Channel 10
Channel 09	32		31	GND
GND	30		29	Channel 08
Channel 13	28		27	GND
GND	26		25	Channel 12
Channel 15	24		23	GND
GND_SENSE	22		21	Channel 14
Ppmu_Sense12	20		19	Ppmu_Sense14
Ppmu_Sense13	18		17	Ppmu_Sense15
Ppmu_Sense11	16		15	Ppmu_Sense10
Ppmu_Sense09	14		13	Ppmu_Sense08
Open Drain 2	12		11	Open Drain 0
Open Drain 3	10		9	Open Drain 1
Ppmu_Sense06	8		7	Ppmu_Sense05
Ppmu_Sense07	6		5	Ppmu_Sense04
Ppmu_Sense01	4		3	Ppmu_Sense02
Ppmu_Sense00	2		1	Ppmu_Sense03
Latch/GND				Latch/GND



M9195B



M9195A/B Front Panel and Connector Pin-out

* The **CLK In** SMB connector allows you to input a 10 MHz or 100 MHz reference clock

Returning a module for service

Refer to the M9195A/B Startup Guide for details.

APIs and Programming

The M9195A/B is supported by multiple APIs and programming approaches. The following references present overview material on these:

IVI-COM

- Programming Guide:
  "Using Visual C++"

IVI-C

- Programming Guide:
  "Using Visual C#"

LabVIEW

- LabVIEW Driver help:
  "VI Tree"

VEE

- Programming Guide:
  "Using Keysight VEE Pro"

MatLAB

- Programming Guide
  "MATLAB"

STIL

See chapter in this section on "Using STIL Files" on page 49.

Bulk Data

In this document see section on "Using Bulk Data Files" on page 79.

OpenXML Spread Sheets Files

In this document, section on "Using OpenXML (Spreadsheet) Data Files" on page 83.

2 Theory of Operation

The following block diagram presents a conceptual overview of the M9195A/B.

The flow of information in the system is mainly from the upper left to the lower right.

Click within the dash-lined boxes for details on specific material.

To get a quick overview of system operations, click through these topics:

M9195A/B
PXIe Digital Stimulus/Response

PXIe chassis backplane

(connectors XP3 and XP4)

x8, Gen 2

Module interface

GPIO2 (External2)
GPIO1 (External1)
PXI_TRIG[0:7]
Triggers and Markers
PXI_STAR
PXIe_DSTARB (trigger only)
PXIe_DSTARC (marker only)

These two signals can each be a trigger input or a marker output.

The Sync connector is used to implement MatchLoops across multiple M9195B modules. In order to do this, the Sync signals are daisy-chained together between modules.

Front panel Clk In
(10 MHz or 100 MHz)

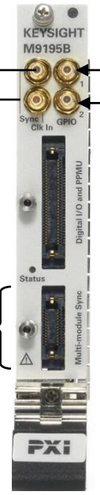
PXIe_CLK100

PXIe_DSTARA
(10 MHz or 100 MHz)

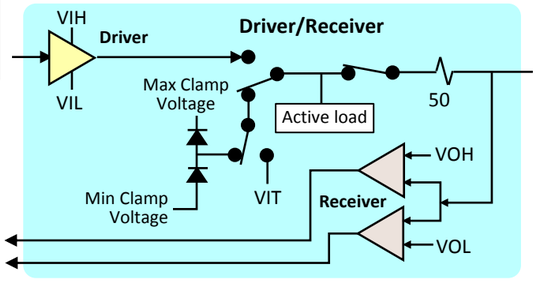
Clock Selector/ Generator

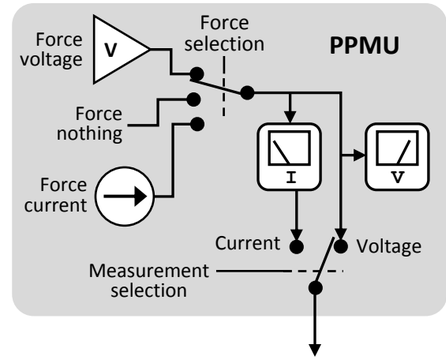
System clock

Multi-module Sync Connector: This connector is only present on the M9195B, and is used to synchronize software and hardware triggers across multiple modules.



Digital I/O and PPMU





Stimulus Sequencer

Bank 3 sequencer
Bank 2 sequencer
Bank 1 sequencer
Bank 0 sequencer

The four stimulus sequencers (one for each bank) can operate independently and in parallel, each producing their own 4-bit output pattern at independent clock rates. See the mapping between banks and channels below. If an output pattern is specified that spans more than a single bank, the four sequencers will be slaved together, and will no longer operate independently. In this case, only one pattern (which can be 16 bits wide) can be output at a time.

Channel 15
Channel 1
Channel 0

Data for compares

Channel 15
Channel 1
Channel 0

Pattern Cache

Results Memory

2 GB memory

Response Comparator

Driver
Receiver

PPMU

24

Channel 15

Internal Loopback – This feature allows each channel's Receiver to monitor its associated Driver. This permits the Response Data Comparator to receive and process data generated by the Stimulus Data sequencer. This capability enables trial programs to be developed and executed without connection of the M9195A/B to any cable or device.

16 high speed channels

PPMU

56

Channel 1

PPMU

57

Channel 0

For the 16 high speed channels, "output" refers to DUT outputs while "input" refers to DUT inputs.

DUT (device under test)

Four M9195B modules are configured below in a multi-module system. Up to 12 total M9195B modules can be configured in a multi-module system.

Slave modules

Master module

The Sync connectors are daisy-chained together for multi-module MatchLoop operation.

Y1250A Four module sync cable

This cable connects from the master module to the 3 slave modules.

High Voltage Channels

High voltage channel 23 (associated with high speed channel 14) — 41 — HV23
High voltage channel 22 (associated with high speed channel 10) — 44 — HV22
High voltage channel 21 (associated with high speed channel 6) — 40 — HV21
High voltage channel 20 (associated with high speed channel 2) — 37 — HV20

High speed channel number
High voltage (HV) channel number

Channel numbers and banks

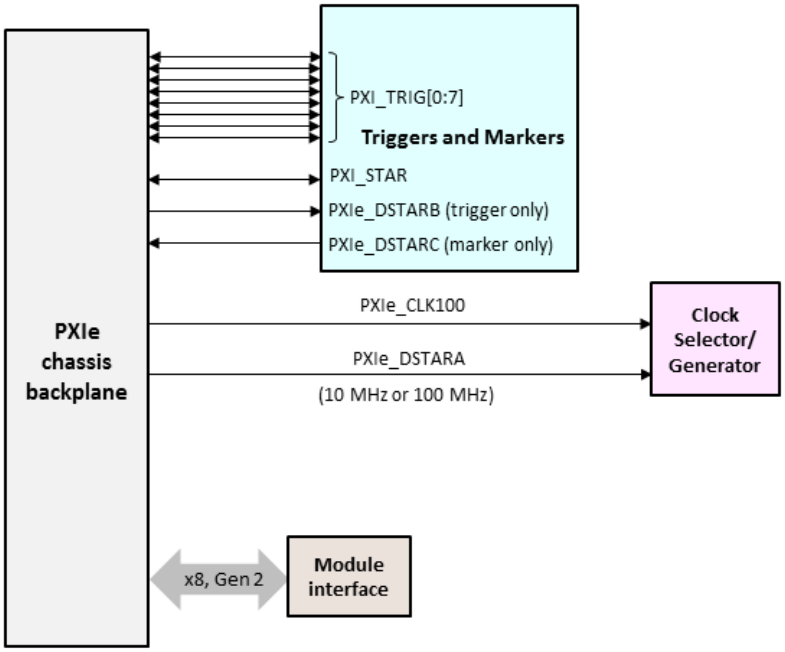
15	23/14	13	12	11	22/10	9	8	7	21/6	5	4	3	20/2	1	0
Bank 3				Bank 2				Bank 1				Bank 0			

In single module operation, I/O operations can be performed with each bank independently of the other banks, allowing four different DUTs to be tested simultaneously. Alternatively, all four banks can be combined to provide 16 channels of synchronized I/O to the DUT. See the User Guide for other supported configurations.

Open Drain Auxiliary Outputs

10 — Open Drain 3
12 — Open Drain 2
9 — Open Drain 1
11 — Open Drain 0

PXIe Interface



With an aggregate throughput maximum of 4 GB/s, the PXIe backplane provides sufficient bandwidth for multiple DSR modules.

The DSR connects with the host controller over a PXIe interface supporting transfer rates of up 5 Gb/s per lane with up to 8 lanes available. The DSR interface fully conforms with PXI Systems Alliance PXI Express hardware and software specifications. For more information on PXIe standards, see the PXI Systems Alliance website.

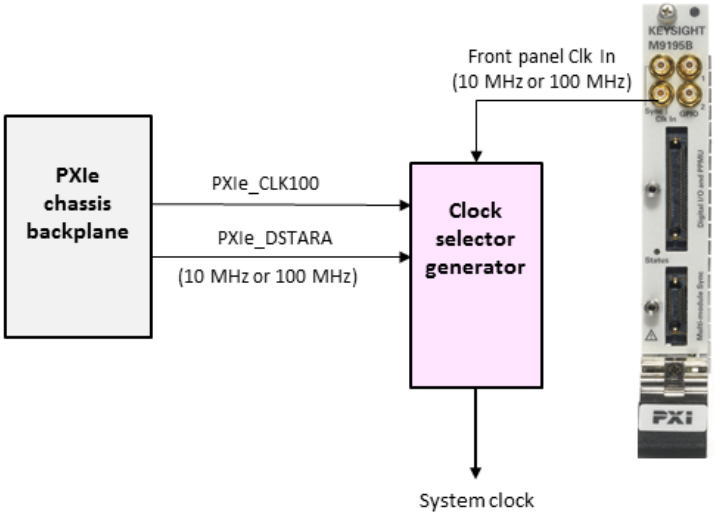
The interface supports the legacy PXI single-ended trigger bus as well as PXI_STAR to support multiple triggering and synchronization options.

The module can be driven by an embedded controller such as an M9037A or by a PC, external to the PXIe chassis.

For the best performance in a Keysight M9018A PXIe chassis, position DSR modules in slots 2, 6, 11 and 15.

Reference Clock

M9195A/B clocking options allow the use of either external or PXIe bus sources for a reference clock.



Typically, the underlying system clock for DSR modules is derived from the differential reference clock signal on the PXIe bus, PXIe_CLK100. This clock is sourced by the PXIe chassis itself and delivered independently to each module. As a differential signal, PXIe_CLK100 delivers great noise immunity and device-to-device skew of less than 250 picoseconds.

The external clock option consists of a 10 or 100 MHz signal through the front panel SMB Clk In connector.

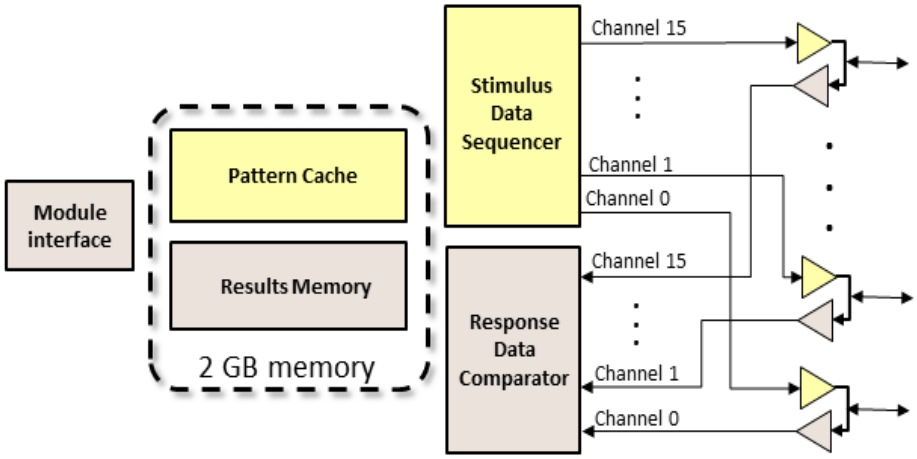
The modules can also use a PXIe_DSTARA clock input, thus allowing a system timing module to replace the backplane clock with a higher precision timebase.

Specific API allow the clock to configured:

- Programming Guide
  IVI-COM: ***"IKtMDsrInstrumentReferenceClock.Configure"***
  For IVI-C: ***"KtMDsr_InstrumentReferenceClockConfigure"***
- LabVIEW Driver help:
  "Instrument Reference Clock Configure"

Whatever the source, the module derives is a 100 MHz clock to provide a reference timebase with accuracy of +25ppm and period jitter of less than 2 picoseconds RMS. This reference can support IO channels in up to four different time domains (one per bank, see "Sites and Banks" on page 27) with channel-to-channel jitter within those time domains of less than 25 picoseconds RMS.

Memory



The DSR includes 2 GB of SDRAM to hold test patterns on the way to the DUT, and response data returned from the DUT.

By default, about 1.9 GB is allocated to Pattern Cache, and 100 MB to Results Memory. This 95% allocation to Pattern Cache reflects a typical test scenario where data is stored to the Results Memory only when the Response Data Comparator is set to store "Cycles with Failing Compare."

Specific API calls re-configure memory allocation as needed.:

- Programming Guide:
  IVI-COM: ***"IKtMDsrInstrument.ConfigureMemory"***
  IVI-C: ***"KtMDsr_InstrumentConfigureMemory"***
- LabVIEW Driver help:
  "***Reference Clock Configure"***

APIs support reporting of the amount of cache remaining after patterns have been compiled to it:

- Programming Guide:
  IVI-COM: *"**IKtMDsrInstrument.RemainingCache"***
  IVI-C: *"**KtMDsr_GetRemainingCache"***
- LabVIEW Driver help:
  "***Instrument Get Remaining Cache"***

APIs can also report the amount of memory used by test vectors:

- Programming Guide:
  IVI-COM: ***"IKtMDsrInstrument.PatternCacheSize"***
  IVI-C: ***"KtMDsr_GetRemainingCache"***

- LabVIEW Driver help:
  "Instrument Get Pattern Cache Size"

The APIs support clearing the cache:

- Programming Guide:
  IVI-COM: ***"IKtMDsrInstrument.ClearCache"***
  IVI-C: ***"KtMDsr_InstumentClearCache"***
- LabVIEW Driver help:
  "Instrument Clear Cache"

On the results side, the APIs can reserve specific amounts of memory for results through the configure memory and query for this amount:

- Programming Guide:
  IVI-COM: ***"IKtMDsrInstrument.ResultSize"***
  IVI-C: ***"KtMDsr_InstumentGetResultSize"***
- LabVIEW Driver help:
  "Instrument Get Result Size"

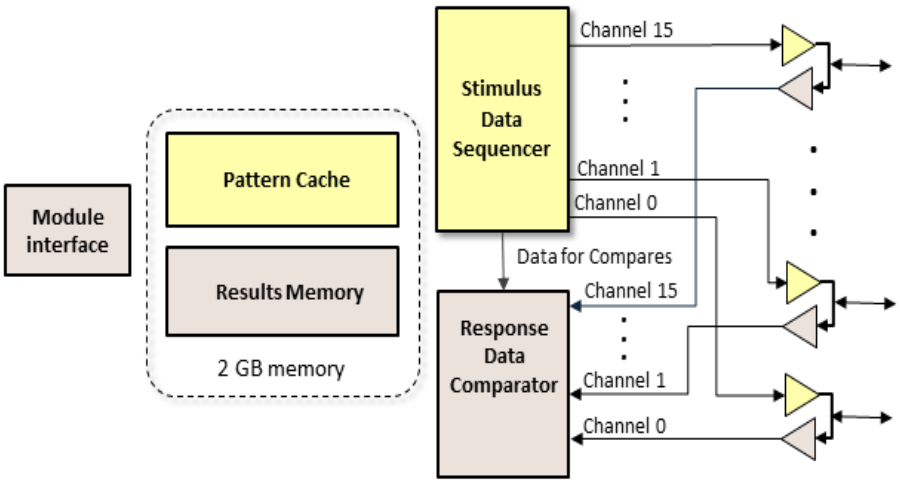
Results Memory is typically much smaller than Pattern Cache because the Response Data Comparator only creates a record when a compare operation is present in a vector and then, depending on test configuration, the Comparator may only store the record to Results Memory when a compare failure is detected.

The Response Data Comparator will simply stop writing result records to memory when the space allocated for the pattern has been filled.

The APIs provide a number of fetch operations with which the contents of results memory for specific patterns are retrieved from the Results Memory and assemble in to comprehensible data:

- Programming Guide:
  IVI-COM; IVI-C: ***"Acquiring Results and Data Collection"***
- LabVIEW Driver help:
  "Acquiring Results and Data Collection"

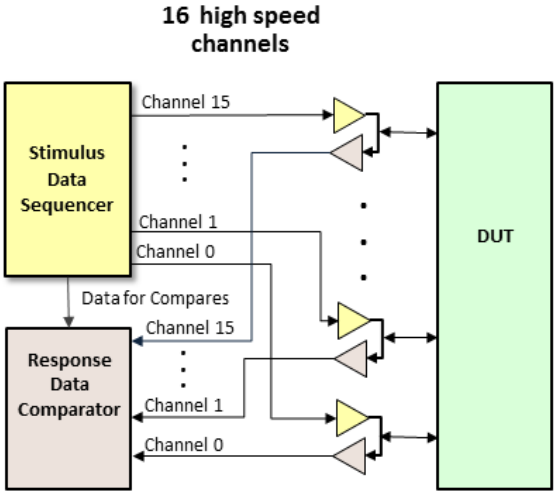
Stimulus Data Sequencer



The Stimulus Data Sequencer fetches pattern data from the Pattern Cache and outputs it to the DUT. In addition to containing pattern data, the Pattern Cache also contains data indicating the DUT's expected response to the pattern data. This is the *expected response data* which, again, is stored with the pattern data. See "“Response Data Comparator” on page 19" for information on how the expected response data is used.

The Stimulus Data Sequencer can operate as one site of up to 16 channels, or it can operate as four individual, 4-channel sites. For more information, see "“Sites and Banks” on page 27.

Digital IO Channels



Introduction

Each module provides 16 bi-directional digital IO channels. In multi-module configurations, up to 12 modules can be configured to support 192 modules.

On each module, the 16 bi-directional digital IO channels are numbered channel 0 through channel 15.

In a multi-module configuration, up to 12 M9195B modules can be configured in a multi-module instrument, which will support 192 channels (12 modules x 16 channels/module).

In a multi-module instrument, the channel numbers have this format:

```
<mm><cc>
```

Where <mm> is the module number (0-11) while <cc> is the channel number (0-15) on the specified module. For example channel 603 represents module 6 and channel 3 on that module, while channel 1114 represents module 11 and channel 14 on that module. "Multi-Module Operations" on page 26 for more details.

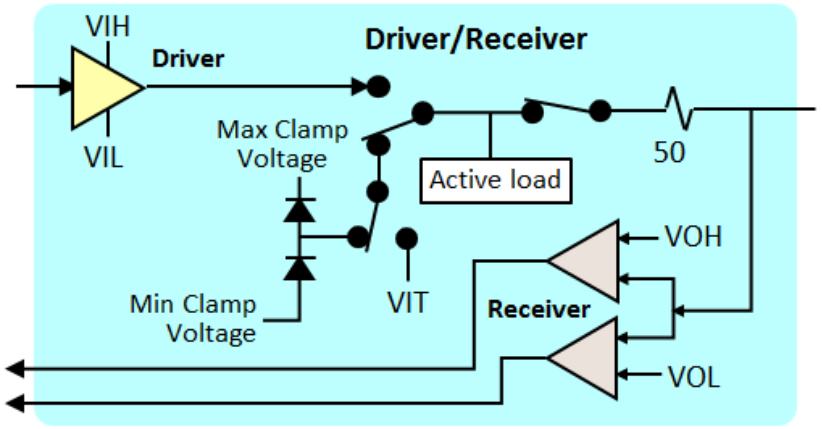
Programming

See the KtMDsr Programming Guide and LabVIEW Driver help files for overviews on IO Channel programming:

- Programming Guide:
  IVI-COM and IVI-C: ***"Channels Overview", "Channels", "Multi-Module Operation"***
- LabVIEW Driver help:
  "Channels Overview"

Driver/Receiver Electronics

The following diagram summarizes Driver/Receiver channel electronics.



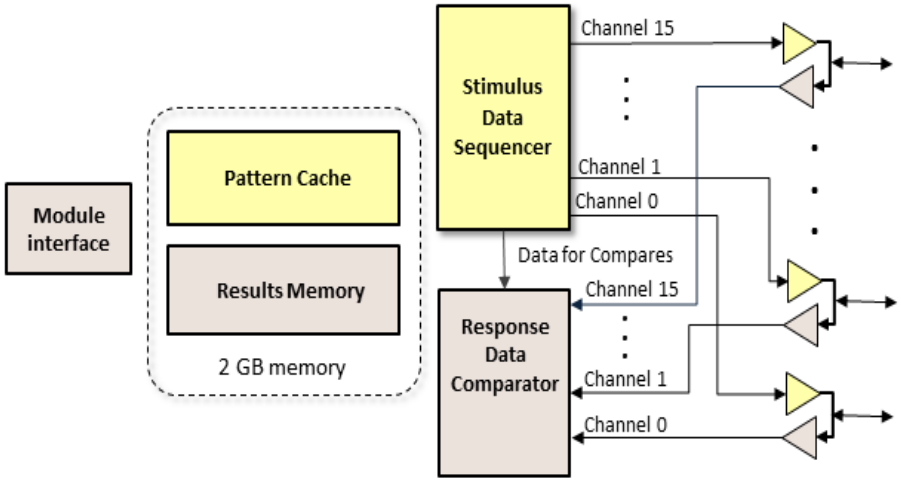
When the DSR is sending a signal to the DUT, the output Driver is connected. The high and low Driver voltages are controlled by VIH and VIL, respectively.

When the DSR is receiving a DUT output signal, the Receiver data comparators compare the DUT signal to the high and low threshold voltages, VOH and VOL, respectively. The default Receiver configuration is high input impedance. Alternatively, the user can specify 50 ohm input impedance or can specify an active load.

Response Data Comparator

The Response Data Comparator receives the response from the DUT and determines if the response is correct. As shown in “Driver/Receiver Electronics”, the user can specify the high and low threshold voltages for signals from the DUT.

In determining whether the DUT response is correct, the Response Data Comparator uses the expected response data that is stored with the pattern data. The data path for the expected response data is shown as the arrow labeled “Data for Compares” between the Stimulus Data Sequencer and the Response Data Comparator.



The Response Data Comparator can be set to store all results, or only clock cycles in which failures are indicated.

The Response Data Comparator can be set to incorporate compensation for delays resulting from cabling and connectors between the DUT and the module. For more on this, refer to the channel overview topics:

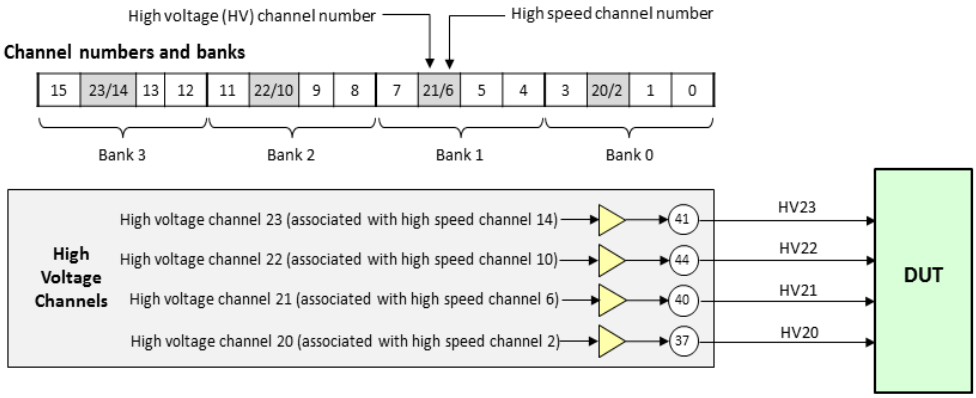
- Programming Guide:
  IVI-COM and IVI-C: ***“Channels Overview”***, ***“Channels”*** and ***“ResponseDelay Compensation”***
- LabVIEW Driver help:
  “Channels Overview” and ***“Channel Response Delay Compensation”***

Once results move from the Response Data Comparator to memory, a number of API functions support further movement. For an overview:

- Programming Guide and LabVIEW Driver help:
  “Fetch” or
  “Results”

High Voltage Channels

As shown in the diagram below, each of the banks supports one high voltage channel associated with an IO channel line.



The M9195A/B provides four high voltage channels (HV20 - HV23, 0V to +13.5V). These channels are slower-speed I/O channels that can be used for flash programming memory devices such as EEPROMS.

The HV20-HV23 channels are typically used as a programming enable voltage (sometimes called V_{PP}) and should be maintained as long as the program/verify mode is operating when programming your EEPROM.

NOTE These four HV channels may not be used simultaneously with their corresponding high speed channel. See the data channel dependency table below.

High-Speed Data Channel Dependency

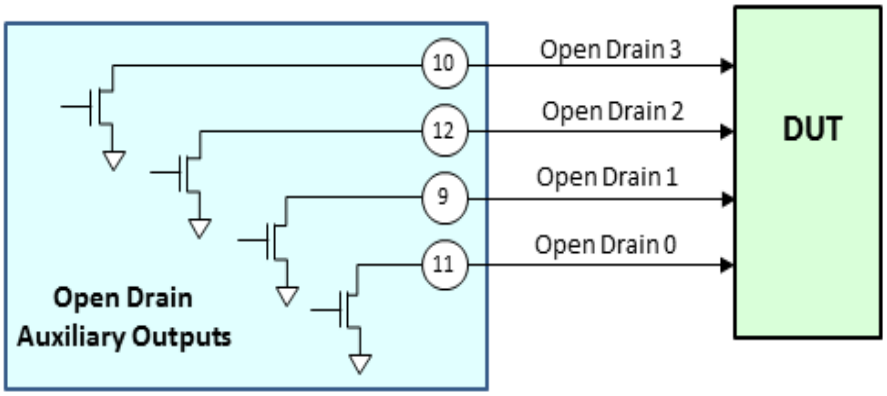
Each high voltage channel is associated with a high-speed channel on its respective IO bank. It is important to note that a high-speed channel may not be used at the same time as its corresponding high voltage channel (for example, channels 2 and 20 may not be used at the same time). These dependencies are outlined in the table below.

IO Bank / HV channel	Tied to Channel
IO Bank 0, HV20	CH02
IO Bank 1, HV21	CH06
IO Bank 2, HV22	CH10
IO Bank 3, HV23	CH14

NOTE

When a high voltage channel is enabled, the associated high-speed channel will be disabled and set to high impedance. When a high-speed channel is enabled, the associated high voltage channel will be disabled. A disabled high voltage channel will appear as 50 ohms to ground.

Open Drain Auxiliary Outputs



Each M9195A/B provides four general-purpose open-drain auxiliary outputs that can be used for driving relays on your Device Under Test (DUT) interface board. These lines are designed for low voltage application (up to 16 VDC) and are independent of any other data or HV lines on the module.

Each line offers built-in inductive feedback protection, drive for latching or non-latching relays, and open load and short circuit protection. Each open-drain line features a 10 kΩ pull-up resistor to +5 volts, 0.3 Ω (typical) on-resistance, and can sink up to 1 A (maximum) of load current.

For more information, see:

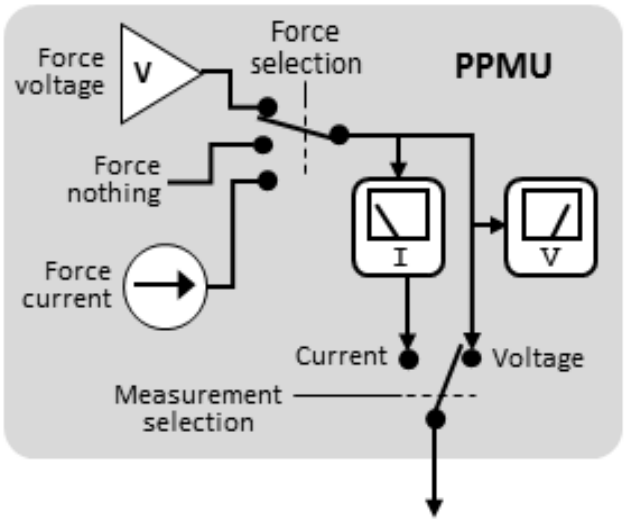
- Programming Guide:
  "SetAuxilaryOutputControl
  "QueryAuxiliaryOutputControl"
- LabVIEW Driver help:
  Auxiliary

Per-Pin Parametric Measurement Unit (PPMU)

Each of the 16 IO Channels on a M9195A/B can be configured as a PPMU line.

PPMU operations involve five measurement modes:

- Force Nothing, Measure Voltage (FNMV)
- Force Voltage, Measure Current (FVMI)
- Force Voltage, Measure Voltage (FVMV)
- Force Current, Measure Voltage (FIMV)
- Force Current, Measure Current (FIMI)



PPMU Applications

PPMU applications include open and shorts testing as well as precise measurement of leakage currents. During PPMU testing, voltage clamps may be set to prevent damage to the M9195A/B. See the following section for details on Voltage Clamps.

Voltage Clamps

The on-board clamps prevent large voltage transient spikes when changing operating mode or current range.

The specified clamp voltages are active in two situations:

- When the DSR is sourcing a digital signal via the PatternSite or the StaticSite
- When the DSR is in Ppmu mode, and the Ppmu is forcing a current, the clamps are enabled or disabled using the ClampVoltagePpmuEnable function

At other times, the clamps are inactive. This includes:

- When the pattern, capture, or static sites are sensing an input
- When the Ppmu is forcing a voltage
- When the channels are disabled
- When the DSR is forcing a voltage from an active pattern site or static site

Setting the clamp voltages does not change the setting of ClampVoltagePpmuEnable. This setting determines if the DSR should enable the clamps when the PPMU is forcing a current. The clamps are never active when the PPMU is forcing a voltage. Setting and clearing the ClampVoltagePpmuEnable has no effect on the settings of the clamp voltages

For more detail see:

- Programming Guide:
  IVI-COM: ***"IKtMDsrChannels.ClampVoltagePpmuEnable"***
  IVI-C: ***"KtMDsr_ChannelClampVoltagePpmuEnable"***
- LabVIEW driver help:
  "Channel Clamp Voltage Ppmu Enable"

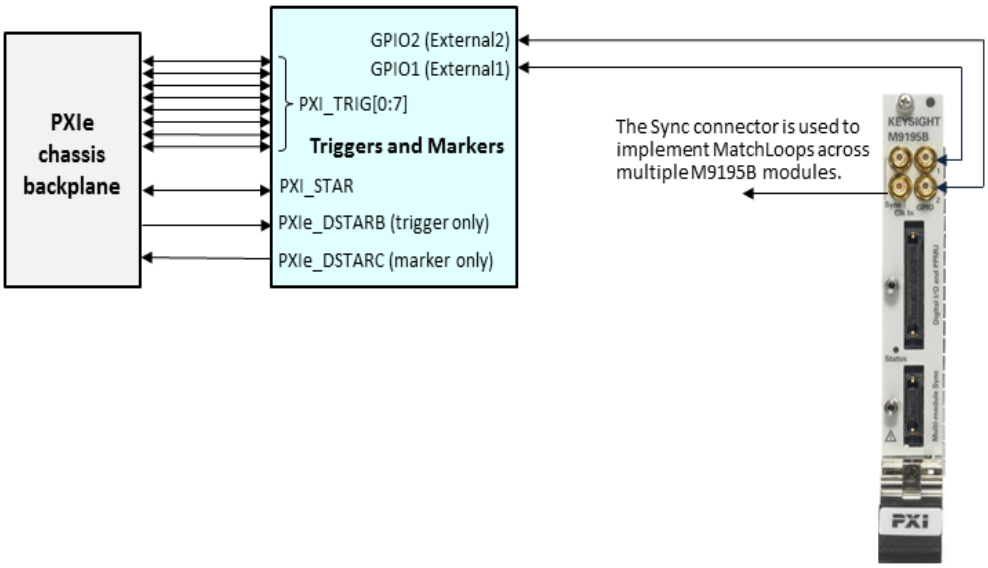
Triggers and Markers

The M9195A/B supports both Triggers and Markers to synchronize testing. Triggers are inputs to the DSR, and can be used to initiate the generation of a pattern or the continuation of a pattern that has already been started. Markers are outputs from the DSR, and can be used to trigger other devices, either other modules in the same chassis or external devices.

CAUTION

Most of the DSR signals associated with triggers and markers are bi-directional, that is, they can be programmed to be either a trigger input or a marker output. Because of this, care must be taken to avoid contention between signals. This can occur if two marker outputs are programmed on the same signal line, such as between the DSR and another module in the chassis.

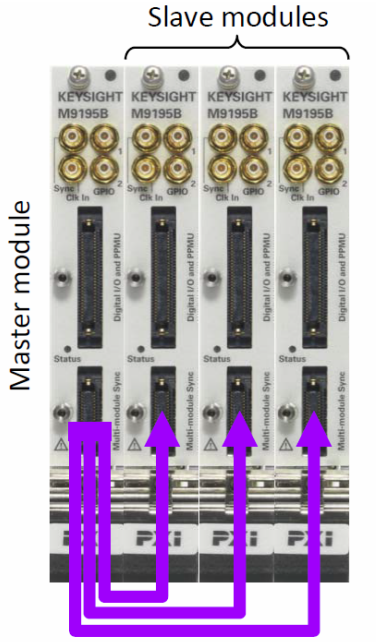
- Programming Guide:
  IVI-C, IVI-COM: ***"Triggers and Markers"***
- LabVIEW Driver help:
  "Triggers and Markers"



Multi-Module Operations

When two or more M9195B modules are licensed for multi-module operation, they can be connected via the Y1250A or Y1251A multi-module sync cables. The major advantage from this configuration is that sites of more than 16 channels can be created.

NOTE The driver detects the presence of the sync cable at Initialize. If the sync cable is not detected, an error is thrown the first time a PatternSite is activated. As with other warnings, the error is only thrown the first time a PatternSite is activated.



Y1250A 4-module
multi-module sync cable

A PattenSite in a multi-module configuration can extend across two or more modules. However, there can be only one PatternSite when more than one module is involved.

In a multi-module configuration, the Master Module is defined by the first module in the driver initialize string:

```
                                  Module0(Master)   Module1(slave)   Module2(slave)
Driver Initialize ("PXI2:00...”; “PXI5:00...”; “PXI8:00...”;)
```

Channels are assigned on this order:

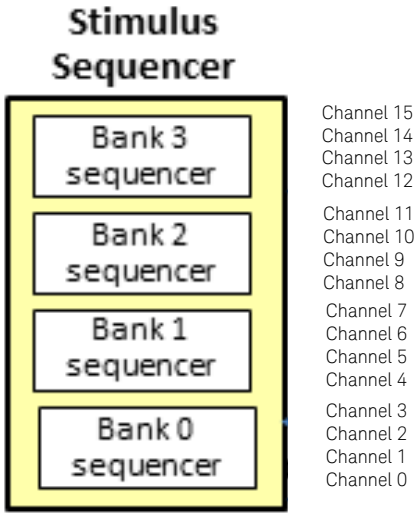
Master channels: 0 - 15

Module1 channels: 100 - 115

Module2 channels: 200 - 215

Sites and Banks

On M9195A/B modules, banks are defined as four IO channels with the same Stimulus Sequencer. So, a bank is the smallest collection of IO channels that can share timing parameters.



One M9195A/B module can support up to four sites as long as each site is confined to a bank. A site can extend to include IO channels from more than one bank, but then it can be the only “active” site on the module. An “active” site is one that is moving test vectors to and from a DUT, such as a Pattern Site or Capture Site.

Importantly, a *site* is not a DUT. Multiple sites can test the same DUT, or one site can test multiple DUTs.

The are four site types supported by the M9195A/B DSR:

- ***Pattern*** – The DSR sends pattern vectors of multiple signals to the DUT and montiors responses.

  See “Using PatternSites” on page 36.

- ***Capture*** – A subset of Pattern Sites in which the DSR sends only one signal, typically a clock, to the DUT and captures the response pattern from the DUT.

  See “Using CaptureSites” on page 40.

- ***Static*** – The DSR delivers DC voltage levels over specific IO channels and monitors DC voltage levels over others.

  See “Using StaticSites” on page 42.

- ***PPMU*** – See “Per-Pin Parametric Measurement Unit (PPMU)” *on page 23*.

  See also “Using PpmuSites” on page 44.

3 Using the Four DSR Site Types

A site specifies a mapping to physical channels and triggers, and is used to bind Patterns, PPMU, and Static operations to physical channels. This chapter looks at operation and configuration of DSR sites and steps necessary to run tests on different site types:

- Sites, see below.
- "Using PatternSites" on page 36
- "Using CaptureSites" on page 40
- "Using StaticSites" on page 42
- "Using PpmuSites" on page 44

For site examples, complete programming examples showing site setup (C++ using IVI-C and IVI-Com, C#, VisualBasic, etc.) are located in the following folders:

C:\Program Files\IVI Foundation\IVI/Drivers\KtMDsr\Examples

C:\Program Files (x86)\IVI Foundation\IVI\Drivers\KtMDsr\Examples

Sites

A site is a group of channels that are organized around a common purpose. For example, four DSR channels can be grouped into a PatternSite, and then two channels can be used to send pattern data to the DUT while the remaining two channels can be used to receive data from the DUT.

Four Site Types

The following table summarizes the four DSR site types.

Active Site	Usage
PpmuSite	A site is activated for PPMU measurements in order to perform various analog measurement and analog stimulus operations. This includes selecting a forcing function such as voltage, current, or none, and selecting a simultaneous measurement function which is voltage, current, or none.
PatternSite	A PatternSite has the functions and attributes necessary to control the execution of a test pattern. This includes operations that are used to run the pattern, monitor its execution, and modify the pattern in cache memory so it can be modified between runs. To activate a pattern, both a site and a PatternExec (PatternBurst and DCLevels) are provided. The driver implementation must compile a pattern, for a specific site, then download it to the module. To reduce download and compile times for large patterns, functions are provided in the pattern site for loading a pre-compiled pattern site, and for activating a pattern site that has been stored in cache memory on the module.

Active Site	Usage
StaticSite	A StaticSite provides a way to programmatically set the logical values of the signals in a site. This gives the opportunity to programmatically write the values to each channel. There are two primary uses of StaticSite: • Verifying the physical connections and software configuration of a pattern site. By using the static site methods to read and write the signals, it is possible to verify the connectivity without the overhead of creating a pattern. This permits verifying the mapping of signals to the module outputs and inputs, including the software layers and the physical wiring. • Utilizing extra bits on the module for programmed IO when there are leftover bits not being used for pattern IO or PPMU.
CaptureSite	CaptureSites provide a straightforward mechanism to capture data in the DSR. They are similar to a PatternSite except that no pattern definition is required. A capture site requires only the timing information for the site (in terms of a WaveformTable block), the DCLevels for the site (in terms of a DCLevels block), and instead of pattern definition, the amount of time to operate the capture. **Note:** CaptureSites are only supported on a single module.

For further overview on sites, see:

- Programming Guide:
  IVI-C and IVI-COM: ***"Active Sites"***
- LabVIEW Driver help:
  "Active Sites"

Depending on your configuration and your options, multiple sites can co-exist and run simultaneously on the DSR. For example, you can have a 4-channel PatternSite and a 4-channel CaptureSite running simultaneously, as long as certain channel assignment requirements are met.

The diagram on the next page shows several example single-site and multiple-site configurations. Keysight recommends that you review this diagram in detail.

Example single-site and multi-site configurations for a single-module M9195A and M9195B

PatternSite or CaptureSite (shades of blue) -- References to PatternSites implicitly include CaptureSites.

PpmuSite

StaticSite

Bank #	Channel #	Configuration 1	Configuration 2	Configuration 3	Configuration 4
Bank 3	15	StaticSite A	StaticSite A	PatternSite A	
	14	PatternSite D	StaticSite A	PatternSite A	
	13	PatternSite D	StaticSite A	PatternSite A	
	12	PatternSite D		PatternSite A	StaticSite A
Bank 2	11	PatternSite C		PatternSite A	StaticSite A
	10	PatternSite C	PpmuSite A	PatternSite A	
	9	PatternSite C	PpmuSite A	PatternSite A	
	8	PatternSite C	PpmuSite A	PatternSite A	
Bank 1	7	PpmuSite A	PpmuSite A	PatternSite A	
	6	PatternSite B		PatternSite A	HV channel 21
	5	PatternSite B	PatternSite A	PatternSite A	
	4	PatternSite B	PatternSite A	PatternSite A	
Bank 0	3	PatternSite A	PatternSite A	PatternSite A	PatternSite A
	2	PatternSite A	PatternSite A	PatternSite A	PatternSite A
	1	PatternSite A	PatternSite A	PatternSite A	PatternSite A
	0	PatternSite A	PatternSite A	PatternSite A	PatternSite A

NOTE: The phrase "multi-site" refers to multiple PatternSites and/or CaptureSites running simultaneously. It does not refer to, for example, having a PatternSite running at the same time as a StaticSite. In the four configurations above, only Configuration 1 represents a multi-site configuration since there are 4 PatternSites that could be running simultaneously.

Configurations 2, 3, and 4 are all single-site configuration since there is only one PatternSite in each configuration.

To enable multi-site operation on the M9195B, the S04 Multi-Site option must be purchased.

Configuration 2: If a PatternSite expands beyond a single bank, then only one PatternSite can be executed at a time. In this example, the PatternSite includes all channels of Bank 0 and two channels of Bank 1. Another PatternSite cannot be executed because the existing PatternSite is not constrained to a single bank.

In this configuration, the S04 Multi-site Enabled option is unnecessary because only one PatternSite is possible.

When a PatternSite crosses bank boundaries, the PatternSite channels should always be contiguous.

The channels not used in the PatternSite can be used for PpmuSites or for StaticSites. PpmuSites and StaticSites are not affected by bank boundaries and can contain 1-16 channels, depending on what channels are available. In this configuration, the PpmuSite contains 4 channels while the StaticSite contains 3 channels. Multiple StaticSites and multiple PpmuSites can be executing simultaneously, i.e. operation is not constrained to just one PpmuSite and one StaticSite.

Configuration 3: In this example, all 16 channels are dedicated to a single PatternSite.

Configuration 4: In this configuation, Bank 0 is used as a PatternSite while high-voltage channel 21 is also used. Because high-voltage channel 21 is being used, high-speed channel 6 will not be available (high voltage channels 20-23 map into high-speed channels 2, 6, 10, 14, respectively). There are two ways that channel 21 can be used:

1. Channel 21 could be included as part of the Site definition of the PatternSite. For example, the high voltage signal could be switched synchronously with channels 0-3 of the PatternSite (although not at the same speed as channels 0-3). If channel 21 is included in the PatternSite, then the PatternSite has crossed bank boundaries and no other PatternSites can be created.

2. Channel 21 could be contained within its own StaticSite as a means to programmatically control the high voltage signals. In this case, the PatternSite is constrained to one bank (channels 0-3) and it would be possible to implement other PatternSites in this configuration.

Configuration 1: Up to four independent PatternSites can be executed as long as the following requirements are met:

1. Each PatternSite is constrained to a single bank. In this example, four PatternSites are created, PatternSites A, B, C, and D. Each PatternSite resides in a single bank (but doesn't necessarily need to consume all of its channels).
2. For the M9195B, the S04 Multi-Site option must be purchased. This option is not required on the M9195A, the M9195A is always multi-site enabled.

If the S04 Multi-site Enable option is not purchased on the M9195B, only one PatternSite is allowed, which can occupy from 1-16 channels as shown in Configurations 2 and 3.

Channels not used in PatternSites can be used in PpmuSites and StaticSites.

Site Configuration Guidelines

Summarized below are the guidelines that apply to creating single-site and multi-site configurations. Note that “multi-site” refers to multiple PatternSites and/or CaptureSites that are active simultaneously. It does not refer to, for example, having a PatternSite active at the same time as a StaticSite.

NOTE Because of their similarities, references to PatternSites implicitly include CaptureSites.

1. The M9195A module includes a multi-site license (S04), which allows up to four PatternSites to be active simultaneously. The multi-site license is an option on the M9195B. To enable multi-site operation on the M9195B, the S04 multi-site option must be purchased.
2. The following rules apply to a single module with a multi-site license:
   - **a** If each of several PatternSites are constrained to occupy their own 4-channel bank, up to four simultaneous PatternSites can be active simultaneously on a single module.
   - **b** If a PatternSite is spread across more than one bank, only one PatternSite can be active at one time.
   - **c** If a M9195B module does not have a multi-site license, then only one PatternSite can be active at one time. This PatternSite can occupy from 1 to 16 channels.
   - **d** Any channels not used in a PatternSite can be used in a StaticSite and/or a PpmuSite. Multiple PatternSites and/or StaticSites can exist simultaneously.
3. The following rules apply to a multi-module instrument, which consists of 2 to 12 M9195B modules (the M9195A module cannot be configured as a multi-module instrument):
   - **a** Only one PatternSite can be active at one time, regardless of how many modules are in the multi-module instrument.
   - **b** Because only one PatternSite can be active in a multi-module instrument, the S04 multi-site license is not needed in a multi-module instrument.
   - **c** At least one channel of a multi-module instrument PatternSite must be in Bank 0 of the master module.
   - **d** Any channels not used in a PatternSite in a multi-module instrument can be used in a StaticSite and/or a PpmuSite. Multiple PatternSites and/or StaticSites can exist simultaneously.

Signals

The first step in creating a site is to define the Signals that will be in the site. Next, the Site name is defined followed by specifying the channels in the site. At the same time the channels are defined, the previously-defined Signal names are assigned to the desired channels.

Signal Direction

When defining a Signal, the DSR requires that you specify its Direction, that is, that you declare whether the signal is a DUT output, DUT input, or both (InOut). This specification permits the DSR to avoid driving pins that are DUT outputs and attempting to take input from pins that are DUT inputs. When a Signal is declared as InOut, that is, to be used as both an Input to the DUT and an output from the DUT, then the DSR applies either stimulus or measure responses on this channel, depending on the particular I/O operation.

NOTE "Inputs" and "output" are always with reference to the DUT. So "Inputs" are inputs to the DUT and "Outputs" are outputs from the DUT.

The PPMU ignores the Signal Direction setting in order to permit making the full suite of PPMU measurements on both DUT inputs and outputs.

Since the CaptureSite does not have a pattern, CaptureSites use the Signal Direction to determine whether a pin should be used to generate a clock or to capture data.

The following table lists interaction between Signal Direction and Site operations.

Table Summarizing Signal Directions and Signal Operation

Signal Direction*	*StaticSite*		*Pattern Site*		*Capture*
	Read Operation	Write Operation	Compare Operation	Output Operation	
In	Reads return 0 for any channel that is defined to be a DUT input (DSR output).	Performs a write operation to the DUT.	Error is generated if: **1** WFC's are defined that perform compares (parse + runtime). **2** Direction is inconsistent with the WFC's at compile time. See **NOTE 2**.	DSR asserts the signal on those channels specified as DUT inputs or DUT inputs/outputs.	DSR output (DUT input) is generated on those channels that have waveform table characters that specify output event(s).
InOut	If the DSR is set to DSR out, the DSR will switch to DSR in to perform the read, i.e. it will cease driving its output channel(s).	If the DSR is set to DSR in, the DSR will switch to DSR out to perform the write, i.e. it will begin driving its output channel(s).	The Fail bit is set based on the compare operation with DUT outputs.	Error is generated if: **1** WFC's are defined that perform output (parse + runtime). **2** Direction is inconsistent with the WFC's at compile time. See **NOTE 3**.	Error is generated on activation (see **NOTE 4** below).
Out	Read returns the DUT output value. See **NOTE 1**.	Writes do nothing on channels defined as DUT outputs. No error is generated if this is attempted.			DSR input (DUT output) is captured on those channels that have waveform table characters that specify a compare operation.

* Signal Direction is with respect to the DUT.

NOTES:

1 If a Static Site was performing DSR output (Direction = In), followed by an Inactivate, setting Direction = Out, and then an Activate, the DSR will still be driving its outputs. The DSR will cease driving its output upon the first read operation.

2 Compares are only supported when the DUT is supplying a signal (Direction = Out). Hence, this case generates an error.

3 DSR outputs can only be supplied to DUT inputs (Direction = In). Hence, this case generates an error.

4 Capture site signals must be either DUT inputs or outputs. Hence a Direction of Ipout is not supported for capture sites.

Active Sites

Active Sites are dynamic repeated capabilities (see "***Dynamic Repeated Capabilities"*** in the *KtMDsr Programming Guide* or the LabVIEW driver help file).

An active site is created by calling an *Activate* function and passing it a site name, which is a STIL component. The site contains a list of hardware channels and optionally other resources such as trigger paths. When a site is activated, a dynamic repeated capability is created that is used to perform operations on that active site. The name for the dynamic repeated capability is the site name. For example, to make PPMU measurements, a site is defined that contains the necessary channels, then that site is activated. Once activated, conventional repeated capabilities functions and attributes can be used to perform PPMU measurements on those sites.

Common Site Operations

- The Active Site is created with an *Activate* call (or one of the special activate calls for the pattern sites). The activate site name is then the name of the site itself; a new name is not created.
- When the application is done with the Active Site and the channels are to be used for other operations, the site is *Inactivated* rather than *Removed*. When a site is *Inactivated*, the site definition still exists by the same name; however, the Active Site no longer exists.
- In addition to *Inactivate*, active sites have a function called *InactivateAndDisable*. It performs an inactivate operation and also disables the signals in the site. That is, the drive is turned off and each channel is put into a high-impedance state. If a subsequent site is defined with different signal directions, then InactivateAndDisable should be used.

Transitioning Between Active Sites

When Ppmu, pattern, capture, or static operations are completed on a site, the channels can be released for use on another site by calling one of two functions.

The *InactivateAndDisable* function releases the channels and places each channel in a disabled state (similar to the IVI Disable functions). That is, the drive on all channels is shut off. If transitioning between sites that change the direction of the signals on the channels, InactivateAndDisable should be used when inactivating the first site. If the signal directions are changed in subsequent sites, InactivateAndDisable should be used in most cases.

The *Inactivate* function releases the channels, but does not change their state. This is useful if a signal needs to be retained between different operations. For example, an application may execute a digital pattern to configure a DUT, then transition to making Ppmu measurements without causing unwanted signal transitions on the DUT inputs.

Once a site has been inactivated, individual channels from the site may be used in other sites. This means it is possible to have shared channels between Sites defined, but you cannot have two Sites active at the same time with shared channels.

NOTE Signal lines are not driven until the new active site defines a signal level. For PatternSites and CaptureSites, this occurs when Initiate is called. For Static sites, this occurs when one of the write functions is called. For the Ppmu, this occurs when one of the force functions is called.

Generation of Results for PatternSites, CaptureSites, and StaticSites

The Generations of Results Diagram shows how the DSR module generates results for the four Site types based on the following parameters:

- The DUT output voltage relative to VOL and VOH. VOL and VOH are the DUT output low and high voltage thresholds, respectively.
- The position of VOL and VOH relative to each other. For normal operation VOH > VOL.
- The Pattern Site compare codes provided in WaveformTables.
- The types of results: Fail, Trinary, Valid, etc.

The comparators shown in the Generation of Results Diagram convey how the Compare Responses are generated based on VOL, VOH and the DUT output voltage.

Generation of Results for PatternSites, CaptureSites, and StaticSites

This diagram shows how the DSR generates results for PatternSites, CaptureSites, and StaticSites based on the following parameters:

- The DUT output voltage relative to Vol and Voh. Vol and Voh are the DUT output low and output high voltage thresholds, respectively.
- The position of Vol and Voh relative to each other. For normal operation, Voh > Vol.
- The PatternSite and CaptureSite Compare Codes that are provided in waveform tables.
- The type of results: Fail, Trinary, Valid, etc.

Comparators are shown below to convey how the Compare Responses are generated based on Vol, Voh, and the DUT output voltage.

These are the characters that appear in the pattern vectors, and can be any desired characters. These characters are not defined by STIL.

```
Timing basic  {
  WaveformTable one {
    Period '32ns';
     Waveforms  {
       All    {  01  { '0ns' 0/1;   '16ns' L/H;}}
     } // end waveforms
  } // end waveform table one
} // end timing basic
```

0 = ForceDown at 0 ns and a CompareLow at 16ns
1 = ForceUp at 0 ns and a CompareHigh at 16 ns

STIL-defined characters

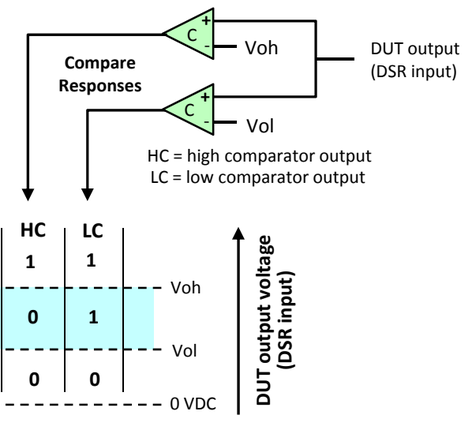
The generated WaveformTable will define the waveform characters for all of the defined Signals as follows:

0	ForceDown at DriveTime **"D" also used**
1	ForceUp at DriveTime
Z	ForceOff at DriveTime
P	ForcePrior at DriveTime
H	CompareHigh at CompareTime
L	CompareLow at CompareTime
T	CompareOff at CompareTime
X	CompareUnknown at CompareTime

Compare Codes

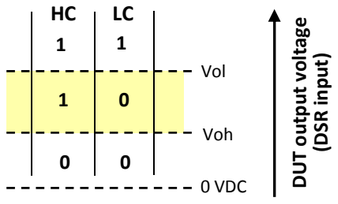
The comparators produce a 1 when the "+" input voltage is greater than the "-" input voltage.

Normal operation: Vol < Voh



These are the 3 normal Compare Responses that will be generated.

Unexpected operation: Vol > Voh



It is possible to set Vol above Voh, which produces what's called the "unexpected state" (yellow region) with compare codes of HC = 1 and LC = 0 (indicating that the voltage is above the high threshold and below the low threshold). This is not a useful state, so setting Vol and Voh in this manner should be avoided.

PatternSite and CaptureSite Results					
Compare Codes (these are provided in waveform tables)	Compare Response		Pattern site results		Capture site results
	HC	LC	Fail	Trinary	Fail
L = CompareLow (DSR input needs to be below Vol)	1	1	1	3 (11)	1
	1	0	0	0 (00)	0
	0	1	1	2 (10)	1
	0	0	0	0 (00)	0
H = CompareHigh (DRS input needs to be above both Vol and Voh)	1	1	0	0 (00)	0
	1	0	1	1 (01)	1
	0	1	1	2 (10)	1
	0	0	1	1 (01)	1
T = CompareOff (DSR input needs to be above Vol and below Voh)	1	1	1	3 (11)	1
	1	0	1	1 (01)	1
	0	1	0	0 (00)	0
	0	0	1	1 (01)	1
X = CompareUnknown (No comparison is performed.)	1	1	No results are generated, and a reference is returned to an empty array.		An error is generated, this isn't a valid Compare Code for capture sites.
	1	0			
	0	1			
	0	0			

Static Site Results				
Compare Response			ReadThreeStateVector	
HC	LC	ReadVector	Data	Valid
1	1	1	1	1
1	0	0	0	0
0	1	0	0	0
0	0	0	0	1

Unexpected state, i.e. DUT output voltage higher than Voh and lower than Vol.
Mid-band, i.e. DUT voltage higher than Vol and and lower than Voh.

Using PatternSites

PatternSites involve multiple DSR channels simultaneously sending and receiving signals to the DUT as directed by the test program.

For an overview of possible PatternSite configurations see the diagram: Common Scenarios for Creating PatternSites.

Programming Details

For specific details in programming PatternSites, refer the *Programming Guide*, the LabVIEW driver help system, and the Example folder:

- Programming Guide
  IVI-COM, IVI-C, or LabVIEW: ***"PatternSites Overview"***

For a graphical summary of the programming steps, see PatternSite Sequence.

PatternSite Options

- ConditionalLoopFailureMode
- ConditionalLoopSuccessMode
- EndBehavior
- ForceValidateEnable
- TriggerMode
- TriggerSource
- WhatToLog

Once a PatternSite is activated, changes to PatternSite do not affect the site. For example, if you change the trigger source, if won't affect sites that have already been activated.

For details for the sequence to program and execute a PatternSite, see:

- Programming Guide
  IVI-COM, IVI-C: ***"PatternSite Sequence"***
- LabVIEW Driver Help
  IVI-COM, IVI-C: ***"PatternSite Sequence"***

The End Behavior for pattern sites is respected after the pattern completes. Thus, clocks can be continuously generated while other channels are used for PPMU measurements or pattern or static operations.

When a channel transitions from being driven from a previous setting (because it was Inactivated and not disabled) to a new setting, the transition may pass through some intermediate values. The transition behaves as follows:

- If the level being driven from the previous operation is the same as the target level, the transition does not disturb the signal. Therefore if transitioning between sites that use DCLevels to define the signal levels, and if using the same DCLevels block, the transitions do not disturb the signals.
- If different DCLevels are specified between the sites, the new DCLevels settings are applied before new digital data is applied. If the old digital level is different from the new digital level, this can result in an intermediate level appearing.
- If the transition is from the PPMU to a site controlled with DCLevels, the DCLevels settings are applied before new digital data is applied—but before the PPMU is disabled. This momentarily activates both drive signals, resulting in a step occurring during the transition from one level to the next.
- If the transition is from a site controlled with DCLevels to the PPMU, the PPMU is turned on before the DCLevels are withdrawn. This can result in a step occurring during the transition from one level to the next.

Markers and Triggers

For PatternSites Markers and Triggers can also be added. Within a Site Markers0-2 and Triggers0-5 can be mapped to physical or software triggers. (“Loading STIL files in the IVI API” on page 76)

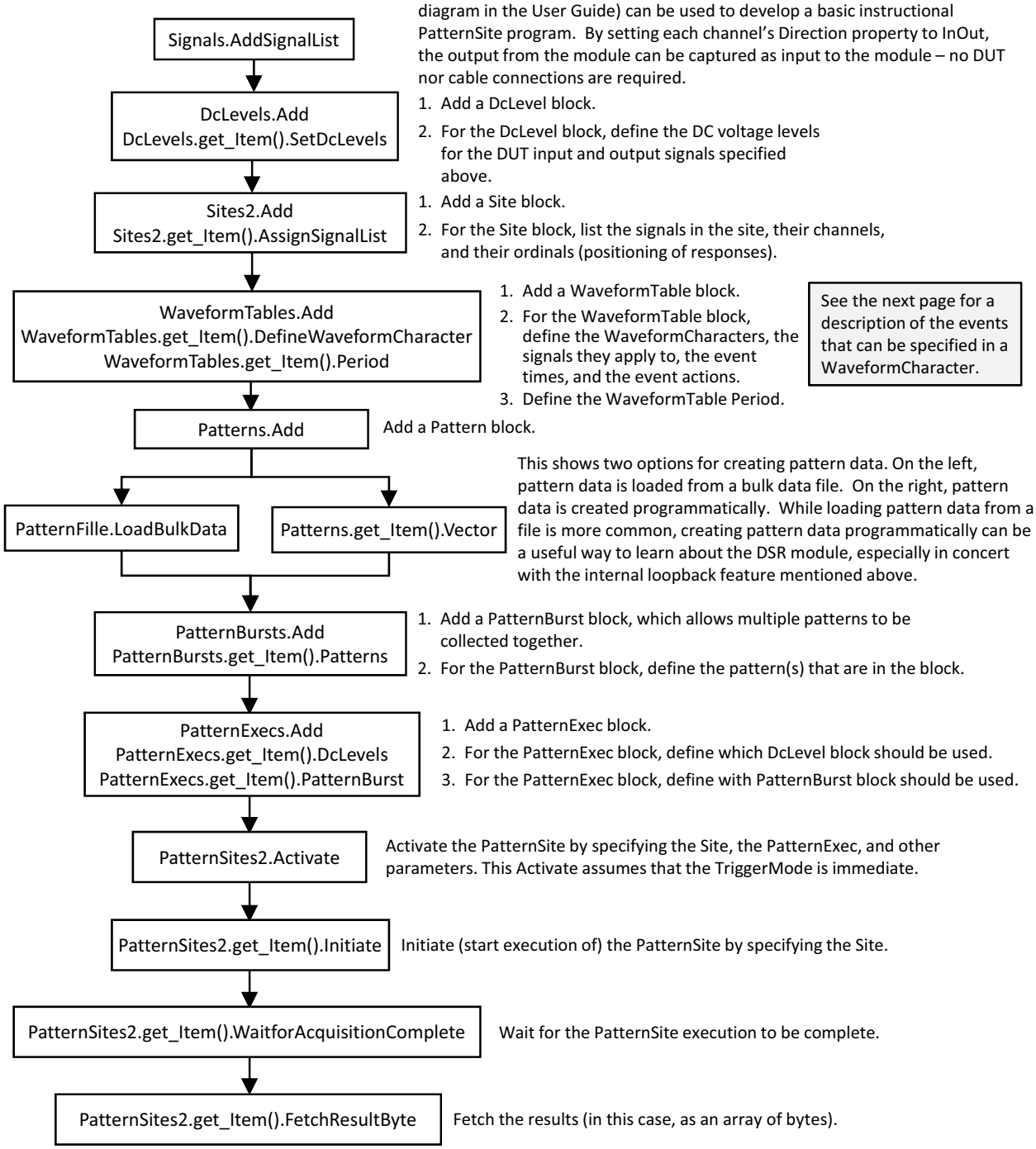
Along with this mapping, Triggers and Markers can be configured for Direction (Marker or Trigger), PulseWidth, Polarity, Termination, and ThresholdLevel. Hardware Trigger properties such as direction are also configurable.

- Programming Guide:
  “***Triggers and Markers”***

Most hardware Triggers are bidirectional (other than PXIe_DSTARB and PXIe_DSTARC), but can only be used in one direction at a time. Either as a Trigger (IN to the DSR) or a Marker (OUT of the DSR).

The examples folder, see *CS_UsingMarkers*.

PatternSite Sequence

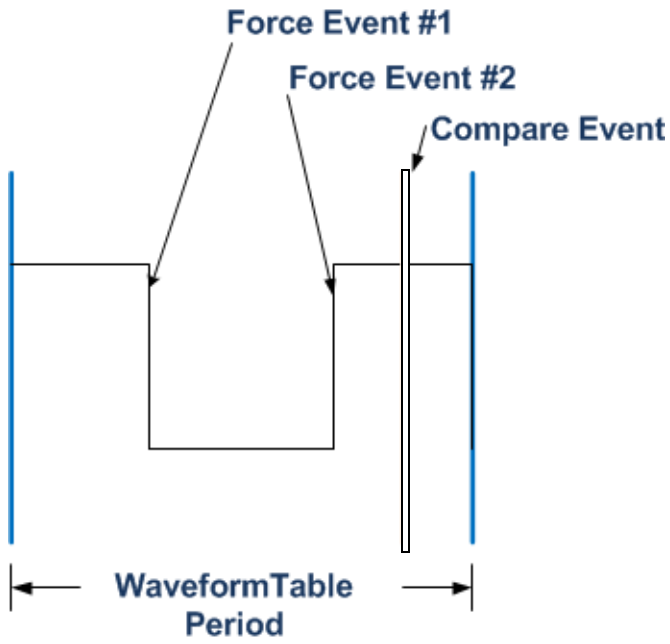


WaveformCharacter

Guidelines for creating WaveformTable characters, known as WaveformCharacters, are listed below:

1. A WaveformCharacter can specify a maximum of two Force Events and a maximum of one Compare Event. This allows 0, 1, or 2 Force Events, and 0 or 1 Compare Events during the WaveformTable Period.
2. For the maximum data rate of 250 MHz (corresponding to a WaveformTable Period of 4 ns), Force Events and Compare Events can be specified at 0 ns, 1 ns, 2 ns, or 3 ns. A time of 4 ns cannot be specified since it corresponds to 0 ns of the *next* period.
3. For the maximum data rate of 250 MHz (again, corresponding to a WaveformTable Period of 4 ns), the time between two consecutive Force Events must be 2 ns or greater. This requirement includes consecutive periods. For example, if a Force Even occurs at 3 ns in one period, a Force Event cannot occur at 0 ns in the next period.

The diagram below shows an example WaveformCharacter that is defined to have two Force Events and one Compare Event:



PatternSite Results

See the Generation of Results Diagram.

PatternSite Scenarios

PatternSite scenarios.

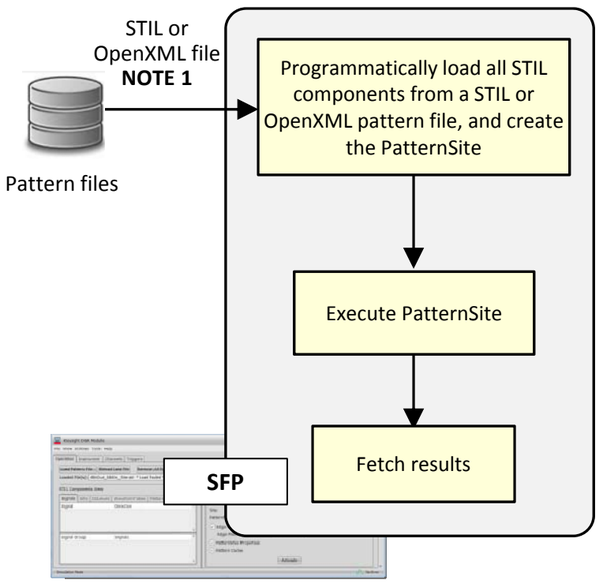
Common scenarios for creating PatternSites

This diagram shows several common scenarios for creating PatternSites. PatternSites can be created from pattern files or they can be created programmatically. Options 1 and 2 show two different ways of creating PatternSites from pattern files. In both cases, the DSR SFP can be used to perform the same operations.

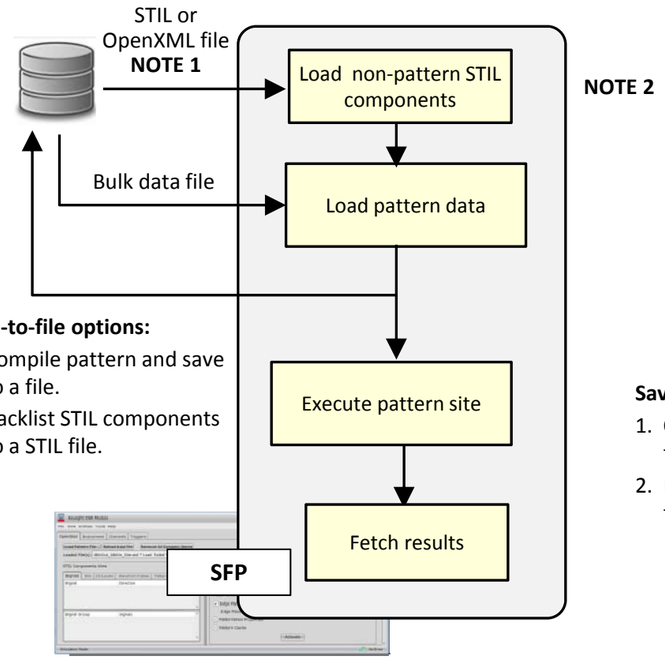
In Option 3, the non-pattern STIL components are created programmatically while the pattern data is loaded from a bulk data file.
In Option 4, all STIL components are created programmatically.

Options 2, 3, and 4 also show two different ways that the STIL components can be stored to a file.

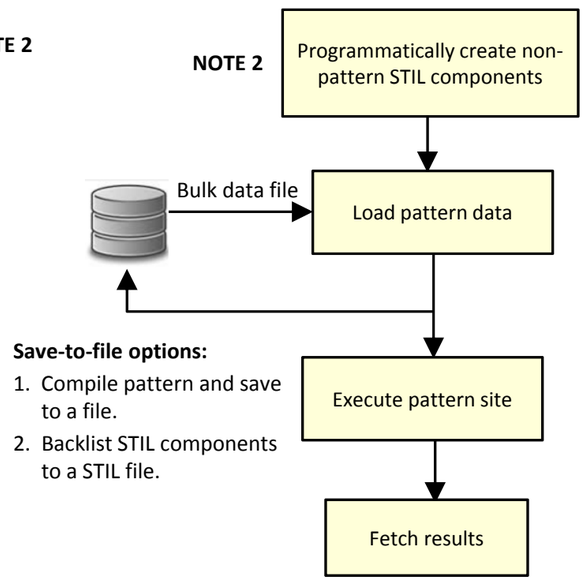
Option 1: All STIL components are loaded from a STIL or OpenXML file, and used to create the PatternSite. These files can also be loaded from the SFP and used to create a PatternSite.



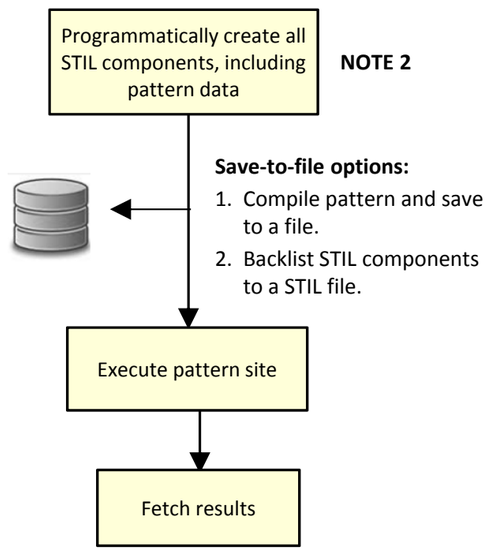
Option 2: Non-pattern STIL components are loaded from a STIL or OpenXML file while the pattern data is loaded from a bulk data file. These files can also be loaded from the SFP and used to create a PatternSite.



Option 3: Non-pattern STIL components are created programmatically while the pattern data is loaded from a bulk data file.



Option 4: All STIL components are created programmatically, including the pattern data.



NOTES:

1. STIL files are the preferred format of pattern files compared to OpenXML files.
2. The minimum required non-pattern STIL components are Signals, Sites, WaveformTable(s) and DCLevels.

Using CaptureSites

In CaptureSites, the DUT typically receives only a clock signal from the DSR and returns patterns to the DSR.

NOTE CaptureSites are not supported on M9195B multi-module implementations.

For Signal Direction in CaptureSites:

- Data may be captured from Signals defined as Out.
- Clocks are generated on Signals defined as In.
- InOut direction is not supported for CaptureSites.

NOTE Signal Direction is always relative to the Device Under Test (DUT). Part of defining a Signal is declaring the direction, whether In (DUT Input), Out (DUT Output), or both (InOut).

Programming Details

For specific details in using the IVI Functions in CaptureSites, refer to the *Programming Guide* or the LabVIEW driver help system

- Programming Guide:
  IVI-COM, IVI-C: "***CaptureSites Overview"***
- LabVIEW Driver help:
  IVI-COM, IVI-C: "***CaptureSites Overview"***

The diagram on the next page shows a typical CaptureSite Sequence.

Command Sequencing

The CaptureSite properties (RequestedResultCount, TriggerMode, TriggerSource) need to be set before activating a CaptureSite. Once a CaptureSite is activated, changes to CaptureSite do not affect the site. For example, if you change the trigger source, it won't affect sites that have already been activated.

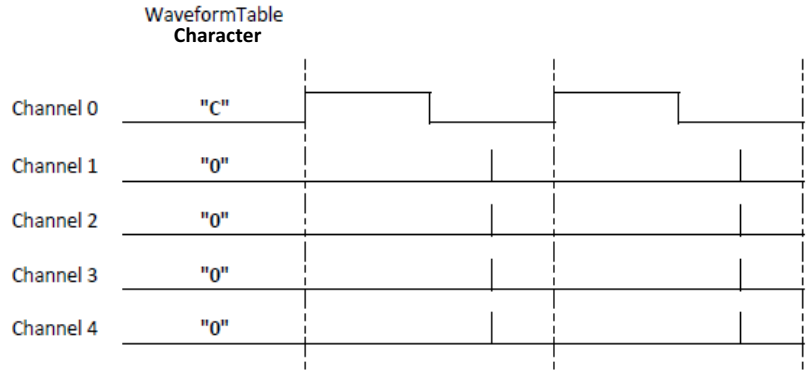
CaptureSite Results

See Generation of Results Diagram.

CaptureSite Sequence

For this CaptureSite, it is assumed that Channel 0 is programmed as a DUT input, and that the WaveformTable character "C" has been programmed to act as a clock, namely to go high at the start of the period and go low half-way through the period.

It is further assumed that Channels 1-4 are DUT Outputs (DSR inputs), and are sampled by the WaveformTable character "0" which uses 'L' (CompareLow) in its definition. This sampling occurs three-quarters of the way through each period as indicated by the tic marks.



Signals.AddSignalList

Specify the signal names and their direction for the 5 signals shown above.

DcLevels.Add
DcLevels.get_Item().SetDcLevels

1. Add a DcLevel block.
2. For the DcLevel block, define the DC voltage levels for the DUT input and output signals specified above.

Sites2.Add
Sites2.get_Item().AssignSignalList

1. Add a Site block.
2. For the Site block, list the signals in the site, their channels, and their ordinals (positioning of responses).

WaveformTables.Add
WaveformTables.get_Item().DefineWaveformCharacter
WaveformTables.get_Item().Period

1. Add a WaveformTable block.
2. For the WaveformTable block, define the waveform characters, the signals they apply to, the event times, and the event actions. In this example, it is assumed that the WaveformTable character "C" is used to generate a periodic clock on Channel 0 while the WaveformTable character "0" is used to sample the signals on Channels 1-4.
3. For the WaveforM9195-90002
4. mTable block, define the waveform period. This controls how often the input signals are sampled.

CaptureSites.RequestedResultCount

Specify how many times the waveform table characters will be repeated. For waveform table characters that specify compares (i.e. are doing input to the DSR), this count then specifies how many result are to be obtained.

CaptureSites.Activate

Activate the CaptureSite.

CaptureSites.get_Item().WaitforAcquisitionComplete

Wait for the CaptureSite execution to be complete (achieved when the RequestedResultCount number of inputs have been received).

CaptureSites.get_Item().FetchResultByte

Fetch the results (in this case, as an array of bytes).

Using StaticSites

For Signal Direction in StaticSites:

- Stimulus may be applied to Signals defined as either In or InOut.
- Response may be received from Signals defined as either Out or InOut.

NOTE Signal Direction is always relative to the Device Under Test (DUT). In = DUT input; Out = DUT output; InOut = both.

Programming Details

For specific details in using the IVI Functions in StaticSites, refer to the *Programming Guide* or the LabVIEW driver help system:

- Programming Guide:
  "***StaticSites Overview"***
- LabVIEW Help driver:
  "***StaticSites Overview"***

For details on the programming steps, see StaticStites Sequence.

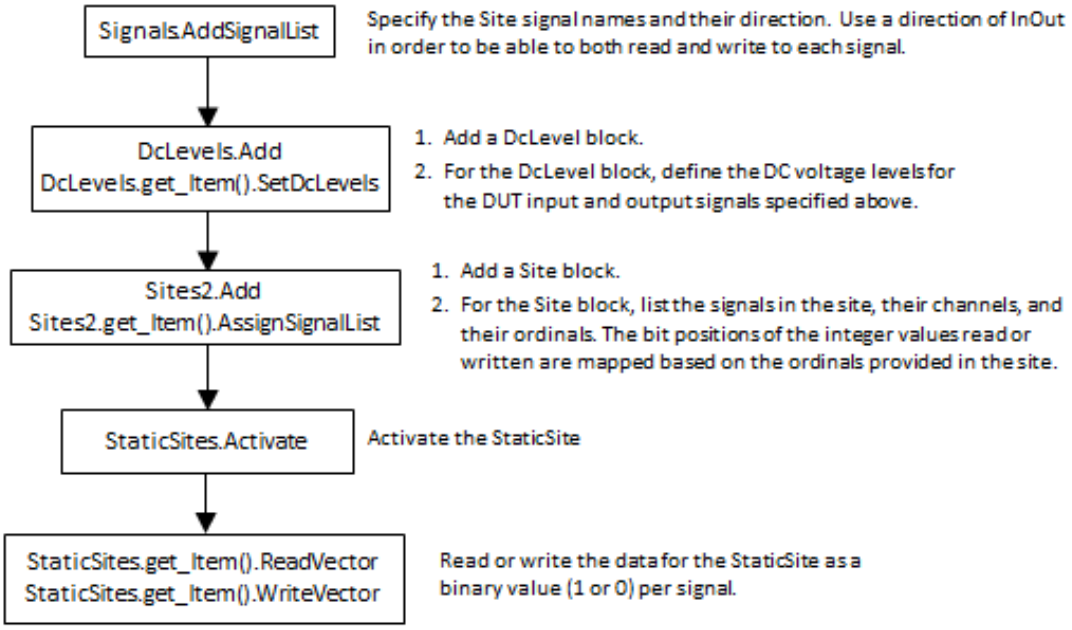
Static Site Results

See the Generation of Results Diagram.

StaticSite Sequence

NOTE Once a StaticSite is activated, changes to the Site do not affect its operation.

Provides the ability to perform static reads and writes on the channels



Using PpmuSites

PPMU Sites allow DC tests of DUT IO cells:

- Force Nothing, Measure Voltage (FNMV)
- Force Voltage, Measure Current (FVMI)
- Force Voltage, Measure Voltage (FVMV)
- Force Current, Measure Voltage (FIMV)
- Force Current, Measure Current (FIMI)

Programming Details

For specific details in using PpmuSites, refer to the *Programming Guide* and the LabVIEW help file:

- Programming Guide:
  IVI-COM, IVI-C: "***PpmuSites Overview"***
- LabVIEW Driver help:
  IVI-COM, IVI-C: "***PpmuSites Overview"***

NOTE

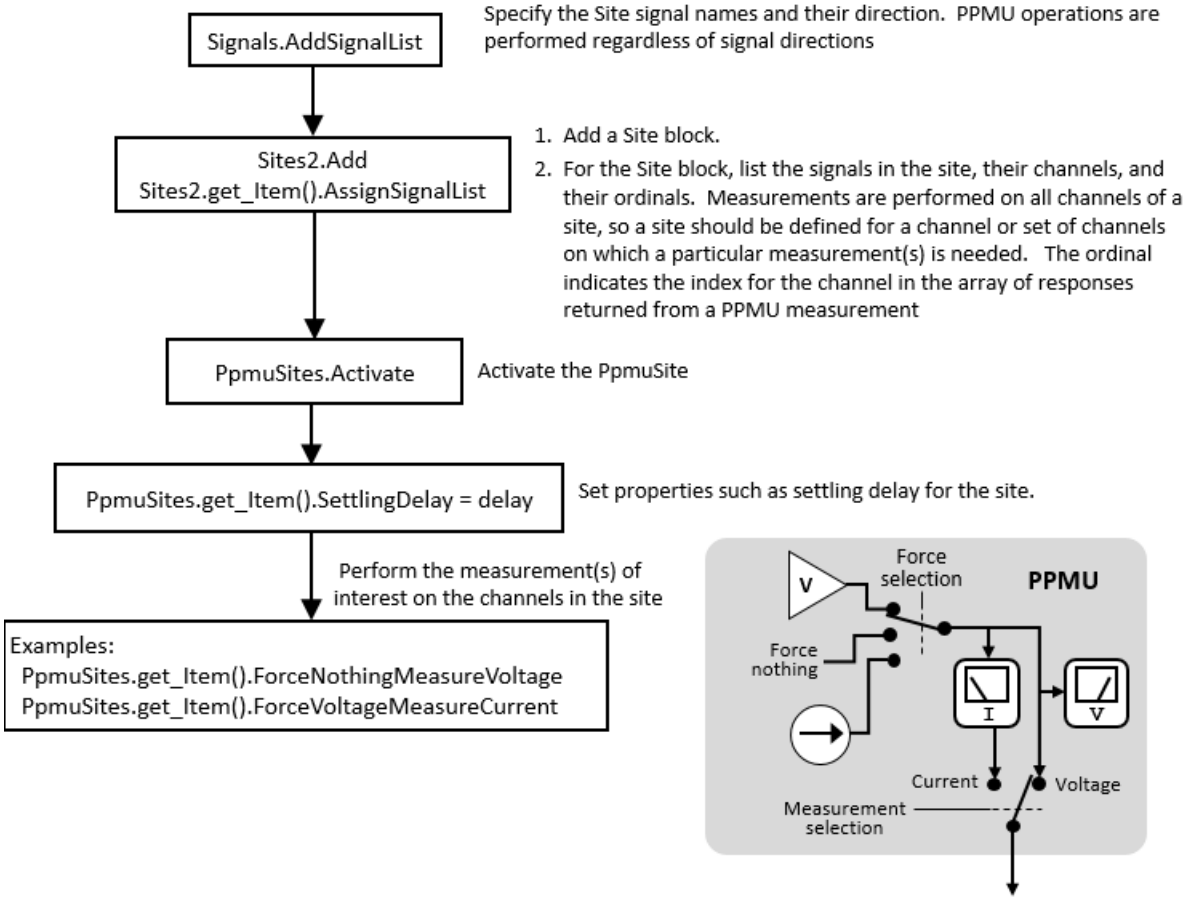
The current and voltage parameters cannot be set independently. The read-only Current Limit property returns which current range is used to make the measurement. This is a function of which shunt resistance is switched in to make the measurement, which in turn is a function of the expected current specified in the **ForceVoltageMeasureCurrent** method.

For the **ForceCurrentMeasureCurrent** and **ForceCurrentMeasureVoltage** methods, the specified force current sets the range. For the **ForceVoltageMeasureVoltage** method, the current range is set to 40 mA, since that gives the best chance of avoiding an overload condition.

PPMU Site Sequence

NOTE Once a PpmuSite is activated, changes to the Site do not affect its operation.

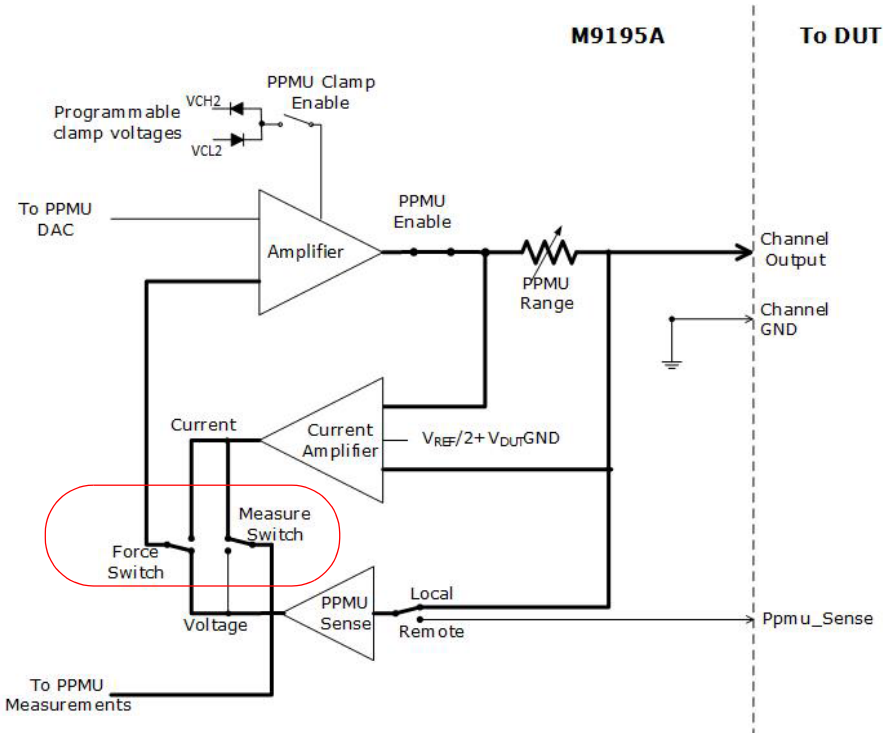
Provides the ability to perform per-pin parametric measurements on each of the channels. PPMU operations are conducted on each channel defined in the site; results are returned in arrays.



Example PPMU Operation

The following diagram and code example show demonstrate a ForceVoltageMeasureCurrent (FVMI) configuration.

NOTE In the simplified schematic below, reconfiguring the Force and Measure switches will setup different test types, including: FNMV, FVMV, FIMV and FIMI.



Each channel's measurement result is placed at the index in the array specified by the ordinal in the site definition. The maximum compliance voltage is dependent on the current range. Below is a applicable table of values for this test setup.

Current Range	Voltage
±40 mA	-2.0 to +5.75 V
±25 mA	-2.0 to +6.0 V
±2 mA (and less)	-2.0 to +6.5 V

A positive current measurement indicates a flow of current out of the DSR and into the DUT.

For details on setting PPMU clamps, see “Voltage Clamps” on page 23.

Example: Using ForceVoltageMeasureCurrent

The following C# program example demonstrates the complete process of setting the Ppmu to Force Voltage and Measure Current.

```
   public void ForceVoltageMeasureCurrentExample()
   {
      const string ppmuSiteName = "PpmuSite";
      string[] PpmuSiteSignalNames = {"Channel0", "Channel1",
"Channel2", "Channel3"};
      int[] PpmuSiteChannelNumbers = {0, 1, 2, 3};
      var Measurements = new Double[PpmuSiteChannelNumbers.Length];
      var VoltageLevels = new Double[] { -1.0, 0.0, +1.0, +2.0, +3.0 };
      Double ExpectedCurrent = 0.001; // actual measured current
depends on load

      Driver.Sites.Add(ppmuSiteName);

      foreach (int ChannelNumber in PpmuSiteChannelNumbers)
      {
         int Ordinal = ChannelNumber;
         Driver.Signals.Add(PpmuSiteSignalNames[ChannelNumber]);
         Driver.Signals.Item[PpmuSiteSignalNames[ChannelNumber]]
.Direction = KtMDsrSignalDirectionEnum.KtMDsrSignalDirectionInOut;
         Driver.Sites.Item[ppmuSiteName].AssignSignal(PpmuSite
SignalNames[ChannelNumber], ChannelNumber, Ordinal);
      }

      Driver.PpmuSites.Activate(ppmuSiteName);

      foreach (Double VoltageLevel in VoltageLevels)
      {
         Measurements =
Driver.PpmuSites.Item[ppmuSiteName].ForceVoltageMeasureCurrent(Voltag
eLevel, ExpectedCurrent);
         foreach (Double Measurement in Measurements)
      {
      Debug.WriteLine("Forcing: {0}, Measured: {1}", VoltageLevel,
Measurement);
      }
      Debug.WriteLine(Environment.NewLine);
      }
   }
```

4 Using STIL Files

"STIL (pronounced like "style") is the *IEEE Standard Test Interface Language (STIL) for Digital Test Vector Data*. STIL is a standard pattern definition language, defined by a series of IEEE 1450 specifications. STIL is a very complete language for defining stimulus and response. The Keysight DSR implements a subset of STIL, and also adds a few custom block types"

NOTE It is beyond the scope of this User Guide to fully describe the STIL language and all the ways it may be used. You should refer to the STIL (IEEE 1450) standards documentation for detailed information.

Overview

This section describes the STIL language and file format.

The STIL specification requires that the component definitions in a STIL file occur in a specified order.

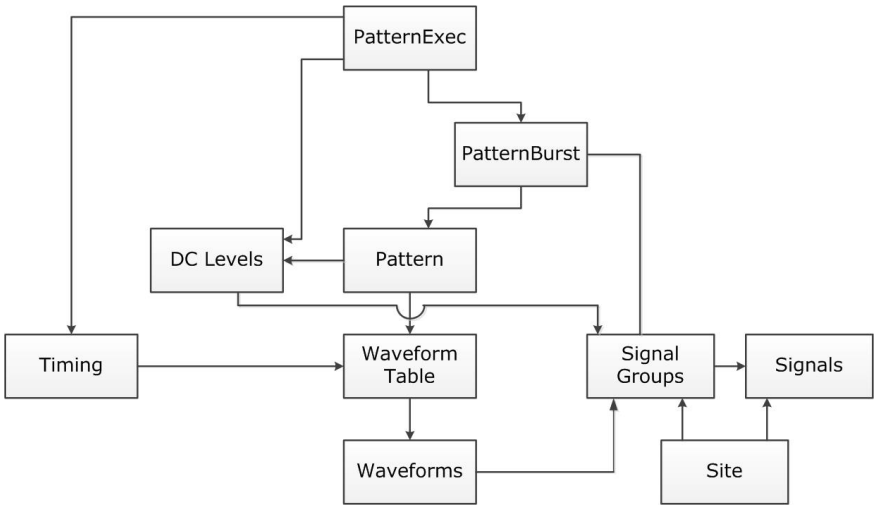
The Signals STIL block must be defined first, before any other blocks are used. After Signals, the other components may be defined in any order (since references to undefined STIL components are permitted from the API). Although the DSR accepts definitions in an arbitrary order, the STIL input file should be in the order defined by the IEEE 1450 specification.

The highest level structure in STIL (PatternExec) references all the pieces that go into a pattern. The individual components represent timing, amplitude, pattern, branching, etc.

Patterns are constructed from building blocks that are collectively called STIL components. The STIL components closely follow the block definitions that are provided by STIL (IEEE 1450). They are dynamically created and defined by either loading a pattern file or by programmatically providing the definition.

STIL Blocks

The following diagram illustrates the basic STIL building blocks and how they interrelate.



STIL Block Diagram

The following table lists the STIL elements in the *recommended* block order in a STIL file:

STIL Block	Description
Signals	The Signals block defines the signals that are used by the various blocks to compose the patterns. Signals must be defined in the signals block before they are referred to from one of the other blocks. Only one Signals block is allowed.
SignalGroups	SignalGroups permit referring to multiple signals with a single name. This simplifies defining items like DCLevels where numerous signals may need to be assigned at the same time. Only one SignalGroup block is allowed. A SignalGroup is defined with a string that contains one or more signals. There are basic operators available for referencing a series of signals and for adding and removing signals from a group.
Sites	The Sites block is a Keysight extension to the STIL standard. It specifies how a pattern definition is applied to the physical DSR. This includes: mapping the signals to DSR channels, specifying an ordinal for each signal and channel, and mapping the trigger references in the pattern.
DCLevels	The DCLevels block specifies two things. First, the voltage levels associated with either detecting or producing a logical 1 or 0. Second, the DCLevels also describe how each signal line should be electrically loaded. The DCLevels are defined independently for each signal in the pattern, so there is a dependency to the signal definitions.

STIL Block	Description
Timing (includes WaveformTable)	A WaveformTable defines the waveform that should be produced, or expected by the DSR when a particular waveform character is executed on a particular signal. Only one Timing block is allowed per WaveformTable. The waveform table name is used in the pattern to specify how each vector of the pattern should be executed. The waveform tables define the timing characteristics, including the total time for the bit period, and the time within the period when responses should be expected or stimulus levels generated. The timing information in the waveform tables may be specified independently for each signal; therefore, signals are dependent on the waveform tables.
PatternBurst	The PatternBurst is a list of pattern blocks. The PatternBurst structure permits specifying a list of patterns blocks to execute in a sequence. Patterns are permitted to be defined in terms of macros. The PatternBurst also has a reference to the macros used when evaluating the patterns.
PatternExec	The PatternExec is the highest level definition of a pattern. In the general IEEE 1450 (STIL) model it can have other information, but in the M9195A/B Soft Front Panel it only contains the list of pattern blocks that should be executed and the DCLevels for each signal in the block.
MacroDefs	The MacroDefs block identifies the pattern block used to fill in the macros specified when evaluating a pattern.
Pattern	The Pattern blocks contain the actual ones and zeros that are produced and expected by the DSR. The pattern is defined in terms of vectors that specify the values for all signals, and a reference to a waveform table. Each vector specifies the stimulus and/or expected response value for each signal in the pattern. The timing characteristics are permitted to change for each vector in the pattern; therefore, each vector has an independent waveform table defined.

General STIL Statements

The following general statements are supported:

STIL Statement	Description
STIL	The STIL version number for M9195A/B must be 1.0. If another version is encountered, M9195A/B will not process the file.
Header	Optional block after the STIL statement and before any other STIL block. Typically includes Title, Date, Source, and History statements.
Ann	Annotation statements are supported. Note that the text may span multiple lines.
Include	Include statement is supported. The `IfNeed` clause is not supported, any included files are loaded. File path resolution is supplied by the programming environment.
UserKeywords	UserKeywords may be used to specify the M9195A/B defined keywords.
User Functions	Not supported in the M9195A/B.

NOTE The concept of Inheritance is not supported in the M9195A/B DSR module.

STIL Block Names

The M9195A/B supports the following STIL blocks that are permitted to have names (domain names):

- PatternExec
- PatternBurst
- WaveformTable
- Pattern
- MacroDefs
- DCLevels
- Site
- Spec (only one Spec block may be defined in a STIL file)[1]
- Per STIL standard, any block can be defined once (and only once) without a name. This null-named block becomes the default block ('**!Default'**).
- Once the STIL file is loaded, all blocks are accessible from the IVI API by name. The default block (that is the unnamed block), can be referenced from IVI using the special name '**!Default**'. This token is a legal IVI repeated capability selector. '!' is a special character in STIL and should not be used elsewhere in a name.

1 STIL SPEC blocks are containers for Category information. Spec blocks are treated as global definitions in STIL.

STIL Signal and SignalGroups Blocks

Signals must be defined before they are used.

A *signal* is a symbolic representation of a single DSR channel; however, it is not associated with a physical channel. By separating the concept of a signal from a channel it is possible to define patterns that can be bound to a different set of channels. This separates the pattern definition from the details of the physical connector. It also permits the same pattern to be executed on multiple sites at the same time.

The Signals block differs somewhat from the other STIL components.

- The other STIL components are always referenced using the block name. For instance, the block name is passed to the Item method to access the block to configure it. The block name is also used whenever the block is referenced. Thus, the block name provides a way to manage multiple copies of the block.
- For Signals, the block name for a Signal is the name of that Signal. This signal name is used primarily in the pattern definition itself, so the block name is not directly used. Signals also differ from other STIL components because they are defined for the entire DSR (that is, they are global in scope).
- Signals must be defined before they are used.

Signals must be defined as **In, Out,** or **InOut.**

  - **Supply** is not supported.
  - **Pseudo** is not supported.

STIL Signal Blocks not supported in the KtMDsr IVI API

- Alignment
- Termination
- DataBitCount
- DefaultState

Supported STIL Signal Blocks

Supported STIL Signal Blocks	Description
Base	STIL permits a base to be specified for a series of bits. It may be Hexadecimal or Decimal. In either case, the specification is followed by a list of waveform characters that specify the mapping between bits in the number to waveform characters.
ScanIn/ScanOut	Scan is handled on arbitrary lines since it is expanded in software. Since any signal can receive Scan data, these attributes are not necessary for M9195A/B but are required by STIL. Therefore M9195A/B only permits shift on pins that are defined as ScanIn or ScanOut.

STIL Expressions

There are two expressions used frequently in STIL:

STIL Expression	Description
SignalExpressions	This is used for a group of signals. It is used to define a group (for example, `sig0...sig9`), but is also permitted in most circumstances where a group or signal is permitted. The syntax provides a convenient way to reference a series of signals
Numeric expressions	This specifies a time.

Signal Reference Expressions

When defining a pattern, it is frequently useful to deal with numerous signals at the same time. In many applications, such as when defining a bus, all signals may define the same waveform characters and DC levels. Signal Expressions provide a simple way to specify a group of signals.

Most DSR entries that accept a signal, also accept a signal expression. Signal expressions are created using the following syntax:

Expression	Definition
+ or ,	Signals can be added into the expression using plus or comma. So 'Signal1+Signal2' refers to both signals.
[..]	A range of signals that have a numeric suffix defined with square brackets can be added using '...'. For instance, if you have defined 'Signal[0]' , 'Signal[1]', and 'Signal[2]' , you can refer to all three with the syntax 'Signal[0..2]'.
-	Signals can be removed from an expression using minus. For instance, 'Signal[0..2]-Signal[1]' is the same as 'Signal[0]+Signal[2]'.
()	Parenthesis can be used for grouping operations.

In addition, a SignalGroup can be defined. In STIL, a Signal Group is defined using the SignalGroup block, which similarly permits defining a group as a signal reference expression.

Once defined, a signal group can be used just like a signal expression that includes each signal in the group. The '-' (minus) operation can be used to take signals away from a group within a larger expression. For instance, 'Group-Signal' or 'Group1-Group2' are both legal.

Examples of Signal Expressions

Expression	Examples	Note
The sum (or difference) of signals or signal groups.	Signal1+Signal2 D3+D2+D1+D0 SignalGroup1+ SignalGroup2	Operands may be signals or signal groups, separated by plus '+' or minus '-' signs, and grouped by parentheses.
Removing a signal reference expression from another signal or group	SignalGroup1 - Signal1	Signal1 must be part of SignalGroup1 in order to remove it from SignalGroup1.
A comma-separated list of Signals	Signal1, Signal2, Signal3, Signal10, Signal11, Signal12	
A range of Signals, in either ascending or descending order	Data[0..15] Data[5..0]	Data[0..15] yields 16 signals: Data[0], Data[1], ... Data[15] Data[5..0] yields 6 signals: Data[5], Data[4], Data[3], Data[2], Data[1], Data[0]
The sum (or difference) of signal ranges	Signal[0..7]+Data[0..3] Signal[0..7]-Signal[3] Data[0..15]-Data[2]	
Parenthesis for grouping	Data[0..15]-(Data[0]+Data[1])	

Numeric Expressions

The KtMDsr driver provides conventional numeric expression evaluation. There are several benefits:

- There is utility in having units conversions built into the pattern file formats. For example, using the unit designators, a time of 22 nanoseconds can be specified as '22 ns' instead of '22e-9 seconds'.
- The expressions can be used to evaluate times in the waveform table definition in terms of previous edges. For instance, the special character, '@' (commercial at) can be used to indicate the previously specified time. Using this, if an edge in a pattern specification were set to '@+5ns', the edge would be established relative to the previous edge.

- Most importantly, the expression can be reevaluated after the pattern has been compiled. This provides a way to change a family of related values based on a variable setting without the overhead of recompiling a pattern.

Evaluating expressions The KtMDsr expressions, in general, are strings that are evaluated to produce a number. The strings are reasonably conventional expressions that provide the usual meanings of the following.

Number parsing	Interpretation
+, -, *, /	Numeric operations with conventional precedence
^	Exponentiation
( )	Conventional meaning of parenthesis
;	Semicolon separates multiple operations; the result of evaluating a string is the value obtained from the final semicolon separated string.
<identifier>=<expression>	An equal operation assigns the expression to a variable, and implicitly creates the variable.
<identifier>	When used in an expression, then an identifier is treated as a variable and replaced with the value most recently assigned to it.
@<digit>	The commercial at (@) followed by a digit indicates one of the ten expression evaluations on a logical line. Thus @1 is the value of the first expression evaluated on a logical line. This is convenient when defining waveform tables, so that the latter times in a waveform table can be defined based on the earlier values.
@	The commercial at (@) without a digit indicates the most recently evaluated expression. The utility is similar to @<digit>.

Reevaluating expressions after compiling a pattern As noted, the primary utility of the expressions is that both DC Levels and times within a waveform table may be reevaluated without recompiling a pattern. For details, see the functions: **PatternSite > ReevaluateVariables** and **PatternSite > ReevaluateDcLevels** in the API.

Units The DSR handles all the conventional engineering prefixes. The DSR accepts physical units for inputs to help make the input more readable; however, it does not actually utilize the units in interpreting the numbers. The physical units accepted are seconds, Volts, and Amps.

The value entered	Interpreted as
4 ns	4E-9 seconds
.000001 kV	1E-3 Volts (i.e., 1 millivolt)
33 μs	33E-6 seconds (i.e., 33 microseconds)
1 μA	1E-6 Amps (i.e., 1 microamp)

STIL Site Block

For an overview of the architecture of sites, see "Sites" on page 29.

Sites is used to specify how a pattern is applied to the physical DSR. This includes:

- Mapping the Signals to DSR Channels
- Specifying an Ordinal for each Signal and Channel that specifies the position that Signal will take in response data.
- Mapping the Trigger references in the pattern to physical trigger sources.
- Mapping of Marker references in the pattern to physical marker destinations.

Since the Site specifies this physical mapping, it is used to bind Patterns, PPMU operations, and StaticSite operations each to physical channels.

A Site block specifies the mapping of the signals to physical pins on your test device.

Site Syntax

The Site syntax in a STIL file is:

```
Site <name> {
   Channels {
          ((SIGNAL) (DECIMAL_INTEGER) (DECIMAL_INTEGER);)*}
   }
}
```

The first parameter after the Signal is the channel number assigned to it. The second is an optional ordinal to specify the order in which data from this signal is packed into results.

Using Ordinals

An Ordinal specifies the bit location (or array location) where data associated with a Signal will be placed.

- In the case of DSR Pattern, Static and Capture operations, the ordinal is a zero-based bit position, which may be the same or different than the designated channel number for that Signal.
- For DSR PPMU operations, an ordinal specifies the array element that contains data for this Signal. This is the element number to use when packing measurement results into bytes or arrays.

The way that an ordinal is assigned to a Signal depends on the way the Signal or Site is defined.

- From the API, an ordinal is specified from IKtMDsrSite.AssignSignal or IKtMDsrSite.AssignSignalList.

- In a STIL file, an ordinal is specified in the Site block.
- In an OpenXML (spreadsheet) file, an ordinal is specified for a Signal on the Site sheet of the .xlsx file.

Trigger Specification

The **`Site`** also defines the physical signal used to Trigger the Watch/OnTrigger syntax.

- Sixteen software triggers are available for the DSR (SoftwareTrigger0 – SoftwareTrigger15) plus the hardware triggers: PXI_TRI[7:0], PXI_STAR, and PXIe_STARB.
- Trigger0 – Trigger5 are mapped to SoftwareTrigger0 – Software Trigger15 or to hardware triggers in the Site Definition.
- The trigger mode determines whether the pattern begins executing immediately (Trigger Mode = **Immediate**) after it is initiated or whether it waits for a trigger event (Trigger Mode = **Triggered**).
- The TriggerSource string is used to get or set the trigger source, which is then used to begin execution of a particular pattern.

Triggers are valid for Pattern Sites and Capture Sites. They begin the execution of the pattern on Initiate. If Trigger Mode is set to **Immediate**, the pattern begins executing immediately. If Trigger Mode is set to **Triggered**, the hardware fully initializes, then waits for a trigger event before proceeding.

The following syntax is used:

```
Triggers {
   (Trigger<n> EVENT;)*
}
```

where 'n' is a digit from 0 to 5. For example:

```
Triggers {
   Trigger3 "SoftwareTrigger0";
}
```

This example maps Trigger 3 to Software Trigger0

Marker Specification

The Site also defines the signals used to mark a vector (Trigger Out). The available Triggers are: PXI_TRIG(0-7), External1 & 2, PXI_STAR, and PXIe_DSTARC. Up to 3 markers are allowed per Site. The syntax is:

```
Markers {
                (Marker<n> EVENT;)*
}
```

Where ‘n’ is an integer 0-2. For example:

```
Markers {
                Marker0 “PXI_TRIG0”;
}
```

Triggers and Markers are bidirectional, but can only be used in one direction at a time for all Sites in a driver instance. Because of this, we need to explicitly set the direction in the triggers interface. See:

KtMDsr Programming Guide:
“Triggers and Markers”

In order to mark a Vector we will override the CrossReference (XRef) tag with a marker name. The marker will be output during the period of next vector after the XRef tag. For example:

```
Pattern "testpat"  {
    W WFT1;
V{ Bank0=TTTT; }
                                V{ Bank0=TTTT; }
                                V{ Bank0=1111; }
                                V{ Bank0=0100; }
                                V{ Bank0=0000;}
                                X Marker0;
                                V{ Bank0=TTTT; }   //
Marker0 output beginning at this Vector
                                V{ Bank0=TTTT; }
                                V{ Bank0=1111; }
                                V{ Bank0=0100; }

                                V{ Bank0=TTTT; }
                                V{ Bank0=TTTT; }
                                V{ Bank0=TTTT; }
                                V{ Bank0=TTTT; }
```

STIL DCLevels Blocks

The STIL (IEEE 1450.2) standard defines the DCLevels block. This is supported as follows:

General Syntactic Extensions per IEEE 1450.2

Generally, if a default DCLevels block is defined (one with no name), it is applied to all patterns. In addition, a DCLevels block may be provided by referencing it in either the PatternExec or Pattern itself. However, DCLevels can not be changed on-the-fly.

The following syntaxes are extended consistent with IEEE 1450.2:

- STIL block may specify 'DCLevels 2002;'
- The PatternExec may specify a DCLevels block.

DCLevels

DCLevels	Description
SignalGroups	This specifies the group definition that these DCLevels apply to.

NOTE **Inherit DCLevels** is not supported in the DSR module. Instead, the KtMDsr IVI driver uses Logic Families.

DCLevels parameters

DCLevels Parameter	Description
VIH	This defines the drive high voltage to be applied to Signals in the specified SignalExpression. In most situations, a single dc_expr is present. Multiple dc_expr statements are present to support changing values during one Vector.
VIL	This defines the drive low voltage to be applied to Signals in the specified SignalExpression. In most situations, a single dc_expr is present. Multiple dc_expr statements are present to support changing values during one Vector.
VCOM	This defines the input commutation voltage for differential Signals in the specified SignalExpression. VCOM is used to specify when IOH or IOL switch to drive or sync current. VCOM, IOH, and IOL are used together.
VOH	This defines the compare high voltage to be applied to Signals in the specified SignalExpression. In most situations, a single dc_expr is present. Multiple dc_expr statements are present to support changing values during one Vector.

DCLevels Parameter	Description
VOL	This defines the compare low voltage to be applied to Signals in the specified SignalExpression. In most situations, a single dc_expr is present. Multiple dc_expr statements are present to support changing values during one Vector.
IOH	This defines the output high load current to be applied to Signals in the specified SignalExpression. IOH is typically a negative current value.
IOL	This defines the output low load current to be applied to Signals in the specified SignalExpression. IOL is typically a positive current value.
VHYST	Hysteresis applied to the DUT output low and high thresholds, VOL and VOH. This is the peak-to-peak hysteresis voltage. Inputs are coerced to 0, 50 mV or 100 mV.
VHH	Voltage forced by the DSR on the high voltage pin(s) when high voltage is active. There are four high voltage channels. For example, if the DSR detects a low-to-high transition at 2.5 V, a 100 mV hysteresis would mean that the low-to-high transition is detected at 2.55 V and the high-to-low transition is detected at 2.45 V, a 100 mV difference. Noise could mean the signal crosses the threshold of 2.5 V more than once.
VIT	This defines the termination voltage for ResistiveTermination. ReistiveTermination is typically used to provide impedance matching to the DUT. This capability is typically implemented on ATE by using the internal impedance of the tester driver. This is distinct from usage of the Termination statement in 14.1 and 17.1 of the IEEE 1450-1999 Standard.

NOTE

ClampHi and ClampLo are not set in DCLevels. However, clamp voltages are defined in the Channel functions. For overview on this, see "Voltage Clamps" on page 23.

NOTE

DCLevel Limitations

- Power supply sourcing is not specified
- A single set of DCLevels may be specified for each Signal. Since Spec, Category, and Selector blocks are supported, DCLevels can be defined for multiple categories.
- DC Sets may not be specified.
- DCLevels can be changed on-the-fly.
- DC Sequence is not supported, consistent with the lack of support for power supply control.
- IEEE 1450.2 defines extensions to the waveform characters to permit designating additional force and compare voltages for a single signal. These are not supported.

STIL Timing and Waveform Tables Blocks

The M9195A/B supports up to 32 WaveformTables per PatternSite and up to 15 waveforms per signal. The following are the blocks allowed in a WaveformTable:

STIL Timing Block	Description
Period	Specifies the basic sample rate per vector. However, there can be four events within that period.
Waveforms	Waveforms are supported subject to the following:
SignalReferenceExpression	This is the specification of the Signals to which these waveforms apply. Subject to the same limitations as the SignalGroups definition.

Waveform Character Specification

The Waveform Table includes the specification of the waveforms for various waveform characters. The following describes the supported syntax:

- Waveform Character list may be specified and the '/' syntax used to separate event definitions.
- '@' and '@*n*' are used to incorporate the time expression from other times in this waveform in the calculation of a given time.
- M9195A/B supports up to 15 distinct waveform characters for each signal. Typical characters include: U (force up), D (force down), Z (force high-impedance), P (force prior), H (compare high), L (compare low), X or x (don't compare), T (three-state compare). Note that a sixteenth waveform character can be used to implement a waveform-character-scoped prior operation. You can specify this by leaving the waveform character absent.

NOTE Waveform Characters (WFCs) and Events are different concepts. WFCs are user defined, in terms of events.

- M9195A/B supports up to two force events per waveform period.
- M9195A/B supports one compare event per waveform.

The following are NOT supported:

- References to subwaveforms are not supported.
- '\rN' causes a sub waveform to repeated. Since sub waveforms are not supported this is not supported (only used with sub waveforms).
- Inherit Waveform Table
- SubWaveforms

NOTE The DSR module does not support Inheritance. Therefore, a Period must be specified in the Waveform Table.

Creating Waveforms and WaveformTables

Waveform Tables define the pattern application rate and define how the patterns are actually applied. A STIL test can have multiple waveform tables.

The waveform needs to describe every user definable character that is in the pattern for each signal. A simple WaveformTable might be:

```
Timing basic  {
   WaveformTable one {
      Period '38.46ns';
         Waveforms  {
            Sclk    {  T   { '0ns' U; '19.23ns' D; }}
            Sclk    {  01  { '0ns' D/U; }}
            Sdata   {  01  { '0ns' D/U; }}
            Sdata   {  Z   { '0ns' Z;   }}
            Sdata   {  LH  { '0ns' L/H; }}
            VIO     {  01  { '0ns' D/U; }}
      } // end waveforms
   } // end waveform table one
} // end timing basic
```

Waveform Tables have two basic elements: Period and Waveforms.

- Period defines the pattern or vector application rate. This is the period for a single pattern element. For example, `Period '38.46ns'` defines a 26MHz pattern application rate used for RFFE. Each vector or pattern is clocked every 38.46ns.
- Waveforms describes the actual waveform characteristics.

The WaveformTable translates user defined pattern characters to specific hardware actions. In the example above, we defined the characters 'T', '0', '1', 'Z', 'L', and 'H'. These characters are used in the actual patterns. The WaveformTable translates these pattern characters to the hardware specific functions 'U', 'D', 'Z', 'L', 'H', and 'X'. U/D/Z/L/H/X are STIL reserved characters that map to hardware actions.

The hardware specific functions are not user definable and include:

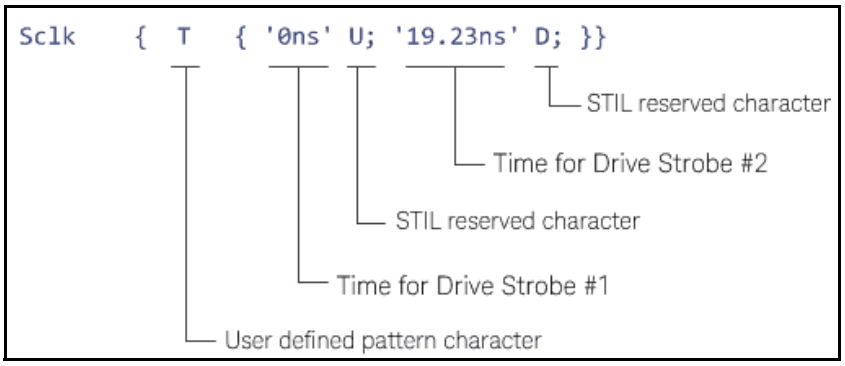
- 'U' (which means to Force High)
- 'D' (which means to Force Low),
- 'Z' (which means that driver is put into a high Z state). When this character is used, the voltage assigned to VIT is output from channel. If you do not specify VIT in DCLevels, when Z is applied, the channel is set to high impedance (Force Low).
- 'L' (which means that the tester pin is expecting to receive a low state (Compare Low))
- 'H' (which means that the tester pin is expecting to receive a high state (Compare High))

- 'X' (which means that we are receiving a don't care state (No Compare)).

In the example above:

- we used a user definable character 'T' to mean a toggle in the pattern. In the following waveform statement:

```
Sclk    {  T   { '0ns' U; '19.23ns' D; '19.23ns' L;}}
```



'T' is the user defined pattern character. The 'U' and 'D' are the STIL reserved characters that describe the specific hardware function. When programming the Sclk pin in the pattern, you will see the character T. When the pattern is applied, T means drive Up at 0ns (this is the first strobe) and drive low at 19.23ns (this is the second strobe). The drive strobes are relative to the Period that was defined.

In this case, T creates a 26MHz clock in a single vector of the pattern. You don't need to use two vectors to create the clock. Normally, if you use two vectors you would try to drive a 0 then 1, etc. That would result in a 26/2 MHz clock. Using T allows Double Data Rate, and the 0/1 occurs within the same period...or 26MHz.

- we also used '0' and a '1' to mean a drive low then high state, respectively. In the statement:

```
Sclk    {  01  { '0ns' D/U; }}
```

'0' and '1' are the user-defined characters. 'D' and 'U' are the STIL reserved characters. This line is interpreted wherever the Pattern has a '0', then at 0 nS, the line is driven Down. Likewise when the pattern has a '1', then at 0 nS, the line is driven Up.

- 'Z', 'L', and 'H' are the same as the hardware specific functions. They could have different meanings if it makes more sense to the programmer.

```
Sdata   {  Z   { '0ns' Z;   }}
Sdata   {  LH  { '0ns' L/H; }}
```

In the first of these two lines, 'Z' is used as a user-definable character which, in this example, has the same meaning as the STIL reserved character; namely, a high impedance state. This works only if VIT is

assigned NaN. If VIT is set to some voltage in the STIL file, the DUT will see 50 ohms into some voltage which could be zero or anything that makes sense to the driving circuit.

In the second line, 'L' and 'H' are the User-defined characters which mean the same as the STIL characters. This line is interpreted wherever the Pattern has an 'L', then at 0 nS, the line is driven Low. Likewise when the pattern has an 'H', then at 0 nS, the line is driven High.

NOTE You can define multiple WaveformTables and can change which WaveformTable to use.

STIL Pattern Blocks

Patterns is one of the STIL Components. As with all STIL components, they can be created using the IVI API, or by loading a file in one of the file formats supported by the DSR software.

A Pattern block provides the data that is used to stimulate the DUT and the data that is expected back from DUT. This data includes the stimulus and response values and also various non-data instructions.

The data output on each signal in the pattern in a single WaveformTable period is referred to as a Vector. Each Vector in the Pattern can be output using a different WaveformTable. Thus, each vector in the pattern includes the Signals being defined, the waveform character for each signal, and the WaveformTable that defines the waveform character and specifies the period for this cycle.

The *Vector* method may be called multiple times to provide a complete definition of a clock cycle. This permits independently specifying signals or signal groups with separate calls. Vectors may be labeled using *LabelVector* or from within a loaded file. This makes them available as locations accessible with GoTo as well as Pattern Update methods.

Creating Patterns

The Pattern blocks contain the actual ones and zeros that are produced and expected by the M9195A/B. The pattern is defined in terms of vectors that specify the values for all signals, and a reference to a waveform table.

Each vector specifies the stimulus and/or expected response value for each signal in the pattern. Here is an example:

```
Pattern Read_data_pattern  {
   WaveformTable one;
      //This refers to the WaveFormTable in the Timing block.
   V {'Sclk+Sdata+VIO'=000; }
      //The first vector needs to define a state for every signal.
      //In this case, all pins driven to 0. Remember that the 0
      //is a user definable character. The WaveformTable translates
      //the 0 to a hardware specific action. In this example, a 0 is
      //defined to be a D (or drive down action at 0ns into the v
      //vector period).
Loop 0 {
   V {'Sclk+Sdata+VIO'=T01; } // 1st instance of the Vector
   V {'Sclk+Sdata+VIO'=T11; } // 2nd instance of Vector
   V {'Sclk+Sdata+VIO'=T01; } //3rd instance of Vector
   V {'Sclk+Sdata+VIO'=T11; } // 4th instance of Vector
   }
} // end Pattern
```

This example specifies three channels: Sclk, Sdata, and VIO. These three all start with the User Defined character '0' which in our Waveform Table meant the line is driven Down.

Then we crate an indefinite program loop. At the first instance of the Vector, Sclk is Toggled, Sdata is driven Down, and VIO is driven Up.

At the second instance of the Vector, Sclk is once again Toggled, Sdata is driven Up, and VIO is driven Up.

At the third instance, Sclk is once again Toggled, Sdata is driven Down, and VIO is driven Up.

At the fourth instance of the Vector, Sclk is once again Toggled, Sdata is driven Up, and VIO is driven Up.

There are many different pattern execution control words like Loop that determine how your apply the pattern. In this case, Loop 0 creates a infinite loop. Infinite loops are useful in debugging patterns where you may need to connect an oscilloscope and look at the pattern.

Supported Non-Data Instructions

NOTE The timing characteristics are permitted to change for each vector in the pattern; therefore each vector has an independent waveform table defined.

Instruction	Notes
CrossReference	Places a mark in the pattern that is included in the response log. This simplifies relating the response log to the line in the pattern that generated it.
GoTo	Forces the execution to transfer to a locations identified with a label.
Loop	Indicates a block to be repeated a fixed number of times.
Macro	Executes the pattern from the specified macro. Optionally includes parameter substitution if defined using the *MacroWithSubstitution* function.
MatchLoop	Indicates a block should be repeated until all Compare operations pass, or a specified number of loops are completed.
Shift	Shift is a parameter substitution operation. It indicates that the parameters from the input parameter list should be executed until the list is exhausted. Shift may only be used within a Pattern that is designated as a macro using MacroDef.
WatchLoop	Indicates a block should be repeated until an external trigger event is received, or a specified number of loops are completed.
Pattern Labels	Pattern labels are supported. They are used with Goto statements, and to flag special statements to the M9195A/B.

Each instruction is programmed using an Index that indicates the location within the pattern. When the Pattern is completely defined, the various instructions are sorted by Index and executed in that order. There are performance benefits to using consecutive Indexes, but Indexes can be assigned in any order. Although the Vector function may require multiple invocations to fully define a vector, no two distinct instructions can have the same Index.

In general, the mechanism for capturing data is to specify a CompareLow operation on the Signals where the data is being captured. Then the DSR is configured to log all compares. Thus if the input is low, the compare succeeds and '0' is logged; if the input is high, the compare fails and '1' is logged. Thus the log represents the data values of the input.

Cyclized Data

Cyclized data is specified as: **Signal Reference Expression = Vector Data**

The Signal Reference Expression has the same syntax limitations as described for SignalGroups.

Vector data may be specified in:

- Waveform Characters
- Hexadecimal

Non-Cyclized data

STIL permits specifying non-cyclized data. The Keysight M9195A/B does not support this. Multiple-bit cyclized data is not supported.

STIL Expressions not Supported:

STIL Expression	Description
\w	Set to waveform characters
\rN xxxx	Repeat the character XXXX N number of times. (e.g.; '\r2 ab' is the same as 'abab', which depending on the context may be a hex number or a waveform character). **Note, this is not supported in the KtMDsr driver.**
\h	Set to hex, use the conversion from hex to waveform characters specified in the signal/groups
\hCHARS	Set to hex, use the specified characters to convert from hex to waveform characters.
\d	Set to decimal, uses the conversion from hex to waveform characters specified in the signal/groups
\dCHARS	Set to decimal, use the specified characters to convert from hex to waveform characters.
\iN	Necessary to supported multi-bit specifications.

STIL Macros Blocks

Macros

Macros are supported. They are expanded when a STIL input is compiled.

Constant Substitution

Macros permit constant substitution using the '%' syntax. A '%' in the Macro definition is replaced by a waveform character specified in the Macro invocation. For example, the invocation:

```
myMacro { mySignal = "W" }
```

Then, in the body of the macro, the sequence:

```
Vector { All = {AAB%CCD ; }}
```

Is expanded to:

```
Vector { All = {AABWCCD ; }}
```

% substitution is supported.

Scan substitution with # and Shift

Macros permit a string of values, typically the values used to setup a scan chain, to be specified using a macro. The string of values is specified on the macro invocation, and the character '#' is used to indicate where the values should go in the resulting expansion.

For example, Macro Definition:

```
MyMacro {
      V { All=1111111; }
      V { All=0000000; }
      Shift {
                V { All=AAB#CCD; }
      }
}
```

Calling the macro from a pattern:

```
Pattern TestPattern1 {
          W one;
. . .
          Macro MyMacro {mySignal=WXYZ;}
```

```
    .   .   .
}
```

The call to MyMarco would be equivalent to:

```
        V { All=1111111; }
            V { All=0000000; }
            V { All=AABWCCD; }  // First shift, WFC W is
  substituted
            V { All=AABXCCD; }  // Next shift, WFC X is
  substituted
            V { All=AABYCCD; }  // Next shift, WFC Y is
  substituted
            V { All=AABZCCD; }  // Next shift, WFC Z is
  substituted
```

This assumes that the waveform characters 1,0,A,B,C,D,W,X,Y,Z are all defined.

As part of the expansion, shift statements are expanded into the pattern. Therefore, the expanded macro consumes the same amount of pattern memory it would if the SCAN were manually expanded by the customer.

The Shift and # syntaxes are accepted.

Scan Data

Scan data is supported, but is converted into pattern data in a Macro expansion (1 vector for each scan step). This expansion is done when the pattern is compiled, and therefore, scan data can only be used when invoking a macro.

KtMDSR-Specific STIL

Backus-Naur Form Description for KtMDsr STIL

Notes:

1. Language constructs from STIL.1 and STIL.2 are so noted by comments
   a. STIL.1 = PatternBurst-Fixed
   b. STIL.2 = DCLevels, PatternExec-DCLevels
2. KtMDsr stil allows only one each of the certain block types (see Note 3, below) which may be either:
   a. an unnamed global block
   b. a named block with a direct reference
3. The following UserKeywords have been added;
   a. Site – a new top level STIL block construct used to define signal name to tester channel mapping and data logging
   b. Channels – a sub block within a Channel block to define Signal to physical channel mapping.
   c. Triggers – a sub-block with a Site block to define triggers to be used with a pattern
   d. Markers - a sub-block with a Site block to define Markers to be used with a pattern
   e. WatchLoop – new pattern statement to loop until trigger condition is meant or count is exceeded.
   f. VCOM, VHH, VIT, VHYST – additional DCLevels.
4. Block ordering is as defined in STIL0, Clause 7.1

Legend:

- All tokens are white space separated (new-line has no special significance)
- Bold tokens are Literals
- “{}” Braces refer to literal braces in the input, including the productions “{*” and “*}” where the star appears literally in the input.
- “;” Semicolons refer to literal semicolons in the input
- “()” Parenthesis set off a group of tokens and may be followed by “*” indicating 0 or more, or “+” meaning one or more, or nothing meaning optional.
- “<>” Angle brackets set off a group of tokens and indicates that one and only one may be used.

- // is a comment in the grammar itself
- Tokens in little-caps refer to syntactic elements defined by STIL, and generally the token describes the usage. Where tokens in little-caps have quotes around them, they must be quoted strings in the input.
- "'" single quotes are literals and indicate a timing expression.

STIL 1.0 { Design 2005; DCLevels 2002; }

Header {

(**Title** "TITLE_STRING";)

(**Date** "DATE_STRING";)

(**Source** "SOURCE_STRING";)

(**History** {

(**Ann** {* ANNOTATION **}**)*

}) // *end History*

} // *end Header*

(**Ann** {* ANNOTATION **}**)*

UserKeywords Site Channels Triggers Markers WatchLoop VCOM VHH VIT VHYST; // *DSR specific*

Signals {

(SIG_NAME < **In** | **Out** | **InOut** >;)*

(SIG_NAME < **In** | **Out** | **InOut** > {

(**Base** < **Hex** > WAVEFORM_CHARACTER_LIST ;)

(**ScanIn** (DECIMAL_INTEGER) ;) // *length required when Base Hex*

(**ScanOut** (DECIMAL_INTEGER) ;) // *length required when Base Hex*

})* // *end sig_name*

} // *end Signals*

(SignalGroups (DOMAIN_NAME) **{**

(GROUPNAME = *sigref_expr*;)*

})* // *end SignalGroups*

(**Site** (SITE_NAME) {

(**Channels** {

(SIGNAL_NAME

DECIMAL_INTEGER // *tester channel number*

(DECIMAL_INTEGER) ;)+// *the order in which data is packed into results*

}) // *end _Channels*

(**Triggers** {

(<TRIGGER0|TRIGGER1|TRIGGER2|TRIGGER3|TRIGGER4|TRIGGER5> "TRIGGER_SOURCE";)* })?

// Trigger source is

// string defining the physical trigger input.

(**Markers** {

(<MARKER0 | MARKER1 | MARKER2 | > "MARKER_SOURCE";)* })?

// Marker source is

// *string defining the physical marker output.*

})* // *end _Site*

(**Spec** (SPEC_NAME) { *// this block statement defines variable values for a given category, SPEC_NAME is ignored*

(**Category** CAT_NAME { // *CAT_NAME is ignored*

(**VAR_NAME** = *time_expr;*)* // *define only the Typ value*

})* // *end Category*

}) // *end Spec*

(**DCLevels** (DC_LEVELS_NAME) { // *STIL.2*

(*sigref_expr* { // *individual settings may be in any order*

(**VIH** '*dc_expr*';)

(**VIL** '*dc_expr*';)

(**VOH** '*dc_expr*';)

(**VOL** '*dc_expr*';)

(**VIT** '*dc_expr*';)

(**VCOM** '*dc_expr*';)

(**VHH** '*dc_expr*';)

(**IOH** '*dc_expr*';)

(**IOL** '*dc_expr*';)

(**VHYST** '*dc_expr*';)

})* // *end sigref_expr*

})* // end DCLevels

(**Timing** (TIME_DOMAIN_NAME) {

WaveformTable WFT {

Period '*time_expr*';

Waveforms {

(*sigref_expr*

(WFC {

('*time_expr*' *event* ;)+

})* // *end wfc*

(WFC_LIST {

('*time_expr*'
<*event_list* | *event* > ;)*

})* // *end wfc_list*

})**+** // end Waveform

} // end Waveforms

} // end WaveformTables

}) // end Timing

PatternBurst PAT_BURST_NAME {

(**SignalGroups** GROUPS_DOMAIN **;**)*

(**MacroDefs** MACROS_DOMAIN **;**)*

(**PatList** {

((PAT_NAME);)*

})+ // *end of PatList*

} // *end of PatternBurst*

PatternExec (PAT_EXEC_NAME) {

(**Timing** TIMING_NAME ;)

(**PatternBurst** PAT_BURST_NAME ;)

(**DCLevels** (DC_LEVELS_NAME);) // *STIL.2*

} // *end PatternExec*

MacroDefs (MACRO_DOMAIN_NAME) {

(MACRO_NAME {

(**PATTERN-STATEMENT** | **Shift** (vec_data | #))*

})* // *end macro_name*

} // *endMacroDefs*

Pattern PATTERN_NAME {

(LABEL :)

(**V(ector)** { (cyclized-data)* })

(**C(ondition)** { (cyclized-data)* })

(**Macro** MACRONAME ;)

(**Macro** MACRONAME { (scan-data)*(cyclized-data)* }

(**Loop** LOOPCNT { (pattern-statements)* })

(**MatchLoop** < LOOPCNT | **Infinite** > {

(pattern-statements) +—

}) // *end MatchLoop*

(**Goto** LABELNAME ;)

(**WatchLoop** < LOOPCNT | **Infinite** >
<TRIGGER0|TRIGGER1|TRIGGER2|TRIGGER3|TRIGGER4|TRIGGER5> {

(**pattern-statements**) +

})

(**X** < "IDENTIFIER" |INTEGER|MARKER0|MARKER1|MARKER2> ;) //
STIL.1

} // *end Pattern*

NameSpace Notes

- To support managing multiple unrelated patterns in memory at the same time, the DSR supports a NameSpace. The symbols that are used are uniquely qualified with the NameSpace so that the same symbol can be used in multiple pattern files with distinct meanings.
- Signals are an exception, and are always in the Default NameSpace. SignalNames get a namespace as well.
- When a file is loaded, all the symbols defined, except for Signals, are placed into the designated NameSpace. All symbols referenced, except for Signals, are referenced within the designated NameSpace. This means that <namespace> is prepended to each name.
- To reference symbols from the API that are within a NameSpace, precede the symbol name with the NameSpace name and an exclamation mark (!). For example, 'MyNameSpace!WaveformTable1'.
- If the NameSpace is null, all symbols are put into the default NameSpace.
- Multiple files may be loaded into the same NameSpace.

Loading STIL files in the IVI API

This section provides a brief introduction to loading STIL files, checking for errors, and storing STIL files. It briefly lists the methods/properties that load STIL files.

Load STIL File

IVI Method: PatternFile > PatternFileLoad

This function loads a STIL, OpenXml, or bulk data file and creates objects based on the content of the file. If an element is loaded and there is already one by that name in the same namespace, an error is generated.

- If the file suffix is .stil, the file is parsed as STIL. If the file suffix is .xlsx, the file is read as an OpenXML file with the DSR defined STIL data. Otherwise, the file is read as an ASCII text file that contains bulk pattern data in the DSR defined format.
- If the file is in the BulkData format, the file must include a header (see "Using Bulk Data Files" on page 79) that defines the Signals to which the waveform characters are to be applied, as described in bulk data file format. The name of the generated pattern block is the base portion of the file name. For example, a file named **patternfile.txt** would produce a pattern block named **patternfile**.

- If an OpenXML file is loaded that does not contain either a PatternExec or a PatternBurst, they are created automatically for that file. If additional OpenXML files are loaded simultaneously, the names of the generated blocks may collide with the names created for the original file. Therefore, to load multiple OpenXML files simultaneously, you should include a definition of the PatternExec and PatternBurst.
- Duplicate block names may appear within the pattern file. If they do, they must be unique names in the file.

Accessing block names from IVI

Most named blocks defined in STIL are accessible from IVI.

In order to support loading multiple STIL files, the block names within a file may have an additional string concatenated with the name when the file is loaded. This is done by the customer specifying a STIL namespace parameter to the IVI API to load the file. Doing this has the following behaviors:

- The namespace string is attached to the front of each name, and a '!' separator is used. Note that any default blocks already have the '!' separator and it is not duplicated.
- References within that STIL file also have the namespace prepended.

Example

For the following source code:

```
Timing {
   WaveformTable mywft {
      Period ‘8ns’;
      Waveforms {
         Data { 01 { ‘0ns’ D/U; }
            X { ‘2ns’ ForceOff; }
         }
      }
   }
…
      Vector { Data = 0; }
      Vector { Data = 1; }
```

When loaded with the namespace 'Foo' appears as if it had been defined as follows:

```
Timing Foo!Default {
   WaveformTable Foo!mywft {
      Period ‘8ns’;
      Waveforms {
         Foo!Data { 01 { ‘0ns’ ForceDown/ForceUp; }
         X { ‘2ns’ ForceOff; }
         }
      }
```

```
   }
}
…
Vector { Foo!Data = 0; }
   Vector { Foo!Data = 1; }
```

STIL Examples

Complete programming examples (C++ using IVI-C and IVI-Com, C#, VisualBasic, etc.) are located in the following folders:

C:/Program Files/IVI Foundation/IVI/Drivers/KtMDsr/Examples

C:/Program Files (x86)/IVI Foundation/IVI/Drivers/KtMDsr/Examples

5 Using Bulk Data Files

The BulkData format provides a way to load only the data that makes up the pattern. The result of a loaded a bulk data file is a pattern STIL component that can be used with the additional definition information from STIL, OpenXML, or the KtMDsr API.

The BulkData format is a flexible format based on a text file. The content can be just the waveform characters that define the pattern, or waveform timing information and line numbers may be included.

A bulk data file may have its load operation deferred until the pattern is compiled and written to memory. This allows much larger patterns to be loaded. When LoadBulkDataDeferred is used, a STIL component for the pattern is created, but it does not contain any pattern elements because the file is not actually read until compile time. At compile time, the file is read, compiled, and written to the DSR memory a small block at a time, and again, no STIL component pattern elements are created. This is done so that the DSR can handle very large data files (up to 125 million vectors) without exceeding computer memory.

This chapter describes the syntax of the bulk data file.

Header Line

The header line establishes the format that will be used to read the bulk data file.

The header line can be the first non-comment line in the file, or it can be provided via the API. By providing the header line in the API, an arbitrary file containing only the target data may be loaded.

The header line has the following format:

```
[[“Index”] ["WaveformTable"](<SignalList>)+ ]
```

If the word *Index* is present, then an optional *Pattern Index* is parsed from the input. If it is not present, then the parser will automatically assign an index starting with 0. Indexes are numbered beginning with zero (0).

If the word *WaveformTable* is present, then an optional *WaveformTable* name is parsed from the input. If it is not present, the parser will use the waveform table specified in the *LoadBulkData* function call for each vector in the pattern.

The signal list is a space or comma (,) separated list of signals. This provides the list of signals associated with the columns of data.

Note that although signal names are separated by white space and comma in the header line, no separator is required between signals in the pattern area.

The Bulk Data file Syntax

The file is read using the following rules:

- Header format [<optional index><optional WFT><signal>...]
- Comments are designated with a hash (#) or double slash (//). Comments extend to the end of line. Blank lines are also treated as comments.
- The first non-comment line of the file may be a header that indicates the syntax of the file. If the header begins with the word “Index”, a numeric Index is expected on each data line. If the header has the word “WaveformTable” then a WaveformTable is parsed on each line. Finally, the header has a list of signals, indicating the order and signal name to associate with each position in the data lines.
- If no header line is present, the file is assumed to only contain bulk data with an unchanging waveform table and signals as provided via the API.
- Data lines are parsed based on the header line. Individual waveform characters may be separated by comma or space or may be grouped together.

- A hyphen (-) can be from the previous vectors, and used in place of a waveform character. This indicates that the waveform character on that signal should be repeated. In STIL this is indicated by leaving the waveform character unspecified, but it must be explicit in bulk data.

Notes and examples

Generally, values provided by the API override any values appear in the file. For example, if the signal list is provided in the API, any signals appears on the header line are ignored.

The following are typical examples:

```
# this file includes a header showing indexes
# and waveform table

[Index WaveformTable Signal1 Signal2 Signal3 Clock]
0 One 011T
1 011T #The waveform table defaults to previous line
2 100T
3 1 0 1 T #separators are ignored between characters
```

The following example shows a more terse description. When using this syntax, the waveform table must be specified in the function invocation and the indexes are automatically generated counting from 0.

```
# this datafile only defines file includes a header
# showing indexes and waveform table
[Signal1 Signal2 Signal3 Clock]
0       1  1    T
0       1  1    T
0       0  0    T
0       0  1    T
```

This next example is the most dense, but has exactly the same meaning as the previous one, except the signal list must be provided via the API (and the comment has been left out).

```
011T
011-
100-
101-
```

Here is an example bulk data file that does not include a header. Because this file does not include a header, it must be provided via the API.

```
# BulkData test file
U D U L H L L
U D U L H L L
U D U L H L L
U D U L H L L
U D U L H L L
U D U L H L L
U D U L H L L
U D U L H L L
U D U L H L L
U D U L H L L
U D U L - L L
U - D - U - L
- - - - - - -
L L L L L L L
```

This last example shows a bulk data file with a WaveformTable and no Index:

```
# Test file for BulkData
[WaveformTable Sig1 Sig2 Sig3 Sig4 Sig5 Sig6 Sig7]
  WFT1  U  D   U   L H L L
  WFT1  U  D   U   L H L L
      U  D   U   L H L L
  WFT2  -  -   -   - - - -
  WFT1  U  D   U   L H L L
      U  D   U   L H L L
  WFT1  U  D   U   L H L L
  WFT1  U  D   U   L H L L
  WFT1  U  D   U   L - L L
  WFT1  U  D   U   L - - L
```

Loading a Bulk Data File

There are two API methods of loading a Bulk Data file: **LoadBulkData** and **LoadBulkDataDeferred**. Both methods must follow the rules described previously.

LoadBulkData

This method loads the bulk data file into a pattern of a specified name using the bulk data format. Header in file can be over-riden here.

6 Using OpenXML (Spreadsheet) Data Files

OpenXML files are spreadsheet files where the content is based on STIL. OpenXML files do not provide any features or capabilities that are not available from STIL, but they do provide the following benefits:

- They are a convenient tool for editing several blocks at the same time. The spreadsheet “sheet” model provides a convenient way to organize the data needed to fully describe a pattern.
- For many applications the spreadsheet is useful for calculating and manipulating values. For example, edge times or voltages can be calculated with formulas.
- For some applications, the actual bulk data that makes up the pattern can be easily acquired in a spreadsheet.
- You do not have to learn and deal with the syntax of the STIL language. However, you do need to deal with the syntax required by the M9195A/B OpenXML input.

NOTE Comma-separated lists are not supported in OpenXML files.

OpenXML Files

M9195A/B OpenXML files are standard spreadsheet files (in a workbook) that contain all of the necessary information (signals, waveform tables, DCLevels, Sites, etc.) for the M9195A/B module. Each “sheet” in the workbook corresponds to a specific type of information.

The M9195A/B OpenXML format is based on the STIL grammar. STIL is heavily based on “blocks”. With only minor exceptions, the sheets in the OpenXML workbook correspond to the STIL blocks.

Each sheet name (the string on the tab across the bottom in Excel). Is both the default type of block, and the name for the block generated. For example, the default sheet has a tab called “Pattern” and a pattern block of name pattern is generated from that sheet.

The M9195A/B OpenXML parser allows the OpenXML to override the default name and type by putting an entry in cell **A1** (the upper left corner). An entry of the form:

```
<type>:<name>
```

Is used to indicate that this sheet if of the specified type and name. For example if the following were in cell A1:

```
Pattern:SPI_Test
```

Then that sheet is parsed as a Pattern sheet with the name “SPI_Test”. Using this syntax numerous patterns (or other blocks) can be defined.

A rectangular array of cells is taken from each sheet to define the contents of that sheet. It can be preceded by comment lines, and any arbitrary data can follow it. The rectangular array always beings with a title row that defines the columns in the table. These column headers are defined for each sheet type.

General Syntax Rules

The following rules generally describe how OpenXML input is parsed.

- Formatting (colors, fonts, sizes, cell sizes) are ignored
- Generally strings are case sensitive
- If a value is generated with a spreadsheet formula, that value is used as it appears in the spreadsheet tool. If a value is entered as a formula in string, then that string will transferred to the instrument driver and evaluated by the driver. This can be useful when using formula's that are recalculated in the driver, or when using IEEE 1450 symbols such as “@”.
- An entire line can be commented out by making the first non-blank cell on a row begin with “#” or “//”.

The DSR module uses the following rules to convert an OpenXML file into STIL Components:

- The KtMDsr OpenXML syntax uses the sheets in the file to define STIL blocks (STIL Components). The type of sheet is taken from the sheet name or cell A1 (see below). The following are the kinds of sheets:
  - **Signals**. The Signals sheet defines the STIL signals and the STIL SignalGroups. The name of this sheet is always Signals since Signals are global and can never have a block name.
  - **DCLevels**. The DCLevels sheet defines a STIL DCLevels block.
  - **WaveformTable**. The WaveformTable sheet defines a STIL WaveformTable block.
  - **Pattern**. The Pattern sheet defines a STIL Pattern block.
  - **Site**. The Site sheet defines a STIL Site block.
- The OpenXML reader will look at cell A1 and at the name of the sheet to determine the type and name of the sheet.

  - To specify the type and name in cell A1, the contents of the cell must be a string containing a colon. The substring to the left of the colon is the type of sheet. The substring to the right of the colon indicates the sheet name. For example, **`Pattern:Beta`**
  - To specify the type using the sheet name, the sheet name must start out with one of the defined type names. The name of the sheet may be optionally specified by concatenating that on the end of the name with an optional hyphen (-) separator. For example, **`Pattern-Beta`** would indicate this is a pattern sheet and it is called Beta.

- The pattern reader will ignore comments. These are lines that:
  - begin with the hash character (#) character
  - begin with a double slash (//)
  - are blank lines
- The pattern reader looks for a rectangular table. The width of the table is defined by the first non-comment line. This first non-comment line contains headers that define the columns.
  - The table extends till the end of the sheet or a blank line. Comments may be added to the right of the table once the header has established the width. To add a comment to the right of the header, start it with a comment character (# or //).
- In cells where a numeral is required, either a spreadsheet number or a string may be entered. If a string is entered, the "Numeric Expressions" evaluator is used to convert the string to a number. This permits using engineering units and prefixes. For example, a time can be entered as the string "10ns". Additionally, the numeric variables implemented by the KtMDsr driver may be used. This permits deferring the evaluating of the strings until the pattern is activated at which time the expressions may be evaluated with new variables using the ReevaluateExpressions function.
- In general, formatting is ignored. It is generally useful to use cell shading to improve the readability of a pattern; however, it is ignored by the KtMDsr.

Sheet Types

The following table lists the types of sheets used in an OpenXML spreadsheet:

Sheet type	Summary of usage and notes on usage.
Signals	Specify signals and groups. Same usage as the STIL Signal and SignalGroup blocks put together. Only one Signal sheet is permitted.
WaveformTables	Define WaveformTables, multiple allowed
DCLevels	Specify the DCLevels for each signal, multiple allowed
PatternExec	Define the PatternBurst and the PatternExec. Multiple are allowed. PatternBursts defined in one PatternExec may be used in other PatternExecs.
Pattern	Pattern definition, multiple allowed
Settings	Optional Sheet that contains general settings for the pattern
MacroDef	Optional Sheet. A MacroDef sheet is very similar to a Pattern sheet, the difference is that the MacroDef sheet is made up of macro definitions. Thus, the MacroDef sheet permits the pattern data to include MacroDef/EndMacro statements whereas the Pattern sheet does not.

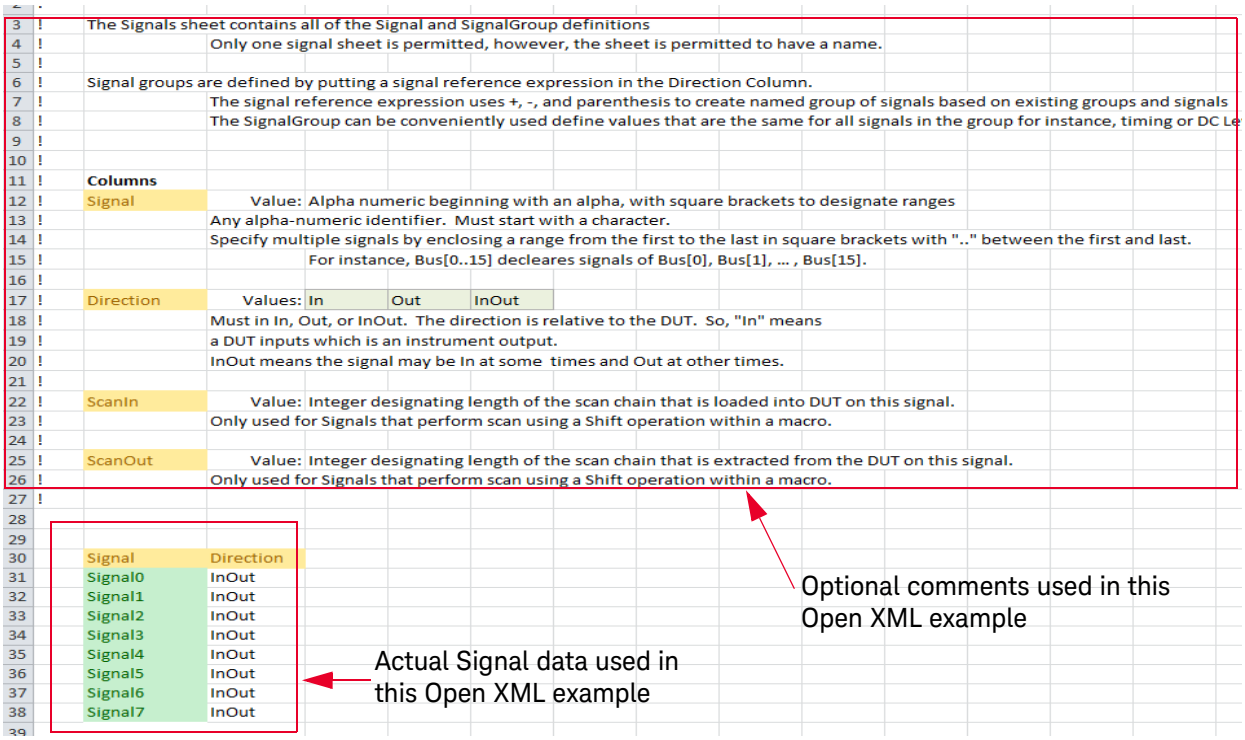
The following pages describe in detail the individual spreadsheet “sheets” along with very simple examples of each sheet. Note that each example sheet shows several comments along with the actual data.

Signals Sheet

This sheet defines the Signals and the SignalGroups used in the pattern. Two columns are required: Signal and Direction.

Column	Definition
Signal	Name of the signal or signal group. Follows STIL syntax. This column may include a range or signal reference expression. Names that define a range are permitted for example: Data[0..15] This defines 16 signals of name: Data[0], Data[1],],..., Data[15]. When the Signal column contains a (valid) signal reference expression, then the signal name is taken to be a signal group and is defined with that expression. For example, ("Signal1+Signal2").
Direction	Defines the signal direction: either In, Out, or InOut. These are defined DUT-centric. For example, "Out" means an output from the DUT, which as an input to the DSR.

In addition, this same sheet can be used to define a signal group. That is, a single name that can be used to refer to a group of signals.

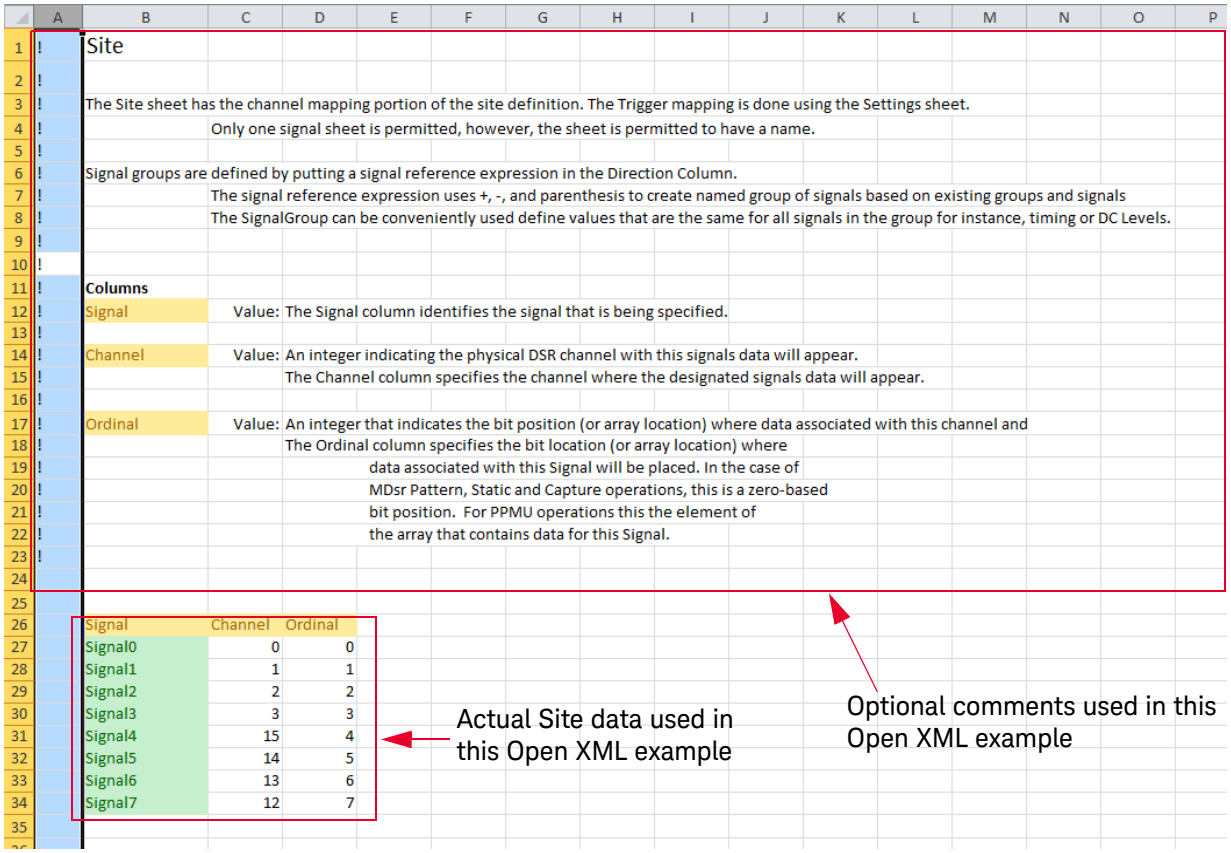


Simple example of a Signals sheet in an OpenXML Spreadsheet

Site Sheet

The Site sheet defines the Channel that a Signal should be mapped to on the M9195A/B. The ordinal is used when reading values from the site. The ordinal is used to set the element of the array that contains the value for this Channel, or the bit position where this Channel's result is.

Column	Definition
Signal	The signal name as defined. A signal group or range does not make sense because each needs its own Channel setting
Channel	The channel on the M9195A/B (0-15, and 20-23)
Ordinal	The Ordinal column specifies the bit location (or array location) where data associated with this Signal will be placed. In the case of M9195A/B Pattern, Static and Capture operations, this is a zero-based bit position. For PPMU operations this the element of the array that contains data for this Signal. .



Simple example of a Site sheet in an OpenXML Spreadsheet

WaveformTable Sheet

The WaveformTable sheet defines a single waveform table. Several may be defined. A single waveform table defines how the DSR should produce all the waveform characters that appear in the pattern definition. Waveform is unique in that it sets a value based on the upper left corner. The upper left corner defines the Period for that table.

Column	Definition
Period=<period>	The entries in this column are the signal names. This can be "Signal Reference Expressions" or simple signal names.
Character	The waveform character for which the transitions are being defined. As with STIL, multiple characters can be defined on the same line. To do so, each character is separated with a slash (/). Then the actions corresponding to each character is separated with a slash.

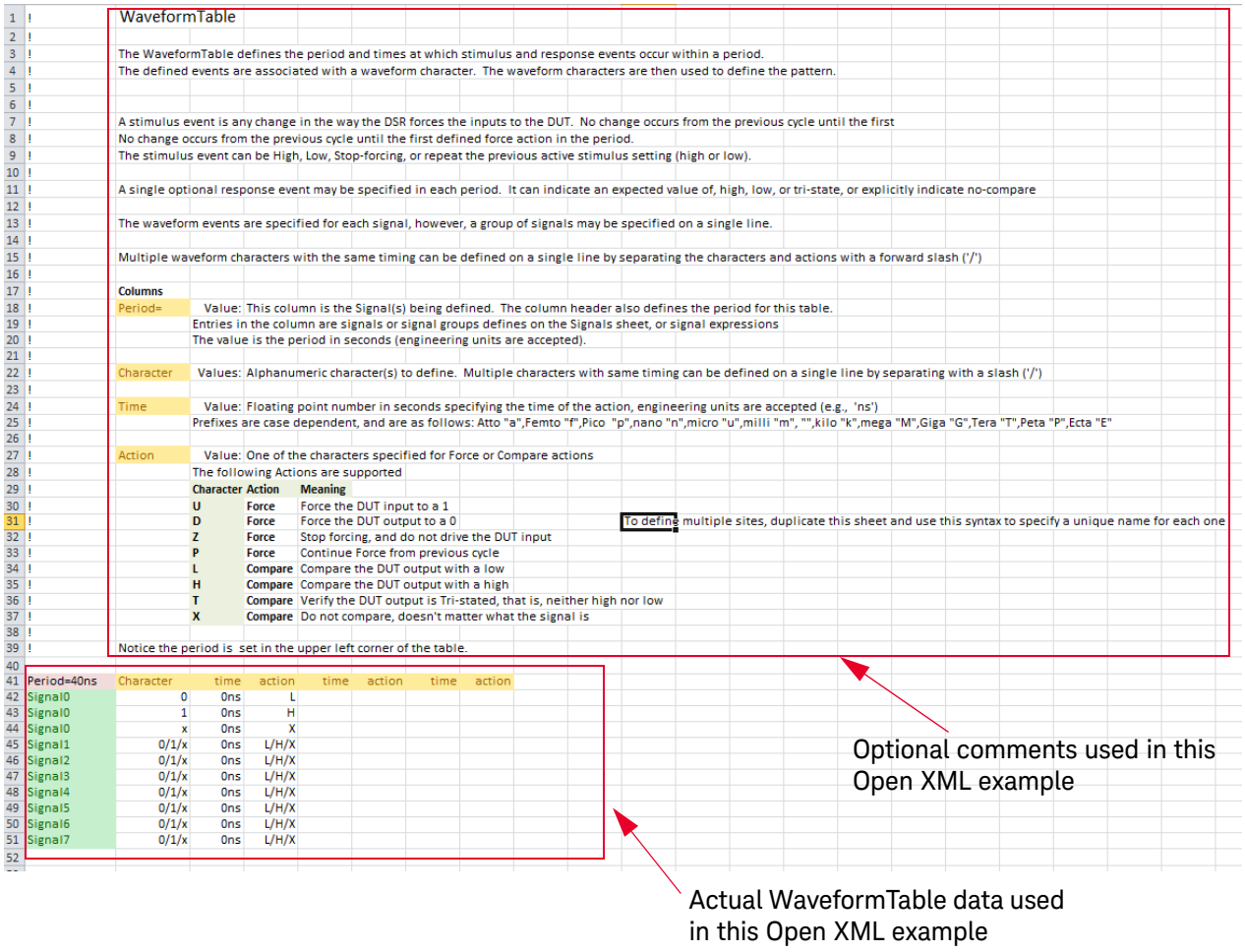
Column	Definition
Time (up to 3 columns)	The *n*th time column is the time at which to apply the action specified in the *n*th action column.
Action (up to 3 columns)	The action columns specify an action character. Characters are defined below.

The following table defines the Action characters that are supported

Character	Action	Definition
U	Force	Force the output to a 1
D	Force	Force the output to a 0
Z	Force	Stop forcing, and do not drive the output
P	Force	Continue Force from previous cycle
L	Compare	Compare the input with a low
H	Compare	Compare the input with a high
T	Compare	Verify the input is Three-stated, that is, neither high nor low
X	Compare	Do not compare, doesn't matter what the signal is, but log data

Note that the STIL pattern of defining multiple waveform characters on a single line is supported. For example, waveform characters A,B,C, and D are all defined by this line:

Character	time	action	time	action	time	action
A/B/C/D	10ns	D/U/D/U	150ns	Z	190ns	X/X/L/H



Simple example of a Waveform Table sheet in an OpenXML Spreadsheet

DCLevels Sheet

The DCLevels sheet is used to define a STIL DCLevels block. There can be multiple occurrences of DCLevels sheets, but each must have a unique name.

The DCLevels sheet has the following columns:

Column	Definition
Signal	"Signal Reference Expressions" that describes the signals that this DC Level defines
VOH	The output high voltage from the DUT. This is the minimum voltage that will be treated as High as measured at the DUT output
VOL	The output low voltage from the DUT. This is the maximum voltage that will be treated as Low as measured at the DUT output
VHYST	This is the hysteresis used when measuring DUT outputs. The input value is rounded to either 0, 75mV, or 150mV
VIH	This is the input to the DUT. This is the voltage that will be presented at the DUT input for a High.
VIL	This is the input to the DUT. This is the voltage that will be presented at the DUT input for a Low.
IOL	This is the low output current for use with the active load. To turn on the active load, set values other than NaN for Ioh, Iol, and Vcom that is injected into the DUT when the DUT is driving When active load is on, and the DUT voltage output exceeds Vcom, Ioh is output. If the DUT voltage is less than Vcom, Iol is input.
IOH	This is the high output current for use with the active load. To turn on the active load, set values other than NaN for Ioh, Iol, and Vcom that is injected into the DUT when the DUT is driving When active load is on, and the DUT voltage output exceeds Vcom, Ioh is output. If the DUT voltage is less than Vcom, Iol is input.
VCOM	This is the commutation voltage for use with the active load. To turn on the active load, set values other than NaN for Ioh, Iol, and Vcom that is injected into the DUT when the DUT is driving When active load is on, and the DUT voltage output exceeds Vcom, Ioh is output. If the DUT voltage is less than Vcom, Iol is input.
VIT	This is the termination voltage used with the 50 Ω termination. To turn on this mode, set the Vit value and leave the active load settings of Ioh, Iol, and Vcom blank (or put in 'NaN' for not a number).
VHH	VHH is the high voltage output used in the high voltage output mode on the high voltage channels (20–23).

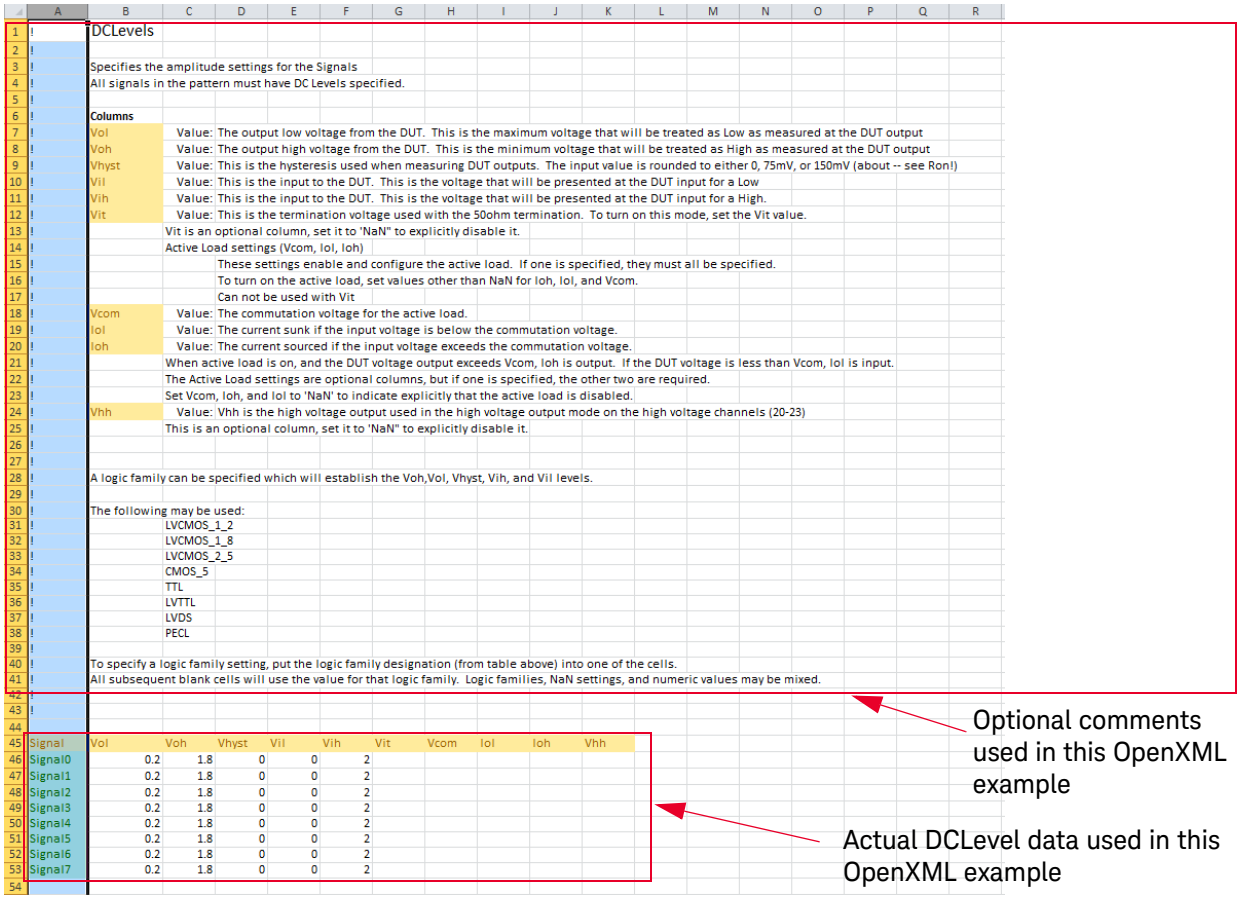
Any cell within the table can also specify a logical level from a standard logic family. Once a standard logic family is encountered, it is used for the rest of the blank cells in the table.

The following standard logic families are supported:

- LVCMOS_1_2
- LVCMOS_1_8
- LVCMOS_2_5
- CMOS_5
- TTL
- LVTTL
- LVDS
- PECL

All signals must have DCLevels specified for them.

NOTE NaN is defined as “Not a Number.” It is a numeric data type value representing an undefined or unrepresentable value, especially in floating-point calculations.



Signal	Vol	Voh	Vhyst	Vil	Vih	Vit	Vcom	Iol	Ioh	Vhh
Signal0	0.2	1.8	0	0	2					
Signal1	0.2	1.8	0	0	2					
Signal2	0.2	1.8	0	0	2					
Signal3	0.2	1.8	0	0	2					
Signal4	0.2	1.8	0	0	2					
Signal5	0.2	1.8	0	0	2					
Signal6	0.2	1.8	0	0	2					
Signal7	0.2	1.8	0	0	2					

FSimple example of a DCLevels sheet in an OpenXML Spreadsheet

PatternExec Sheet

If the optional PatternExec sheet is not defined, a PatternExec and PatternBurst is automatically created named PatternExec_<*filename*> and PatternBurst_<*filename*> respectively. The PatternBurst contains the single default pattern (Pattern!Default), and the PatternExec references that PatternBurst and the default DC Levels (DCLevels!Default).

The PatternExec sheet permits explicitly defining a PatternExec instead of using the abbreviated syntax in the Settings sheet. The PatternExec provides the PatternBurst and the DCLevels that will be used to generate a named pattern.

The syntax for specifying the PatternBurst is a plus (+) separated list of pattern sheets. The syntax for specifying the DCLevels is just the name of the DCLevels sheet.

The columns in the PatternExec sheet are:

Column	Description
SheetType	This syntax indicates if this line is defining a PatternBurst or a DCLevels. Thus, this column needs to contain either the string "PatternBurst" or the string "DCLevels".
Name	This column identifies the names of the sheets to be used. For PatternBurst, it is a plus (+) separated list of sheet names. For DCLevels it is a single sheet name.

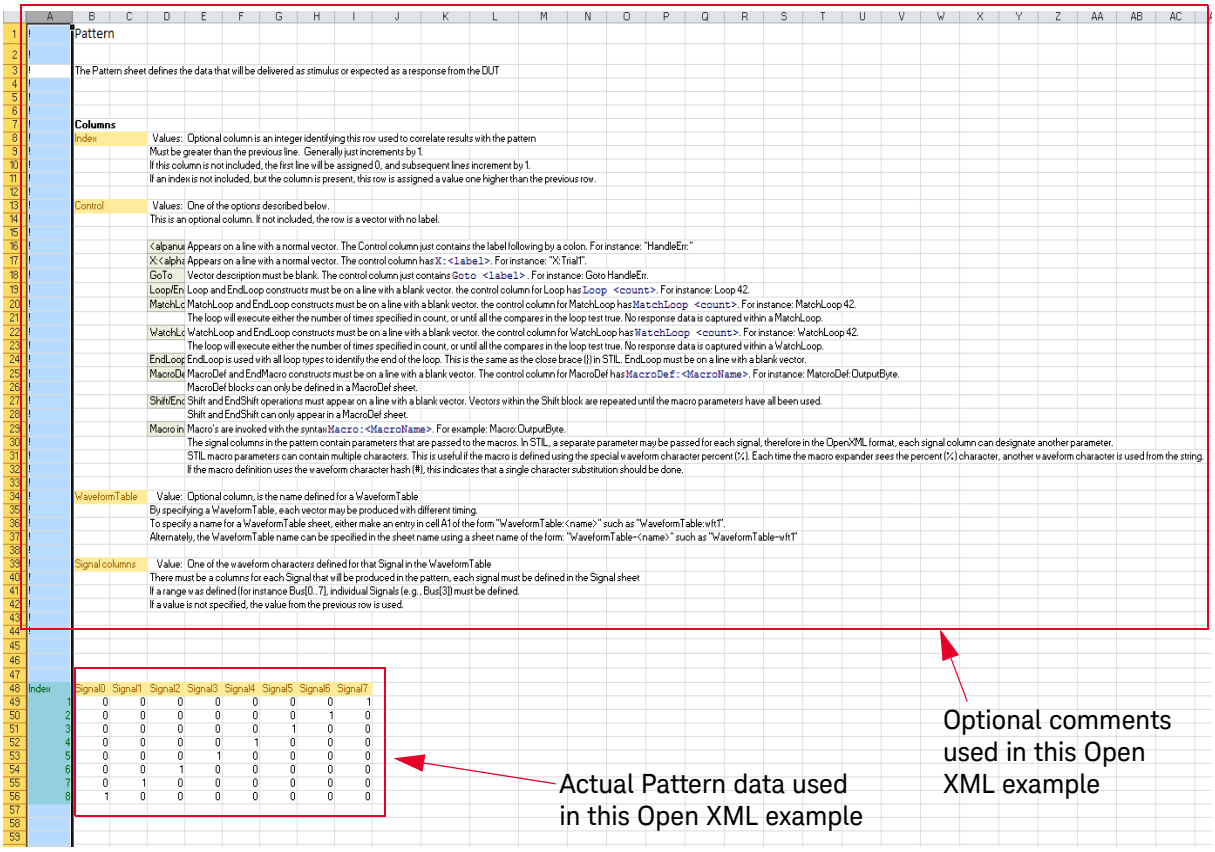
Pattern Sheet

The Pattern sheet has the pattern definition. Multiple pattern sheets can be defined.

The pattern table has the following columns:

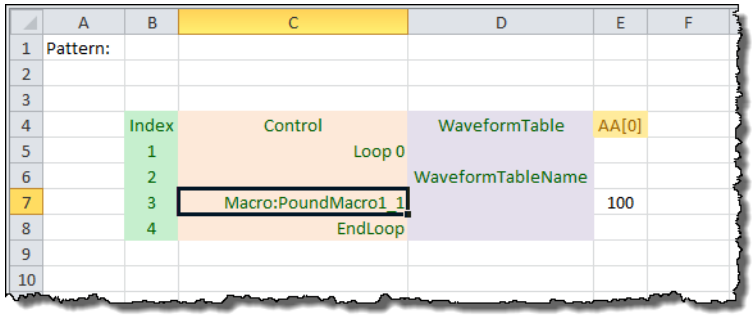
Column Name	Definition
Index	This is an optional column that permits entering a pattern index. The pattern index is returned with the comparison response data, so including it in the pattern makes it easier to identify the pattern line that was responsible for a given response. If the pattern index is not filled in, consecutive numbers starting from 0 are used. When assigning an index, it is beneficial to increment the previous line by one. This creates a more efficient compiled pattern than incrementing the index by other amounts.
WaveformTable	This is an optional column that permits entering a WaveformTable for this row. Using this, a different WaveformTable, defining different characters, timings or period can be used for each vector. If the column is specified, but the entry is blank, this vector will be produced with the same WaveformTable as the previous vector. If no column is specified, and if there is a single WaveformTable defined, that will be used.
Signal columns	Additional columns can be put in the pattern sheet, where each column defines another signal. The column header is merely the name of the signal as defined on the Signal sheet. Note that if the signal sheet defines a group of Signal using a syntax like: "Chan[0..10]", a separate column must be defined in the pattern for each Chan signal using the syntax: Chan[0], Chan[1], etc. (for all 11 signals in the range).

Column Name	Definition	
Control	This is an optional column for entering control information. This is an optional column. If not included, the row is a vector with no label:	
	Label:	Appears on a line with a normal vector. The Control column just contains the label following by a colon. For example: "HandleErr:"
	X:Label	Appears on a line with a normal vector. The control column has X:<label>. For example: "X:Trial1".
	GoTo	Vector description must be blank. The control column contains Goto <label>. For example: "Goto HandleErr".
	Loop / EndLoop	Loop and EndLoop constructs must be on a line with a blank vector. The control column for Loop has Loop <count>. For example: Loop 42.
	MatchLoop / EndLoop	MatchLoop and EndLoop constructs must be on a line with a blank vector. The control column for MatchLoop has `MatchLoop <count>`. For example: `MatchLoop 42`. The loop will execute either the number of times specified in <count>, or until all the compares in the loop test true. No response data is captured within a MatchLoop.
	WatchLoop / EndLoop	WatchLoop and EndLoop constructs must be on a line with a blank vector. WatchLoop Indicates a block should be repeated until an external trigger event is received. WatchLoop syntax is: WatchLoop <LoopCount><Trigger>
	EndLoop	EndLoop is used with all loop types to identify the end of the loop. This is the same as the close brace (}) in STIL. EndLoop must be on a line with a blank vector.
	MacroDef / EndMacro	MacroDef and EndMacro constructs must be on a line with a blank vector. The control column for MacroDef has `MacroDef:<MacroName>`. For example: MatcroDef:OutputByte. MacroDef blocks can only be defined in a MacroDef sheet.
	Shift / Endshift	Shift and EndShift operations must appear on a line with a blank vector. Vectors within the Shift block are repeated until the macro parameters have all been used. Shift and EndShift can only appear in a MacroDef sheet.
	Macro Invocation	Macro's are invoked with the syntax `Macro:<MacroName>`. For example: **Macro:OutputByte**. The signal columns in the pattern contain parameters that are passed to the macros. In STIL, a separate parameter may be passed for each signal, therefore in the OpenXML format, each signal column can designate another parameter. STIL macro parameters can contain multiple characters. This is useful if the macro is defined using the special waveform character percent (%). Each time the macro expander sees the percent (%) character, another waveform character is used from the string. If the macro definition uses the waveform character hash (#), this indicates that a single character substitution should be done.



Simple example of a Pattern sheet in an OpenXML Spreadsheet

Another example of a Pattern Sheet showing MacroDefs:



	A	B	C	D	E	F
1	Pattern:					
2						
3						
4		Index	Control	WaveformTable	AA[0]	
5		1	Loop 0			
6		2		WaveformTableName		
7		3	Macro:PoundMacro1_1		100	
8		4	EndLoop			
9						
10						

Simple example of a Pattern sheet with a MacroDefs call

MacroDefs Sheet

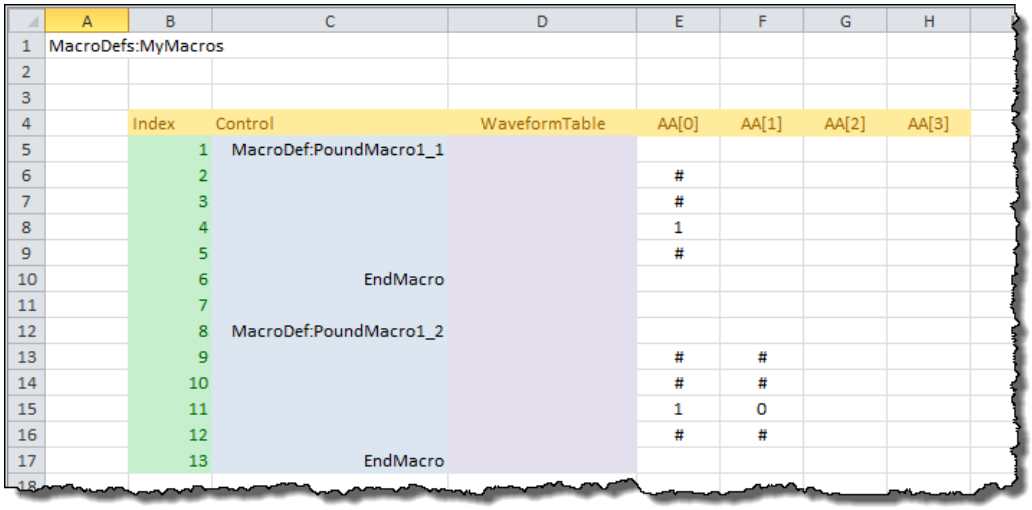
Similar to Pattern sheet, the optional MacroDefs sheet encloses all of the pattern code in a Macro/EndMacro.

MacroDef:<name> -- Defines a macro with the given name

EndMacro -- Defines a match loop. The count is the number of passes to dwell in the loop, Within a macro, a Shift may be provided.

Shift -- On a line by itself with no vector defined. Shift is a parameter substitution operation. It indicates that the parameters from the input parameter list should be executed until the list is exhausted. Shift may only be used within a Pattern that is designated as a macro using MacroDef.

EndShift -- On a line by itself with no vector defined



	A	B	C	D	E	F	G	H
1	MacroDefs:MyMacros							
2								
3								
4		Index	Control	WaveformTable	AA[0]	AA[1]	AA[2]	AA[3]
5		1	MacroDef:PoundMacro1_1					
6		2			#			
7		3			#			
8		4			1			
9		5			#			
10		6	EndMacro					
11		7						
12		8	MacroDef:PoundMacro1_2					
13		9			#	#		
14		10			#	#		
15		11			1	0		
16		12			#	#		
17		13	EndMacro					

Simple example of a MacroDefs sheet in an OpenXML Spreadsheet

NOTE MacroDef statements and the WaveformTable syntax or a Vector must be on the same line. If they are not on the same line, a syntax error will occur.

NOTE In order to use Shift and EndShift, the Signal Sheet must have ScanIn and ScanOut defined. ScanIn and ScanOut requires a length value, which is the number of characters to be shifted. Note that while STIL does not require a length, it is required in the OpenXML implementation of ScanIn and ScanOut.

Figure shows another MacroDefs sheet example showing the use of Shift/EndShift:

	B	C	D	E	F	G	H	I	J	K	L	M	N
4													
5	Control	Out[1]	Out[2]	Out[3]	Out[4]	Out[5]	Out[6]	Out[7]	Out[8]	In[1]	In[2]	In[3]	In[4]
6	MacroDef:test												
7		X	X	X	X	X	X	X	X	1	0	0	
8		0	1	0	0	0	0	0	0	0	1	0	
9		0	0	1	0	0	0	0	0	0	0	1	
10		0	0	0	1	0	0	0	0	0	0	0	
11	Label1:	0	0	0	0	1	0	0	0	0	0	0	
12		0	0	0	0	0	1	0	0	0	0	0	
13	GoTo Label1												
14	EndMacro												
15	MacroDef: foo												
16	Loop [10]												
17		0	H	H	L	L	L	L	L	0	0	1	
18		0	H	L	L	L	L	L	L	0	0	0	
19		0	H	L	L	L	L	L	L	0	0	0	
20	EndLoop												
21	End Macro												
22	MacroDef:test2	0	H	L	L	L	L	L	L	0	0	0	
23		0	H	L	L	L	L	L	L	0	0	0	
24		0	H	L	L	L	L	L	L	0	0	0	
25		0	T	T	T	T	T	T	T	1	0	0	
26		0	T	T	T	T	T	T	T	0	0	1	
27		0	T	T	T	T	T	T	T	0	0	0	
28													
29	X:foo	0	H	L	L	L	L	L	L	0	0	0	
30	X:123	0	T	T	T	T	T	T	T	0	0	0	
31	X:1f00	0	T	T	T	T	T	T	T	0	0	0	
32	X:foo2	0	T	T	T	T	T	T	T	0	0	0	
33	EndMacro												
34													
35	MacroDef:test3												
36		0	%	T	%	T	%	T	T	0	0	0	
37	Shift												
38		0	%	T	%	T	%	T	T	0	0	0	
39	End Shift												
40		0	%	T	%	T	%	T	T	0	0	0	
41		0	%	T	%	T	%	T	T	0	0	0	
42		0	%	T	%	T	%	T	T	0	0	0	
43		0	T	T	T	T	T	T	T	#	#	#	
44	EndMacro												
45													
46													
47													
48													

Site | WaveformTable | DCLevels | PatternExec | Pattern | Pattern (2) | MacroDefs | Settings

MacroDefs sheet example showing Shift/EndShift

Settings Sheet

The settings sheet is optional and contains general settings for the pattern.

The columns in the settings sheet are:

Column	Definition
Sheet	This is the name of the sheet to set. At this time, the only syntaxes supported are: • Sheet is set to "PatternExec" and the setting to "PatternBurst". The value is set to a <Name>=<plus (+) separated list of patterns to execute>. A PatternExec of the specified name is created with a pattern burst composed of the listed patterns. • Sheet is set to the name of a Site sheet. The setting is set to `Trigger <0|1|2|3|4|5> and Marker <0|1|2>`. The value is set to a trigger source. For example "`SoftwareTrigger0`". This syntax binds the trigger source to the Trigger specified in the pattern, within the WatchLoop instructions.
Setting	This is the parameter to set. The allowed values are dependent on the sheet type.
Value	This is the value to set the parameter to. The meaning is dependent on the setting and sheet type.

Using an OpenXML file with the M9195A/B

IVI Functions

The following IVI functions may be used to load an OpenXML file to the module:

Method	Description
Load	This function loads a STIL or OpenXml file and creates objects based on the content of the file. The objects created are dependent on the actual file. When an element is loaded and one by that name already exists in the same namespace, the duplicate element will replace the original one.
LoadAndUpdateOpenXml	This function loads an OpenXml file and creates STIL components based on the content of the file. The STIL Components created are dependent on the file contents. Syntax errors are written to a new sheet in the original OpenXml file.

Syntax errors can either be:

- Read out one at a time from the API
- Displayed by the SFP
- Incorporated into the Spreadsheet file as an additional sheet.

M9195-90005
www.keysight.com