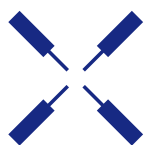


SHFQC+ User Manual

8.5 GHz Quantum Controller



Zurich
Instruments

SHFQC+ User Manual

Zurich Instruments AG

Revision 25.04

Copyright © 2008-2025 Zurich Instruments AG

The contents of this document are provided by Zurich Instruments AG (ZI), "as is". ZI makes no representations or warranties with respect to the accuracy or completeness of the contents of this publication and reserves the right to make changes to specifications and product descriptions at any time without notice.

LabVIEW is a registered trademark of National Instruments Inc. MATLAB is a registered trademark of The MathWorks, Inc. All other trademarks are the property of their respective owners.

Table of Contents

Declaration of Conformity	
1. Change Log.....	2
1. 1. Release 25.04.....	2
1. 2. Release 25.01.....	2
1. 3. Release 24.10.....	2
1. 4. Release 24.07.....	2
1. 5. Release 24.04.....	3
1. 6. Release 24.01.....	3
1. 7. Release 23.10.....	3
1. 8. Release 23.06.....	4
1. 9. Release 23.02.....	4
1. 10. Release 22.08.....	5
1. 11. Release 24.07 Additional Information.....	5
2. Getting Started	10
2. 1. Quick Start Guide.....	10
2. 2. Inspect the Package Contents.....	11
2. 3. Handling and Safety Instructions.....	12
2. 4. Software Installation.....	14
2. 5. Connecting to the Instrument.....	22
2. 6. Software Update.....	38
2. 7. Troubleshooting.....	39
3. Functional Overview	43
3. 1. Features.....	43
3. 2. Front Panel Tour.....	46
3. 3. Back Panel Tour.....	47
3. 4. Ordering Guide.....	48
4. Tutorials.....	51
4. 1. Signal Generator Tutorials.....	51
4. 2. Quantum Analyzer Tutorials.....	94
5. Functional Description.....	150
5. 1. Setup Functionality.....	150
5. 2. Measurement Functionality.....	170
6. Specifications.....	253
6. 1. General Specifications.....	253
6. 2. Analog Interface Specifications.....	254
6. 3. Digital Waveform Generation of Signal Generator Channel.....	258
6. 4. Digital Signal Processing of Quantum Analyzer Channel.....	258
6. 5. Digital Interface Specifications.....	259

Table of Contents

- 7. Device Node Tree261
 - 7.1. Introduction261
 - 7.2. Reference Node Documentation264

CE Declaration of Conformity



The manufacturer

Zurich Instruments
Technoparkstrasse 1
8005 Zurich
Switzerland

declares that the product

SHFQC+, Super High Frequency Qubit Controller

is in conformity with the provisions of the relevant Directives and Regulations of the Council of the European Union:

Directive / Regulation	Conformity proven by compliance with the standards
2014/30/EU (Electromagnetic compatibility [EMC])	EN 61326-1:2013, EN 55011:2016, EN 55011:2016/A1:2017, EN 55011:2016/A11:2020 (Group 1, Class A and B equipment)
2014/35/EU (Low voltage equipment [LVD])	EN 61010-1:2010, EN 61010-1:2010/A1:2019, EN 61010-1:2010/A1:2019/AC:2019-04
2011/65/EU, as amended by 2015/863 and 2017/2102 (Restriction of the use of certain hazardous substances [RoHS])	EN IEC 63000:2018
(EC) 1907/2006 (Registration, Evaluation, Authorisation, and Restrictions of Chemicals [REACH])	-

Zurich, October 20th, 2022

Flavio Heer, CTO

UKCA Declaration of Conformity



The manufacturer

Zurich Instruments
Technoparkstrasse 1
8005 Zurich
Switzerland

declares that the product

SHFQC+, Super High Frequency Qubit Controller

is in conformity with the provisions of the relevant UK Statutory Instruments:

Statutory Instruments	Conformity proven by compliance with the standards
S.I. 2016/1091 (Electromagnetic Compatibility Regulations)	EN 61326-1:2013, EN 55011:2016, EN 55011:2016/A1:2017, EN 55011:2016/A11:2020 (Group 1, Class A and B equipment)
S.I. 2016/1101 (Electrical Equipment (Safety) Regulations)	EN 61010-1:2010, EN 61010-1:2010/A1:2019, EN 61010-1:2010/A1:2019/AC:2019-04
S.I. 2012/3032 (Restriction of the Use of Certain Hazardous Substances Regulations)	EN IEC 63000:2018

Zurich, October 20th, 2022

A handwritten signature in black ink that reads 'Flavio Heer'.

Flavio Heer, CTO

1. Change Log

Info

A complete summary of all changes can be found in the [LabOne Release Notes](#). This page only lists changes not present in the LabOne Release Notes.

1.1. Release 25.04

Release date: 30-April-2025

See [Release Notes 25.04](#) for a detailed list of all changes.

1.2. Release 25.01

Release date: 31-January-2025

See [Release Notes 25.01](#) for a detailed list of all changes.

1.3. Release 24.10

Release date: 31-Oct-2024

1.3.1. General

- ─ Input/Output: Changed the granularity of center frequency from 100 MHz to **200 MHz**. An error will be emitted whenever a frequency that is not a multiple of 200 MHz is requested.

1.3.2. QA Channel

- ─ Generator: Added error reporting for too fast waveform playback trigger, i.e. holdoff error.

1.3.3. SG Channels

- ─ Output: Added a read-only node `/DEV.../SGCHANNELS/n/INTERNAL` that indicates if SG-Channel `n` is internal-use only. For example, the value of the node on SG Channel 1 of an SHFQC2+ is 0, whereas the value on SG Channel 5 of an SHFQC2+ is 1. Without the RTR option, all channels are physical and this always returns 0.

1.4. Release 24.07

Release date: 31-Jul-2024

1.4.1. General

- ─ Resolved a bug that triggered an unnecessary reconfiguration of the synthesizer frequency upon re-applying the already configured frequency value.
- ─ Improved configuration of the synthesizer to yield more stable lock.
- ─ Prevents LabOne version downgrade for the case in which the synthesizer is not supported in that version.

1.4.2. QA Channel

- QA Setup:
 - Corrected the reported length in the warning that is issued when the waveform length gets rounded to the sample granularity
- QA Result Logger:
 - Changed representation of un-acquired data in Result Logger to NaN.

1.4.3. SG Channels

- AWG:
 - Introduced sequencer command `configureFeedbackProcessing()` for dynamic feedback configuration. Documentation on how to change code can be found in [Flexible Feedback Processing](#).
 - Sequencer now reports an error if an AWG program executes a command table entry that has not been defined.
 - Sequencer now reports an error if a command uses a trigger counter argument and the counter value has already been surpassed.

1.5. Release 24.04

Release date: 30-Apr-2024

- Official support of the SHFQC+ instrument, including updated documentation
- SHFQC+: Added Output Muting functionality to SG and QA channels, available through LabOne Q
- SG and QA Channels: Outputs of RF and LF paths are now aligned by default.
- QA Channel: Added RF/LF Interlock to the In/Out Tab
- QA Channel: Changed the default integration delay to 212ns in both Spectroscopy and Readout mode.
- QA Channel: Changed the minimum integration delay to 8ns in Readout mode.
- Improved the optimization pass of the AWG compiler that removes unused registers.
- Support for sequencer command `setRate` has been removed.
- Support for sequencer command `waitTrigger` has been removed, use `waitDigTrigger` instead.

1.6. Release 24.01

Release date: 31-Jan-2024

- SG Channels: Extended the GUI for the Output Router and Adder to show how and where different settings are applied in the signal chain for each SG channel.
- QA Channel: Added hold-off error count clear and job index nodes for readout and spectroscopy respectively: `/DEV.../QACHANNELS/0/READOUT/RESULT/ERROR/CLEAR`, `/DEV.../QACHANNELS/0/SPECTROSCOPY/RESULT/ERROR/CLEAR`, `/DEV.../QACHANNELS/0/READOUT/RESULT/ERROR/JOBIDX`, and `/DEV.../QACHANNELS/0/SPECTROSCOPY/RESULT/ERROR/JOBIDX`.
- QA Channel: Renamed hold-off error count nodes to `/DEV.../QACHANNELS/0/READOUT/RESULT/ERROR/COUNT` (from `/DEV.../QACHANNELS/0/READOUT/RESULT/ERRORS`) and `/DEV.../QACHANNELS/0/SPECTROSCOPY/RESULT/ERROR/COUNT` (from `/DEV.../QACHANNELS/0/SPECTROSCOPY/RESULT/ERRORS`).
- SeqC command `setDIO` now has constant latency, no matter its arguments.

1.7. Release 23.10

Release date: 31-Oct-2023

- SG Channels: Introduced the Output Router and Adder option to enable flexible routing of digital signals between analog outputs.
- SG Channels: Added amplitude registers to the command table to allow independent sweeping or changing of multiple sets of amplitudes.
- QA Channel: Added a second trigger output for the QA Sequencer.
- Added automatic fallback to a link-local IP address in case no DHCP server could be found.
- Added Ethernet-over-USB support on the USB (not maintenance) interface.

Improved ZSync support for multi-state discrimination: ZSync bits assigned to constant values or results of qudits not present in the integrator mask of the **startQA** command are no longer forwarded via the PQSC. Furthermore, changed the default value of the node `/DEV.../QACHANNELS/n/READOUT/MULTISTATE/ZSYNC/PACKED` to be true.

- The holdoff time of the Internal Trigger must now be a multiple of 100 ns, to improve consistency with the PQSC and ensure phase reproducibility when using the LF path.
- SG Channels: The center frequency when using the LF path must now be a multiple of 100 MHz, to ensure phase reproducibility when using the LF path.
- SG Channels: Fixed a minor floating point rounding artifact such that the default marker delay is now correctly displayed as 0s.
- Introduced a new high-performance data-server kernel. It improves reliability and performances of communication with the instrument.

1.8. Release 23.06

Release date: 30-Jun-2023

- General: Added ability to reset all node settings to preset values by writing to `/DEV.../SYSTEM/PRESET/LOAD` node. The nodes `/DEV.../SYSTEM/PRESET/BUSY` and `/DEV.../SYSTEM/PRESET/ERROR` allow for monitoring of the preset status.
- QA Channels: Added switchable signal paths: the RF (0.5 - 8.5 GHz) path and LF (DC - 800 MHz) path. Added nodes for switching between RF and LF paths of the QA channel input and output, respectively, as `/DEV.../QACHANNELS/n/INPUT/RFLFPATH` and `/DEV.../QACHANNELS/n/OUTPUT/RFLFPATH`. Furthermore, the node `/DEV.../QACHANNELS/n/OUTPUT/RFLFINTERLOCK` allows the interlock to be enabled, such that the RF/LF path setting of the output is always configured to match that of the input.
- QA Channels: Cleaned up marker source selection by removing non-functional source settings for the node `/DEV.../QACHANNELS/n/MARKERS/m/SOURCE`, namely "Channel 2, Sequencer Trigger Output" and "Channel 2, Readout done" selection options.
- QA Channels: Fixed a sequencer bug in which **playZero** commands were sometimes skipped when multiple successive **playZero** commands were used with large sample counts (e.g. 131056).
- QA Channels: Added an optional synchronization check which ensures that all participants have reported their readiness before a program or internal triggers are executed. The synchronization check can be enabled by using the following node: `/DEV.../QACHANNELS/n/SYNCHRONIZATION/ENABLE`.
- QA Channels: Fixed a bug in which the spectroscopy delay node `/DEV.../QACHANNELS/n/SPECTROSCOPY/DELAY` didn't accept 0 ns after it was set to 4 ns.
- SG Channels: Updated the default values of trigger input settings to better reflect typical usage. New default values are as follows: Trigger level is now 1 V by default (calibration can lead to values slightly different than 1.0 V), trigger slope detection is now the rising edge by default.
- SG Channels: Introduced the `/DEV.../SGCHANNELS/n/SYNCHRONIZATION/ENABLE`, `/DEV.../SYSTEM/SYNCHRONIZATION/SOURCE`, and `/DEV.../SYSTEM/INTERNALTRIGGER/SYNCHRONIZATION/ENABLE` nodes to make it possible to keep waveform playback synchronized across a full QCCS setup, even in the presence of non-deterministic data transfer times.
- SG Channels: Deprecated the digital mixer reset functionality.
- Manual: Added a section on how to use the synchronization check in the AWG Tab.
- Manual: Added tips to the Basic Waveform Generation Tutorial on how to achieve phase reproducibility in the LF path by using appropriate center frequency and trigger holdoff time settings.
- LabOne: Improved labeling of Trigger settings in the SG AWG, QA Generator, and DIO tabs of the LabOne UI to more clearly mark how a trigger input source corresponds to the front panel input of an SG or QA channel.

1.9. Release 23.02

Release date: 28-Feb-2023

- SG Channels: Extended functionality of **resetOscPhase** to able to reset the phase of the digital mixer (e.g. to enable reproducible phase setting when using the LF path).
- SG Channels: Added ability for internal feedback to use multi-state discrimination data to enable reset of multi-level systems.
- QA Channels: Added power-spectral-density (PSD) measurement capability to the spectroscopy mode. The PSD measurement gets controlled via the node tree branch `/DEV.../QACHANNELS/n/SPECTROSCOPY/PSD`.
- QA Channels: Added node `/DEV.../QACHANNELS/n/INPUT/ADCOVERRANGECOUNT`, allowing users to independently monitor ADC overrange conditions.
-

Manual: Improved UI documentation. Added additional documentation on usage of internal feedback, **playZero**, and **playHold**.

- SG Channels: Improved efficiency of **playZero** and **playHold** to use fewer assembly instructions.
- SG Channels: Improved timing jitter of Output when triggering an SG channel over ZSync.
- LabOne: Improved support for alternative hardware components.
- LabOne: Added ability for the device to prevent LabOne changes that are incompatible with the device hardware.
- Outputs: Improved alignment between SG and QA channel outputs when triggered by the Internal Trigger.
- Scope: Fixed a bug that caused a wrong effective scope delay at startup.
- Outputs: Fixed a bug that caused overrange conditions during the ADC calibration at startup.
- ZSYNC/DIO: Fixed a bug that caused the DIO interface to behave wrongly when switching between LVCMOS and LVDS outputs in non-manual mode.
- LabOne: Fixed the value of legacy node `/DEV.../CLOCKBASE` for SHF devices. To have always the correct value, use the device node `/DEV.../SYSTEM/PROPERTIES/TIMEBASE` instead.

1.10. Release 22.08

Release date: 31-Aug-2022

- LabOne: Added support for the SHFQC Qubit Controller, including full support in the LabOne UI.
- SG Channel: Added new SG-channel sequencer command **getFeedback** that can retrieve ZSync data or qubit data from the QA channel.
- SG Channel: Added ability of `executeTableEntry` to use variable arguments corresponding to qubit data received over ZSync or from the QA channel.
- SG Channel: Fixed a bug in which the command table always required a waveform to make parameter changes.
- SG Channel: Added support for 16 sample waveforms with the command table.
- SG Channel: Added a new sequencer command **playHold** to allow the AWG to hold waveform and marker data for a specified number of samples.
- SG Channel: Improved the speed with which the SG channel AWGs can be enabled.
- QA Channel: Improved the maximal repetition rate, with which the QA readout can be started without missing triggers to 1/(440 ns).
- QA Channel: Fixed a bug, where the error "The sign ADC reported unexpected dataconverter errors!" was issued during an over range condition at the QA channel inputs.
- ZSync/DIO: Added an internal trigger unit that allows the SG and QA channels to be synchronized with each other.
- ZSync/DIO: `/DEV.../DIOS/0/MODE` node changed keyword arguments to enable control of DIO values by the sequencer from `chanNseq` or `channel1N_sequencer` to `sgchanNseq` or `sgchannel1N_sequencer` for the SG channels and `qachanNseq` or `qachannel1N_sequencer` for the QA channel.
- ZSync/DIO: `/DEV.../SGCHANNELS/n/AWG/ELF/DATA` node accepts raw data as 8-, 16-, and 64-bit integer vectors in addition to 32-bit words.
- ZSync/DIO: `/DEV.../QACHANNELS/0/GENERATOR/ELF/DATA` node accepts raw data as 8-, 16-, and 64-bit integer vectors in addition to 32-bit words.

Release date: 01-Apr-2022

Highlights:

- Initial release of the SHFQC user manual.

1.11. Release 24.07 Additional Information

1.11.1. Flexible feedback processing

Feedback in the QCCS is realized by low-latency messages sent between different parties. Regardless of their origin, a sequencer can act on such messages, either by branching or with a conditional pulse play. The first is done by the instructions **getFeedback** followed by branching instructions such as **if**, while the latter by the instruction **executeTableEntry**. For the lowest latency, a conditional play should be preferred.

In most feedback-based experiments, each sequencer of an instrument (e.g. SG Channel 1 of an SHFQC or AWG3 of an HDAWG) only acts on a fraction of the feedback word. To process the feedback word, each sequencer has mask and shift operations available to it, which reduces the feedback word to the required subset of information. An offset could be added; the purpose is to execute a feedback action from a subset of the command table. Multiple feedback sources, as

described below, are available for both feedback instructions and must be specified as first argument. Each source has its dedicated feedback processing chain.

The parameters for processing can now be changed dynamically in realtime between feedback operations, while previously they were static through the entire execution of a sequence.

Format of feedback messages

- **ZSync** : 16 bit messages. The format is given by the source unit. Please refer to the PQSC user manual.
- **Internal** : 32 bit messages, composed of 16 contiguous couples of bits. Each couple represents one integration unit of the QA channel. If the 16W option is not present, only the first eight are available. When Multistate Discrimination (MSD) is enabled, each couple gives the state of the respective qudit in both bits. Without MSD only the lowest bit in each couple is used.



Status until L1 24.04

There are multiple processing sources are available as follows:

Source	Constant	Processing	Description
ZSync	ZSYNC_DATA_RAW	Nothing, feedback word as-is	Returns the data received from ZSync as-is without processing
	ZSYNC_DATA_PQSC_REGISTER	$((\text{word} \gg \text{shift}) \& \text{mask}) + \text{offset}$	Gets last readout register forwarded by the PQSC with processing
	ZSYNC_DATA_PQSC_DECODER	$((\text{word} \gg \text{shift}) \& \text{mask}) + \text{offset}$	Gets last output of the decoder received from the PQSC with processing
Internal	QA_DATA_RAW	Nothing, feedback word as-is	Returns the data received from the QA channel as is without processing. Can be used only with getFeedback
	QA_DATA_PROCESSED	$((\text{word} \gg \text{shift}) \& \text{mask}) + \text{offset}$	Returns the processed data received from the QA channel. When used with getFeedback , the offset is omitted.

The processing of non-RAW sources can be controlled by the values set in the following nodes, located in the awg branch of the considered channel.

Source constant	Controls	Limits
ZSYNC_DATA_PQSC_REGISTER	shift = zsync/register/shift	0 - 15
	mask = zsync/register/mask	0 - 0xFFFF
	offset = zsync/register/offset	0 - 4095
ZSYNC_DATA_PQSC_DECODER	shift = zsync/decoder/shift	0 - 7
	mask = zsync/decoder/mask	0 - 0xFF
	offset = zsync/decoder/offset	0 - 4095
QA_DATA_PROCESSED	shift = intfeedback/direct/shift	0 - 32
	mask = intfeedback/direct/mask	0 - 0xFFFFFFFF
	offset = intfeedback/direct/offset	0 - 4095

Example

Active qubit reset

To perform active qubit reset, the command table of the SG channel that controls the target qubit would be programmed with these two entries:

Index	Waveform	playZero	Oscillator	Comment
0	None	WFM_LEN	None	No action
1	wfm_pi	None	0	Pi-pulse

The oscillator 0 should be set to the the e-g transition frequency.

If the feedback is routed through the PQSC, and assuming that **RESULT_INDEX** holds the index of the result specified in the register forwarding unit, the feedback control nodes would be programmed as follows:

```
shfqc.sgchannels[SG_CHANNEL].awg.zsync.register.shift(RESULT_INDEX * 2)
shfqc.sgchannels[SG_CHANNEL].awg.zsync.register.mask(0b1)
shfqc.sgchannels[SG_CHANNEL].awg.zsync.register.offset(0)
```

and the seqc would look like this

```
waitZSyncTrigger();
playZero(feedback_latency_pz);
executeTableEntry(ZSYNC_DATA_PQSC_REGISTER, feedback_latency);
```

If feedback is done with the internal feedback unit, and assuming that **INT_UNIT** holds the index of the QA integration unit that reads the target qubit, the feedback control nodes would be programmed as follows:

```
shfqc.sgchannels[SG_CHANNEL].awg.intfeedback.direct.shift(INT_UNIT * 2)
shfqc.sgchannels[SG_CHANNEL].awg.intfeedback.direct.mask(0b1)
shfqc.sgchannels[SG_CHANNEL].awg.intfeedback.direct.offset(0)
```

and the seqc would look like this

```
waitZSyncTrigger();
playZero(feedback_latency_pz);
executeTableEntry(QA_DATA_PROCESSED, feedback_latency);
```

New behavior since L1 24.07

The configuration of the feedback processing has been changed from nodes to a dedicated seqc instruction: **configureFeedbackProcessing**. In this way the processing parameters can be changed dynamically during the sequence, so different feedback actions can be performed sequentially with the same feedback input. For ZSync feedback, the feedback process chain is not tied anymore to a specific feedback processing unit (register forwarding or decoder). Therefore, the source selectors have been renamed to be more generic.

The processing sources are available as follows:

Source	Constant	Processing	Description
ZSync	ZSYNC_DATA_RAW	Nothing, feedback word as-is	Returns the data received from ZSync as-is without processing
	ZSYNC_DATA_PROCESSED_A	$((\text{word} \gg \text{shift}) \& (2^{**\text{length}} - 1)) + \text{offset}$	Returns last feedback received from ZSync with processing

Source	Constant	Processing	Description
	ZSYNC_DATA_PROCESSED_B	$((\text{word} \gg \text{shift}) \& (2^{**}\text{length} - 1)) + \text{offset}$	Returns last feedback received from ZSync with processing
Internal	QA_DATA_RAW	Nothing, feedback word as-is	Returns the data received from the QA channel as is without processing. Can be used only with getFeedback
	QA_DATA_PROCESSED	$((\text{word} \gg \text{shift}) \& (2^{**}\text{length} - 1)) + \text{offset}$	Returns the processed data received from the QA channel.

ZSYNC_DATA_PROCESSED_A and ZSYNC_DATA_PROCESSED_B offer identical capabilities on the ZSync feedback source, but they can be configured differently.

The processing of non-RAW sources can be controlled by the command **configureFeedbackProcessing** as follows:

```
void configureFeedbackProcessing(FB_PATH, SHIFT, LENGTH, OFFSET)
```

- ─ **FB_PATH** specify the feedback path whose parameters should be changed. It can be ZSYNC_DATA_PROCESSED_A, ZSYNC_DATA_PROCESSED_B or QA_DATA_PROCESSED.
- ─ **SHIFT** Specify how many bits the feedback word should be right shifted. It can be between 0 and 15 for ZSync paths and between 0 and 31 for the internal path.
- ─ **LENGTH** sets the length of the trimming of the feedback message after the shift. It's implemented as binary masking with a mask equal to $2^{**}\text{LENGTH} - 1$, or in other words, a mask with **LENGTH** ones. For example, to reduce a message to a single bit it should be set to 1, to reduce a message to two bits it should be set to 2 and so on. It can be between 1 and 16. If the feedback is processed with **executeTableEntry**, only values up to 12 are meaningful.
- ─ **OFFSET** sets the additive offset applied after shift and length trimming. It can be between 0 and 4095.

The instruction is blocking and requires a minimal waveform length of 48 to be gap-free.

If the feedback processing is not configured, the feedback message will be passed as-is; ZSYNC_DATA_PROCESSED_A or ZSYNC_DATA_PROCESSED_B will behave identically as ZSYNC_DATA_RAW. Same for QA_DATA_PROCESSED and QA_DATA_RAW.

The constants ZSYNC_DATA_PQSC_REGISTER and ZSYNC_DATA_PQSC_DECODER are kept for limited backwards compatibility, but they are deprecated. They now behave identically to ZSYNC_DATA_PROCESSED_A and ZSYNC_DATA_PROCESSED_B respectively.

The control of parameters with nodes is not available anymore.

Example

Active qubit reset

To implement the same sequence as with the older version, the following seqc can be used for ZSync feedback

```
configureFeedbackProcessing(ZSYNC_DATA_PROCESSED_A, RESULT_INDEX * 2, 1, 0);
waitZSyncTrigger();
playZero(feedback_latency_pz);
executeTableEntry(ZSYNC_DATA_PROCESSED_A, feedback_latency);
```

For internal feedback, ZSYNC_DATA_PROCESSED_A should be replaced with QA_DATA_PROCESSED.

In both case, no feedback processing node setting is required anymore.

Active qutrit reset

To perform active qubit reset, the command table of the SG channel that controls the target qubit would be programmed with these entries:

Index	Waveform	playZero	Oscillator	Comment
0	None	WFM_LEN	None	No action
1	wfm_pi_eg	None	0	Pi-pulse e-g
2	wfm_pi_fe	None	1	Pi-pulse f-e

The oscillator 0 should be set to the e-g transition frequency, while oscillator 1 with the f-e frequency.

The sequence should be as follow

```
configureFeedbackProcessing(ZSYNC_DATA_PROCESSED_A, RESULT_INDEX * 2, 2, 0);
waitZSyncTrigger();
playZero(feedback_latency_pz);
executeTableEntry(ZSYNC_DATA_PROCESSED_A, feedback_latency);
configureFeedbackProcessing(ZSYNC_DATA_PROCESSED_A, RESULT_INDEX + 1, 1, 0);
executeTableEntry(ZSYNC_DATA_PROCESSED_A);
```

In the first conditional playback, the entire two-bit feedback word is used, so any entry of the command table entry can be used, depending on the starting condition. During the playback, the feedback processing is reconfigured, so that in the second playback, an e-g pulse is played only if the starting state was f.

2. Getting Started

This first chapter guides you through the initial set-up of your SHFQC+ Instrument in order to make your first measurements.

Please refer to:

- [Quick Start Guide](#) for a Quick Start Guide for the impatient.
- [Inspect the Package Contents](#) for inspecting the package content and accessories.
- [Handling and Safety Instructions](#) for a list of essential handling and safety instructions.
- [Software Installation - Software Update](#) for help connecting to the SHFQC+ Instrument with the LabOne software.
- [Troubleshooting](#) for a handy list of troubleshooting guidelines.

This chapter is delivered as a hard copy with the instrument upon delivery. It is also the first chapter of the SHFQC+ User Manual.

2.1. Quick Start Guide

This page addresses all the people who have been impatiently awaiting their new gem to arrive and want to see it up and running quickly. Please proceed with the following steps:


1. [Inspect the package contents](#). Besides the Instrument there should be a country-specific power cable, a USB cable, an Ethernet cable, a ZSync cable, and a hard copy of the [Getting Started guide](#).
2. Check [Handling and Safety Instructions](#) for the Handling and Safety Instructions.
3. Download and install the latest LabOne software from the [Zurich Instruments Download Center](#).
4. Choose the download file that suits your computer (e.g. Windows with 64-bit addressing). For more detailed information see [Software Installation](#).
5. Connect the instrument to the power outlet. Turn it on and connect it to a switch in the LAN using the Ethernet cable.
6. Start the LabOne User Interface from the Windows Start Menu. The default web browser will open and display your instrument in a start screen as shown below. Use Chrome, Edge, Firefox, or Opera for best user experience.



7. The LabOne User Interface start-up screen will appear. Click the **Open** button on the lower right of the page. The default configuration will be loaded and the first signals can be generated. If the user interface does not start up successfully, please refer to [Connecting to the Instrument](#).

If any problems occur while setting up the instrument and software, please see [Troubleshooting](#) at the end of this chapter for troubleshooting.

When connecting cables to the instrument's SMA ports, use a torque wrench specified for brass core SMA (4 in-lbs, 0.5 Nm). Using a standard SMA torque wrench (8 in-lbs) or a wrench without torque limit can damage the connectors.

After you have finished using the instrument, it is recommended to shut it down using the soft power button on the front panel of the instrument or by clicking on the  button at the bottom left of the user interface screen before turning off the power switch on the back panel of the instrument.

Once the Instrument is up and running we recommend going through some of the tutorials given in [Tutorials](#). The functional description of the SHFQC+ can be found in [Functional Description](#) and provides a general introduction to the various tools and tables in each section describing every setting. In the same section, [Functional Description](#) provides an overview of the different UI tabs. For specific application know-how, the [blog section](#) of the Zurich Instruments website will serve as a valuable resource that is constantly updated and expanded.

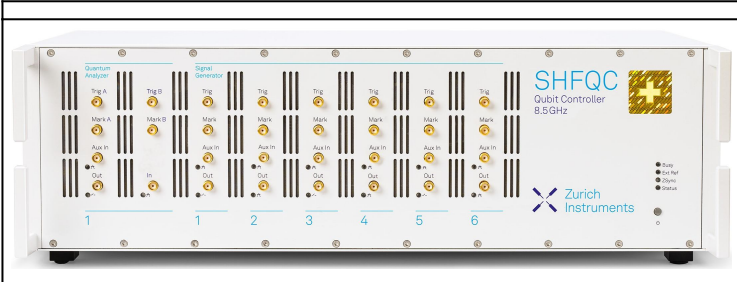




2.2. Inspect the Package Contents


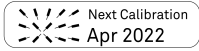

If the shipping container appears to be damaged, keep the container until you have inspected the contents of the shipment and have performed basic functional tests.

Please verify the following:

- You have received 1 Zurich Instruments SHFQC+ Instrument
- You have received 1 power cord with a power plug suited to your country
- You have received 1 USB 3.0 cable and/or 1 LAN cable (category 5/6 required)
- You have received 1 Zurich Instruments ZSync cable
- You have received a printed version of the "Getting Started" section
- The "Next Calibration" sticker on the rear panel of the instrument indicates a date approximately 2 years in the future → Zurich Instruments recommends calibration intervals of 2 years
- The MAC address of the instrument is displayed on a sticker on the back panel

Table 2.1: Package contents for the SHFQC+

	SHFQC+ instrument
	the power cord (e.g. EU norm)
	the USB 3.0 cable
	the power inlet, with power switch
	the LAN / Ethernet cable (category 5/6 required)

	the ZSync cable
	the "Next Calibration" sticker on the back panel of the instrument
	the MAC address sticker on the back panel of the instrument

The SHFQC+ Instrument is equipped with a multi-mains switched power supply, and therefore can be connected to most power systems in the world. The fuse holder is integrated with the power inlet and can be extracted by grabbing the holder with two small screwdrivers at the top and at the bottom at the same time. A spare fuse is contained in the fuse holder. The fuse description is found in the specifications chapter.

Carefully inspect your instrument. If there is mechanical damage or the instrument does not pass the basic tests, then you should immediately notify the Zurich Instruments support team through [email](#).

2.3. Handling and Safety Instructions

The SHFQC+ Instrument is a sensitive piece of electronic equipment, and under no circumstances should its casing be opened, as there are high-voltage parts inside which may be harmful to human beings. There are no serviceable parts inside the instrument. Do not install substitute parts or perform any unauthorized modification to the product. Opening the instrument immediately voids the warranty provided by Zurich Instruments.

Do not use this product in any manner not specified by the manufacturer. The protective features of this product may be affected if it is used in a way not specified in the operating instructions.

The following general safety instructions must be observed during all phases of operation, service, and handling of the instrument. The disregard of these precautions and all specific warnings elsewhere in this manual may negatively affect the operation of the equipment and its lifetime.

Zurich Instruments assumes no liability for the user's failure to observe and comply with the instructions in this user manual.

Caution

The SMA connectors on the front panel are made for transmitting radio frequencies and can be damaged if handled inappropriately. Take care when attaching or detaching cables or when moving the instrument.

Table 2.2: Safety Instructions

Ground the instrument	The instrument chassis must be correctly connected to earth ground by means of the supplied power cord. The ground pin of the power cord set plug must be firmly connected to the electrical ground (safety ground) terminal at the mains power outlet. Interruption of the protective earth conductor or disconnection of the protective earth terminal will cause a potential shock hazard that could result in personal injury and potential damage to the instrument.
Ground loops	The SMA connectors are not floating. For sensitive operations and in order to avoid ground loops, consider adding dc-blocks at the Inputs of the device.
Electromagnetic environment	This equipment has been certified to conform with industrial electromagnetic environment as defined in EN 61326-1. Emissions, that exceed the levels required by the document referenced above, can occur when connected to a test object.





Measurement category	This equipment is of measurement category I (CAT I). Do not use it for CAT II, III, or IV. Do not connect the measurement terminals to mains sockets.
Maximum ratings	The specified electrical ratings for the connectors of the instrument should not be exceeded at any time during operation. Please refer to the Specifications for a comprehensive list of ratings.
Do not service or adjust anything yourself	There are no serviceable parts inside the instrument.
Software updates	Frequent software updates provide the user with many important improvements as well as new features. Only the last released software version is supported by Zurich Instruments.
Warnings	Instructions contained in any warning issued by the instrument, either by the software, the graphical user interface, the notes on the instrument or mentioned in this manual, must be followed.
Notes	Instructions contained in the notes of this user manual are of essential importance for correctly interpreting the acquired measurement data.
Location and ventilation	This instrument or system is intended for indoor use in an installation category II and pollution degree 2 environment as per IEC 61010-1. Do not operate or store the instrument outside the ambient conditions specified in the Specifications section. Do not block the ventilator opening on the back or the air intake on the chassis side and front, and allow a reasonable space for the air to flow.
Cleaning	To prevent electrical shock, disconnect the instrument from AC mains power and disconnect all test leads before cleaning. Clean the outside of the instrument using a soft, lint-free cloth slightly dampened with water. Do not use detergent or solvents. Do not attempt to clean internally.
AC power connection and mains line fuse	For continued protection against fire, replace the line fuse only with a fuse of the specified type and rating. Use only the power cord specified for this product and certified for the country of use. Always position the device so that its power switch and the power cord are easily accessible during operation.
Main power disconnect	Unplug product from wall outlet and remove power cord before servicing. Only qualified, service-trained personnel should remove the cover from the instrument.
RJ45 sockets labeled ZSync	The RJ45 sockets on the back panel labeled "ZSync 1/2" are not intended for Ethernet LAN connection. Connecting an Ethernet device to these sockets may damage the instrument and/or the Ethernet device.
Operation and storage	Do not operate or store the instrument outside the ambient conditions specified in the Specifications section.
Handling	Handle with care. Do not drop the instrument. Do not store liquids on the device, as there is a chance of spillage resulting in damage.
Safety critical systems	Do not use this equipment in systems whose failure could result in loss of life, significant property damage or damage to the environment.

If you notice any of the situations listed below, immediately stop the operation of the instrument, disconnect the power cord, and contact the support team at Zurich Instruments, either through the website form or through [email](#).

Table 2.3: Unusual Conditions

Fan is not working properly or not at all	Switch off the instrument immediately to prevent overheating of sensitive electronic components.
Power cord or power plug on instrument is damaged	Switch off the instrument immediately to prevent overheating, electric shock, or fire. Please exchange the power cord only with one for this product and certified for the country of use.
Instrument emits abnormal noise, smell, or sparks	Switch off the instrument immediately to prevent further damage.
Instrument is damaged	Switch off the instrument immediately and ensure it is not used again until it has been repaired.

Table 2.4: Symbols

	Earth ground
	Chassis ground
	Caution. Refer to accompanying documentation
	DC (direct current)

2.4. Software Installation

The SHFQC+ Instrument is operated from a host computer with the LabOne software. To install the LabOne software on a computer, administrator rights may be required. In order to simply run the software later, a regular user account is sufficient. Instructions for downloading the correct version of the software packages from the Zurich Instruments website are described below in the platform-dependent sections. It is recommended to regularly update to the latest software version provided by Zurich Instruments. Thanks to the Automatic Update check feature, the update can be initiated with a single click from within the user interface, as shown in [Software Update](#).

2.4.1. Installing LabOne on Windows

The installation packages for the Zurich Instruments LabOne software are available as Windows installer .msi packages. The software is available on the [Zurich Instruments Download Center](#). Please ensure that you have administrator rights for the PC on which the software is to be installed. See [LabOne compatibility](#) for a comprehensive list of supported Windows systems.

2.4.2. Windows LabOne Installation

1. The SHFQC+ Instrument should not be connected to your computer during the LabOne software installation process.
2. Start the LabOne installer program with a name of the form **LabOne64-XX.XX.XXXXX.msi** by a double click and follow the instructions. Windows Administrator rights are required for installation. The installation proceeds as follows:
 - On the welcome screen click the **Next** button.

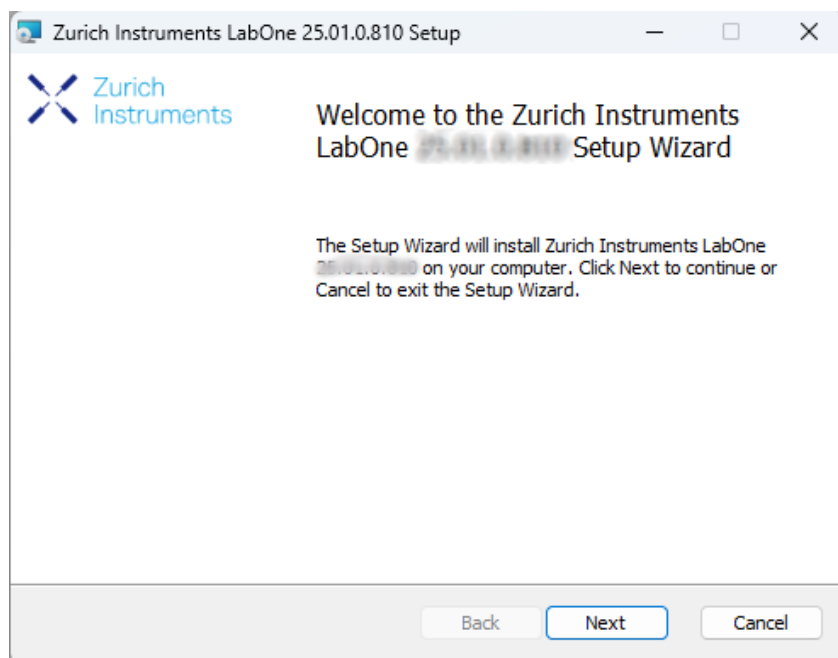


Figure 2.1: Installation welcome screen

- After reading through the Zurich Instruments license agreement, check the "I accept the terms in the License Agreement" check box and click the **Next** button.
- Review the features you want to have installed. For the SHFQC+ Instrument the "SHFQC+ Series Device", "LabOne User Interface" and "LabOne APIs" features are required. Please install the features for other device classes as well, if required. To proceed click the **Next** button.

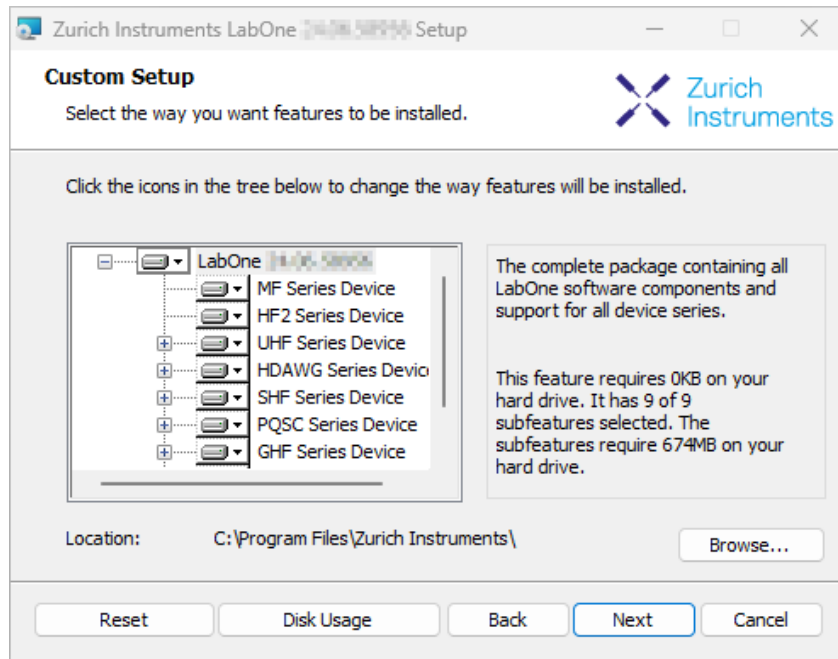


Figure 2.2: Custom setup screen

- Select whether the software should periodically check for updates. Note, the software will still not update automatically. This setting can later be changed in the user interface. If you would like to install shortcuts on your desktop area, select "Create a shortcut for this program on the desktop". To proceed click the **Next** button.

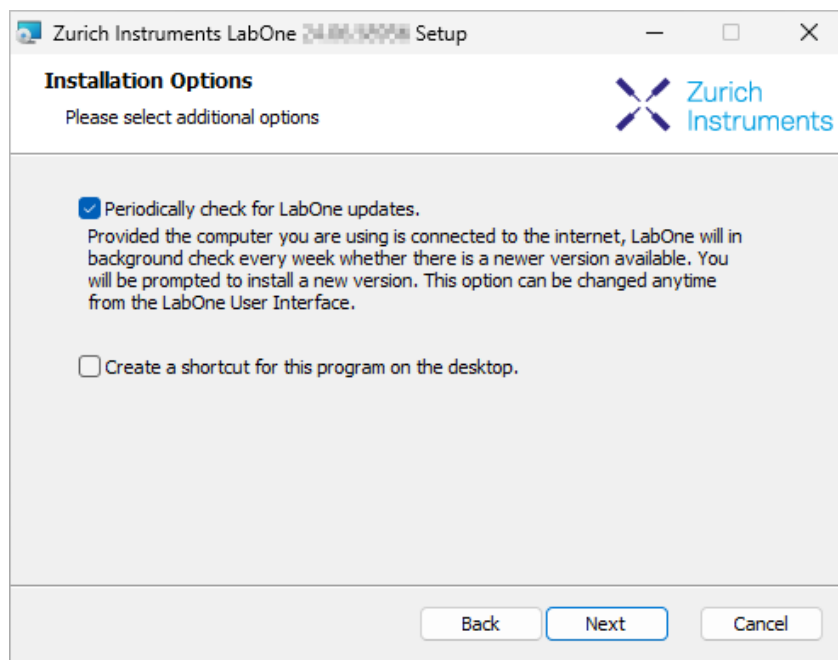


Figure 2.3: Automatic update check

- Click the **Install** button to start the installation process.
- Windows may ask up to two times to reboot the computer if you are upgrading. Make sure you have no unsaved work on your computer.

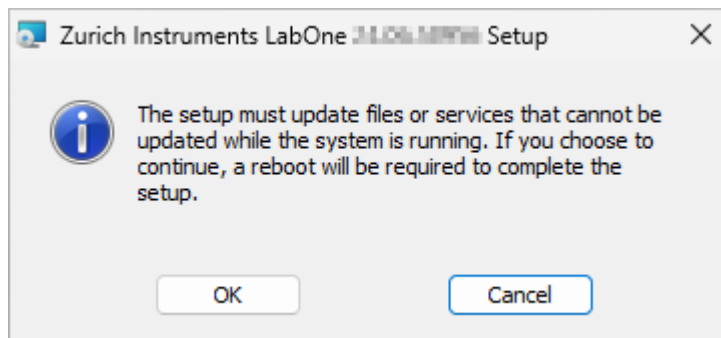


Figure 2.4: Installation reboot request

- During the first installation of LabOne, it is required to confirm the installation of some drivers from the trusted publisher Zurich Instruments. Click on **Install**.

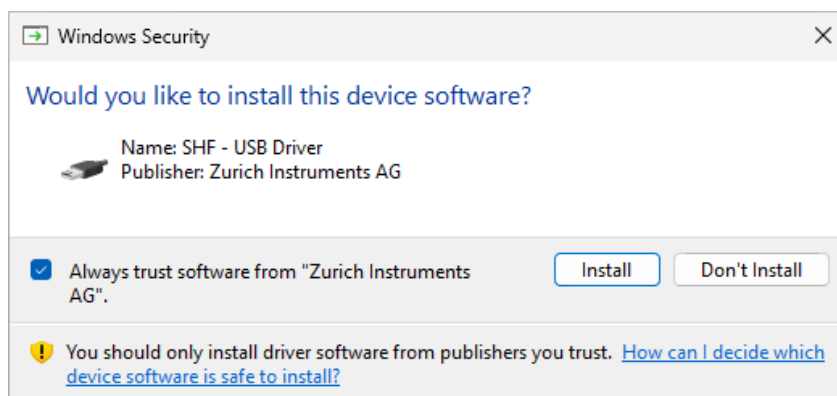


Figure 2.5: Installation driver acceptance

- Click **OK** on the following notification dialog.

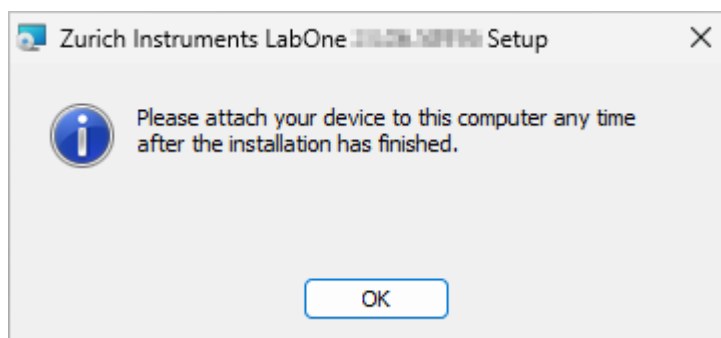


Figure 2.6: Installation completion screen

- Click **Finish** to close the Zurich Instruments LabOne installer.
- You can now start the LabOne User Interface as described in [LabOne Software Start-up](#) and choose an instrument to connect to via the Device Connection dialog shown in [Device Connection dialog](#).

Warning

Do not install drivers from another source other than Zurich Instruments.

2.4.3. Running LabOne manually from the Command Line

After installing the LabOne software, the Web Server and Data Server can be started manually using the command-line. The more common way to start LabOne under Windows is described in [LabOne Software Start-up](#). The advantage of using the command line is being able to observe and change the behavior of the Web and Data Servers.

Running the Web Server from the Command Line

Before running the Web Server from the terminal, the user needs to ensure there is no other instance of the Web Server running in the background, since only one instance of the Web Server can run on a computer at a time. This can be checked using the Tray Icon as shown below.

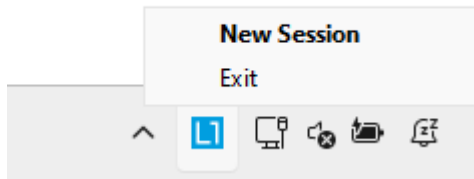


Figure 2.7: LabOne Tray Icon in Windows 11

To start the Web Servers manually, open a command-line terminal (Command Prompt, PowerShell (Windows) or Bash (Linux)). The current working directory needs to be the installation directory of the Web Server, usually `C:\Program Files\Zurich Instruments\LabOne\WebServer`. The behavior of the Web Server can be changed by providing command line arguments. For a detailed list of all arguments see the command line help text:

```
$ ziWebServer --help
```

One useful application of running the Webserver manually from a terminal window is to change the data directory from its default path in the user home directory. The data directory is a folder in which the LabOne Webserver saves all the measured data in the format specified by the user.

The corresponding command line argument to specify the data path is `--data-path` and the command to start the LabOne Webserver with a non-default directory path, e.g., `C:\data` is

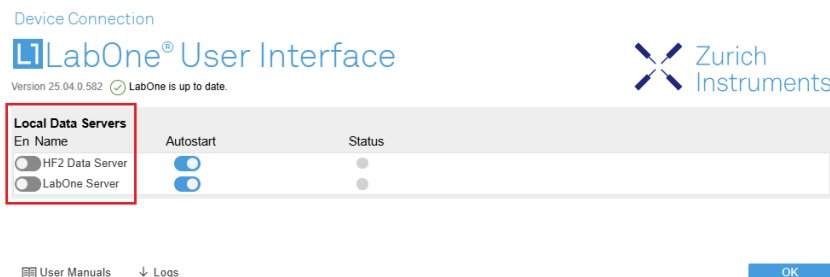
```
C:\Program Files\Zurich Instruments\LabOne\WebServer> ziWebServer --data-path "C:\data"
```

Running the Data Server from the Command Line

By default, the Data Server runs on Windows as a background service. To avoid conflicts with TCP port assignment, before running the Data Server from the terminal the user needs to ensure that the Data Server running in the background is stopped.

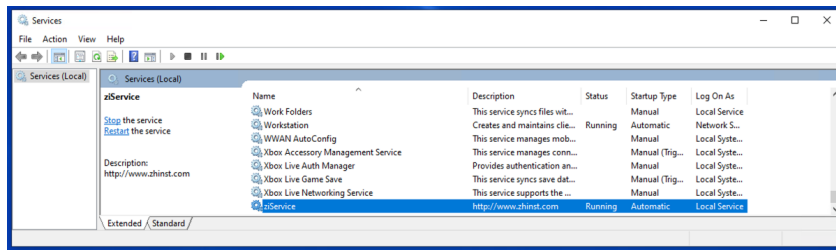
There are two ways to enable/disable the data servers, one from the LabOne user interface and one from the Windows services application.

In the "Advanced" mode of LabOne Session Manager, press the "Configure" button to open the following window for switching on/off the data servers.



Alternatively, open the Windows "Services" app as shown below, look for the `ziService`, right click on it and click "Stop".

2.4. Software Installation



Now that the Data Server is not running anymore in the background, it can be started manually. Open a command-line terminal (Command Prompt, PowerShell (Windows) or Bash (Linux)) and run:

```
PS C:\Users\user> & 'C:\Program Files\Zurich  
Instruments\LabOne\DataServer\ziDataServer.exe'
```

To show logs with higher verbosity, the `--debug 1` flag can be used:

```
PS C:\Users\user> & 'C:\Program Files\Zurich  
Instruments\LabOne\DataServer\ziDataServer.exe' --debug 1
```

2.4.4. Windows LabOne Uninstallation

To uninstall the LabOne software package from a Windows computer, one can open the "Installed apps" page from the Windows start menu and search for LabOne. By selecting the LabOne item in the list of apps, the user has the option to "Uninstall" or "Modify" the software package as shown in Figure 2.8.

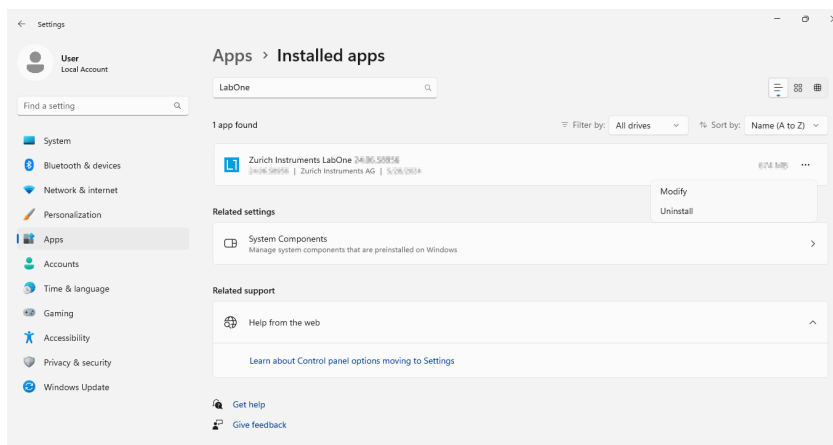


Figure 2.8: Uninstallation of LabOne on Windows computers

Warning

Although it is possible to install a new version of LabOne on a currently-installed version, it is highly recommended to first uninstall the older version of LabOne from the computer and then, install the new version. Otherwise, if the installation process fails, the current installation is damaged and cannot be uninstalled directly. The user will need to first repair the installation and then, uninstall it.

In case a current installation of LabOne is corrupted, one can simply repair it by selecting the option "Modify" in Figure 2.8. This will open the LabOne installation wizard with the option "Repair" as shown in Figure 2.9.

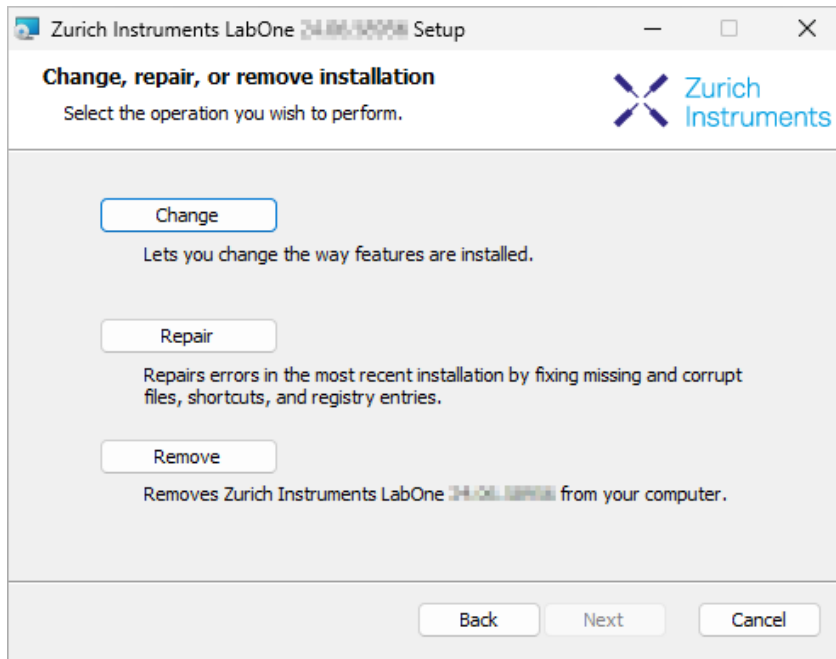


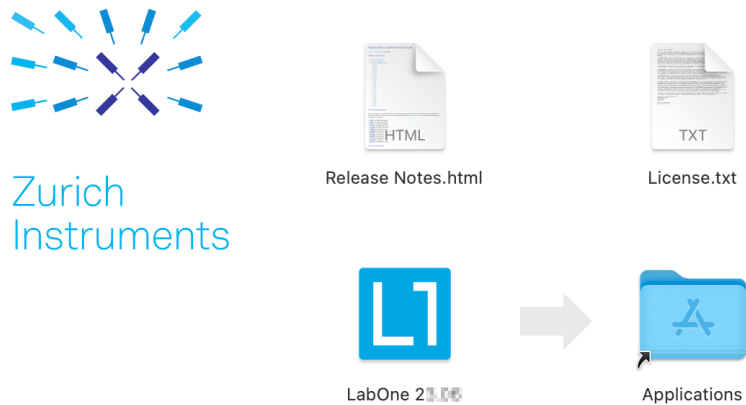
Figure 2.9: Repair of LabOne on Windows computers

After finishing the repair process, the normal uninstallation process described above can be triggered to uninstall LabOne.

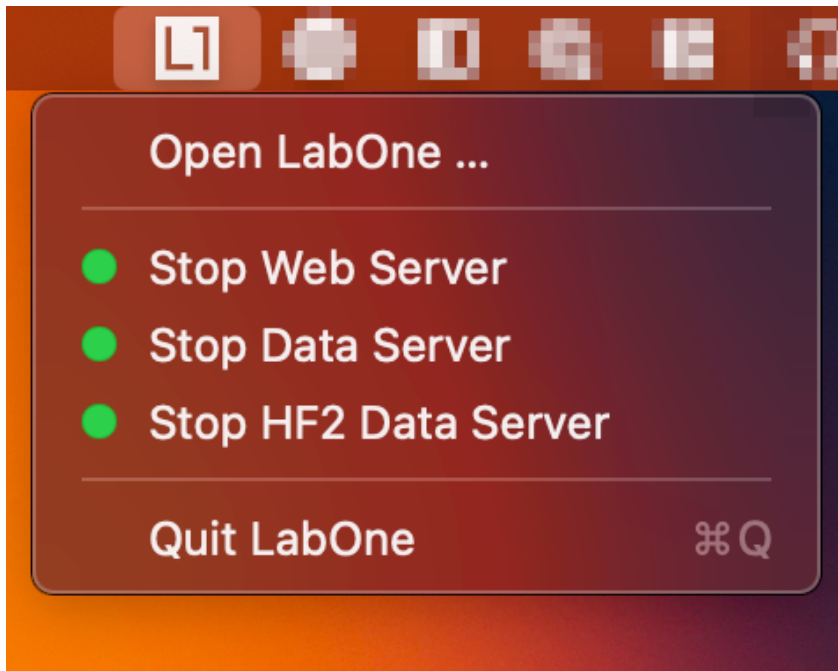
2.4.5. Installing LabOne on macOS

LabOne supports both Intel and ARM (M-series) architectures within a single universal disk image (DMG) file available in our Download Center.

- Download and double-click the DMG file to mount the image.



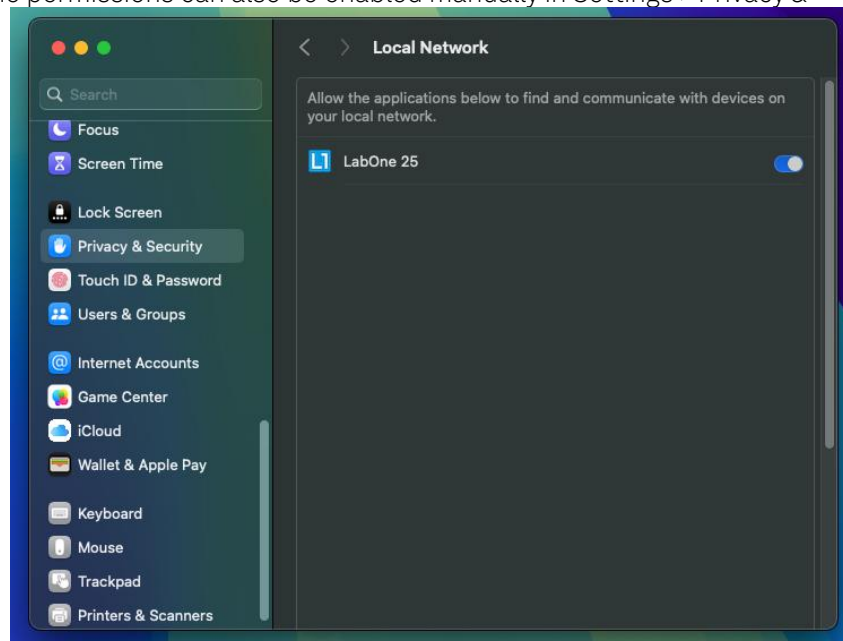
- The image contains a single LabOne application with all services needed.
- Once the application is started, a labone icon will appear in the menu bar. It allows the user to easily open a new session and shows the status of all services.



Note

LabOne needs Local Network Access permissions. When LabOne is first started, a pop-up will appear asking to grant such permissions.

If you miss the pop-up, the permissions can also be enabled manually in Settings > Privacy &



Security > Local Network.

2.4.6. Uninstalling LabOne on macOS

To uninstall LabOne on macOS, simply drag the LabOne application to the trash bin.

2.4.7. Application Content

The LabOne application contains all resources available for macOS. This includes:

- The binaries for the Web Server and Data Servers.
- The binaries for the C, MATLAB, and LabVIEW APIs.
- An offline version of the user manuals.

- The latest firmware images for all instruments.

To access this content, right-click on the LabOne application and select "Show Package Contents". Then, go into Contents/Resources.

Note

Since the application name contains a space, one needs to escape it when using the command line to access the contents: `cd /Applications/LabOne\ XX.XX.app/Contents/Resources`

2.4.8. Start LabOne Manually on the Command Line

To start the LabOne services like the data server and web server manually, one can use the command line.

The data server binary is called **ziDataServer** (**ziServer** for HF2 instruments) and is located at `Applications/LabOne\ XX.XX.app/Contents/Resources/DataServer/`.

The web server binary is called **ziWebServer** and is located at `Applications/LabOne\ XX.XX.app/Contents/Resources/DataServer/`.

Note

No special command line arguments are needed to start the LabOne services. Use the `--help` argument to see all available options.

2.4.9. Installing LabOne on Linux

2.4.10. Requirements

Ensure that the following requirements are fulfilled before trying to install the LabOne software package:

1. LabOne software supports typical modern GNU/Linux distributions (Ubuntu 14.04+, CentOS 7+, Debian 8+). The minimum requirements are glibc 2.17+ and kernel 3.10+.
2. You have administrator rights for the system.
3. The correct version of the LabOne installation package for your operating system and platform have been downloaded from the Zurich Instruments [Download Center](#):

```
LabOneLinux<arch>-<release>-<revision>.tar.gz,
```

Please ensure you download the correct architecture (x86-64 or arm64) of the LabOne installer. The `uname` command can be used in order to determine which architecture you are using, by running:

```
uname -m
```

in a command line terminal. If the command outputs **x86_64** the x86-64 version of the LabOne package is required, if it displays **aarch64** the ARM64 version is required.

2.4.11. Linux LabOne Installation

Proceed with the installation in a command line shell as follows:

1. Extract the LabOne tarball in a temporary directory:

```
tar xzvf LabOneLinux<arch>-<release>-<revision>.tar.gz
```

2. Navigate into the extracted directory.

```
cd LabOneLinux<arch>-<release>-<revision>
```

3. Run the install script with administrator rights and proceed through the guided installation, using the default installation path if possible:

```
sudo bash install.sh
```

The install script lets you choose between the following three modes:

- Type "a" to install the Data Server program, the Web Server program, documentation and APIs.
 - Type "u" to install **udev** support (only necessary if HF2 Instruments will be used with this LabOne installation and not relevant for other instrument classes).
 - Type "ENTER" to install both options "a" and "u".
4. Test your installation by running the software as described in the next section.

2.4.12. Running the Software on Linux

The following steps describe how to start the LabOne software in order to access and use your instrument in the User Interface.

1. Start the Web Server program at a command prompt:

```
$ ziWebServer
```

2. Start an up-to-date web browser and enter the **127.0.0.1:8006** in the browser's address bar to access the Web Server program and start the LabOne User Interface. The LabOne Web Server installed on the PC listens by default on port number 8006 instead of 80 to minimize the probability of conflicts.
3. You can now start the LabOne User Interface as described in [LabOne Software Start-up](#) and choose an instrument to connect to via the Device Connection dialog shown in [Device Connection dialog](#).

Important

Do not use two Data Server instances running in parallel; only one instance may run at a time.

2.4.13. Uninstalling LabOne on Linux

The LabOne software package copies an uninstall script to the base installation path (the default installation directory is **/opt/zi/**). To uninstall the LabOne package please perform the following steps in a command line shell:

1. Navigate to the path where LabOne is installed, for example, if LabOne is installed in the default installation path:

```
$ cd /opt/zi/
```

2. Run the uninstall script with administrator rights and proceed through the guided steps:

```
$ sudo bash uninstall_LabOne<arch>-<release>-<revision>.sh
```

2.5. Connecting to the Instrument

The Zurich Instruments SHFQC+ is operated using the LabOne software. After the installation as described in [Software Installation](#), the instrument can be connected to the host computer using either the USB 3.0 or the 1 Gbit/s Ethernet (1GbE). Please use the respective cables supplied with the instrument. Once one of the physical connection achieved successfully, the LabOne software can recognize the instrument.

Note

The following web browsers are supported (latest versions).



- Using the 1GbE port, it is possible to connect the instrument to an existing local area network (LAN) or establish a point-to-point connection to the host computer. For further details, see [1GbE Connectivity](#)
- Using the USB port requires point-to-point connection to the host computer. For further information, see [USB Connectivity](#).

Note

It is recommended to use the 1GbE port for communicating with the instrument, especially for long-running experiments while measured signals are continuously acquired for an extended period of time. This is to avoid possible interruptions that the USB protocol might cause depending on the host computer's USB settings.

2.5.1. LabOne Software Architecture

The Zurich Instruments LabOne software gives quick and easy access to the instrument from a host PC. LabOne also supports advanced configurations with simultaneous access by multiple software clients (i.e., LabOne User Interface clients and/or API clients), and even simultaneous access by several users working on different computers. Here we give a brief overview of the architecture of the LabOne software. This will help to better understand the following chapters.

The software of Zurich Instruments equipment is server-based. The servers and other software components are organized in layers as shown in [Figure 2.10](#).

- The lowest layer running on the PC is the LabOne Data Server, which is the interface to the connected instrument.
- The middle layer contains the LabOne Web Server, which is the server for the browser-based LabOne User Interface.
- The graphical user interface, together with the programming user interfaces, are contained in the top layer.

The architecture with one central Data Server allows multiple clients to access a device with synchronized settings. The following sections explain the different layers and their functionality in more detail.

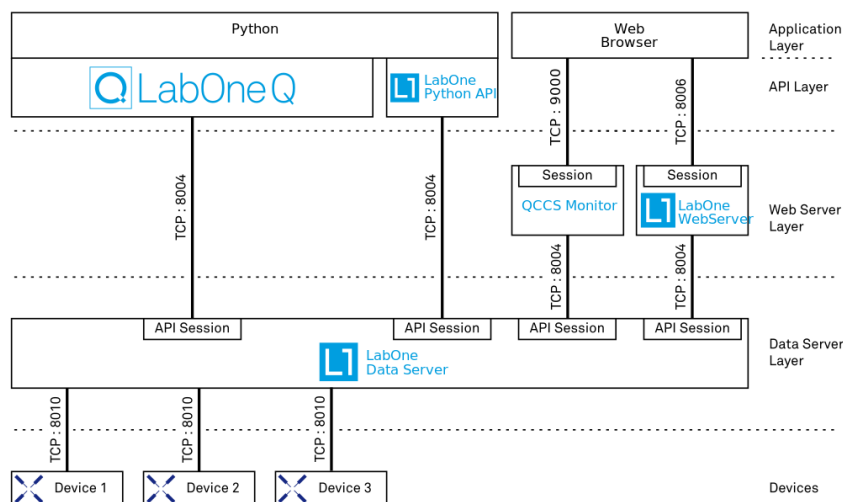


Figure 2.10: LabOne Software architecture

2.5.2. LabOne Data Server

The **LabOne Data Server** program is a dedicated server that is in charge of all communication to and from the device. The Data Server can control a single or also multiple instruments. It will distribute the measurement data from the instrument to all the clients that subscribe to it. It also ensures that settings changed by one client are communicated to other clients. The device settings are therefore synchronized on all clients. On a PC, only a single instance of a LabOne Data Server should be running.

2.5.3. LabOne Web Server

The LabOne Web Server is an application dedicated to serving up the web pages that constitute the LabOne user interface. The user interface can be opened with any device with a web browser. Since it is touch enabled, it is possible to work with the LabOne User Interface on a mobile device - like a tablet. The LabOne Web Server supports multiple clients simultaneously. This means that more than one session can be used to view data and to manipulate the instrument. A session could be running in a browser on the PC on which the LabOne software is installed. It could equally well be running in a browser on a remote machine.

With a LabOne Web Server running and accessing an instrument, a new session can be opened by typing in a network address and port number in a browser address bar. In case the Web Server runs on the **same** computer, the address is the localhost address (both are equivalent):

- ─ **127.0.0.1:8006**
- ─ **localhost:8006**

In case the Web Server runs on a **remote** computer, the address is the IP address or network name of the remote computer:

- ─ **192.168.x.y:8006**
- ─ **myPC.company.com:8006**

The most recent versions of the most popular browsers are supported: Chrome, Firefox, Edge, Safari and Opera.

2.5.4. LabOne API Layer

The instrument can also be controlled via the application program interfaces (APIs) provided by Zurich Instruments. APIs are provided in the form of DLLs for the following programming environments:

- ─ MATLAB
- ─ Python
- ─ LabVIEW
- ─ .NET
- ─ C

APIs are provided in the form of DLLs for the following programming environments:

- ─ MATLAB
- ─ Python

An extensive Python API and python-based drivers are provided for the following frameworks:

- ─ <https://github.com/zhinst/zhinst-toolkit>[Zurich Instruments Toolkit]
- ─ <https://github.com/zhinst/zhinst-qcodes>[QCoDeS]
- ─ <https://github.com/zhinst/zhinst-labber>[Labber]

The instrument can therefore be controlled by an external program, and the resulting data can be processed there. The device can be concurrently accessed via one or more of the APIs and via the user interface. This enables easy integration into larger laboratory setups. See the LabOne Programming Manual for further information. Using the APIs, the user has access to the same functionality that is available in the LabOne User Interface.

2.5.5. LabOne Software Start-up

This section describes the start-up of the LabOne User Interface which is used to control the SHFQC+ Instrument. If the LabOne software is not yet installed on the PC please follow the

instructions in [Software Installation](#). If the device is not yet connected please find more information in [Visibility and Connection](#).

The LabOne User Interface start-up link can be found under the Windows 10/11 Start Menu. As shown in [Figure 2.11](#), click on **Start Menu → Zurich Instruments LabOne**. This will open the User Interface in a new tab in your default web browser and start the LabOne Data Server and LabOne Web Server programs in the background. A detailed description of the software architecture is found in [LabOne Software Architecture](#).

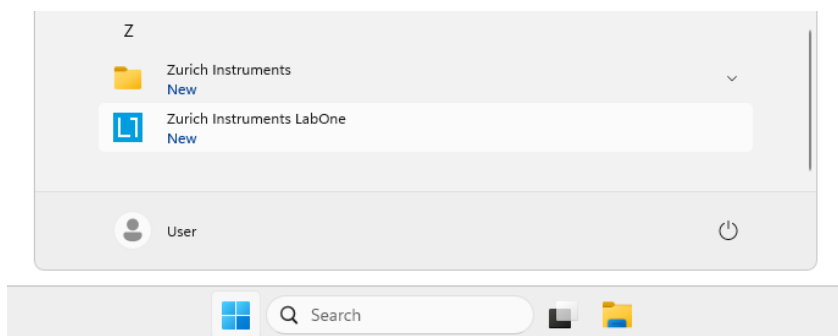


Figure 2.11: Link to the LabOne User Interface in the Windows 11 Start Menu

LabOne is an HTML5 browser-based program. This simply means that the user interface runs in a web browser and that a connection using a mobile device is also possible; simply specify the IP address (and port 8006) of the PC running the user interface.

Note

By creating a shortcut to Google Chrome on your desktop with the Target `path\to\chrome.exe -app=http://127.0.0.1:8006` set in Properties you can run the LabOne User Interface in Chrome in application mode, which improves the user experience by removing the unnecessary browser controls.

After starting LabOne, the Device Connection dialog [Figure 2.12](#) is shown to select the device for the session. The term "session" is used for an active connection between the user interface and the device. Such a session is defined by device settings and user interface settings. Several sessions can be started in parallel. The sessions run on a shared LabOne Web Server. A detailed description of the software architecture can be found in the [LabOne Software Architecture](#).



Figure 2.12: Device Connection dialog

The Device Connection dialog opens in the Basic view by default. In this view, all devices that are available for connection are represented by an icon with serial number and status information. If required, a button appears on the icon to perform a firmware upgrade. Otherwise, the device can be connected by a double click on the icon, or a click on the **Open** button at the bottom right of the dialog.

In some cases it's useful to switch to the Advanced view of the Device Connection dialog by clicking on the "Advanced" button. The Advanced view offers the possibility to select custom device and UI settings for the new session and gives further connectivity options that are particularly useful for multi-instrument setups.

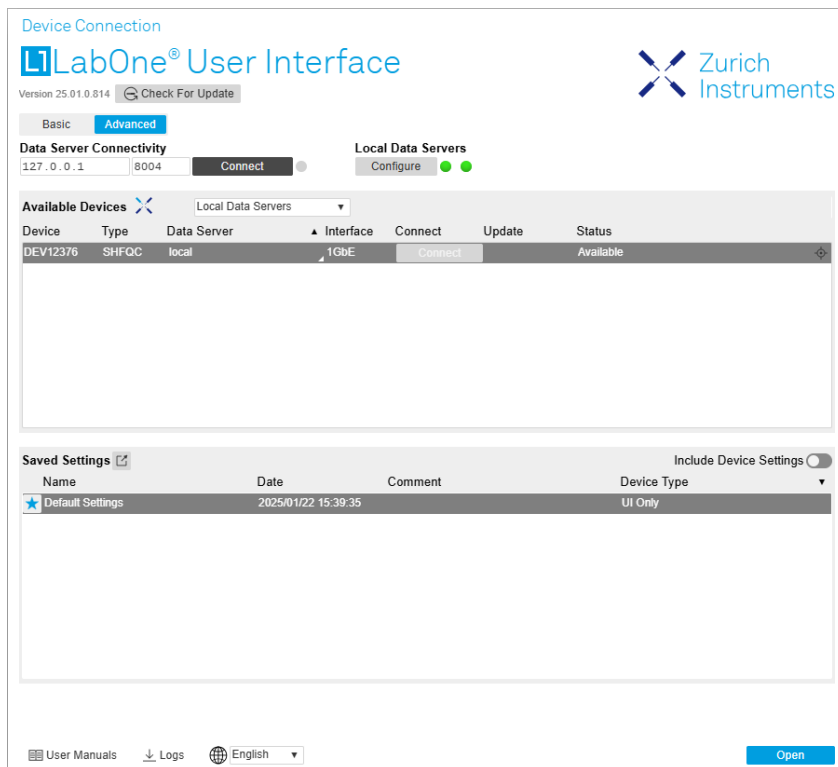


Figure 2.13: Device Connection dialog (Advanced view)

The Advanced view consists of three parts:

- Data Server Connectivity
- Available Devices
- Saved Settings

The Available Devices table has a display filter, usually set to **Default Data Server**, that is accessible by a drop-down menu in the header row of the table. When changing this to **Local Data Servers**, the Available Devices table will show only connections via the Data Server on the host PC and will contain all instruments directly connected to the host PC via USB or to the local network via 1GbE. When using the **All Data Servers** filter, connections via Data Servers running on other PCs in the network also become accessible. Once your instrument appears in the Available Devices table, perform the following steps to start a new session:

1. Select an instrument in the **Available Devices** table.
2. Select a setting file in the **Saved Settings** list unless you would like to use the Default Settings.
3. Start the session by clicking on **Open**

Note

By default, opening a new session will only load the UI settings (such as plot ranges), but not the device settings (such as signal amplitude) from the saved settings file. In order to include the device settings, enable the **Include Device Settings** checkbox. Note that this can affect existing sessions since the device settings are shared between them.

Note

In case devices from other Zurich Instruments series (UHF, HF2, MF, HDAWG, PQSC, GHF, or SHF) are used in parallel, the list in **Available Devices** section can contain those as well.

The following sections describe the functionality of the **Device Connection** dialog in detail.

2.5.6. Data Server Connectivity

The Device Connection dialog represents a Web Server. However, on start-up the Web Server is not yet connected to a LabOne Data Server. With the **Connect/Disconnect** button the connection to a Data Server can be opened and closed.

This functionality can usually be ignored when working with a single SHFQC+ Instrument and a single host computer. Data Server Connectivity is important for users operating their instruments from a remote PC, i.e., from a PC different to the PC on which the Data Server is running or for users working with multiple instruments. The Data Server Connectivity function then gives the freedom to connect the Web Server to one of several accessible Data Servers. This includes Data Servers running on remote computers, and also Data Servers running on an MF Series instrument.

In order to work with a UHF, HF2, HDAWG, PQSC, GHF, or SHF instrument remotely, proceed as follows. On the computer directly connected to the instrument (Computer 1) open a User Interface session and change the Connectivity setting in the Config tab to "From Everywhere". On the remote computer (Computer 2), open the Device Connection dialog by starting up the LabOne User Interface and then go to the Advanced view by clicking on **Advanced** on the top left of the dialog. Change the display filter from Default Data Server to All Data Servers by opening the drop-down menu in the header row of the Available Devices table. This will make the Instrument connected to Computer 1 visible in the list. Select the device and connect to the remote Data Server by clicking on **Connect**. Then start the User Interface as described above.

Note

When using the filter "All Data Servers", take great care to connect to the right instrument, especially in larger local networks. Always identify your instrument based on its serial number in the form DEV0000, which can be found on the instrument back panel.

2.5.7. Available Devices

The Available Devices table gives an overview of the visible devices. A device is ready for use if either marked free or connected. The first column of the list holds the **Enable** button controlling the connection between the device and a Data Server. This button is greyed out until a Data Server is connected to the LabOne Web Server using the **Connect** button. If a device is connected to a Data Server, no other Data Server running on another PC can access this device.

The second column indicates the serial number and the third column shows the instrument type. The fourth column shows the host name of the LabOne Data Server controlling the device. The next column shows the interface type. For SHFQC+ Instruments the interfaces USB or 1GbE are available and are listed if physically connected. The LabOne Data Server will scan for the available devices and interfaces every second. If a device has just been switched on or physically connected it may take up to 20 s before it becomes visible to the LabOne Data Server.

Table 2.5: Device Status Information

Available	The device is not in use by any LabOne Data Server and can be connected by clicking the Enable button. Alternatively, a session can also be started by clicking on the Open button, without prior connection.
In use by	The device is in use by a LabOne Data Server. As a consequence the device cannot be accessed by the specified interface. To access the device a disconnect is needed. The additional message "FW upgrade available" or "FW downgrade available" may also be displayed in this state.
Connected	The device is connected to a LabOne Data Server, either on the same PC (indicated as local) or on a remote PC (indicated by its IP address). The user can start a session to work with that device.
Device FW upgrade required	The firmware is out of date and must be upgraded before the device can be used. Please first upgrade the firmware by clicking on the Upgrade FW button as described in Software Update .
Device FW upgrade available. Please update	The firmware is out of date but the device can still be used. It is highly recommended to upgrade the firmware by clicking on the Upgrade FW button as described in Software Update .


Device FW downgrade available	The firmware of the device is newer than the version supplied with the installed LabOne software. This could be due to reverting to an earlier LabOne version. The device can still be used but it is also possible to downgrade to the older firmware version if for any reason this is necessary. Click on the Downgrade FW button to downgrade the firmware. It is strongly advised to upgrade LabOne instead of downgrading the firmware.
Device FW upgrade required. Please use USB firmware upgrade utility	The firmware of UHFLI/UHFQA is too old to be updated from the Device Connection dialog. Please first upgrade the firmware using the USB Firmware Upgrade Utility provided with LabOne software.
Device not yet ready	The device is visible and starting up. When the device is ready it will be flagged as Available.

2.5.8. Saved Settings

Settings files can contain both UI and device settings. UI settings control the structure of the LabOne User Interface, e.g. the position and ordering of opened tabs. Device settings specify the set-up of a device. The device settings persist on the device until the next power cycle or until overwritten by loading another settings file.

The columns are described in [Table 2.6](#). The table rows can be sorted by clicking on the column header that should be sorted. The default sorting is by time. Therefore, the most recent settings are found on top. Sorting by the favorite marker or setting file name may be useful as well.

Table 2.6: Column Descriptions

	Allows favorite settings files to be grouped together. By activating the stars adjacent to a settings file and clicking on the column heading, the chosen files will be grouped together at the top or bottom of the list accordingly. The favorite marker is saved to the settings file. When the LabOne user interface is started next time, the row will be marked as favorite again.
Name	The name of the settings file. In the file system, the file name has the extension .md.
Date	The date and time the settings file was last written.
Comment	Allows a comment to be stored in the settings file. By clicking on the comment field a text can be typed in which is subsequently stored in the settings file. This comment is useful to describe the specific conditions of a measurement.
Device Type	The instrument type with which this settings file was saved.

Special Settings Files

Certain file names have the prefix "last_session_". Such files are created automatically by the LabOne Web Server when a session is terminated either explicitly by the user, or under critical error conditions, and save the current UI and device settings. The prefix is prepended to the name of the most recently used settings file. This allows any unsaved changes to be recovered upon starting a new session.

If a user loads such a last session settings file the "last_session_" prefix will be cut away from the file name. Otherwise, there is a risk that an auto-save will overwrite a setting which was saved explicitly by the user.

The settings file with the name "Default Settings" contains the default UI settings. See button description in [Table 2.7](#).

Table 2.7: Button Descriptions

Open	The settings contained in the selected settings file will be loaded. The button "Include Device Settings" controls whether only UI settings are loaded, or if device settings are included.
Include Device Settings	Controls which part of the selected settings file is loaded upon clicking on Open. If enabled, both the device and the UI settings are loaded.

Auto Start	Skips the session dialog at start-up if selected device is available. The default UI settings will be loaded with unchanged device settings.
-------------------	--

Note

The user setting files are saved to an application-specific folder in the directory structure. The best way to manage these files is using the File Manager tab.

Note

The factory default UI settings can be customized by saving a file with the name "default_ui" in the Config tab once the LabOne session has been started and the desired UI setup has been established. To use factory defaults again, the "default_ui" file must be removed from the user setting directory using the File Manager tab.

Note

Double clicking on a device row in the Available Devices table is a quick way of starting the default LabOne UI. This action is equivalent to selecting the desired device and clicking the **Open** button.

Double clicking on a row in the Saved Settings table is a quick way of loading the LabOne UI with those UI settings and, depending on the "Include Device Settings" checkbox, device settings. This action is equivalent to selecting the desired settings file and clicking the **Open** button.

2.5.9. Tray Icon

When LabOne is started, a tray icon appears by default in the bottom right corner of the screen, as shown in the figure below. By right-clicking on the icon, a new web server session can be opened quickly, or the LabOne Web and Data Servers can be stopped by clicking on Exit. Double-clicking the icon also opens a new web server session, which is useful when setting up a connection to multiple instruments, for example.

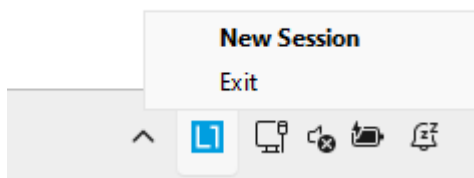


Figure 2.14: LabOne Tray Icon in Windows 11

2.5.10. Messages

The LabOne Web Server will show additional messages in case of a missing component or a failure condition. These messages display information about the failure condition. The following paragraphs list these messages and give more information on the user actions needed to resolve the problem.

Lost Connection to the LabOne Web Server

In this case the browser is no longer able to connect to the LabOne Web Server. This can happen if the Web Server and Data Server run on different PCs and a network connection is interrupted. As long as the Web Server is running and the session did not yet time out, it is possible to just attach to the existing session and continue. Thus, within about 15 seconds it is possible with **Retry** to recover the old session connection. The **Reload** button opens the Device Connection dialog shown in [Figure 2.12](#). The figure below shows an example of the Connection Lost dialog.

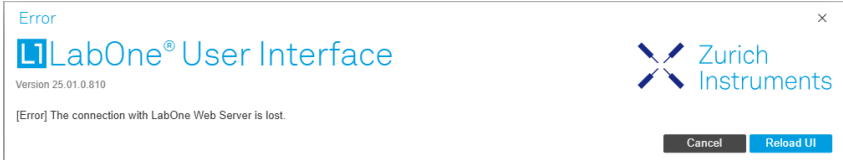


Figure 2.15: Dialog: Connection Lost

Reloading...

If a session error cannot be handled, the LabOne Web Server will restart to show a new Device Connection dialog as shown in [Figure 2.12](#). During the restart a window is displayed indicating that the LabOne User Interface will reload. If reloading does not happen the same effect can be triggered by pressing F5 on the keyboard. The figure below shows an example of this dialog.

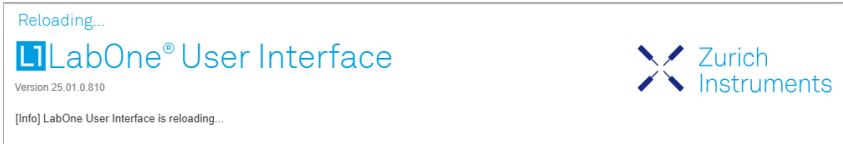


Figure 2.16: Dialog: Reloading

No Device Discovered

An empty "Available Devices" table means that no devices were discovered. This can mean that no LabOne Data Server is running, or that it is running but failed to detect any devices. The device may be switched off or the interface connection fails. For more information on the interface between device and PC see [Visibility and Connection](#). The figure below shows an example of this dialog.

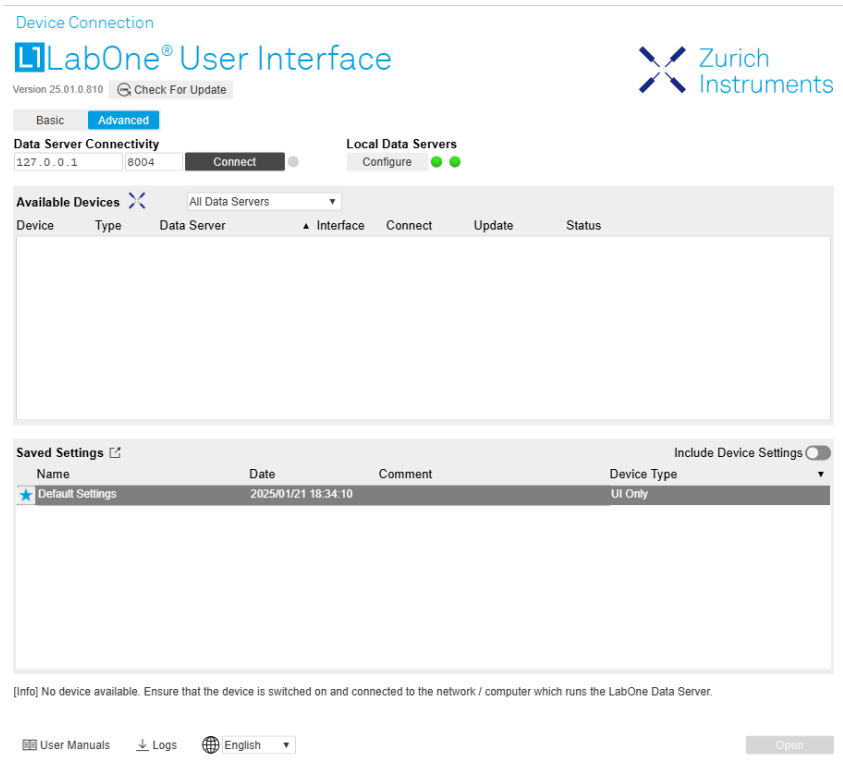


Figure 2.17: No Device Discovered

No Device Available

If all the devices in the "Available Devices" table are shown grayed, this indicates that they are either in use by another Data Server, or need a firmware upgrade. For firmware upgrade see [Software Update](#). If all the devices are in use, access is not possible until a connection is relinquished by another Data Server.

2.5.11. Visibility and Connection

There are several ways to connect the instrument to a host computer. The device can either be connected by Universal Serial Bus (USB) or by 1 Gbit/s Ethernet (1GbE). The USB connection is a point-to-point connection between the device and the PC on which the Data Server runs. The 1GbE connection can be a point-to-point configuration or an integration of the device into the local network (LAN). Depending on the network configuration and the installed network card, one or the other connectivity is better suited.

If an instrument is connected to a network, it can be accessed from multiple host computers. To manage the access to the instrument, there are two different connectivity states: visible and connected. It is important to distinguish if an instrument is just physically connected over 1GbE or actively controlled by the LabOne Data Server. In the first case the instrument is visible to the LabOne Data Server. In the second case the instrument is logically connected.

Connectivity Example shows some examples of possible configurations of computer-to-instrument connectivity.

- Data Server on PC 1 is connected to device 1 (USB) and device 2 (USB).
- Data Server on PC 2 is connected to device 4 (TCP/IP).
- Data Server on PC 3 is connected to device 5.
- The device 3 is free and visible to PC 1 and PC 2 over TCP/IP.
- Devices 2 and 4 are physically connected by TCP/IP and USB interface. Only one interface is logically connected to the Data Server.

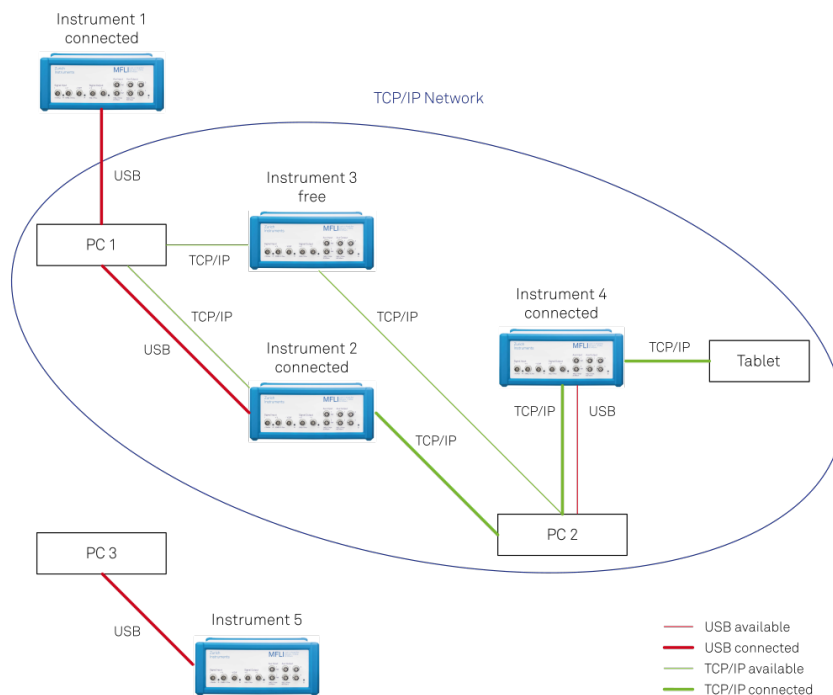


Figure 2.18: Connectivity Example

Visible Instruments

An instrument is visible if the Data Server can identify it. On a TCP/IP network, several PCs running a Data Server will detect the same instrument as visible, i.e., discover it. If a device is discovered, the LabOne Data Server can initiate a connection to access the instrument. Only a single Data Server can be connected to an instrument at a time.

Connected Instrument

Once connected to an instrument, the Data Server has exclusive access to that instrument. If another Data Server from another PC already has an active connection to the instrument, the instrument is still visible but cannot be connected.

Although a Data Server has exclusive access to a connected instrument, the Data Server can have multiple clients. Because of this, multiple browser and API sessions can access the instrument simultaneously.

2.5.12. USB Connectivity

To control the device over USB, connect the instrument with the supplied USB cable to the PC on which the LabOne Software is installed. The USB driver needed for controlling the instrument is included in the LabOne Installer package. Ensure that the instrument uses the latest firmware. The software will automatically use the USB interface for controlling the device if available. If the USB connection is not available, the 1GbE connection may be selected. It is possible to enforce or exclude a specific interface connection.

Note

To use the device exclusively over the USB interface, modify the shortcut of the LabOne User Interface and LabOne Data Server in the Windows Start menu. Right-click and go to Properties, then add the following command line argument to the Target LabOne User Interface:

```
--interface-usb true --interface-ip false
```

An instrument connected over USB can be automatically connected to the Data Server because there is only a single host PC to which the device interface is physically connected. [Table 2.8](#) provides an overview of the two settings.

Table 2.8: Settings auto-connect

Setting	Description
auto-connect = on	If a device is attached via a USB cable, a connection will be established automatically by the Data Server. This is the default behavior.
auto-connect = off	To disable automatic connection via USB, add the following command line argument when starting the Data Server: <code>--auto-connect=off</code> .

On Windows, both behaviors can be forced by right clicking the LabOne Data Server shortcut in the Start menu, selecting "Properties" and adding the text `--auto-connect=off` or `--auto-connect=on` to the Target field, see [Figure 2.19](#).

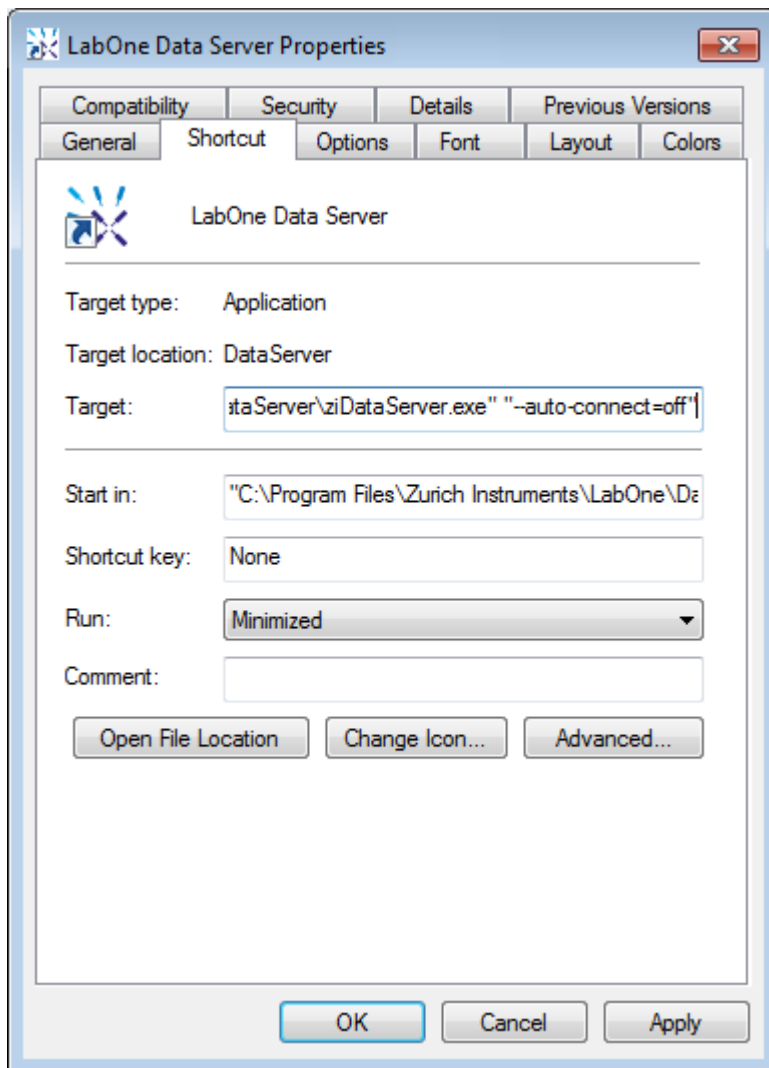


Figure 2.19: Setting auto-connect in Windows

2.5.13. 1GbE Connectivity

There are three methods for connecting to the device via 1GbE:

- Multicast DHCP
- Multicast point-to-point (P2P)
- Static Device IP

Multicast DHCP is the simplest and preferred connection method. Other connection methods can become necessary when using network configurations that conflict with local policies.

Multicast DHCP

The most straightforward TCP/IP connection method is to rely on a network configuration to recognize the instrument. When connecting the instrument to a local area network (LAN), the DHCP server will assign an IP address to the instrument like to any PC in the network. In case of restricted networks, the network administrator may be required to register the device on the network by means of the MAC address. The MAC address is indicated on the back panel of the instrument. The LabOne Data Server will detect the device in the network by means of a multicast.

If the network configuration does not support multicast, or if the host computer has other network cards installed, it is necessary to use a static IP setup as described below. The instrument is configured to accept the IP address from the DHCP server, or to fall back to the IP address **192.168.1.10** if it does not get the address from the DHCP server.

Requirements:

■

Network supports multicast

Multicast Point-to-Point

Setting up a point-to-point (P2P) network consisting only of the host computer and the instrument avoids problems related to special network policies. Since it is nonetheless necessary to stay connected to the internet, it is recommended to install two network cards in the computer, one of which is used for internet connectivity, the other can be used for connecting to the instrument. Alternatively, internet connectivity can be established via wireless LAN.

In such a P2P network the IP address of the host computer needs to be set to a static value, whereas the IP address of the device can be left dynamic.

1. Connect the 1GbE port of the network card that is dedicated for instrument connectivity directly to the 1GbE port of the instrument
2. Set this network card to static IP in TCP/IPv4 using the address **192.168.1.n**, where $n=[2..9]$ and the mask **255.255.255.0**. (On Windows go to **Control Panel → Internet Options → Network and Internet → Network and Sharing Center → Local Area Connection → Properties**).

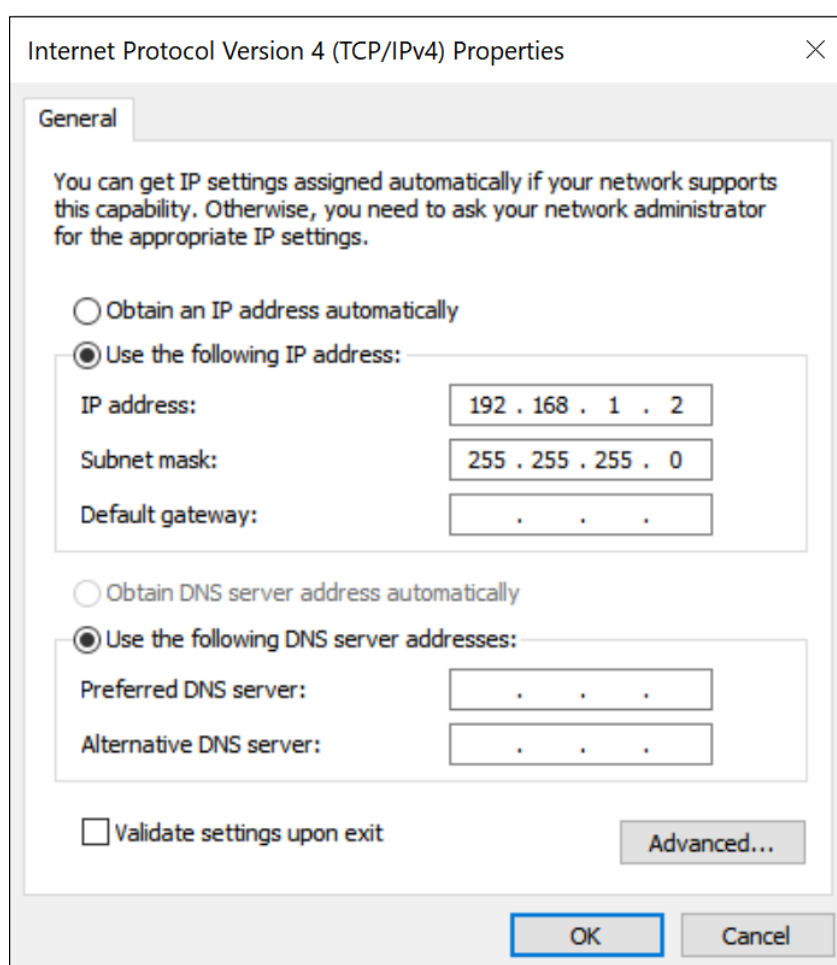


Figure 2.20: Static IP configuration for the host computer

3. Start up the LabOne User Interface normally. If your instrument does not show in the list of Available Devices, the reason may be that your network card does not support multicast. In that case, see [Static Device IP](#).

Requirements:

- Two network cards needed for additional connection to internet
- Network card of PC supports multicast
- Network card connected to the device must be in static IP4 configuration

Note

A power cycle of the instrument is required if it was previously connected to a network that provided an IP address to the instrument.

Note

Only IP v4 is currently supported. There is no support for IP v6.

Note

If the instrument is detected by LabOne but the connection can not be established, the reason can be the firewall blocking the connection. It is then recommended to change the P2P connection from Public to Private. On Windows this is achieved by turning on network discovery in the Private tab of the network's advanced sharing settings as shown in the figure below.

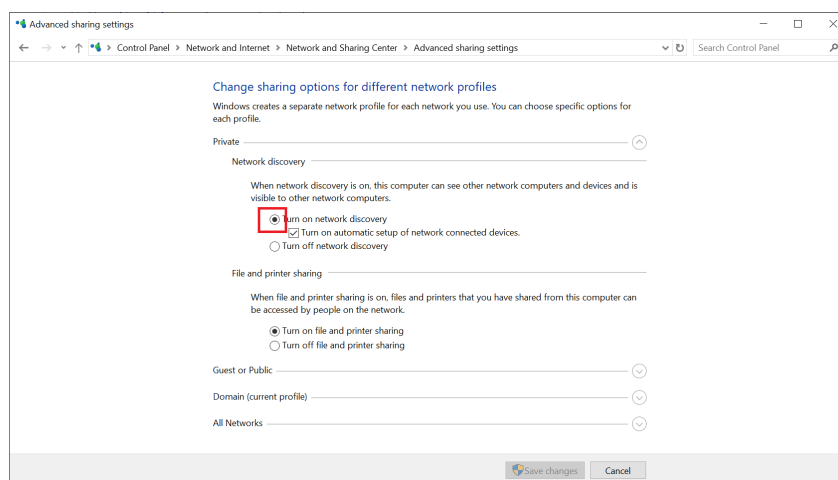


Figure 2.21: Turn on network discovery for Private P2P connection

Warning

Changing the IP settings of your network adapters manually can interfere with its later use, as it cannot be used anymore for network connectivity until it is configured again for dynamic IP.

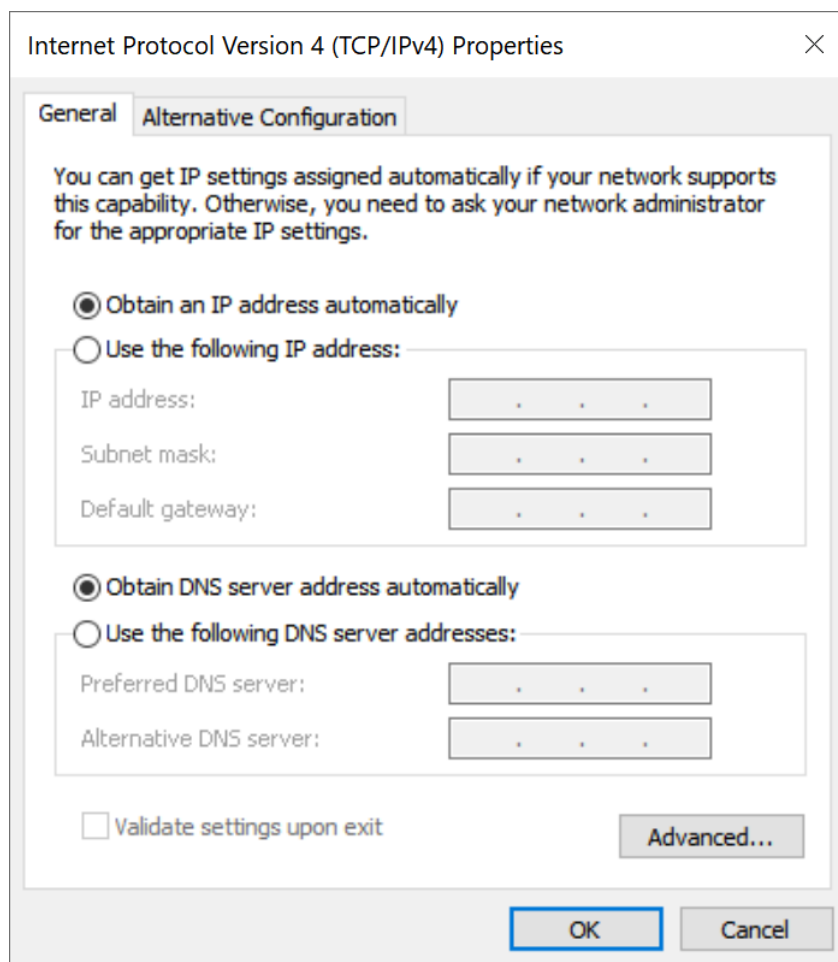


Figure 2.22: Dynamic IP configuration for the host computer

Static Device IP

Although it is highly recommended to use dynamic IP assignment method in the host network of the instrument, there may be cases where the user wants to assign a static IP to the instrument. For instance, when the host network only contains Ethernet switches and hubs but no Ethernet routers are included, there is no DHCP server to dynamically assign an IP to the instrument. It is still advised to add an Ethernet router to the network and benefit from dynamic IP assignment; however, if a router is not available, the instrument can be configured to work with a static IP.

Note that the static IP assigned to the instrument must be within the same range of the IP assigned to the host computer. Whether the host computer's IP is assigned statically or by a fallback mechanism, one can find this IP by running the command `ipconfig` or `ipconfig/all` in the operating system's terminal. As an example, Figure 2.23 shows the outcome of running `ipconfig` in the terminal.

```
Ethernet adapter Ethernet 4:

Connection-specific DNS Suffix . : 
Link-local IPv6 Address . . . . . : fe80::f3ad:19ae:ffd9:f8ef%17
Autoconfiguration IPv4 Address. . . : 169.254.16.57
Subnet Mask . . . . . : 255.255.0.0
Default Gateway . . . . . :
```

Figure 2.23: IP and subnet mask of host computer

It shows the network adapter of the host computer can be reached via the IP **169.254.16.57** and it uses a subnet mask of **255.255.0.0**. To make sure that the instrument is visible to this computer, one needs to assign a static IP of the form **169.254.x.x** and the same subnet mask to the instrument. To do so, the user should follow the instructions below.

1. Attach the instrument using an Ethernet cable to the network where the user's computer is hosted.
2. Attach the instrument via a USB cable to the host computer and switch it on.
3. Open the LabOne user interface (UI) and connect to the instrument via USB.
4. Open the "Device" tab of the LabOne UI and locate the "Communication" section as shown in [Configuration of static IP in LabOne UI](#).
5. Write down the desired static IP, e.g. **169.254.16.20**, into the numeric field "IPv4 Address".
6. Add the same subnet mask as the host computer, e.g. **255.255.0.0** to the numeric field "IPv4 Mask".
7. You can leave the field "Gateway" as **0.0.0.0** or change to be similar to the IP address but ending with 1, e.g. **169.254.16.1**.
8. Enable the radio button for "Static IP".
9. Press the button "Program" to save the new settings to the instruments.
10. Power cycle the instrument and remove the USB cable. The instrument should be visible to LabOne via Ethernet connection.

The screenshot shows the 'Communication' section of the LabOne UI. Under 'Current Configuration', the 'Interface' is set to 'USB', the 'MAC Address' is '80:2F:DE:00:0D:15', and the 'IPv4 Address' is '169.254.16.20'. Under 'Network Configuration 1GbE', the 'Static IP' toggle is turned on, and the 'IPv4 Address' is '169.254.16.20', the 'IPv4 Mask' is '255.255.0.0', and the 'Gateway' is '0.0.0.0'. A 'Save' button is at the bottom.

Figure 2.24: Configuration of static IP in LabOne UI

To make sure the IP assignment is done properly, one can use the command **ping** to check if the instrument can be reached through the network using its IP address. [Figure 2.25](#) shows the outcome of **ping** when the instrument is visible via the IP **169.254.16.20**.

```
C:\> ping 169.254.16.20

Pinging 169.254.16.20 with 32 bytes of data:
Reply from 169.254.16.20: bytes=32 time<1ms TTL=64
Reply from 169.254.16.20: bytes=32 time<1ms TTL=64
Reply from 169.254.16.20: bytes=32 time<1ms TTL=64
Reply from 169.254.16.20: bytes=32 time<1ms TTL=64

Ping statistics for 169.254.16.20:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

Figure 2.25: Instrument visible through pinging

If set properly according to the instructions above, the instrument will use the same static IP configurations after each power cycle.

Fallback Device IP

When configured to a dynamic address, but no DHCP server is present in the network, e.g., device connected directly to a PC, the instrument falls back on an IP address in the local link IP range that is **169.254.x.x**. If the host computer has also an IP address within the same range, the instrument becomes visible to the LabOne data server running on the host computer. This way, there is no need to go through the process described above to assign a static IP to the instrument.

2.6. Software Update

2.6.1. Overview

It is recommended to regularly update the LabOne software on the SHFQC+ Instrument to the latest version. In case the Instrument has access to the internet, this is a very simple task and can be done with a single click in the software itself, as shown in [Updating LabOne using Automatic Update Check](#). If you use one of the LabOne APIs with a separate installer, don't forget to update this part of the software, too.

2.6.2. Updating LabOne using Automatic Update Check

Updating the software is done in two steps. First, LabOne is updated on the PC by downloading and installing the LabOne software from the Zurich Instruments downloads page, as shown in [Software Installation](#). Second, the instrument firmware needs to be updated from the Device Connection dialog after starting up LabOne. This is shown in [Updating the Instrument Firmware](#). In case "Periodically check for updates" has been enabled during the LabOne installation and LabOne has access to the internet, a notification will appear on the Device Connection dialog whenever a new version of the software is available for download. This setting can later be changed in the Config tab of the LabOne user interface. In case automatic update check is disabled, the user can manually check for updates at any time by clicking on the button [Check For Update](#) in the Device Connection dialog. In case an update is found, clicking on the button "Update Available" shown in [Figure 2.26](#) will start a download of the latest LabOne installer for Windows or Linux, see [Figure 2.27](#). After download, proceed as explained in [Software Installation](#) to update LabOne.

Device Connection

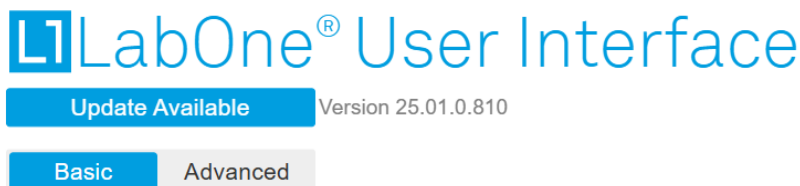


Figure 2.26: Device Connection dialog: LabOne update available

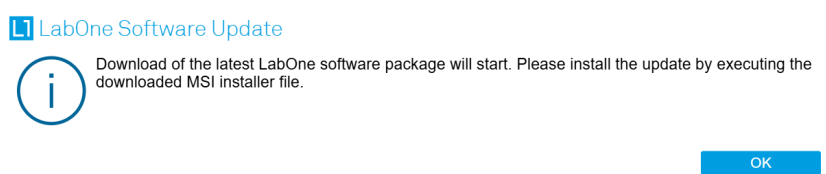


Figure 2.27: Download LabOne MSI using Automatic Update Check feature

2.6.3. Updating the Instrument Firmware

The LabOne software consists of both software that runs on your PC and software that runs on the instrument. In order to distinguish between the two, the latter will be called firmware for the rest of this document. When upgrading to a new software release, it's also necessary to update the instrument firmware.

If the firmware needs an update, this is indicated in the Device Connection dialog of the LabOne user interface under Windows.

In the Basic view of the dialog, there will be a button "Upgrade FW" appearing together with the instrument icon as shown in Figure 2.28. In the Advanced view, there will be a link "Upgrade FW" in the Update column of the Available Devices table. Click on **Upgrade FW** to open the firmware update start-up dialog shown in Figure 2.29. The firmware upgrade takes approximately 2 minutes.

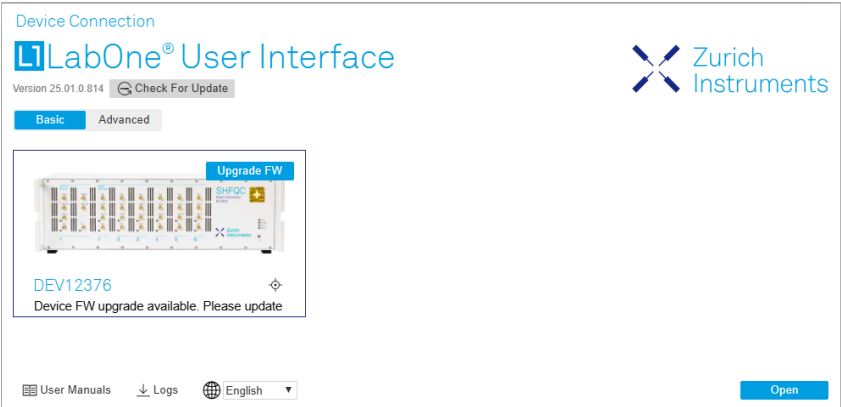


Figure 2.28: Device Connection dialog with available firmware update

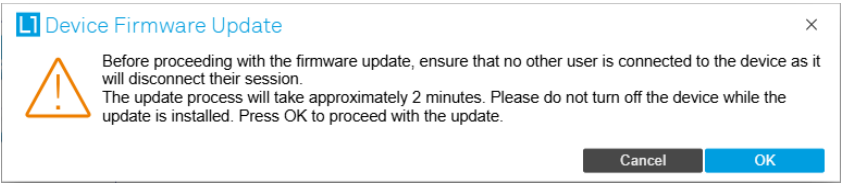


Figure 2.29: Device Firmware Update start-up dialog

Important

Do not disconnect the USB or 1GbE cable to the Instrument or power-cycle the Instrument during a firmware update.

If you encounter any issues while upgrading the instrument firmware, please contact Zurich Instruments at support@zhinst.com.

2.7. Troubleshooting

This section aims to help the user solve and avoid problems while using the software and operating the instrument.

2.7.1. Common Problems

Your SHFQC+ Instrument is an advanced piece of laboratory equipment which has many more features and capabilities than a traditional signal generator. In order to benefit from these, the user needs access to a large number of settings in the API or the LabOne User Interface. The complexity of the settings might overwhelm a first-time user, and even expert users can get surprised by certain combinations of settings. This section provides an easy-to-follow checklist to solve the most common mishaps.

Table 2.9: Common Problems

Problem	Check item
The software cannot be installed or uninstalled	Please verify you have administrator/root rights.
The software cannot be updated	Please use the Modify option in Windows Apps & Features functionality. In the software installer select Repair, then uninstall the old software version, and install the new version.
The Instrument does not turn on	Please verify the power supply connection and inspect the fuse. The fuse holder is integrated in the power connector on the back panel of the instrument.

Problem	Check item
The Instrument performs close to specification, but higher performance is expected	After 2 years since the last calibration, a few analog parameters are subject to drift. This may cause inaccurate measurements. Zurich Instruments recommends re-calibration of the Instrument every 2 years.
The Instrument measurements are unpredictable	Please check the Status Tab to see if there is any active warning (red flag), or if one has occurred in the past (yellow flag).
The Instrument does not generate any output signal	Verify that the signal output switch of the right signal output channel has been activated in the Output tab.
The LabOne User Interface does not start	Verify that the LabOne Data Server (ziDataServer.exe) and the LabOne Web Server (ziWebServer.exe) are running via the Windows Task Manager. The Data Server should be started automatically by ziService.exe and the Web Server should be started upon clicking "Zurich Instruments LabOne" in the Windows Start Menu. If both are running, but clicking the Start Menu does not open a new User Interface session in a new tab of your default browser then try to create a new session manually by entering 127.0.0.1:8006 in the address bar of your browser.
The user interface does not start or starts but remains idle	Verify that the Data Server has been started and is running on your host computer.
The user interface is slow and the web browser process consumes a lot of CPU power	Make sure that the hardware acceleration is enabled for the web browser that is used for LabOne. For the Windows operating system, the hardware acceleration can be enabled in Control Panel → Display → Screen Resolution . Go to Advanced Settings and then Trouble Shoot. In case you use a NVIDIA graphics card, you have to use the NVIDIA control panel. Go to Manage 3D Settings, then Program Settings and select the program that you want to customize.

2.7.2. Location of the Log Files

The most recent log files of the LabOne Web and Data Server programs are most easily accessed by clicking on **Logs** in the **LabOne Device Connection dialog** of the user interface. The Device Connection dialog opens on software start-up or upon clicking on **Session Manager** in the Config tab of the user interface.

The location of the Web and Data Server log files on disk are given in the sections below.

Windows

The Web and Data Server log files on Windows can be found in the following directories.

- ─ LabOne Data Server (**ziDataServer.exe**):
C:\Windows\ServiceProfiles\LocalService\AppData\Local\Temp\Zurich Instruments\LabOne\ziDataServerLog
- ─ LabOne Web Server (**ziWebServer.exe**):
C:\Users\[USER]\AppData\Local\Temp\Zurich Instruments\LabOne\ziWebServerLog

Note

The `C:\Users\[USER]\AppData` folder is hidden by default under Windows. A quick way of accessing it is to enter `%AppData%\..` in the address bar of the Windows File Explorer.

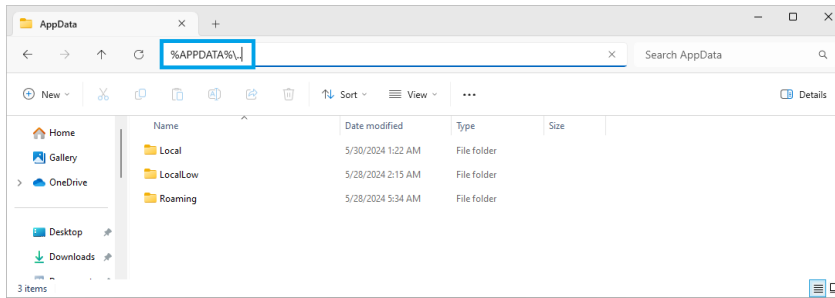


Figure 2.30: Using the

Linux and macOS

The Web and Data Server log files on Linux or macOS can be found in the following directories.

- LabOne Data Server (**ziDataServer**):
/tmp/ziDataServerLog_[USER]
- LabOne Web Server (**ziWebServer**):
/tmp/ziWebServerLog_[USER]

2.7.3. Prevent web browsers from sleep mode

It often occurs that an experiment requires a long-time signal acquisition; therefore, the setup including the measurement instrument and LabOne software are left unattended. By default, many web browsers go to a sleep mode after a certain idle time which results in the loss of acquired data when using the web-based user interface of LabOne for measurement. Although it is recommended to take advantage of LabOne APIs in these situations to automate the measurement process and avoid using web browsers for data recording, it is still possible to adjust the browser settings to prevent it from entering the sleep mode. Below, you will find how to modify the settings of your preferred browser to ensure a long-run data acquisition can be implemented properly.

Edge

1. Open **Settings** by typing `edge://settings` in the address bar
2. Select **System** from the icon bar.
3. Find the **Never put these sites to sleep** section of the **Optimized Performance** tab.
4. Add the IP address and the port of LabOne Webserver, e.g., `127.0.0.1:8006` or `192.168.73.98:80` to the list.

Chrome

1. While LabOne is running, open a tab in Chrome and type `chrome://discards` in the address bar.
2. In the shown table listing all the open tabs, find LabOne and disable its **Auto Discardable** feature.
3. This option avoids discarding and refreshing the LabOne tab as long as it is open. To disable this feature permanently, you can use an extension from the Chrome Webstore.

Firefox

1. Open **Advanced Preferences** by typing `about:config` in the address bar.
2. Look for `browser.tabs.unloadOnLowMemory` in the search bar.
3. Change it to **false** if it is **true**.

Opera

1. Open **Settings** by typing **opera://settings** in the address bar.
2. Locate the **User Interface** section in the **Advanced** view.
3. Disable the **Snooze inactive tabs to save memory** option and restart Opera.

Safari

1. Open **Debug** menu.
2. Go to **Miscellaneous Flags**.
3. Disable **Hidden Page Timer Throttling**.

3. Functional Overview

This chapter provides the overview of the features provided by the SHFQC Instrument. Unless explicitly stated otherwise, all contents of the manual apply to both the original SHFQC as well as the SHFQC+. The first section contains the description of the functional diagram and the hardware and software feature list. The next section details the front panel and the back panel of the measurement instrument. The following section provides product selection and ordering support.

3.1. Features

The **SHFQC+ Instrument** consists of one Quantum Analyzer readout channel, and 6 Signal Generator control channels. Both types of channels consists of several interface units processing analog signals (dark blue color), and several internal units that process digital data (light blue color). On the Signal Generator channels, the internal units that process digital data are called Digital Signal Units (or DSUs). The interface units that connect to the front panel are depicted on the left-hand side and the units at the back panel are depicted on the right-hand side of the Figure. Arrows between the panels and the interface units indicate selected physical connections and data flow, whereas double arrows indicate complex-valued information processing. Information indicated in orange is linked to options that can be either ordered at purchase or upgraded later. The [ordering guide](#) details the available upgrade options for the SHFQC+ Qubit Controller and whether the option can be upgraded directly in the field.

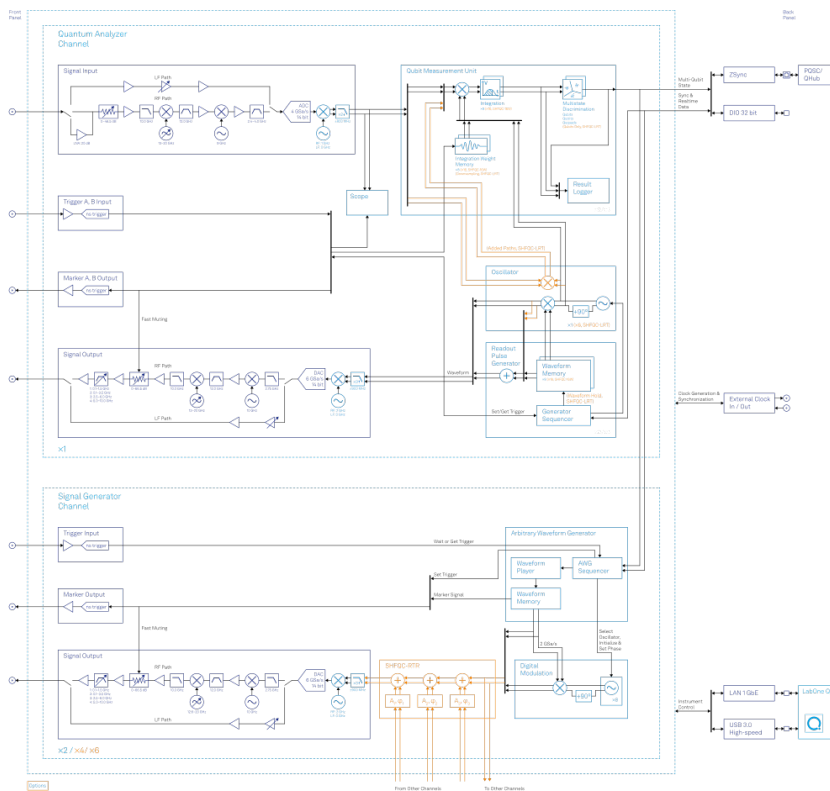


Figure 3.1: SHFQC+ Instrument functional diagram

3.1.1. Quantum Analyzer Channel

The Quantum Analyzer readout channel has signal generation (**Readout Pulse Generator**, **Oscillator**) and signal analysis (**Qubit Measurement Unit**) functionality, as well as a **Monitor Scope**. The digital, complex-valued signal from the signal generation is up-converted to microwave frequencies in the analog domain using the **Quantum Analyzer Signal Output Module**. After passing through the device under test, the analog signal is first down-converted and then digitized in the **Quantum Analyzer Signal Input** before the complex-valued signal is being analyzed in the **Qubit Measurement Unit**.

The Quantum Analyzer channel can be configured in two modes:

Qubit Readout

In the **Qubit Readout** mode, the **Readout Pulse Generator** outputs the sum of up to 16 individual, user-defined arbitrary waveforms that are stored in separate **Waveform Memory** blocks, controlled by the **Generator Sequencer**. In the **Qubit Measurement Unit**, the signal is then first integrated using up to 16 user-defined, complex-valued **Integration Weight Memories**. The results of the integration are then discriminated between different qubit states, and the results are forwarded to either the Signal Generator channels or other instruments in real time using either the **32-bit DIO** or the **ZSync** links.

Spectroscopy

In the **Spectroscopy** mode, the **Readout Pulse Generator** controls an **Oscillator**, hence a single frequency microwave signal is sent to the experiment. The **Qubit Measurement Unit** then correlates this signal with the original oscillator signal and displays transmission data in the Sweeper module.

Super-high-frequency Signal Inputs

- Low-noise SHF Inputs, 0.5 - 8.5 GHz frequency range with 1 GHz bandwidth when using the RF path, DC - 800 MHz when using the LF path
- Broadband double super-heterodyne frequency down-conversion, free from mixer calibration
- Calibrated (Input) Power Range, selectable from -50 dBm to 10 dBm when using the RF path, from -30 dBm to +10 dBm when using the LF path

Super-high-frequency Signal Outputs

- Low-noise SHF Outputs, 0.5 - 8.5 GHz frequency range with 1 GHz bandwidth when using the RF path, DC - 800 MHz when using the LF path
- Broadband double super-heterodyne frequency up-conversion, free from mixer calibration
- Calibrated (Output) Power Range, selectable from -30 dBm to 10 dBm when using the RF path, from -30 dBm to +5 dBm when using the LF path

Readout Pulse Generator

- Arbitrary waveform capability
- Advanced sequencing
 - looping, branching
 - advanced trigger control
- Up to 16 freely configurable waveform memory blocks of 4 kSa (total 64 kSa)

Qubit Measurement Unit

- Up to 16 complex integrators with programmable Integration Weight memory (this requires the SHFQC-16W option.)
- Multistate Discrimination for up to 4 states per qubit
- Result Logger with real-time averaging and data logging

Scope

- Real-time scope to view Input Signals
- Displays complex signals in time and frequency domain

Available Options

- SHFQC-16W: Readout up to 16 qubits instead of 8 qubits
- SHFQC-LRT: Integration length up to 32 μ s instead of 2 μ s for up to 6 qubits in parallel

3.1.2. Signal Generator Channel

Each of the 6 control channels has an arbitrary waveform generator **AWG** and **Modulation** functionality. The digital, complex-valued signal from the Digital Signal Unit is up-converted to microwave frequencies in the analog domain using the **Signal Generator Signal Output** Module.

Super-high-frequency Signal Outputs

- Low-noise SHF Outputs, DC - 8.5 GHz frequency range, 1 GHz modulation bandwidth
- Broadband double super-heterodyne frequency up-conversion
- Calibrated (Output) Power Range, selectable from -30 dBm to 10 dBm when using the RF path and from -30 dBm to 5 dBm when using the LF path

Advanced Pulse Sequencer

- Arbitrary Waveform Generator capability
- Advanced sequencing
 - looping, branching
 - command table
 - advanced trigger control
- Digital modulation

Available Options

- SHFQC-LRT: Signals from up to 3 additional Digital Signal Units can be routed and added to any Output

3.1.3. Shared Resources

The Quantum Analyzer and Signal Generator channels share several functionalities that are used for communication (**32-bit DIO**, **ZSync**) and inter-channel synchronization.

Hardware Trigger Engine

- shared between all channels and modes
- 2 Marker Outputs and Trigger Inputs of the Quantum Analyzer channel
- 1 Marker Output and Trigger Input for each Signal Generator channel

High-speed Connectivity

- SMA connectors on front and back panel for triggers, signals and external clock
- USB 3.0 high-speed host interface
- Maintenance USB connection
- LAN/Ethernet 1 Gbit/s controller interface
- DIO: 32-bit digital input-output port
- ZSync connection for clock synchronization and fast data transfer
- Clock input/output connectors (10/100 MHz)

Software Features

- LabOne Graphic User Interface: Web-based with multi-instrument control
- [Zurich Instruments LabOne Q](#) software for high-level programming of quantum computing experiments.
- Data server with multi-client support
- [LabOne APIs](#), including Python, C, LabVIEW, MATLAB, .NET

- Turnkey software and firmware features for fast system tune-up

3.2. Front Panel Tour

The front panel SMA connectors and control LEDs are arranged as shown in [Figure 3.2](#) and listed in [Table 3.1](#).

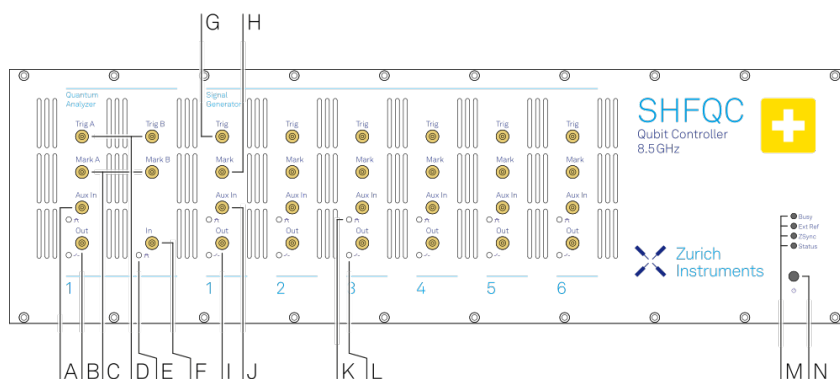



Figure 3.2: SHFQC+ Qubit Controller 8.5 GHz front panel

Table 3.1: SHFQC+ Qubit Controller front panel description

Position	Label / Name	Description
A	Aux In	analog Auxiliary Input, max. 10 V
B	Out	single-ended waveform Quantum Analyzer Signal Output, 0.5-8.5 GHz, max. 10 dBm
C	Mark	Quantum Analyzer TTL Marker Outputs A and B
D	Trig	Quantum Analyzer TTL Trigger Inputs A and B
E	Input Signal Over	<div>off</div> <div>power of input signal within the input power range</div> <div>red</div> <div>power of input signal exceeds the input power range</div>
F	In	single-ended waveform Quantum Analyzer Signal Input, DC-8.5 GHz, max. 10 dBm
G	Trig	Signal Generator TTL Trigger Input
H	Mark	Signal Generator TTL Marker Output
I	Out	single-ended waveform Signal Generator Signal Output, DC-8.5 GHz, max. 10 dBm
J	Aux In	analog Auxiliary Input, max. 10 V
K	Aux In Over	unused
L	Output On	<div>off</div> <div>Output disabled</div> <div>blue</div> <div>Output enabled</div>
M	multicolor LEDs	<div>off</div> <div>Instrument off or uninitialized</div> <div>blink</div> <div>all LEDs blink for 5 seconds → indicator used by the Identify Device functionality</div>
	Busy	unused

Position	Label / Name	Description
	Ext Ref	<p>off External Reference Signal not present/detected</p> <p>blue External Reference Signal is present and locked on to</p> <p>yellow External Reference Signal present, but not locked on to</p> <p>red External Reference Signal present, but lock failed</p>
	ZSync	<p>off no connection</p> <p>blue <ul style="list-style-type: none"> — steady: ZSync fully connected AND synchronized — blinking: ZSync synchronized but not yet fully connected </p> <p>yellow ZSync plugged in, but not connected</p> <p>red ZSync interface error</p>
	Status	<p>off Instrument off or uninitialized</p> <p>blue Instrument is initialized and has no warnings or errors</p> <p>yellow Instrument has warnings</p> <p>red Instrument has errors</p>
N	 Soft power button	<p>Power button with incorporated status LED</p> <p>off Instrument off and disconnected from mains power</p> <p>blue <ul style="list-style-type: none"> — flashing rapidly (>1/sec): Firmware is starting — flashing slow (<1/sec): Firmware ready, waiting for connection — constant: Instrument ready and active connection over USB or Ethernet </p> <p>red <ul style="list-style-type: none"> — breathing: Instrument off but connected to mains power → safe to power off using the rear panel switch, or restart using the soft power button — flashing: Instrument booting up — constant: Fatal error occurred </p>

3.3. Back Panel Tour

The back panel is the main interface for power, control, service and connectivity to other ZI instruments. Please refer to [Figure 3.3](#) and [Table 3.2](#) for the detailed description of the items.

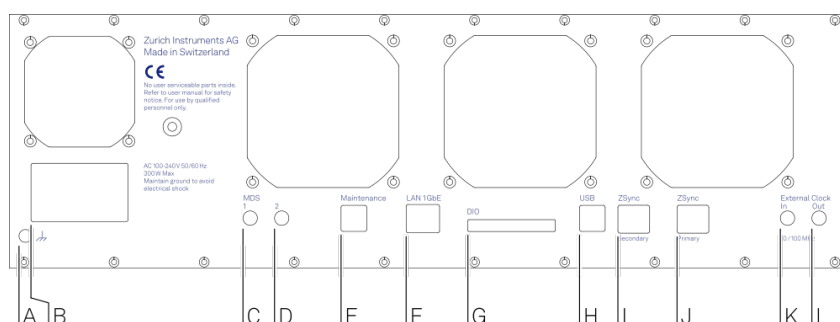



Figure 3.3: SHFQC+ back panel

Table 3.2: SHFQC+ Instrument back panel description

Position	Label / Name	Description
A	 Earth ground	4 mm banana jack connector for earth ground, electrically connected to the chassis and the earth pin of the power inlet
B	AC 100 - 240 V	Power inlet, fuse holder, and power switch
C	MDS 1	Unused
D	MDS 2	Unused
E	Maintenance	Maintenance USB port. Only for instrument maintenance, not for regular operation. Please use the USB port (see position H) instead. Some of the instruments have this port labeled as USB 1.
F	LAN 1GbE	1 Gbit LAN connector for instrument control
G	DIO 32bit	32-bit digital input/output (DIO) connector
H	USB	Universal Serial Bus (USB) 3.0 port for instrument control and data acquisition. Some of the instruments have this port labeled as USB 2.
I	ZSync Secondary	Unused Attention: This is not an Ethernet plug, connection to an Ethernet network might damage the Instrument.
J	ZSync Primary	Primary inter-instrument synchronization bus connector Attention: This is not an Ethernet plug, connection to an Ethernet network might damage the instrument.
K	External Clk In	External Reference Clock Input (10 MHz/100 MHz) for synchronization with other instruments
L	External Clk Out	External Reference Clock Output (10 MHz/100 MHz) for synchronization with other instruments

3.4. Ordering Guide

Table 3.3 provides an overview of the available SHFQC+ products and options. Upgradeable features are options that can be purchased anytime without the need to send the Instrument back to Zurich Instruments.

Table 3.3: SHFQC+ Instrument product codes for ordering

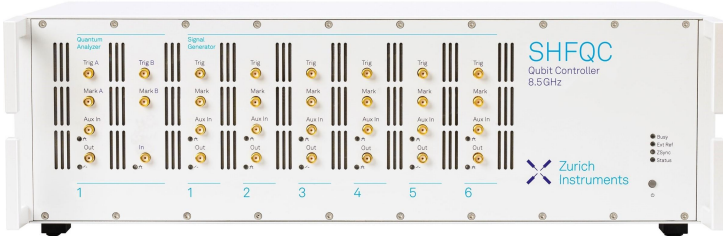
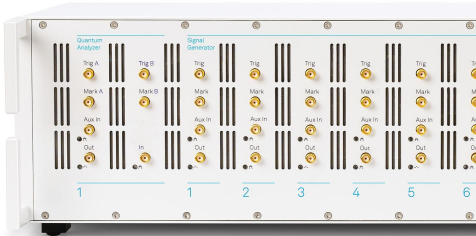
Product code	Product name	Description	Field upgrade possible
SHFQC2+	SHFQC+ Qubit Controller 2-Channel Configuration	Base instrument with 2 SG channels enabled	-
SHFQC4+	SHFQC+ Qubit Controller 4-Channel Configuration	Base instrument with 4 SG channels enabled	-
SHFQC6+	SHFQC+ Qubit Controller 6-Channel Configuration	Base instrument with 6 SG channels enabled	-
SHFQC-2T4	SHFQC2+ to SHFQC4+ Upgrade Option	Software Upgrades the SHFQC2+ to a SHFQC4+ or the SHFQC2 to a SHFQC4	yes
SHFQC-2T6	SHFQC2+ to SHFQC6+ Upgrade Option	Software Upgrades the SHFQC2+ to a SHFQC6+ or the SHFQC2 to a SHFQC6	yes
SHFQC-4T6	SHFQC4+ to SHFQC6+ Upgrade Option	Software Upgrades the SHFQC4+ to a SHFQC6+ or the SHFQC4 to a SHFQC6	yes
SHFQC-16W	SHFQC-16W Integration Weights Extension Option	Option for all variants of the SHFQC+ and SHFQC	yes

Product code	Product name	Description	Field upgrade possible
SHFQC-RTR	SHFQC+ Output Router and Adder Option	Option for all variants of the SHFQC+ and SHFQC	yes
SHFQC-LRT	SHFQC+ Long Readout Time Option	Option for all variants of the SHFQC+ and SHFQC	yes

Table 3.4: Product selector SHFQC+

Channel	Feature	SHFQC2+ / SHFQC4+ / SHFQC6+	SHFQCX+ + SHFQC-16W	SHFQCX+ + SHFQC-RTR	SHFQCX+ + SHFQC-LRT
Quantum Analyzer	Readout Channels	1 input and 1 output			
	Frequency range	DC - 8.5 GHz			
	Vertical resolution Input/Output	14 bit			
	Total number of Markers/Triggers	2/2			
	Number of integration weights	8	16	8	8
	Number of readout waveforms	8	16	8	8
	Oscillators	1	1	1	6
	Sequencing	yes			
Signal Generator	Control Channels	2/4/6			
	Frequency range	DC - 8.5 GHz			
	Number of independent RF control bands (>1 GHz)	1/2/3			
	Vertical resolution Output	14 bit			
	Total number of Markers/Triggers	2/2 (SHFQC2+) 4/4 (SHFQC4+) 6/6 (SHFQC6+)			
	Digital oscillators per channel	8			
	Digital signal units	2/4/6	2/4/6	6	2/4/6
	Digital signal routing and adding	not applicable	not applicable	up to 3 digital signals can be routed and added to any output channel	not applicable
	Pulse-level Sequencing	yes			
Shared resources	ZSync capability	yes			
	USB 3.0	yes			
	LAN 1 Gbit/s	yes			

Table 3.5: Differences between SHFQC and SHFQC+

Category	SHFQC	SHFQC+
Description	Base instrument	Improved phase noise Improved output noise Fast output muting functionality
External differences	Base instrument	Holographic "+" sticker on front panel
Product image		
Upgrading from SHFQC to SHFQC+	Contact us to discuss the possibilities for your instruments!	-

4. Tutorials

The tutorials in this chapter have been created to allow users to become more familiar with the operation of the SHFQC+ Qubit Controller. The first tutorials are useful at an early stage to learn how to use the General User Interface to configure the Signal Generator control channels. Later tutorials focus on controlling the instrument using our basic Python API and introducing the Quantum Analyzer readout channel — including tips and tricks for optimally programming the instrument for typical superconducting qubits applications.

In order to successfully carry out the tutorials it is assumed that users have certain laboratory equipment and basic equipment handling knowledge.

[LabOne Q](#) is the recommended control software to operate the SHFQC+ for Quantum Technology applications.

Note

In the tutorials, we use both the General User Interface and the Python API to control the instrument.

Note

For all tutorials, you must have LabOne installed as described in the chapter [Getting Started](#).

Note

This chapter is constantly being upgraded and new documentation is added. For the latest version of the documentation, please always refer to the [online documentation](#).

4.1. Signal Generator Tutorials

The tutorials in this subchapter have been created to allow users to become more familiar with the Signal Generator Control Channels of the SHFQC+.

Note

In the tutorials, we use both the General User Interface and the Python API to control the instrument.

Note

For all tutorials, you must have LabOne installed as described in the chapter [Getting Started](#).

Note

This chapter is constantly being upgraded and new documentation is added. For the latest version of the documentation, please always refer to the [online documentation](#).

4.1.1. Basic Sine Generation

Note

This tutorial is applicable to all SHFQC+ Instruments.

Goals and Requirements

The goal of this tutorial is to demonstrate basic sine generation with the Signal Generator channels of the SHFQC+. We demonstrate how to configure the sine generator to produce a single frequency component at the desired frequency in the range 0 GHz to 8.5 GHz. In order to visualize the multi-channel signals, an oscilloscope with sufficient bandwidth and channel number is required. This can be an external scope, or the scope of the SHFQC+, for which a loopback configuration is needed with the Quantum Analyzer channel.

Preparation

Connect the cables as illustrated below. Make sure that the instrument is powered on and connected by Ethernet to your local area network (LAN) where the host computer resides. After starting LabOne, the default web browser opens with the LabOne graphical user interface.

Note

The instrument can also be connected via the USB interface, which can be simpler for a first test. As a final configuration for measurements, it is recommended to use the 1GbE interface, as it offers a larger data transfer bandwidth.

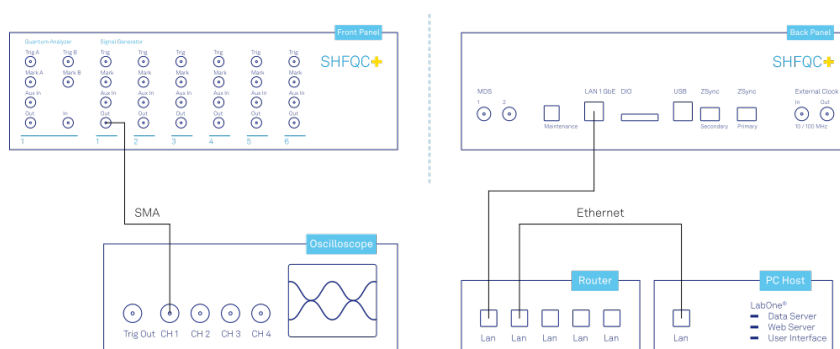


Figure 4.1: Connections for the basic sine generation tutorial

The tutorial can be started with the default instrument configuration (e.g. after a power cycle) and the default user interface settings (e.g. after pressing F5 in the browser).

Generating a Sinusoidal Signal

Note

This tutorial focuses on how to use the sine generator to produce a signal at a single, continuous frequency without any AWG control. This mode of operation is distinct from the method of modulating the output of the AWG output described in the [Digital Modulation Tutorial](#), and the two approaches generally do not need to be employed simultaneously.

In this tutorial we generate a continuous sinusoidal signal at a single frequency and visualize it with a scope. In a first step, we use the [In/Out Tab](#) to enable the Output of the first Signal Generator channel of

the SHFQC+ and set the Output Range. We also set its RF Center Frequency to 1 GHz. Depending on the desired center frequency, either the RF or LF paths can be used. In this example, we will use the RF path, but the LF path can also be used for center frequencies in the range 0 - 2 GHz. Additionally, we configure the scope with a suitable time base (e.g. 500 ps per division) and range (e.g. 0.2 V per division). The following table summarizes the necessary settings.

Table 4.1: Settings: enable the output

Tab	Section	Label	Setting / Value / State
In/Out	SG Channel 1	On	ON
In/Out	SG Channel 1	Range (dBm)	10
In/Out	SG Channel 1	Center Freq (Hz)	1.0 G
In/Out	SG Channel 1	Output path	RF

In addition to turning on the output, we must also configure the sine generator. We set the amplitudes of the I and Q components to yield a single sideband signal, and we set the oscillator frequency to 100 MHz. We also enable the I and Q signals so that an output signal is actually generated. With the RF center frequency set at 1.0 GHz and the oscillator set to 100 MHz, the final output frequency is 1.1 GHz.

Note

To access the I and Q settings of the Sine Generator, it is necessary to expand the menu in the Sine Generator section of the Digital Modulation Tab. The settings are collapsed by default.

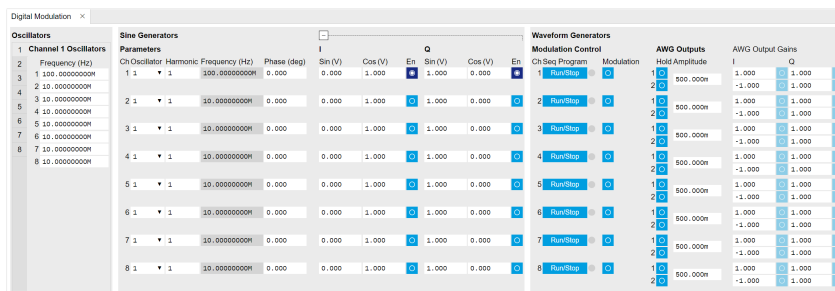


Figure 4.2: LabOne UI: Digital Modulation tab

Table 4.2: Settings: configure the sine generator

Tab	Section	Sub-section	Label	#	Setting / Value / State
Digital Modulation	Channel 1 Oscillators		Frequency	1	100 M
Digital Modulation	Sine Generators	Parameters	Oscillator	1	1
Digital Modulation	Sine Generators	Parameters	Harmonic	1	1
Digital Modulation	Sine Generators	I	Sin(V)	1	0.0
Digital Modulation	Sine Generators	I	Cos(V)	1	1.0
Digital Modulation	Sine Generators	I	En	1	ON
Digital Modulation	Sine Generators	Q	Sin(V)	1	1.0
Digital Modulation	Sine Generators	Q	Cos(V)	1	0.0
Digital Modulation	Sine Generators	Q	En	1	ON

With these settings, we observe a continuously playing 1.1 GHz signal on the scope.

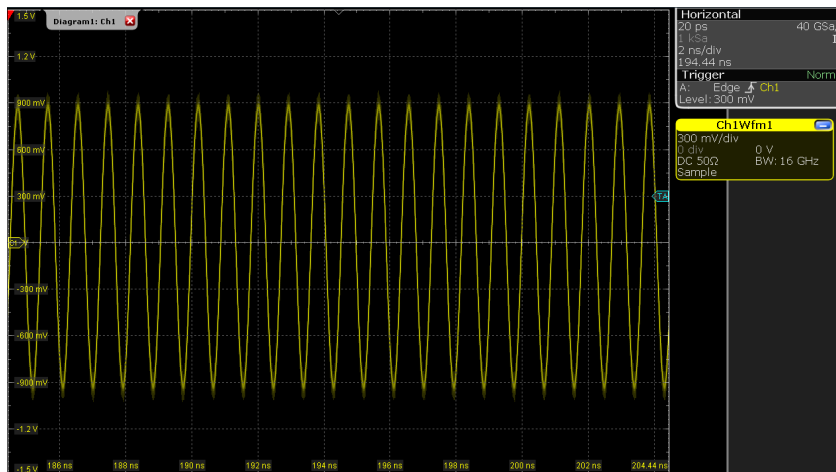


Figure 4.3: Scope trace of a 1.1-GHz signal

Note

The oscillator used for the sine generation, its harmonic, and the phase of the sine generator can be used to further customize the output signal of the sine generator.

4.1.2. Basic Waveform Playback

Note

This tutorial is applicable to all SHFQC+ Instruments.

Goals and Requirements

The goal of this tutorial is to demonstrate the basic use of the Signal Generator channels of the SHFQC+, by demonstrating simple waveform generation and playback. In order to visualize the multi-channel signals, an oscilloscope with sufficient bandwidth and channel number is required. This can be an external scope, or the scope of the SHFQC+, for which a loopback configuration is needed with the Quantum Analyzer channel.

Preparation

Connect the cables as illustrated below. Make sure that the instrument is powered on and connected by Ethernet to your local area network (LAN) where the host computer resides. After starting LabOne, the default web browser opens with the LabOne graphical user interface.

Note

The instrument can also be connected via the USB interface, which can be simpler for a first test. As a final configuration for measurements, it is recommended to use the 1GbE interface, as it offers a larger data transfer bandwidth.

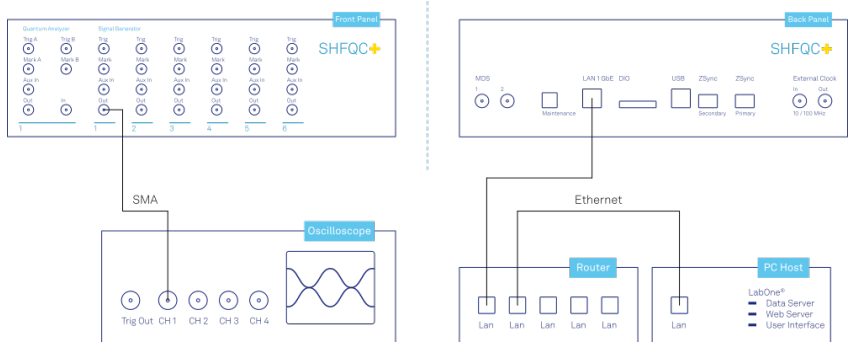


Figure 4.4: Connections for the arbitrary waveform generator basic playback tutorial

The tutorial can be started with the default instrument configuration (e.g. after a power cycle) and the default user interface settings (e.g. after pressing F5 in the browser).

Waveform Generation and Playback

In this tutorial we generate signals with the AWG and visualize them with the scope. In a first step we enable the Output of the Signal Generator channel of the SHFQC+ and set the Output Range. We also set the RF center frequency to 1 GHz. In this example, we will use the RF path, which supports center frequencies in the range 0.6 - 8 GHz. When using the LF path, the center frequency can be set in the range 0 - 2 GHz. Additionally, we configure the scope with a suitable time base (e.g. 500 ns per division) and range (e.g. 0.2 V per division). The following table summarizes the necessary settings.

Table 4.3: Settings: enable the output

Tab	Sub-tab	Label	Setting / Value / State
In/Out	SG Channel 1	On	ON
In/Out	SG Channel 1	Range (dBm)	10
In/Out	SG Channel 1	Center Freq (Hz)	1.0 G
In/Out	SG Channel 1	Output Path	RF

Table 4.4: Settings: configure the external scope

Scope Setting	Value / State
Ch1 enable	ON
Ch1 range	0.2 V/div
Timebase	500 ns/div
Trigger source	Ch1
Trigger level	200 mV
Run / Stop	ON

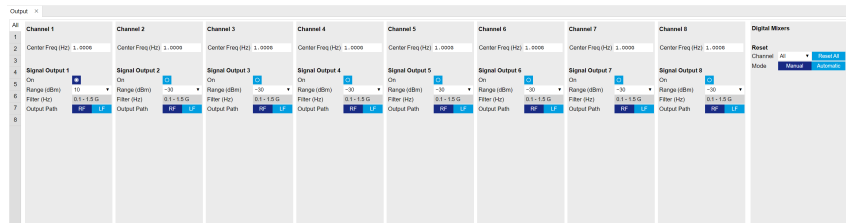


Figure 4.5: LabOne UI: Output tab

In the [In/Out Tab](#), we configure the first Signal Generator output channel.

The final signal amplitude is determined by the dimensionless signal amplitude stored in the waveform memory scaled to the set Range in dBm of the channel. The necessary settings are summarized in the following table.

Table 4.5: Settings: configure the AWG output

Tab	Sub-tab	Section	#	Label	Setting / Value / State
AWG	Control			Sampling Rate	2 GHz
Digital Modulation		Modulation Control	1	Modulation	OFF
Digital Modulation		AWG Outputs		Amplitude	1.0

To operate the AWG we need to specify a sequence program through a C-type language. This program is then compiled and uploaded to the instrument where it is executed in real time. Writing the sequence program can be done interactively by typing the program in the sequence window. Let's start by typing the following code into the sequence editor.

```
wave w_gauss = 1.0*gauss(8000, 4000, 1000);
playWave(1, w_gauss);
```

In the first line of the program, we generate a waveform with a Gaussian shape with a length of 8000 samples and store the waveform under the name **w_gauss**. The peak center position 4000 and the standard deviation 1000 are both defined in units of samples. You can convert them into time by dividing by the chosen Sampling Rate (2.0 GSa/s by default). The waveform generated by the **gauss** function has a peak amplitude of 1. This amplitude is dimensionless and the output amplitude of the physical signal is given by this number multiplied with the voltage determined by the selected output range (here we chose 0 dBm). To calculate the maximum amplitude V_p in Volts use: $V_p = \sqrt{2 * 10^{P_{max}/10} * 10^{-3} W * 50 \Omega}$, where P_{max} is the Range setting in dBm. V_p corresponds to the peak voltage of a signal of a given power when connected to a 50Ω load. To calculate the RMS amplitude V_{rms} , divide by $\sqrt{2}$, i.e. $V_{rms} = \frac{V_p}{\sqrt{2}}$. Of course, the scaling factor of 1.0 in the waveform definition can be replaced by any other value. Finally, the code line is terminated by a semicolon according to C conventions.

With the second line of the program, the generated waveform **w_gauss** is played on the output of the first Signal Generator channel.


We use the syntax **playWave(1,w_gauss)** to play a Gaussian signal in the real quadrature of the complex output. For a more detailed discussion of how the **playWave** command routes the AWG outputs to generate complex signals, see the [Digital Modulation Tutorial](#). Note that the syntax of the **playWave** command and the values of other parameters, such as the waveform amplitude, can yield signals that are either below or above the maximum output power. If a signal happens to be above the maximum output power, it will clip at the DAC and may be distorted. For more details on **playWave** and how different amplitude settings influence the final signal, see the [Modulation Tutorial](#).

Note

For this tutorial, we will keep the description of the Sequencer instructions short. You can find the full specification of the LabOne Sequencer language in [LabOne Sequence Programming](#)

Note

The AWG has a waveform granularity of 16 samples, and a minimum waveform length of 32 samples when using **playWave** commands or 16 samples when using the command table (see the [Pulse-level Sequencing Tutorial](#)). It's recommended to use waveform lengths that are multiples of 16, to avoid having ill-defined samples between successively played waveforms. Waveforms that are not multiple of 16 samples are automatically padded with 0s and a compiler warning is issued.

By clicking on , the sequence program is compiled into sequence instructions that are then uploaded to the device together with the waveform data. A successful upload is indicated by a green Compiler Status LED. Any error that causes an upload failure of either the sequencer instruction or waveform data is indicated by a red status light.

Note

The Advanced tab shows how the sequence instructions translate to assembly language for the onboard FPGA.

By clicking on the Waveform sub-tab, we see that our Gaussian waveform appeared in the list. The Memory Usage field at the bottom of the Waveform sub-tab shows what fraction of the instrument memory is filled by the waveform data. The Waveform Viewer sub-tab allows you to graphically display the currently marked waveform in the list.

Clicking on **Single** executes the uploaded AWG program. Since we have armed the scope previously with a suitable trigger level, it has captured our Gaussian pulse with a FWHM of about 1.5 μ s and a carrier frequency of 1.0 GHz, as shown in Figure 4.6.

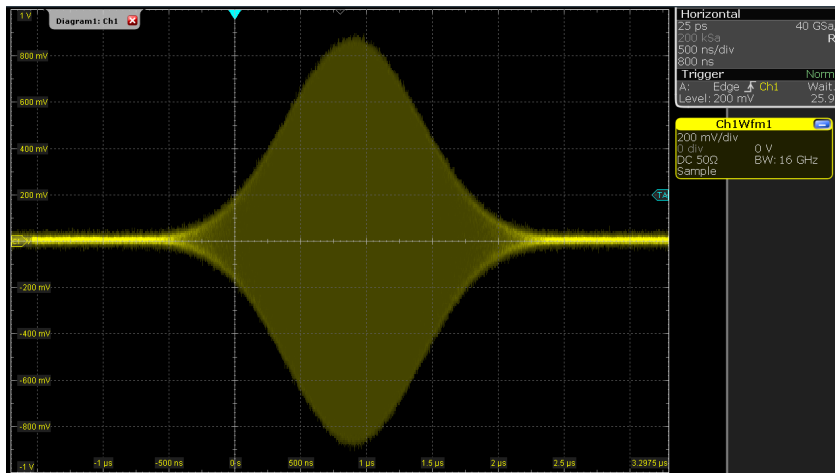


Figure 4.6: Scope shot of a Gaussian pulse generated by the AWG

The LabOne Sequencer language offers various run-time control. An important functionality, e.g. for real-time averaging of an experiment, is the repetition of a sequence. In the following example, all the code within the curly brackets { ... } is repeated 5 times. Upon clicking **Save** and **Single**, we observe 5 short Gaussian pulses in a new scope shot, see Figure 4.7.

```
wave w_gauss = 1.0 * gauss(640, 320, 50);

repeat (5) {
    playWave(1, w_gauss);
}
```

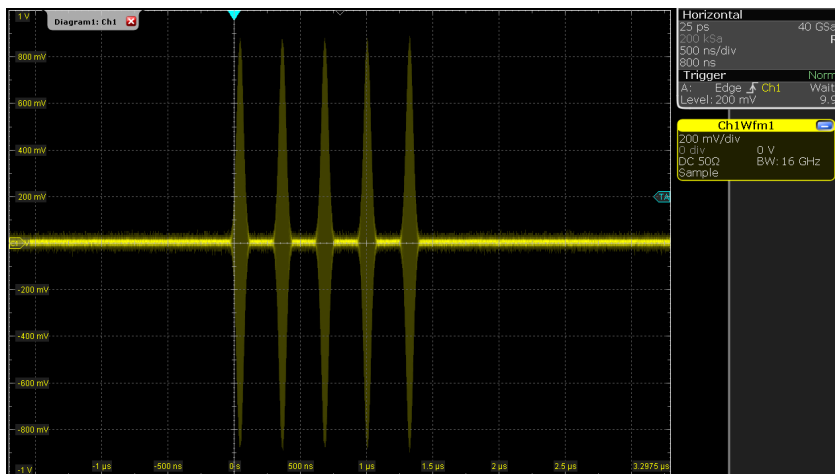


Figure 4.7: Burst of Gaussian pulses generated by the AWG and captured by the scope

In order to generate more complex waveforms, the LabOne Sequencer programming language offers a rich toolset for waveform manipulation. In addition to a selection of standard waveform generation functions, waveforms can be added, multiplied, scaled, concatenated, and truncated. It's also possible to use compile-time evaluated loops to generate pulse series with systematic parameter variations – see [LabOne Sequence Programming](#) for more information. In the following code example, we make use of some of these tools to generate a pulse with a smooth rising edge, a flat plateau, and a smooth falling edge. We use the `cut` function to cut a waveform at defined sample indices, the `ones` function to generate a waveform with constant level 1.0 and length 320, and the `join` function to concatenate three (or arbitrarily many) waveforms.

```
wave w_gauss = gauss(640, 320, 50);
wave w_rise = cut(w_gauss, 0, 319);
wave w_fall = cut(w_gauss, 320, 639);
```

```

wave w_flat = rect(320, 1.0);

wave w_pulse = join(w_rise, w_flat, w_fall);

while (true) {
    playWave(1, w_pulse);
}

```

Note that we replaced the finite repetition by an infinite repetition by using a **while** loop. Loops can be nested in order to generate complex playback routines. The output generated by the program above is shown in [Scope shot of an infinite pulse series generated by the AWG](#).

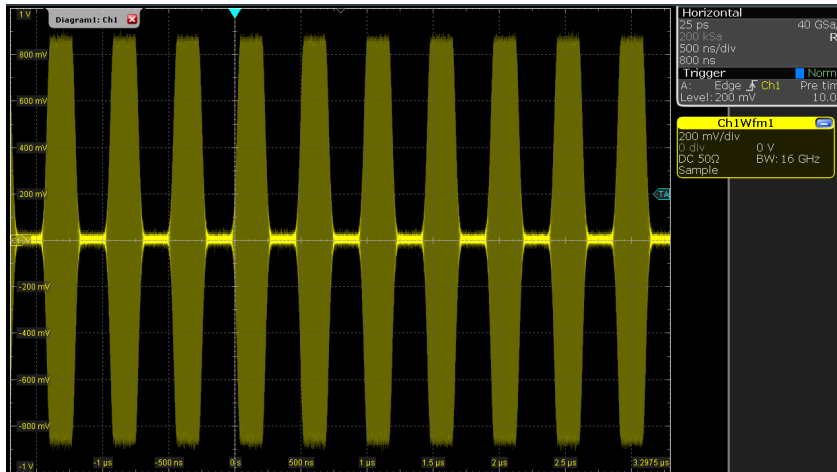


Figure 4.8: Scope shot of an infinite pulse series generated by the AWG

As programs get longer, it becomes useful to store and recall them. Clicking on **Save as...** allows you to store the present program under a new name. Clicking on **Save** then saves your program to the file name displayed at the top of the editor. As you begin to work on sequence programs more regularly, it's worth using some of the editor keyboard shortcuts listed in [Sequence Editor Keyboard Shortcuts](#).

It's also possible to iterate over the samples of a waveform array and calculate each one of them in a loop over a compile-time variable

cvar. This often allows sequences to go beyond the possibilities of using the predefined waveform generation function, particularly when using nested formulas of elementary functions like in the following example. The waveform array needs to be pre-allocated e.g. using the instruction **zeros**.

```

const N = 1024;
const width = 100;
const position = N/2;
const f_start = 0.1;
const f_stop = 0.2;
cvar i;
wave w_array = zeros(N);
for (i = 0; i < N; i++) {
    w_array[i] = sin(10/(cosh((i-position)/width)));
}

playWave(w_array);

```

It is also possible to use waveforms stored as a list of values in a file. If the file is stored in the location (C:\Users\<user name>\Documents\Zurich Instruments\LabOne\WebServer\awg\waves\ under Windows or ~/Zurich Instruments/LabOne/WebServer/awg/waves/ under Linux), you can then play back the wave by referring to the file name without extension in the sequence program:

```
playWave("wave_file");
```

If you prefer, you can also store it in a **wave** data type first and give it a new name:

```

wave w = "wave_file";
playWave(w);

```

For more information about the file format, please refer to the AWG Module Section of the LabOne Programming Manual.

Using the LF Path

The LF path bypasses the upconversion chain to allow center frequencies in the range DC to 2 GHz to be generated. The AWG sequencer can be programmed in the same way as with the RF path. The main differences is that the maximum output power of the LF path is +5 dBm (compared to +10 dBm for the RF path) and that the latency of the LF path is shorter than that of the RF path, due to the shorter analog path.

The center frequency of the LF path can be set in multiples of 100 MHz, just as with the RF path. When combined with correct usage of **waitDigTrigger** and **resetOscPhase** commands, this ensures that the initial phase of a played waveform will be reproducible within a given experimental run, as in the following example:

```
const length = 128;
const amp = 1;
wave = gaussian(length, amp, length/2, length/8);

while (1) {
    waitDigTrigger(1);
    resetOscPhase();
    playWave(1, 2, wave);
}
```

4.1.3. Triggering and Synchronization

Note

This tutorial is applicable to all SHFQC+ Instruments.

Goals and Requirements

The goal of this tutorial is to show how to use the Signal Generator channels of the SHFQC+ as a trigger source, as well as how to configure the SHFQC+ to respond to an external trigger. In order to visualize the multi-channel signals, an oscilloscope with sufficient bandwidth and channel number is required. This can be an external scope, or the scope of the SHFQC+, for which a loopback configuration is needed with the Quantum Analyzer channel.

Preparation

Connect the cables as illustrated below. Make sure that the instrument is powered on and connected by Ethernet to your local area network (LAN) where the host computer resides. After starting LabOne, the default web browser opens with the LabOne graphical user interface.

Note

The instrument can also be connected via the USB interface, which can be simpler for a first test. As a final configuration for measurements, it is recommended to use the 1GbE interface, as it offers a larger data transfer bandwidth.

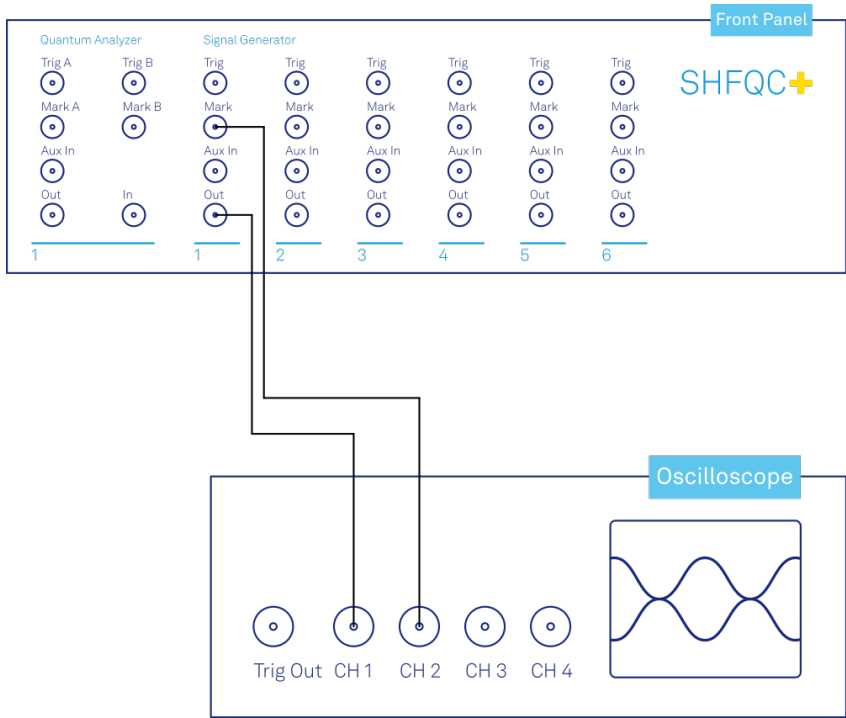


Figure 4.9: Connections for the arbitrary waveform generator triggering and synchronization tutorial

The tutorial can be started with the default instrument configuration (e.g. after a power cycle) and the default user interface settings (e.g. after pressing F5 in the browser).

Generating and Responding to Triggers

In this tutorial you will learn about the most important use cases:

- Generating a TTL signal with the AWG to trigger another piece of equipment
- Triggering the Signal Generator channels of the AWG with an external TTL signal

Generating Markers with the AWG

To begin with, we generate a trigger output with the Signal Generator channel 1. As this tutorial is an extension of the [Basic Waveform Playback Tutorial](#), configure the SHFQC+ as follows:

Table 4.6: Settings: configure the output

Tab	Sub-tab	Label	Setting / Value / State
In/Out	SG Channel 1	On	ON
In/Out	SG Channel 1	Range (dBm)	10
In/Out	SG Channel 1	Center Freq (Hz)	1.0 G
In/Out	SG Channel 1	Output Path	RF
In/Out	SG Channel 2	On	ON
In/Out	SG Channel 2	Range (dBm)	10
In/Out	SG Channel 2	Center Freq (Hz)	1.0 G
In/Out	SG Channel 2	Output Path	RF

Table 4.7: Settings: configure the external scope

Scope Setting	Value / State
Ch1 enable	ON

Scope Setting	Value / State
Ch1 range	0.2 V/div
Ch2 enable	ON
Ch2 range	0.5 V/div
Timebase	500 ns/div
Trigger source	Ch2
Trigger level	200 mV
Run / Stop	ON

After configuring the output using the table above, we use the SHFQC+ to generate a trigger output. There are two ways of generating trigger output signals with the Signal Generators AWG: as markers that are part of a waveform and played with sample precision, or by controlling trigger bits through the sequencer.

The method using markers is recommended when precise timing is required, and/or complicated serial bit patterns need to be played on the Marker outputs. Marker bits are part of every waveform, and are set to zero by default. Each waveform is represented by an array of 16-bit words: 14 bits of each word represent the analog waveform data, and the remaining 2 bits represent two digital marker channels. Hence, upon playback, a digital signal with sample-precise alignment with the analog output is generated.

Generating a TTL output signal using a sequencer instruction is simpler, but the timing resolution is lower than when using markers. The sequencer instructions play at the sequencer clock cycle of 4 ns, whereas the markers are part of the waveform and therefore have a resolution of 0.5 ns. The method using sequencer instructions is useful to generate a single trigger signal at the start of an AWG program, for instance.

Table 4.8: Comparison: AWG markers and triggers

	Marker	Trigger
Implementation	Part of waveform	Sequencer instruction
Timing control	High	Low
Generation of serial bit patterns	Yes	No
Cross-device synchronization	Yes	Yes

Let us first demonstrate the use of **markers**. In the following code example we first generate a Gaussian pulse. This is identical as in the [Basic Waveform Playback Tutorial](#), where the generated wave already included marker bits - they were simply set to zero by default. We use the **marker** function to assign the desired non-zero marker bits to the wave. The **marker** function takes two arguments: the first is the length of the wave in samples; the second is the marker configuration in binary encoding, where the value 0 stands for both marker bits low, the values 1, 2, and 3 stand for the first, the second, and both marker bits high, respectively. We use this to construct the wave called **w_marker**.

```
const marker_pos = 3000;

wave w_gauss = gauss(8000, 4000, 1000);
wave w_left = marker(marker_pos, 0);
wave w_right = marker(8000-marker_pos, 1);
wave w_marker = join(w_left, w_right);
wave w_gauss_marker = w_gauss + w_marker;

playWave(1, w_gauss_marker);
```

The waveform addition with the '+' operator adds up analog waveform data but also combines marker data. The wave **w_gauss** contains zero marker data, whereas the wave **w_marker** contains zero analog data. Consequently the wave called **w_gauss_marker** contains the merged analog and marker data. We use the integer constant **marker_pos** to determine the point where the first marker bit flips from 0 to 1 somewhere in the middle of the Gaussian pulse.

Note

The add function and the '+' operator combine marker bits by a logical OR operation. This means combining 0 and 1 yields 1, and combining 1 and 1 yields 1 as well.

There is a certain freedom to assign different marker bits to the Mark outputs. The following table summarizes the settings to apply in order to output marker bit 1 on the Mark output of the Signal Generator channel 1. (% else -\$) Mark 1.

Table 4.9: Settings: configure the AWG marker output and scope trigger

Tab	Sub-tab	Section	#	Label	Setting / Value / State
DIO		Marker Out	1	Signal	Output 1 Marker 1

Figure 4.10 shows the AWG signal captured by the scope as a yellow curve. The green curve shows the second scope channel displaying the marker signal. Try changing the `marker_pos` constant and re-running the sequence program to observe the effect on the temporal alignment of the Gaussian pulse. After the waveform has finished playing, the marker bit returns to a value of zero automatically, as no more waveform is being played.

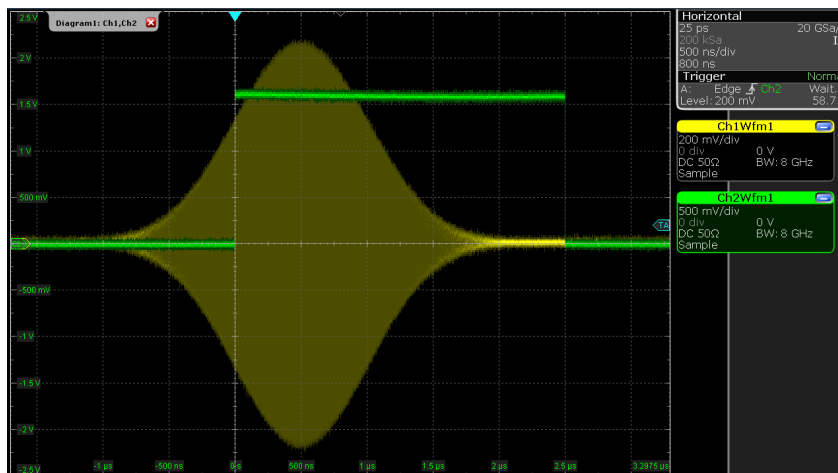


Figure 4.10: Gaussian pulse and square marker signal generated by the AWG and captured by the scope

Let us now demonstrate the use of **sequencer instructions** to generate a trigger signal. Copy and paste the following code example into the Sequence Editor.

```
wave w_gauss = gauss(8000, 4000, 1000);

setTrigger(1);
playWave(1, w_gauss);
waitWave();
setTrigger(0);
```

Each AWG core has four trigger output states available to it. The `setTrigger` function takes a single argument encoding the four trigger output states in binary manner – the integer number 1 corresponds to a configuration of 0/0/0/1 for the trigger outputs 4/3/2/1. The binary integer notation of the form 0b0000 is useful for this purpose – e.g. `setTrigger(0b0011)` will set trigger outputs 1 and 2 to 1, and trigger outputs 3 and 4 to 0. We included a `waitWave` instruction after the `playWave` instruction. It ensures that the subsequent `setTrigger` instruction is executed only after the Gaussian wave has finished playing, and not during waveform playback.

Note

The `waitWave` instruction represents a means to control the timing of instructions in the Wait & Set and the Playback queues. In the example above, the `waitWave` instruction puts the playback of the next instruction in the Wait & Set queue, in this case `setTrigger(0)`, on hold until the waveform is finished. Without the `waitWave` instruction, the AWG trigger would return to zero at the beginning of the waveform playback.

Note

The use of `waitWave` is explicitly not required between consecutive `playWave` and `playZero` instructions. Sequential instructions in the Playback queue are played immediately after one another, back to back.

We reconfigure the Mark 1 connector in the DIO tab such that it outputs `AWG Trigger 1`, instead of `Output 1 Marker 1`. The rest of the settings can stay unchanged.

Table 4.10: Settings: configure the AWG trigger output

Tab	Sub-tab	Section	#	Label	Setting / Value / State
DIO		Marker Out	1	Signal	AWG Trigger 1

Figure 4.11 shows the AWG signal captured by the scope. This looks very similar to Figure 4.10 in fact. With this method, we're less flexible in choosing the trigger time, as the rising trigger edge will always be at the beginning of the waveform. But we don't have to bother about assigning the marker bits to the waveform.

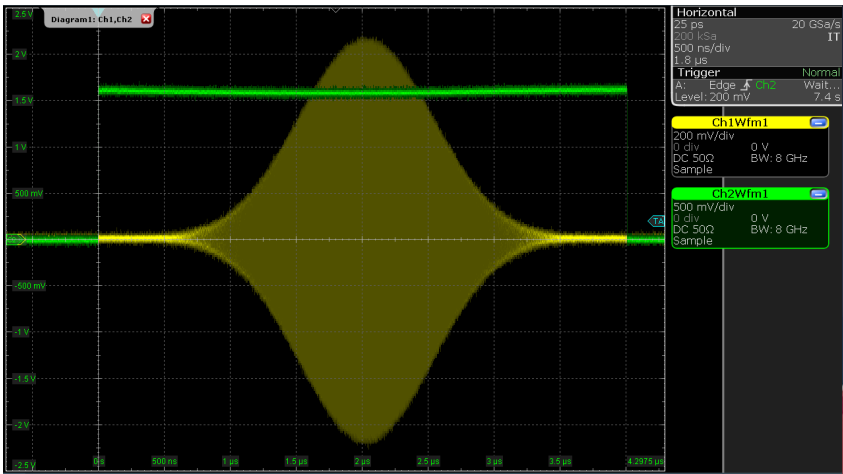


Figure 4.11: Gaussian pulse and trigger signal generated by the AWG and captured by the scope

Triggering the AWG

Note

This section shows how to use the SHFQC+ to generate and respond to external triggers. To synchronize the outputs of different channels on the same SHFQC+, it is recommended to use the Internal Trigger Unit.

For this part of the tutorial, connect the cables as illustrated below.

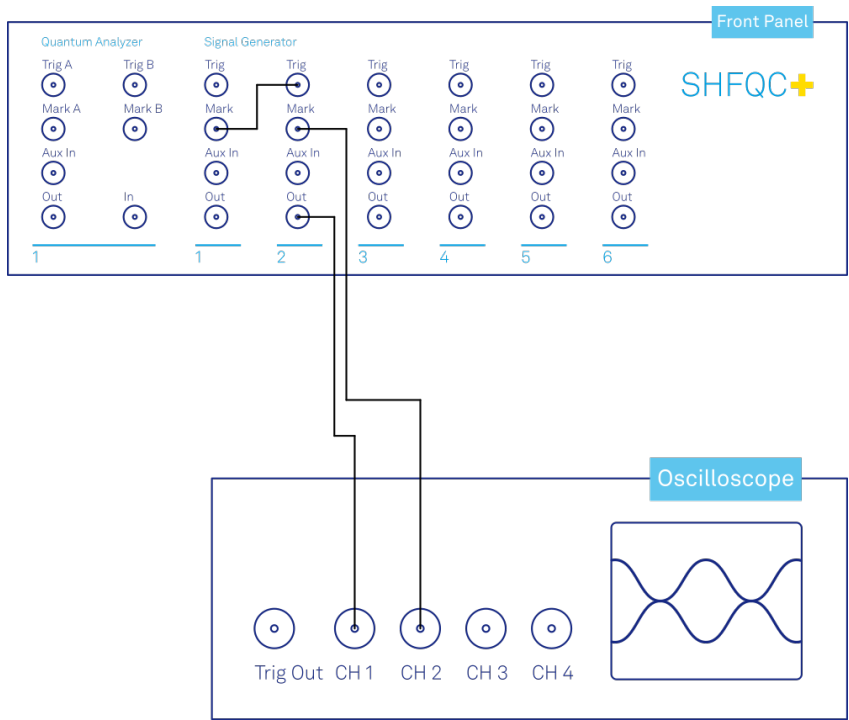


Figure 4.12: Connections for the arbitrary waveform generator basic playback tutorial

In this section we show how to trigger the AWG with an external TTL signal. We start by using the Signal Generator channel 1 of the SHFQC+ to generate a periodic TTL signal. As shown in [Figure 4.12](#), the Mark output of channel 1 is connected to the Trig input of channel 2. We monitor the marker and signal outputs of channel 2 on a scope.

```

wave m_high = marker(8000,1); //marker high for 8000 samples
wave m_low = marker(8000,0); //marker low for 8000 samples
wave m = join(m_high, m_low);

while (1) {
    playWave(m);
}
```

Compile and run the above program on the AWG core of channel 1. Then configure the Mark 1 and Mark 2 to use **Output 1 Marker 1**:

Table 4.11: Settings: configure the AWG marker output

Tab	Sub-tab	Section	#	Setting / Value / State
DIO	Marker	Source	1	Output 1 Marker 1
DIO	Marker	Source	2	Output 1 Marker 1

Next we configure channel 2 to respond to the trigger generated by channel 1. Internally, the AWG core of each channel has 2 digital trigger input channels. These are not directly associated with physical device inputs but can be freely configured to probe a variety of internal or external signals. Here, we link the AWG Digital Trigger 1 of Channel 2 to the physical Trig 2 connector, and we configure it to trigger on the rising edge.

Table 4.12: Settings: configure the AWG digital trigger input

Tab	#	Sub-tab	Section	Label	Setting / Value / State
AWG	2	Trigger	Digital Trigger 1	Signal	Trigger In 2
AWG	2	Trigger	Digital Trigger 1	Slope	Rise

Finally, we modify the previous AWG program by adding a **while** loop so that the sequence can be repeated infinitely and by including a **waitDigTrigger** instruction just before the **playWave** instruction. The result is that upon every repetition inside the infinite **while** loop, the AWG will wait for a rising edge on Trig 2.

```

const marker_pos = 3000;

wave w_gauss = gauss(8000, 4000, 1000);
wave w_left = marker(marker_pos, 0);
wave w_right = marker(8000-marker_pos, 1);
wave w_marker = join(w_left, w_right);
wave w_gauss_marker = w_gauss + w_marker;

while (1) {
    //wait for external trigger
    waitDigTrigger(1);
    playWave(1, w_gauss_marker);
}

```

Compile and run the above program on the AWG core of channel 2. [Figure 4.13](#) shows the pulse series as seen on the scope: the pulses are now spaced by the oscillator period of 8 μ s, unlike previously when the period was determined by the length of the waveform **w_gauss**. Try changing the trigger signal frequency or unplugging the trigger cable to observe the immediate effect on the signal.

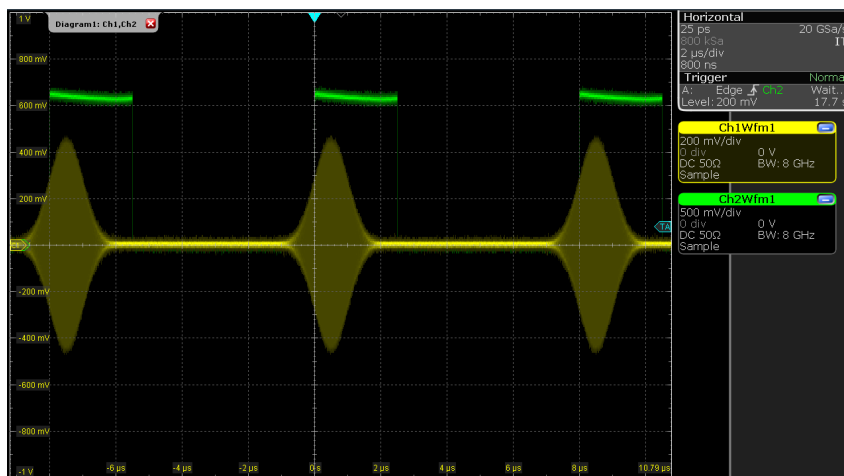


Figure 4.13: Externally triggered pulse series generated by the AWG.

Synchronizing outputs of different channels

For this part of the tutorial, connect the cables as illustrated below.

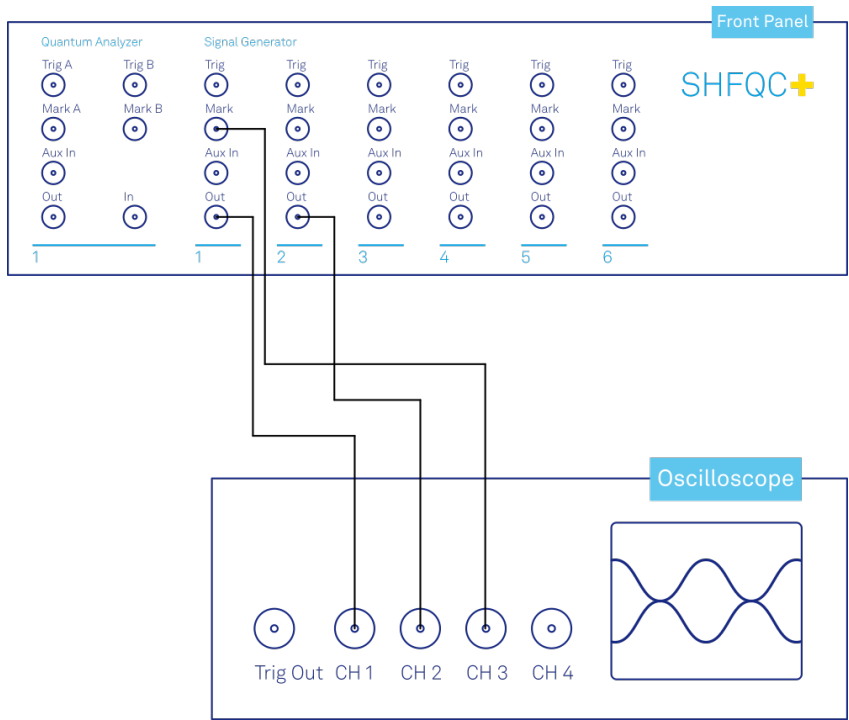


Figure 4.14: Connections for the synchronizing outputs of multiple channels.

In this section we will show how to use the Internal Trigger Unit to synchronize the outputs of multiple channels of the SHFQC+.

Note

In this tutorial, we show how to synchronize two Signal Generator Channel outputs of the SHFQC+, but the Internal Trigger Unit can also be used to synchronize the Quantum Analyzer and Signal Generator channel outputs with each other by using the `waitDigTrigger` command in the Quantum Analyzer sequence and configuring the digital trigger of the Quantum Analyzer channel to use the Internal Trigger Unit.

To configure the internal trigger, set the following settings in the tables below.

Table 4.13: Settings: configure the Internal Trigger Unit

Tab	Sub-tab	Section	#	Setting / Value / State
DIO	Internal Trigger	Repetitions		1e9
DIO	Internal Trigger	Holdoff		100 ns

Table 4.14: Settings: configure the AWGs to use the Internal Trigger Unit

Tab	#	Sub-tab	Section	Label	Setting / Value / State
AWG	1	Trigger	Digital Trigger 1	Signal	Internal Trigger
AWG	2	Trigger	Digital Trigger 1	Signal	Internal Trigger

The number of repetitions (ranging from 1 to more than 1e9) determines how many triggers will be sent out. The holdoff time (minimum 100 ns, resolution of 100 ns, maximum 4000 s) determines the time in seconds between the individual trigger events, typically chosen to be longer than the longest part of the sequence. For example, in a Ramsey experiment, the hold-off time might be slightly longer than the length of two $\pi/2$ -pulses plus the length of the longest evolution time and the length of the readout pulse. After entering the settings above, the Internal Trigger Unit is configured but is not yet sending out triggers. We will enable the triggers after uploading our sequences.

Compile and run the sequencer code below (which is the same as in the section above) to SG channels 1 and 2 of the SHFQC+.

```
const MARKER_POS = 3000;
```

```

wave w_gauss = gauss(8000, 4000, 1000);
wave w_left = marker(MARKER_POS, 0);
wave w_right = marker(8000-MARKER_POS, 1);
wave w_marker = join(w_left, w_right);
wave w_gauss_marker = w_gauss + w_marker;

while (1) {
    //wait for internal trigger
    waitDigTrigger(1);
    playWave(1,2, w_gauss_marker);
}

```

The sequencers are now waiting until they receive a trigger. Enable the internal trigger by clicking **Run/Stop** button in the Internal Trigger section of the DIO Tab. Figure 4.15 shows that the outputs of channels 1 and 2 are synchronized in time.



Figure 4.15: Synchronized output signals of the SHFQC+.

4.1.4. Digital Modulation

Note

This tutorial is applicable to all SHFQC+ Instruments.

Goals and Requirements

The goal of this tutorial is to demonstrate the use of the digital modulation feature of the AWG of the Signal Generator channels. In order to visualize the generated signals, an oscilloscope with sufficient bandwidth and channels is required. It can also be helpful to use a scope with FFT functionality to visualize the spectrum of the output signal. This can be an external scope, or the scope of the SHFQC+, for which a loopback configuration is needed with the Quantum Analyzer channel.

Preparation

Connect the cables as illustrated below. Make sure that the instrument is powered on and connected by Ethernet to your local area network (LAN) where the host computer resides. After starting LabOne, the default web browser opens with the LabOne graphical user interface.

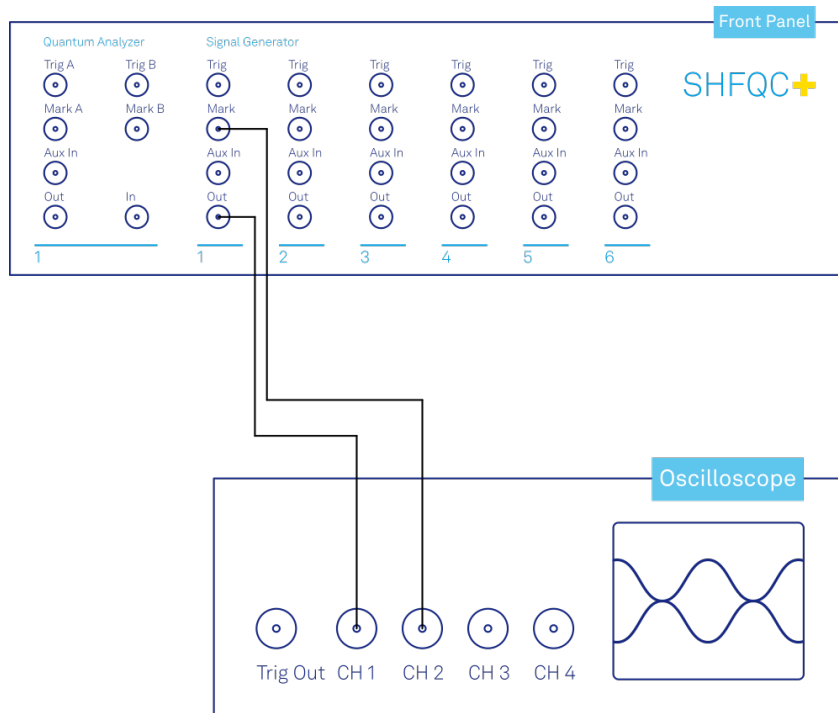


Figure 4.16: Connections for the arbitrary waveform generator digital modulation tutorial

The tutorial can be started with the default instrument configuration (e.g. after a power cycle) and the default user interface settings (e.g. as is the case after pressing F5 in the browser).

Note

The instrument can also be connected via the USB interface, which can be simpler for a first test. As a final configuration for measurements, it is recommended to use the 1GbE interface, as it offers a larger data transfer bandwidth.

Generating a Single Sideband Signal

Note

This tutorial focuses on how to use the sine generator to modulate the output of the AWG core so that it can be used for generating a pulse sequence at a single sideband. This mode of operation is distinct from the method of generating a single, continuous frequency described in the [Basic Sine Generation Tutorial](#), and the two approaches should generally not be employed simultaneously.

In digital modulation mode, the output of the AWG is multiplied with the signal of the internal sine generator of the instrument. There are numerous advantages to using digital modulation in comparison to simply generating the sinusoidal signal directly using the waveform memory, such as the ability to change the frequency without uploading a new waveform, extremely high frequency resolution independent of AWG waveform length, phase-coherent generation of signals (because the oscillators keep running even when the AWG is off), and more. The goal of this section is to demonstrate how to use the modulation mode.

The superheterodyne upconversion scheme of the SHFQC+ consists of a chain of several conversion steps, which can be summarized in a simple model for the generated RF voltage:

$$V_{\text{RF}}(t) = V_0 \operatorname{Re}[A(w_I(t) + iw_Q(t))e^{+i\phi}e^{+i2\pi f_{\text{Osc}}t}e^{+i2\pi f_{\text{RF}}t}], \quad (1)$$

where w_I and w_Q are the baseband I and Q waveforms played by the AWG core, A is a global amplitude for scaling the AWG signal, f_{Osc} is the oscillator frequency set in the Digital Modulation tab, ϕ is a phase offset of the sine generator and is set in the Digital Modulation tab, f_{RF} is the RF center frequency, and

$$V_0 = \sqrt{2 * 10^{P_{\max}/10} * 10^{-3} W * 50 \Omega}$$

is the maximum output voltage determined by the range setting P_{\max} with a 50Ω load.

Note

This way of up-converting provides an intuitive way of understanding the Signal Generators' channel output spectrum. If one defines a waveform in the AWG and performs the standard complex Fourier transform on it, the channels output spectrum is directly given by shifting the spectrum by the chosen center frequency, i.e. by replacing DC by the center frequency value.

This expression can be written using real-valued sines and cosines rather than complex exponentials:

$$V_{RF}(t) = V_0 A [w_I(t) \cos(2\pi f_{Osc} t + \phi) - w_Q(t) \sin(2\pi f_{Osc} t + \phi)] \cos(2\pi f_{RF} t) \\ - V_0 A [w_I(t) \sin(2\pi f_{Osc} t + \phi) + w_Q(t) \cos(2\pi f_{Osc} t + \phi)] \sin(2\pi f_{RF} t). \quad (2)$$

We now look at how the Signal Generator channel of the SHFQC+ actually generates modulated signals. In the LabOne UI, there are four AWG output gains that can be used to set up single-sideband modulation. These AWG output gains are multiplied by the AWG outputs. The AWG output gains can be set individually to make it easier to calibrate DRAG pulses, for example.

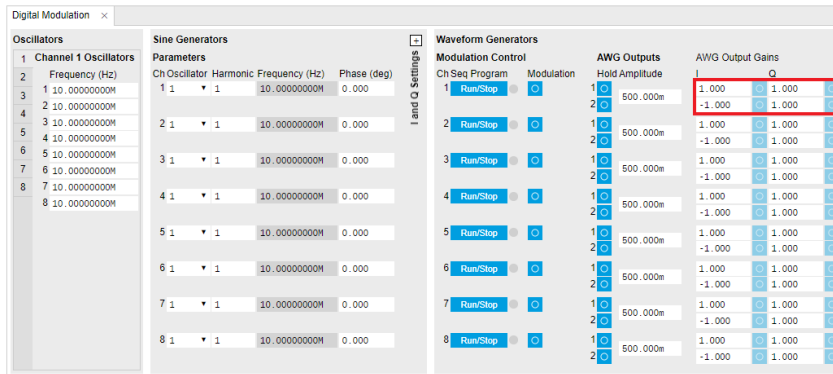


Figure 4.17: Digital Modulation tab in the LabOne UI

When modulation is enabled, the AWG output gains control of the signs and amplitudes of the sinusoids that are multiplied with the AWG outputs. To make use of the four AWG output gains settings needed for a complex, dual-channel signal, the sequencer code must make use of the **playWave** command in the form **playWave(1,2, wI, 1,2, wQ)**. Figure 4.18 shows how the different parts of the **playWave** command map to the different gain nodes in the digital modulation process.

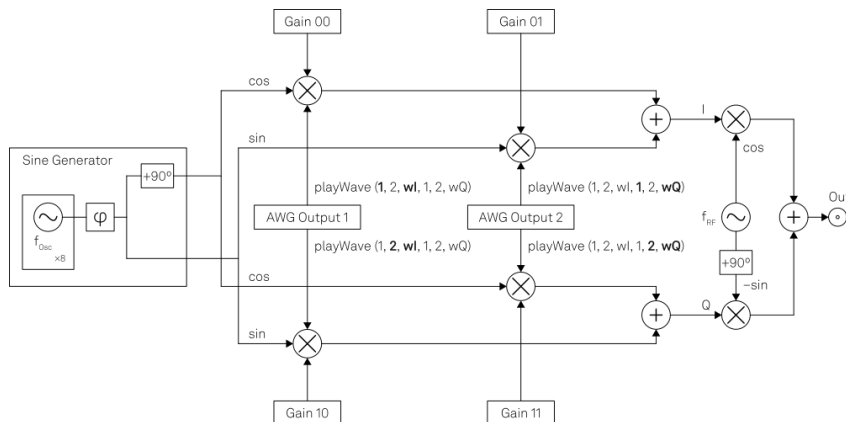


Figure 4.18: Diagram of digital modulation processing chain and settings. Bolded parts of the text show how the different parts of the 'playWave' command map to the different gain nodes.

We can summarize the signal generated by the FPGA using the following expression:

$$V_{RF}(t) = V_0 A [\text{Gain00} \times w_I(t) \cos(2\pi f_{Osc}t + \phi) + \text{Gain01} \times w_Q(t) \sin(2\pi f_{Osc}t + \phi)] \cos(2\pi f_{RF}t) \\ - [\text{Gain10} \times w_I(t) \sin(2\pi f_{Osc}t + \phi) + \text{Gain11} \times w_Q(t) \cos(2\pi f_{Osc}t + \phi)] \sin(2\pi f_{RF}t) \quad (3)$$

where **Gain_{ij}** of Channel **n** corresponds to the node path `<dev>/SGCHANNELS/<chan>/AWG/OUTPUTS/<i>/GAINS/<j>`. The choice of **Gain00 = Gain10 = Gain11 = 1.0** and **Gain01 = -1.0** yields the same expression as in Equation 2 and therefore leads to single sideband modulation at the upper sideband for positive oscillator frequencies. Note that in this simplified overview of the digital modulation and upconversion chain, we have lumped together the digital upconversion to 2.0 GHz and the digital-to-analog conversion with the analog upconversion pathway into a single upconversion step. See also [In/Out Tab](#).

Note

Depending on the selected value of **f_{Osc}**, the voltage measured on a scope may not correspond exactly to the formula above due to the effect of the filters used in the analog upconversion.

Note

For HDAWG users: The SHFQC+ and HDAWG use different sign conventions for achieving upper sideband modulation. Because the HDAWG is typically used in combination with physical IQ mixers when generating an RF signal, the HDAWG assumes a negative time dependence in the exponential, whereas the SHFQC+ assumes a positive time dependence. To achieve upper sideband modulation on both instruments, there is therefore a relative sign swap needed on the gain settings **Gain01** and **Gain10**.

The following table summarizes the parameter names and their corresponding node paths. For more information on setting node values via API, see the [Using the Python API Tutorial](#). See also [Node Documentation](#).

Table 4.15: Summary of parameters used in AWG signal generation

Parameter name	Symbol	Node path
I waveform	w_I(t)	Defined in sequence
Q waveform	w_Q(t)	Defined in sequence
AWG output gains	Gain_{ij}	<code><dev>/SGCHANNELS/<chan>/AWG/OUTPUTS/<i>/GAINS/<j></code>
Global amplitude	A	<code><dev>/SGCHANNELS/<chan>/AWG/OUTPUTAMPLITUDE</code>
Oscillator frequency	f_{Osc}	<code><dev>/SGCHANNELS/<chan>/OSCS/<OSC>/FREQ</code>
Sine generator phase	φ	<code><dev>/SGCHANNELS/<chan>/SINES/0/PHASESHIFT</code>
RF center frequency	f_{RF}	<code><dev>/SYNTHESIZERS/<synth>/CENTERFREQ</code>
Output range	P_{max}	<code><dev>/SGCHANNELS/<chan>/OUTPUT/RANGE</code>

We now show how to generate a modulated AWG signal. We monitor the AWG signal using one channel of an external scope and use the second scope channel for triggering purposes. The following tables summarize the settings to enable the SHFQC+ outputs and configure the sine generator, as well as to configure the external scope.

Table 4.16: Settings: enable the output

Tab	Section	Sub-Section	Label	Setting / Value / State
In/Out	SG Channel 1		On	ON
In/Out	SG Channel 1		Range (dBm)	10
In/Out	SG Channel 1		Center Freq (Hz)	1.0 G
In/Out	SG Channel 1		Output Path	RF

```
| Digital Modulation | Waveform Generators | Modulation | 1 | ON | | Digital Modulation | AWG Outputs |
Amplitude | 0.5 | | Digital Modulation | AWG Output Gains | 1 | 1.0 | | Digital Modulation | AWG
Output Gains | 2 | -1.0 | | Digital Modulation | AWG Output Gains | Q | 1 | 1.0 | | Digital Modulation |
AWG Output Gains | Q | 2 | 1.0 | | Digital Modulation | Channel 1 Oscillators | Frequency (Hz) | 1 | 10.0 M |
| Digital Modulation | Sine Generators | 1 | En | OFF | | Digital Modulation | Sine Generators | Q | En |
OFF | | DIO | Marker | Source | 1 | Output 1 Marker 1 |
```

Table 4.17: Settings: Configure the external scope

Scope Setting	Value / State
Ch1 enable	ON
Ch1 range	0.2 V/div
Ch2 enable	ON
Ch2 range	0.5 V/div
Timebase	1 us/div
Trigger source	Ch2
Trigger level	200 mV
Run / Stop	ON

Note

For HDAWG users: Enabling digital modulation on the SHFQC+ is analogous to enabling Sine12 modulation on Wave 1 and Sine21 modulation on Wave 2.

A sine generator is a direct digital synthesis (DDS) unit that converts a digital oscillator signal (essentially just an incrementing phase) to a sinusoid with a certain phase offset and harmonic multiplier using a look-up table containing one period of the sinusoid signal. The digital oscillator in turn is a phase accumulator with a very precise frequency derived from the instrument's main clock. The digital oscillators on the instrument are represented in the Oscillators section of the Digital Modulation tab. Each Signal Generator output channel of the SHFQC+ has 8 oscillators associated with it, although only one can be used by the sine generator at a time. For details on how to switch between oscillators during a sequence, see the [Command Table Tutorial](#).

In this example, we use a Gaussian pulse for the I waveform and a derivative of a Gaussian for the Q waveform. When combined, this generates a DRAG pulse.

```
wave w_gauss = gauss(1024, 512, 128);
wave w_drag = drag(1024, 512, 128);
wave m_high = marker(512, 1);
wave m_low = marker(512, 0);
wave m = join(m_high, m_low);
wave w_gauss_marker = w_gauss + m;

resetOscPhase();

playWave(1,2, w_gauss_marker, 1,2, w_drag);
```

We also configure the FFT settings on the scope.

Table 4.18: Settings: Configure the external scope

Scope Setting	Value / State
Acquisition Time	10 us
FFT Center	1 GHz
FFT Span	100 MHz
Resolution BW	200 kHz

Save and play the Sequencer program with the above settings. The upper plot in [Figure 4.19](#) shows the AWG signals captured by the scope. We see that the resulting DRAG pulse is a combination of a Gaussian waveform with a derivative of a Gaussian waveform generated by the `drag()` function in

SeqC. The FFT of the scope trace shows that there is a dip in the spectrum, a key characteristic of the DRAG pulse combination.

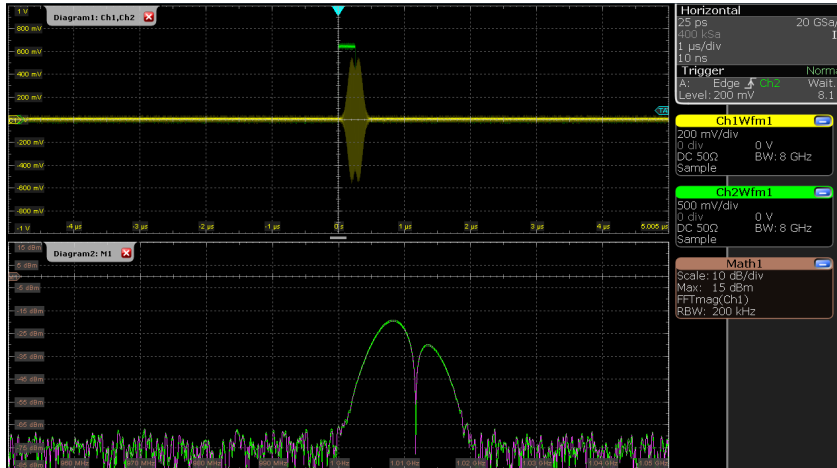


Figure 4.19: Dual-channel signal generated by the AWG and captured by the scope. The top half of the figure shows a pulse that is a combination of a Gaussian pulse and a derivative of a Gaussian pulse, modulated at 10 MHz and upconverted with an RF center frequency of 1.0 GHz. The bottom part of the figure shows the FFT of the scope trace, demonstrating the characteristic spectral dip of the DRAG pulse.

Note

There are two ways of generating AWG signals with a single frequency component at the front panel output when digital modulation is enabled. For completely real signals that require only a single AWG output, `playWave(1,2, wI)` suffices to generate a single sideband signal. For complex signals requiring dual-channel waveforms, `playWave(1,2, wI, 1,2, wQ)` is needed.

Note

To avoid saturating the output when using `playWave(1,2, wI, 1,2, wQ)` syntax, it is necessary to either set the value of the global amplitude to 0.5 or to scale the waveform in the sequencer code similarly. A value of up to 1.0 can safely be used when `playWave(1,2, wI)` is used for generating single sideband real signals.

So far in this tutorial, we have shown how to achieve single sideband modulation with the `playWave` command, but to efficiently use the instruction memory of the SHFQC+ and ensure smooth, back-to-back waveform playback, it is recommended to use the command table, which requires assigning the waveform an index and using the `executeTableEntry` command instead of `playWave`. To assign an index of 0 to a waveform, the command `assignWaveIndex(1,2, wI, 0)` should be used for single-AWG-channel signals and `assignWaveIndex(1,2, wI, 1,2, wQ, 0)` for dual-channel signals. For more details, see the [Command Table Tutorial](#).

Rapid Phase Changes

The SHFQC+ supports rapid, real-time changes of the carrier phase in modulation mode through the sequencer instructions `setSinePhase` and `incrementSinePhase`, as well as through the [command table](#). This capability is particularly valuable when generating long patterns of pulses with varying phases, e.g. to account for AC Stark shift in qubit control sequences, or to realize phase cycling protocols.

In addition, there is the possibility to reset the starting phase of one or multiple oscillators at the beginning of a pulse sequence using the `resetOscPhase` instruction. Thus it can be ensured that the carrier-envelope offset, and thus the final output signal, is identical from one repetition to the next.

In the following AWG sequencer program, we generate a series of 4 dual-channel square pulses that are played back-to-back. We initialize the oscillator phase by a `resetOscPhase` instruction. In this form without an argument, the instruction will reset the phases of all oscillators accessible by this core (here oscillators 1 through 8 of Channel 1). Alternatively, an argument in binary representation,

e.g. **0b0101**, allows us to reset only a subset of these oscillators. We then set the phase of the sine generator to 45 degrees using the **setSinePhase** instruction. Subsequently, we play back the dual-channel waveform 4 times, and after each playback instruction, we increase the phase of the sine generator by 90 degrees. The corresponding instruction **incrementSinePhase** takes effect at the end of the previous waveform playback, which allows us to change the phase precisely in between waveforms. Upload the following sequence program to the AWG and run the sequence.

```
const LENGTH = 48;

wave w = ones(LENGTH);
wave m_high = marker(LENGTH/2, 1); //marker high
wave m_low = marker(LENGTH/2, 0); //marker low
wave m = join(m_high, m_low); //join marker waveforms
wave wm = w + m; //combine marker and ones waveform data

while (true) {
    resetOscPhase();

    setSinePhase(45);
    playWave(1,2, wm);

    incrementSinePhase(90);
    playWave(1,2, w);

    incrementSinePhase(90);
    playWave(1,2, w);

    incrementSinePhase(90);
    playWave(1,2, w);
}
```

Configure the scope according to the following settings.

Table 4.19: Settings: Configure the external scope

Scope Setting	Value / State
Ch1 enable	ON
Ch1 range	0.2 V/div
Ch2 enable	ON
Ch2 range	0.5 V/div
Timebase	10 ns/div
Trigger source	Ch2
Trigger level	200 mV
Run / Stop	ON

We also change the oscillator frequency to make it easier to visual the phase changes.

Table 4.20: Settings: enable the output

Tab	Section	Sub-Section	Label	Setting / Value / State
Digital Modulation	Channel 1 Oscillators	Frequency (Hz)	1	-500.0 M

Figure 4.20 shows the resulting signal. Three of the instantaneous phase increments of 90 degrees are visible as transient features. In a real use case, the phase changes usually occur in between pulses when the envelope signal is zero-valued, and these transients are then absent.

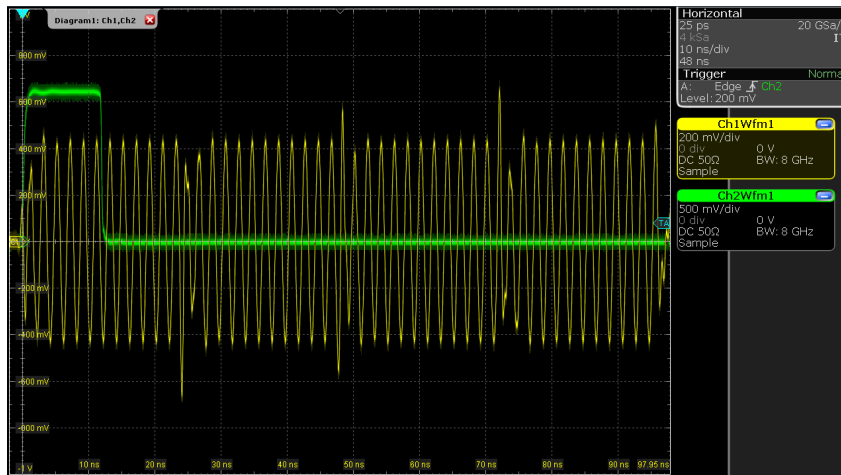


Figure 4.20: Amplitude-modulated dual-channel signal with rapid real-time phase increments generated by the SHFQC+.

Note

The phase increment due to the `incrementSinePhase` instruction takes effect at the end of the previous waveform playback. In case the instruction is placed in the sequencer code before the first `playWave` instruction, the phase increment will only happen after the `playWave` instruction.

Performing Frequency Sweeps

By using the sequencer commands `setOscFreq`, `configFreqSweep`, and `setSweepStep`, it is possible to set the oscillator frequency as part of a sequence and even perform frequency sweeps quickly while using a minimum number of sequencer instructions. Using these instructions, the oscillator frequency can be changed on a timescale of approximately 100 ns. The timing of the frequency update is deterministic. The waveforms that follow the frequency update will wait until the update has finished. A `waitWave` command after the waveform playback instructions is required to ensure that any subsequent frequency update does not happen during the waveform playback.

Note

The `setOscFreq`, `configFreqSweep`, and `setSweepStep` commands are intended to change the oscillator frequency between pulses. To sweep the frequency during a pulse, it's best to encode the frequency change in the waveform, e.g. using the `chirp` waveform generation function.

```
const START_FREQ = -100e6; //start frequency in Hz
const FREQ_INC = 200; //increment in Hz
const N_STEPS = 1e6; //number of frequency steps
const OSC = 0; //oscillator to sweep

const MEAS = 2048; //measurement window in samples

const LENGTH = 160; //length of pulse in samples

wave w = gauss(LENGTH, 1, LENGTH/2, LENGTH/8);
wave m_high = marker(LENGTH/2, 1); //marker high
wave m_low = marker(LENGTH/2, 0); //marker low
wave m = join(m_high, m_low); //join marker waveforms
wave wm = w + m; //combine marker and ones waveform data

//set up frequency sweep
configFreqSweep(OSC, START_FREQ, FREQ_INC);

var i;
for (i = 0; i < N_STEPS; i++) {
```

```

    setSweepStep(OSC,i);

    resetOscPhase();

    playWave(1,2, wm);
    playZero(MEAS);
    waitWave(); //to ensure setSweepStep does not execute during the play
instructions
}

```

Upload and run the above sequencer code on the AWG core of the Signal Generator channel 1. To make it easier to observe the frequency sweep on a scope, the length of the **MEAS** constant can be increased (e.g. with a measurement length of $2e8$ samples, the frequency will update every 10 ms).

Note

Multiple frequency sweeps can be configured in parallel, such that each oscillator of a given channel can be swept independently of the others.

4.1.5. Using the Python API

Note

This tutorial is applicable to all SHFQC+ Instruments.

Goals and Requirements

The previous tutorials showed how to use the SHFQC+ with the LabOne user interface. However, APIs provide an important alternative method to controlling the SHFQC+. In this tutorial, we focus on the Zurich Instruments Toolkit, showing how to use it to connect to the SHFQC+, as well as how to upload and play a sequence of the Signal Generator channel that uses user-defined waveforms. The Toolkit is based on the core Python API, **zhinst-core**. In this tutorial you will learn how to:

- connect to the instrument using Python
- control the Output, Modulation, and DIO settings of the instrument using nodes
- compile and upload a sequence using Python
- include user-defined waveforms in a sequence with Python

Preparation

Connect the cables as illustrated below. Make sure that the instrument is powered on and connected by Ethernet to your local area network (LAN) where the host computer resides. After starting LabOne, the default web browser opens with the LabOne graphical user interface.

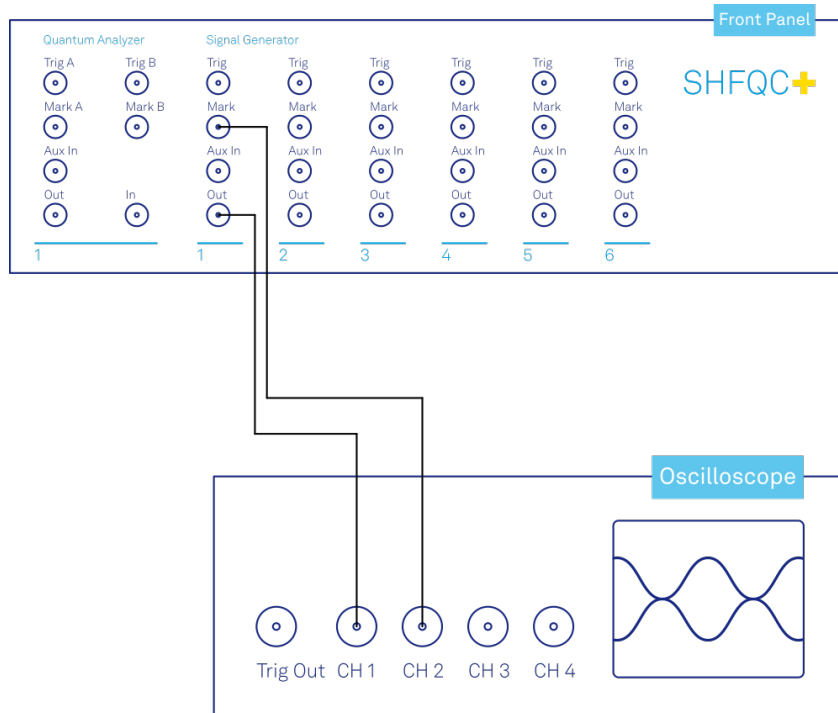


Figure 4.21: Connections for the arbitrary waveform generator Python tutorial

The tutorial can be started with the default instrument configuration (e.g. after a power cycle) and the default user interface settings (e.g. as is after pressing F5 in the browser).

Note

The instrument can also be connected via the USB interface, which can be simpler for a first test. As a final configuration for measurements, it is recommended to use the 1GbE interface, as it offers a larger data transfer bandwidth.

Connecting to to the instrument

Note

This tutorial makes use of the Zurich Instruments Toolkit. Setting a node in the Toolkit uses the format "device.path.to.node(value)." For the base Python API core, the equivalent node setting would be `daq.set(f'/{device_id}/path/to/node', value)`.

First we connect to the SHFQC+ using Python. For this we first create a session with the Zurich Instruments Toolkit and then connect to the instrument using the following code and by replacing **DEVXXXXX** with the id of our SHFQC+ instrument, e.g. **DEV12001**:

```
## Load the LabOne API and other necessary packages
from zhinst.toolkit import Session

DEVICE_ID = 'DEVXXXXX'
SERVER_HOST = 'localhost'

### connect to data server
session = Session(SERVER_HOST)

### connect to device
device = session.connect_device(DEVICE_ID)
```

Defining the data server allows users to connect to the instrument in the local network when using **localhost** or to specify a specific address, for example when a remote connection needs to be

established to the instrument. Remember that for a [remote connection](#), **Connectivity** needs to be set **From Everywhere**.

After successfully running the above code snippet, we check whether the Data Server, instrument firmware, and zhinst versions are compatible with each other:

```
device.check_compatibility()
```

If it does not throw an error, we are now in the position to access the device. If it returns an error, resolve the mismatched components identified in the error message.

Often the first parameters that need to be set for every experiment are the Center Frequency and Range of the Input or Output Channel. To see the parameter updates that will be performed by the Python script, open the [In/Out Tab](#)

of the GUI and select the **All**-subtab. In our Python script, we use the following code snippet to set the nodes for the Center Frequency of the Signal Generators Channel 1 to 6 GHz and the Output Range to 10 dBm.

```
SG_CHAN_INDEX=0
synth = device.sgchannels[SG_CHAN_INDEX].synthesizer()

rf_frequency = 6.0 # GHz
device.synthesizers[synth].centerfreq(rf_frequency*1e9)
output_range = 10.0
device.sgchannels[SG_CHAN_INDEX].output.range(output_range)
```

Note

When using the LF path, the corresponding node for setting the center frequency is **device.sgchannels[sg_chan_index].digitalmixer.centerfreq(value)**. This value can be set independently for each Signal Generator Channel.

Observe how the corresponding GUI values in the first panel of the Output tab change their values correspondingly. Note that in the SHFQC+, there are 4 synthesizers. Synthesizer 0 drives the Quantum Analyzer channel, whereas synthesizers 1 to 3 drive each drive two subsequent Signal Generator channels, i.e. 1&2, 3&4, or 5&6. Therefore it often makes sense to set the synthesizers using the

device.synthesizers[synth].centerfreq node and not within the **device.sgchannels[SG_CHAN_INDEX]** branch. To check which synthesizer is being used by a particular Signal Generator channel, you can query the node:

```
device.sgchannels[SG_CHAN_INDEX].synthesizer()
```

Note

Note that in the GUI and on the front panel of the instrument, lists (e.g. Channel numbers) always start at 1, but all representations in the APIs start counting at 0. Hence, Channel 1 on the front panel corresponds to **SG_CHAN_INDEX=0** in the API.

Note

To find out which node is linked to a specific setting in the GUI, either check out the command log at the bottom of the user interface or the [node tree documentation](#).

If we set an invalid value, e.g. a value of 6.05 GHz for the Center Frequency (note that this value can only be set in multiples of 100 MHz) through

```
device.synthesizers[synth].centerfreq(6.05*1e9)
```

then the instrument rounds the value automatically to the nearest possible value (here: 6.1 GHz). This is immediately indicated in the GUI or by querying the node:

```
device.synthesizers[synth].centerfreq()
```

In preparation for running a sequence in the next section, we will set several node values together using the API:

```
## Determine which synthesizer is used by the desired channel
synth = device.sgchannels[SG_CHAN_INDEX].synthesizer()

with device.set_transaction():
    # RF output settings
    device.sgchannels[SG_CHAN_INDEX].output.range(10)
# output range in dBm
    device.sgchannels[SG_CHAN_INDEX].output.rflfpath(1)           # use RF path,
not LF path
    device.synthesizers[synth].centerfreq(6.0e9)                 # synthesizer
frequency in Hz
    device.sgchannels[SG_CHAN_INDEX].output.on(1)                # enable output

    # Digital modulation settings
    device.sgchannels[SG_CHAN_INDEX].awg.outputamplitude(0.5)    # amplitude for
the AWG outputs
    device.sgchannels[SG_CHAN_INDEX].oscs[0].freq(10.0e6)        # frequency of
oscillator 1 in Hz
    device.sgchannels[SG_CHAN_INDEX].oscs[1].freq(-500e6)        # frequency of
oscillator 2 in Hz
    device.sgchannels[SG_CHAN_INDEX].awg.modulation.enable(1)    # enable
digital modulation

    # Trigger and marker settings
    device.sgchannels[SG_CHAN_INDEX].marker.source(4)            # use first
marker bit of waveform as marker source
```

Using these settings, we set the RF center frequency and output power, turn on the output, set up digital modulation settings for generating complex signals, and select the marker source for triggering the scope. We also use a transactional set, which is useful for setting many nodes at the same time. This method is faster than using a `daq.setInt` or `daq.setDouble` command for each node setting, because with a transactional set the communication latency has to be paid only once.

Uploading and running sequences

We now show how to upload a sequence via API. Very often, user-defined waveforms will be needed. We therefore also cover how to use custom waveforms in a sequence, as it is possible to load a waveform directly from the API. In the sequence the waveform should be declared using the `placeholder` function to define size and type of the waveform.

```
const LENGTH = 1024;
wave w = placeholder(LENGTH, true, false); // Create a waveform of size LENGTH,
with one marker
assignWaveIndex(1,2, w, 10);                // Create a wave table entry with
placeholder waveform, index 10

resetOscPhase();                            // Reset oscillator phase
playWave(1,2, w);                           // Play wave
```

We upload this sequence to the SHFQC+ using the following Python code:

```
## Define string that contains sequence from above
seqc_program = """\
const LENGTH = 1024;
wave w = placeholder(LENGTH, true, false); // Create a waveform of size LENGTH,
with one marker
assignWaveIndex(1,2, w, 10);                // Create a wave table entry with
placeholder waveform, index 10
```

```

resetOscPhase();           // Reset oscillator phase
playWave(1,2, w);         // Play wave
"""

## Upload the sequence
device.sgchannels[SG_CHAN_INDEX].awg.load_sequencer_program(seqc_program)

```

In addition to being able to set nodes, the Toolkit offers built-in functions for commonly performed actions, such as configuring the output and digital modulation settings as well as compiling and uploading sequences. The uploaded sequence will not run until a valid waveform has been loaded. This can be done for example in Python.

```

import numpy as np
from zhinst.toolkit import Waveforms

##Generate a waveform and marker
LENGTH = 1024
wave = np.exp(np.linspace(0, -5, LENGTH)) #exponentially decaying waveform
marker = np.concatenate([np.ones(32), np.zeros(LENGTH-32)]).astype(int) #marker
waveform with 32 samples high

## Upload waveforms
waveforms = Waveforms()
waveforms[10] = (wave, None, marker) # I-component wave, Q-component None, marker
device.sgchannels[SG_CHAN_INDEX].awg.write_to_waveform_memory(waveforms)

```

Now that we've uploaded both the sequence and the waveforms, we can run the sequence:

```

## Enable sequencer with single mode true
single = 1
device.sgchannels[SG_CHAN_INDEX].awg.enable_sequencer(single = single)

```

After running the sequence, we observe the signal shown in [Figure 4.22](#) on the scope.

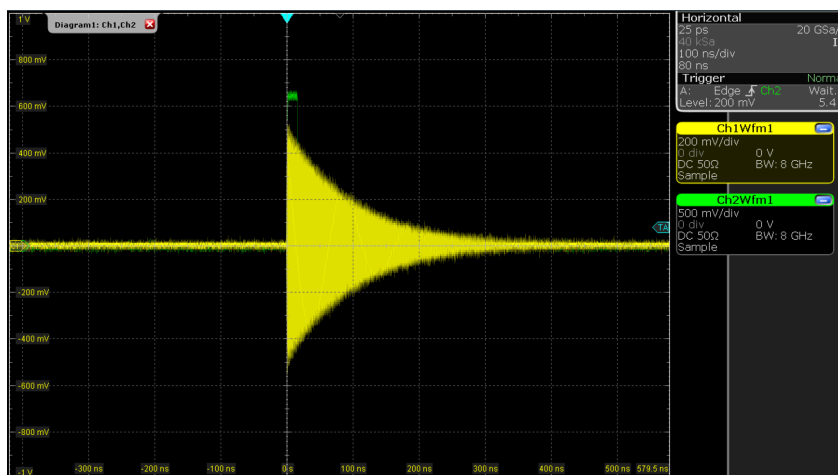


Figure 4.22: Waveform loaded by the API

The custom waveform data can be arbitrary, but consider that the final signal will pass through the analog output stage of the instrument where the signals get interpolated from 2 GSa/s to 6 GSa/s. This means that the signal may not correspond exactly to the programmed waveform. In particular, this concerns sharp transitions from one sample to the next.

Depending on the output channel assignment (the optional first arguments of `assignWaveIndex` and `playWave` instructions), the AWG compiler may create implicit waveform table entries. Therefore, we recommend a usage of the instructions `placeholder`, `assignWaveIndex`, and `playWave` that is as explicit as possible. The following code, for example, is valid but not recommended because it is not easy to read:

```

const LENGTH = 1024;
wave w = placeholder(LENGTH);
assignWaveIndex(1, w, 10);
assignWaveIndex(w, w, 11);

```



```
playWave(1, w);
playWave(w, w);
```

Instead, it's recommended to use a unique waveform variable name for each intended single-channel memory entry, and to use this variable name with consistent output channel assignment in `placeholder`, `assignWaveIndex`, and `playWave` as is done in the following example:

```
const LENGTH = 1024;
wave w_a = placeholder(LENGTH, true, false); // Allocate a waveform with one
marker
wave w_b = placeholder(LENGTH, true, false); // Allocate a waveform with one
marker
wave w_c = placeholder(LENGTH, false, false); // Allocate a waveform without
markers
assignWaveIndex(1, 2, w_a, 10); // Declare a single-channel
waveform w_a, slot 10
assignWaveIndex(1, 2, w_b, 1, 2, w_c, 11); // Declare a dual-channel
waveform with w_b and w_c respectively as real and imaginary part, slot 11

playWave(1, 2, w_a); // Play a single channel waveform
(only amplitude modulation)
playWave(1, 2, w_b, 1, 2, w_c); // Play a dual channel waveform
(full IQ modulation)
```

In the latter case, a possible Python code to update the wave table is shown below. Note that we use the full amount of markers available in the instrument, one per physical channel. The marker integer array encodes the available markers in its least significant bit.

```
##Generate a waveform and marker
LENGTH = 1024
wave_a = np.exp(np.linspace(0, -5, LENGTH))
wave_b = np.exp(np.linspace(0, -15, LENGTH))
wave_c = np.exp(np.linspace(0, -2.5, LENGTH))

marker_a = np.concatenate([np.ones(32), np.zeros(LENGTH-32)]).astype(int)
marker_bc = np.concatenate([np.ones(32), np.zeros(LENGTH-32)]).astype(int)

##Convert and send them to the instrument
waveforms = Waveforms()
waveforms[10] = (wave_a, None, marker_a)
waveforms[11] = (wave_b, wave_c, marker_bc)
device.sgchannels[SG_CHAN_INDEX].awg.write_to_waveform_memory(waveforms)
```

4.1.6. Pulse-level Sequencing with the Command Table

Note

This tutorial is applicable to all SHFQC+ Instruments.

Goals and Requirements

Pulse-level sequencing is an efficient way to encode pulses in a sequence by uploading a minimal amount of information to the device, allowing measurements to be performed more quickly and programmed more intuitively. The goal of this tutorial is to demonstrate pulse-level sequencing using the command table feature of the Signal Generator channels of the SHFQC+.

Preparation

Connect the cables as illustrated below. Make sure that the instrument is powered on and connected by Ethernet to your local area network (LAN) in which the control computer resides. After starting LabOne, the default web browser opens with the LabOne graphical user interface.

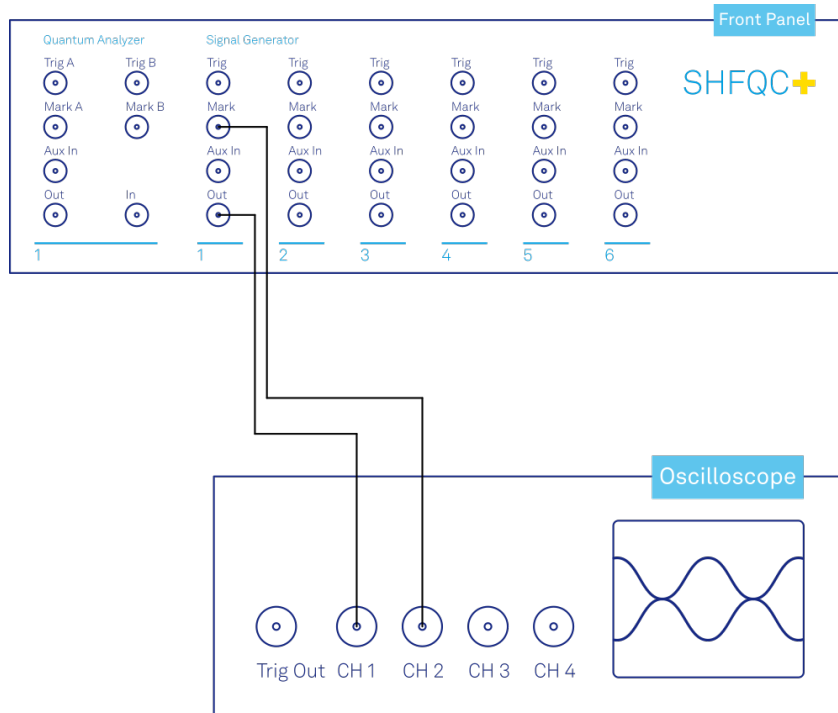


Figure 4.23: Connections for the arbitrary waveform generator command table tutorial

The tutorial can be started with the default instrument configuration (e.g. after a power cycle) and the default user interface settings (e.g. after pressing F5 in the browser). Additionally, this tutorial requires the use of one of our APIs, in order to be able to define and upload the command table itself. The examples shown here use the Python API - for an introduction see also the [Python tutorial](#). Similar functionality is also available for other APIs.

Note

The instrument can also be connected via the USB interface, which can be simpler for a first test. As a final configuration for measurements, it is recommended to use the 1GbE interface, as it offers a larger data transfer bandwidth.

Configure the Output

Note

This tutorial makes use of the Zurich Instruments Toolkit. Setting a node in the Toolkit uses the format "device.path.to.node(value)." For the base Python API core, the equivalent node setting would be `daq.set(f'/{DEVICE_ID}/path/to/node', value)`.

Note

The minimum waveform length when using the command table is 16 samples.

To begin with, we configure the output and digital modulation settings of the SHFQC+, to be able to observe our signal on a scope. We use the Zurich Instruments Toolkit, available in Python, to set the corresponding nodes after connecting to the instrument. The code below establishes a connection to the device before setting the node values (see also the [Using the Python API Tutorial](#)).

```

## Load the LabOne API and other necessary packages
from zhinst.toolkit import Session, CommandTable

DEVICE_ID = 'DEVXXXXX'
SERVER_HOST = 'localhost'

### connect to data server
session = Session(SERVER_HOST)

### connect to device
device = session.connect_device(DEVICE_ID)

SG_CHAN_INDEX = 0 # which channel to be used, here: first channel

##determine which synthesizer is used by the desired channel
synth = device.sgchannels[SG_CHAN_INDEX].synthesizer()

with device.set_transaction():
    # RF output settings
    device.sgchannels[SG_CHAN_INDEX].output.range(10) #output range in dBm
    device.sgchannels[SG_CHAN_INDEX].output.rflpath(1) #use RF path, not LF path
    device.synthesizers[synth].centerfreq(1.0e9) #set the corresponding
    synthesizer frequency in Hz
    device.sgchannels[SG_CHAN_INDEX].output.on(1) #enable output
    # Digital modulation settings
    device.sgchannels[SG_CHAN_INDEX].awg.outputamplitude(0.5) #set the amplitude
    for the AWG outputs
    device.sgchannels[SG_CHAN_INDEX].oscs[0].freq(10.0e6) #frequency of
    oscillator 1 in Hz
    device.sgchannels[SG_CHAN_INDEX].oscs[1].freq(-150.0e6) #frequency of
    oscillator 2 in Hz
    device.sgchannels[SG_CHAN_INDEX].awg.modulation.enable(1) #enable digital
    modulation
    # Triggering settings
    device.sgchannels[SG_CHAN_INDEX].marker.source(0) #AWG trigger 1

```

In this case, we will use Signal Generator output channel 1 with a maximum output power of 10 dBm and an RF center frequency of 1.0 GHz. We will also enable digital modulation using an oscillator frequency of 10 MHz. This will yield a final output frequency of 1.01 GHz after configuring upper sideband modulation with the command table later. The amplitude of the AWG outputs is set to 0.5 to avoid saturating the outputs.

Introduction to the Command Table

The command table allows the sequencer to group waveform playback instructions with other timing-critical phase and amplitude setting commands into a single instruction that executes within one sequencer clock cycle of 4 ns. The command table is a unit separate from the sequencer and waveform memory and can thus be exchanged separately. Both the phase and the amplitude can be set in absolute and in incremental modes. Additionally, the active oscillator can be set with the command table, enabling fast, phase-coherent frequency switching on a given output channel. Even when not using digital modulation or amplitude settings, working with the command table has the advantage of being more efficient in sequencer instruction memory compared to standard sequencing. Starting a waveform playback with the command table always requires just a single sequencer clock cycle, as opposed to 2 or 3 when using a **playWave** instruction.

When using the command table, three components play together during runtime to generate the waveform output and apply the phase and amplitude setting instructions:

- Sequencer: the unit executing the runtime instructions, namely in this context the **executeTableEntry** instruction. This instruction executes one entry of the command table, and its input argument is a command table index. In its compiled form, which can be seen in the AWG Advanced sub-tab, the sequence program can contain up to 32768 instructions.
- Wave table: a list of up to 16000 indexed waveforms. This list is defined by the sequence program using the index assignment instruction **assignWaveIndex** combined with a waveform or waveform placeholder. The wave table index referring to a waveform can be used in two ways: it is

referred to from the command table, and it is used to directly write waveform data to the instrument memory using the node `<DEVICE_ID>/SGCHANNELS/<SG_CHAN_INDEX>/AWG/WAVEFORM/WAVES/<WAVE_INDEX>` [Node Documentation](#)

- Command table: a list of up to 4096 indexed entries (command table index), each containing the index of a waveform to be played (wave table index), a sine generator phase setting, a set of four AWG amplitude settings for complex modulation, and an oscillator index selection. The command table is specified by a JSON formatted string written to the node `<DEVICE_ID>/SGCHANNELS/<SG_CHAN_INDEX>/AWG/COMMANDTABLE/DATA`

Basic command table use

We start by defining a sequencer program that uses the command table.

```
seqc_program = """\
// Define waveform
wave w_a = gauss(2048, 1, 1024, 256);

// Assign a single channel waveform to wave table entry 0
assignWaveIndex(1,2, w_a, 0);

// Reset the oscillator phase
resetOscPhase();

// Trigger the scope
setTrigger(1);
setTrigger(0);

// execute the first command table entry
executeTableEntry(0);
// execute the second command table entry
executeTableEntry(1);
"""

## Upload sequence
device.sgchannels[SG_CHAN_INDEX].awg.load_sequencer_program(seqc_program)
```

The sequence can be compiled and uploaded via API using the methods shown in the [Python API Tutorial](#). The sequence defines a Gaussian pulse of unit amplitude and length of 2048 samples. This waveform is then assigned as a dual-channel waveform with explicit output assignment to the wave table entry with index 0, and the final lines execute the two first command table entries. This program cannot be run yet, as the command table is not yet defined.

Note

If a sequence program contains a reference to a command table entry that has not been defined, or if a command table entry refers to a waveform that has not been defined, the sequence program can't be run.

In general the command table is defined as a JSON formatted string. Below, we show an example of how to define a command table with two table entries using Python. For ease of programming, here we define the command table as a **CommandTable** object, which is converted into a JSON string automatically at upload. Such object also validate the fields of the command table.

```
## Load CommandTable class
from zhinst.toolkit import CommandTable

## Initialize command table
ct_schema = device.sgchannels[SG_CHAN_INDEX].awg.commandtable.load_validation_schema()
ct = CommandTable(ct_schema)

## Index of wave table and command table entries
TABLE_INDEX = 0
```

```

WAVE_INDEX = 0
gain = 1.0

## Waveform with amplitude and phase settings
ct.table[TABLE_INDEX].waveform.index = WAVE_INDEX
ct.table[TABLE_INDEX].amplitude00.value = gain
ct.table[TABLE_INDEX].amplitude01.value = -gain
ct.table[TABLE_INDEX].amplitude10.value = gain
ct.table[TABLE_INDEX].amplitude11.value = gain
ct.table[TABLE_INDEX].phase.value = 0

## Same waveform with different amplitude and phase settings
ct.table[TABLE_INDEX+1].waveform.index = WAVE_INDEX
ct.table[TABLE_INDEX+1].amplitude00.value = gain/2
ct.table[TABLE_INDEX+1].amplitude01.value = -gain/2
ct.table[TABLE_INDEX+1].amplitude10.value = gain/2
ct.table[TABLE_INDEX+1].amplitude11.value = gain/2
ct.table[TABLE_INDEX+1].phase.value = 180

```

In this example, we generate a first command table entry with index "TABLE_INDEX", which plays the waveform referenced in the wave table at index "WAVE_INDEX", with amplitude and phase settings specified. The four amplitude settings of the command table have the same effect as the four gain settings of the [Digital Modulation Tutorial](#), with analogous naming convention, i.e. **amplitude01** maps to **Gain01**. The signs of the amplitudes are chosen to yield upper sideband modulation when using a positive oscillator frequency.

Note

Here we use a single-channel waveform, since we modulate only the amplitude of our pulses. Therefore, coefficients **amplitude01** and **amplitude11** are not strictly needed. We left them here and in the following examples to show how to use it even with dual-channel waveforms.

To upload the command table to the Signal Generator channel of the SHFQC+, we need to connect to the device and then write the command table to the correct node. In Python, this is achieved as follows:

```

## Upload command table
device.sgchannels[SG_CHAN_INDEX].awg.commandtable.upload_to_device(ct)

```

Note

During compilation of a sequencer program, any previously uploaded command table is reset, and will need to be uploaded again before it can be used.

Now that we've uploaded both the sequence and the command table, we can run the sequence:

```

device.sgchannels[SG_CHAN_INDEX].awg.enable_sequencer(single = True)

```

The expected output is shown in [Figure 4.24](#). Note how the amplitude of the second waveform is half the magnitude of the first waveform, and that there is a phase shift of 180 degrees between them. This is due to the amplitude and phase settings in the command table. **Also note that these amplitude settings are persistent.** If a value is not explicitly specified in the command table, it uses either the default value or the value set by a previous usage of the 'executeTableEntry' instruction.

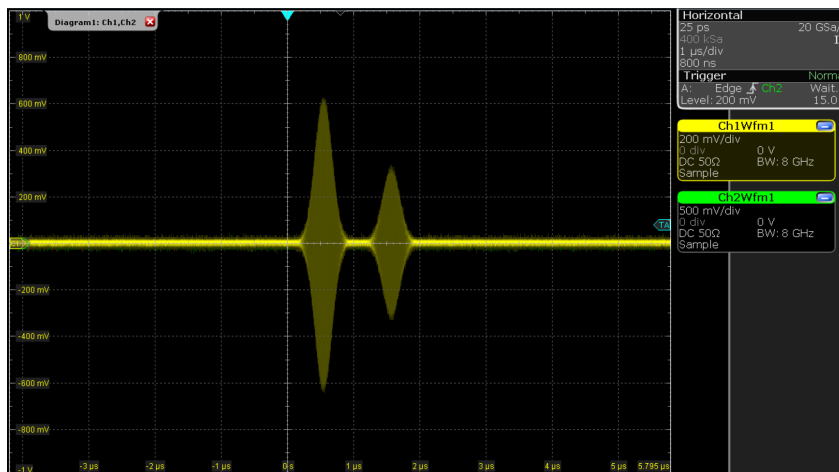


Figure 4.24: Output of the first channel from the basic command table example

Note

When a command table entry is called, the amplitude and phase are set persistently. Subsequent waveform playbacks on the same channel will need to take this into account, unless the amplitude and phase settings are explicitly included for them in their corresponding command table entries. Additionally, the values of the command table amplitude and phase settings take precedence over the corresponding gain and phase node settings set via API or in the LabOne UI, e.g. the value of **Gain01** will have no effect if **amplitude01** is specified in the command table entry.

Efficient pulse incrementation

One illustrative use case of the command table feature is the efficient incrementation of the amplitude or phase of a waveform.

We again start by writing a sequencer program that plays two entries of the command table.

```
seqc_program = """\
// Define a single waveform
wave w_a = ones(1024);

// Assign a single channel waveform to wave table entry
assignWaveIndex(1,2, w_a, 0);

// Reset the oscillator phase
resetOscPhase();

// Trigger the scope
setTrigger(1);
setTrigger(0);

// execute the first command table entry
executeTableEntry(0);
repeat(20) {
    executeTableEntry(1);
}
"""

## Upload sequence
device.sgchannels[SG_CHAN_INDEX].awg.load_sequencer_program(seqc_program)
```

Here we have defined a single wave table entry, where both channels contain the same constant waveform.

In Python we then define a command table with just two entries, in this case both referencing the same waveform index. In the second command table entry, we set the **increment** field to **true**, such

that the amplitude is incremented each time that the second command table entry is called in the sequence.

```
## Initialize command table
ct_schema = device.sgchannels[SG_CHAN_INDEX].awg.commandtable.load_validation_schema()
ct = CommandTable(ct_schema)

## Waveform with initial amplitude
ct.table[0].waveform.index = 0
ct.table[0].amplitude00.value = 0
ct.table[0].amplitude01.value = 0
ct.table[0].amplitude10.value = 0
ct.table[0].amplitude11.value = 0

## Waveform with incremented amplitude
ct.table[1].waveform.index = 0
ct.table[1].amplitude00.value = 0.05
ct.table[1].amplitude01.value = -0.05
ct.table[1].amplitude10.value = 0.05
ct.table[1].amplitude11.value = 0.05
ct.table[1].amplitude00.increment = True
ct.table[1].amplitude01.increment = True
ct.table[1].amplitude10.increment = True
ct.table[1].amplitude11.increment = True

## Upload command table
device.sgchannels[SG_CHAN_INDEX].awg.commandtable.upload_to_device(ct)
## Enable sequencer
device.sgchannels[SG_CHAN_INDEX].awg.enable_sequencer(single = 1)
```

After uploading the command table to the instrument and executing the sequencer program, the channel then produces the output shown in Figure 4.25. Here, the first call to the first command table entry plays the waveform with all amplitude settings set to 0. The subsequent calls to the second command table entry increment these amplitudes each time by 0.05, with a negative increment on **amplitude01**, and a positive increment on the others. Although in this example we increment all amplitudes together, it is possible to increment only a subselection of the amplitude settings as well, by changing the appropriate increment settings to False. Incrementing amplitudes this way enables waveform memory-efficient amplitude sweeps.

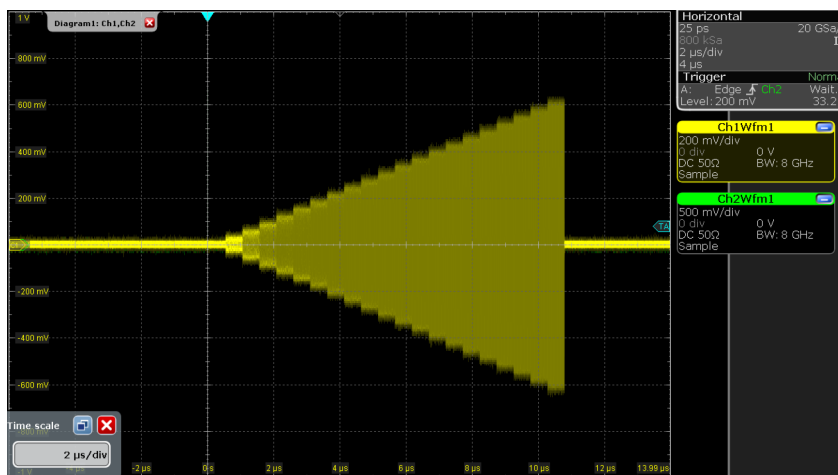


Figure 4.25: Incrementing waveform amplitudes using the command table increment functionality

Note

The amplitude of the waveform generated at the output can be influenced in several different ways: through the amplitude of the waveform itself, through the amplitude settings in the command table, through the output amplitude setting in the [Modulation Tab](#), and finally through the Range setting of the SHFQC+ Signal Generator output channel.

It is possible to perform multi-dimensional amplitude sweeps by making use of the amplitude registers of the command table. Each channel has four independent amplitude registers (indexed [0...3]), with each register storing the amplitude last played on that register. By default, amplitude register with index zero is used. It is therefore possible to keep the amplitude of one register constant while sweeping the amplitude of another register. This can be useful for probing dynamics in a multi-level system.

As an example, we will use the following sequence:

```
seqc_program = ""\
//Constant definitions
const readout = 512; //length of readout in samples

//Waveform definition
wave wI1 = gauss(128, 1, 64, 16);
wave wI2 = gauss(256, 1, 128, 32);

//Assign index and outputs
assignWaveIndex(1,2,wI1,0);
assignWaveIndex(1,2,wI2,1);

var i = 10;
executeTableEntry(0);
do {
    executeTableEntry(2);
    executeTableEntry(1);
    playZero(readout);
    i-=1;
} while(i);
""

# Upload sequence
device.sgchannels[SG_CHAN_INDEX].awg.load_sequencer_program(seqc_program)
```

The first `executeTableEntry` command initializes the amplitude that will be swept without playing a pulse. The second `executeTableEntry` plays a constant-amplitude Gaussian pulse (128 samples long). The third `executeTableEntry` plays a different Gaussian pulse (256 samples long), the amplitude of which will be swept. The loop will play 10 different amplitudes. We also need to define and upload a command table to go with the sequence:

```
# Initialize command table
ct_schema = device.sgchannels[0].awg.commandtable.load_validation_schema()
ct = CommandTable(ct_schema)

# Initialize amplitude register 1
ct.table[0].amplitude00.value = 0.0
ct.table[0].amplitude00.increment = False
ct.table[0].amplitude10.value = 0.0
ct.table[0].amplitude10.increment = False
ct.table[0].amplitudeRegister = 1

# Swept Gaussian pulse
ct.table[1].waveform.index = 1
ct.table[1].amplitude00.value = 0.05
ct.table[1].amplitude00.increment = True
ct.table[1].amplitude10.value = 0.05
ct.table[1].amplitude10.increment = True
ct.table[1].amplitudeRegister = 1

# Constant Gaussian pulse
ct.table[2].waveform.index = 0
ct.table[2].amplitude00.value = 0.9
ct.table[2].amplitude10.value = 0.9
ct.table[2].amplitudeRegister = 0
```


Upload command table

```
device.sgchannels[SG_CHAN_INDEX].awg.commandtable.upload_to_device(ct)
```

The first command table entry (index 0) sets the initial amplitude (in this case, 0.0) of amplitude register 1. The second table entry (index 1) increments the amplitude of amplitude register 1 and plays the Gaussian pulse with waveform index 1. The third table entry (index 2) plays the constant-amplitude Gaussian pulse (waveform index 0) using amplitude register 0.

We now run the sequence:

```
device.sgchannels[SG_CHAN_INDEX].awg.enable_sequencer(single = 1)
```

We observe the signal shown in the figure below, which shows a constant-amplitude Gaussian pulse interleaved with a Gaussian pulse whose amplitude is swept. In total, there are 10 different amplitudes of the swept pulse.

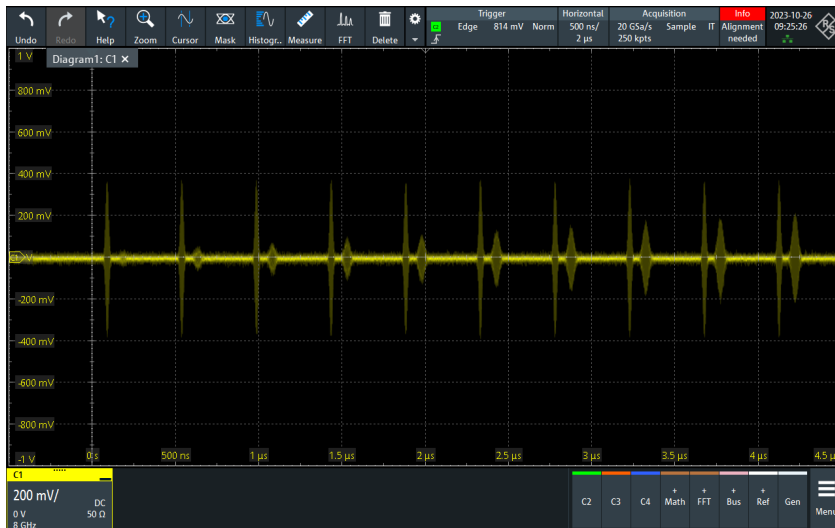


Figure 4.26: Using the amplitude registers to sweep the amplitude of one pulse while keeping the amplitude of another constant.

Phase sweeps can be achieved in a similar way by using the command table below.

Define command table**## Initialize command table**

```
ct_schema = device.sgchannels[SG_CHAN_INDEX].awg.commandtable.load_validation_schema()
```

```
ct = CommandTable(ct_schema)
```

Waveform with initial phase

```
ct.table[0].waveform.index = 0
```

```
ct.table[0].phase.value = 90
```

Waveform with incremented phase

```
ct.table[1].waveform.index = 0
```

```
ct.table[1].phase.value = 0.1
```

```
ct.table[1].phase.increment = True
```

Upload command table

```
device.sgchannels[SG_CHAN_INDEX].awg.commandtable.upload_to_device(ct)
```

Enable sequencer

```
device.sgchannels[SG_CHAN_INDEX].awg.enable_sequencer(single = 1)
```

In this case, executing the first table entry will set the phase to 90 degrees, and the second table entry will increment this value each time it is called in steps of 0.1 degrees.

Pulse-level sequencing with the command table

All previous examples have used the pulse library in the AWG sequencer to define waveforms. In more advanced scenarios, waveforms are uploaded through the API, as we will demonstrate next.

We start with the following sequence program, where we assign wave table entries using the placeholder command with a waveform length as argument.

```
seqc_program = """\
// Define two wave table entries through placeholders
assignWaveIndex(1,2, placeholder(32), 0);
assignWaveIndex(1,2, placeholder(64), 1);

// Reset the oscillator phase
resetOscPhase();

// Trigger the scope
setTrigger(1);
setTrigger(0);

// execute command table
executeTableEntry(0);
executeTableEntry(1);
executeTableEntry(2);
"""

## Upload sequence
device.sgchannels[SG_CHAN_INDEX].awg.load_sequencer_program(seqc_program)
```

In this form, the sequence program cannot be run, first because the command table is not yet uploaded, and second because the waveform memory in the wave table has not been defined. We can use the numpy package to define complex-valued Gaussian waveforms directly in Python, and upload them to the instrument using the appropriate node.

```
import numpy as np
from zhinst.toolkit import Waveforms

## parameters for waveform generation
amp_1 = 1
length_1 = 32
width_1 = 1/4
amp_2 = 1
length_2 = 64
width_2 = 1/4
x_1 = np.linspace(-1, 1, length_1)
x_2 = np.linspace(-1, 1, length_2)

## define waveforms as list of real-values arrays - here: Gaussian functions
waves = [
    [amp_1*np.exp(-x_1**2/width_1**2)],
    [amp_2*np.exp(-x_2**2/width_2**2)]]

## upload waveforms to instrument
waveforms = Waveforms()
for i, wave in enumerate(waves):
    waveforms[i] = (wave[0])

device.sgchannels[SG_CHAN_INDEX].awg.write_to_waveform_memory(waveforms)
```

Finally, we also generate and upload a command table to the instrument.

```
## Define command table
## Initialize command table
ct_schema = device.sgchannels[SG_CHAN_INDEX].awg.commandtable.load_validation_schema()
ct = CommandTable(ct_schema)

## Waveform 0 with oscillator 1
ct.table[0].waveform.index = 0
ct.table[0].amplitude00.value = 1.0
```

```

ct.table[0].amplitude01.value = -1.0
ct.table[0].amplitude10.value = 1.0
ct.table[0].amplitude11.value = 1.0
ct.table[0].phase.value = 0.0
ct.table[0].oscillatorSelect.value = 0

## Waveform 1 with oscillator 2
ct.table[1].waveform.index = 0
ct.table[1].amplitude00.value = 1.0
ct.table[1].amplitude01.value = -1.0
ct.table[1].amplitude10.value = 1.0
ct.table[1].amplitude11.value = 1.0
ct.table[1].phase.value = 0.0
ct.table[1].oscillatorSelect.value = 1

## Waveform 1 with oscillator 1 and different phase
ct.table[2].waveform.index = 0
ct.table[2].amplitude00.value = 1.0
ct.table[2].amplitude01.value = -1.0
ct.table[2].amplitude10.value = 1.0
ct.table[2].amplitude11.value = 1.0
ct.table[2].phase.value = 90.0
ct.table[2].oscillatorSelect.value = 0

## Upload command table
device.sgchannels[SG_CHAN_INDEX].awg.commandtable.upload_to_device(ct)
## Enable sequencer
device.sgchannels[SG_CHAN_INDEX].awg.enable_sequencer(single = 1)

```

Running the sequencer program will produce output as shown in Figure 4.27.

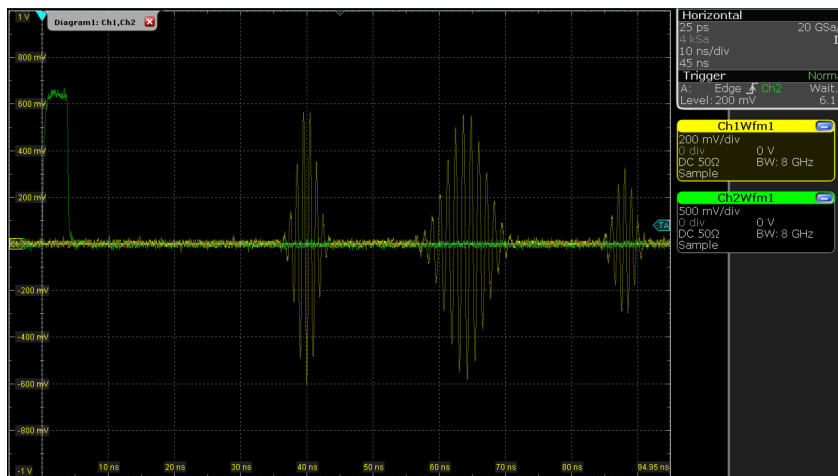


Figure 4.27: Advanced command table example output, including oscillator selection

The first command table entry plays a Gaussian pulse with amplitude settings for upper sideband modulation, a phase of 0 degrees, and using oscillator 1 (at 10 MHz). The second command table entry plays a different Gaussian pulse envelope with similar amplitude and phase settings, but now using oscillator 2 (at 500 MHz, leading to an output frequency of 500 MHz). The third and final command table entry plays the first Gaussian pulse envelope with different amplitude and phase settings, but again using oscillator 1. Such a set of pulses could correspond to playing an X-gate on qubit 1, then an X-gate on qubit 2, then a Y/2-gate on qubit 1 again. Using the **oscillatorSelect** field thereby allows users to interleave pulses for different qubits while maintaining phase coherence between oscillator switches. Because each channel has 8 oscillators, this allows gates for up to 8 different qubits or transitions to be interleaved on the same RF line.

It is also possible to define a command table entry that changes parameters without playing a waveform. This can be particularly useful for efficient nested loops, e.g. Rabi amplitude sweeps with cyclic or sequential averaging. Furthermore, it is possible to define a **playZero** (and other waveforms) from within the command table as well. To see this functionality, upload the following sequence:

```

seqc_program = ""
// Define waveform

```

```

const len = 1024;
const amp = 1;
wave w = gauss(len,amp,len/2,len/8);

// Assign waveform index
assignWaveIndex(1,2, w, 0);

// Reset the oscillator phase
resetOscPhase();

// Trigger the scope
setTrigger(1);
setTrigger(0);

executeTableEntry(0); //set initial parameters
repeat (5) {
    executeTableEntry(1); //play waveform
    executeTableEntry(2); //playZero
    executeTableEntry(3); //set different parameters
}
"""

## Upload sequence
device.sgchannels[SG_CHAN_INDEX].awg.load_sequencer_program(seqc_program)

```

After uploading the sequence, we upload the following command table as well:

```

## Initialize command table
ct_schema = device.sgchannels[SG_CHAN_INDEX].awg.commandtable.load_validation_schema()
ct = CommandTable(ct_schema)

## Initial amplitude and phase settings
ct.table[0].amplitude00.value = 0.1
ct.table[0].amplitude01.value = -0.1
ct.table[0].amplitude10.value = 0.1
ct.table[0].amplitude11.value = 0.1
ct.table[0].phase.value = 0.0

## Play waveform
ct.table[1].waveform.index = 0

## Play playZero
ct.table[2].waveform.playZero = True
ct.table[2].waveform.length = 32

## Set new parameters
ct.table[3].amplitude00.value = 0.05
ct.table[3].amplitude00.increment = True
ct.table[3].amplitude01.value = -0.05
ct.table[3].amplitude01.increment = True
ct.table[3].amplitude10.value = 0.05
ct.table[3].amplitude10.increment = True
ct.table[3].amplitude11.value = 0.05
ct.table[3].amplitude11.increment = True

## Upload command table
device.sgchannels[SG_CHAN_INDEX].awg.commandtable.upload_to_device(ct)
## Enable sequencer
device.sgchannels[SG_CHAN_INDEX].awg.enable_sequencer(single = 1)

```

The above combination of sequence and command table will use the first `executeTableEntry` command (table index 0) to set initial amplitude and phase parameters without playing a waveform. The second `executeTableEntry` command (table index 1) plays a waveform using the parameters

set by the previous command. The third **executeTableEntry** plays a **playZero** of length 32 samples. The fourth **executeTableEntry** (table index 3) sets new parameters without playing a waveform. Because of the repeat loop, the sequence will play the pulse 5 times, each time with a different set of parameters. In total, we play a waveform with 5 different sets of parameters, but we need only two command table entries (table indices 0 and 3) to set the parameters and one entry to play the waveform (table index 1). We would still need only these three table entries (four including the **playZero**) even if we want to do a parameter sweep of 100 or 1000 different values (e.g. with **repeat (100)**).

Note

The benefit of using **playZero** and **playHold** from within the command table is that they will map to a single assembly instruction. Alternatively, the instructions **playZero** and **playHold** can be used directly in the sequencer without the command table and still map to a single instruction, if the following condition are fulfilled: - Length argument less than 1 MSa - Sample rate argument is left empty or set to **AWG_RATE_2000MHZ** (the default value)

It is better to use the command table in the case the criteria above are not fulfilled, or for minimal play length of 16 samples, or if the command table is randomly accessed in real-time with a variable.

Command table entries fields

The documentation of all possible parameters in the command table JSON file can be found by pulling the schema from the device itself using the node `</dev>/SGCHANNELS/<n>/AWG/COMMANDTABLE/SCHEMA`. The Python **CommandTable** object automatically uses the schema from the device when initialized like this:

```
## Initialize command table
ct_schema = awg.commandtable.load_validation_schema()
ct = CommandTable(ct_schema)
```

Table 4.21 contains all elements that can be programmed as part of a command table entry as well as the default value which is applied if this element is not specified by the user. Table 4.22 contains all parameters of a **waveform** element, as well as each parameter's default value. Analogously, Table 4.24 contains the parameters of a phase type element (**phase**), Table 4.25 those of an amplitude type entry (**amplitude00**, **amplitude01**, **amplitude10** or **amplitude11**) and Table 4.23 contains the oscillator selector (**oscillatorSelect**).

If a phase element is specified in any entry of the command table, the absolute phase will be set to zero at the start of the execution.

Table 4.21: Elements of a command table entry

Field	Description	Type	Range/ Value	Mandatory	Default
index	Index of the entry	Integer	[0—4095]	yes	mandatory
waveform	Waveform command and its properties	Waveform		no	No waveform played
oscillatorSelect	Oscillator used for the modulation	Oscillator Select		no	No change of oscillator
phase	Phase command of the modulation	Phase		no	No change to phase setting
amplitude00	Amplitude command for AWG output gain00	Amplitude		no	No change to amplitude setting
amplitude01	Amplitude command for AWG output gain01	Amplitude		no	No change to amplitude setting
amplitude10	Amplitude command for AWG output gain10	Amplitude		no	No change to amplitude setting

Field	Description	Type	Range/Value	Mandatory	Default
amplitude11	Amplitude command for AWG output gain11	Amplitude		no	No change to amplitude setting

Table 4.22: Parameters of the Waveform element of a command table entry

Field	Description	Type	Range/Value	Mandatory	Default
index	Index of the waveform to play as defined with the assignWaveIndex sequencer instruction	integer	[0—15999]	if playZero or playHold is False	No waveform played
length	The length of the waveform in samples	integer	[16—WFM_LEN]	if playZero or playHold is True	the waveform length as declared in the sequence
samplingRateDivider	Integer exponent n of the sampling rate divider: $\text{SampleRate} / 2^n$	integer	[0—13]	no	0
playZero	Play a zero-valued waveform for specified length of waveform	bool	[True,False]	no	False
playHold	Hold the value of the last waveform and marker sample played for specified length	bool	[True,False]	no	False

Table 4.23: Parameters of a Oscillator Select element of a command table entry

Field	Description	Type	Range/Value	Mandatory	Default
value	Index of oscillator that is selected for sine/cosine generation	integer	[0—7]	Yes	mandatory

Table 4.24: Parameters of a Phase element of a command table entry

Field	Description	Type	Range/Value	Mandatory	Default
value	Phase value of the given sine generator in degree	float	[-180.0—180.0) values outside of this range will be clamped	Yes	mandatory
increment	Set to true for incremental phase value, or to false for absolute	bool	[True,False]	No	False

Table 4.25: Parameters of an Amplitude element of a command table entry

Field	Description	Type	Range/Value	Mandatory	Default
value	Amplitude scaling factor of the given AWG channel	float	[-1.0—1.0]	Yes	mandatory
increment	Set to true for incremental amplitude value, or to false for absolute	bool	[True,False]	No	False

4.2. Quantum Analyzer Tutorials

The tutorials in this subchapter have been created to allow users to become more familiar with the Quantum Analyzer Readout Channel of the SHFQC+.

Note

In the tutorials, we use both the General User Interface and the Python API to control the instrument.

Note

For all tutorials, you must have LabOne installed as described in the chapter [Getting Started](#).

Note

This chapter is constantly being upgraded and new documentation is added. For the latest version of the documentation, please always refer to the [online documentation](#).

4.2.1. Resonator Spectroscopy

Note

This tutorial is applicable to all SHFQC+ Instruments.

Goals and Requirements

[LabOne Q](#) is the recommended control software to operate the SHFQC+ for Quantum Technology applications.

The goal of this tutorial is to demonstrate how to use the SHFQC+ to perform resonator spectroscopy measurement using the Quantum Analyzer readout channel with the LabOne User Interface (UI) and [Zurich Instruments Toolkit API](#).

Note

For zhinst-toolkit users, please find the example in https://github.com/zhinst/zhinst-toolkit/blob/main/examples/shfqc_resonator_spectroscopy_cw.md, and the zhinst-toolkit documentation.

For LabOne Q users, please find the example in https://github.com/zhinst/laboneq/blob/main/examples/01_qubit_characterization/01_cw_resonator_spec_shfsg_shfqa_shfqc.ipynb and https://github.com/zhinst/laboneq/blob/main/examples/01_qubit_characterization/02_pulsed_resonator_spec_shfsg_shfqa_shfqc.ipynb, and the LabOne Q User Manual.

Preparation

Please follow the preparation steps in [Connecting to the Instrument](#) and connect the instrument in a loopback configuration as shown in [Figure 4.28](#) or to a device under test.

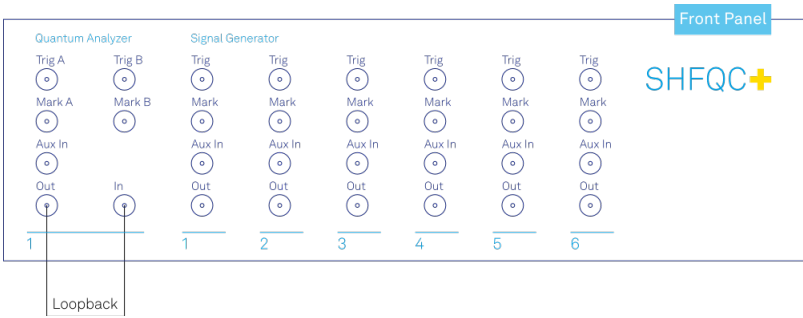


Figure 4.28: SHFQC+ connection.

Tutorial

In this tutorial, we sweep frequency of output signal and measure the frequency response of the cable or the device under test.

LabOne UI

This section shows how to use the LabOne UI to configure the instrument, run the measurement and monitor the measurement results.

- 1. Configure the instrument
 - 1. Set center frequency and power range of input and output signals
- Configure these parameters on the [Input and Output Tab](#) as in [Figure 4.29](#) and in [Table 4.26](#).

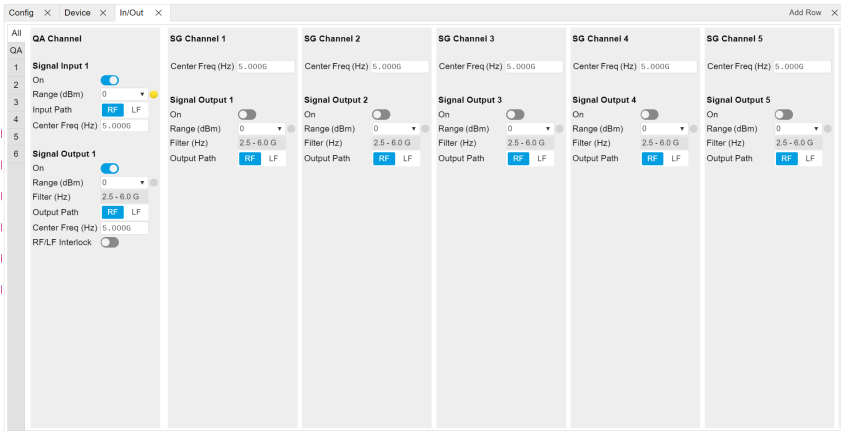


Figure 4.29: Configurations on In/Out Tab.

Table 4.26: Settings of QA Channel 1 on In/Out Tab

Parameter	Setting	Description
QA Channel Selection	All	Select All to display all Channels.
Cent Freq (Hz)	5 GHz	Set center frequency of the frequency sweep.
Signal Input 1 On	Enable	Enable the Signal Input 1.
Signal Input 1 Range (dBm)	0 dBm	Set power range of Signal Input 1 to 0 dBm. This setting allows the instrument to acquire a input signal with a power up to 0 dBm.
Signal Input 1 Input Path	RF	Set input path of Signal Input 1 to RF path.
Signal Output 1 On	Enable	Enable the Signal Output 1.
Signal Output 1 Range	0 dBm	Set power range of Signal Output 1 to 0 dBm. This setting allows the instrument to output a signal with a power up to 0 dBm.

Parameter	Setting	Description
Signal Output 1 Output Path	RF	Set output path of the Signal Output 1 to RF path.

2. Upload and compile measurement sequence
 The measurement sequence is defined on the Sequence sub-tab of the [Readout Pulse Generator](#) tab, see [Figure 4.30](#) and [Table 4.27](#).

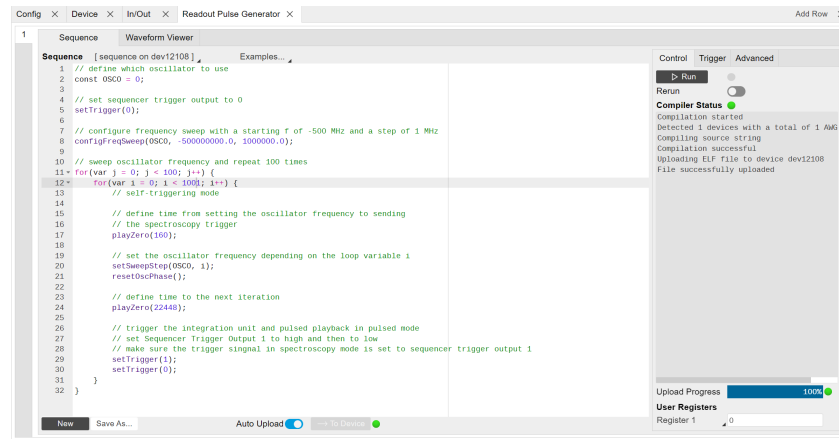


Figure 4.30: Configurations on Readout Pulse Generator Tab.

Table 4.27: Settings of QA Channel 1 on Readout Pulse Generator Tab.

Parameter	Setting	Description
QA Channel Selection	1	Select QA Channel 1.
Sub-Tab Display	Sequence	Select Sequence sub-tab and paste the sequence program below or load <code>ziGenerator_functional_spectroscopy.seq</code> from the "Examples" library and modify it. The Waveform Viewer sub-tab displays readout envelope if it is uploaded.
Compile	Click "To Device"	Compile the sequence program by clicking "To Device".
Return	Disable	Disable return function.
Run/Stop	Disable	Disable Run/Stop.

Below is the the sequence program for the measurement. In the inner loop, the `setSweepStep(OSC0, i)` command sets frequency of digital oscillator 0 to i-th frequency in an array configured by `configFreqSweep(OSC0, -500000000.0, 1000000.0)` command, the `setTrigger(value)` command sets Sequencer Trigger 1 Output to high and then low to start integration, and waveform generation if pulsed waveform is desired, the `playZero(samples)` command define time to the next `playZero(samples)`. The measurement is repeated 100 times by the outer loop.

```
// define which oscillator to use
const OSC0 = 0;

// set sequencer trigger output to 0
setTrigger(0);

// configure frequency sweep with a starting f of -500 MHz and a step
of 1 MHz
configFreqSweep(OSC0, -500000000.0, 1000000.0);

// sweep oscillator frequency and repeat 100 times
for(var j = 0; j < 100; j++) {
    for(var i = 0; i < 1001; i++) {
        // self-triggering mode
```

```

// define time from setting the oscillator frequency to
sending
// the spectroscopy trigger
playZero(160);

// set the oscillator frequency depending on the loop variable
i

setSweepStep(OSC0, i);
resetOscPhase();

// define time to the next iteration
playZero(22448);

// trigger the integration unit and pulsed playback in pulsed
mode
// set Sequencer Trigger Output 1 to high and then to low
// make sure the trigger signal in spectroscopy mode is set
to sequencer trigger output 1
setTrigger(1);
setTrigger(0);
}
}

```

3. Configure signal generation and data acquisition
Signal generation and data acquisition are defined on [QA Setup Tab](#) and [QA Result logger Tab](#). The baseband readout signal is generated by a digital oscillator directly ("Continuous"), or by mixing the signal from the digital oscillator and a waveform envelop saved in the waveform memory ("Pulse"). The input signal after frequency down-conversion is integrated for 512 ns started 224 ns later after receiving a trigger from Sequencer 1 Trigger Output 1.

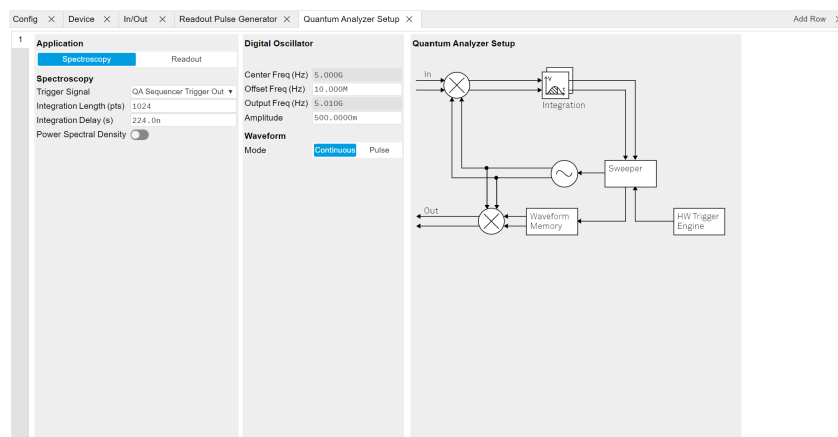


Figure 4.31: Configurations on QA Setup Tab.

Table 4.28: Settings of QA Channel 1 on QA Setup Tab, see details on [QA Setup Tab](#)

Parameter	Setting	Description
QA Channel Selection	1	Select QA Channel 1.
Application Mode	Spectroscopy	Use spectroscopy mode for resonator spectroscopy measurement. In spectroscopy mode, frequency sweep is done by sweeping the frequency of the digital oscillator.
Trigger Signal	Sequencer 1 Trigger Output 1	Select Sequencer 1 Trigger Output 1 as the trigger source to trigger both output pulse generation and integration. This selection matches the trigger setting written in the sequence program.
Integration Length (pts)	1048	Set the integration length in number of samples.

Parameter	Setting	Description
Integration Delay (s)	224n	Set the delay time after receiving a trigger before starting integration. This setting ensures only expected input signal is integrated. The internal delay from signal generation to integration is about 224 ns. Therefore integration delay is > 224 ns if the propagation delay from front panel signal output port to signal input port is not negligible.
Power Spectral Density	Disable	Disable the Power Spectral Density measurement function.
Digital Oscillator Amplitude	0.5	Set the amplitude factor of the digital oscillator to 0.5. The range of amplitude factor is from 0 to 1.
Waveform Mode	Continuous	Set the Waveform Mode to continuous, so the output waveform is continuous. Select "Pulse" and upload a .csv file with complex data for waveform envelope if pulsed output waveform is desired.

On the QA Result Logger Tab, it defines the source of the result, and how the result is averaged and displayed, see [Figure 4.32](#) and [Table 4.29](#). After configuration of all parameters, the QA Result Logger should be enabled to be ready to receive measurement results.

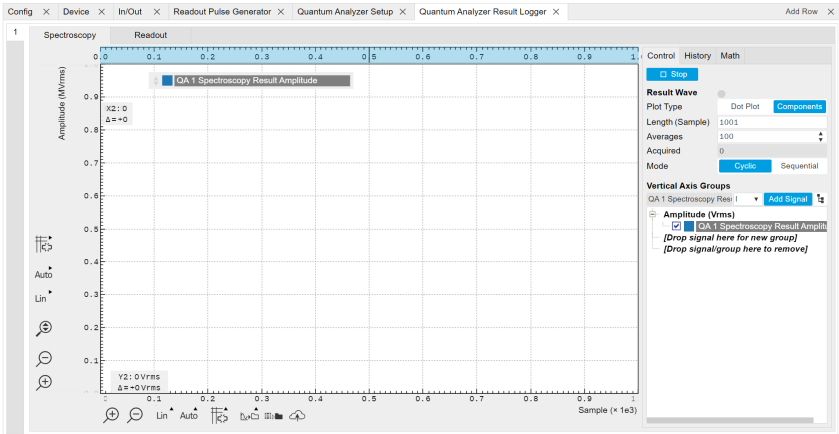


Figure 4.32: Configurations on QA Result Logger Tab.

Table 4.29: Settings of QA Channel 1 on QA Result Logger Tab.

Parameter	Setting	Description
QA Channel Selection	1	Select QA Channel 1.
Sub-Tab	Spectroscopy	Select Spectroscopy sub-tab to monitor measurement result when using the Spectroscopy mode.
Plot Type	Components	Select Components to display I, Q, amplitude or phase versus sample points. Select Dot Plot to display I versus Q with scattered dots.
Result Length (Sample)	1001	Set result length in number of samples. The number must match what is set in the sequence program.
Averages	100	Set number of averages. The number must match what is set in the sequence program.
Average Mode	Cyclic	Set the average mode to cyclic. This setting must match how the loop is configured in the sequence program.
Vertical Axis Groups	Add amplitude and phase	Select amplitude and phase to be displayed on the plot.

Parameter	Setting	Description
Run/Stop	Enable	Enable the result logger to receive and display measurement results.

- Run the measurement
By clicking "Run/Stop" icon on the Readout Pulse Generation Tab, the measurement is started and finished in seconds.
- Monitor the measurement result
The measurement result (complex data) is normalized by the integration length and displayed on the QA Result Tab, as shown in Figure 4.33. Select "Dot Plot" to display result in IQ (complex) plane.

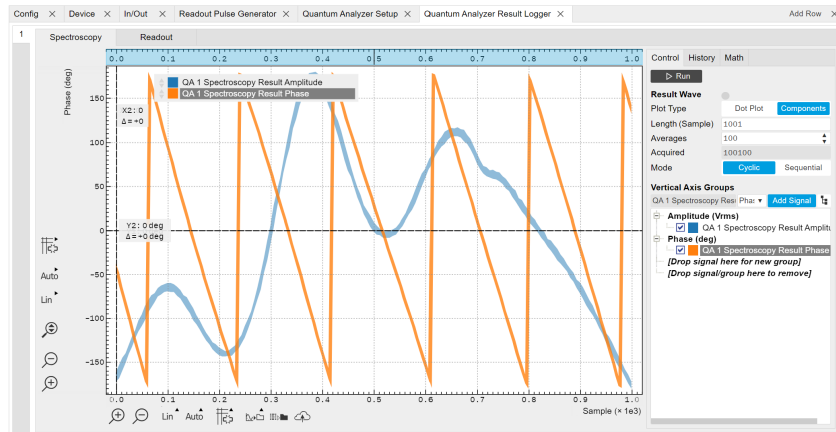


Figure 4.33: Measurement results on the QA Result Logger Tab.

zhinst-toolkit

In specific, we demonstrate how to run a frequency sweep, obtain the transmission data and plot it. For this, the Sweeper is configured to enable a continuous output signal that first probes the device under test and is then correlated with the generated signal. As a result, we obtain the amplitude and phase response in transmission of our device under test.

- Connect the Instrument
Create a toolkit session to the data server and connect the Instrument with the device ID, e.g. 'DEV12001', see [Connecting to the Instrument](#).

```
# Load the LabOne API and other necessary packages
from zhinst.toolkit import Session
```

```
DEVICE_ID = 'DEVXXXXX'
SERVER_HOST = 'localhost'
```

```
session = Session(SERVER_HOST)          ## connect to data server
device = session.connect_device(DEVICE_ID) ## connect to device
```

- Create a Sweeper and configure it
Python API [SHFSweeper](#) class is the core of the spectroscopy measurements. It defines all relevant parameters for frequency sweeping and sequencing. Toolkit wraps around the SHFSweeper and exposes an interface that is similar to the LabOne modules, meaning the parameters are exposed in a node tree like structure.

```
sweeper = session.modules.shfqa_sweeper
sweeper.device(device)
CHANNEL_INDEX = 0 # physical Channel 1

sweeper.sweep.start_freq(-500e6) # in units of Hz
sweeper.sweep.stop_freq(500e6) # in units of Hz
sweeper.sweep.num_points(1001)
sweeper.sweep.oscillator_gain(0.8) # amplitude scaling factor, 0 to 1
sweeper.sweep.use_sequencer = True # True (recommended): sequencer-based
sweep; False: host-driven sweep
```

```

sweeper.average.integration_delay(224e-9) # internal delay is about 224 ns
sweeper.average.integration_time(10e-6) # in units of second
sweeper.sweep.wait_after_integration(1e-6)
# waiting 1 us after integration before starting a new measurement
sweeper.average.num_averages(200)
sweeper.average.mode("sequential") # "sequential" or "cyclic" averaging

sweeper.rf.channel(CHANNEL_INDEX)
sweeper.rf.center_freq(5e9) # in units of Hz
sweeper.rf.input_range(0) # in units of dBm
sweeper.rf.output_range(0) # in units of dBm

with device.set_transaction():
    device.qachannels[CHANNEL_INDEX].input.on(1)
    device.qachannels[CHANNEL_INDEX].output.on(1)

```

All available settings of the Sweeper can be find using this command.

```
list(sweeper)
```

The data class `RfConfig` includes the channel-specific settings, such as the center frequency in units of Hz, and the power ranges of the Input and Output channel.

The data class `SweepConfig` allows to configure the sweeps start and stop frequency as the in-band offset frequency. In addition, it contains the number of measured points in the sweep 'num_points', and the mapping of the points ('linear' or 'logarithmic'). The gain of the oscillator 'oscillator_gain' changes the output amplitude of the probe signal relative to the channels power ranges. Hence, the output power of the probe signal hence changes quadratically with this number.

Please note that in this example, the configuration option `use_sequencer` is not explicitly set to its default value `True`. In this mode, a SeqC program is automatically generated, uploaded and compiled in the SHFQC+ Sequencer, and the frequency of the digital oscillator is controlled by the Sequencer. This mode allows a fast resonator spectroscopy with predicted cycle time of $t_{\text{settling time}} + t_{\text{integration delay}} + t_{\text{integration time}} + t_{\text{wait after integration}}$, see [Table 4.30](#). If `set_sequencer` is `False`, the sweeper is controlled by the host computer, and the frequency sweep is slower.

The data class `AvgConfig` contains all settings related to averaging and integration. The 'integration_time' defines how long the signal is integrated in the qubit_measurement_unit, which can be up to ~16.7 ms. The mode 'sequential' or 'cyclic' define, whether each point is first averaged 'num_averages' times before the frequency is changed, or whether every sweep is averaged 'num_averages' times.

Please note that default values of Sweeper parameters listed in [Table 4.30](#) depend on the zhinst version.

Table 4.30: Default values of Sweeper parameters.

Parameter	Description	Default Value for zhinst < 22.08	Default Value for zhinst >= 23.02
settling time	Waiting time after having set the new frequency until issuing the <code>setTrigger</code> command, which triggers the spectroscopy unit.	200 ns	80 ns
integration delay	Delay after the internal trigger arrives at the spectroscopy unit until the integration of the input signal starts.	0 ns	224 ns (zhinst-utils version ≥ 0.1.5)
envelope delay	Delay After the internal trigger arrives at the spectroscopy unit until the playback of the envelope waveform starts.	0 ns	0 ns
integration time	Time to integrate the input data.	1 ms	1 ms
wait after integration	Wait time after the end of the integration until the start of the next cycle.	72 ns	0 ns

Parameter	Description	Default Value for zhinst < 22.08	Default Value for zhinst >= 23.02
predicted cycle time	Calculated duration of each cycle of the spectroscopy loop. Note that this property only applies in self-triggered mode, which is active when the trigger source is set to None and use_sequencer is True .	-	"settling time" + "integration delay" + "integration time" + "wait after integration" (read-only)

3. Run the measurement with continuous output waveform and plot the data

```
result = sweeper.run()
num_points_result = len(result["vector"])
print(f"Measured at {num_points_result} frequency points.")
sweeper.plot()
```

After executing `sweeper.run()`, all above parameters are updated, and a SeqC program is automatically generated, uploaded and compiled based on the sweep parameters, see in [Figure 4.34](#). In the program, `ConfigFreqSweep` sets the start frequency and the frequency increment in units of Hz for a chosen oscillator, and `setSweepStep` sets the oscillator frequency. The oscillator phase is reset by `resetOscPhase` before each measurement. The trigger generated in the sequencer is used to start the integration, and the `playZero` sets the cycle duration. The measurement is repeated using the nested `for` loop according to the averaging mode. The measurement starts after enabling the sequencer.



Figure 4.34: SeqC program in the Readout Pulse Generator Tab.

The result returned from `sweeper.run()` is complex data E_{sweeper} in units of Vrms. It is averaged and normalized by the integration length. The power P and phase ϕ can be calculated as,

$$P = 10 \lg\left(\frac{|E_{\text{sweeper}}|^2}{R} 1000\right),$$

$$\phi = \arctan \frac{\Im(E_{\text{sweeper}})}{\Re(E_{\text{sweeper}})}, \quad (1)$$

where R is $50 \, \Omega$, 1000 is the conversion factor from W to mW. With `sweeper.plot()`, the power and phase are calculated and plotted, see [Figure 4.35](#).

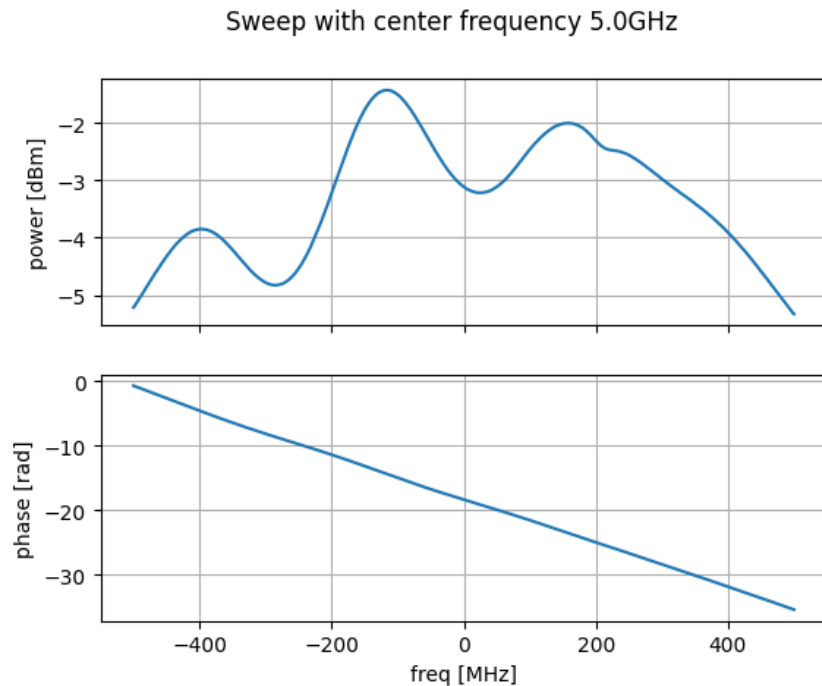


Figure 4.35: Measurement result with continuous waveform output.

4. Run the measurement with pulsed output waveform and plot the data
In contrast to continuous readout waveform generation, pulsed readout waveform generation requires an envelope data. Create a complex flat-top Gaussian envelope with 10 μ s duration and 50 ns rise and fall time. Enable the pulsed mode by `sweeper.envelope.enable(True)` and upload the envelope to the waveform memory. This envelope can be displayed on the Waveform Viewer of the Readout Pulse Generator.

```
from scipy.signal import gaussian
import numpy as np

SAMPLING_FREQUENCY = 2e9 # in units of Hz
ENVELOPE_DURATION = 10.0e-6 # in units of second
ENVELOPE_RISE_FALL_TIME = 0.05e-6 # in units of second

rise_fall_len = int(ENVELOPE_RISE_FALL_TIME * SAMPLING_FREQUENCY)
std_dev = rise_fall_len // 10
gauss = gaussian(2 * rise_fall_len, std_dev)
complex_amplitude = (1 - 1j)/np.sqrt(2)
flat_top_gaussian = np.ones(int(ENVELOPE_DURATION * SAMPLING_FREQUENCY)) *
complex_amplitude
flat_top_gaussian[0:rise_fall_len] = gauss[0:rise_fall_len] * complex_ampli
tude
flat_top_gaussian[-rise_fall_len:] = gauss[-rise_fall_len:] * complex_ampli
tude
sweeper.average.integration_delay(224e-9) # in units of second
sweeper.envelope.enable(True) # True: Pulsed mode; False: Continuous mode
sweeper.envelope.waveform(flat_top_gaussian) # upload envelope waveform

result = sweeper.run()
num_points_result = len(result["vector"])
print(f"Measured at {num_points_result} frequency points.")
sweeper.plot()
```

After executing `sweeper.run()`, all above parameters are updated, a SeqC program is automatically generated, uploaded and compiled based on the sweep parameters, see Figure 4.34, and the result is downloaded after the measurement is done. The power and phase are calculated, see in Figure 4.36. The result is consistent with measurement with continuous waveform output.

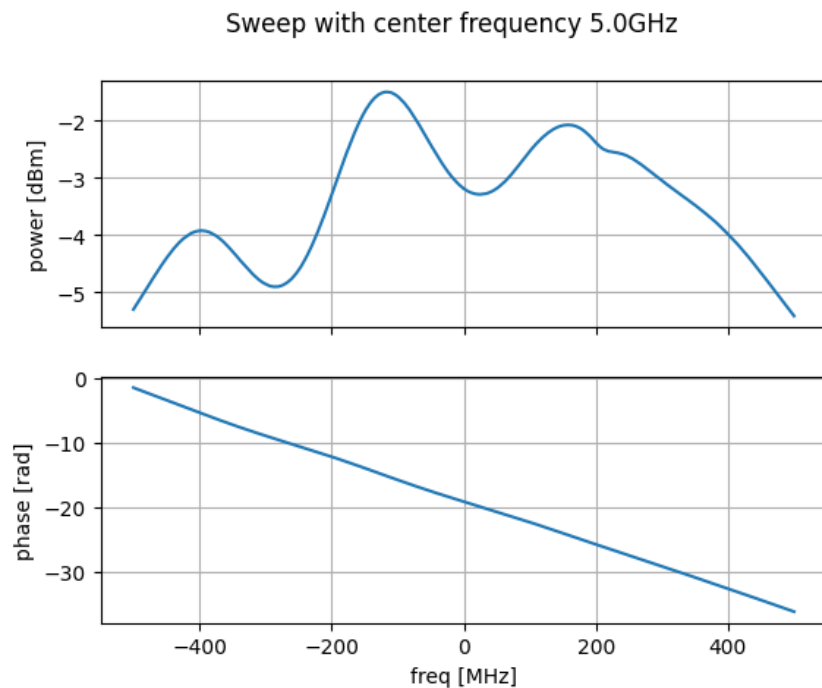


Figure 4.36: Measurement results with pulsed waveform output.

4.2.2. Multiplexed Qubit Readout

Note

This tutorial is applicable to all SHFQC+ Instruments.

Goals and Requirements

LabOne Q is the recommended control software to operate the SHFQC+ for Quantum Technology applications.

The goal of this tutorial is to demonstrate how to use SHFQC+ to perform multiplexed qubit readout using the LabOne User Interface (UI) and [Zurich Instruments Toolkit API](#).

For qubit control with the SHFSG+ Signal Generator, the SHFQC+ Qubit Controller and the HDAWG Arbitrary Wave Generator, and instrument synchronization and feedback with the PQSC Programmable Quantum System Controller, see tutorials section under a specific instrument found in the [Online Documentation](#).

Note

For zhinst-toolkit users, please find the example in https://github.com/zhinst/zhinst-toolkit/blob/main/examples/shfqa_qubit_readout_measurement.md, and the zhinst-toolkit documentation.

For SHFQC+ users, please find the examples in GitHub, <https://github.com/zhinst>.

Preparation

Please follow the preparation steps in [Connecting to the Instrument](#) and connect the instrument in a loopback configuration as shown in [Figure 4.37](#) or to a signal under test.

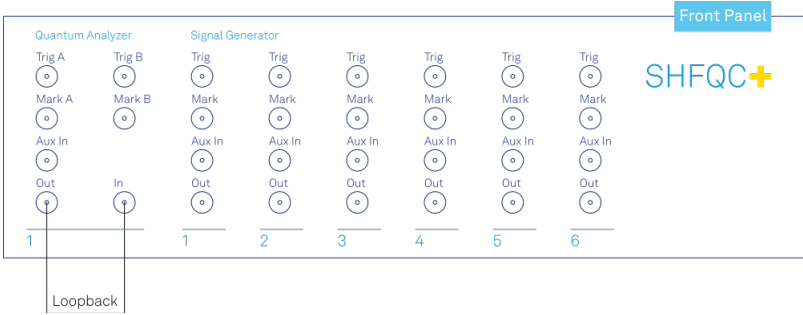


Figure 4.37: SHFQC+ connection.

Tutorial

In this tutorial, the instrument runs a measurement that performs readout of 8 qubits in parallel. The measurement is repeated 100×100 times and 100 averaged complex-valued results are returned.

LabOne UI

This section shows how to use LabOne UI to configure the instrument, run the measurement and monitor the measurement results.

- 1. Configure the instrument
 - 1. Set center frequency and power range of input and output signals
- Configure these parameters on the [Input and Output Tab](#) as in [Figure 4.38](#) and in [Table 4.31](#).

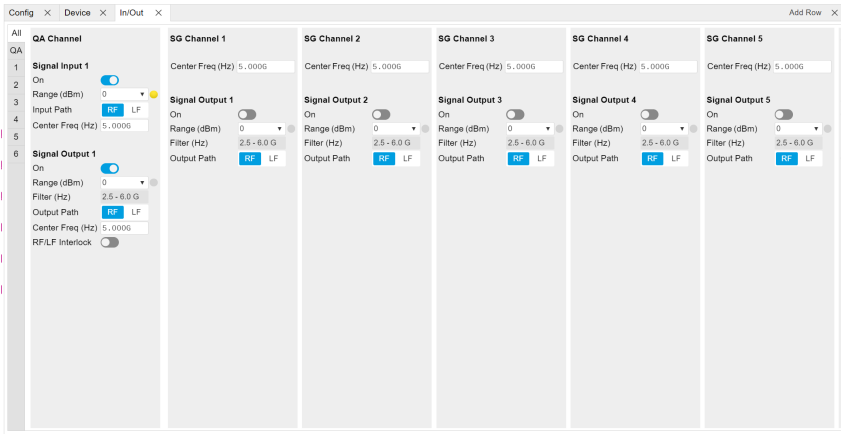


Figure 4.38: Configurations on In/Out Tab.

Table 4.31: Settings of QA Channel 1 on In/Out Tab

Parameter	Setting	Description
QA Channel Selection	All	Select All to display all Channels.
Cent Freq (Hz)	5 GHz	Set center frequency of the frequency sweep.
Signal Input 1 On	Enable	Enable the Signal Input 1.
Signal Input 1 Range (dBm)	0 dBm	Set power range of Signal Input 1 to 0 dBm. This setting allows the instrument to acquire a input signal with a power up to 0 dBm.
Signal Input 1 Input Path	RF	Set input path of Signal Input 1 to RF path.
Signal Output 1 On	Enable	Enable the Signal Output 1.
Signal Output 1 Range	0 dBm	Set power range of Signal Output 1 to 0 dBm. This setting allows the instrument to output a signal with a power up to 0 dBm.

Parameter	Setting	Description
Signal Output 1 Output Path	RF	Set output path of the Signal Output 1 to RF path.

2. Upload and compile measurement sequence
The measurement sequence is defined on the Sequence Sub-Tab of the [Readout Pulse Generator Tab](#), see [Figure 4.39](#) and [Table 4.32](#).

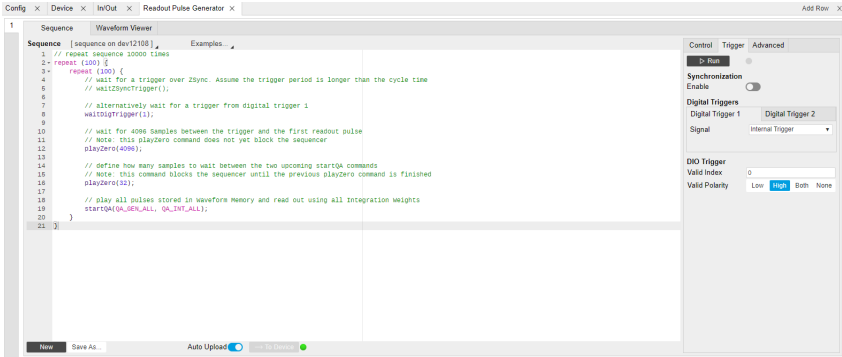


Figure 4.39: Configurations on Readout Pulse Generator Tab.

Table 4.32: Settings of QA Channel 1 on Readout Pulse Generator Tab.

Parameter	Setting	Description
QA Channel Selection	1	Select QA Channel 1.
Sub-Tab Display	Sequence	Select Sequence sub-tab and paste the sequence program below. The Waveform Viewer sub-tab displays waveforms saved in Waveform Memory slots and Integration Weight units.
Compile	Click "To Device"	Compile the sequence program by clicking "To Device".
Digital Triggers	Digital Trigger 1	Select Digital Trigger 1.
Digital Trigger 1 Signal	Internal Trigger	Select Internal Trigger as the trigger source of Digital Trigger 1.
Return	Disable	Disable return function.
Run/Stop	Enable	Run the sequence.

Below is the the sequence program for the measurement. In the inner loop, the `waitDigTrigger(1)` command waits for a digital trigger to continue the sequence, the first `playZero` command sets the waiting time after receiving a trigger before running the `startQA` command, the `startQA(QA_GEN_ALL, QA_INT_ALL, true)` command sends a trigger to generate output waveform and start integration. The measurement is repeated 100 times by the outer loop.

```
// repeat sequence 10000 times
repeat (100) {
    repeat (100) {
        // wait for a trigger over ZSync. Assume the trigger period is
        longer than the cycle time
        // waitZSyncTrigger();

        // alternatively wait for a trigger from digital trigger 1
        waitDigTrigger(1);

        // wait for 4096 Samples between the trigger and the first
        readout pulse
        // Note: this playZero command does not yet block the
        sequencer
        playZero(4096);
    }
}
```

```
// define how many samples to wait between the two upcoming
startQA commands
// Note: this command blocks the sequencer until the previous
playZero command is finished
playZero(32);

// play all pulses stored in Waveform Memory and read out
using all Integration Weights
startQA(QA_GEN_ALL, QA_INT_ALL);
}
}
```

The digital trigger set on the Trigger sub-tab is the Internal Trigger. The configuration of the Internal Trigger is shown in [Figure 4.40](#) and [Table 4.33](#).

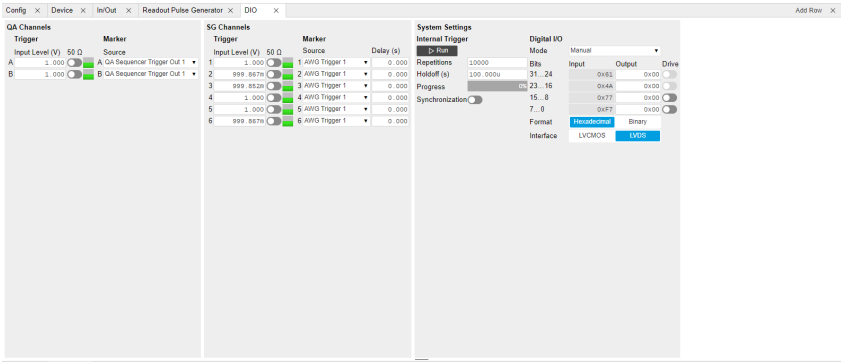


Figure 4.40: Configurations of Internal Trigger on DIO Tab.

Table 4.33: Settings of Internal Trigger on DIO Setup Tab, see details on [DIO Tab](#)

Parameter	Setting	Description
Repetitions	10000	Set number of repetitions.
Holdoff (s)	100u	Set holdoff time to 100 μ s.
Synchronization	Disable	Disable Synchronization.
Run/Stop	Disable	Disable the internal trigger.

3. Configure signal generation and data acquisition
Signal generation and data acquisition are defined on [QA Setup Tab](#) and [QA Result logger Tab](#).

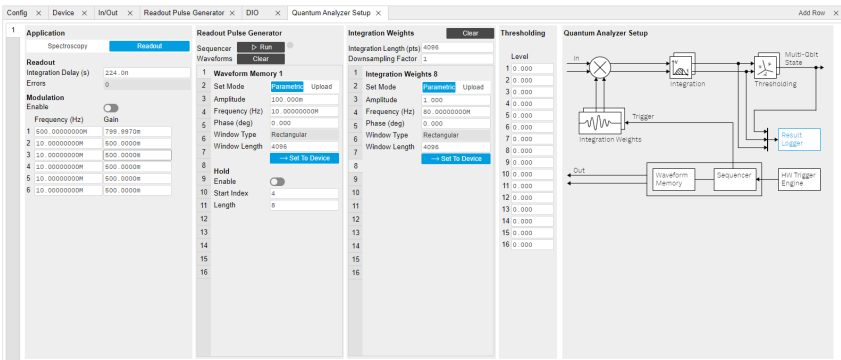


Figure 4.41: Configurations on QA Setup Tab.

The baseband readout waveform is generated by summing up all waveforms saved in the Waveform Memory. The input signal after frequency down-conversion is integrated with 8 integration weights saved in integration weight units for 2048 ns started 224 ns after receiving a digital trigger. We use Readout mode for multiplex qubit readout ([Table 4.34](#)) and generate both readout waveforms and integration weights parametrically ([Table 4.35](#) and [Table 4.36](#)). In this tutorial, the integration weight is simply a conjugate of the readout waveforms. To achieve high readout fidelity optimal weights should be measured and uploaded,

see how to measure optimal weights in [Integration Weights Measurement](#). Thresholds need to be uploaded if qubit state discrimination is required.

Table 4.34: Application Settings of QA Channel 1 on QA Setup Tab.

Parameter	Setting	Description
QA Channel Selection	1	Select QA Channel 1.
Application Mode	Readout	Use Readout mode for multiplex qubit readout. In Readout mode, the output waveform is generated by summing up all readout waveforms in the Waveform Memory slots, thus the instrument is able to readout up to 16 qubits per channel in parallel. In spectroscopy mode, the readout waveform is generated by a digital oscillator. Since there is only 1 digital oscillator per channel, only single qubit readout is possible in Spectroscopy mode.
Integration Delay (s)	224 n	Set the delay time after receiving a trigger before starting integration. This setting ensures that only the expected input signal is integrated. The internal delay from signal generation to integration is about 224 ns. Therefore, an integration delay > 224 ns is necessary if the propagation delay from front panel signal output port to signal input port is not negligible.

Table 4.35: Readout Pulse Generation Settings of QA Channel 1 on QA Setup Tab.

Parameter	Setting	Description
QA Channel Selection	1	Select QA Channel 1.
Clear Waveform	Click "Clear"	Clear all waveforms saved in Waveform Memory. Clear all waveforms before uploading new ones to avoid incorrect waveform generation or output overflow.
Waveform Memory <i>i</i> Set Mode	Parametric	Generate waveform parametrically saved in Waveform Memory slot <i>i</i> (<i>i</i> is from 1 to 8). The parametrically generated waveform is $Ae^{i(2\pi ft + \frac{\pi}{180}\phi)}$, where A is the dimensionless amplitude factor of the waveform, f is the frequency in units of Hz, φ is the phase in units of degree.
Waveform Memory <i>i</i> Amplitude	0.1	Set amplitude factor A of the parametrically generated waveform saved in Waveform Memory slot <i>i</i> (<i>i</i> is from 1 to 8) to 0.1. This setting ensures the amplitude factor of sum of all waveforms is ≤ 1 .
Waveform Memory <i>i</i> Frequency (Hz)	10e6× <i>i</i>	Set readout frequency f of the parametrically generated waveform saved in Waveform Memory slot <i>i</i> (<i>i</i> is from 1 to 8) to 10e6× <i>i</i> Hz.
Waveform Memory <i>i</i> Phase (Deg)	0	Set phase φ of the parametrically generated waveform saved in Waveform Memory slot <i>i</i> (<i>i</i> is from 1 to 8) to 0 degree.
Waveform Memory <i>i</i> Window Length	4096	Set length of the parametrically generated waveform saved in Waveform Memory slot <i>i</i> (<i>i</i> is from 1 to 8) in number of samples.
Waveform Memory <i>i</i> Set To Device	click "Set To Device"	Upload the parametrically generated waveform to Waveform Memory slot <i>i</i> (<i>i</i> is from 1 to 8).

Table 4.36: Integration Weights Settings of QA Channel 1 on QA Setup Tab.

Parameter	Setting	Description
QA Channel Selection	1	Select QA Channel 1.
Integration Length	4096	Set the integration length in number of samples.

Parameter	Setting	Description
Clear Weights	Click "Clear"	Clear all weights saved in Integration Weight slots. Clear all weights before uploading new ones to avoid incorrect weight generation.
Integration Weights i Set Mode	Parametric	Generate integration weight parametrically saved in Integration Weights unit i (i is from 1 to 8). The parametrically generated integration weight is $A'e^{-i(2\pi f't + \frac{\pi}{180}\phi)}$, where A' is the dimensionless amplitude factor of the integration weight, f' is the frequency in units of Hz, ϕ' is the phase in units of degree.
Integration Weights i Amplitude	1	Set amplitude factor A' of the parametrically generated integration weight saved in Integration Weights unit i (i is from 1 to 8)
Integration Weights i Frequency (Hz)	$10e6 \times i$	Set readout frequency f' of the parametrically generated integration weight saved in Integration Weights unit i (i is from 1 to 8) to $10e6 \times i$ Hz same as the frequency of readout waveform saved in Waveform Memory slot i.
Integration Weights i Phase (Deg)	0	Set phase ϕ' of the parametrically generated integration weight saved in Integration Weights unit i (i is from 1 to 8) to 0 degree.
Integration Weights i Window Length	4096	Set length of the integration weight saved in Integration Weights unit i (i is from 1 to 8) in number of samples.
Integration Weights i Set To Device	Click "Set To Device"	Upload the parametrically generated integration weight to Integration Weight unit 1. Use the same setting for 8 integration weights saved in the first 8 Integration Weight Units.

How the measurement results are averaged and displayed are defined on QA Result Logger Tab, see [Figure 4.42](#) and [Table 4.37](#).

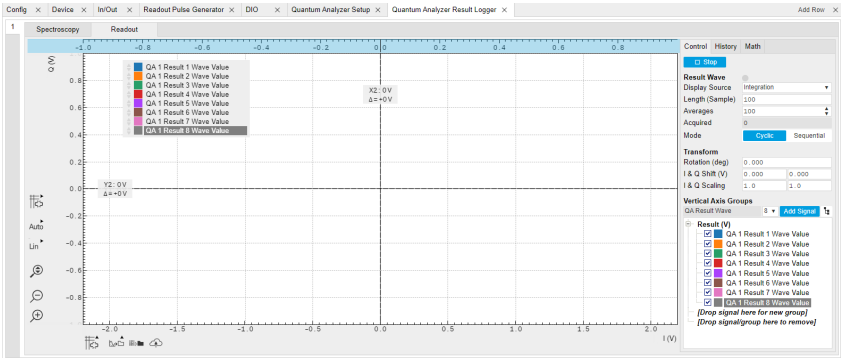


Figure 4.42: Configurations on QA Result Logger Tab.

Table 4.37: Settings of QA Channel 1 on QA Result Logger Tab, see details on [QA Result Logger Tab](#)

Parameter	Setting	Description
QA Channel Selection	1	Select QA Channel 1.
Sub-Tab	Readout	Select Readout sub-tab to monitor measurement result in Readout Mode.
Display Source	Integration	Display result after integration to monitor results in IQ plane. Choose "Integration (I, Q, Amp, Phase)" If it is desired. Choose "Threshold" if qubit state discrimination is desired.

Parameter	Setting	Description
Result Length (Sample)	100	Set result length in number of samples. The number must match what is set in the sequence program.
Averages	100	Set the number of averages. The number must match what is set in the sequence program.
Average Mode	Cyclic	Set the average mode to cyclic. This setting must match how the loop is configured in the sequence program.
Vertical Axis Groups	Add QA 1 Result i Wave Value (i is from 1 to 8)	Add 8 qubit results to the plot.
Run/Stop	Enable	Run the result logger to receive and display measurement results.

- Run the measurement
Click "Run/Stop" icon on the System Settings sub-tab of DIO tab to run the measurement.
- Monitor the measurement results
The measurement result is displayed on QA Result Tab, as shown in Figure 4.43. The data format of measurement result is complex data. The spread of the readout results indicates that each component of the input signal has a similar amplitude but different phase delay.

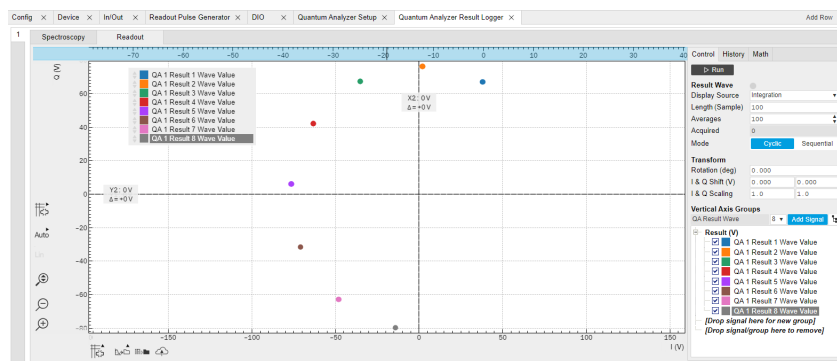


Figure 4.43: Measurement results on QA Result Logger Tab.

zhinst-toolkit

- Connect the instrument
Create a toolkit session to the data server and connect the device with the device ID, e.g. 'DEV12001', see [Connecting to the Instrument](#).

```
from zhinst.toolkit import Session, SHFQChannelMode, Waveforms
from scipy.signal import gaussian
import numpy as np

DEVICE_ID = 'DEVXXXXX'
SERVER_HOST = 'localhost'

session = Session(SERVER_HOST)          ## connect to data server
device = session.connect_device(DEVICE_ID) ## connect to device
```

- Generate readout pulses and integration weights
To readout 8 qubits in parallel, 8 readout pulses with different amplitude, frequency and phase are summed up to generate a single output signal. Please note that the maximum amplitude of the sum of all readout pulses should not exceed 1.

```
# generate readout pulses
NUM_QUBITS = 8
RISE_FALL_TIME = 10e-9 # in units of second
SAMPLING_RATE = 2e9 # in units of Hz
PULSE_DURATION = 500e-9 # in units of second
FREQUENCIES = np.linspace(32e6, 120e6, NUM_QUBITS) # in units of Hz
```

```

SCALING = 0.9 / NUM_QUBITS # amplitude scaling factor

rise_fall_len = int(RISE_FALL_TIME * SAMPLING_RATE)
pulse_len = int(PULSE_DURATION * SAMPLING_RATE)
std_dev = rise_fall_len // 10

gauss = gaussian(2 * rise_fall_len, std_dev)
flat_top_gaussian = np.ones(pulse_len)
flat_top_gaussian[0:rise_fall_len] = gauss[0:rise_fall_len]
flat_top_gaussian[-rise_fall_len:] = gauss[-rise_fall_len:]
flat_top_gaussian *= SCALING
time_vec = np.linspace(0, PULSE_DURATION, pulse_len)

readout_pulses = Waveforms()
for i, f in enumerate(FREQUENCIES):
    readout_pulses.assign_waveform(
        slot=i,
        wave1=flat_top_gaussian * np.exp(2j * np.pi * f * time_vec)
    )

# generate integration weights
ROTATION_ANGLE = 0
weights = Waveforms()
for waveform_slot, pulse in readout_pulses.items():
    weights.assign_waveform(
        slot=waveform_slot,
        wave1=np.conj(pulse[0] * np.exp(1j * ROTATION_ANGLE)) / np.abs(pulse[0])
    )

```

In this tutorial, the envelope of all readout pulses is flat-top Gaussian with pulse length of 500 ns and rise and fall time of 10 ns, all amplitude are equally scaled by a factor of 0.9 and divided by the number of qubits, 8 readout frequencies span from 32 MHz to 120 MHz, and all phases are set to 0. The zhinst-toolkit class [Waveforms](#) is for converting waveform data written in Python to data that can be uploaded to the instrument correctly. In Readout mode, the frequency down-converted signal is integrated with the integration weights. Tutorial [Integration Weights Measurement](#) shows how to measure integration weights to improve the SNR. In this tutorial, conjugated readout pulses with the amplitude scaling factor of 1 are used and uploaded to the integration weight memory. The integration delay can be measured with the SHFQC+ Scope triggered by Sequencer Monitor Trigger.

3. Configure the Channel

Configure the Channel such that the readout pulses are integrated with different integration weights in parallel, and the measurement is repeated 10000 times.

```

CHANNEL_INDEX = 0 # physical Channel 1
NUM_READOUTS = 100
NUM_AVERAGES = 100
MODE_AVERAGES = 0 # 0: cyclic; 1: sequential;
INTEGRATION_TIME = PULSE_DURATION # in units of second

# upload readout pulses and integration weights to waveform memory
device.qachannels[CHANNEL_INDEX].generator.clearwave() # clear all readout waveforms
device.qachannels[CHANNEL_INDEX].generator.write_to_waveform_memory(readout_pulses)
device.qachannels[CHANNEL_INDEX].readout.integration.clearweight() # clear all integration weights
device.qachannels[CHANNEL_INDEX].readout.write_integration_weights(
    weights=weights,
    # compensation for the delay between generator output and input of the integration unit
    integration_delay=224e-9
)

with device.set_transaction():

```



```

# configure inputs and outputs
device.qachannels[CHANNEL_INDEX].configure_channel(
    center_frequency=5e9, # in units of Hz
    input_range=-30, # in units of dBm
    output_range=-30, # in units of dBm
    mode=SHFQChannelMode.READOUT, # READOUT or SPECTROSCOPY
)
device.qachannels[CHANNEL_INDEX].input.on(1)
device.qachannels[CHANNEL_INDEX].output.on(1)

# configure sequencer
device.qachannels[CHANNEL_INDEX].generator.configure_sequencer_triggering(
    aux_trigger=8, # internal trigger
    play_pulse_delay=0, # 0s delay between startQA trigger and the
    readout pulse
)

seqc_program = f"""
    repeat({int(NUM_READOUTS * NUM_AVERAGES)}){{
        waitDigTrigger(1);
        startQA(QA_GEN_ALL, QA_INT_ALL, true, 0, 0x0);
    }}
"""
device.qachannels[CHANNEL_INDEX].generator.load_sequencer_program(seqc_program)

# configure internal trigger
device.system.internaltrigger.repetitions(int(NUM_READOUTS * NUM_AVERAGES))
device.system.internaltrigger.holdoff(100e-6)

# configure QA setup and QA result logger
device.qachannels[CHANNEL_INDEX].readout.integration.length(int(INTEGRATION_TIME * SAMPLING_RATE))
device.qachannels[CHANNEL_INDEX].readout.configure_result_logger(
    result_length=NUM_READOUTS,
    result_source='result_of_integration',
    # "result_of_integration" or "result_of_discrimination".
    num_averages=NUM_AVERAGES,
    averaging_mode=MODE_AVERAGES,
)

```

The Readout pulses and integration weights with assigned waveform memory slots are uploaded using `generator.write_to_waveform_memory` to the waveform memory after clear all waveforms which may saved in the waveform memory previously.

The input range and output range of the Channel 1 is set to -30 dBm, and the center frequency is 5 GHz. There are 2 application modes see [Quantum Analyzer Setup Tab](#). For multiplex readout, Readout mode is selected in order to use customized integration weights for different qubits. All settings are configured using `qachannels[n].configure_channel` function. The function `configure_sequencer_triggering` and `load_sequencer_program` are used to configure the sequence trigger and upload the sequence program, respectively. The measurement sequence is defined by the SeqC program such that it sends out the readout pulse and integrates the signal for 500 ns with the integration delay of 224 ns after receiving a trigger (Internal Trigger). The measurement is repeated 10000 times. These parameters are configured by the function `readout.configure_result_logger`.

The result after integration and averaging is saved to the QA Result Logger. This configuration can be used to calibrate control pulses and characterize the qubits, measure thresholds without averaging for state discrimination or readout fidelity if the result source is set to 'result_of_discrimination' and the thresholds are updated using `device.qachannels[CHANNEL_INDEX].readout.discriminators[n].threshold()` (n is the qubit index).

4. Run the measurement, download and plot the result
Before starting the measurement, the QA Result Logger is enabled to be ready to get the result, and the Sequencer is enabled by `enable_sequencer` to be ready to run the sequence once receive an internal trigger. The measurement result is returned by `readout.read` function, and it is also displayed in QA Result Logger Tab see [Figure 4.44](#).


```

device.qachannels[CHANNEL_INDEX].readout.run() # enable QA Result Logger
device.qachannels[CHANNEL_INDEX].generator.enable_sequencer(single=True)
device.system.internaltrigger.enable(1)
readout_results = device.qachannels[CHANNEL_INDEX].readout.read()

```

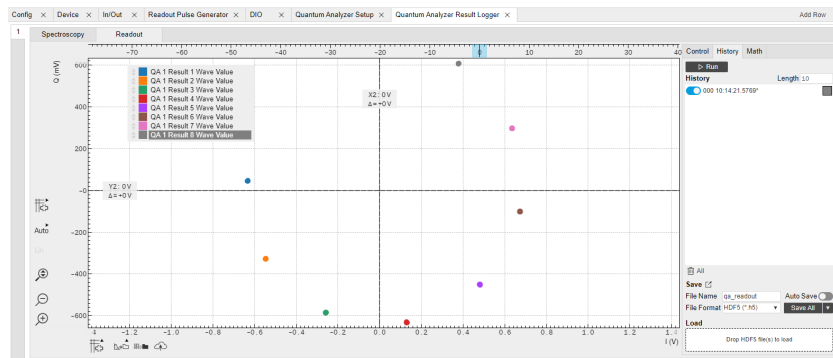


Figure 4.44: Multiplexed readout of 8 qubits in loopback configuration.

Use the following code snippet to plot the readout result as seen in [Figure 4.45](#)

```

import matplotlib.pyplot as plt

plt.figure()
for i in range(NUM_QUBITS):
    plt.plot(readout_results[i].real, readout_results[i].imag, '.', label =
f'Q{i}')
plt.legend()
plt.grid("both")
plt.axis("equal")
plt.xlabel("I (Vrms)")
plt.ylabel("Q (Vrms)")
plt.tight_layout()

```

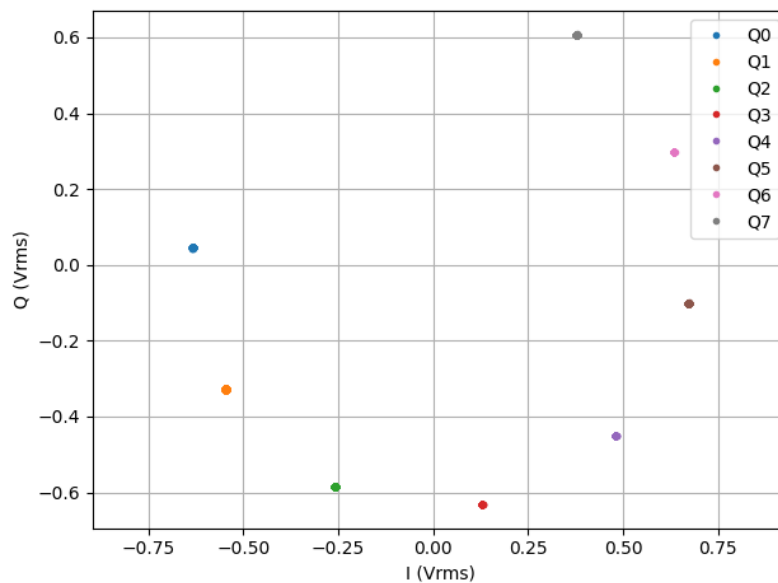


Figure 4.45: Multiplexed readout of 8 qubits in loopback configuration.

4.2.3. Integration Weights Measurement

Note

This tutorial is applicable to all SHFQC+ Instruments and no additional instrumentation is needed.

Goals and Requirements

LabOne Q is the recommended control software to operate the SHFQC+ for Quantum Technology applications.

The goal of this tutorial is to demonstrate how to use the SHFQC+ Scope to measure integration weights that are needed for high-fidelity single-shot readout using the LabOne User Interface (UI) and [Zurich Instruments Toolkit API](#).

For qubit control with the SHFSG+ Signal Generator, the SHFQC+ Qubit Controller and the HDAWG Arbitrary Wave Generator, and instrument synchronization and feedback with the PQSC Programmable Quantum System Controller, see tutorials in the [Online Documentation](#).

Note

For zhinst-toolkit users, please find the example in https://github.com/zhinst/zhinst-toolkit/blob/main/examples/shfqa_qubit_readout_weights.md, and the zhinst-toolkit documentation.

For LabOne Q users, please find the example in https://github.com/zhinst/laboneq/blob/main/examples/01_qubit_characterization/12_readoutweight_calibration_shfsg_shfqa_shfqc.ipynb, and the [LabOne Q User Manual](#).

Preparation

Please follow the preparation steps in [Connecting to the Instrument](#) and connect the instrument in a loopback configuration as shown in [Figure 4.46](#) or to a device under test.

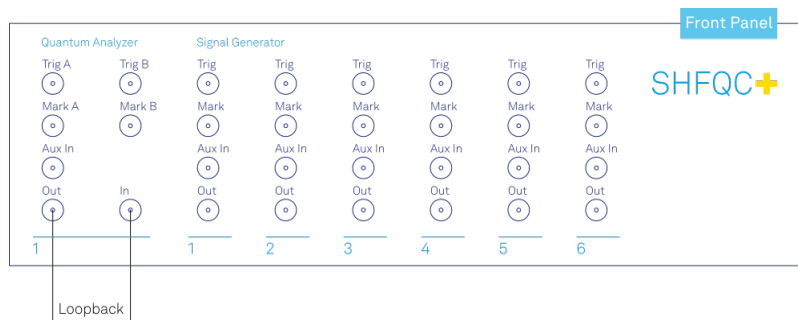


Figure 4.46: SHFQC+ connection.

Tutorial

In the tutorial, readout signals are recorded while the qubit is in ground and excited state. The readout pulse is generated by the SHFQC+ Readout Pulse Generator, and the signal recording is done by the SHFQC+ Scope. The integration weight is derived from the difference of the recorded readout signals.

LabOne UI

This section shows how to use LabOne UI to configure the instrument, run the measurement, monitor the measurement results and calculate the integration weight.

1. Configure the instrument
 1. Set center frequency and power range of input and output signals
Configure these parameters on the [Input and Output Tab](#) as in [Figure 4.47](#) and in [Table 4.38](#).

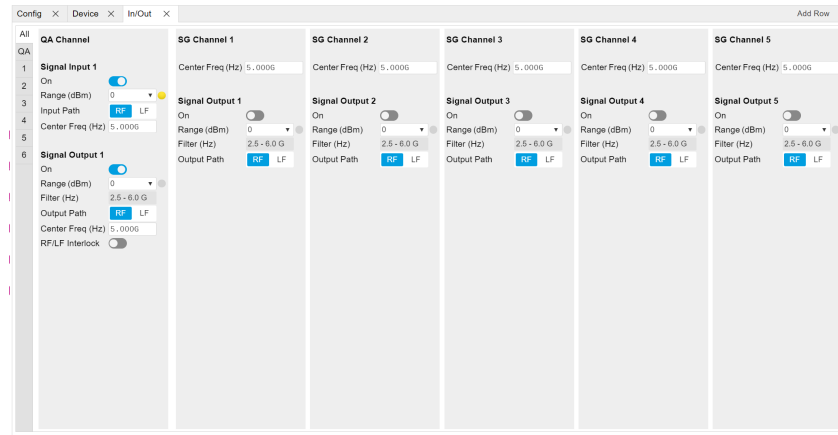


Figure 4.47: Configurations on In/Out Tab.

Table 4.38: Settings of QA Channel 1 on In/Out Tab

Parameter	Setting	Description
QA Channel Selection	All	Select All to display all Channels.
Cent Freq (Hz)	5 GHz	Set center frequency of the frequency sweep of QA Channel 1.
Signal Input 1 On	Enable	Enable the Signal Input 1.
Signal Input 1 Range (dBm)	0 dBm	Set power range of Signal Input 1 to 0 dBm. This setting allows the instrument to acquire a input signal with a power up to 0 dBm.
Signal Input 1 Input Path	RF	Set input path of Signal Input 1 to RF path.
Signal Output 1 On	Enable	Enable the Signal Output 1.
Signal Output 1 Range	0 dBm	Set power range of Signal Output 1 to 0 dBm. This setting allows the instrument to output a signal with a power up to 0 dBm.
Signal Output 1 Output Path	RF	Set output path of the Signal Output 1 to RF path.

2. Upload and compile measurement sequence

The measurement sequence is defined on the Sequence Sub-Tab of the [Readout Pulse Generator Tab](#), see [Table 4.39](#).

Table 4.39: Settings of QA Channel 1 on Readout Pulse Generator Tab.

Parameter	Setting	Description
QA channel Selection	1	Select QA channel 1.
Sub-Tab Display	Sequence	Select Sequence sub-tab and paste the sequence program below. The Waveform Viewer sub-tab displays waveforms saved in Waveform Memory slots and Integration Weight units.
Compile	Click "To Device"	Compile the sequence program by clicking "To Device".
Digital Triggers	Digital Trigger 1	Select Digital Trigger 1.
Digital Trigger 1 Signal	Internal Trigger	Select Internal Trigger as the trigger source of Digital Trigger 1.
Return	Disable	Disable return function.
Run/Stop	Enable	Run the sequence.

Below is the the sequence program for the measurement. In the loop, the `waitDigTrigger` command waits a trigger (see [Table 4.40](#)) to continue the sequence,

the `startQA` command sends a trigger to generate output waveform and start integration, it also sends a Sequencer Monitor Trigger (third argument of the `startQA` command) to trigger SHFQC+ Scope to record input signal before integration. The measurement is repeated 100 times to get averaged readout pulses according to qubit prepared in ground and excited state.

```
// repeat sequence 100 times, i.e. readout qubit in group state and
// excited state, and then repeat this 50 times
repeat (100) {
    // wait for a trigger over ZSync. Assume the trigger period is
    // longer than the cycle time
    // waitZSyncTrigger();

    // alternatively wait for a trigger from digital trigger 1
    waitDigTrigger(1);

    // play readout waveform stored in Waveform Memory slot 1, send a
    // trigger to start integration, and send a Sequencer Monitor trigger to
    // trigger the Scope
    startQA(QA_GEN_0, QA_INT_0, true);
}
```

The digital trigger set on the Trigger sub-tab is Internal Trigger. The configuration of the internal is shown in Table 4.40.

Table 4.40: Settings of Internal Trigger on DIO Setup Tab, see details on [DIO Tab](#)

Parameter	Setting	Description
Repetitions	100	Set number of repetitions.
Holdoff (s)	100u	Set holdoff time to 100 μ s.
Synchronization	Disabled	Disable Synchronization.
Run/Stop	Disable	Disable the internal trigger.

3. Configure signal generation

Signal generation is defined on [QA Setup Tab](#), see [Figure 4.48](#) and [Table 4.41](#). Since we are only interested in the signal before integration, the settings for integration weights is not needed.

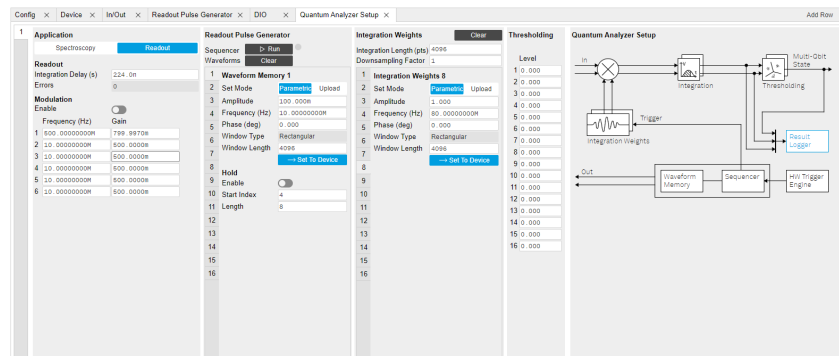


Figure 4.48: Configurations on QA Setup Tab.

Table 4.41: Settings of QA Channel 1 on QA Setup Tab.

Parameter	Setting	Description
QA Channel Selection	1	Select QA Channel 1.
Application Mode	Readout	Use Readout mode for integration weight measurement.
Clear Waveform	Click "Clear"	Clear all waveforms saved in Waveform Memory. Clear all waveforms before uploading new ones to avoid incorrect waveform generation or output overflow.

Parameter	Setting	Description
Waveform Memory 1 Set Mode	Parametric	Generate waveform parametrically in Waveform Memory slot 1. The parametrically generated waveform is $Ae^{i(2\pi f t + \frac{\pi}{180}\phi)}$, where A is the dimensionless amplitude factor of the waveform, f is the frequency in units of Hz, φ is the phase in units of degree.
Amplitude	0.5	Set amplitude factor A to 0.5.
Frequency (Hz)	10M	Set readout frequency f to 10 MHz.
Phase (Deg)	0	Set phase φ to 0 degree.
Window Length	4096	Set length of the readout waveform in number of samples.
Set To Device	click "Set To Device"	Upload the parametrically generated waveform to Waveform Memory slot 1.

4. Configure SHFQC+ Scope

The SHFQC+ Scope is configured to record readout pulses before integration, and display averaged readout pulses according to qubit in ground state and excited state, see [Figure 4.49](#) and [Table 4.42](#). The figure shows the SHFQA+ Scope configuration for the same measurement. Note that the input sources of the first and the second Scope channel are fixed to "Signal Input 1", and only SG channels can be selected as the input sources of the third and the fourth Scope channel.

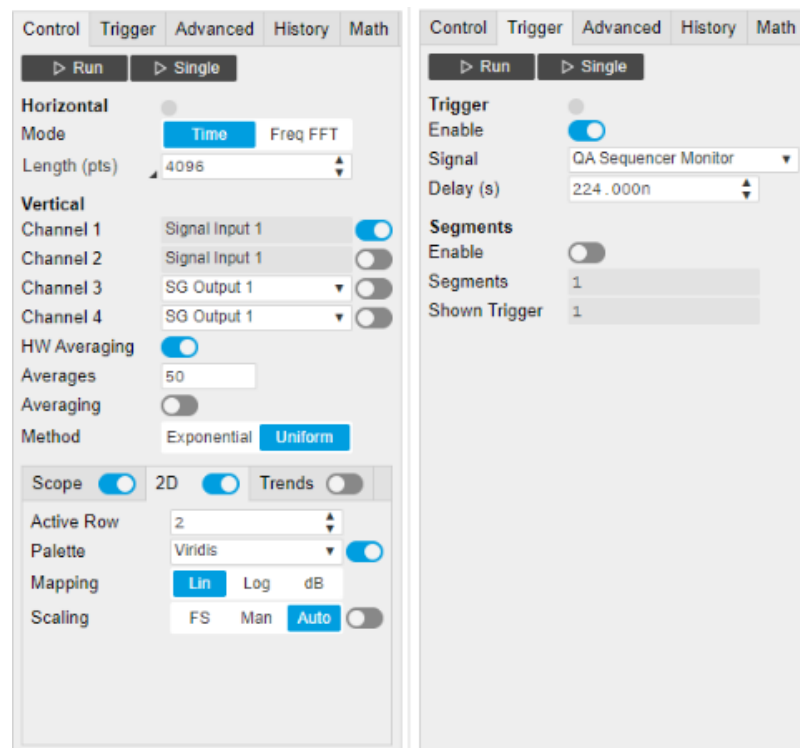


Figure 4.49: Configurations on Scope Tab.

Table 4.42: Settings on Scope Tab

Parameter	Setting	Description
Horizontal Mode	Time	Display data in time-domain.
Horizontal Length (pts)	4096	Set recording length in number of samples. This setting has to be \geq readout pulse length.
Channel 1 Signal Selection	Signal Input 1	Monitor signal comes from Signal Input 1 on Scope Channel 1.

Parameter	Setting	Description
Channel 1 Enable	Enable	Enable Scope Channel 1.
HW Averaging	Enable	Enable hardware averaging.
Averages	50	Set number of averages to 50. This setting matches what is defined in the sequence program.
Display Mode	Scope	Display the Scope trace. Enable "2D" if 2D trace is desired.
Add Signal	Scope Wave Channel 1 I and Q	Add Scope Wave Channel 1 I and Scope Wave Channel 1 Q to the plot.
Trigger Mode	Enable	Enable trigger mode so that scope recording starts only after receiving a trigger.
Trigger Signal	Sequencer 1 Monitor Trigger	Select Sequencer 1 Monitor Trigger as the trigger to trigger the Scope.
Trigger Delay (s)	224n	Set trigger delay to 224 ns. The Scope starts to record data 224 ns later after receiving a trigger. This setting must has to match signal propagation delay including internal delay about 224 ns and external delay depending on the signal path between the front panel Signal Output port and Signal Input port.
Segments Enable	Enable	Enable Segments measurement. In Segments measurement, the scope records $n \times m$ data, where n is the number of segments, m is the recording length in number of samples.
Segments	2	Set number of segments to 2. 1 for recording readout pulse when qubit in ground state, another for qubit in excited state.
Run Mode	Single	Using Single mode for the measurement.

- Run the measurement
By clicking "Run/Stop" icon on the System Settings sub-tab of DIO tab, the measurement is started and finished in seconds.
- Monitor the measurement results and calculate the integration weight
The recorded readout pulses are displayed on the Scope Tab, as shown in Figure 4.50. The integration weights is calculated by taking the difference of the measured traces according to the qubit is in ground state and excited state.

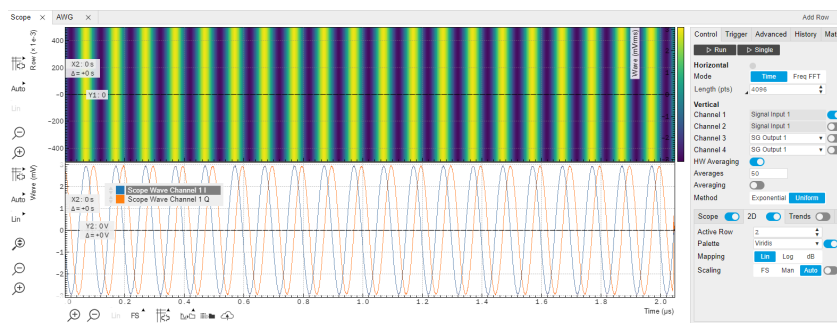


Figure 4.50: Recorded readout pulses on Scope Tab.

zhinst-toolkit

- Connect the instrument
Create a toolkit session to the data server and connect the device with the device ID, e.g. 'DEV12001', see [Connecting to the Instrument](#).

```
# Load the LabOne API and other necessary packages
from zhinst.toolkit import Session, SHFQChannelMode, Waveforms
from scipy.signal import gaussian
import numpy as np
```

```

DEVICE_ID = 'DEVXXXXX'
SERVER_HOST = 'localhost'

session = Session(SERVER_HOST)          ## connect to data server
device = session.connect_device(DEVICE_ID) ## connect to device

```

2. Generate readout pulses
In the tutorial, the envelope of the readout waveforms is flat-top Gaussian with pulse length of 500 ns and rise and fall time of 10 ns, amplitude factor of 0.9, and 8 readout frequencies span from 32 MHz to 120 MHz. The amplitude factor is not scaled by the number of qubits as in [Multiplexed Qubit Readout](#) because the readout output signal is generated from one of the readout waveforms, see the sequence program in the following section. The zhinst-toolkit class [Waveforms](#) is for converting waveform data written in Python to data that can be uploaded to the instrument correctly.

```

NUM_QUBITS = 8
SAMPLING_FREQUENCY = 2e9
RISE_FALL_TIME = 10e-9
PULSE_DURATION = 500e-9

rise_fall_len = int(RISE_FALL_TIME * SAMPLING_FREQUENCY)
pulse_len = int(PULSE_DURATION * SAMPLING_FREQUENCY)
std_dev = rise_fall_len // 10

gauss = gaussian(2 * rise_fall_len, std_dev)
flat_top_gaussian = np.ones(pulse_len)
flat_top_gaussian[0:rise_fall_len] = gauss[0:rise_fall_len]
flat_top_gaussian[-rise_fall_len:] = gauss[-rise_fall_len:]
# Scaling
flat_top_gaussian *= 0.9

readout_pulses = Waveforms()
time_vec = np.linspace(0, PULSE_DURATION, pulse_len)

for i, f in enumerate(np.linspace(2e6, 32e6, NUM_QUBITS)):
    readout_pulses.assign_waveform(
        slot=i,
        wave1=flat_top_gaussian * np.exp(2j * np.pi * f * time_vec)
    )

```

3. Configure the channel
Configure center frequency, input and output power range and application mode of the channel using [qachannels\[n\].configure_channel](#), turn on the Input and Output, and upload the readout waveforms using [generator.write_to_waveform_memory](#).

```

# configure inputs and outputs
CHANNEL_INDEX = 0 # physical Channel 1

device.qachannels[CHANNEL_INDEX].configure_channel(
    center_frequency=5e9, # in units of Hz
    input_range=0, # in units of dBm
    output_range=-5, # in units of dBm
    mode=SHFQChannelMode.READOUT, # SHFQChannelMode.READOUT or
    SHFQChannelMode.SPECTROSCOPY
)
device.qachannels[CHANNEL_INDEX].input.on(1)
device.qachannels[CHANNEL_INDEX].output.on(1)

# write waveforms to the Waveform Memory
device.qachannels[CHANNEL_INDEX].generator.write_to_waveform_memory(readout_pulses)

```

4. Configure the Scope

Configure the Scope to record 2 segments of data with length of 500 ns which are averaged 50 times using `scopes[n].configure`. The trigger of the Scope is 1 Sequencer 1 Monitor Trigger which is enabled by the `startQA` command.

```
SCOPE_CHANNEL = 0
RECORD_DURATION = 500e-9 # in units of second
NUM_SEGMENTS = 2
NUM_AVERAGES = 50
NUM_MEASUREMENTS = NUM_SEGMENTS * NUM_AVERAGES
SAMPLING_FREQUENCY = 2e9 # in units of Hz

device.scopes[0].configure(
    input_select={SCOPE_CHANNEL: f"channel{CHANNEL_INDEX}_signal_input"},
    num_samples=int(RECORD_DURATION * SAMPLING_FREQUENCY),
    trigger_input=f"channel{CHANNEL_INDEX}_sequencer_monitor0",
# Sequencer 1 monitor trigger
    num_segments=NUM_SEGMENTS,
    num_averages=NUM_AVERAGES,
    trigger_delay=214e-9, # record the data 214 ns later after receiving a
    trigger
)
```

5. Configure and run the measurement, and calculate the integration weights
In the measurement, the Sequencer is triggered by Internal Trigger using `configure_sequencer_trigger`. The integration weights for different qubits are measured sequentially with the `for` loop. In each loop, the `seqc_program` is different and is uploaded to the instrument using `load_sequencer_program`, both Scope and Sequencer run in Single mode set by `scopes[n].run` and `enable_sequencer` respectively, the Generator sends 100 pulses to readout 1 of the qubits prepared in ground and excited state, and the Scope acquires 2 segments of data which are averaged 50 times, and the data is downloaded using `scopes[n].read`, and integration weights are calculated in the end.

```
results = []

device.qachannels[CHANNEL_INDEX].generator.configure_sequencer_triggering(
    aux_trigger=8, # internal trigger
    play_pulse_delay=0, # 0s delay between startQA trigger and the readout
    pulse
)
device.system.internaltrigger.repetitions(int(NUM_SEGMENTS * NUM_AVERAGES))
device.system.internaltrigger.holdoff(100e-6)

for i in range(NUM_QUBITS):
    qubit_result = {
        'weights': None,
        'ground_states': None,
        'excited_states': None
    }
    print(f"Measuring qubit {i}.")

    # upload sequencer program
    seqc_program = f"""
        repeat({NUM_MEASUREMENTS}) {{
            waitDigTrigger(1);
            startQA(QA_GEN_{i}, 0x0, true, 0, 0x0); // only QA_GEN_{i}
            matters for this measurement
        }}
    """
    device.qachannels[CHANNEL_INDEX].generator.load_sequencer_program(seqc_program)

    # Start a measurement
    device.scopes[SCOPE_CHANNEL].run(single=True)
```



```

device.qachannels[CHANNEL_INDEX].generator.enable_sequencer(single=True)
device.system.internaltrigger.enable(1)

# get results to calculate weights and plot data
scope_data, *_ = device.scopes[0].read()

# Calculates the weights from scope measurements
# for the excited and ground states
split_data = np.split(scope_data[SCOPE_CHANNEL], 2)
ground_state_data = split_data[0]
excited_state_data = split_data[1]
qubit_result['ground_state_data'] = ground_state_data
qubit_result['excited_state_data'] = excited_state_data
qubit_result['weights'] = np.conj(excited_state_data - ground_state_data)
a)
results.append(qubit_result)

```

The following code snippet can be used to plot readout traces (Figure 4.51) and calculate integration weights (Figure 4.52) of the last qubit.

Note

In order to achieve the highest possible resolution in the signal after integration, it's advised to scale the dimensionless readout integration weights with a factor so that their maximum absolute value is equal to 1.

```

import matplotlib.pyplot as plt

fig, (ax0, ax1) = plt.subplots(nrows = 2, sharex = True)

t = np.linspace(0, RECORD_DURATION, int(RECORD_DURATION * SAMPLING_FREQUENCY/16)*16)
ax0.plot(t/1e-9, ground_state_data.real, '-', label = f'real')
ax0.plot(t/1e-9, ground_state_data.imag, '-', label = f'imag')
ax1.plot(t/1e-9, excited_state_data.real, '-', label = f'real')
ax1.plot(t/1e-9, excited_state_data.imag, '-', label = f'imag')

ax0.set_title('Ground state')
ax1.set_title('Excited state')
ax0.legend(loc='upper right')
ax1.legend(loc='upper right')
ax0.set_ylabel('Amplitude (Vrms)')
ax1.set_ylabel('Amplitude (Vrms)')
ax1.set_xlabel(r'Time (ns)')
plt.tight_layout()

plt.figure()
plt.plot(t/1e-9, results[0]['weights'])
plt.xlabel('Time (ns)')
plt.ylabel('Amplitude (Vrms)')
plt.tight_layout()

```

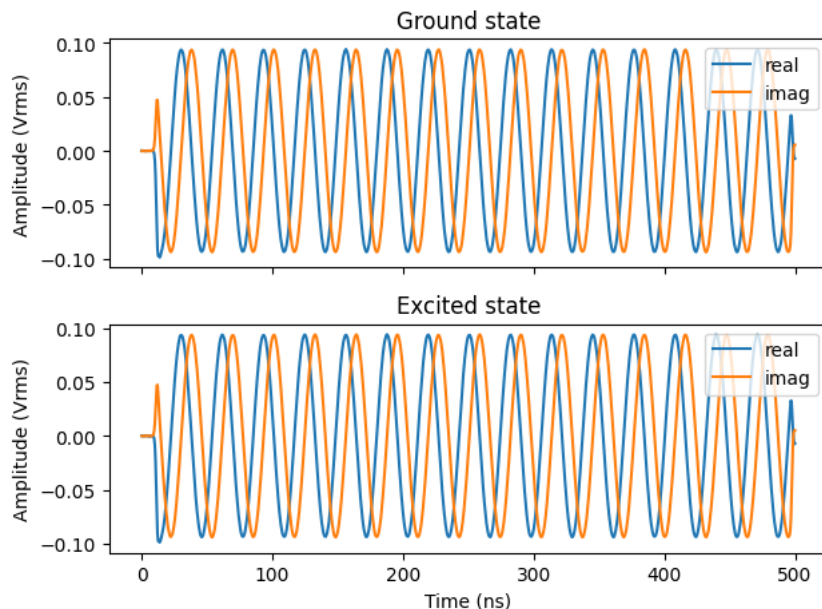


Figure 4.51: Readout pulses recorded by the Scope in loopback configuration

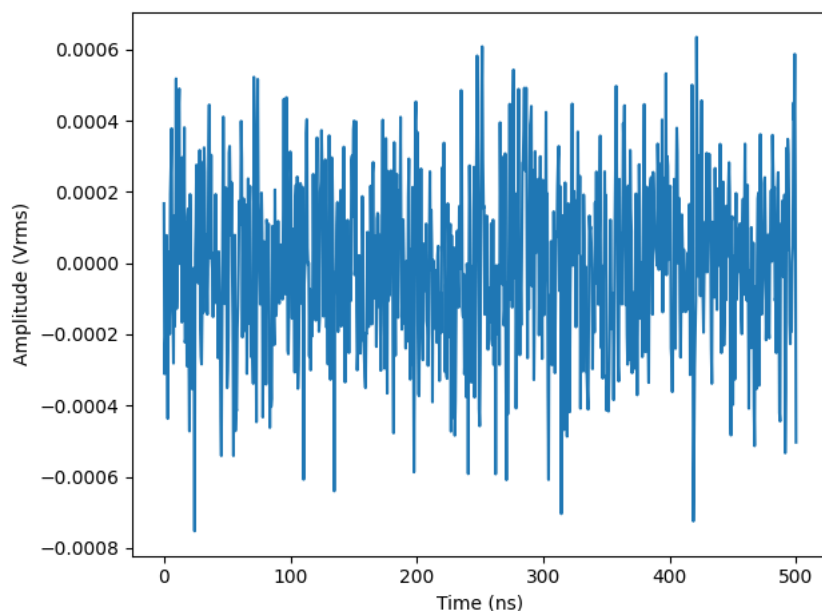


Figure 4.52: Integration weights calculated from the readout pulses

4.2.4. Multistate Discrimination

Note

This tutorial is applicable to all SHFQC+ Instruments.

Goals and Requirements

[LabOne Q](#) is the recommended control software to operate the SHFQC+ for Quantum Technology applications.

The goal of this tutorial is to demonstrate how to use SHFQC+ to perform multistate discrimination using [Zurich Instruments Toolkit API](#).

Note

For zhinst-toolkit users, please find the example in https://github.com/zhinst/zhinst-toolkit/blob/main/examples/SHFQC+_multistate_discrimination.md, and the zhinst-toolkit documentation.

Preparation

Please follow the preparation steps in [Connecting to the Instrument](#) and connect the instrument in a loopback configuration as shown in [Figure 4.53](#) or to a device under test.

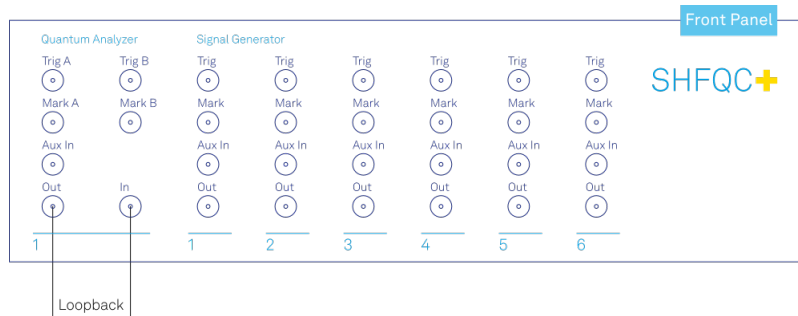


Figure 4.53: SHFQC+ connection.

Tutorial

The tutorial uses simulated qudit data as readout signal to explain how to measure integration weights and how to discriminate the qudits.

1. Connect the Instrument
Create a toolkit session to the data server and connect the Instrument with the device ID, e.g. 'DEV12001', see [Connecting to the Instrument](#).

```
# load the LabOne API and other necessary packages
from zhinst.toolkit import Session, SHFQChannelMode, Waveforms
from zhinst.utils.shfqa.multistate import QuditSettings
import numpy as np
import matplotlib.pyplot as plt
import textwrap

DEVICE_ID = 'DEVXXXXX'
SERVER_HOST = 'localhost'

session = Session(SERVER_HOST) # connect to data server
device = session.connect_device(DEVICE_ID) # connect to device
SHFQA_SAMPLING_FREQUENCY = 2.0e9 # in units of Hz
```

2. Configure the Channel
Configure the Channel using `qachannels[n].configure_channel` such that the center frequency is 5 GHz, the output range is -5 dBm, the input range is 0 dBm, and the channel mode is the Readout mode. Both input and output of the Channel are turned on.

```
CHANNEL_INDEX = 0 # physical Channel 1

device.qachannels[CHANNEL_INDEX].configure_channel(
    center_frequency=5e9, # in units of Hz
    input_range=0, # in units of dBm
    output_range=-5, # in units of dBm
    mode=SHFQChannelMode.READOUT
)
device.qachannels[CHANNEL_INDEX].input.on(1) # turn on Channel Input
device.qachannels[CHANNEL_INDEX].output.on(1) # turn on Channel Output
```

3. Generate and upload readout pulses

To simulate different qudit states, readout waveforms are generated from simulated readout envelopes, see R. Blanchetti, [PRL 105](#). The way to generate and upload readout waveforms is the same no matter whether using multistate discrimination mode or not. For each qudit state, a separate simulated waveform needs to be uploaded. Thus, the number of qudits that can be measured in a single channel is restricted by the maximum number of waveform memory slots, 8 or 16 per channel. 4 qudits (0: qutrit, 1: ququad, 2: qutrit, 3: qubit) with in total 12 states will be measured if the Instrument is SHFQA2 or SHFQC with 16W option or SHFQA4, 2 qudits (0: qutrit, 1: ququad) with in total 7 states will be measured if the instrument is SHFQA2 or SHFQC without 16W option. The simulated readout envelopes are loaded and used according to this qudits setting, the .csv file can be found in [GitHub](#). With the number of qudits, qudit type and offset frequency setting, the readout waveforms are generated using [Waveforms](#) and uploaded to the waveform memory (12 or 7 waveform memory slots are used) using [generator.write_to_waveform_memory](#), shown in [Figure 4.54](#). The plotting function can be found in the [GitHub](#). In real measurement, number of qudit readout waveforms required to be generated and uploaded is same as the number of qudits.

```
# dictionary mapping the qudit index to the number of states
if device.max_qubits_per_channel >= 16:
    QUDITS_NUM_STATES = {0: 3, 1: 4, 2: 3, 3: 2}
else:
    QUDITS_NUM_STATES = {0: 3, 1: 4}

# Note: The total number of states is restricted limited by the total
# number of waveform generator units.
total_num_states = sum(QUDITS_NUM_STATES.values())
assert total_num_states <= device.max_qubits_per_channel, (
    "Cannot upload all simulated waveforms as the total number of states, "
    f"summed over all qudits, amounts to {total_num_states}, "
    f"which is more than the number of {device.max_qubits_per_channel} "
    "generator waveforms on the device."
)

# load simulated reference traces (envelope only)
signals_simulated = np.loadtxt("example_multistate_signals.csv", dtype="complex128")

# check that enough simulated traces are available to cover all states
assert len(signals_simulated) >= max(QUDITS_NUM_STATES.values())

# Note: The number of samples will also be used for the scope measurement
num_samples = signals_simulated.shape[1]

# generate readout signal of all qudits
signals_time = np.linspace(0, num_samples / SHFQA_SAMPLING_FREQUENCY, num_samples) # time axis
QUDITS_FREQUENCIES = {0: -10e6, 1: -5e6, 2: 0e6, 3: 5e6, 4: 10e6}
# readout offset frequency
qudits_signals = {}

for qudit_idx, num_states in QUDITS_NUM_STATES.items():
    states_signals = []
    for signal_idx, signal in enumerate(signals_simulated[:num_states]):
        states_signals.append(
            signal
            * np.exp(2j * np.pi * QUDITS_FREQUENCIES[qudit_idx] * signals_time)
            / len(QUDITS_NUM_STATES) # this has to be scaled down by
            dividing the number of qudits
        )
    qudits_signals[qudit_idx] = states_signals

# convert the qudit signal to the waveform which can be uploaded to the
# memory
WAVEFORM_IDX_MAPPING = {}
simulated_waveforms = Waveforms()
```

```

waveform_idx = 0
for qudit_idx, states_signals in qudits_signals.items():
    for state_idx, signal in enumerate(states_signals):
        simulated_waveforms.assign_waveform(slot=waveform_idx,
        wave1=signal)
        WAVEFORM_IDX_MAPPING[(qudit_idx, state_idx)] = waveform_idx
        waveform_idx += 1

# upload the waveforms to the device
device.qachannels[CHANNEL_INDEX].generator.clearwave()
device.qachannels[CHANNEL_INDEX].generator.write_to_waveform_memory(simulated_waveforms)

```

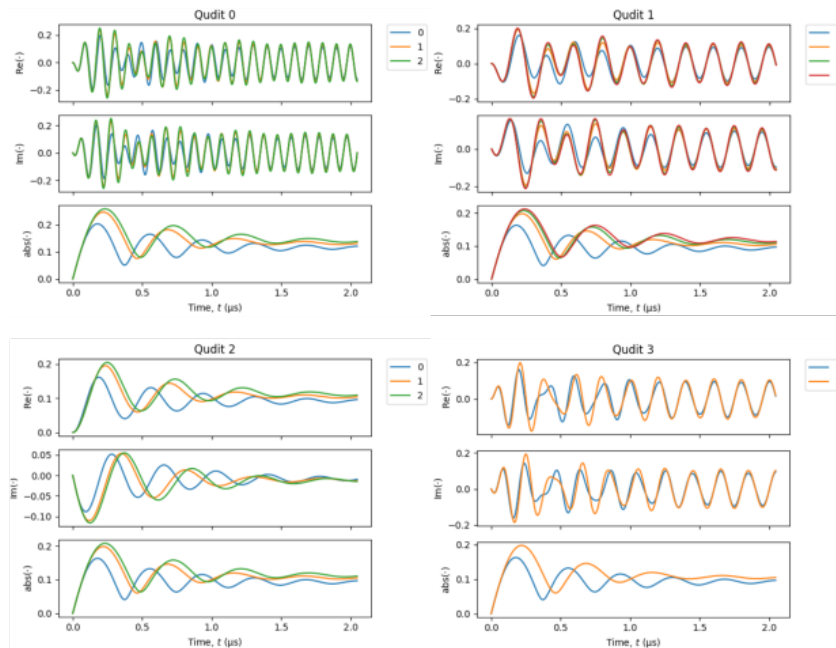


Figure 4.54: Simulated readout waveforms of 4 qudits. Blue line: state $|0\rangle$. Orange line: state $|1\rangle$. Green line: state $|2\rangle$. Red line: state $|3\rangle$.

4. Measure integration weight

Integration weights of each qudit can be calculated by taking the difference of any 2 reference traces of each qudit, i.e. n reference traces and $n(n-1)/2$ integration weights for a qudit with n states. The reference trace means the readout signal acquired by the Instrument when a qudit prepared in one of the states to be discriminated, for example state $|1\rangle$, and the rest qudits remain in ground states.

To measure reference traces, all simulated readout pulses will be sent out sequentially defined by the sequence program uploaded using `load_sequencer_program`, and recorded by the Scope. The scope is configured using `scopes[n].configure`. The SeqC code in the Generator is written such that each state of a qudit is sequentially measured in a `for` loop, and 4 qudits are therefore measured with 4 subsequent `for` loops, then the measurement is repeated 2000 times. The Scope records each state with 1 segment and averages it by 2000 times. The data downloaded after the averaging using `scopes[n].read` is then reshaped to build qudit reference traces, see Figure 4.55.

In real measurement, only one generator mask (the first argument of `startQA`) is needed for one qudit therefore using `gen_mask = (1 << {qudit_idx})` in the most inner `for` loop instead. The setting of integration mask does not matter in this step because the interested signals are the frequency down-converted signal before integration.

```

def simulated_qudit_seqc_program(
    qudits_num_states,
    wvfm_idx_mapping,
    num_repetitions,
    cycle_time=4e-6,
):
    seqc_program = textwrap.dedent(

```

```

        f"""
        const PLAY_ZERO_CYCLES = {cycle_time} * DEVICE_SAMPLE_RATE;
        info("PLAY_ZERO_CYCLES: %d", PLAY_ZERO_CYCLES);
        // repeat the measurement
        repeat({num_repetitions}) {{
    """
    )

    # generate masks to enable the integration of all qudits in the
    dictionary
    qa_int_mask = ""
    for qudit_idx in qudits_num_states.keys():
        if qa_int_mask:
            qa_int_mask += " | "
        qa_int_mask += f"QA_INT_{qudit_idx}"

    # generate n (n is the number of qudits) for loops sequentially

    # in each for loop, specific simulated qudit readout signal is used for
    each state of a qudit
    for qudit_idx, num_states in qudits_num_states.items():
        first_wave_idx = wvfm_idx_mapping[(qudit_idx, 0)] # index of the
        first state of the qudit

        seqc_program += textwrap.indent(
            textwrap.dedent(
                f"""
                // generate and measure reference traces for qudit {qudit_idx}
                for(cvar i = 0; i < {num_states}; i++) {{
                    // mask to enable the playback of the simulated trace
                    // for a specific qudit state
                    cvar gen_mask = (1 << ({first_wave_idx} + i));
                    // cvar gen_mask = (1 << {qudit_idx}); // for real
measurement

                    // wait for the next repetition period
                    playZero(PLAY_ZERO_CYCLES);

                    // play back different waveforms based on the bit mask
                    // and measure the qudit
                    startQA(gen_mask, {qa_int_mask}, true, 0, 0x0);
                }}
                """
            ),
            "    ",
        )

        seqc_program += textwrap.dedent(
            """
            } // end of repeat({num_repetitions})
            """
        )
    )
    return seqc_program

# generate and upload the sequence
NUM_REPETITIONS = 2000
seqc_program = simulated_qudit_seqc_program(
    QUDITS_NUM_STATES, WAVEFORM_IDX_MAPPING, num_repetitions=NUM_REPETITIONS
)
device.qachannels[CHANNEL_INDEX].generator.load_sequencer_program(seqc_program)

# configure the scope

```

```

SCOPE_IDX = 0 # only one scope on the device
SCOPE_CHANNEL = 0 # from 0 to 3, 4 in total
SCOPE_TRIGGER_CHANNEL = f"chan{CHANNEL_INDEX}seqmon0" # the scope will be
triggered by the sequence monitor trigger
SCOPE_TRIGGER_DELAY = 200e-9
# start recording 200 ns later after receiving a trigger

device.scopes[SCOPE_IDX].configure(
    input_select={SCOPE_CHANNEL: f"channel{CHANNEL_INDEX}_signal_input"},
    num_samples=num_samples,
    trigger_input=SCOPE_TRIGGER_CHANNEL,
    num_segments=total_num_states,
    num_averages=NUM_REPETITIONS,
    trigger_delay=SCOPE_TRIGGER_DELAY,
)

# arm the scope
device.scopes[SCOPE_CHANNEL].run(single=True)

# set the integration delay equals to the scope trigger delay,
# so the recorded data can be used for state discrimination directly.
device.qachannels[CHANNEL_INDEX].readout.integration.delay(SCOPE_TRIGGER_DE
LAY)

# run the sequencer
device.qachannels[CHANNEL_INDEX].generator.enable_sequencer(single=True)

# get the scope results and reshape it
scope_data, *_ = device.scopes[SCOPE_IDX].read()
scope_data_segments = np.reshape(
    scope_data[SCOPE_CHANNEL], [total_num_states, num_samples]
)

# build list of reference traces for each qudit
qudits_ref_traces = {}
for qudit_idx, num_states in QUDITS_NUM_STATES.items():
    ref_traces = []
    for state_idx in range(num_states):
        ref_traces.append(
            scope_data_segments[WAVEFORM_IDX_MAPPING[(qudit_idx, state_idx)
]]
    )

    qudits_ref_traces[qudit_idx] = ref_traces

```

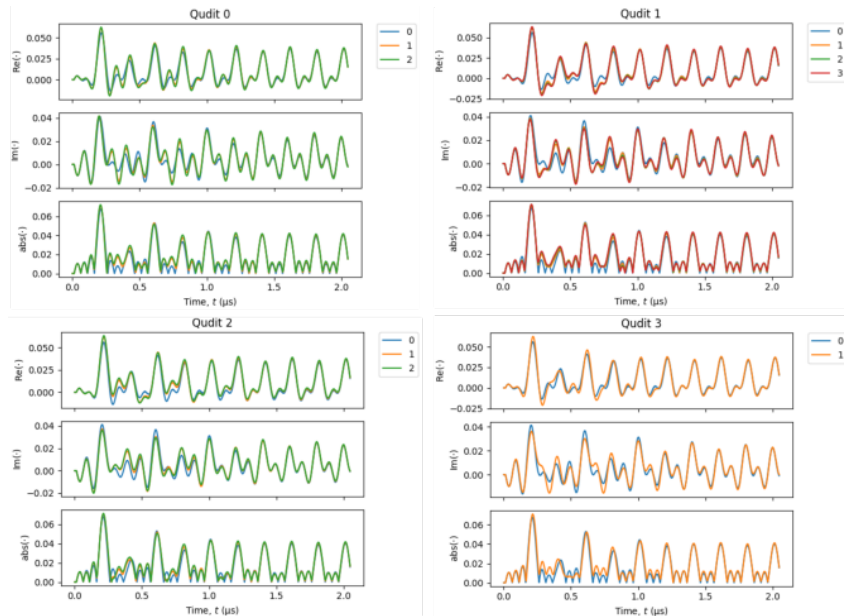



Figure 4.55: Reference traces of 4 qudits. Blue line: qudit in state $|0\rangle$ and the rest qudits are in state $|0\rangle$. Orange line: qudit in state $|1\rangle$ and the rest qudits are in state $|0\rangle$. Green line: qudit in state $|2\rangle$ and the rest qudits are in state $|0\rangle$. Red line: qudit in state $|3\rangle$ and the rest qudits are state in $|0\rangle$.

All integration weights calculated by the utility function `QuditSettings` with the reference traces are shown in Figure 4.56.

```
all_qudit_settings = { }
for qudit_idx, ref_traces in qudits_ref_traces.items():
    all_qudit_settings[qudit_idx] = QuditSettings(ref_traces)
```

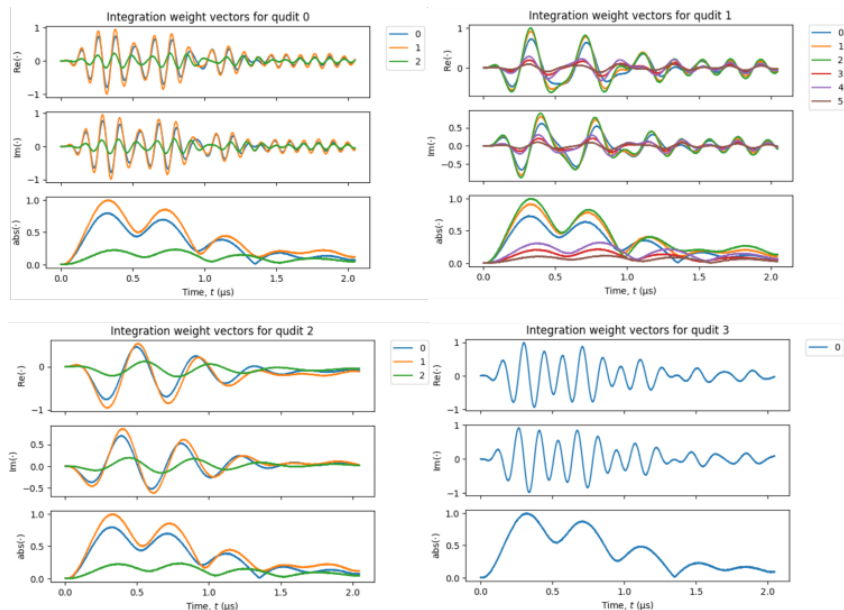


Figure 4.56: Integration weights of 4 qudits. For a qudit with

5. Discriminate qudit state

To discriminate qudit state which is defined by `qudits[n].configure`, thresholds and assignment matrix are required. Using `readout.configure_result_logger` to configure the source of readout result from `result_of_discrimination`.

Thresholds are used to discriminate the results after integration and get 0 or 1, and assignment matrix are used to convert the result after thresholding to the correct digital representation of qudit state are required. These are calculated by the utility function `QuditSettings`' too, therefore no additional

measurement and manual calculation are needed. []

(`#shfqa_fig_tutorial_msd_thresholds`) shows the

histogram of qudits at each state and the thresholds returned from

[`QuditSettings`]. The script to measure and plot the histogram can be found on GitHub and Online Documentation.

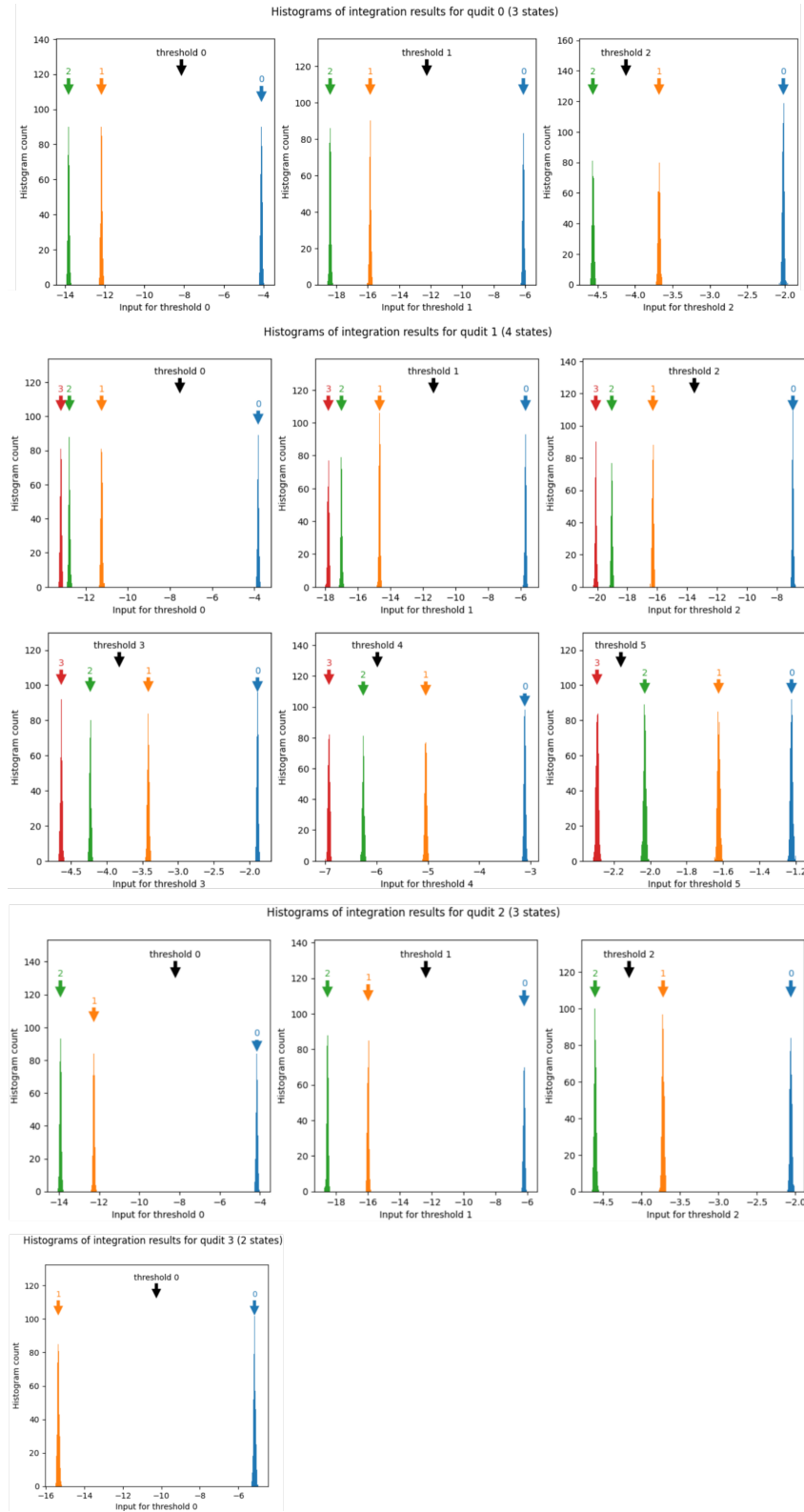


Figure 4.57: Histogram of qudits at different states and thresholds. Blue: qudit in state $|0\rangle$. Orange: qudit in state $|1\rangle$. Green: qudit in state $|2\rangle$. Red: qudit in state $|3\rangle$. For qudit with

The default state discrimination mode is 2-state discrimination, so the multistate readout mode has to be enabled for this measurement. Before upload new qudit settings, all qudits should be disabled to avoid mixing up of the new and old settings. The readout result source has to be `[result_of_discrimination]` to get digital results representing qudit states. To show how measurement result looks like after state discrimination, the same sequence program used for integration weight measurement, and the simulated qudit readout waveforms are reused. The Instrument sequentially sends out simulated readout signal according to each state of all qudits, and integrates the down-converted signal with $n - 1$ weights and gets $n(n - 1)/2$ integrated results, where n is the number of qudit states. In the sequence, 4

integrator masks are used for all qudits for simplicity, and only first $n - 1$ integrators are used directly for integration. The $n(n - 1)/2$ results are from the direct integration with $n - 1$ integrators and $(n - 1)(n - 2)/2$ differences of the integration results. They are discriminated by $n(n - 1)/2$ thresholds, and then converted by the assignment matrix to 2-bit data representing the qudit state. The measurement is repeated 2000 times and the first 24 results of all qudits are shown in Figure 4.58. Based on the measured results downloaded using `get_qudits_results` and expected readout states, the fidelity matrix can be calculated. All calculation and plotting functions are detailed on the [Online Documentation](#) page and the [GitHub](#) page.

```
# enable the multistate discrimination
device.qachannels[CHANNEL_INDEX].readout.multistate.enable(1)

# disable all qudits before configure them
device.qachannels[CHANNEL_INDEX].readout.multistate.qudits["*"].enable(0)

# configure the new qudit settings on the device
for qudit_idx, qudit_settings in all_qudit_settings.items():
    device.qachannels[CHANNEL_INDEX].readout.multistate.qudits[qudit_idx].configure(
        qudit_settings
    )

result_length = NUM_REPETITIONS * total_num_states

# configure the result logger
device.qachannels[CHANNEL_INDEX].readout.configure_result_logger(
    result_length=result_length, result_source="result_of_discrimination"
)

# arm the result logger
device.qachannels[CHANNEL_INDEX].readout.run()

# run the sequencer
device.qachannels[CHANNEL_INDEX].generator.enable_sequencer(single=True)

# download the results
qudits_results = device.qachannels[
    CHANNEL_INDEX
].readout.multistate.get_qudits_results()
```

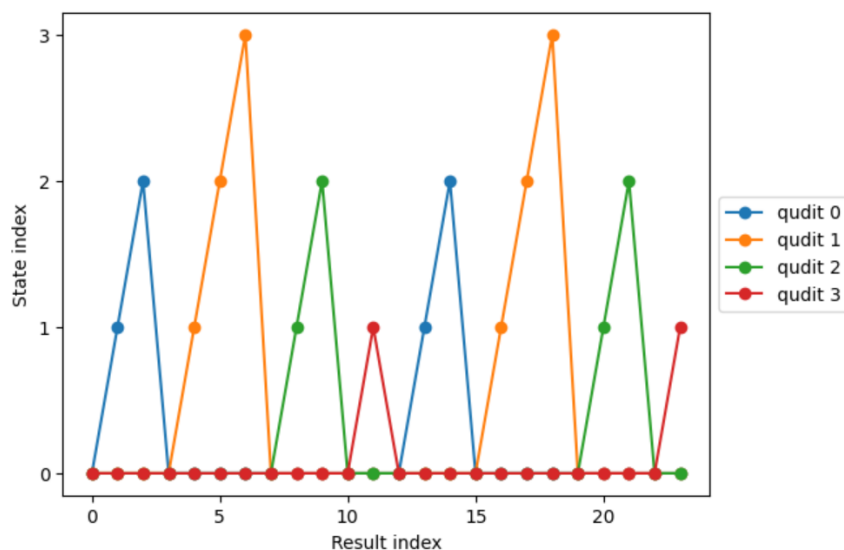


Figure 4.58: Multistate discrimination of qudits. Only 24 out of 2000 data are shown in the plot.

In real measurement, multistate discrimination of multiple qudits can be done in parallel by running a new SeqC program shown below before the above script. In the sequence, output readout waveform is generated by adding up 4 readout waveforms according to 4 qudits, and

12 integrators according to 12 states in total are used. The result after discrimination can be sent to a control instrument for feedback experiment via DIO or via ZSync through a [PQSC](#), e.g. active reset.

```
seqc_program = textwrap.dedent(
    """
    const PLAY_ZERO_CYCLES = 4e-6 * 2e9;
    // repeat the measurement
    repeat(2000) {
        // wait for the next repetition period
        playZero(PLAY_ZERO_CYCLES);

        // play back readout waveform of 4 qudits
        // and measure the qudits
        startQA(QA_GEN_0 | QA_GEN_1 | QA_GEN_2 | QA_GEN_3, QA_INT_ALL,
            true, 0, 0x0);
    }
    """
)
```

4.2.5. Power Spectrum Density Measurement

Note

This tutorial is applicable to all SHFQC+ Instruments.

Goals and Requirements

[LabOne Q](#) is the recommended control software to operate the SHFQC+ for Quantum Technology applications.

The goal of this tutorial is to demonstrate how to use the SHFQC+ to perform power spectral density measurement using the LabOne User Interface (UI) and [Zurich Instruments Toolkit API](#).

Note

For zhinst-toolkit users, please find the example in https://github.com/zhinst/zhinst-toolkit/blob/main/examples/shfqa_shfqc_power_spectral_density.md, and the zhinst-toolkit documentation.

Preparation

Please follow the preparation steps in [Connecting to the Instrument](#) and connect the instrument in a loopback configuration as shown in [Figure 4.59](#) or to a device under test.

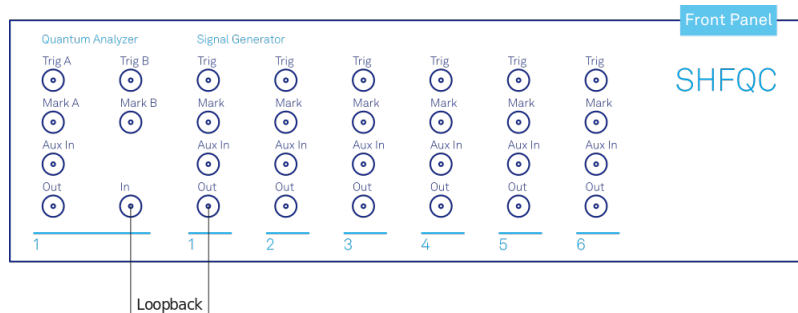


Figure 4.59: SHFQC+ connection.

Tutorial

In this tutorial, we use a SHFQC+ channel to generate a signal under test, and measure the power spectral density of the signal when it is on and off using another channel.

LabOne UI

This section shows how to use LabOne UI to configure the instrument, run the measurement and monitor the measurement results.

1. Configure the instrument
 1. Set center frequency and power range of input and output signals
Configure these parameters on the [Input and Output Tab](#) as in [Figure 4.60](#) and in [Table 4.43](#).

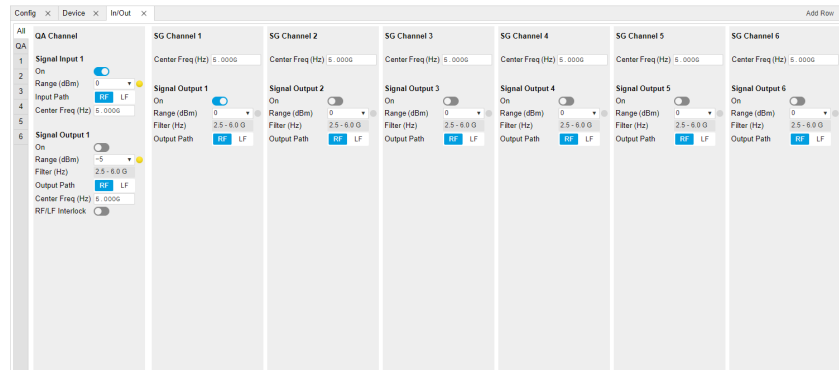


Figure 4.60: Configurations on In/Out Tab.

Table 4.43: Settings on In/Out Tab

Parameter	Setting	Description
QA Channel Selection	All	Select All to display all Channels.
Cent Freq (Hz)	5 GHz	Set center frequency of the QA Channel and the SG Channel 1 to 5 GHz.
Signal Input 1 On	Enable	Enable the Signal Input.
Signal Input 1 Range (dBm)	0 dBm	Set power range of Signal Input to 0 dBm. This setting allows the instrument to acquire a input signal with a power up to 0 dBm.
Signal Input 1 Input Path	RF	Set input path of the Signal Input to RF path.
SG Signal Output 1 On	Enable	Enable the Signal Output.
SG Signal Output 1 Range	0 dBm	Set power range of Signal Output to 0 dBm. This setting allows the instrument to output a signal with a power up to 0 dBm.
SG Signal Output 1 Output Path	RF	Set output path of Signal Output to RF path.

2. Upload and compile measurement sequence
The measurement sequence is defined on the Sequence Sub-Tab of the [Readout Pulse Generator Tab](#), see [Figure 4.61](#) and [Table 4.44](#).
The power spectral density of input signal is measured by calculating square of measurement result after integration at different frequencies as $S_{xx}(f) = \lim_{N \rightarrow \infty} \frac{(\Delta t)^2}{T} \left| \sum_{n=-N}^N x_n e^{-i2\pi f n \Delta t} \right|^2$, where $\Delta t = 1/f_s$ is the time step, f_s is the sampling rate, $T = (2N + 1)\Delta t$ is the integration length in seconds, $2N + 1$ is the integration length in samples, x_n is the n -th complex data of the input signal, $e^{-i2\pi f n \Delta t}$ is the integration weight. By sweeping the frequency of integration weight, the power spectral density of input signal is calculated by the instrument, and it returns the real-valued power spectral density in units of V_{rms}^2/Hz .

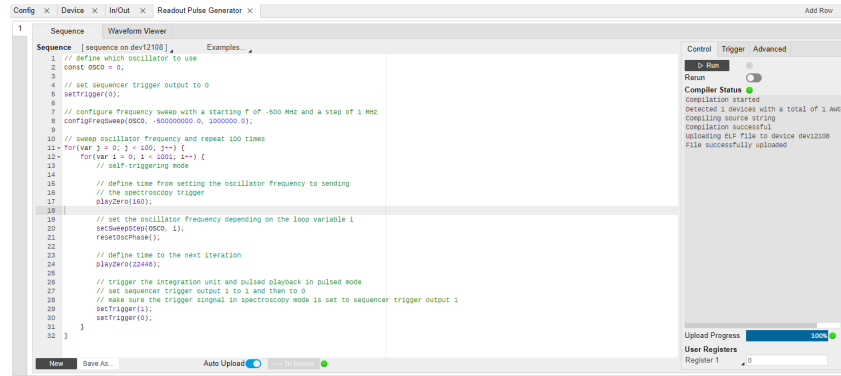


Figure 4.61: Configurations on Readout Pulse Generator Tab.

Table 4.44: Settings of Readout Pulse Generator Tab

Parameter	Setting	Description
Sub-Tab Display	Sequence	Select Sequence sub-tab and paste the sequence program below or load <code>ziGenerator_functional_spectroscopy.seqc</code> from the "Examples" library and modify it.
Compile	Click "To Device"	Compile the sequence program by clicking "To Device".
Return	Disable	Disable return function.
Run/Stop	Disable	Disable Run/Stop.

Below is the the sequence program for the measurement. In the inner loop, the `setSweepStep(OSC0, i)` command sets frequency of digital oscillator 0 to i-th frequency in an array configured by `configFreqSweep(OSC0, -500000000.0, 1000000.0)` command, the `setTrigger(value)` command sets Sequencer Trigger 1 Output to high and then low to start integration, and waveform generation if pulsed waveform is desired, the `playZero(samples)` command define time to the next `playZero(samples)`. The measurement is repeated 100 times by the outer loop.

```
// define which oscillator to use
const OSC0 = 0;

// set sequencer trigger output to 0
setTrigger(0);

// configure frequency sweep with a starting f of -500 MHz and a step of 1 MHz
configFreqSweep(OSC0, -500000000.0, 1000000.0);

// sweep oscillator frequency and repeat 100 times
for(var j = 0; j < 100; j++) {
    for(var i = 0; i < 1001; i++) {
        // self-triggering mode

        // define time from setting the oscillator frequency to
        sending
        // the spectroscopy trigger
        playZero(160);

        // set the oscillator frequency depending on the loop variable
        i
        setSweepStep(OSC0, i);
        resetOscPhase();

        // define time to the next iteration
        playZero(22448);

        // trigger the integration unit and pulsed playback in pulsed
```

```
mode
    // set sequencer trigger output 1 to 1 and then to 0
    // make sure the trigger singnal in spectroscopy mode is set
to sequencer trigger output 1
    setTrigger(1);
    setTrigger(0);
}
```

3. Configure signal generation and data acquisition
The input signal under test is generated using SG Channel 1 as shown in [Figure 4.62](#) and [Table 4.45](#).

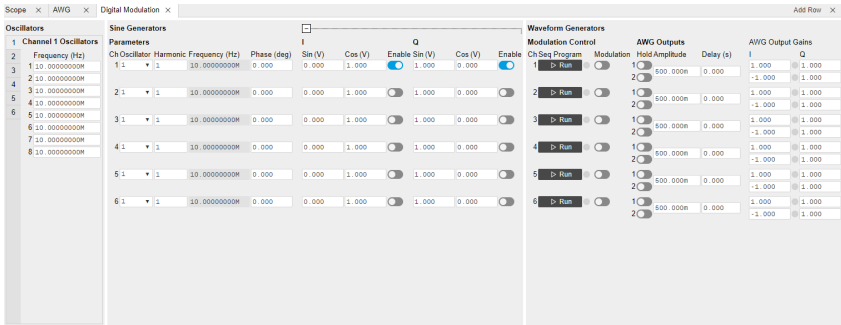


Figure 4.62: Configurations of SG Channel 1 on Digital Modulation Tab.

Table 4.45: Settings of Digital Modulation Tab.

Parameter	Setting	Description
Channel 1 Oscillators	1	Select SG Channel 1.
Frequency (Hz)	10 MHz	Set oscillator frequency.
Sine Generators Channel 1 Oscillator	1	Select oscillator 1.
Sine Generators Channel 1 Harmonic	1	Set harmonic order of the oscillator.
Sine Generators Channel 1 I Sin (V)	0	Set amplitude of Sin component to 0.
Sine Generators Channel 1 I Cos (V)	1	Set amplitude of Cos component to 1.
Sine Generators Channel 1 Q Sin (V)	1	Set amplitude of Sin component to 0.
Sine Generators Channel 1 Q Cos (V)	0	Set amplitude of Cos component to 1.
Sine Generators Channel 1 I Enable	Enable	Enable I component.
Sine Generators Channel 1 Q Enable	Enable	Enable Q component.

Data acquisition are defined on [QA Setup Tab](#) and [QA Result logger Tab](#), see [Figure 4.63](#) and [Table 4.46](#).

The input signal after frequency down-conversion is integrated at each frequency for 512 ns started 224 ns later after receiving a trigger from Sequencer 1 Trigger Output 1, and the frequency sweep is repeated 100 times. The result after each integration is squared and then averaged cyclically.

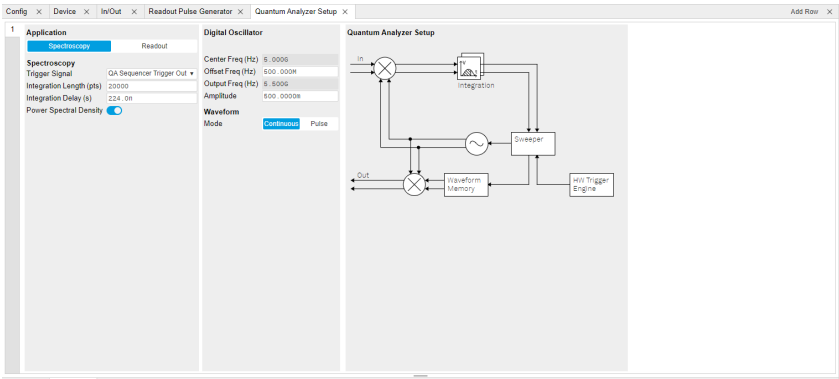


Figure 4.63: Configurations of QA Channel 2 on QA Setup Tab.

Table 4.46: Settings of QA Setup Tab

Parameter	Setting	Description
Application Mode	Spectroscopy	Use spectroscopy mode for power spectral density measurement.
Trigger Signal	QA Sequencer Trigger Output 1	Select the right trigger source to integration. This selection matches the trigger setting written in the sequence program.
Integration Length (pts)	20000	Set the integration length in number of samples. Note that the integration length which defines measurement bandwidth must to be much longer than $1/\Delta f$, where Δf is the frequency step set in sequence program. In this tutorial, integration length is 10 μ s, and frequency stem is 1 MHz.
Power Spectral Density	Enable	Enable the Power Spectral Density measurement function.

On the QA Result Logger Tab, it defines how the result is averaged and displayed. After all configuration, the QA Result Logger should be enabled to be ready to receive measurement results, see settings in [Table 4.47](#)

Table 4.47: Settings of QA Result Logger Tab

Parameter	Setting	Description
Sub-Tab	Spectroscopy	Select Spectroscopy sub-tab to monitor measurement result when using the Spectroscopy mode.
Plot Type	Components	Select Components to display I, Q, amplitude or phase vs sample points. Select Dot Plot to display I vs Q with scattered dots.
Result Length (Sample)	1001	Set result length in number of samples. The number must match what is set in the sequence program.
Averages	100	Set the number of averages. The number must match what is set in the sequence program.
Average Mode	Cyclic	Set the average mode to cyclic. This setting must match how the loop is configured in the sequence program.
Vertical Axis Groups	Add amplitude and phase	Select amplitude and phase to be displayed on the plot.
Run/Stop	Enable	Enable the result logger to receive and display measurement results.

2. Run the measurement
By clicking "Run/Stop" icon on the Readout Pulse Generation Tab, the measurement is started and finished in seconds.
3. Monitor the measurement result
The measurement result is normalized by the integration length and displayed on QA Result Tab, as shown in [Figure 4.64](#). To remove the background noise from the instrument and

environment, repeat the measurement with Signal Output 4 is OFF, see the result in [Figure 4.65](#).

The unit of returned result in linear scale is V_{rms}^2/Hz . On the plot, the result is displayed in dB scale which is derived by $dBV_{rms}^2/Hz = 20 \log(V_{rms}^2/Hz)$. The result in units of dBV_{rms}^2/Hz can be converted to dBm/Hz by $\frac{P_{dBV_{rms}^2/Hz}}{2} + 13$, where $P_{dBV_{rms}^2/Hz}$ is the result shown in units of dBV_{rms}^2/Hz . Therefore the power spectral density at at 5.01 GHz is -61.3 dBm/Hz. With configured output power of Signal Output 4 about -6 dBm, and measurement bandwidth $1/10\mu s = 100$ kHz, the power spectral density is $-6 - \log(10^5) = -56$ dBm/Hz. The difference between the 2 values is attribute to the uncertainty of the input and output power and the attenuation over the signal path.

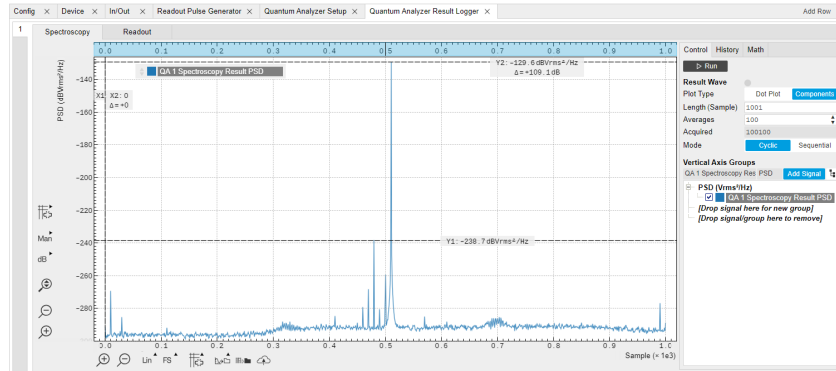


Figure 4.64: power spectral Density when input signal is ON on QA Result Logger Tab.

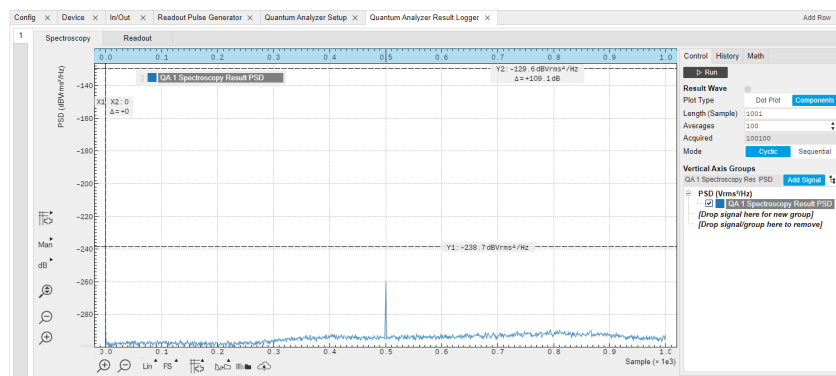


Figure 4.65: power spectral Density when input signal is OFF on QA Result Logger Tab.

zhinst-toolkit

Python API [SHFSweeper](#) class is the core of the power spectral density measurements. It defines all relevant parameters for frequency sweeping and sequencing. Toolkit wraps around the SHFSweeper and exposes an interface that is similar to the LabOne modules, meaning the parameters are exposed in a node tree like structure. The power spectral density of input signal is measured using the SHFQC+ Sweeper as for [Resonator Spectroscopy Measurement](#) with `sweeper.sweep.psd(True)`.

By calculating square of measurement result after integration at different frequencies, the power spectral density at frequency f is $S_{xx}(f) = \lim_{N \rightarrow \infty} \frac{(\Delta t)^2}{T} |\sum_{n=-N}^N x_n e^{-i2\pi f n \Delta t}|^2$, where $\Delta t = 1/f_s$ is the time step, f_s is the sampling rate, $T = (2N + 1)\Delta t$ is the integration length in seconds, $2N + 1$ is the integration length in samples, x_n is the n -th complex data of the input signal, $e^{-i2\pi f n \Delta t}$ is the integration weight. By sweeping the frequency of integration weight, the power spectral density of input signal is calculated by the instrument, and it returns the real-valued power spectral density in units of V_{rms}^2/Hz .

1. Connect the Instrument
Create a toolkit session to the data server and connect the Instrument with the device ID, e.g. 'DEV12001', see [Connecting to the Instrument](#).

```
# Load the LabOne API and other necessary packages
from zhinst.toolkit import Session
```



```

DEVICE_ID = 'DEVXXXXX'
SERVER_HOST = 'localhost'

session = Session(SERVER_HOST)          ## connect to data server
device = session.connect_device(DEVICE_ID) ## connect to device

```

2. Create a Sweeper and configure it

```

sweeper = session.modules.shfqa_sweeper
sweeper.device(device)

# configure QA channel
CHANNEL_INDEX = 0 # QA Channel

sweeper.sweep.start_freq(-500e6) # in units of Hz
sweeper.sweep.stop_freq(500e6) # in units of Hz
sweeper.sweep.num_points(1001)
sweeper.average.num_averages(100)

sweeper.average.integration_time(10e-6) # in units of second
sweeper.sweep.wait_after_integration(1e-6)
# waiting 1 us after integration before starting a new measurement

sweeper.average.mode("cyclic") # "sequential" or "cyclic" averaging

sweeper.rf.channel(CHANNEL_INDEX)
sweeper.rf.center_freq(5e9) # in units of Hz
sweeper.rf.input_range(0) # in units of dBm
sweeper.sweep.psd(True)

device.qachannels[CHANNEL_INDEX].input.on(1)

```

The sweep parameters are configured such that the offset frequency of integration weight is linearly swept over 1001 points from -500 MHz to 500 MHz around the center frequency of 5 GHz with the input power range of 0 dBm and the oscillator gain of 0.5, the input signal is integrated for 10 μ s after receiving a trigger, and the measurement is repeated 100 times. There are 2 sweeping modes, sequencer-based mode (default) and host-driven mode. The sequencer-based mode is recommended to achieve the highest sweep speed. In the sequencer-based mode, the frequency sweep is done by updating the frequency of the digital oscillator via the SeqC commands `configFreqSweep` and `setSweepStep`. This mode allows a fast resonator spectroscopy measurement with predicted cycle time of $t_{\text{settling time}} + t_{\text{integration delay}} + t_{\text{integration time}} + t_{\text{wait after integration}}$. In the host-driven mode, the frequency sweep is done by updating the frequency of the digital oscillator via software, therefore measurement speed is limited by communication between instrument and host computer. Logarithmic or other non-linear frequency sweep is supported by the host-driven sweep mode only.

To get a better signal-to-noise ratio (SNR), the measurement is repeated, and the result is averaged. There are 2 ways for averaging, sequential and cyclic. In the sequential mode, measurement is repeated after each frequency step. In the cyclic mode, measurement is repeated after sweeping through all frequencies.

3. Configure a signal under test

For simplicity, we generate a continuous waveform using SG Output 1 with the following codes.

```

CHANNEL_INDEX_OUTPUT = 0 # SG channel 1

with device.set_transaction():
    device.synthesizers[1].centerfreq(5e9) # center frequency in units of
    Hz
    device.sgchannels[CHANNEL_INDEX_OUTPUT].output.range(0) # power range
    in units of dBm
    device.sgchannels[CHANNEL_INDEX_OUTPUT].output.on(1) # turn on output
    device.sgchannels[CHANNEL_INDEX_OUTPUT].oscs[0].freq(10e6)
    device.sgchannels[CHANNEL_INDEX_OUTPUT].sines[0].i.enable(1)
    device.sgchannels[CHANNEL_INDEX_OUTPUT].sines[0].q.enable(1)

```

4. Run the measurement and plot the data

```
psd = sweeper.run()
sweeper.plot()
```

After executing `sweeper.run()`, all above parameters are updated, and a SeqC program is automatically generated, uploaded and compiled based on the sweep parameters, see in [Figure 4.66](#). In the program, `ConfigFreqSweep` sets the start frequency and the frequency increment in units of Hz for a chosen oscillator, and `setSweepStep` sets the oscillator frequency. The oscillator phase is reset by `resetOscPhase` before each measurement. The trigger generated in the sequencer is used to start the integration, and the `playZero` sets the cycle duration. The measurement is repeated using the nested `for` loop according to the averaging mode. The measurement starts after enabling the sequencer.

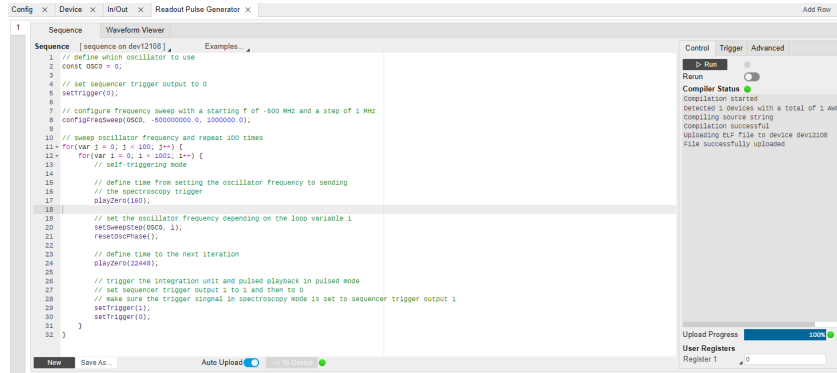


Figure 4.66: SeqC program in the Readout Pulse Generator Tab.

The result returned from `sweeper.run()` is the power spectral density in units of Vrms^2/Hz . It is averaged and normalized by the integration length. The `sweeper.plot()` converts the unit of power spectral density from Vrms^2/Hz to dBm/Hz by

$$P_{\text{dBm}/\text{Hz}} = 10 \log\left(\frac{P_{\text{Vrms}^2/\text{Hz}} \times 1000}{R}\right), \quad (1)$$

where R is $50 \, \Omega$, 1000 is the conversion factor from W to mW. The power spectral density in units of dBm/Hz is shown in [Figure 4.67](#).

To remove the background noise from the instrument and environment repeat the measurement with Signal Output is disabled.

Sweep with center frequency 5.0GHz

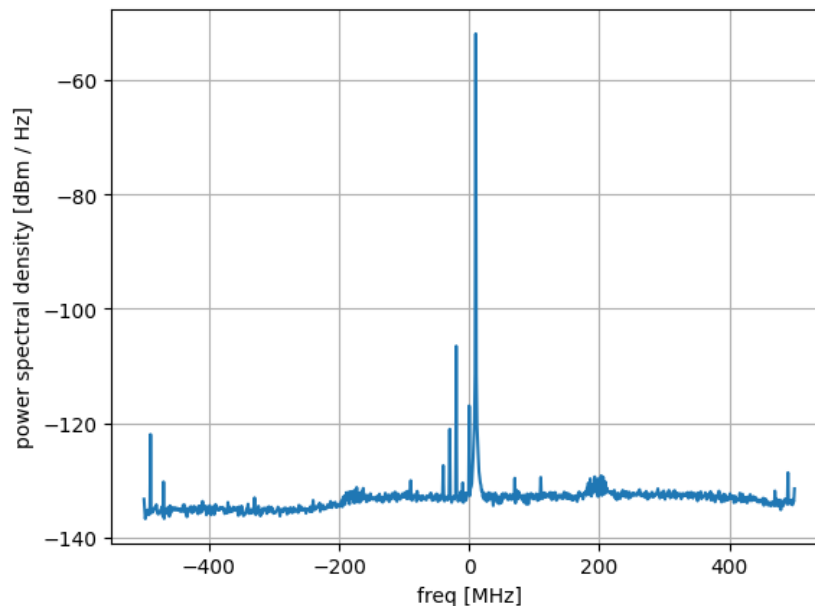


Figure 4.67: Power spectral density of signal from the SG Signal Output 1.

4.2.6. Long Readout Time (LRT) Option

Note

This tutorial is applicable to all SHFQC+ Instruments.

Goals and Requirements

[LabOne Q](#) is the recommended control software to operate the SHFQC+ for Quantum Technology applications.

The goal of this tutorial is to demonstrate how to use SHFQC+ and the Long Readout Time (LRT) option to perform long qubit readout using the LabOne User Interface (UI) and [Zurich Instruments Toolkit API](#).

For qubit control with the SHFSG+ Signal Generator, the SHFQC+ Qubit Controller and the HDAWG Arbitrary Wave Generator, and instrument synchronization and feedback with the PQSC Programmable Quantum System Controller, see tutorials section under a specific instrument found in the [Online Documentation](#).

Preparation

Please follow the preparation steps in [Connecting to the Instrument](#) and connect the instrument in a loopback configuration as shown in [Figure 4.68](#) or to a signal under test.

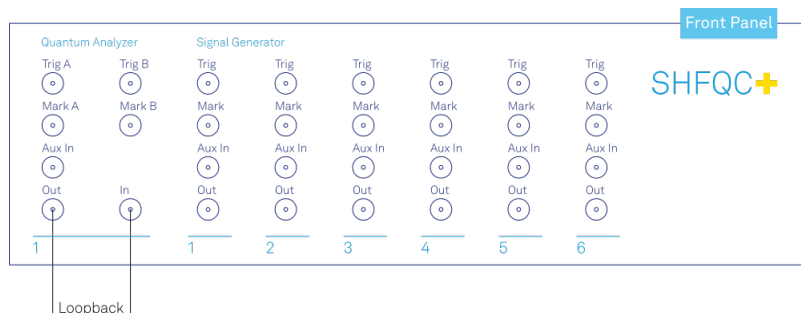


Figure 4.68: SHFQC+ connection.

Tutorial

In this tutorial, the instrument runs a measurement that performs long readout of 2 qubits in parallel. The measurement is repeated 100×100 times and 100 averaged complex-valued results are returned.

LabOne UI

This section shows how to use LabOne UI to configure the instrument, run the measurement and monitor the measurement results.

1. Configure the instrument
 1. Set center frequency and power range of input and output signals
Configure these parameters on the [Input and Output Tab](#) as in [Figure 4.69](#) and in [Table 4.48](#).

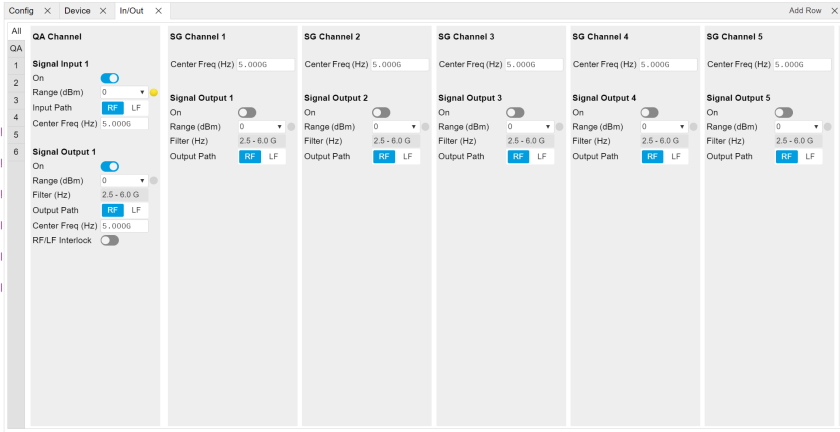


Figure 4.69: Configurations on In/Out Tab.

Table 4.48: Settings of QA Channel 1 on In/Out Tab

Parameter	Setting	Description
QA Channel Selection	All	Select All to display all Channels.
Cent Freq (Hz)	5 GHz	Set center frequency of the frequency sweep.
Signal Input 1 On	Enable	Enable the Signal Input 1.
Signal Input 1 Range (dBm)	0 dBm	Set power range of Signal Input 1 to 0 dBm. This setting allows the instrument to acquire an input signal with a power up to 0 dBm.
Signal Input 1 Input Path	RF	Set input path of Signal Input 1 to RF path.
Signal Output 1 On	Enable	Enable the Signal Output 1.
Signal Output 1 Range	0 dBm	Set power range of Signal Output 1 to 0 dBm. This setting allows the instrument to output a signal with a power up to 0 dBm.
Signal Output 1 Output Path	RF	Set output path of the Signal Output 1 to RF path.

2. Upload and compile measurement sequence
The measurement sequence is defined on the Sequence Sub-Tab of the [Readout Pulse Generator Tab](#), see [Figure 4.70](#) and [Figure 4.70](#).

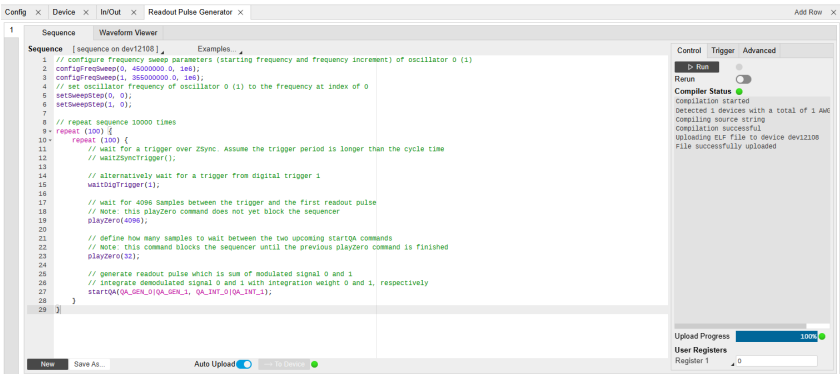


Figure 4.70: Configurations on Readout Pulse Generator Tab.

Table 4.49: Settings of QA Channel 1 on Readout Pulse Generator Tab.

Parameter	Setting	Description
QA Channel Selection	1	Select QA Channel 1.

Parameter	Setting	Description
Sub-Tab Display	Sequence	Select Sequence sub-tab and paste the sequence program below. The Waveform Viewer sub-tab displays waveforms saved in Waveform Memory slots and Integration Weight units.
Compile	Click "To Device"	Compile the sequence program by clicking "To Device".
Digital Triggers	Digital Trigger 1	Select Digital Trigger 1.
Digital Trigger 1 Signal	Internal Trigger	Select Internal Trigger as the trigger source of Digital Trigger 1.
Return	Disable	Disable return function.
Run/Stop	Enable	Run the sequence.

Below is the sequence program for the measurement. In the inner loop, the `waitDigTrigger(1)` command waits for a digital trigger to continue the sequence, the first `playZero` command sets the waiting time after receiving a trigger before running the `startQA` command, the `startQA(QA_GEN_0|QA_GEN_1, QA_INT_0|QA_INT_1, true)` command sends a trigger to generate output waveform and start integration. The measurement is repeated 100 times by the outer loop.

```
// configure frequency sweep parameters (starting frequency and
frequency increment) of oscillator 0 (1)
configFreqSweep(0, 45000000.0, 1e6);
configFreqSweep(1, 355000000.0, 1e6);
// set oscillator frequency of oscillator 0 (1) to the frequency at
index of 0
setSweepStep(0, 0);
setSweepStep(1, 0);

// repeat sequence 10000 times
repeat (100) {
    repeat (100) {
        // wait for a trigger over ZSync. Assume the trigger period is
longer than the cycle time
        // waitZSyncTrigger();

        // alternatively wait for a trigger from digital trigger 1
        waitDigTrigger(1);

        // wait for 4096 Samples between the trigger and the first
readout pulse
        // Note: this playZero command does not yet block the
sequencer
        playZero(4096);

        // define how many samples to wait between the two upcoming
startQA commands
        // Note: this command blocks the sequencer until the previous
playZero command is finished
        playZero(32);

        // generate readout pulse which is sum of modulated signal 0
and 1
        // integrate demodulated signal 0 and 1 with integration
weight 0 and 1, respectively
        startQA(QA_GEN_0|QA_GEN_1, QA_INT_0|QA_INT_1);
    }
}
```

The digital trigger set on the Trigger sub-tab is the Internal Trigger. The configuration of the Internal Trigger is shown in [Figure 4.71](#) and [Table 4.50](#).

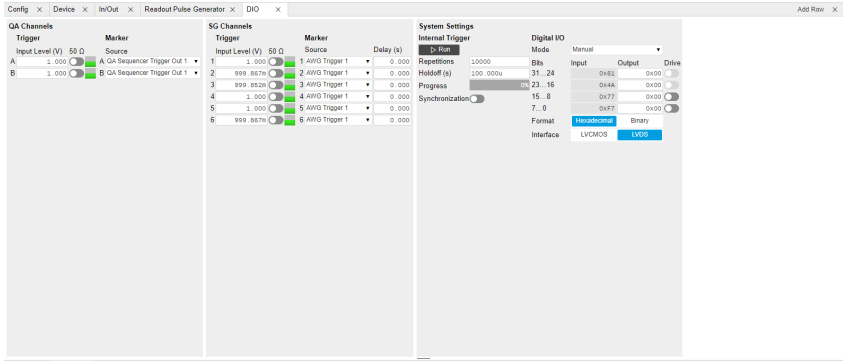


Figure 4.71: Configurations of Internal Trigger on DIO Tab.

Table 4.50: Settings of Internal Trigger on DIO Setup Tab, see details on [DIO Tab](#)

Parameter	Setting	Description
Repetitions	10000	Set number of repetitions.
Holdoff (s)	100u	Set holdoff time to 100 μ s.
Synchronization	Disable	Disable Synchronization.
Run/Stop	Disable	Disable the internal trigger.

3. Configure signal generation and data acquisition
Signal generation and data acquisition are defined on [QA Setup Tab](#) and [QA Result logger Tab](#).

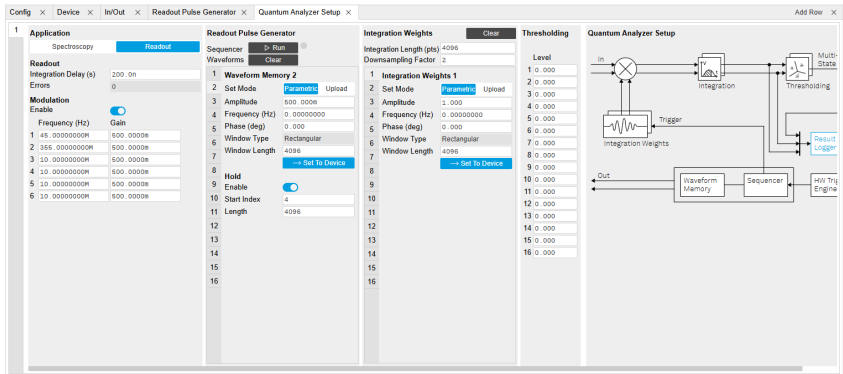


Figure 4.72: Configurations on QA Setup Tab.

The readout waveform is generated by summing up of modulated signal 0 and 1, where the modulated signal 0 (1) is an envelope waveform saved in Waveform Memory slot 0 (1) modulated by the numerical oscillator 0 (1). The length of envelope waveform 0 (1) is doubled by holding the 4th sample of the envelope with 4096 Samples for length of 4096 Samples long.

The input signal is demodulated by oscillator 0 and 1, and then integrated with integration weights 0 and 1, respectively. The integration weight length and integration length is doubled by configuring the downsampling factor to 2, so the integration weight is extended by inserting the same sample after each original weight sample, i.e. the original weight data is **a1, a2...a4096**, the extended weight with the downsampling factor of 2 is **a1, a1, a2, a2...a4096, a4096**.

Note

The power consumption of the instrument increases when the "Modulation" is enabled. Please ensure a proper airflow and ambient temperature around the instrument when the modulation is enabled, and disable it if no longer required.

We use Readout mode for multiplex qubit readout ([Table 4.51](#)) and generate both readout envelope waveforms and integration weights parametrically ([Table 4.52](#) and [Table 4.53](#)). To achieve high readout fidelity optimal weights should be measured and uploaded, see how to measure optimal weights in [Integration Weights Measurement](#). Thresholds need to be uploaded if qubit state discrimination is required.

Table 4.51: Application Settings of QA Channel 1 on QA Setup Tab.

Parameter	Setting	Description
QA Channel Selection	1	Select QA Channel 1.
Application Mode	Readout	Use Readout mode for multiplexed qubit readout. In Readout mode, the output waveform is generated by summing up all readout waveforms in the Waveform Memory slots, thus the instrument is able to readout up to 16 qubits per channel in parallel. In spectroscopy mode, the readout waveform is generated by a digital oscillator. Since there is only 1 digital oscillator per channel, only single qubit readout is possible in Spectroscopy mode.
Integration Delay (s)	224 n	Set the delay time after receiving a trigger before starting integration. This setting ensures that only the expected input signal is integrated. The internal delay from signal generation to integration is about 224 ns. Therefore, an integration delay > 224 ns is necessary if the propagation delay from front panel signal output port to signal input port is not negligible.
Modulation Enable	Enable	Enable modulation so that the oscillator i is used to modulate the envelope waveform saved in the Waveform Memory slot i
Frequency of Oscillator 1 (Hz)	45 MHz	Set frequency of oscillator 0 to 45 MHz. This step is not needed if the oscillator is controlled by the Generator.
Frequency of Oscillator 2 (Hz)	355 MHz	Set frequency of oscillator 1 to 355 MHz. This step is not needed if the oscillator is controlled by the Generator.
Gain of Oscillator 1	0.5	Set gain of oscillator 0 to 0.5.
Gain of Oscillator 2	0.5	Set gain of oscillator 1 to 0.5.

Table 4.52: Readout Pulse Generation Settings of QA Channel 1 on QA Setup Tab.

Parameter	Setting	Description
QA Channel Selection	1	Select QA Channel 1.
Clear Waveform	Click "Clear"	Clear all waveforms saved in Waveform Memory. Clear all waveforms before uploading new ones to avoid incorrect waveform generation or output overflow.
Waveform Memory i Set Mode	Parametric	Generate waveform parametrically saved in Waveform Memory slot i (i is from 1 to 2). The parametrically generated waveform is $\mathbf{A}e^{i(2\pi ft + \frac{\pi}{180}\phi)}$, where \mathbf{A} is the dimensionless amplitude factor of the waveform, f is the frequency in units of Hz, ϕ is the phase in units of degree.
Waveform Memory i Amplitude	0.5	Set amplitude factor \mathbf{A} of the parametrically generated waveform saved in Waveform Memory slot i (i is from 1 to 2) to 0.1. This setting ensures the amplitude factor of sum of all waveforms is ≤ 1 .
Waveform Memory i Frequency (Hz)	0	Set frequency f of the parametrically generated waveform saved in Waveform Memory slot i (i is from 1 to 2) to 0 Hz. The offset frequency is determined by the oscillator frequency.
Waveform Memory i Phase (Deg)	0	Set phase ϕ of the parametrically generated waveform saved in Waveform Memory slot i (i is from 1 to 2) to 0 degree.

Parameter	Setting	Description
Waveform Memory <i>i</i> Window Length	4096	Set length of the parametrically generated waveform saved in Waveform Memory slot <i>i</i> (<i>i</i> is from 1 to 2) in number of samples.
Waveform Memory <i>i</i> Set To Device	click "Set To Device"	Upload the parametrically generated waveform to Waveform Memory slot <i>i</i> (<i>i</i> is from 1 to 2).
Hold Enable in Waveform Memory <i>i</i>	Enable	Enable waveform hold function in Waveform Memory slot <i>i</i> (<i>i</i> is from 1 to 2).
Hold Start Index in Waveform Memory <i>i</i>	4	Set the index of the waveform sample in Waveform Memory Slot <i>i</i> (<i>i</i> is from 1 to 2) where to hold playback to 4.
Hold Length in Waveform Memory <i>i</i>	4096	Set duration in number of samples in Waveform Memory Slot <i>i</i> (<i>i</i> is from 1 to 2) for which to hold the playback to 4096.

Table 4.53: Integration Weights Settings of QA Channel 1 on QA Setup Tab.

Parameter	Setting	Description
QA Channel Selection	1	Select QA Channel 1.
Integration Length	4096	Set the integration length in number of samples.
Downsampling Factor	2	Set downsampling factor to 2 to double integration weight length and integration length.
Clear Weights	Click "Clear"	Clear all weights saved in Integration Weight slots. Clear all weights before uploading new ones to avoid incorrect weight generation.
Integration Weights <i>i</i> Set Mode	Parametric	Generate integration weight parametrically saved in Integration Weights unit <i>i</i> (<i>i</i> is from 1 to 2). The parametrically generated integration weight is $A' e^{-i(2\pi f' t + \frac{\pi}{180} \phi')}$, where A' is the dimensionless amplitude factor of the integration weight, f' is the frequency in units of Hz, ϕ' is the phase in units of degree.
Integration Weights <i>i</i> Amplitude	1	Set amplitude factor A' of the parametrically generated integration weight saved in Integration Weights unit <i>i</i> (<i>i</i> is from 1 to 2)
Integration Weights <i>i</i> Frequency (Hz)	0	Set frequency f' of the parametrically generated waveform saved in Integration Weights unit <i>i</i> (<i>i</i> is from 1 to 2) to 0 Hz. The offset frequency is determined by the oscillator frequency.
Integration Weights <i>i</i> Phase (Deg)	0	Set phase ϕ' of the parametrically generated integration weight saved in Integration Weights unit <i>i</i> (<i>i</i> is from 1 to 2) to 0 degree.
Integration Weights <i>i</i> Window Length	4096	Set length of the integration weight saved in Integration Weights unit <i>i</i> (<i>i</i> is from 1 to 2) in number of samples.
Integration Weights <i>i</i> Set To Device	Click "Set To Device"	Upload the parametrically generated integration weight to Integration Weight unit <i>i</i> (<i>i</i> is from 1 to 2).

How the measurement results are averaged and displayed is defined on QA Result Logger Tab, see [Figure 4.73](#) and [Table 4.54](#).

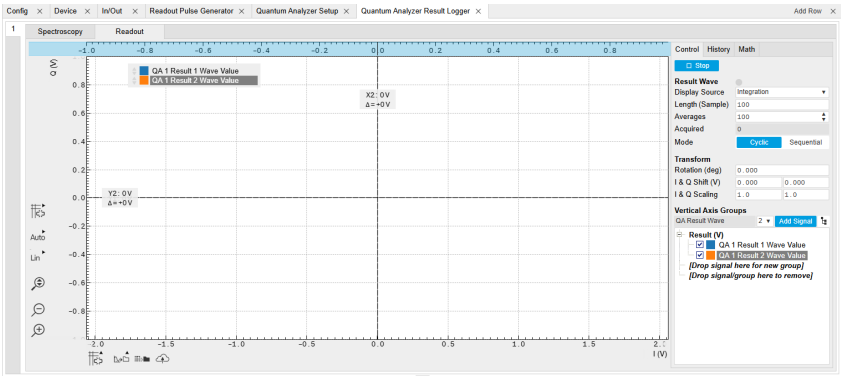


Figure 4.73: Configurations on QA Result Logger Tab.

Table 4.54: Settings of QA Channel 1 on QA Result Logger Tab, see details on [QA Result Logger Tab](#)

Parameter	Setting	Description
QA Channel Selection	1	Select QA Channel 1.
Sub-Tab	Readout	Select Readout sub-tab to monitor measurement result in Readout Mode.
Display Source	Integration	Display result after integration to monitor results in IQ plane. Choose "Integration (I, Q, Amp, Phase)" If it is desired. Choose "Threshold" if qubit state discrimination is desired.
Result Length (Sample)	100	Set result length in number of samples. The number must match what is set in the sequence program.
Averages	100	Set the number of averages. The number must match what is set in the sequence program.
Average Mode	Cyclic	Set the average mode to cyclic. This setting must match how the loop is configured in the sequence program.
Vertical Axis Groups	Add QA 1 Result i Wave Value (i is from 1 to 2)	Add 2 qubit results to the plot.
Run/Stop	Enable	Run the result logger to receive and display measurement results.

2. Run the measurement
 3. Click "Run/Stop" icon on the System Settings sub-tab of DIO tab to run the measurement.
 3. Monitor the measurement results
- The measurement result is displayed on QA Result Tab, as shown in [Figure 4.74](#). The data format of measurement result is complex data. The spread of the readout results indicates that each component of the input signal has a similar amplitude but different phase delay.

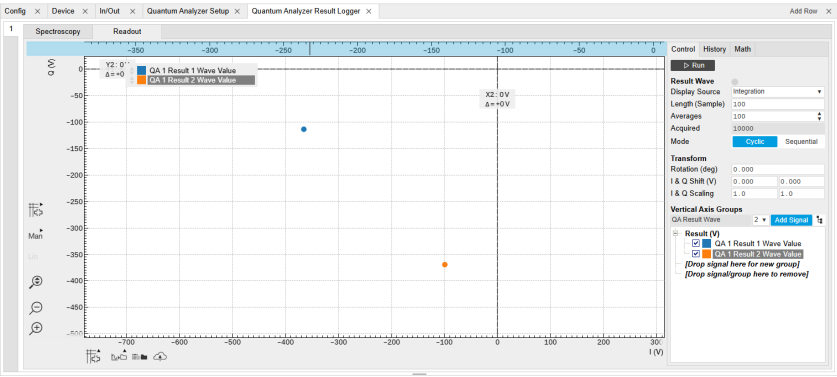


Figure 4.74: Measurement results on QA Result Logger Tab.

zhinst-toolkit

1. Connect the instrument
Create a toolkit session to the data server and connect the device with the device ID, e.g. 'DEV12001', see [Connecting to the Instrument](#).

```
from zhinst.toolkit import Session, SHFQChannelMode, Waveforms
from scipy.signal import gaussian
import numpy as np

DEVICE_ID = 'DEVXXXXX'
SERVER_HOST = 'localhost'

session = Session(SERVER_HOST)          ## connect to data server
device = session.connect_device(DEVICE_ID) ## connect to device
```

2. Generate readout pulses and integration weights

To readout 2 qubits with 4.096 μ s readout time in parallel, the readout waveform is generated by summing up of modulated signal 0 and 1, where the modulated signal 0 (1) is an envelope waveform saved in Waveform Memory slot 0 (1) modulated by the numerical oscillator 0 (1). The length of envelope waveform 0 (1) is doubled by holding the 2048th sample of the envelope with 4096 Samples for length of 4096 Samples long.

In this tutorial, the envelope of all readout pulses is flat-top Gaussian with pulse length of 2.048 μ s and rise and fall time of 2 ns, all amplitude are equally scaled by a factor of 1 and divided by the number of qubits, all phases are set to 0 and the envelopes are modulated by the oscillators with frequency of 45 MHz and 355 MHz. The zhinst-toolkit class `Waveforms` is for converting waveform data written in Python to data that is uploaded to the instrument correctly.

The input signal is demodulated by oscillator 0 and 1, and then integrated with integration weights 0 and 1, respectively. The integration weight length and integration length are doubled by configuring the downsampling factor to 2, so the integration weight is extended by inserting the same sample after each original weight sample, i.e. the original weight data is `a1, a2...a4096`, the extended weight with the downsampling factor of 2 is `a1, a1, a2, a2...a4096, a4096`.

```
# generate readout pulses
NUM_QUBITS = 2
RISE_FALL_TIME = 2e-9 # in units of second
SAMPLING_RATE = 2e9 # in units of Hz
PULSE_DURATION = 2048e-9 # in units of second
FREQUENCIES = [45e6, 355e6] # in units of Hz
PHASE = 0
SCALING = 1 / NUM_QUBITS # amplitude scaling factor

CHANNEL_INDEX = 0 # physical Channel 1

rise_fall_len = int(RISE_FALL_TIME * SAMPLING_RATE)
pulse_len = int(PULSE_DURATION * SAMPLING_RATE)
std_dev = rise_fall_len // 10

gauss = gaussian(2 * rise_fall_len, std_dev)
flat_top_gaussian = np.ones(pulse_len)
flat_top_gaussian[0:rise_fall_len] = gauss[0:rise_fall_len]
flat_top_gaussian[-rise_fall_len:] = gauss[-rise_fall_len:]
flat_top_gaussian *= SCALING
time_vec = np.linspace(0, PULSE_DURATION, pulse_len)

readout_pulses = Waveforms()
for i, f in enumerate(FREQUENCIES):
    readout_pulses.assign_waveform(
        slot=i,
        wave1=flat_top_gaussian * np.exp(1j * PHASE)
    )

# generate integration weights
```

```

ROTATION_ANGLE = 0
weights = Waveforms()
for waveform_slot, pulse in readout_pulses.items():
    weights.assign_waveform(
        slot=waveform_slot,
        wave1=np.conj(pulse[0] * np.exp(1j * ROTATION_ANGLE)) / np.abs(pulse[0])
    )

# upload readout pulses and integration weights to waveform memory
device.qachannels[CHANNEL_INDEX].generator.clearwave() # clear all readout waveforms
device.qachannels[CHANNEL_INDEX].generator.write_to_waveform_memory(readout_pulses)

device.qachannels[CHANNEL_INDEX].readout.integration.clearweight() # clear all integration weights
device.qachannels[CHANNEL_INDEX].readout.write_integration_weights(
    weights=weights,
    # compensation for the delay between generator output and input of the integration unit
    integration_delay=224e-9
)

# define parameters to generate longer readout pulse and integrate longer
device.qachannels[CHANNEL_INDEX].modulation.enable(1)
with device.set_transaction():
    for i in range(NUM_QUBITS):
        device.qachannels[CHANNEL_INDEX].oscs[i].gain(0.5)

device.qachannels[CHANNEL_INDEX].generator.waveforms[i].hold.enable(1) # configure hold parameters after uploading readout waveforms
device.qachannels[CHANNEL_INDEX].generator.waveforms[i].hold.samples.startindex(2048)
device.qachannels[CHANNEL_INDEX].generator.waveforms[i].hold.samples.length(4096)
device.qachannels[CHANNEL_INDEX].readout.integration.downsampling.factor(2)
# integration weight x2, and integration length x2

```

The conjugated readout pulses with the amplitude scaling factor of 1 used as integration weights are uploaded to the integration weight memory for simplicity. Check the Tutorial [Integration Weights Measurement](#) to see how to measure integration weights to improve readout SNR. The integration delay is set to 224 ns in the loopback configuration.

Note

The power consumption of the instrument increases when the "Modulation" is enabled. Please ensure a proper airflow and ambient temperature around the instrument when the modulation is enabled, and disable it if no longer required.

3. Configure the Channel

Configure the Channel such that the readout pulses are integrated with different integration weights in parallel, and the measurement is repeated 10000 times.

```

NUM_READOUTS = 100
NUM_AVERAGES = 100
MODE_AVERAGES = 0 # 0: cyclic; 1: sequential;
INTEGRATION_TIME = PULSE_DURATION # in units of second

with device.set_transaction():
    # configure inputs and outputs
    device.qachannels[CHANNEL_INDEX].configure_channel(
        center_frequency=5e9, # in units of Hz
        input_range=0, # in units of dBm
        output_range=0, # in units of dBm
        mode=SHFQChannelMode.READOUT, # READOUT or SPECTROSCOPY
    )

```

```

    )
    device.qachannels[CHANNEL_INDEX].input.on(1)
    device.qachannels[CHANNEL_INDEX].output.on(1)

    # configure sequencer
    device.qachannels[CHANNEL_INDEX].generator.configure_sequencer_triggering(
        aux_trigger=8, # internal trigger
        play_pulse_delay=0, # 0s delay between startQA trigger and the
        readout pulse
    )

    seqc_program = f"""
        // configure frequency sweep parameters (starting frequency and
        frequency increment) of oscillator 0 (1)
        configFreqSweep(0, {FREQUENCIES[0]}, 1e6);
        configFreqSweep(1, {FREQUENCIES[1]}, 1e6);
        // set oscillator frequency of oscillator 0 (1) to the frequency at
        index of 0
        setSweepStep(0, 0);
        setSweepStep(1, 0);

        // repeat sequence 10000 times
        repeat (100) {{
            repeat (100) {{
                // wait for a trigger over ZSync. Assume the trigger period
                is longer than the cycle time
                // waitZSyncTrigger();

                // alternatively wait for a trigger from digital trigger 1
                waitDigTrigger(1);

                // wait for 4096 Samples between the trigger and the first
                readout pulse
                // Note: this playZero command does not yet block the
                sequencer
                playZero(4096);

                // define how many samples to wait between the two upcoming
                startQA commands
                // Note: this command blocks the sequencer until the
                previous playZero command is finished
                playZero(32);

                // generate readout pulse which is sum of modulated signal
                0 and 1
                // integrate demodulated signal 0 and 1 with integration
                weight 0 and 1, respectively
                startQA(QA_GEN_0|QA_GEN_1, QA_INT_0|QA_INT_1);
            }}
        }}
    """
    device.qachannels[CHANNEL_INDEX].generator.load_sequencer_program(seqc_
    program)

    # configure internal trigger
    device.system.internaltrigger.repetitions(int(NUM_READOUTS * NUM_AVERAG
    ES))
    device.system.internaltrigger.holdoff(100e-6)

    # configure QA setup and QA result logger
    device.qachannels[CHANNEL_INDEX].readout.integration.length(int(INTEGRA

```

```

TION_TIME * SAMPLING_RATE))
    device.qachannels[CHANNEL_INDEX].readout.configure_result_logger(
        result_length=NUM_READOUTS,
        result_source='result_of_integration',
# "result_of_integration" or "result_of_discrimination".
        num_averages=NUM_AVERAGES,
        averaging_mode=MODE_AVERAGES,
    )

```

The input range and output range of the Channel 1 is set to 0 dBm, and the center frequency is 5 GHz. There are 2 application modes see [Quantum Analyzer Setup Tab](#). For multiplexed readout, Readout mode is selected in order to use customized integration weights for different qubits. All settings are configured using `qachannels[n].configure_channel` function. The function `configure_sequencer_triggering` and `load_sequencer_program` are used to configure the sequence trigger and upload the sequence program, respectively. The measurement sequence is defined by the SeqC program such that it sets oscillator frequencies, and sends out the readout pulse and integrates the signal with the integration delay of 224 ns after receiving a trigger (Internal Trigger). The measurement is repeated 10000 times. These parameters are configured by the function `readout.configure_result_logger`.

The result after integration and averaging is saved to the QA Result Logger. This configuration can be used to calibrate control pulses and characterize the qubits, measure thresholds without averaging for state discrimination or readout fidelity if the result source is set to 'result_of_discrimination' and the thresholds are updated using `device.qachannels[CHANNEL_INDEX].readout.discriminators[n].threshold()` (n is the qubit index).

4. Run the measurement, download and plot the result
Before starting the measurement, the QA Result Logger is enabled to be ready to get the result, and the Sequencer is enabled by `enable_sequencer` to be ready to run the sequence once an internal trigger is received. The measurement result is returned by `readout.read` function, and it is also displayed in QA Result Logger Tab see [Figure 4.75](#).

```

device.qachannels[CHANNEL_INDEX].readout.run() # enable QA Result Logger
device.qachannels[CHANNEL_INDEX].generator.enable_sequencer(single=True)
device.system.internaltrigger.enable(1)
readout_results = device.qachannels[CHANNEL_INDEX].readout.read()

```

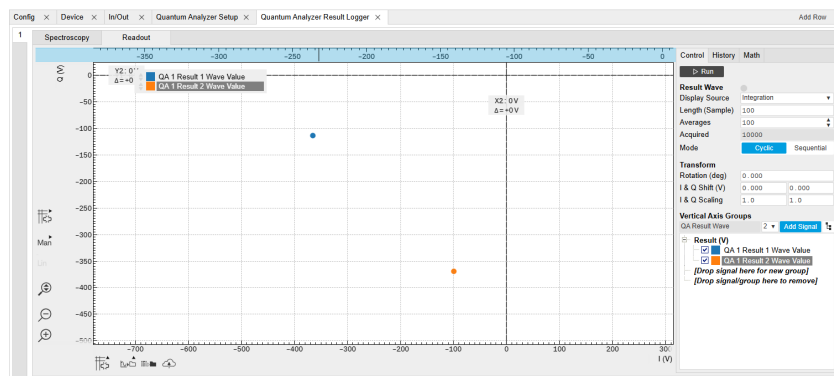


Figure 4.75: Multiplexed readout of 2 qubits in loopback configuration.

Use the following code snippet to plot the readout result as seen in [Figure 4.76](#)

```

import matplotlib.pyplot as plt

plt.figure()
for i in range(NUM_QUBITS):
    plt.plot(readout_results[i].real, readout_results[i].imag, '.', label =
f'Q{i}')
plt.legend()
plt.grid("both")
plt.axis("equal")
plt.xlabel("I (Vrms)")
plt.ylabel("Q (Vrms)")
plt.tight_layout()

```

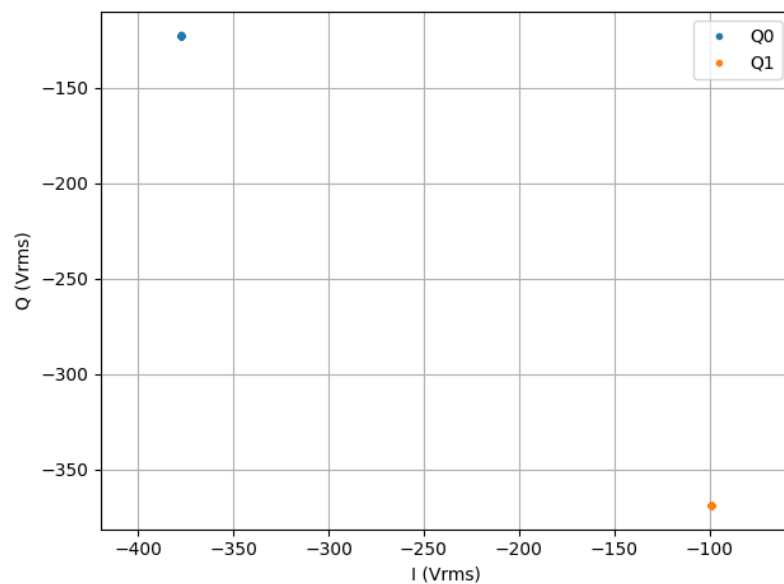


Figure 4.76: Multiplexed readout of 2 qubits in loopback configuration.

5. Functional Description

This chapter gives a detailed description of the [setup](#) and [measurement](#) functionality of the Zurich Instruments SHFQC+. The sections provide details and a complete settings overview of the **setup** configurations - mainly accessible through our LabOne general user interface - and the **measurement** functionality blocks depicted in the [functional diagram](#). The explanations focus on introducing the respective functionalities and how to configure them using either the APIs and/or the LabOne user interface.

5.1. Setup Functionality

This chapter gives a detailed description of the setup functionality available through the LabOne User Interface (UI) that is common to all Zurich Instruments' devices. LabOne provides a Data Server and a Web Server to control the Instrument with any of the most common web browsers (e.g. Firefox, Chrome, Edge, etc.). This platform-independent architecture supports interaction with the Instrument using various devices (PCs, tablets, smartphones, etc.) - even at the same time if needed.

On top of standard functionality like acquiring and saving data points, or session-handling, the SHFQC-specific functionality of the GUI is provided in the [Measurement Functionality](#).

Note

Some of the pictures in the following sections may not show SHFQC-specific nodes, functionality or pictures.

5.1.1. User Interface Overview

UI Nomenclature

This section provides an overview of the LabOne User Interface, its main elements and naming conventions. The LabOne User Interface is a browser-based UI provided as the primary interface to the SHFQC+ instrument. Multiple browser sessions can access the instrument simultaneously and the user can have displays on multiple computer screens. Parallel to the UI, the instrument can be controlled and read out by custom programs written in any of the supported languages (e.g. LabVIEW, MATLAB, Python, C) connecting through the LabOne APIs.

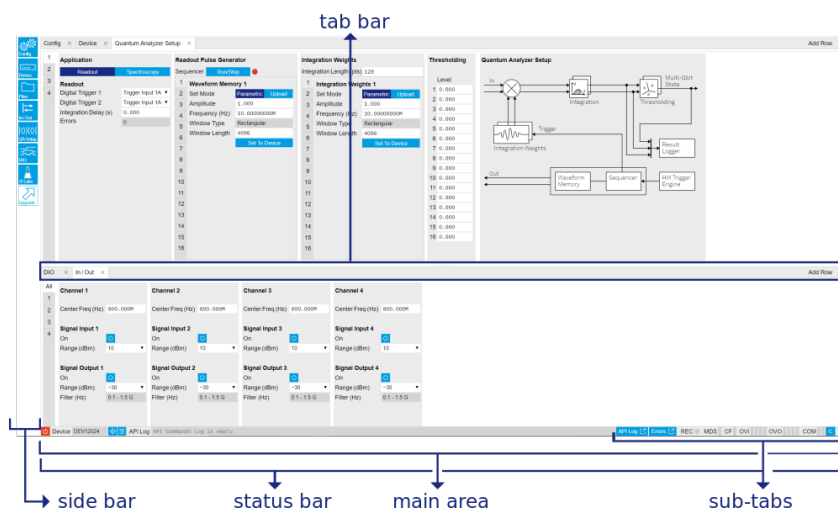


Figure 5.1: LabOne User Interface (default view)

The [LabOne User Interface](#) automatically opens some tabs by default after a new UI session has been started. At start-up, the UI is divided into two tab rows, each containing a tab structure that gives access to the different LabOne tools. Depending on display size and application, tab rows can be freely added and deleted with the control elements on the right-hand side of each tab bar.

Similarly, the individual tabs can be deleted or added by selecting app icons from the side bar on the left. A click on an icon adds the corresponding tab to the display, alternatively the icon can be dragged and dropped into one of the tab rows. Moreover, tabs can be moved by drag-and-drop within a row or across rows.

Table 5.1 gives a brief descriptions and naming conventions for the most important UI items.

Table 5.1: LabOne User Interface features

Item name	Position	Description	Contains
side bar	left-hand side of the UI	contains app icons for each of the available tabs - a click on an icon adds or activates the corresponding tab in the active tab row	app icons
status bar	bottom of the UI	contains important status and warning indicators, device and session information, and access to the command log	status indicators
main area	center of the UI	accommodates all active tabs – new rows can be added and removed by using the control elements in the top right corner of each tab row	tab rows, each consisting of tab bar and the active tab area
tab area	inside of each tab	provides the active part of each tab consisting of settings, controls and measurement tools	sections, plots, sub-tabs, unit selections

Further items are highlighted in Figure 5.2.

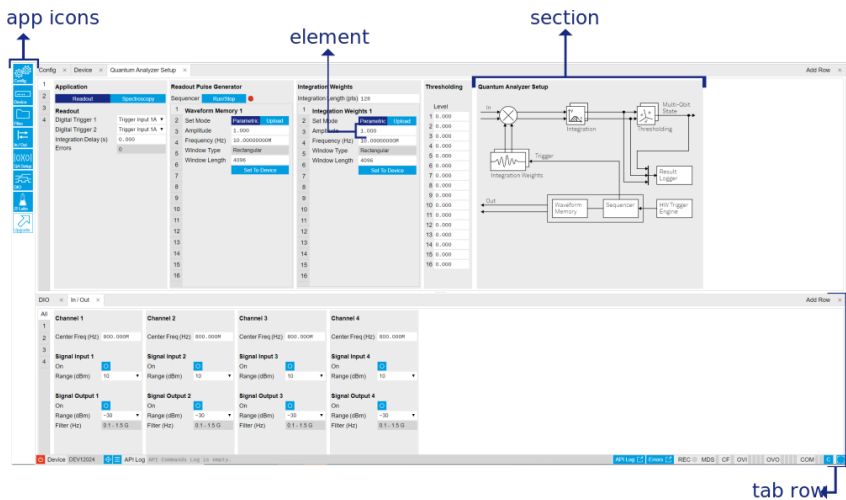







Figure 5.2: LabOne User Interface (more items)

Unique Set of Analysis Tools

All instruments feature a comprehensive tool set for signal generation and sequence programming.

The following table gives the overview of all app icons. Note that the selection of app icons may depend on the upgrade options installed on a given instrument.

Table 5.2: Overview of app icons and short description

Control/ Tool	Option/ Range	Description
Config		Provides access to software configuration.
Device		Provides instrument specific settings.
Files		Access settings and measurement data files on the host computer.
In/Out		Gives access to all controls relevant for the Signal Inputs and Signal Outputs of each channel.
Mod		Access to all the settings of the digital modulation.







Control/Tool	Option/Range	Description
DIO		Gives access to all controls relevant for the digital inputs and outputs including DIO, Trigger Inputs, Trigger Outputs, and Marker Outputs.
AWG		Generate arbitrary signals using sequencing and sample-by-sample definition of waveforms.
ZI Labs		Experimental settings and controls.

Table 5.3 provides a quick overview over the different status bar elements along with a short description.

Table 5.3: Status bar description

Control/Tool	Option/Range	Description
Command log	last command	Shows the last command. A different formatting (MATLAB, Python, ..) can be set in the config tab. The log is also saved in [User] \Documents\Zurich Instruments\LabOne\WebServer\Log
Show Log		Show the command log history in a separate browser window.
Errors	Errors	Display system errors in separate browser tab.
Device	devXXX	Indicates the device serial number.
Identify Device		When active, device LED blinks
Shutdown		Shuts down the instrument.
MDS	grey/green/red/yellow	Multiple device synchronization indicator. Grey: Nothing to synchronize - single device on the UI. Green: All devices on the UI are correctly synchronized. Yellow: MDS sync in progress or only a subset of the connected devices is synchronized. Red: Devices not synchronized or error during MDS sync.
REC	grey/red	A blinking red indicator shows ongoing data recording (related to global recording settings in the Config tab).
RCO	grey/yellow/red	Router Channel Overflow - Red: present overflow condition on the channel. Yellow: indicates an overflow occurred in the past.
CF	grey/yellow/red	Clock Failure - Red: present malfunction of the external 10 MHz reference oscillator. Yellow: indicates a malfunction occurred in the past.
OVI	grey/yellow/red	Signal Input Overload - Red: present overload condition on the signal input also shown by the red front panel LED. Yellow: indicates an overload occurred in the past.
OVO	grey/yellow/red	Overload Signal Output - Red: present overload condition on the signal output. Yellow: indicates an overload occurred in the past.
COM	grey/yellow/red	Packet Loss - Red: present loss of data between the device and the host PC. Yellow: indicates a loss occurred in the past.
COM	grey/yellow/red	Sample Loss - Red: present loss of sample data between the device and the host PC. Yellow: indicates a loss occurred in the past.
Reset status flags		Clear the current state of the status flags
MOD	grey/green	MOD - Green: indicates which of the modulation kits is enabled.
PID	grey/green	PID - Green: indicates which of the PID units is enabled. Red: indicates PID unit is in PLL or ExtRef mode but is not locked. Yellow: indicates PID unit was not locked in the past.
Full Screen		Toggles the browser between full screen and normal mode.

Plot Functionality

Several tools, such as the Waveform Viewer, provide a graphical display of data in the form of plots. These are multi-functional tools with zooming, panning and cursor capability. This section introduces some of the highlights.

Plot Area Elements

Plots consist of the plot area, the X range and the range controls. The X range (above the plot area) indicates which section of the wave is displayed by means of the blue zoom region indicators. The two ranges show the full scale of the plot which does not change when the plot area displays a zoomed view. The two axes of the plot area instead do change when zoom is applied.

The [mouse functionality](#) inside of a plot greatly simplifies and speeds up data viewing and navigation.







Table 5.4: Mouse functionality inside plots




Name	Action	Description	Performed inside
Panning	left click on any location and move around	moves the waveforms	plot area
Zoom X axis	mouse wheel	zooms in and out the X axis	plot area
Zoom Y axis	shift + mouse wheel	zooms in and out the Y axis	plot area
Window zoom	shift and left mouse area select	selects the area of the waveform to be zoomed in	plot area
Absolute jump of zoom area	left mouse click	moves the blue zoom range indicators	X and Y range, but outside of the blue zoom range indicators
Absolute move of zoom area	left mouse drag-and-drop	moves the blue zoom range indicators	X and Y range, inside of the blue range indicators
Full Scale	double click	set X and Y axis to full scale	plot area

Each plot area contains a legend that lists all the shown signals in the respective color. The legend can be moved to any desired position by means of drag-and-drop.

The X range and Y range plot controls are described in [Table 5.5](#).

Table 5.5: Plot control description

Control/Tool	Option/Range	Description
Axis scaling mode		Selects between automatic, full scale and manual axis scaling.
Axis mapping mode		Select between linear, logarithmic and decibel axis mapping.
Axis zoom in		Zooms the respective axis in by a factor of 2.
Axis zoom out		Zooms the respective axis out by a factor of 2.
Rescale axis to data		Rescale the foreground Y axis in the selected zoom area.
Save figure		Generates PNG, JPG or SVG of the plot area or areas for dual plots to the local download folder.

Control/Tool	Option/Range	Description
Save data		Generates a CSV file consisting of the displayed wave or histogram data (when histogram math operation is enabled). Select full scale to save the complete wave. The save data function only saves one shot at a time (the last displayed wave).
Cursor control		Cursors can be switch On/Off and set to be moved both independently or one bound to the other one.
Net Link		Provides a LabOne Net Link to use displayed wave data in tools like Excel, MATLAB, etc.

Cursors and Math









The plot area provides two X and two Y cursors which appear as dashed lines inside of the plot area. The four cursors are selected and moved by means of the blue handles individually by means of drag-and-drop. For each axis, there is a primary cursor indicating its absolute position and a secondary cursor indicating both absolute and relative position to the primary cursor.

Cursors have an absolute position which does not change upon pan or zoom events. In case a cursor position moves out of the plot area, the corresponding handle is displayed at the edge of the plot area. Unless the handle is moved, the cursor keeps the current position. This functionality is very effective to measure large deltas with high precision (as the absolute position of the other cursors does not move).

The cursor data can also be used to define the input data for the mathematical operations performed on plotted data. This functionality is available in the Math sub-tab of each tool. The [Table 5.6](#) gives an overview of all the elements and their functionality. The chosen Signals and Operations are applied to the currently active trace only.

Table 5.6: Plot math description


Control/Tool	Option/Range	Description
Source Select		Select from a list of input sources for math operations.
	Cursor Loc	Cursor coordinates as input data.
	Cursor Area	Consider all data of the active trace inside the rectangle defined by the cursor positions as input for statistical functions (Min, Max, Avg, Std).
	Tracking	Display the value of the active trace at the position of the horizontal axis cursor X1 or X2.
	Plot Area	Consider all data of the active trace currently displayed in the plot as input for statistical functions (Min, Max, Avg, Std).
	Peak	Find positions and levels of up to 5 highest peaks in the data.
	Trough	Find positions and levels of up to 5 lowest troughs in the data.
	Histogram	Display a histogram of the active trace data within the x-axis range. The histogram is used as input to statistical functions (Avg, Std). Because of binning, the statistical functions typically yield different results than those under the selection Plot Area.
	Resonance	Display a curve fitted to a resonance.
	Linear Fit	Display a linear regression curve.
Operation Select		Select from a list of mathematical operations to be performed on the selected source. Choice offered depends on the selected source.
	Cursor Loc: X1, X2, X2-X1, Y1, Y2, Y2-Y1, Y2 / Y1	Cursors positions, their difference and ratio.

Control/Tool	Option/Range	Description
	Cursor Area: Min, Max, Avg, Std	Minimum, maximum value, average, and bias-corrected sample standard deviation for all samples between cursor X1 and X2. All values are shown in the plot as well.
	Tracking: Y(X1), Y(X2), ratioY, deltaY	Trace value at cursor positions X1 and X2, the ratio between these two Y values and their difference.
	Plot Area: Min, Max, Pk Pk, Avg, Std	Minimum, maximum value, difference between min and max, average, and bias-corrected sample standard deviation for all samples in the x axis range.
	Peak: Pos, Level	Position and level of the peak, starting with the highest one. The values are also shown in the plot to identify the peak.
	Histogram: Avg, Std, Bin Size, (Plotter tab only: SNR, Norm Fit, Rice Fit)	A histogram is generated from all samples within the x-axis range. The bin size is given by the resolution of the screen: 1 pixel = 1 bin. From this histogram, the average and bias-corrected sample standard deviation is calculated, essentially assuming all data points in a bin lie in the center of their respective bin. When used in the plotter tab with demodulator or boxcar signals, there additionally are the options of SNR estimation and fitting statistical distributions to the histogram (normal and rice distribution).
	Resonance: Q, BW, Center, Amp, Phase, Fit Error	A curve is fitted to a resonator. The fit boundaries are determined by the two cursors X1 and X2. Depending on the type of trace (Demod R or Demod Phase) either a Lorentzian or an inverse tangent function is fitted to the trace. The Q is the quality factor of the fitted curve. BW is the 3dB bandwidth (FWHM) of the fitted curve. Center is the center frequency. Amp gives the amplitude (Demod R only), whereas Phase returns the phase at the center frequency of the resonance (demod Phase only). The fit error is given by the normalized root-mean-square deviation. It is normalized by the range of the measured data.
	Linear Fit: Intercept, Slope, R ²	A simple linear least squares regression is performed using a QR decomposition routine. The fit boundaries are determined by the two cursors X1 and X2. The parameter outputs are the Y-axis intercept, slope and the R ² -value, which is the coefficient of determination to determine the goodness-of-fit.
Add		Add the selected math function to the result table below.
Add All		Add all operations for the selected signal to the result table below.
Clear Selected		Clear selected lines from the result table above.
Clear All		Clear all lines from the result table above.
Copy		Copy selected row(s) to Clipboard as CSV
Unit Prefix		Adds a suitable prefix to the SI units to allow for better readability and increase of significant digits displayed.
CSV		Values of the current result table are saved as a text file into the download folder.
Net Link		Provides a LabOne Net Link to use the data in tools like Excel, MATLAB, etc.
Help		Opens the LabOne User Interface help.

Note

The standard deviation is calculated using the formula $\sqrt{\frac{1}{N-1} \sum_{i=1}^N (x_i - \bar{x})^2}$ for the unbiased estimator of the sample standard deviation with a total of N samples x_i and an arithmetic average \bar{x} . The formula above is used as-is to calculate the standard deviation for the Histogram Plot Math tool. For large number of points (Cursor Area and Plot Area tools), the more accurate pairwise algorithm is used (Chan et al., "Algorithms for Computing the Sample Variance: Analysis and Recommendations", The American Statistician 37 (1983), 242-247).

Tree Selector

The Tree selector allows one to access streamed measurement data in a hierarchical structure by checking the boxes of the signals that should be displayed. The tree selector also supports data selection from multiple instruments, where available. Depending on the tool, the Tree selector is either displayed in a separate Tree sub-tab, or it is accessible by a click on the  button.

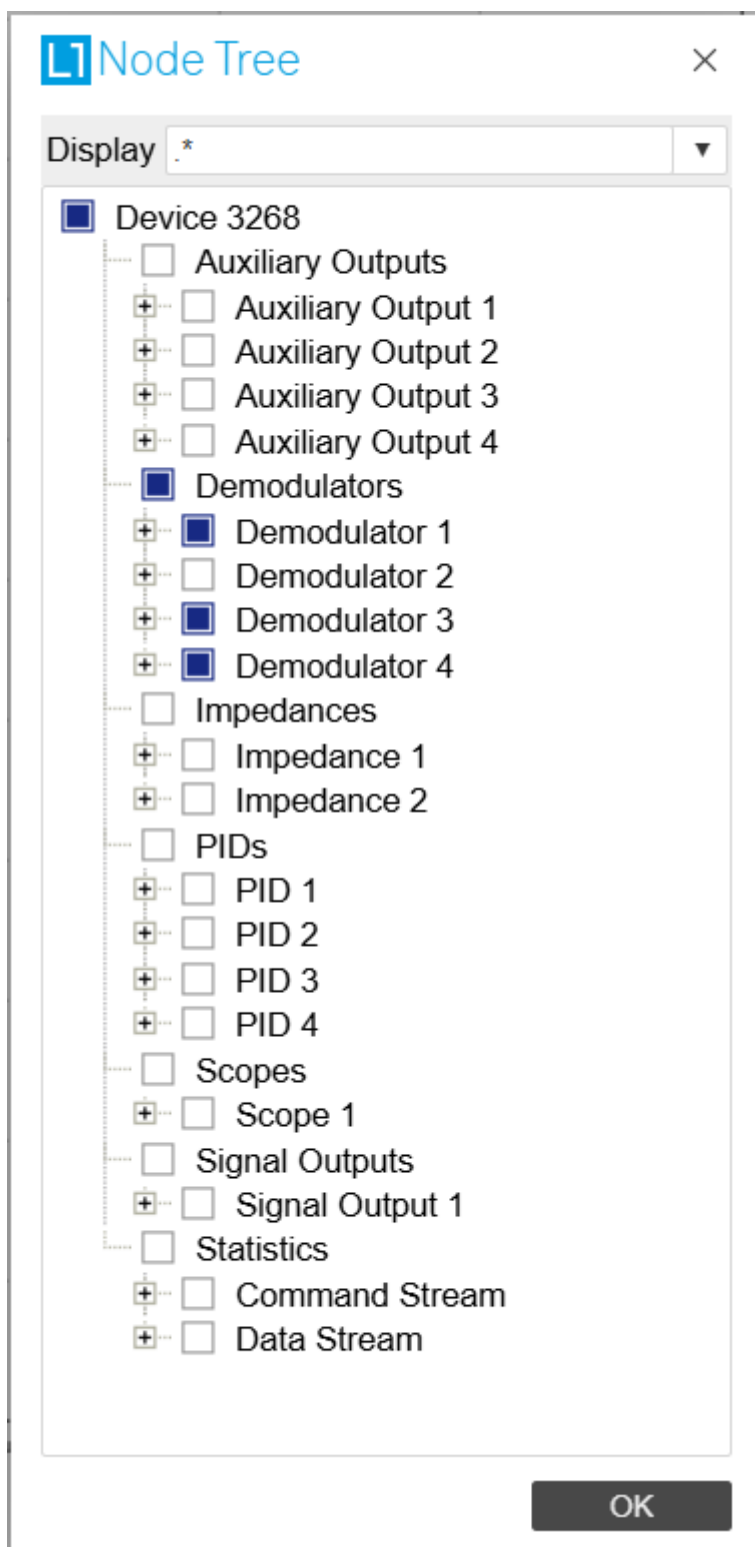


Figure 5.3: Tree selector with Display drop-down menu

Vertical Axis Groups

Vertical Axis groups are available as part of the plot functionality in many of the LabOne tools. Their purpose is to handle signals with different axis properties within the same plot. Signals with different units naturally have independent vertical scales even if they are displayed in the same plot. However, signals with the same unit should preferably share one scaling to enable quantitative comparison. To this end, the signals are assigned to specific axis group. Each axis group has its own axis system. This default behavior can be changed by moving one or more signals into a new group.

The tick labels of only one axis group can be shown at once. This is the foreground axis group. To define the foreground group click on one of the group names in the Vertical Axis Groups box. The current foreground group gets a high contrast color.

Select foreground group

Click on a signal name or group name inside the Vertical Axis Groups. If a group is empty the selection is not performed.

Split the default vertical axis group

Use drag-and-drop to move one signal on the field **[Drop signal here to add a new group]**. This signal will now have its own axis system.

Change vertical axis group of a signal

Use drag-and-drop to move a signal from one group into another group that has the same unit.

Group separation

In case a group hosts multiple signals and the unit of some of these signals changes, the group will be split in several groups according to the different new units.

Remove a signal from the group

In order to remove a signal from a group drag-and-drop the signal to a place outside of the Vertical Axis Groups box.

Remove a vertical axis group

A group is removed as soon as the last signal of a custom group is removed. Default groups will remain active until they are explicitly removed by drag-and-drop. If a new signal is added that match the group properties it will be added again to this default group. This ensures that settings of default groups are not lost, unless explicitly removed.

Rename a vertical axis group

New groups get a default name "Group of ...". This name can be changed by double-clicking on the group name.

Hide/show a signal

Uncheck/check the check box of the signal. This is faster than fetching a signal from a tree again.

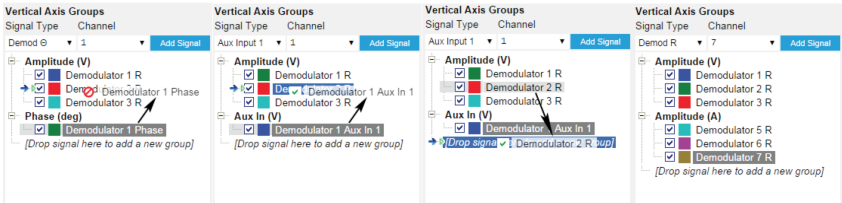



Figure 5.4: Vertical Axis Group typical drag and drop moves.

Demodulator data is only available when using a Zurich Instruments lock-in amplifier from the SHF, UHF, HF, or MF series.

Table 5.7: Vertical Axis Groups description

Control/Tool	Option/Range	Description
Vertical Axis Group		Manages signal groups sharing a common vertical axis. Show or hide signals by changing the check box state. Split a group by dropping signals to the field [Drop signal here to add new group]. Remove signals by dragging them on a free area. Rename group names by editing the group label. Axis tick labels of the selected group are shown in the plot. Cursor elements of the active wave (selected) are added in the cursor math tab.
Signal Type		Select signal types for the Vertical Axis Group.
Channel	integer value	Selects a channel to be added.
Signal	integer value	Selects signal to be added.
Add Signal		Adds a signal to the plot. The signal will be added to its default group. It may be moved by drag and drop to its own group. All signals within a group share a common y-axis. Select a group to bring its axis to the foreground and display its labels.
Window Length	2 s to 12 h	Window memory depth. Values larger than 10 s may cause excessive memory consumption for signals with high sampling rates. Auto scale or pan causes a refresh of the display for which only data within the defined window length are considered.

Trends

The Trends tool lets the user monitor the temporal evolution of signal features such as minimum and maximum values, or mean and standard deviation. This feature is available for the tab. Using the Trends feature, one can monitor all the parameters obtained in the [Math sub-tab](#) of the corresponding tab.

The Trends tool allows the user to analyze recorded data on a different and adjustable time scale much longer than the fast acquisition of measured signals. It saves time by avoiding post-processing of recorded signals and it facilitates fine-tuning of experimental parameters as it extracts and shows the measurement outcome in real time.

To activate the Trends plot, enable the Trends button in the Control sub-tab of the corresponding main tab. Various signal features can be added to the plot from the Trends sub-tab in the [Vertical Axis Groups](#). The vertical axis group of Trends has its own Run/Stop button and Length setting independent from the main plot of the tab. Since the Math quantities are derived from the raw signals in the main plot, the Trends plot is only shown together with the main plot. The Trends feature is only available in the LabOne user interface and not at the API level.

5.1.2. Config Tab

The Config tab provides access to all major LabOne settings and is available on all SHFQC+ instruments.

Features

- define instrument connection parameters
- browser session control
- define UI appearance (grids, theme, etc.)
- store and load instrument settings and UI settings
- configure data recording

Description

The Config tab serves as a control panel for all general LabOne settings and is opened by default on start-up. Whenever the tab is closed or an additional one of the same type is needed, clicking the following icon will open a new instance of the tab.

Table 5.8: App icon and short description

Control/Tool	Option/Range	Description
Config		Provides access to software configuration.

The Config tab (see [LabOne UI: Config tab](#)) is divided into four sections to control connections, sessions, settings, user interface appearance and data recording.

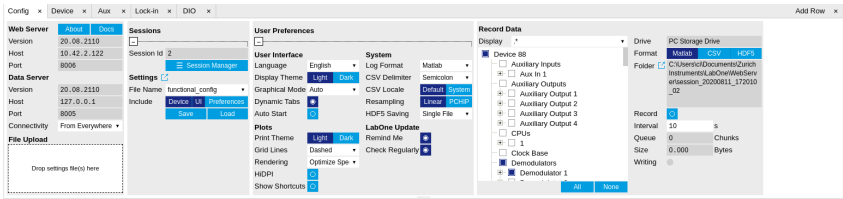


Figure 5.5: LabOne UI: Config tab

The **Connection** section provides information about connection and server versions. Access from remote locations can be restricted with the connectivity setting.

The **Session** section provides the session number which is also displayed in the status bar. Clicking on Session Dialog opens the session dialog window (same as start up screen) that allows one to load different settings files as well as to connect to other instruments.

The **Settings** section allows one to load and save instrument and UI settings. The saved settings are later available in the session dialog.

The **User Preferences** section contains the settings that are continuously stored and automatically reloaded the next time an SHFQC+ instrument is used from the same computer account.

For low ambient light conditions the use of the dark display theme is recommended (see [Figure 5.6](#)).

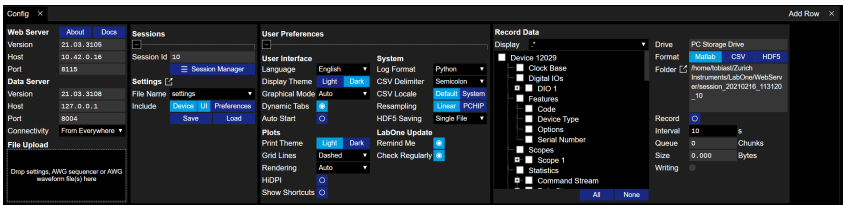





Figure 5.6: LabOne UI: Config tab - dark theme

Functional Elements

Table 5.9: Config tab

Control/Tool	Option/Range	Description
About	About	Get information about LabOne software.
Web Server Version and Revision	string	Web Server version and revision number
Host	default is localhost: 127.0.0.1	IP-Address of the LabOne Web Server
Port	4 digit integer	LabOne Web Server TCP/IP port
Data Server Version and Revision	string	Data Server version and revision number

Control/Tool	Option/Range	Description
Host	default is localhost: 127.0.0.1	IP-Address of the LabOne Data Server
Port	default is 8004	TCP/IP port used to connect to the LabOne Data Server.
Connect/Disconnect		Connect/disconnect the LabOne Data Server of the currently selected device. If a LabOne Data Server is connected only devices that are visible to that specific server are shown in the device list.
Status	grey/green	Indicates whether the LabOne User Interface is connected to the selected LabOne data server. Grey: no connection. Green: connected. Red: error while connecting.
Connectivity	From Everywhere	Forbid/Allow to connect to this Data Server from other computers.
	Localhost Only	
File Upload	drop area	Drag and drop files in this box to upload files. Clicking on the box opens a file dialog for file upload. Supported files: Settings (*.xml).
Session Id	integer number	Session identifier. A session is a connection between a client and LabOne Data Server.
Session Manager		Open the session manager dialog. This allows for device or session change. The current session can be continued by pressing cancel.
File Name	selection of available file names	Save/load the device and user interface settings to/from the selected file on the internal flash drive. The setting files can be downloaded/uploaded using the Files tab.
Include Device		Enable Save/Load of Device settings.
Include UI		Enable Save/Load of User Interface settings.
Include Preferences		Enable loading of User Preferences from settings file.
Save		Save the user interface and device setting to a file.
Load		Load the user interface and device setting from a file.
Display Theme	Dark	Choose theme of the user interface.
	Light	
Plot Print Theme	Dark	Choose theme for printing SVG plots.
	Light	
Plot Grid	None	Select active grid setting for all SVG plots.
	Dashed	
	Solid	
Plot Rendering		Select rendering hint about what tradeoffs to make as the browser renders SVG plots. The setting has impact on rendering speed and plot display for both displayed and saved plots.
	Auto	Indicates that the browser shall make appropriate tradeoffs to balance speed, crisp edges and geometric precision, but with geometric precision given more importance than speed and crisp edges.
	Optimize Speed	The browser shall emphasize rendering speed over geometric precision and crisp edges. This option will sometimes cause the browser to turn off shape anti-aliasing.

Control/Tool	Option/Range	Description
	Crisp Edges	Indicates that the browser shall attempt to emphasize the contrast between clean edges of artwork over rendering speed and geometric precision. To achieve crisp edges, the user agent might turn off anti-aliasing for all lines and curves or possibly just for straight lines which are close to vertical or horizontal.
	Geometric Precision	Indicates that the browser shall emphasize geometric precision over speed and crisp edges.
Resampling Method		Select the resampling interpolation method. Resampling corrects for sample misalignment in subsequent scope shots. This is important when using reduced sample rates with a time resolution below that of the trigger.
	Linear	Linear interpolation
	PCHIP	Piecewise Cubic Hermite Interpolating Polynomial
Show Shortcuts	ON / OFF	Displays a list of keyboard and mouse wheel shortcuts for manipulating plots.
Dynamic Tabs	ON / OFF	If enabled, sections inside the application tabs are collapsed automatically depending on the window width.
Graphical Mode	Collapsed	Select the display mode for the graphical elements. Auto format will select the format which fits best the current window width.
	Auto	
	Expanded	
Log Format	.NET	Choose the command log format. See status bar and [User] \Documents\Zurich Instruments\LabOne\WebServer\Log
	MATLAB	
	Python	
CSV Delimiter	Tab	Select which delimiter to insert for CSV files.
	Comma	
	Semicolon	
CSV Locale	System locale. Use the symbols set in the language and region settings of the computer	Select the locale used for defining the decimal point and digit grouping symbols in numeric values in CSV files. The default locale uses dot for the decimal point and no digit grouping, e.g. 1005.07. The system locale uses the symbols set in the language and region settings of the computer.
	Default locale. Dot for the decimal point and no digit grouping, e.g. 1005.07	
HDF5 Saving	Multiple files. Each measurement goes in a separate file	For HDF5 file format only: Select whether each measurement should be stored in a separate file, or whether all measurements should be saved in a single file.
	Single file. All measurements go in one file	
Auto Start	ON / OFF	Skip session manager dialog at start-up if selected device is available. In case of an error or disconnected device the session manager will be reactivated.
Update Reminder	ON / OFF	Display a reminder on start-up if the LabOne software wasn't updated in 180 days.
Update Check	ON / OFF	Periodically check for new LabOne software over the internet.
Drive		Select the drive for data saving.
Format	HDF5	File format of recorded and saved data.

Control/Tool	Option/Range	Description
	MATLAB	
	CSV	
Open Folder		Open recorded data in the system File Explorer.
Folder	path indicating file location	Folder containing the recorded data.
Save Interval	Time in seconds	Time between saves to disk. A shorter interval means less system memory consumption, but for certain file formats (e.g. MATLAB) many small files on disk. A longer interval means more system memory consumption, but for certain file formats (e.g. MATLAB) fewer, larger files on disk.
Queue	integer number	Number of data chunks not yet written to disk.
Size	integer number	Accumulated size of saved data in the current session.
Record	ON / OFF	Start and stop saving data to disk as defined in the selection filter. Length of the files is determined by the Window Length setting in the Plotter tab.
Writing	grey/green	Indicates whether data is currently written to disk.
Display	filter or regular expression	Display specific tree branches using one of the preset view filters or a custom regular expression.
Tree	ON / OFF	Click on a tree node to activate it.
All		Select all tree elements.
None		Deselect all tree elements.

5.1.3. Device Tab

The Device tab is the main settings tab for the connected instrument and is available on all SHFQC+ instruments.


Features

- Option and upgrade management
- External clock referencing (10/100 MHz)
- Instrument connectivity parameters
- Device monitor

Description

The **Device tab** serves mainly as a control panel for all settings specific to the instrument that is controlled by LabOne in this particular session. Whenever the tab is closed or an additional one of the same type is needed, clicking the following icon will open a new instance of the tab.

Table 5.10: App icon and short description

Control/Tool	Option/Range	Description
Device		Provides instrument specific settings.

The [Device tab](#) is divided into five sections: general instrument information, configuration, communication parameters, statistics, and a device monitor.

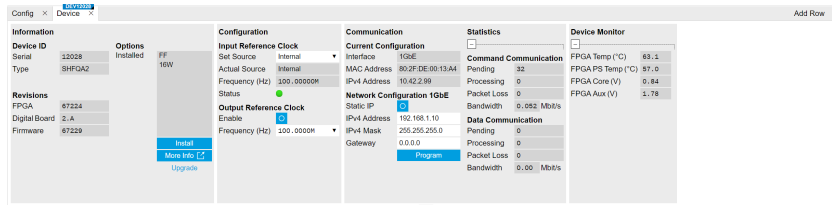


Figure 5.7: LabOne UI: Device tab

The **Information** section provides details about the instrument hardware and indicates the installed upgrade options. This is also the place where new options can be added by entering the provided option key.

The **Configuration** section allows one to change the reference from the internal clock to an external 10 / 100 MHz reference. The reference is to be connected to the Clock Input on the instrument back panel. The section also allows one to select a frequency of 10 or 100 MHz of the reference clock output, which is generated at the Clock Output on the instrument back panel.

The **Communication** section offers access to the instruments TCP/IP settings.

The **Statistics** section gives an overview on communication statistics.

Note

Packet loss on data streaming over UDP, TCP or USB: data packets may be lost if total bandwidth exceeds the available physical interface bandwidth. Data may also be lost if the host computer is not able to handle high-bandwidth data.


Packet loss on command streaming over TCP or USB: command packets should never be lost as it creates an invalid state.

The **Device Monitor** section is collapsed by default and generally only needed for servicing. It displays vitality signals of some of the instrument's hardware components.

Functional Elements

Table 5.11: Device tab

Control/Tool	Option/Range	Description
Serial	1-4 digit number	Device serial number
Type	string	Device type
FPGA	integer number	HDL firmware revision.
Digital Board	version number	Hardware revision of the FPGA base board.
Firmware	integer number	Revision of the device internal controller software.
Installed Options	short names for each option	Options that are installed on this device.
Install	Install	Click to install options on this device. Requires a unique feature code and a power cycle after entry.
More Information		Display additional device information in a separate browser tab.
Upgrade Device Options		Display available upgrade options.
Input Reference Clock Source		Selects Internal, External or the ZSync clock source as reference. Instruments will be disconnected from ZSync if clock source is changed to Internal or External.
	Internal	The internal 100MHz clock is used as the frequency and time base reference.

Control/Tool	Option/Range	Description
	External	An external clock is intended to be used as the frequency and time base reference. Provide a clean and stable 10MHz or 100MHz reference to the appropriate back panel connector.
	ZSync	A ZSync clock is intended to be used as the frequency and time base reference.
Actual Input Reference Clock Source		Currently active clock source. This might differ from the Set Source choice if the set clock is not available.
	Internal	Internal 100MHz clock is actually used as the frequency and time base reference.
	External	An external clock is actually used as the frequency and time base reference.
	ZSync	ZSync clock is actually used as the frequency and time base reference.
Input Reference Clock Frequency		Indicates the frequency of the input reference clock.
Input Reference Clock Status		Indicates the status of the input reference clock. Green: locked. Yellow: the device is busy trying to lock onto the input reference clock signal. Red: there was an error locking onto the input reference clock signal. The instrument is currently not operational.
Output Reference Clock Enable		Enable clock signal on the reference clock output.
Output Reference Clock Frequency		Selects the frequency of the output reference clock to be 10MHz or 100MHz.
Synchronization Source		Selects the source for synchronization of channels: internal (default) or external
	Internal	Synchronization of all channels of a device that have the corresponding synchronization setting enabled.
	External	Same as internal plus synchronization to other devices via ZSync.
Load Factory Default		Load the factory default settings.
Busy	grey/red	Indicates that the device is busy with either loading, saving or erasing a preset.
Error		Returns a 0 if the last preset operation was successfully completed or 1 if the last preset operation was illegal.
	0	Last preset operation was successfully completed.
	1	Last preset operation was illegal.
Error LED	grey/red	Turns red if the last operation was illegal.
Interface		Active interface between device and data server. In case multiple options are available, the priority as indicated on the left applies.
MAC Address	80:2F:DE:xx:xx:xx	MAC address of the device. The MAC address is defined statically, cannot be changed and is unique for each device.
IPv4 Address	default 192.168.1.10	Current IP address of the device. This IP address is assigned dynamically by a DHCP server, defined statically, or is a fall-back IP address if the DHCP server could not be found (for point to point connections).
Static IP	ON / OFF	Enable this flag if the device is used in a network with fixed IP assignment without a DHCP server.
IPv4 Address	default 192.168.1.10	Static IP address to be written to the device.
IPv4 Mask	default 255.255.255.0	Static IP mask to be written to the device.

Control/Tool	Option/Range	Description
Gateway	default 192.168.1.1	Static IP gateway
Save	Program	Click to save the specified IPv4 address, IPv4 Mask and Gateway to the device. Otherwise, the settings will be lost after power cycling the device.

5.1.4. File Manager Tab


Features

- Download measurement data, instruments settings and log files to a local device
- Manage file structure (browse, copy, rename, delete) on instrument flash drive and attached USB mass storage devices
- Update instrument from USB mass storage
- Quick access to measurement files, log files and settings files
- File preview for settings files and log files

Description

The File Manager tab provides standard tools to see and organize the files relevant for the use of the instrument. Files can be conveniently copied, renamed and deleted. Whenever the tab is closed or an additional one of the same type is needed, clicking the following icon will open a new instance of the tab.

Table 5.12: App icon and short description

Control/Tool	Option/Range	Description
Files		Access settings and measurement data files on the host computer.

The Files tab (see [LabOne UI: File Manager tab](#)) provides three windows for exploring. The left window allows one to browse through the directory structure, the center window shows the files of the folder selected in the left window, and the right window displays the content of the file selected in the center window, e.g. a settings file or log file.

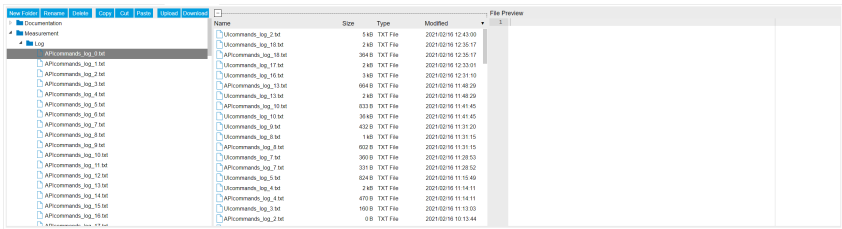


Figure 5.8: LabOne UI: File Manager tab

Functional Elements

Table 5.13: File tab

Control/Tool	Option/Range	Description
New Folder	New Folder	Create new folder at current location.
Rename	Rename	Rename selected file or folder.
Delete	Delete	Delete selected file(s) and/or folder(s).
Copy	Copy	Copy selected file(s) and/or folder(s) to Clipboard.
Cut	Cut	Cut selected file(s) and/or folder(s) to Clipboard.

Control/Tool	Option/Range	Description
Paste	Paste	Paste file(s) and/or folder(s) from Clipboard to the selected directory.
Upload	Upload	Upload file(s) and/or folder(s) to the selected directory.
Download	Download	Download selected file(s) and/or folder(s).

5.1.5. Saving and Loading Data



Overview

In this section we discuss how to save and record measurement data with the SHFQC+ Instrument using the LabOne user interface. In the LabOne user interface, there are 3 ways to save data:

- Saving the data that is currently displayed in a plot
- Continuously recording data in the background
- Saving trace data in the History sub-tab

Furthermore, the History sub-tab supports loading data. In the following, we will explain these methods.


Saving Data from Plots

A quick way to save data from any plot is to click on the [Save CSV](#) icon  at the bottom of the plot to store the currently displayed curves as a comma-separated value (CSV) file to the download folder of your web browser. Clicking on  will save a graphics file instead.

Recording Data

The recording functionality allows you to store measurement data continuously, as well as to track instrument settings over time. The [Config Tab](#) gives you access to the main settings for this function. The Format selector defines which format is used: HDF5, CSV, or MATLAB. The CSV delimiter character can be changed in the User Preferences section. The default option is Semicolon.

The node tree display of the Record Data section allows you to browse through the different measurement data and instrument settings, and to select the ones you would like to record. For instance, the demodulator 1 measurement data is accessible under the path of the form **Device 0000/Demodulators/Demod 1/Sample**. An example for an instrument setting would be the filter time constant, accessible under the path **Device 0000/Demodulators/Demod 1/Filter Time Constant**.

The default storage location is the LabOne Data folder which can, for instance, be accessed by the Open Folder button . The exact path is displayed in the Folder field whenever a file has been written.

Clicking on the Record checkbox will initiate the recording to the hard drive.

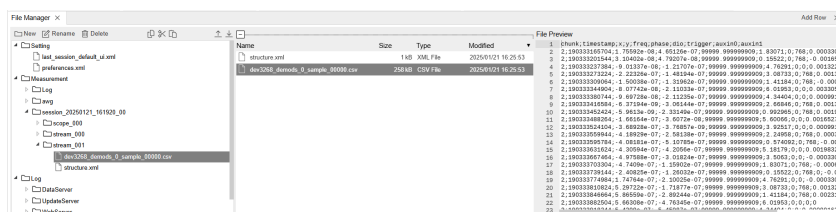


Figure 5.9: Browsing and inspecting files in the LabOne File Manager tab



In case HDF5 or MATLAB is selected as the file format, LabOne creates a single file containing the data for all selected nodes. For the CSV format, at least one file for each of the selected nodes is created from the start. At a configurable time interval, new data files are created, but the maximum size is capped at about 1 GB for easier data handling. The storage location is indicated in the Folder field of the Record Data section.

The [File Manager Tab](#) is a good place to inspect CSV data files. The file browser on the left of the tab allows you to navigate to the location of the data files and offers functionalities for managing files in the LabOne Data folder structure. In addition, you can conveniently transfer files between the folder structure and your preferred location using the Upload/Download buttons. The file viewer on the right side of the tab displays the contents of text files up to a certain size limit. [Figure 5.9](#) shows the Files tab after recording Demodulator Sample and Filter Time Constant for a few seconds. The file viewer shows the contents of the demodulator data file.

Note

The structure of files containing instrument settings and of those containing streamed data is the same. Streaming data files contain one line per sampling period, whereas in the case of instrument settings, the file usually only contains a few lines, one for each change in the settings. More information on the file structure can be found in the LabOne Programming Manual.

History List

Tabs with a history list such as [Scope Tab](#), support feature saving, autosaving, and loading functionality. By default, the plot area in those tools displays the last 100 measurements (depending on the tool, these can be sweep traces, scope shots, DAQ data sets, or spectra), and each measurement is represented as an entry in the History sub-tab. The button to the left of each list entry controls the visibility of the corresponding trace in the plot; the button to the right controls the color of the trace. [^1]Double-clicking on a list entry allows you to rename it. All measurements in the history list can be saved with [Save All](#). Clicking on the [Save Sel](#) button (note the dropdown button ) saves only those traces that were selected by a mouse click. Use the Control or Shift button together with a mouse click to select multiple traces. The file location can be accessed by the Open Folder button . [Figure 5.12.8](#) illustrates some of these features. [Figure 5.10](#) illustrates the data loading feature.

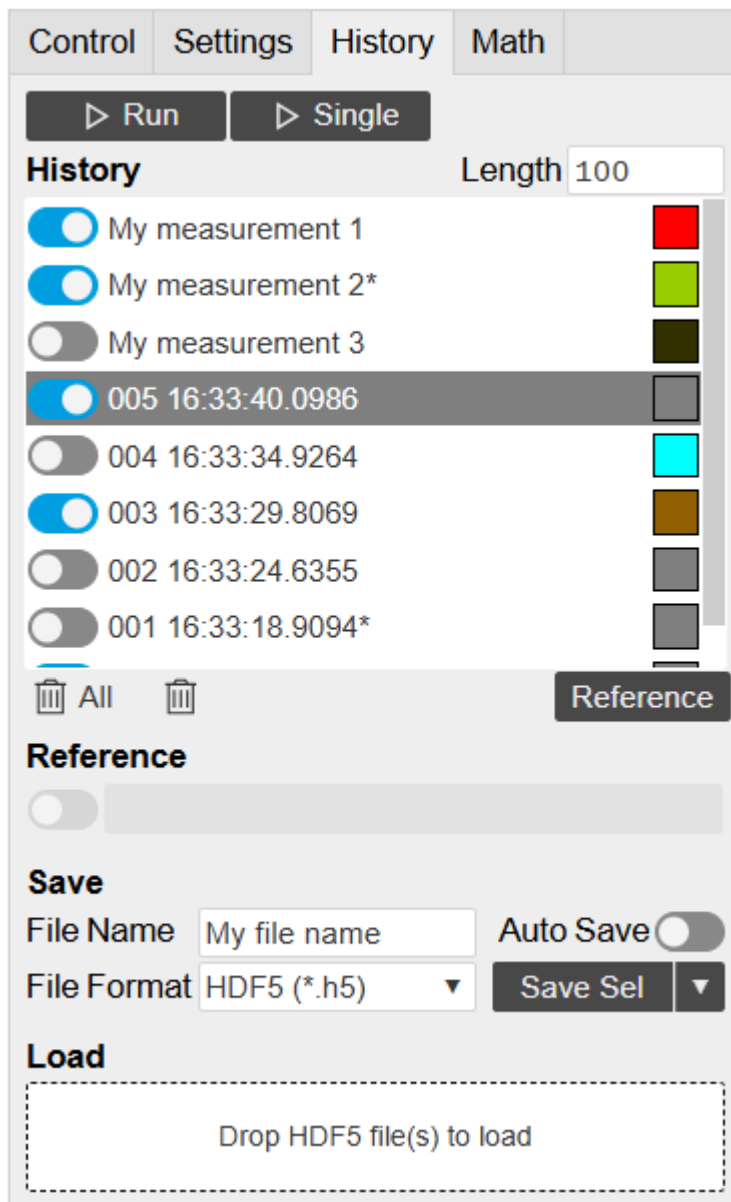


Figure 5.10: History sub-tab features. The entries "My measurement 1" etc. were renamed by the user. Measurement 1, 2, 3, 4 are currently displayed in the plot because their left-hand-side button is enabled. Clicking on Save Sel would save "My measurement 3" and "My measurement 4" to a file, because these entries were selected (gray overlay) by a Control key + mouse click action.

Which quantities are saved depends on which signals have been added to the Vertical Axis Groups section in the **Control** sub-tab. Only data from demodulators with enabled Data Transfer in the Lock-in tab can be included in the files.

The history sub-tab supports an **autosave** functionality to store measurement results continuously while the tool is running. Autosave directories are differentiated from normal saved directories by the text "autosave" in the name, e.g. sweep_autosave_000. When running a tool continuously (**Run/Stop** button) with Autosave activated, after the current measurement (history entry) is complete, all measurements in the history are saved. The same file is overwritten each time, which means that old measurements will be lost once the limit defined by the history Length setting has been reached. When performing single measurements (**Single** button) with Autosave activated, after each measurement, the elements in the history list are saved in a new directory with an incrementing count, e.g. sweep_autosave_001, sweep_autosave_002.

Data which was saved in HDF5 file format can be loaded back into the history list. Loaded traces are marked by a prefix "loaded " that is added to the history entry name in the user interface. The **createdtimestamp** information in the header data marks the time at which the data were measured.

- Only files created by the Save button in the History sub-tab can be loaded.
- Loading a file will add all history items saved in the file to the history list. Previous entries are kept in the list.

- Data from the file is only displayed in the plot if it matches the current settings in the Vertical Axis Group section the tool. Loading e.g. PID data in the Sweeper will not be shown, unless it is selected in the Control sub-tab.
- Files can only be loaded if the devices saving and loading data are of the same product family. The data path will be set according to the device ID loading the data.

Figure 5.11 illustrates the data loading feature.

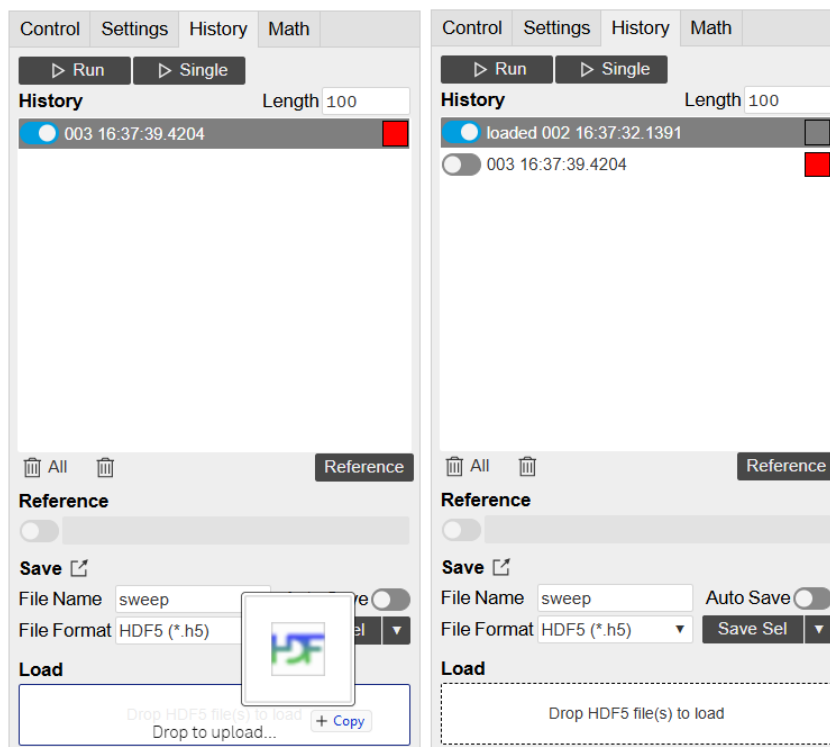


Figure 5.11: History data loading feature. Here, the file sweep_00000.h5 is loaded by drag-and-drop. The loaded data are added to the measurements in the history list.

Supported File Formats

HDF5

Hierarchical Data File 5 (HDF5) is a widespread memory-efficient, structured, binary, open file format. Data in this format can be inspected using the dedicated viewer [HDFview](#). HDF5 libraries or import tools are available for Python, MATLAB, LabVIEW, C, R, Octave, Origin, Igor Pro, and others. The following example illustrates how to access demodulator data from a sweep using the h5py library in Python:

```
import h5py
filename = 'sweep_00000.h5'
f = h5py.File(filename, 'r')
x = f['000/dev3025/demods/0/sample/frequency']
```

The data loading feature of LabOne supports HDF5 files, while it is unavailable for other formats.

MATLAB

The MATLAB File Format (.mat) is a proprietary file format from MathWorks based on the open HDF5 file format. It has thus similar properties as the HDF5 format, but the support for importing .mat files into third-party software other than MATLAB is usually less good than that for importing HDF5 files.

SXM

SXM is a proprietary file format by Nanonis used for SPM measurements.

5.1.6. Upgrade Tab

The Upgrade tab serves as a source of information about the possible upgrade options for the instrument in use. The tab has no functional purpose but provides the user with a quick link to further information about the upgrade options online.

5.1.7. ZI Labs Tab

The ZI Labs tab contains experimental LabOne functionalities added by the ZI development team. The settings found here are often relevant to special applications, but have not yet found their definitive place in one of the other LabOne tabs. Naturally this tab is subject to frequent changes, and the documentation of the individual features would go beyond the scope of this user manual. Clicking the following icon will open a new instance of the tab.

Table 5.14: App Icon and short description

Control/Tool	Option/Range	Description
ZI Labs		Experimental settings and controls.

5.2. Measurement Functionality

In this section, the measurement functionality of the SHFQC+ is described, i.e. the functionality that is useful when setting up and carrying out experiments. Each chapter first introduces the functionality, provides a summary of the functional elements before - if applicable - explaining the representation in the LabOne general user interface.

The instrument functionality is a combination of the functionalities of the SHFQA+ and SHFSG+ and is thus identical in most aspects to what is described in the [SHFSG+](#) and [SHFQA+](#) user manuals. The functionality is represented by a [node tree](#). Each node can either set, read or poll settings or data from the device. This can be done either through the General User Interface, or through our APIs. Most of the functionality resides within the different channels of the SHFQC+, each of which is represented by its own version of the QChannels (`/dev....qachannels/0/...`), or SGChannels (`/dev....sgchannels/n/...`) branch. Functionality that is either independent of the output Channels or shared between them has its own branch, e.g. common device features (`/dev....features/...`), or system features (`/dev....systems/n/...`). All nodes are listed within the [node tree documentation](#).

Note

The following chapters are constantly being upgraded and new documentation is added. For the latest version of the documentation, please always refer to the [online documentation](#).

5.2.1. In/Out Tab

The In / Out tab provides access to the settings of the Instrument's Signal Input and Signal Output of the Quantum Analyzer Channel, as well as the Instrument's Signal Outputs of the Signal Generator Channels. It is available on all SHFQC+ Instruments.

Features Overview

- Enable/disable inputs and outputs
- Define the Center Frequency of the modulation and analysis bands
- Define the input and output power ranges
- Switch between Radio Frequency (RF) and Low Frequency (LF) paths on the Signal Generator Channels

Description

Table 5.15: App icon and short description

Control/ Tool	Option/ Range	Description
Output		Quick overview and access to all the settings for configuring the analog upconversion path.

The SHFQC+ uses the double super-heterodyne frequency upconversion technique to generate its RF output frequencies. The SHFQC+ has two types of channels: One Quantum Analyzer Channel for performing qubit readout measurements, and 2, 4, or 6 Signal Generator Channels for generating signals to control the qubit states.

Note

It is highly recommended to enable all required inputs and outputs and wait for 2 hours after powering on the instrument.

Note

Please wait for at least 0.5 seconds after switching the center frequency or power range before running a measurement.

Note

Please do not change the center frequency or input range while acquiring data. Mishandling of this could lead to an invalid scaling of the result vector.

Signal Generator Channels

Each Signal Generator Channel has its own frequency upconversion chain. Each Signal Generator Channel has two available Output paths: the RF path for generating signals with center frequencies from 0.6 GHz to 8 GHz, and the LF path for generating signals with center frequencies from 0 GHz to 2 GHz. When using the RF path, center frequencies determine the frequency of an analog synthesizer and can be set with a resolution of 0.2 GHz. All variants of the SHFQC+ contain 4 synthesizers. The Quantum Analyzer Channel uses one synthesizer, and each pair of Signal Generator Channels share one synthesizer. This means that Signal Generator Channels 1 and 2 must share the same RF center frequency when using the RF path. To achieve different output frequencies on Signal Generator Channels 1 and 2, digital modulation must be employed (see the [Modulation Tab](#)). When using the LF path, the center frequencies of each channel must be a multiple of 0.1 GHz can be set independently of the other channels in all variants of the SHFQC+ Instrument.

Note

The LF and RF paths can be programmed with the same sequences (see the tutorial [Basic Waveform Playback](#)) but the LF path has a smaller latency than the RF path due to the differences in the analog part of the signal path. The differences in latencies can be compensated by appropriate use of the `playZero` command, described in the Tutorials.

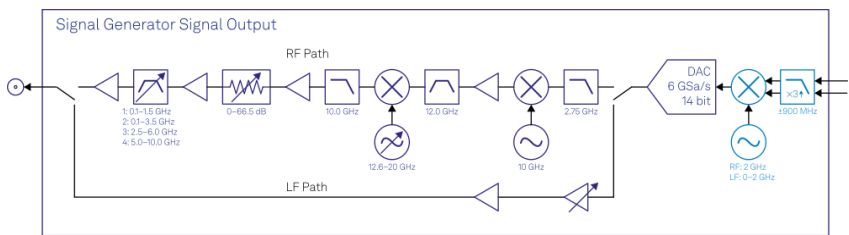


Figure 5.12: Analog Signal Output Stage

When using the [Signal Output](#) of the RF path, the digital 1-GHz-wide modulation band centered around DC is first interpolated by a factor of 3, then digitally upconverted to 2 GHz (light blue elements) before it is passed to the 14-bit DAC. The resulting 2 GHz analog signal (dark blue elements) is then converted to 12 GHz by means of a local oscillator at 10 GHz. To remove all unwanted spurious signals, the signal is strongly filtered before it is down-converted in a second mixing process with a variable local oscillator. Depending on its software-controllable frequency value, the final output frequency band has a center frequency between 0.6-8 GHz and a width of ± 0.5 GHz. Several amplifiers, attenuators, and filters in the up-conversion chain ensure that the different elements are not saturated and that the DAC range is faithfully mapped to the selected [Output Range](#).

When using the LF path, the digital 1-GHz-wide modulation signal is still interpolated by a factor of 3 and passed to the 14-bit DAC, but the analog upconversion chain is bypassed. The center frequency is determined by setting the frequency of the oscillator used in the digital upconversion (fixed at 2 GHz when using the RF path, and can be set to a multiple of 100 MHz in the range 0 - 2 GHz when using the LF path). In this way, signals with center frequencies between 0 and 2 GHz can be generated with the LF path.

The advantages of this up-conversion scheme compared to IQ-mixer-based frequency conversion are that it is calibration-free, wide-band, and stable, in addition to having superior spurious tone performance. The optimal selection of the different gains, attenuators, and filters in the frequency conversion chains are taken over by the SHFQC+, such that only a few settings need to be set in the Output band parameters of the SHFQC+: Center Frequency, Output Range, and Output On.

Note

For both the LF and RF paths, the output power can be set in steps of 5 dBm, in the range -30 dBm to +10 dBm for the RF path and -30 dBm to +5 dBm for the LF path. If the power is set to a value that is outside this range or not a multiple of 5 dBm, the value will automatically be rounded to the nearest multiple of 5 dBm within the range for the path.

Quantum Analyzer Channel

The Quantum Analyzer Channel of the SHFQC+ uses a similar up-conversion chain as for the [Signal Generator](#) Channel. For the [Signal Output](#), the digital 1-GHz-wide analysis band centered around DC is first interpolated by a factor of 3, then digitally upconverted to 2 GHz (light blue elements) before it is passed to the 14-bit DAC. Then, the analog signal (dark blue elements) is mixed to 12 GHz by means of a local oscillator at 10 GHz. To remove all unwanted spurious signals, the signal is strongly filtered before it is down-converted in a second mixing process with a variable local oscillator. Depending on its software-controllable frequency value, the final output frequency band has a center frequency between 1-8 GHz and a width of ± 0.5 GHz. Several amplifiers, attenuators and filters in the up-conversion chain ensure that the different elements are not saturated and that the ADC range is faithfully mapped to the selected Output Range.

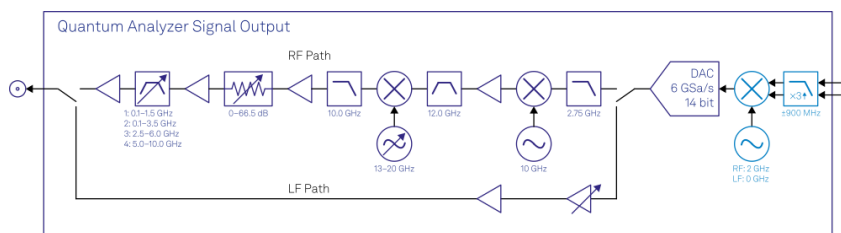


Figure 5.13: Analog signal output stage

The [Signal Input](#) down-conversion chain (dark blue elements) works analogously, but in the backwards direction. The main differences to the Signal Output are the missing selectable filter and the conversion to 3 GHz instead of 2 GHz before the digitization through the 14-bit ADC. Because of the sampling rate of 2 GSa/s, the 3 GHz signal appears as a 1-GHz signal in the digital domain (light blue elements) before being down-converted to DC.

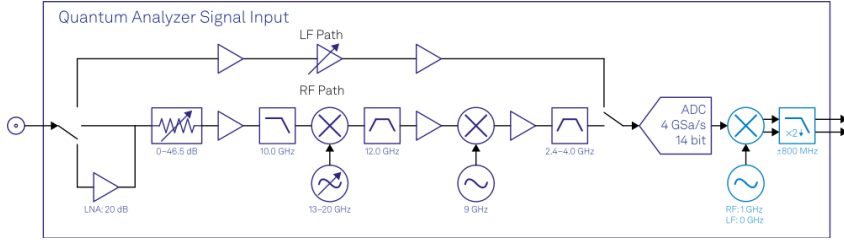


Figure 5.14: Analog signal input stage

Similar to the SG Channels, the QA channel has both an LF and RF path. On the QA Channel, the frequency range of the LF path is from DC - 800 MHz. The LF path can be enabled for the QA Input, for the QA Output, or for both. If the LF path is enabled for both QA Input and QA Output, the frequency of the oscillator before the DAC is set to 0 Hz, then the signal after the DAC is sent out through the output "LF Path", and the input signal through the input "LF path" is converted by the ADC and then mixes with the oscillator at 0 Hz before integration.

Frequency Representation

The frequency f_{out} of the output signal on each channel can be calculated as

$$f_{\text{out}} = f_0 + f_{\text{offset}}, \quad (1)$$

where f_0 is the center frequency (Center Freq (Hz)), f_{offset} is the offset frequency (Offset Freq (Hz)) set by either a Digital Oscillator, or an uploaded waveform (see in [Quantum Analyzer Setup Tab](#) or [Digital Modulation Tab](#)). The range of f_{offset} is from -1 GHz to 1 GHz. Please note that signals with an absolute offset frequency greater than 500 MHz will be attenuated significantly.

The frequency f_{in} of the input signal on each channel can be calculated as

$$f_{\text{in}} = f_0 + f_{\text{IF}}, \quad (2)$$

where f_{IF} is the intermediate frequency (IF) after frequency down-conversion. The down-converted signal can be monitored by the SHFQC+ Scope. In resonator spectroscopy experiments, the signal with f_{IF} is integrated by the same Digital Oscillator. In qubit readout experiments, it is integrated by an uploaded waveform.

Power Representation

The power P_{out} of the output signal on each channel is calculated as

$$P_{\text{out}} = \begin{cases} P_{\text{range, out}} + 20 \log_{10}(g_{\text{osc}}), & \text{Spectroscopy mode, Continuous} \\ P_{\text{range, out}} + 20 \log_{10}(g_{\text{osc}} A), & \text{Spectroscopy mode, Pulse} \\ P_{\text{range, out}} + 20 \log_{10}(A), & \text{Readout mode} \end{cases} \quad (3)$$

where $P_{\text{range, out}}$ is the output power range in units of dBm, g_{osc} ($g_{\text{osc}} \leq 1$) is the amplitude gain of the Digital Oscillator, A ($k \leq 1$) is the amplitude of an uploaded waveform. Please note that the 14-bit vertical resolution of the output signal counts both waveform amplitude and oscillator amplitude gain in Spectroscopy mode. To have full 14-bit vertical resolution on the uploaded waveform, the amplitude gain of the oscillator has to be 1.

The input signal can be monitored by the SHFQC+ Scope. The power P_{in} of the input signal on each channel can be calculated as

$$P_{\text{in}} = \begin{cases} 10 \log_{10} \frac{A_{\text{IF, I}}^2 + A_{\text{IF, Q}}^2}{50} + 30, & \text{RF path} \\ 10 \log_{10} \frac{A_{\text{IF, I}}^2}{100} + 30, & \text{LF path} \end{cases} \quad (4)$$

where $A_{\text{IF, I}}$ ($A_{\text{IF, Q}}$) is the amplitude of IF I (Q) components displayed on the SHFQC+ Scope in units of RMS voltage (V) in the RF (LF) path. The P_{in} is calculated in units of dBm.

In / Out Tab in the LabOne GUI

The In / Out settings can be accessed through the In / Out tab of the LabOne general user interface for the SHFQC+. After clicking on the tab, an [overview subtab](#) opens that displays all settings for all available Quantum Analyzer and Signal Generator Channels.

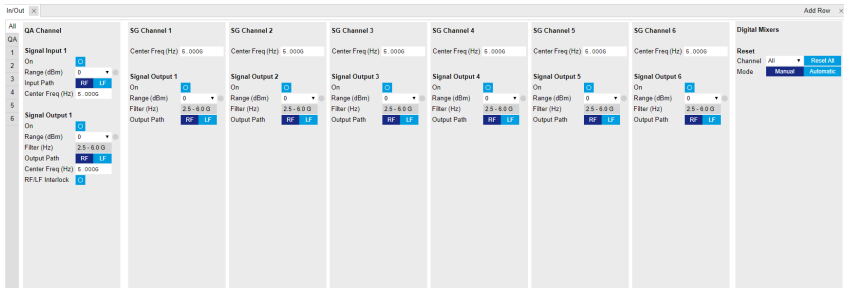


Figure 5.15: The Overview Sub-tab of the In / Out Tab

With the selectors at the left side of the In / Out tab, the [detailed view](#) of the up-conversion chain for the different Signal Generator Channels can be displayed, as well as the down-conversion chain for the Quantum Analyzer Channel. Each detailed view shows the available settings in the first, leftmost panel. In the second panel, graphical representations of the currently selected parameters of the up-conversion chain and down-conversion chain (if applicable) are displayed.

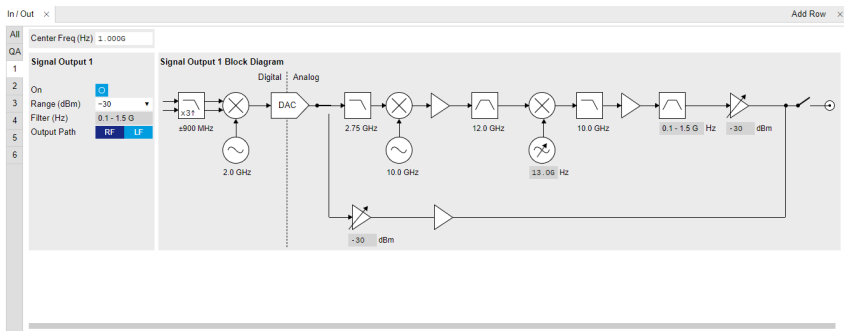


Figure 5.16: A detailed view of a Signal Generator Channel

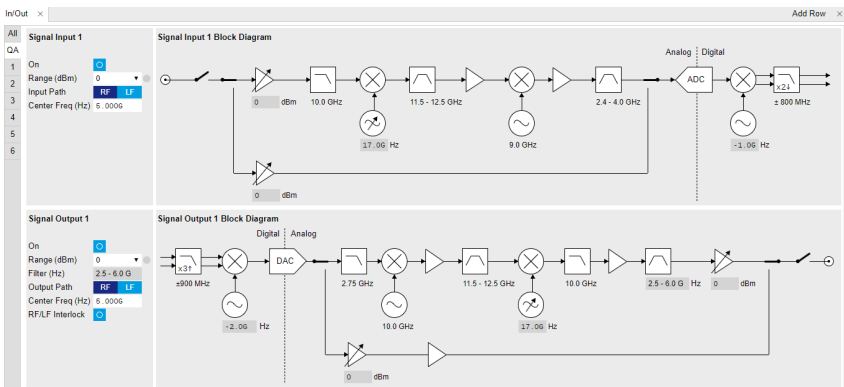


Figure 5.17: A detailed view of a Quantum Analyzer Channel

Functional Elements

The following [Table 5.16](#) summarizes all settings. The modes are all accessible through the SHFQC+ nodes in [Device Node Tree](#).

Table 5.16: Input and Output Settings

Control/Tool	Option/Range	Description
QA Channel		
Input On	On/Off	Enables the Signal Input.

Control/Tool	Option/Range	Description
Input Range	RF: -50 dBm to +10 dBm LF: -30 dBm to +10 dBm	Sets the maximal input power range that maps to the full scale of the ADC. The Ranges decrease in steps of 5 dB from +10 dBm.
Input Overload Status	grey/yellow/red	Indicates whether input power on the front panel P_{in} is or was above input power range $P_{in, range}$. red: P_{in} is $> P_{in, range}$ yellow: P_{in} was $> P_{in, range}$ within the last 200 ms grey: P_{in} is $\leq P_{in, range}$ Note that Input Overload condition may damage the Instrument. Please avoid it by increasing Input Range or attenuating input signal.
Input Path	RF or LF path	Select RF (0.5 - 8.5 GHz) or LF (DC - 800 MHz) input path.
Center Frequency	RF: 1-8 GHz LF: 0 Hz	Sets the center frequency of the analysis band in Hz. The frequency is rounded to the nearest multiple of 200 MHz. It is common to both the Input and Output of the same readout channel.
Output On	On/Off	Enables the Signal Output.
Output Range	RF: -30 dBm to +10 dBm LF: -30 dBm to +5 dBm	Sets the maximal output power range that maps from the full scale of the DAC. The Ranges decrease in steps of 5 dB from +10 dBm (RF) or +5 dBm (LF).
Output Overload Status	grey/yellow/red	Indicates whether output amplitude scaling factor g_{DAC} before DAC is or was above 1. It is checked for an overload condition every 10 ms. red: g_{DAC} is > 1 yellow: g_{DAC} was > 1 within the last 200 ms grey: g_{DAC} is ≤ 1 Note that overshoot of square pulse could cause Output Overload, and using e.g. flat-top Gaussian could avoid this.
Output Filter	Center frequency in RF path: Filter frequency 0.6 - 1.2 GHz: 0.1 - 1.5 GHz 1.4 - 3.0 GHz: 0.1 - 3.5 GHz 3.2 - 5.4 GHz: 2.5 - 6.0 GHz 5.6 - 8.0 GHz: 5.0 - 10.0 GHz	Indicates selected filter in the RF path. The filter value is selected according to the center frequency value and ensures that higher signal harmonics are removed at the Signal Output
Output Path	RF or LF path	Select RF (0.5 - 8.5 GHz) or LF (DC - 800 MHz) output path.
Center Frequency	RF: 1-8 GHz LF: 0 Hz	Sets the center frequency of the analysis band in Hz. The frequency is rounded to the nearest multiple of 200 MHz. It is common to both the Input and Output of the same readout channel.
RF/LF Interlock	On/Off	Enables the RF/LF path interlock between input and output. If enabled (1), the output path is always configured according to the input. The default value is disabled (0).
SG Channels		
Center Frequency	RF: 0.6 - 8 GHz LF: 0 - 2 GHz	Sets the center frequency of the analysis band in Hz. The frequency is rounded to the nearest multiple of 200 MHz (100 MHz) when using the RF (LF) path.
Output On	On/Off	Enables the Signal Output.

Control/Tool	Option/Range	Description
Output Range	RF: -30 dBm to +10 dBm LF: -30 dBm to +5 dBm	Sets the maximal output power range that maps from the full scale of the DAC. The Ranges decrease in steps of 5 dB from +10 dBm (RF) or +5 dBm (LF).
Output Overload Status	grey/yellow/red	Indicates whether output amplitude scaling factor g_{DAC} before DAC is or was above 1. It is checked for an overload condition every 10 ms. red: g_{DAC} is > 1 yellow: g_{DAC} was > 1 within the last 200 ms grey: g_{DAC} is ≤ 1 Note that overshoot of square pulse could cause Output Overload, and using e.g. flat-top Gaussian could avoid this.
Output Filter	Center frequency in RF path: Filter frequency 0.6 - 1.2 GHz: 0.1 - 1.5 GHz 1.4 - 3.0 GHz: 0.1 - 3.5 GHz 3.2 - 5.4 GHz: 2.5 - 6.0 GHz 5.6 - 8.0 GHz: 5.0 - 10.0 GHz	Indicates selected filter in the RF path. The filter value is selected according to the center frequency value and ensures that higher signal harmonics are removed at the Signal Output
Output Path	RF or LF path	Select RF (0.1 - 8.5 GHz) or LF (DC - 2 GHz) output path.
Digital Mixer of SG Channels		
Reset Channel	All or any available channel	Reset only the selected channel.
Reset Mode	Manual/Automaic	Configure the NCO (before DAC) reset mode. In Manual mode the instrument does not automatically reset NCOs when switching a channel from LF to RF mode. In Automatic mode the instrument automatically resets the NCOs of all channels whenever a channel is switched from LF to RF, in order to restore alignment.

5.2.2. Quantum Analyzer Setup Tab


The Quantum Analyzer Setup is the main control panel for the qubit measurement unit on the Instrument (see [Functional Overview](#) for an overview block diagram). It is available on all SHFQC+ Instruments.

Features

- Continuous or pulsed resonator spectroscopy
- Multiplexed qubit readout
- Weighted integration
- Multistate discrimination
- Power spectral density

Description

Table 5.17: App icon and short description

Control/Tool	Option/Range	Description
QA Setup		Configure the Qubit Measurement Unit

The Quantum Analyzer Setup tab is divided into 2 sub-tab groups for resonator spectroscopy (see [LabOne GUI: QA Setup Tab - Spectroscopy Mode](#)) and qubit readout (see [LabOne GUI: QA Setup Tab - Readout Mode](#)) application. By selecting Application Mode, Spectroscopy or Readout, the corresponding sub-tabs provide all configurations of readout pulse generation and acquired data processing (see [LabOne GUI: Readout Pulse Generator Tab - Waveform Viewer](#)). The main differences of the Application Modes are listed below.

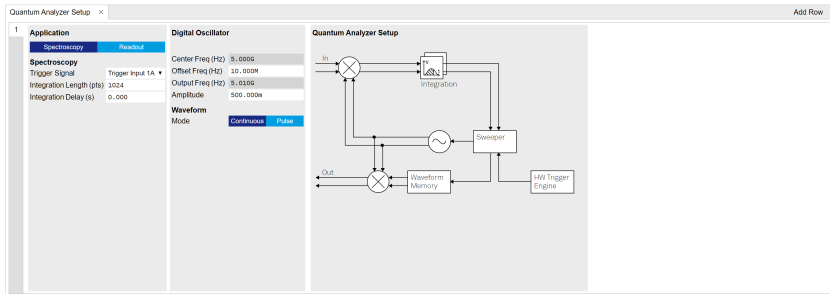


Figure 5.18: LabOne GUI: QA Setup Tab - Spectroscopy Mode

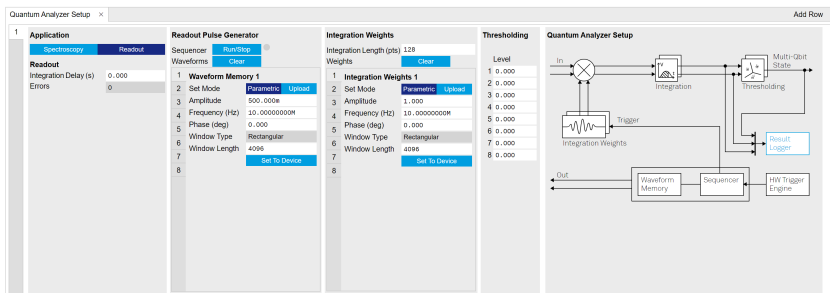


Figure 5.19: LabOne GUI: QA Setup Tab - Readout Mode

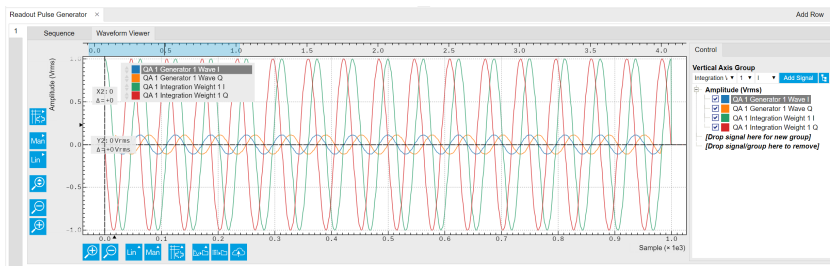


Figure 5.20: LabOne GUI: Readout Pulse Generator Tab - Waveform Viewer

Table 5.18: Main differences of Spectroscopy and Readout mode

Parameters	Spectroscopy mode	Readout mode
Waveform generation	Digital oscillator and envelope	Sample by Sample
Waveform output mode	Continuous or pulsed	Pulsed
Waveform length	Continuous or up to 16 μ s or 32 μ s	Up to 2 μ s
Number of qubits per channel	1	Up to 8 or 16
Integration length	Up to 16.7 ms	Up to 2 μ s
Integration weight unit per channel	Not applicable	Up to 8 or 16
Result normalization after integration	Normalized by integration length in samples	Not normalized
Real-time state discrimination	Not applicable	Yes
Multistate discrimination	Not applicable	Qubits, qutrits and ququads

Spectroscopy Mode

Spectroscopy mode is mainly used for resonator spectroscopy and power spectral density measurement. The SHFQC+ has 1 Digital Oscillator per channel. In Spectroscopy mode, the Digital Oscillator is used for readout waveform generation and integration. The SHFQC+ Sweeper class (API) is the central controller for the Spectroscopy mode, see tutorial [Resonator Spectroscopy](#).

There are 2 operation modes for readout waveform generation, Continuous and Pulsed. In Continuous mode, signal from the Digital Oscillator is routed to the digital IQ mixing stage (see [Functional Overview](#)) for readout waveform generation in the output path, and to multiply the input signal for readout waveform integration in the input path. signal from the Digital Oscillator modulates the envelope from the Waveform Memory and then is routed to the output and input paths same as in Continuous mode. The pulse envelope can be displayed on the Waveform Viewer sub-tab of the Readout Pulse Generator tab. The main differences of the 2 operation modes are listed in [Table 5.19](#).

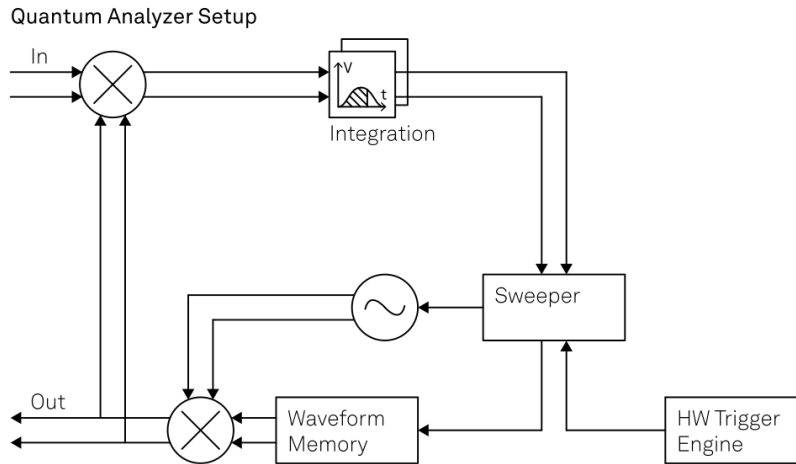


Figure 5.21: Readout waveform generation and integration in Spectroscopy mode

Table 5.19: Main differences of Continuous and Pulsed mode in Spectroscopy

Parameters	Continuous mode	Pulsed mode
Readout pulse length	Continuous wave	Up to 16 μs or 32 μs (up to 32 kSa or 64 kSa)
Envelope delay	Not applicable	Up to 131.1 μs
Readout pulses amplitude	Controlled by output range and gain factor of the Digital Oscillator	Controlled by output range, gain factor of the Digital Oscillator and the envelope

Readout Waveform Output In Spectroscopy Mode

The readout waveform on the Instrument output can be expressed as

$$E_{\text{output}}(t) = C_{P \rightarrow A} \text{Re}[A(t)g_{\text{osc}}e^{i2\pi(f_0+f_{\text{osc}})t+i\phi_{\text{output}}}], \quad (1)$$

where $C_{P \rightarrow A} = 10^{\frac{P_{\text{range, output}} - 10}{20}}$ is the conversion factor converting the power range $P_{\text{range, output}}$ of the output signal in units of dBm to the amplitude in units of V, $A(t)$ is the complex readout envelope in Pulsed mode and $A(t) = 1$ in Continuous mode, g_{osc} is the amplitude gain factor of the baseband Digital Oscillator. The frequency f_{osc} is the offset frequency set by the baseband Digital Oscillator and ϕ_{output} is the global phase, which can be reset by resetting the phase of the baseband Digital Oscillator. The frequency f_0 is the RF center frequency set in the Input/Output tab. Note: when using the LF path the center frequency from the Input/Output tab does not apply and one has to set $f_0 = 0$ in the above formula.

The readout waveform envelop $\mathbf{A(t)}$ in Pulsed mode can be uploaded via the API or manually in the GUI. Drag & drop the .csv file containing the waveform envelope as single column with complex values, for example 0.5+0.5j. The length of the readout pulse is automatically determined from the number of entries, the frequency modulation is automatically added as described above.

The readout waveform envelope $\mathbf{A(t)}$ in Pulsed mode can be displayed on the Waveform Viewer.

Readout Results In Spectroscopy Mode

The readout input signal after analog frequency down-conversion before the ADC is

$$\begin{aligned} E_{\text{before ADC}}(t) &= \text{Re}(\mathbf{A}_{\text{input}}(t)e^{i2\pi f_{\text{before ADC}}t + i\phi_{\text{input}}}) \\ &= \mathbf{A}_{\text{input}}(t) \cos(2\pi f_{\text{before ADC}}t + \phi_{\text{input}}) \\ &= \mathbf{A}_{\text{input}}(t) \frac{e^{i2\pi f_{\text{before ADC}}t + i\phi_{\text{input}}} + e^{-i2\pi f_{\text{before ADC}}t - i\phi_{\text{input}}}}{2}, \end{aligned} \quad (2)$$

where $\mathbf{A}_{\text{input}}(t)$ is the amplitude of input signal on the front panel of the instrument, $f_{\text{before ADC}}$ is the frequency of the input signal before ADC, ϕ_{input} is the global phase of the input signal on the front panel of the instrument. After the ADC, it becomes

$$E_{i, \text{ after ADC}} = \frac{C_{\text{scaling}} \mathbf{A}_{i, \text{ input}}}{2} (e^{i2\pi f_{\text{after ADC}}t_i + i\phi_{\text{input}}} + e^{-i2\pi f_{\text{after ADC}}t_i - i\phi_{\text{input}}}), \quad (3)$$

where C_{scaling} is the conversion factor depending on gain factor, ADC range and bit resolution, $f_{\text{after ADC}}$ is the frequency after the ADC, i means the i -th sample. The signal $E_{i, \text{ after ADC}}$ is then down-converted by a digital oscillator at frequency $f_{\text{input, LO2}}$ of 1 GHz (0 Hz) when using RF (LF) path and filtered the high frequency components, as

$$\begin{aligned} E_{i, \text{ before integration}} &= E_{i, \text{ after ADC}} e^{-i2\pi f_{\text{after, LO2}}t_i} \\ &= \frac{C_{\text{scaling}} \mathbf{A}_{i, \text{ input}}}{2} (e^{i2\pi f_{\text{after ADC}}t_i + i\phi_{\text{input}}} + e^{-i2\pi f_{\text{after ADC}}t_i - i\phi_{\text{input}}}) e^{-i2\pi f_{\text{input, LO2}}t_i}, \\ &= \frac{C_{\text{scaling}} \mathbf{A}_{i, \text{ input}}}{2} (e^{i2\pi f_{\text{baseband}}t_i + i\phi_{\text{input}}} + e^{-i2\pi f_{\text{sum}}t_i - i\phi_{\text{input}}}) \\ &\stackrel{\text{filter}}{=} \begin{cases} \frac{C_{\text{scaling}} \mathbf{A}_{i, \text{ input}}}{2} e^{i2\pi f_{\text{baseband}}t_i + i\phi_{\text{input}}}, & \text{RF path,} \\ C_{\text{scaling}} \mathbf{A}_{i, \text{ input}} \cos(2\pi f_{\text{baseband}}t_i + \phi_{\text{input}}), & \text{LF path,} \end{cases} \end{aligned} \quad (4)$$

where $f_{\text{baseband}} = f_{\text{after ADC}} - f_{\text{input, LO2}} = f_{\text{osc}}$ is the differential frequency, $f_{\text{sum}} = f_{\text{before ADC}} + f_{\text{input, LO2}}$ is the sum frequency. The signal at f_{sum} is filtered out by the digital filter. The baseband signal $E_{i, \text{ before integration}}$ can be monitored by the SHFQC+ Scope as

$$E_{\text{Scope}} = \begin{cases} \frac{\sqrt{2}}{C_{\text{scaling}}} E_{i, \text{ before integration}} = \frac{\mathbf{A}_{i, \text{ input}}}{\sqrt{2}} e^{i2\pi f_{\text{baseband}}t_i + i\phi_{\text{input}}}, & \text{RF path,} \\ \frac{1}{C_{\text{scaling}}} E_{i, \text{ before integration}} = \mathbf{A}_{i, \text{ input}} \cos(2\pi f_{\text{baseband}}t_i + \phi_{\text{input}}), & \text{LF path.} \end{cases} \quad (5)$$

Note that the conversion factor $\sqrt{2}/C_{\text{scaling}}$ is used for the RF path, and $1/C_{\text{scaling}}$ is used for LF path. The signal $E_{i, \text{ before integration}}$ is then demodulated with the signal $e^{-i2\pi f_{\text{osc}}t_i}$ ($f_{\text{osc}} = f_{\text{baseband}}$) from the baseband Digital Oscillator, integrated and normalized by the number of integration samples N ,

$$\begin{aligned} E_{\text{after integration}} &= \frac{1}{N} \sum_{i=1}^N E_{i, \text{ before integration}} e^{-i2\pi f_{\text{osc}}t_i}, \\ &= \begin{cases} \frac{C_{\text{scaling}}}{2N} \sum_{i=1}^N \mathbf{A}_{i, \text{ input}} e^{i\phi_{\text{input}}}, & \text{RF path,} \\ \frac{C_{\text{scaling}}}{2N} \sum_{i=1}^N \mathbf{A}_{i, \text{ input}} (e^{i\phi_{\text{input}}} + e^{-i4\pi f_{\text{baseband}}t_i - i\phi_{\text{input}}}), & \text{LF path.} \end{cases} \end{aligned} \quad (6)$$

If $\mathbf{A}_{i, \text{ input}} = \mathbf{A}_{\text{input}}$ is constant (and integration length is much longer than $1/(2f_{\text{osc}})$ with LF path), then $E_{\text{after integration}} = \frac{C_{\text{scaling}} \mathbf{A}_{\text{input}}}{2} e^{i\phi_{\text{input}}}$. By multiply the factor of $\sqrt{2}/C_{\text{scaling}}$, the units of the results is converted to $\frac{\mathbf{A}_{\text{input}}}{\sqrt{2}} e^{i\phi_{\text{input}}}$, and then can be downloaded from the instrument via the Instrument

node `/dev.../qachannels/n/spectroscopy/result/data/wave`, see [Device Node Tree](#). The power of input signal is then derived as

$$P_{\text{input}} = 10 \log_{10} \frac{|E_{\text{after integration}}|^2}{50} + 30. \quad (7)$$

The power and phase of the input signal can also be calculated and plotted using the Sweeper class.

Power spectral density

Power Spectral Density (PSD) measurements are generally required to characterize an amplification chain. In Spectroscopy mode, a PSD measurement can be performed using the SHFQC+ Sweeper, see [GitHub zhinst-toolkit example](#) or [zhinst-toolkit Online Documentation](#), or using instrument nodes, see [Device Node Tree](#).

Here, the PSD is calculated on the hardware as $S_{xx}(f) = \lim_{N \rightarrow \infty} \frac{(\Delta t)^2}{T} |\sum_{n=-N}^N x_n e^{-i2\pi f n \Delta t}|^2$ (see [Spectral density Wikipedia](#)), where $\Delta t = 1/f_s$ is the time step, f_s is the sampling rate, $T = (2N + 1)\Delta t$ is the integration length in seconds, $2N + 1$ is the integration length in samples, x_n is the n -th complex data of the input signal, $e^{-i2\pi f n \Delta t}$ is the integration weight. This calculation is done by the Instrument, and it returns the real-valued PSD in units of $V_{\text{rms}}^2/\text{Hz}$. The applicable ranges of the PSD measurement are listed in the table below.

Note that the measurement bandwidth is determined by the inverse of integration time, and the frequency step should be less than or equal to the measurement bandwidth. Typically, the PSD measurement requires many averages to be accurate. Setting the number of averages ≥ 1000 is recommended.

Table 5.20: Applicable ranges of PSD measurements

Parameters	Values
LabOne version	≥ 23.02
Number of channels	2 or 4 for SHFQA+; 1 for SHFQC+
Input frequency range	RF: 0.5 - 8.5 GHz LF: DC - 800 MHz
Input Power Range	RF: -50 to +10 dBm LF: -30 to +10 dBm
Input waveform length	continuous or pulsed (> 2 ns)
Measurement bandwidth (1 / integration time)	60 Hz to 500 MHz (16.7 ms to 2 ns)
Number of averages	1 to 131k
Input voltage noise density	see Specifications
Input spurious free dynamic range (excluding harmonics)	see Specifications

Readout Mode

Readout mode is mainly used for multi qubit readout. The SHFQC+ has 8 or 16 readout Waveform Memory slots, and 8 or 16 integration weight units per channel. In readout mode, these memory slots are used for readout pulse generation and weighted integration. The SHFQC+ Readout Pulse Generator is the central controller in Readout mode, see tutorial [Multiplexed Qubit Readout](#).

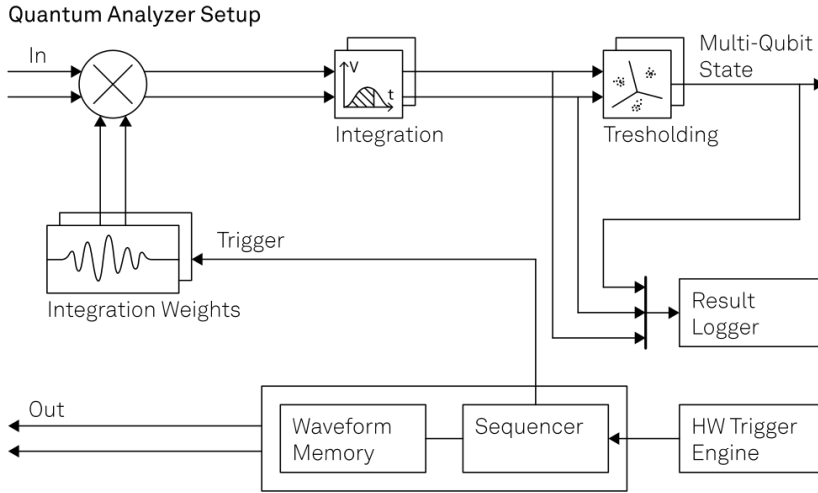


Figure 5.22: Readout waveform generation, integration and discrimination in Readout mode.

There are 2 state discrimination modes, 2-state discrimination (default, see tutorial [Multiplexed Qubit Readout](#) and multistate discrimination (see tutorial [Multistate discrimination](#)). Both modes are based on the linear [Support Vector Machine](#) and one versus one classification. Note that multistate discrimination can only be configured via LabOne APIs.

Base features support multiplexed readout at integration lengths up to 2 μs . For longer integration up to 32 μs , the Long Readout Time (LRT) option is required (see the functional diagram below). The LRT option adds three functions.

- added 6 oscillators for modulation and demodulation
- added Waveform Hold function for readout signal generation
- added downsampling function for weighted integration

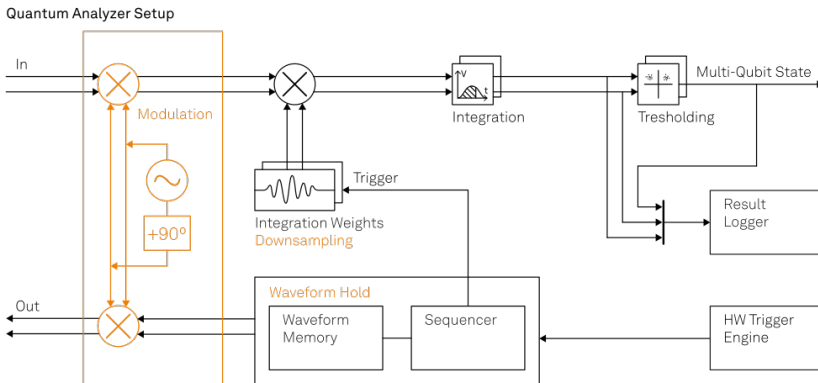


Figure 5.23: Readout waveform generation, integration and discrimination in Readout mode when modulation is enabled with the LRT option.

Readout Waveform Generation In Readout Mode

The readout waveform can be generated parametrically by LabOne GUI and APIs, and then uploaded and saved in the Waveform Memory. All readout waveforms saved in the Waveform Memory can be erased by clicking **Clear** on LabOne GUI or using LabOne APIs. Each readout waveform can be used for a single qudit readout, and up to 8 or 16 qubits can be readout simultaneously using a sum of the readout waveforms saved in the Waveform Memory. The Readout Waveform Generator controls which and how readout waveforms are played, see the [Readout Pulse Generator Tab](#).

The readout waveform saved in the j -th ($j \leq 8$ or 16) Waveform Memory slot is complex data, which can be expressed as

$$E_{i,j,\text{readout}} = A_{i,j,\text{readout}} e^{i2\pi f_{j,\text{offset}} t_i + i\phi_{j,\text{readout}}} \quad (8)$$

where $A_{i,j,\text{readout}}$ ($0 \leq A_{i,j,\text{readout}} \leq 1$) is the amplitude factor of the readout waveform at the i -th sample, $f_{j,\text{offset}}$ is the offset frequency, $\phi_{j,\text{readout}}$ is the global phase.

Without the LRT option, or with the LRT option enabled but both numerical oscillator modulation and waveform hold disabled, the front panel output signal is

$$E_{\text{output}}(t) = \text{Re}[\sum_j A_{j,\text{readout}}(t) e^{i2\pi(f_{j,\text{offset}} + f_0)t + i\phi_{j,\text{readout}}}] \quad (9)$$

where $A_{j,\text{readout}}(t)$ is the waveform envelope from waveform memory slot j . The output signal is sum of selected waveforms saved in the waveform memory. Note that the maximum amplitude factor of the sum of all waveforms in use should not exceed 1.

With the LRT option and numerical oscillator modulation both enabled, the front panel output signal is

$$E_{\text{output}}(t) = \text{Re}[\sum_j A'_{j,\text{readout}}(t) e^{i2\pi(f_{j,\text{offset}} + f_{\text{osc}_j} + f_0)t + i(\phi_{j,\text{readout}} + \phi_{\text{osc}_j})} A_{\text{osc}_j}] \quad (10)$$

where $A'_{j,\text{readout}}(t)$ is the waveform envelope from waveform memory slot j which is extendable by a hold function with configurable hold index and hold length, $f_{j,\text{offset}}$ is set to 0 so the offset frequency is defined by the oscillator j only (f_{osc_j}), A_{osc_j} is the oscillator gain, ϕ_{osc_j} is the oscillator phase. The output signal is sum of all oscillator-modulated waveforms, i.e. $j \leq 6$ limited by the number of available oscillators.

Integration Weights

The integration weights can be parametrically generated or measured with the SHFQC+ Scope for the best SNR (see tutorial [Integration Weights Measurement](#)), and uploaded to the integration weight units. All integration weights saved in the memory can be erased by clicking **Clear** on LabOne GUI Integration Weights sub-tab or using LabOne APIs. The length of the integration weight is automatically extended to 4096 Samples once it's uploaded, and **integration length** is used to configure how long it integrates. The Readout Waveform Generator controls which and how integration weights are used, see [Readout Pulse Generator Tab](#).

The integration weights saved in the k -th integration weight unit is complex data, can be expressed as

$$E_{i,k,\text{weights}} = A_{i,k,\text{weight}} e^{-i2\pi f_{k,\text{weight}} t_i - i\phi_{k,\text{weight}}} , \quad (11)$$

where $A_{i,k,\text{weight}}$ ($0 \leq A_{i,k,\text{weight}} \leq 1$) is the amplitude factor of the integration weight at i -th sample, $f_{k,\text{weight}}$ is the frequency of the integration weight, $\phi_{k,\text{weight}}$ is the global phase of the integration weight. Without the LRT option, or with LRT but modulation disabled, $f_{k,\text{weight}}$ equals one of the offset frequencies defined in readout waveform generation. With LRT and modulation enabled, $f_{k,\text{weight}}$ equals 0 because the signal is demodulated by the corresponding oscillator before weighted integration.

In 2-state discrimination mode, 1 qubit requires 1 integration weight, i.e. the conjugated difference of readout input signal while qubit is prepared in state $|0\rangle$ and $|1\rangle$. In multistate discrimination mode, 1 qudit with n states requires $n(n-1)/2$ integration weights (one vs one classification), i.e. the conjugated differences of any 2 readout input signal while qudit is prepared in state $|i\rangle$ and $|j\rangle$, where i ($0 \leq i \leq n-2$) and j ($1 \leq j \leq n-1$) are integer and $i \neq j$. Only $n-1$ integration weights need to be uploaded, and the integration results from the rest of integration weights are calculated by the Instrument automatically. Real-time multistate discrimination is not available for long ($> 2 \mu\text{s}$ and $\leq 32 \mu\text{s}$) multiplexed readout experiments that supported by the LRT option.

All integration weights saved in the integration weight units can be displayed on the Waveform Viewer, see [Waveform Viewer](#).

Note

In order to achieve the highest possible resolution in the signal after integration, it's advised to scale the dimensionless readout integration weights with a factor so that their maximum absolute value is equal to 1.

Thresholding

Thresholding sub-tab is used to configure thresholds for state discrimination in 2-state discrimination mode. In multistate discrimination mode, thresholds and assignment matrix are configured by LabOne APIs.

The input signal before integration is

$$E_{i, \text{ before integration}} = \begin{cases} C_{\text{scaling}} \sum_j \frac{A_{i, j, \text{ input}}}{2} e^{i2\pi f_{j, \text{ baseband}} t_i + \phi_{j, \text{ input}}}, & \text{RF path,} \\ C_{\text{scaling}} \sum_j A_{i, j, \text{ input}} \cos(2\pi f_{j, \text{ baseband}} t_i + \phi_{j, \text{ input}}), & \text{LF path,} \end{cases} \quad (12)$$

where i indicates the i -th sample of the input signal, j indicates the j -th component of the input signal for a qubit or qudit, $A_{i, j, \text{ input}}$ is the amplitude of j -th components of the input signal, $f_{j, \text{ baseband}}$ is the frequency of j -th component of the input signal, $\phi_{j, \text{ input}}$, is the phase of j -th component of input signal. The signal can be monitored by the SHFQC+ Scope with the same conversion factors as in Spectroscopy mode,

$$E_{\text{Scope}} = \begin{cases} \sum_j \frac{A_{i, j, \text{ input}}}{\sqrt{2}} e^{i2\pi f_{j, \text{ baseband}} t_i + i\phi_{j, \text{ input}}}, & \text{RF path,} \\ \sum_j A_{i, j, \text{ input}} \cos(2\pi f_{j, \text{ baseband}} t_i + \phi_{j, \text{ input}}), & \text{LF path.} \end{cases} \quad (13)$$

Without the LRT option (or with LRT but modulation turned off), the signal is directly integrated using the weights $A_{i, j, \text{ weight}} e^{-i2\pi f_{j, \text{ weight}} t_i - i\phi_{j, \text{ weight}}}$, where $f_{j, \text{ weight}} = f_{j, \text{ baseband}}$. By contrast, when LRT is enabled and modulation is active, the signal is first demodulated, then integrated with the weights $A_{i, j, \text{ weight}} e^{-i\phi_{j, \text{ weight}}}$, where $f_{j, \text{ weight}} = 0$ using a downsampling factor N (an integer from 1 to 16). The j -th integration result after weighted integration is

$$E_{j, \text{ after integration}} = \sum_{i=1}^N E_{i, j, \text{ before integration}} A_{i, j, \text{ weight}} e^{-i2\pi f_{j, \text{ baseband}} t_i - i\phi_{j, \text{ weight}}} \\ = \begin{cases} \frac{C_{\text{scaling}}}{2} \sum_{i=1}^N A_{i, j, \text{ input}} A_{i, j, \text{ weight}} e^{i(\phi_{j, \text{ input}} - \phi_{j, \text{ weight}})}, & \text{RF path,} \\ \frac{C_{\text{scaling}}}{2} \sum_{i=1}^N A_{i, j, \text{ input}} A_{i, j, \text{ weight}} \times \\ (e^{i(\phi_{j, \text{ input}} - \phi_{j, \text{ weight}})} + e^{-i4\pi f_{j, \text{ baseband}} t_i - i\phi_{j, \text{ weight}} - i\phi_{j, \text{ weight}}}), & \text{LF path.} \end{cases} \quad (14)$$

If $\phi_{j, \text{ input}} = \phi_{j, \text{ weight}}$, $A_{i, j, \text{ input}} = A_{i, j, \text{ weight}}$ is a constant, $A_{i, j, \text{ weights}} = 1$, and integration length is much longer than $1/(2f_{j, \text{ baseband}})$ with LF path, the result after integration can be simplified as $E_{j, \text{ after integration}} = \frac{NC_{\text{scaling}} A_{j, \text{ input}}}{2}$. This result with RF (LF) path is complex data, and can be downloaded and displayed on the [Quantum Analyzer Result Tab](#) with a conversion factor of $\sqrt{2}/C_{\text{scaling}}$ ($1/C_{\text{scaling}}$), as $\frac{NA_{j, \text{ input}}}{\sqrt{2}}$ ($\frac{NA_{j, \text{ input}}}{2}$).

To achieve the best SNR, the largest separation of qubit states and discriminate states with real data, one can apply optimal weights $(E_{j, |b\rangle, \text{ Scope}} - E_{j, |a\rangle, \text{ Scope}})^* / A_{\text{norm}}$, where $E_{j, |b\rangle, \text{ Scope}}$ ($E_{j, |a\rangle, \text{ Scope}}$) is the readout signal when qubit in state $|b\rangle$ ($|a\rangle$) before integration recorded by the scope, "*" is a conjugate operation, A_{norm} is a factor to normalize the optimal weights. The readout signal $E_{|b\rangle, \text{ after integration}}$ of a single qubit after weighted integration with the optimal weights is

$$E_{|b\rangle, \text{ after integration}} \\ = \begin{cases} \frac{C_{\text{scaling}}}{2} \sum_{i=1}^N \frac{A_{i, |b\rangle}}{A_{i, \text{ norm}}} e^{i\phi_{|b\rangle}} (A_{i, |b\rangle} e^{i\phi_{|b\rangle}} - A_{i, |a\rangle} e^{i\phi_{|a\rangle}})^*, & \text{RF path,} \\ C_{\text{scaling}} \sum_{i=1}^N \frac{A_{i, |b\rangle}}{A_{i, \text{ norm}}} \cos \phi_{|b\rangle} (A_{i, |b\rangle} \cos \phi_{|b\rangle} - A_{i, |a\rangle} \cos \phi_{|a\rangle}), & \text{LF path,} \end{cases} \quad (15) \\ = \begin{cases} \frac{C_{\text{scaling}}}{2} \sum_{i=1}^N \frac{A_{i, |b\rangle}}{A_{i, \text{ norm}}} [A_{i, |b\rangle} - A_{i, |a\rangle} \cos(\phi_{|b\rangle} - \phi_{|a\rangle}) - iA_{i, |a\rangle} \sin(\phi_{|b\rangle} - \phi_{|a\rangle})], & \text{RF path,} \\ C_{\text{scaling}} \sum_{i=1}^N \frac{A_{i, |b\rangle}}{A_{i, \text{ norm}}} \cos \phi_{|b\rangle} (A_{i, |b\rangle} \cos \phi_{|b\rangle} - A_{i, |a\rangle} \cos \phi_{|a\rangle}), & \text{LF path,} \end{cases}$$

where $A'_{i, \text{norm}} = \sqrt{A_{i, |b\rangle}^2 + A_{i, |a\rangle}^2 - 2A_{i, |a\rangle}A_{i, |b\rangle} \cos(\phi_{|a\rangle} - \phi_{|b\rangle})}$ ($A'_{i, \text{norm}} = A_{i, |b\rangle} \cos \phi_{|b\rangle} - A_{i, |a\rangle} \cos \phi_{|a\rangle}$) is a normalization factor, $A_{i, |b\rangle}$ ($A_{i, |a\rangle}$) and $\phi_{|b\rangle}$ ($\phi_{|a\rangle}$) is the amplitude and phase of the signal in state $|b\rangle$ ($|a\rangle$), respectively. Similarly, the readout signal $E_{|a\rangle}$, after integration after weighted integration with the optimal weights is

$$E_{|a\rangle, \text{ after integration}} = \begin{cases} \frac{C_{\text{scaling}}}{2} \sum_{i=1}^N \frac{A_{i, |a\rangle}}{A'_{i, \text{norm}}} [-A_{i, |a\rangle} + A_{i, |b\rangle} \cos(\phi_{|a\rangle} - \phi_{|b\rangle}) + iA_{i, |b\rangle} \sin(\phi_{|a\rangle} - \phi_{|b\rangle})], & \text{RF path,} \\ C_{\text{scaling}} \sum_{i=1}^N \frac{A_{i, |a\rangle}}{A'_{i, \text{norm}}} \cos \phi_{|a\rangle} (A_{i, |b\rangle} \cos \phi_{|b\rangle} - A_{i, |a\rangle} \cos \phi_{|a\rangle}), & \text{LF path,} \end{cases} \quad (16)$$

Take the real part of the integrated results, and the 2 states can be discriminated by a threshold as

$$\begin{aligned} & \frac{\text{Re}[E_{|b\rangle, \text{ after integration}}] + \text{Re}[E_{|a\rangle, \text{ after integration}}]}{2} \\ & = \begin{cases} \frac{C_{\text{scaling}}}{2} \sum_{i=1}^N \frac{1}{A'_{i, \text{norm}}} (A_{i, |b\rangle}^2 - A_{i, |a\rangle}^2), & \text{RF path,} \\ C_{\text{scaling}} \sum_{i=1}^N \frac{1}{A'_{i, \text{norm}}} (A_{i, |b\rangle}^2 \cos^2 \phi_{|b\rangle} - A_{i, |a\rangle}^2 \cos^2 \phi_{|a\rangle}), & \text{LF path.} \end{cases} \end{aligned} \quad (17)$$

The separation between state $|b\rangle$ and state $|a\rangle$ can be calculated directly as

$\sqrt{\sum_{i=1}^N (A_{i, |b\rangle}^2 + A_{i, |a\rangle}^2 - 2A_{i, |a\rangle}A_{i, |b\rangle} \cos(\phi_{|b\rangle} - \phi_{|a\rangle}))}$, and it is the same as calculated from

$$|\text{Re}[E_{|b\rangle, \text{ after integration}}] - \text{Re}[E_{|a\rangle, \text{ after integration}}]|. \quad (18)$$

In 2-state discrimination mode, all integration weights can be uploaded via LabOne GUI and APIs, and all integration results saved in the result logger can be displayed on the [Quantum Analyzer Result Tab](#) if **integration** is selected as result source. In multistate discrimination mode, all integration weights can only be uploaded via LabOne APIs, and only $n - 1$ integration results are saved in the result logger if **integration** is selected as result source, the rest are calculated automatically in the pairwise difference units.

Before state discrimination, threshold T_j for each qudit has to be estimated and uploaded to the Instrument with an internal conversion factor $\frac{2}{C_{\text{scaling}}} (\frac{1}{C_{\text{scaling}}})$ in RF (LF) path, see tutorial [Multistate discrimination](#)). During thresholding, the real part of the result after integration is compared with a threshold, and it returns 0 or 1. Qubit state discrimination can be done in both 2-state and multistate discrimination modes. The readout result of qubit after thresholding is

$$E_j, \text{ after thresholding} = \begin{cases} 0 & \text{Re}[E_j, \text{ after integration}] \leq T_j, \\ 1 & \text{Re}[E_j, \text{ after integration}] > T_j, \end{cases} \quad (19)$$

where T_j is the threshold of the j -th qubit. The result after thresholding and state assignment can represent qubit state directly.

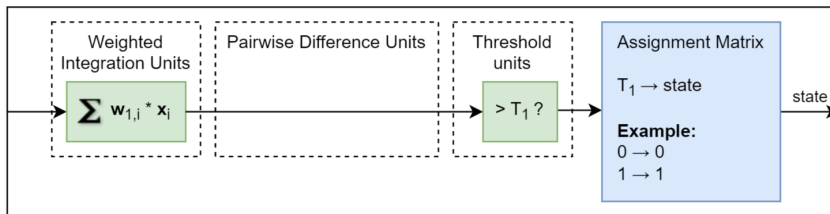


Figure 5.24: Readout data processing for qubits. The green blocks are used for both state discrimination modes and the blue blocks are used for multistate discrimination mode only configured by LabOne APIs. (x_i) is the readout input signal at the (i) -th sample, $(w_{1,i})$ is the optimal integration weight calculated from the difference while qubit is prepared in state $|0\rangle$ and $|1\rangle$, (T_1) is the threshold for the integrated result, $(b_1 = 0)$ or (1) is the binary value after thresholding. The assignment matrix is defined such that if the $(b_1 = 0)$ (1) the result after discrimination is 1 (0), i.e. state $|1\rangle$ ($|0\rangle$).

For a qutrit, 3 thresholds are used, 2 for the integration results in the weighted integration units and 1 for the integration result in the pairwise difference units. After thresholding, the 3-bit data is assigned to 0, 1 or 2 by the assignment matrix, and the discriminated results can be displayed in the [Quantum Analyzer Result Tab](#). The state assignment can be customized by uploading 8 integer numbers (0, 1 or 2) for a single qutrit via API. The discriminated result represented by 2-bit data can be transferred to control instruments via DIO and ZSync for feedback experiment.

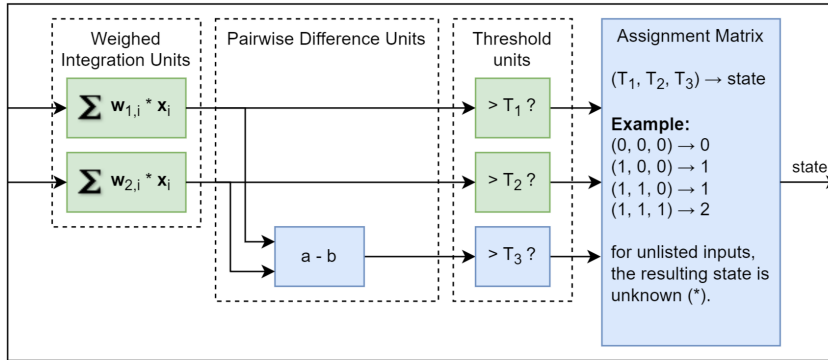


Figure 5.25: Readout data processing for qutrits. The green blocks are used for both state discrimination modes and the blue blocks are used for multistate discrimination mode only configured by LabOne APIs. x_i is the readout input signal at the i -th sample, $w_{1,i}$ and $w_{2,i}$ are the integration weights measured by calculating the readout pulse difference when the qutrit is prepared in state $|0\rangle$ and $|1\rangle$, and state $|0\rangle$ and $|2\rangle$, respectively. $(r_2 - r_1)$ calculated by the instrument is the pairwise difference of the integration results, (T_1) , (T_2) and (T_3) are the thresholds for the 3 integration results, (b_1) , (b_2) and (b_3) are the binary value after the thresholding. The assignment matrix is defined such that the result after discrimination is 0, 1 or 2, i.e. state $|0\rangle$, $|1\rangle$ or $|2\rangle$, respectively. Result with "*" is assigned ambiguously because all states are equally likely.

For a ququad, 6 thresholds are used, 3 for the integration results in the weighted integration units and 3 for the integration results in the pairwise difference units. After thresholding, the 6-bit data is assigned to 0, 1, 2 or 3 by the assignment matrix, and the discriminated results can be displayed in the [Quantum Analyzer Result Tab](#). The state assignment can be customized by uploading 64 decimal numbers (0, 1, 2 or 3) for a single ququad via API. The discriminated result represented by 2-bit data can be transferred to control instruments via DIO and ZSync for feedback experiment.

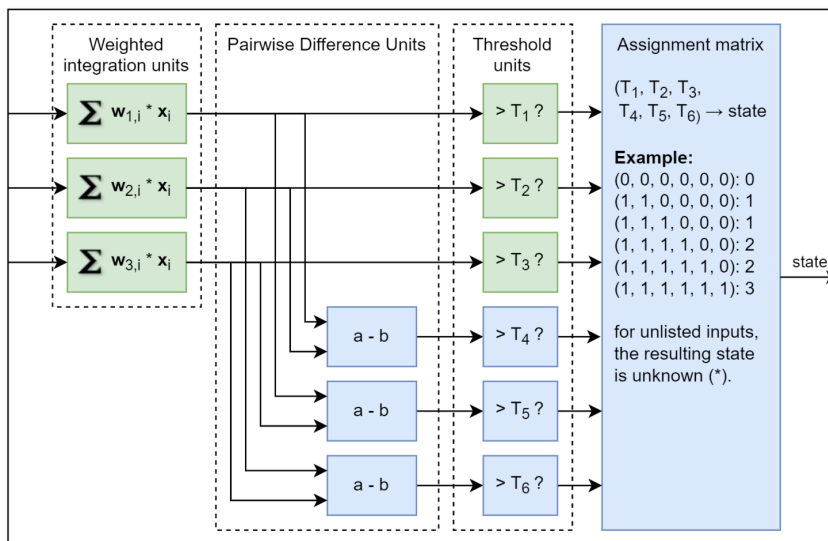


Figure 5.26: Readout data processing for ququads. x_i is the readout input signal at the i -th sample, $w_{1,i}$, $w_{2,i}$ and $w_{3,i}$ are the integration weights measured by calculating the readout pulse difference when the ququad is prepared in state $|0\rangle$ and $|1\rangle$, state $|0\rangle$ and $|2\rangle$, and state $|0\rangle$ and $|3\rangle$, respectively. $(r_2 - r_1)$, $(r_3 - r_1)$, $(r_3 - r_2)$ calculated by the instrument are the pairwise differences of the integration results, (T_1) , (T_2) , (T_3) , (T_4) , (T_5) and (T_6) are the thresholds for 6 integration results, (b_1) , (b_2) , (b_3) , (b_4) , (b_5) and (b_6) are the binary value after thresholding. The assignment matrix is defined such that the result after discrimination is 0, 1, 2 or 3, i.e. state $|0\rangle$, $|1\rangle$, $|2\rangle$ or $|3\rangle$, respectively. Result with "*" is assigned ambiguously because 2 or more states are equally likely.

Functional Elements

Table 5.21: QA setup settings

Control/Tool	Option/Range	Description
Application Mode	Spectroscopy	Using internal digital oscillator for waveform generation and integration.
	Readout	Using uploaded waveform for output signal generation and customized weights for integration.
Spectroscopy		
Trigger Signal		Selects the source of the trigger for the integration and envelope in Spectroscopy mode.
Integration Length	2^2 to 2^{25}	Sets the integration length in Spectroscopy mode in number of samples. Up to 33.5 MSa (2^{25} samples, with granularity of 4 Samples) can be recorded, which corresponds to 16.7 ms.
Integration Delay	-4 ns to 131.1 μ s	Sets the delay of the integration in Spectroscopy mode with respect to the trigger signal. The resolution is 2 ns.
Power Spectral Density	Enable/Disable	Enable or disable power spectral density mode.
Center Frequency	RF: 1-8 GHz LF: 0 Hz	Display center frequency in Spectroscopy mode.
Offset Frequency	-1 to +1 GHz	Set offset frequency to the internal digital oscillator in Spectroscopy mode.
Output Frequency	DC - 8.5 GHz	Display frequency of the output signal in Spectroscopy mode.
Amplitude	0 to 1	Set gain of the internal digital oscillator in Spectroscopy mode. The recommended range is from 0.01 to 1 in pulsed mode.
Waveform Mode	Continuous	The output of the internal digital oscillator is used directly for frequency up-conversion.
	Pulse	The waveform envelope is modulated by the internal digital oscillator before frequency up-conversion.
Length	4 to 32 kSa or 64 kSa (with 16W option)	Indicate the length of uploaded envelope waveform in units of Samples. The granularity is 4 Samples.
Delay	0 ns to 131.1 μ s	Set a delay between readout pulse playback trigger and the first sample of the readout pulse (in Pulsed mode). The resolution is 2 ns.
File Upload	CSV file	Drop CSV file to upload the envelope waveform.
Readout		
Integration Delay	0 ns to 131.1 μ s	Sets a common delay for the start of the readout integration for all Integration Weights with respect to the time when the trigger is received. The resolution is 2 ns.
Errors	Number	Number of hold-off errors detected since last reset.
Modulation (LRT option)	Enable/Disable	Enable or disable modulation.
Frequency (LRT option)	-1 GHz to 1 GHz	Set oscillator frequency.
Gain (LRT option)	0 to 1	Set oscillator gain.
Sequencer Run/Stop	Run or Stop	Enables the Sequencer.
Waveforms Clear		Empty all readout Waveform Memory slots or Integration weight Units.
Waveform/Weight Generation Mode	Parametric or Upload	Select the way to generate waveform.
Parametric Amplitude	0 to 1	Set amplitude factor for parametric readout pulse and integration weight generation.

Control/Tool	Option/Range	Description
Parametric Frequency	-1 to +1 GHz	Set offset frequency for parametric readout pulse or integration weight generation.
Parametric Phase	-180 to 180 degree	Set phase for parametric readout pulse and integration weight generation.
Parametric Window Type	Rectangular	Display window function to be applied in complex exponential function for parametric readout pulse and integration weight generation.
Parametric Window Length	4 to 4096	Length of the selected window in samples for parametric readout pulse and integration weight generation.
Waveform/Weight Set To Device	Yes or No	Set parametrically generated readout pulse and integration weight to waveform memory slot and integration memory slot, respectively.
Hold (LRT option)	Enable/Disable	Enable or disable hold function.
Hold Start Index (LRT option)	4 to 4092	Set an index of waveform data that the corresponding value will be hold for a defined period.
Hold Length (LRT option)	8 to 4194300	Set hold length in number of samples.
Integration Length	4 to 4096	Sets the length of all Integration Weights in number of samples. A maximum of 4096 samples can be integrated, which corresponds to 2.05 μ s. The granularity is 4 Samples.
Downsampling Factor (LRT option)	1 to 16	Set downsampling factor to extend integration length.
Thresholding	-14.51 kV to 14.51 kV	Set threshold for quantum state discrimination. Note that the data before thresholding is not normalized by the integration length.

5.2.3. Quantum Analyzer Result Tab


The Quantum Analyzer Result tab is the interface to the Result Logger unit of the Instrument and displays processed data after the qubit measurement unit (see [Functional Overview](#) for an overview block diagram). It is available on all SHFQC+ Instruments.

Features

- Configure result source, result length and averaging
- Display readout results in Readout and Spectroscopy mode with different coordinates

Description

Table 5.22: App icon and short description

Control/Tool	Option/Range	Description
QA Result		Configure the Result Logger.

The Quantum Analyzer Result tab (see [Figure 5.27](#)) is divided into a display section on the left and a configuration section on the right.

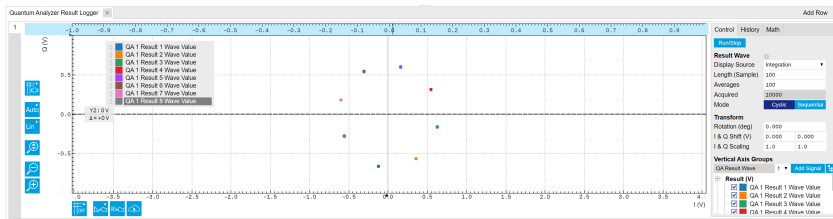


Figure 5.27: LabOne UI: Quantum Analyzer Result Tab.

This tool allows users to acquire, average, and analyze large sets of data sourced at various points of the signal processing chain. The data source setting is listed in [Table 5.23](#)

The data are stored in a vector with a length of up to 2^{19} points and displayed in the plot area on the left once the acquisition is complete. The Result Logger supports hardware averaging and 2 averaging modes. The complex readout result after integration can be displayed in different coordinates, i.e. complex plane (IQ plane), amplitude and different coordinates, i.e. complex plane (IQ plane), amplitude and phase versus measurement points.

Note that the QA Result Logger will be turned off automatically only if the number of acquired data equals the product of the **Length** and the **Averages** set in the tab. A timeout error may occur if the QA Result Logger does not receive enough trigger events, e.g. if the number of readouts configured in the Sequencer is less than the product of the **Length** and **Averages**, or if the readout repetition rate exceeds $1/(440 \text{ ns})$ (including the minimum integration hold off time of 20 ns).

Functional Elements

Table 5.23: QA result settings.

Control/Tool	Option/Range	Description
Spectroscopy		Configure Result Logger and display result from Spectroscopy mode.
Readout		Configure Result Logger and display result from Readout mode.
Run/Stop		Run/Stop the Result Logger.
Plot Type	Dot Plot or Components	Select between dot plot in IQ plane or components plot (I, Q, amplitude and phase) in Spectroscopy sub-tab.
Length	2^0 to 2^{19}	Number of data points to record. One data point corresponds to a single averaged output of the selected source. The granularity is 1.
Averages	2^0 to 2^{17}	Number of averages per recorded data point. The granularity is 1.
Acquired	Length x Averages	Indicate the index of the data point that will be recorded next.
Mode	Cyclic or Sequential	Select Cyclic or Sequential averaging. With Cyclic averaging, the first point of the Result vector is the average of the results number 1, M+1, 2M+1, and so forth, where M is equal to the Length setting. The second point is the average of the results number 2, M+2, 2M+2, and so forth. With Sequential averaging, the first point of the Result vector is the average of the first N results, where N is equal to the Averages setting. The second point of the Result vector is the average of the following N results, and so forth.
Display Source	Integration or Threshold	Data or averaged data after weighted integration (Integration) or state discrimination (Threshold).

5.2.4. Readout Pulse Generator Tab


The Readout Pulse Generator tab is the main control panel for readout measurement sequences. It is available on all SHFQC+ Instruments.

Features Overview

- 1 Sequencer for the Readout Channel
- 8 or 16 readout waveform memory slots, 4 kSa for each memory slot
- 8 or 16 integration weights memory slots, 4 kSa for each memory slot
- Sequence branching
- Access to multiple internal triggers
- Interface to DIO and ZSync for synchronization and feedback
- High-level programming language
- Display waveforms in waveform memory and integration weight units

Description

Table 5.24: App icon and short description

Control/Tool	Option/Range	Description
Generator		Generate readout measurement sequences.

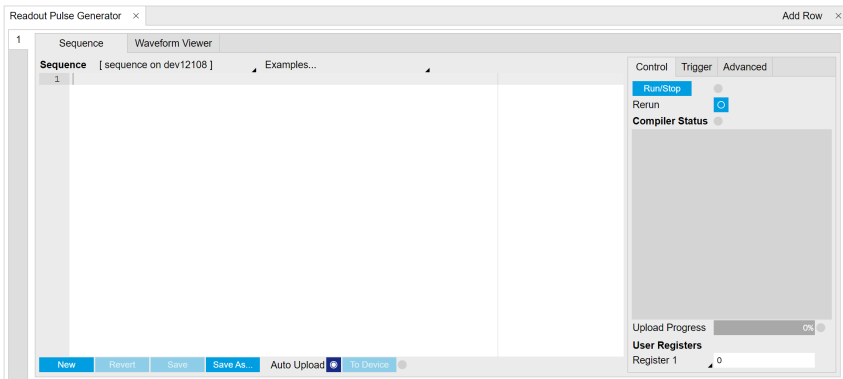


Figure 5.28: SHFQC+ Readout Pulse Generator Tab

The **Sequencer Editor** can be considered the central control unit of the SHFQC+ as it has access to the playback of the Waveform Memories to generate Readout Pulses, the start of the Integration of the Readout Signals from the experiment and the communication with additional devices through the DIO or ZSync. The programming language **SeqC** is based on C and specified in detail in [SeqC language](#). In contrast to other AWG Sequencers, e.g. from the HDAWG, it does not provide writing access to the Waveform Memories and hence does not come with predefined waveforms. The stricter separation between Sequencer and Waveform Memory allows implementation of more advanced and application-specific features, e.g. Time-Staggered Readout, while still providing real-time sequencing.

The Sequencer features a compiler which translates the high-level sequence program ([SeqC](#)) into machine instructions to be stored in the Instrument sequencer memory. The sequence program is written using high-level control structures and syntax that are inspired by human language, whereas machine instructions reflect exactly what happens on the hardware level. Writing the sequence program using [SeqC](#) represents a more natural and efficient way of working in comparison to writing lists of machine instructions, which is the traditional way of programming AWGs. Concretely, the improvements rely on features such as:

- Waveform playback and sequencing in a single script
- Easily readable syntax and naming for run-time variables and constants
- Definition of user functions and procedures for advanced structuring
- Syntax validation

By design, there is no one-to-one link between the list of statements in the high-level language and the list of instructions executed by the Sequencer. In order to understand the execution timing, it's helpful to consider the internal architecture of the Readout Pulse Generator, consisting of the Sequencer itself, and the [Waveform Memory](#) including a Waveform Player.

On the **Control** sub-tab the user configures signal parameters and controls the execution of the sequencer. The sequencer can be started by clicking on **Run/Stop**. When enabling the **Rerun** button, the Sequencer will be restarted automatically when its program completes. The **Compiler Status** shows whether compilation is successful, or generated warnings or errors.

On the **Trigger** sub-tab users can configure the trigger inputs of the sequencer and control the Hardware Trigger Engine functionality of the Instrument. The sequencer can be triggered by Digital trigger source including DIO trigger or ZSync trigger. Only Digital trigger and DIO trigger are configured in this sub-tab. ZSync trigger is configured in [Device Tab](#). There are 2 digital triggers that can be configured for each sequencer. The options of Digital trigger source include,

- the physical trigger input A and B of all channels,
- the sequencer trigger defined in all sequencers, such as through the command **setTrigger**,
- the readout done of all channels,
- the software trigger.

The sequencer can also be triggered by DIO trigger input with chosen Valid bit and Polarity (see [DIO Tab](#))

The **Advanced** sub-tab displays the compiled list of sequencer instructions and the current state of the sequencer on the Instrument. This can help an advanced user in debugging a sequence program and understanding its execution.

Sequencer Operation

Every pulse sequence requires defining a SeqC program. For an example of how to define and upload a sequence, see the [Pulse Spectroscopy Tutorial](#). The status of the upload can be monitored via the [Ready node](#). Once it returns true, the compilation is successful and the program is transferred to the device. If the compilation fails, the [Status node](#) will display debug messages.

After successful uploading of a sequence to the Instrument, the Sequencer can be started using the [Enable node](#).

If the Sequencer should wait for a Trigger Input Signal, it can either directly wait for a [ZSync Trigger](#), or access the [Auxiliary Triggers](#).

With the SHFQC+ utility functions, the above-mentioned steps can be realized by a single function. A set of Python API examples can be found in [GitHub](#)

All nodes for the Sequencer can be accessed through the node trees,

```
/dev..../qachannels/n/generator/...
and
/dev..../qachannels/n/generator/sequencer/....
```

SeqC

The syntax of the LabOne AWG Sequencer programming language is based on C, but with a few simplifications. Each statement is concluded with a semicolon, several statements can be grouped with curly brackets, and comment lines are identified with a double slash.

The following example shows some of the fundamental functionalities: repeated playback, triggering, and single/dual-channel waveform playback and readout. See [Tutorials](#) for a step-by-step introduction with more examples. The command **waitZSyncTrigger** is used to wait a trigger from [PQSC Programmable Quantum System Controller](#) via ZSync. Alternatively, an external digital trigger from the front panel or an internal trigger can also be used to start the sequence by the command **waitDigTrigger**. The first **playZero** sets the delay between the trigger and the first readout pulse, and the second **playZero** sets the delay between the first and the second readout pulse. The command **startQA** sends out internal triggers to play readout pulses saved in the waveform memory, to start integrations with waveforms saved in the integration weight units, and to send a Sequencer monitor trigger which can be used to trigger the Scope. The third **playZero** ensures that the previous commands **playZero** are finished.

```
// repeat sequence 100 times
repeat (100) {
    // wait for a trigger over ZSync. Assume the trigger period is longer than
    the cycle time
    waitZSyncTrigger();
```

```

// alternatively wait for a trigger from digital trigger 1
// waitDigTrigger(1);

// wait for 4096 Samples between the trigger and the first readout pulse
// Note: this playZero command does not yet block the sequencer
playZero(4096);

// define how many samples to wait between the two upcoming startQA commands
// Note: this command blocks the sequencer until the previous playZero
command is finished
playZero(4096);

// play the pulse stored in Waveform Memory slot 0 and read out using
Integration Weight 0
startQA(QA_GEN_0, QA_INT_0, true, 0x0, 0x0);

// minimal duration playZero command to wait until the previous playZero
command is finished
playZero(32);

// play the pulse stored in Waveform Memory slot 0, 1 and 2, and read out
using all Integration Weights
startQA(QA_GEN_0|QA_GEN_1|QA_GEN_2, QA_INT_ALL, true, 0x0, 0x0);
}

```

Keywords and Comments

The following table lists the keywords used in the LabOne AWG Sequencer language.

Table 5.25: Programming keywords

Keyword	Description
const	Constant declaration
var	Integer variable declaration
cvar	Compile-time variable declaration
string	Constant string declaration
true	Boolean true constant
false	Boolean false constant
for	For-loop declaration
while	While-loop declaration
repeat	Repeat-loop declaration
if	If-statement
else	Else-part of an if-statement
switch	Switch-statement
case	Case-statement within a switch
default	Default-statement within a switch
return	Return from function or procedure, optionally with a return value

The following code example shows how to use comments.

```

const a = 10; // This is a line comment. Everything between the double
              // slash and the end of the line will be ignored.

```



```
/* This is a block comment. Everything between the start-of-block-comment and
end-of-block-comment markers is ignored.
```

For example, the following statement will be ignored by the compiler.

```
const b = 100;
*/
```

Constants and Variables

Constants may be used to make the program more readable. They may be of integer or floating-point type. It must be possible for the compiler to compute the value of a constant at compile time, i.e., on the host computer. Constants are declared using the **const** keyword.

Compile-time variables may be used in computations and loop iterations during compile time, e.g. to create large numbers of waveforms in a loop. They may be of integer or floating-point type. They are used in a similar way as constants, except that they can change their value during compile time operations. Compile-time variables are declared using the **cvar** keyword.

Variables may be used for making simple computations during run time, i.e., on the Instrument. The Sequencer supports integer variables, addition, and subtraction. Not supported are floating-point variables, multiplication, and division. Typical uses of variables are to step waiting times, to output DIO values, or to tag digital measurement data with a numerical identifier. Variables are declared using the **var** keyword.

The following code example shows how to use variables.

```
var b = 100; // Create and initialize a variable

// Repeat the following block of statements 100 times
repeat (100) {
    b = b + 1; // Increment b
    wait(b);   // Wait 'b' cycles
}
```

The following table shows the predefined constants. These constants are intended to be used as arguments in certain run-time evaluated functions that encode device parameters with integer numbers.

Constants whose value is marked as "opaque" are meant to always be used instead of their numerical value.

Table 5.26: Predefined Constants

Name	Value	Description
M_E	2.71828182845904523536028747135266250	e
M_LOG2E	1.44269504088896340735992468100189214	log ₂ (e)
M_LOG10E	0.434294481903251827651128918916605082	log ₁₀ (e)
M_LN2	0.693147180559945309417232121458176568	log _e (2)
M_LN10	2.30258509299404568401799145468436421	log _e (10)
M_PI	3.14159265358979323846264338327950288	pi
M_PI_2	1.57079632679489661923132169163975144	pi/2
M_PI_4	0.785398163397448309615660845819875721	pi/4
M_1_PI	0.318309886183790671537767526745028724	1/pi
M_2_PI	0.636619772367581343075535053490057448	2/pi
M_2_SQRTPI	1.12837916709551257389615890312154517	2/sqrt(pi)
M_SQRT2	1.41421356237309504880168872420969808	sqrt(2)

Name	Value	Description
M_SQRT1_2	0.707106781186547524400844362104849039	1/sqrt(2)

Numbers can be expressed using either of the following formatting.

```
const a = 10;           // Integer notation
const b = -10;          // Negative number
const h = 0xdeadbeef;   // Hexadecimal integer
const bin = 0b10101;    // Binary integer
const f = 0.1e-3;       // Floating point number.
const not_float = 10e3; // Not a floating point number
```

Booleans are specified with the keywords `true` and `false`. Furthermore, all numbers that evaluate to a nonzero value are considered **true**. All numbers that evaluate to zero are considered **false**.

Strings are delimited using `"` and are interpreted as constants. Strings may be concatenated using the `+` operator.

```
string AWG_PATH = "awgs/0/";
string AWG_GAIN_PATH = AWG_PATH + "gains/0";
```

Waveform Playback and Predefined Functions

The following table contains the definition of functions for waveform playback and other purposes.

void setDIO(var value)

Writes the value as a 32-bit value to the DIO bus.

The value can be either a const or a var value. Configure the Mode setting in the DIO tab when using this command. The DIO interface speed of 50 MHz limits the rate at which the DIO output value is updated.

Args:

- **value:** The value to write to the DIO (const or var)

var getDIO()

Reads a 32-bit value from the DIO bus.

Returns:

var containing the read value

var getDIOTriggered()

Reads a 32-bit value from the DIO bus as recorded at the last DIO trigger position.

Returns:

var containing the read value

void setTrigger(var value)

Sets the Sequencer Trigger output signals.

The state of the two Sequencer Trigger output signals is defined by the bits in the binary representation of the integer value. Allowed parameter values are 0 to 3. For higher integer values, only the two least-significant bits will have an effect. Binary notation of the form 0b00 is recommended for readability.

Args:

- **value**: to be written to the trigger distribution unit

void wait(var cycles)

Waits for the given number of Sequencer clock cycles (4 ns per cycle). The execution of the instruction adds an offset of 2 clock cycles, i.e., the statement wait(3) leads to a waiting time of $5 * 4 \text{ ns} = 20 \text{ ns}$.

Note: the minimum waiting time amounts to 3 cycles, which means that wait(0) and wait(1) will both result in a waiting time of $3 * 4 \text{ ns} = 12 \text{ ns}$.

Args:

- **cycles**: number of cycles to wait

void waitDIOTrigger()

Waits until the DIO interface trigger is active. The trigger is specified by the Strobe Index and Strobe Slope settings in the AWG Sequencer tab.

var getDigTrigger(const index)

Gets the state of the indexed Digital Trigger input (1 or 2).

The physical signal connected to the Digital Trigger input is to be configured in the Readout section of the Quantum Analyzer Setup tab.

Args:

- **index**: index of the Digital Trigger input to be read; can be either 1 or 2

Returns:

trigger state, either 0 or 1

void error(string msg,...)

Throws the given error message when reached.

Args:

- **msg**: Message to be displayed

void info(string msg,...)

Returns the specified message when reached.

Args:

- **msg**: Message to be displayed

void setUserReg(const register, var value)

Writes a value to one of the User Registers (indexed 0 to 15).

The User Registers may be used for communicating information to the LabOne User Interface or a running API program.

Args:

- **register:** The register index (0 to 15) to be written to
- **value:** The integer value to be written

var getUserReg(const register)

Reads the value from one of the User Registers (indexed 0 to 15). The User Registers may be used for communicating information to the LabOne User Interface or a running API program.

Args:

- **register:** The register to be read (0 to 15)

Returns:

current register value

void playZero(var samples)

Zero Playback, which can be used to specify spacings in number of samples at the sample rate of 2 GSa/s between the execution times of commands, such as startQA. Each playZero command blocks the execution of subsequent commands when a previous Zero Playback is already running. Note: the playback of actual waveforms with the startQA command happens in parallel to the Zero Playback, in contrast to the HDAWG and SHFSG!

Args:

- **samples:** Number of samples for the spacing. The minimal spacing is 32 samples and the granularity is 16 samples.

void playZero(var samples, const rate)

Zero Playback, which can be used to specify spacings in number of samples between the execution times of commands, such as startQA. Each playZero command blocks the execution of subsequent commands when a previous Zero Playback is already running. Note: the playback of actual waveforms with the startQA command happens in parallel to the Zero Playback, in contrast to the HDAWG and SHFSG!

Args:

- **rate:** Sample rate with which the spacing is specified. Divides the device sample rate by 2^{rate} . Note: this rate does not affect the sample rate of the QA waveform generator (startQA command).
- **samples:** Number of samples for the spacing. The minimal spacing is 32 samples and the granularity is 16 samples.

void waitDigTrigger(const index)

Waits for the reception of a trigger signal on the indexed Digital Trigger (index 1 or 2). The physical signals connected to the two AWG Digital Triggers are to be configured in the Trigger sub-tab of the AWG Sequencer tab. The Digital Triggers are configured separately for each AWG Core.

Args:

- **index:** Index of the digital trigger input; can be either 1 or 2.

void configFreqSweep(const oscillator_index, const freq_start, const freq_increment)

Configures a frequency sweep.

Args:

- **freq_increment**: Specify how much to increment the frequency for each step of the sweep [Hz]
- **freq_start**: Specify the start frequency value for the sweep [Hz]
- **oscillator_index**: Index of the oscillator that will be used for the sweep

void resetOscPhase(const mask)

void resetOscPhase()

Reset the phase of the oscillator controllable by the sequencer. Each sequencer can control the oscillator of its QACHANNEL.

void setSweepStep(const oscillator_index, var sweep_index)

Executes a step within a frequency sweep.

Args:

- **oscillator_index**: Index of the oscillator that will be used for the sweep
- **sweep_index**: Sets the step index, from which the frequency is set

void setOscFreq(const oscillator_index, const freq)

Configures the frequency of an oscillator.

Args:

- **freq**: Frequency to be set [Hz]
- **oscillator_index**: Index of oscillator

var getFeedback(const data_type)

Read the last received feedback message. The argument specify which data the function should return.

Args:

- **data_type**: Specifies which data the function should return: ZSYNC_DATA_RAW: Returns the last ZSync message received.

Returns:

var containing the read value

var getFeedback(const data_type, var wait_cycles)

Read the last received feedback message. The argument specify which data the function should return.

Args:

- **data_type**: Specifies which data the function should return: ZSYNC_DATA_RAW: Returns the last ZSync message received.
- **wait_cycles**: Wait for the specified number of cycles after the most recent waitZSyncTrigger() instruction.

Returns:

var containing the read value

void waitZSyncTrigger()

Waits for a trigger over ZSync.

void startQA(const waveform_generator_mask, const weighted_integrator_mask, const monitor, const result_address, const trigger)

Starts the Quantum Analysis (QA) Readout Waveform Generation, Integration, and Result units.

Args:

- ─ **monitor**: Enable the Sequencer Monitor Trigger, which is issued simultaneously with the start of the weighted integration units. In addition to setting this argument to true, the Sequencer Monitor Trigger must be selected as trigger source for the SHFQA Scope in order to align the start of the time trace to the start of the weighted integration. Default: false.
- ─ **result_address**: Specify the address of the PQSC readout register in which to store the readout result from this SHFQA. Please refer to the PQSC user manual for more details. Default: 0x0
- ─ **trigger**: Sets the sequencer trigger output in the same manner as the setTrigger() command. Default: 0b00
- ─ **waveform_generator_mask**: Readout Waveform Generator unit enable mask. Providing a value for this argument is mandatory. The mask can be specified using the predefined constants QA_GEN_n, where n is an index ranging from 0 to 15, except for the 2-channel SHFQA without 16W option, where the range only spans from 0 to 7. To construct more elaborate masks that enable multiple units, combine these predefined constants using the operator | (bit-wise OR). The constant QA_GEN_ALL can be used to enable all units simultaneously. NOTE: the signals from simultaneously enabled Waveform Generation units are combined by a digital adder.
- ─ **weighted_integrator_mask**: Integration unit enable mask, default: QA_INT_ALL. The mask can be specified using the predefined constants QA_INT_n, where n is an index ranging from 0 to 15, except for the 2-channel SHFQA without 16W option, where the range only spans from 0 to 7. To construct more elaborate masks that enable multiple units, combine these predefined constants using the operator | (bit-wise OR). The constant QA_INT_ALL can be used to enable all units simultaneously.

Expressions

Expressions may be used for making computations based on mathematical functions and operators. There are two kinds of expressions: those evaluated at compile time (when the sequencer program is compiled on the computer), and those evaluated at run time.

Compile-time evaluated expressions only involve constants (**const**) or compile-time variables (**cvar**) and can be computed at compile time by the host computer. Such expressions can make use of standard mathematical functions and floating point arithmetic.

Run-time evaluated expressions involve variables (**var**) and are evaluated by the Sequencer on the Instrument. Due to the limited computational capabilities of the Sequencer, these expressions may only operate on integer numbers and there are less operators available than at compile time.

The following table contains the list of mathematical functions supported at compile time.

Table 5.27: Mathematical Functions

Function	Description
const abs(const c)	absolute value
const acos(const c)	inverse cosine
const acosh(const c)	hyperbolic inverse cosine
const asin(const c)	inverse sine
const asinh(const c)	hyperbolic inverse sine
const atan(const c)	inverse tangent
const atanh(const c)	hyperbolic inverse tangent
const cos(const c)	cosine

Function	Description
const cosh(const c)	hyperbolic cosine
const exp(const c)	exponential function
const ln(const c)	logarithm to base e (2.71828...)
const log(const c)	logarithm to the base 10
const log2(const c)	logarithm to the base 2
const log10(const c)	logarithm to the base 10
const sign(const c)	sign function -1 if x<0; 1 if x>0
const sin(const c)	sine
const sinh(const c)	hyperbolic sine
const sqrt(const c)	square root
const tan(const c)	tangent
const tanh(const c)	hyperbolic tangent
const ceil(const c)	smallest integer value not less than the argument
const round(const c)	round to nearest integer
const floor(const c)	largest integer value not greater than the argument
const avg(const c1, const c2,...)	mean value of all arguments
const max(const c1, const c2,...)	maximum of all arguments
const min(const c1, const c2,...)	minimum of all arguments
const pow(const base, const exp)	first argument raised to the power of second argument
const sum(const c1, const c2,...)	sum of all arguments

The following table contains the list of predefined mathematical constants. These can be used for convenience in compile-time evaluated expressions.

Table 5.28: Predefined Constants

Name	Value	Description
AWG_RATE_2000MHZ	0	Constant to set Sampling Rate to 2.0 GHz.
AWG_RATE_1000MHZ	1	Constant to set Sampling Rate to 1.0 GHz.
AWG_RATE_500MHZ	2	Constant to set Sampling Rate to 500 MHz.
AWG_RATE_250MHZ	3	Constant to set Sampling Rate to 250 MHz.
AWG_RATE_125MHZ	4	Constant to set Sampling Rate to 125 MHz.
AWG_RATE_62P5MHZ	5	Constant to set Sampling Rate to 62.5 MHz.
AWG_RATE_31P25MHZ	6	Constant to set Sampling Rate to 31.25 MHz.
AWG_RATE_15P63MHZ	7	Constant to set Sampling Rate to 15.63 MHz.
AWG_RATE_7P81MHZ	8	Constant to set Sampling Rate to 7.81 MHz.
AWG_RATE_3P9MHZ	9	Constant to set Sampling Rate to 3.9 MHz.
AWG_RATE_1P95MHZ	10	Constant to set Sampling Rate to 1.95 MHz.
AWG_RATE_976KHZ	11	Constant to set Sampling Rate to 976 kHz.
AWG_RATE_488KHZ	12	Constant to set Sampling Rate to 488 kHz.
AWG_RATE_244KHZ	13	Constant to set Sampling Rate to 244 kHz.
DEVICE_SAMPLE_RATE	<actual device sample rate>	

Name	Value	Description
ZSYNC_DATA_RAW	opaque	Constant to use as argument to getFeedback. Returns the last ZSync message received.
QA_INT_0	(1 << 0)	Constant to enable Integration unit 0 in the Integration unit enable mask of the function startQA(). To construct more elaborate masks that enable multiple units, combine these predefined constants using the operator
QA_INT_1	(1 << 1)	Constant to enable Integration unit 1 in the Integration unit enable mask of the function startQA(). To construct more elaborate masks that enable multiple units, combine these predefined constants using the operator
QA_INT_2	(1 << 2)	Constant to enable Integration unit 2 in the Integration unit enable mask of the function startQA(). To construct more elaborate masks that enable multiple units, combine these predefined constants using the operator
QA_INT_3	(1 << 3)	Constant to enable Integration unit 3 in the Integration unit enable mask of the function startQA(). To construct more elaborate masks that enable multiple units, combine these predefined constants using the operator
QA_INT_4	(1 << 4)	Constant to enable Integration unit 4 in the Integration unit enable mask of the function startQA(). To construct more elaborate masks that enable multiple units, combine these predefined constants using the operator
QA_INT_5	(1 << 5)	Constant to enable Integration unit 5 in the Integration unit enable mask of the function startQA(). To construct more elaborate masks that enable multiple units, combine these predefined constants using the operator
QA_INT_6	(1 << 6)	Constant to enable Integration unit 6 in the Integration unit enable mask of the function startQA(). To construct more elaborate masks that enable multiple units, combine these predefined constants using the operator
QA_INT_7	(1 << 7)	Constant to enable Integration unit 7 in the Integration unit enable mask of the function startQA(). To construct more elaborate masks that enable multiple units, combine these predefined constants using the operator
QA_INT_8	(1 << 8)	Constant to enable Integration unit 8 in the Integration unit enable mask of the function startQA(). To construct more elaborate masks that enable multiple units, combine these predefined constants using the operator
QA_INT_9	(1 << 9)	Constant to enable Integration unit 9 in the Integration unit enable mask of the function startQA(). To construct more elaborate masks that enable multiple units, combine these predefined constants using the operator
QA_INT_10	(1 << 10)	Constant to enable Integration unit 10 in the Integration unit enable mask of the function startQA(). To construct more elaborate masks that enable multiple units, combine these predefined constants using the operator
QA_INT_11	(1 << 11)	Constant to enable Integration unit 11 in the Integration unit enable mask of the function startQA(). To construct more elaborate masks that enable multiple units, combine these predefined constants using the operator

Name	Value	Description
QA_INT_12	$(1 \ll 12)$	Constant to enable Integration unit 12 in the Integration unit enable mask of the function startQA(). To construct more elaborate masks that enable multiple units, combine these predefined constants using the operator
QA_INT_13	$(1 \ll 13)$	Constant to enable Integration unit 13 in the Integration unit enable mask of the function startQA(). To construct more elaborate masks that enable multiple units, combine these predefined constants using the operator
QA_INT_14	$(1 \ll 14)$	Constant to enable Integration unit 14 in the Integration unit enable mask of the function startQA(). To construct more elaborate masks that enable multiple units, combine these predefined constants using the operator
QA_INT_15	$(1 \ll 15)$	Constant to enable Integration unit 15 in the Integration unit enable mask of the function startQA(). To construct more elaborate masks that enable multiple units, combine these predefined constants using the operator
QA_INT_ALL	$(1 \ll 16) - 1$	Constant to enable all Integration units in the Integration unit enable mask of the function startQA().
QA_INT_NONE	0	Constant to be used in the Integration unit enable mask of the function startQA() to represent the scenario when no integrator is enabled.
QA_GEN_0	$(1 \ll 0)$	Constant to enable Readout Waveform Generator unit 0 in the Readout Waveform Generator unit enable mask of the function startQA(). To construct more elaborate masks that enable multiple units, combine these predefined constants using the operator
QA_GEN_1	$(1 \ll 1)$	Constant to enable Readout Waveform Generator unit 1 in the Readout Waveform Generator unit enable mask of the function startQA(). To construct more elaborate masks that enable multiple units, combine these predefined constants using the operator
QA_GEN_2	$(1 \ll 2)$	Constant to enable Readout Waveform Generator unit 2 in the Readout Waveform Generator unit enable mask of the function startQA(). To construct more elaborate masks that enable multiple units, combine these predefined constants using the operator
QA_GEN_3	$(1 \ll 3)$	Constant to enable Readout Waveform Generator unit 3 in the Readout Waveform Generator unit enable mask of the function startQA(). To construct more elaborate masks that enable multiple units, combine these predefined constants using the operator
QA_GEN_4	$(1 \ll 4)$	Constant to enable Readout Waveform Generator unit 4 in the Readout Waveform Generator unit enable mask of the function startQA(). To construct more elaborate masks that enable multiple units, combine these predefined constants using the operator
QA_GEN_5	$(1 \ll 5)$	Constant to enable Readout Waveform Generator unit 5 in the Readout Waveform Generator unit enable mask of the function startQA(). To construct more elaborate masks that enable multiple units, combine these predefined constants using the operator

Name	Value	Description
QA_GEN_6	$(1 \ll 6)$	Constant to enable Readout Waveform Generator unit 6 in the Readout Waveform Generator unit enable mask of the function startQA(). To construct more elaborate masks that enable multiple units, combine these predefined constants using the operator
QA_GEN_7	$(1 \ll 7)$	Constant to enable Readout Waveform Generator unit 7 in the Readout Waveform Generator unit enable mask of the function startQA(). To construct more elaborate masks that enable multiple units, combine these predefined constants using the operator
QA_GEN_8	$(1 \ll 8)$	Constant to enable Readout Waveform Generator unit 8 in the Readout Waveform Generator unit enable mask of the function startQA(). To construct more elaborate masks that enable multiple units, combine these predefined constants using the operator
QA_GEN_9	$(1 \ll 9)$	Constant to enable Readout Waveform Generator unit 9 in the Readout Waveform Generator unit enable mask of the function startQA(). To construct more elaborate masks that enable multiple units, combine these predefined constants using the operator
QA_GEN_10	$(1 \ll 10)$	Constant to enable Readout Waveform Generator unit 10 in the Readout Waveform Generator unit enable mask of the function startQA(). To construct more elaborate masks that enable multiple units, combine these predefined constants using the operator
QA_GEN_11	$(1 \ll 11)$	Constant to enable Readout Waveform Generator unit 11 in the Readout Waveform Generator unit enable mask of the function startQA(). To construct more elaborate masks that enable multiple units, combine these predefined constants using the operator
QA_GEN_12	$(1 \ll 12)$	Constant to enable Readout Waveform Generator unit 12 in the Readout Waveform Generator unit enable mask of the function startQA(). To construct more elaborate masks that enable multiple units, combine these predefined constants using the operator
QA_GEN_13	$(1 \ll 13)$	Constant to enable Readout Waveform Generator unit 13 in the Readout Waveform Generator unit enable mask of the function startQA(). To construct more elaborate masks that enable multiple units, combine these predefined constants using the operator
QA_GEN_14	$(1 \ll 14)$	Constant to enable Readout Waveform Generator unit 14 in the Readout Waveform Generator unit enable mask of the function startQA(). To construct more elaborate masks that enable multiple units, combine these predefined constants using the operator
QA_GEN_15	$(1 \ll 15)$	Constant to enable Readout Waveform Generator unit 15 in the Readout Waveform Generator unit enable mask of the function startQA(). To construct more elaborate masks that enable multiple units, combine these predefined constants using the operator
QA_GEN_ALL	$(1 \ll 16) - 1$	Constant to enable all Waveform Generator units in the waveform generator unit enable mask of the function startQA().

Name	Value	Description
QA_GEN_NONE	0	Constant to be used in the Waveform Generator unit enable mask of the function startQA() to represent the scenario when no Waveform Generator is enabled.

Control Structures

Functions may be declared using the **var** keyword. **Procedures** may be declared using the **void** keyword. Functions must return a value, which should be specified using the **return** keyword. Procedures can not return values. Functions and procedures may be declared with an arbitrary number of arguments. The **return** keyword may also be used without arguments to return from an arbitrary point within the function or procedure. Functions and procedures may contain variable and constant declarations. These declarations are local to the scope of the function or procedure.

```
var function_name(argument1, argument2, ...) {
    // Statements to be executed as part of the function.
    return constant-or-variable;
}
void procedure_name(argument1, argument2, ...) {
    // Statements to be executed as part of the procedure.

    // Optional return statement
    return;
}
```

An **if-then-else** structure is used to create a conditional branching point in a sequencer program.

```
// If-then-else statement syntax
if (expression) {
    // Statements to execute if 'expression' evaluates to 'true'.
} else {
    // Statements to execute if 'expression' evaluates to 'false'.
}

// If-then-else statement short syntax
(expression)?(statement if true):(statement if false)

// If-then-else statement example
const REQUEST_BIT = 0x0001;
const ACKNOWLEDGE_BIT = 0x0002;
const IDLE_BIT = 0x8000;
var dio = getDIO();

if (dio & REQUEST_BIT) {
    dio = dio | ACKNOWLEDGE_BIT;
    setDIO(dio);
} else {
    dio = dio | IDLE_BIT;
    setDIO(dio);
}
```

A **switch-case** structure serves to define a conditional branching point similarly to the **if-then-else** statement, but is used to split the sequencer thread into more than two branches. Unlike the **if-then-else** structure, the switch statement is synchronous, which means that the execution time is the same for all branches and determined by the execution time of the longest branch. If no default case is provided and no case matches the condition, all cases will be skipped. The case arguments need to be of type **const**.

```
// Switch-case statement syntax
switch (expression) {
```

```

        case const-expression:
            expression;
        ...
default:
    expression;
}
// Switch-case statement example
switch (getDIO()) {
    case 0:
        startQA(QA_GEN_0, QA_INT_0, true, 0x0, 0x0);
    case 1:
        startQA(QA_GEN_1, QA_INT_1, true, 0x0, 0x0);
    case 2:
        startQA(QA_GEN_2, QA_INT_2, true, 0x0, 0x0);
    default:
        startQA(QA_GEN_3, QA_INT_3, true, 0x0, 0x0);
}

```

The **for** loop is used to iterate through a code block several times. The initialization statement is executed before the loop starts. The end-expression is evaluated at the start of each iteration and determines when the loop should stop. The loop is executed as long as this expression is **true**. The iteration-expression is executed at the end of each loop iteration. Depending on how the for loop is set up, it can be either evaluated at compile time or at run time. For a run-time evaluated for loop, use the **var** data type as a loop index. To ensure that a loop is evaluated at compile time, use the **cvar** data type as a loop index. Furthermore, the compile-time for loop should only contain waveform generation/editing operations and it can't contain any variables of type **var**.

The following code example shows both versions of the loop.

```

// For loop syntax
for (initialization; end-expression; iteration-expression) {
    // Statements to execute while end-expression evaluates to true
}

// For loop example (compile-time execution)
cvar i;
wave w_pulses;
for (i = 0; i < 10; i = i + 1) {
    startQA(QA_GEN_0<<1, QA_INT_0, true, 0x0, 0x0);
}

// For loop example (run-time execution)
var k;
var j;
for (j = 9; j >= 0; j = j - 1) {
    startQA(QA_GEN_0, QA_INT_0, true, 0x0, 0x0);
    k += j;
}

```

The **while** loop is a simplified version of the **for** loop. The end-expression is evaluated at the start of each loop iteration. The contents of the loop are executed as long as this expression is **true**. Like the **for** loop, this loop comes in a compile-time version (if the end-expression involves only **cvar** and **const**) and in a run-time version (if the end-expression involves also **var** data types).

```

// While loop syntax
while (end-expression) {
    // Statements to execute while end-expression evaluates to true
}

// While loop example
const STOP_BIT = 0x8000;
var run = 1;
var i = 0;
var dio = 0;
while (run) {
    dio = getDIO();
}

```

```

    run = dio & STOP_BIT;
    dio = dio | (i & 0xff);
    setDIO(dio);
    i = i + 1;
}

```

The **repeat** loop is a simplified version of the **for** loop. It repeats the contents of the loop a fixed number of times. In contrast to the **for** loop, the repetition number of the repeat loop must be known at compile time, i.e., **const-expression** can only depend on constants and not on variables. Unlike the **for** and the **while** loop, this loop comes only in a run-time version. Thus, no **cvar** data types may be modified in the loop body.

```

// Repeat loop syntax
repeat (constant-expression) {
    // Statements to execute
}

// Repeat loop example
repeat (100) {
    setDIO(0x1);
    wait(10);
    setDIO(0x0);
    wait(10);
}

```

Waveform Memory

The Waveform Memory stores the different complex-valued arbitrary waveforms that are used to readout the qubits. They can be accessed through `/dev.../qachannels/n/generator/waveforms/n/wave` and have a maximal length of 4096 samples and a vertical range between -1 and 1 relative to the full scale of the Output Range.

Functional Elements

Table 5.29: SHFQC+ Readout Pulse Generator: Control sub-tab

Control/Tool	Option/Range	Description
Start	ON/OFF	Run the Generator Sequencer.
Rerun	ON/OFF	Puts the Sequencer into single-shot mode or rerun mode.
Status		Display compiler errors and warnings.
Compile Status	grey/green/yellow/red	Sequence program compilation status. Grey: No compilation started yet. Green: Compilation successful. Yellow: Compiler warnings (see status field). Red: Compilation failed (see status field).
Upload Progress	0% to 100%	The percentage of the sequencer program already uploaded to the device.
Upload Status	grey/yellow/green	Indicates the upload status of the compiled sequence. Grey: Nothing has been uploaded. Yellow: Upload in progress. Green: Compiled sequence has been uploaded.
Register Selector	1 to 16	Select the number of the user register value to be edited.
Register	0 to 2^{32}	Integer user register value. The sequencer has reading and writing access to the user register values during run time.
Input File		External source code file to be compiled.
Example File		Load pre-installed example sequence program.
New		Create a new sequence program.

Control/Tool	Option/Range	Description
Revert		Undo the changes made to the current program and go back to the contents of the original file.
Save (Ctrl+S)		Compile and save the current program displayed in the Sequence Editor. Overwrites the original file.
Save As... (Ctrl+Shift+S)		Compile and save the current program displayed in the Sequence Editor under a new name.
Automatic Upload	ON / OFF	If enabled, the sequence program is automatically uploaded to the device after clicking Save and if the compilation was successful.
To Device		Sequence program will be compiled and, if the compilation was successful, uploaded to the device.

Table 5.30: SHFQC+ Readout Pulse Generator: Trigger sub-tab

Control/Tool	Option/Range	Description
Status	grey/green/yellow/red	Displays the status of the sequence on the Instrument. Off: Ready, not running. Green: Running, not waiting for any trigger event. Yellow: Running, waiting for a trigger event. Red: Not Ready.
Digital Trigger	1 or 2	Choose Digital Trigger 1 or Digital Trigger 2
Signal		Selects Digital Trigger source signal. Navigate through the tree view that appears and click on the required signal.
Valid Index		Selects the index n of the DIO interface bit (notation DIO[n] in the Specification chapter of the User Manual) to be used as a VALID signal input, i.e. a qualifier indicating that a valid codeword is available on the DIO interface.
Valid Polarity		Polarity of the VALID bit that indicates that a codeword is available on the DIO interface.
Low	VALID bit must be logical low.	
High	VALID bit must be logical high.	
Both	VALID bit may be logical high or logical low.	
None	VALID bit is ignored.	

Table 5.31: SHFQC+ Readout Pulse Generator: Advanced sub-tab

Control/Tool	Option/Range	Description
Assembly	string	Displays the current sequence program in compiled form. Every line corresponds to one hardware instruction and requires one clock cycle (4 ns) for execution.
Status	running/idle/waiting	Displays the status of the sequencer on the Instrument. Off: Ready, not running. Green: Running, not waiting for any trigger event. Yellow: Running, waiting for a trigger event. Red: Not ready (e.g., pending elf download, no elf downloaded)
Mem Usage	0% to 100%	Size of the current sequence program relative to the device cache memory. The cache memory provides space for a maximum of 16384 instructions.

5.2.5. Digital Modulation Tab


The Digital Modulation tab can be used to configure the digital oscillators, as well as the settings used to modulate pulse sequences and generate sinusoidal signals. It is available on all SHFQC+ instruments.

Features

- Sine generator configuration: frequency, oscillator select, harmonic, phase, amplitude
- Gain settings for upper- or lower-sideband modulation
- Enable pulse modulation or continuous signal output

Description

Table 5.32: App icon and short description

Control/Tool	Option/Range	Description
Mod		Access to all the settings of the digital modulation.

The Digital Modulation tab (see [Figure 5.29](#)) is divided into three sections: Oscillators, Sine Generators, and Waveform Generators.

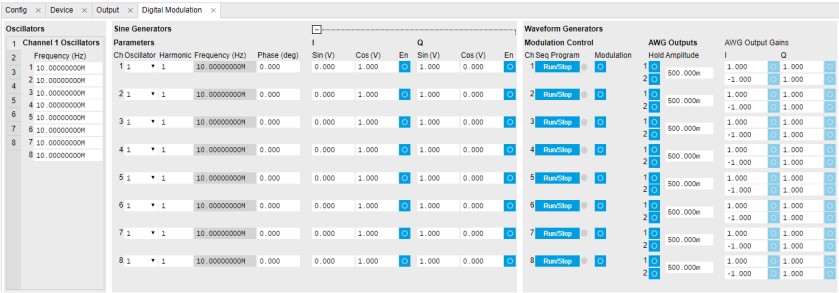


Figure 5.29: LabOne UI: Digital Modulation tab

The purpose of the Digital Modulation tab is to configure the digital sine generator of each SG channel, to enable the modulation (i.e. multiplication) or addition of sinusoidal and AWG signals. The tabular layout of the tab provides a quick overview of the status of the different channels of the instrument.


Conceptually, the tab is laid out as follows: The Oscillators section contains the frequencies of the eight digital oscillators for each channel. The Sine Generators section contains settings such as phase and harmonic for the sine generator of each channel, as well as settings for generating sinusoidal signal outputs. The Waveform Generators section configures how the sine generator is used to modulate the AWG signals. The signal on a given output can be a multiplication, or addition, or both, of AWG and sinusoidal signals, depending which modulation modes are enabled.

The individual sinusoidal and AWG signals are configured in the Sine Generators and Waveform Generators sections, respectively. For an example of how to generate a continuous, sinusoidal signal on a given channel, the see [Basic Sine Generation Tutorial](#). For an example of how to use the sine generator to modulate a pulse sequence from the AWG, see the [Digital Modulation Tutorial](#).

The gain settings in the Waveform Generators and the Sine Generators sections are graphically grouped in pairs, and each pair is associated with an I or Q input to the DAC. For the Sine Generators section, the I and Q pairs are further separated into Sin and Cos terms. In the Waveform Generators section, the I and Q pairs of gain settings. In both cases, the default settings are chosen to generate an upper sideband signal when using a positive oscillator frequency. For a more detailed explanation of how these gain settings are used in generating signals, see the [Digital Modulation Tutorial](#).

Functional Elements

Table 5.33: MOD tab

Control/ Tool	Option/ Range	Description
SG Channels		Select SG Channel to display corresponding set of oscillator frequencies.
Frequency (Hz)		Oscillator frequency.
Oscillator Select		Selection of the oscillator used for the generated sine signal.
Harmonic		Multiplies the oscillator's reference frequency with the integer factor defined by this field.
Frequency (Hz)		Frequency of the selected oscillator.
Phase Shift (deg)		Sets the phase of the sine signal.
I Sin Amplitude		Sets the amplitude of the sine signal sent to the I input of the digital mixer.
I Cos Amplitude		Sets the amplitude of the cosine signal sent to the I input of the digital mixer.
I Enable	ON / OFF	Enables the I input of the digital mixer.
Q Sin Amplitude		Sets the amplitude of the sine signal sent to the Q input of the digital mixer.
Q Cos Amplitude		Sets the amplitude of the cosine signal sent to the Q input of the digital mixer.
Q Enable	ON / OFF	Enables the Q input of the digital mixer.
Run/Stop		Runs the AWG sequencer.
Sequencer Status	grey/ green/red	Displays the status of the sequencer on the instrument. Off: Ready, not running. Green: Running, not waiting for any trigger event. Yellow: Running, waiting for a trigger event. Red: Not ready (e.g., pending elf download, no elf download).
Modulation Enable	ON / OFF	Enables digital modulation of the waveforms generated by the AWG.
AWG Output Amplitude		Sets the amplitude of the AWG output.
Hold	ON / OFF	Keep the last sample (constant) on the outputs even after the waveform program finishes. It is recommended to use only AWG waveforms with lengths equal to a multiple of 16 together with this functionality. Waveforms with other lengths are automatically padded with zeros at the end by the AWG compiler. The status of the hold node is checked only when the AWG is enabled. If hold is disabled after enabling the AWG or when the AWG is not running, AWG output values will still be held.
AWG Output Gain Amplitude		Sets the amplitude scaling factor of the given AWG channel. The amplitude is a dimensionless scaling factor applied to the digital signal.
AWG Output Gain Enable	ON / OFF	Indicates the routing of the AWG signal (row) to the digital mixer inputs (column).

5.2.6. AWG Tab

The AWG tab is available on all SHFQC+ Qubit Controller instruments.

Features


- 4- or 8-channel arbitrary waveform generator
- 98 kSa waveform memory per channel
- Sequence branching

- Digital modulation
- Cross-domain trigger engine
- Sequence Editor with code highlighting and auto completion
- High-level programming language with waveform generation and editing toolset
- Waveform viewer

Description

The AWG tab gives access to the arbitrary waveform generator functionality. Whenever the tab is closed or an additional one of the same type is needed, clicking the following icon will open a new instance of the tab.

Table 5.34: App icon and short description

Control/ Tool	Option/ Range	Description
AWG		Generate arbitrary signals using sequencing and sample-by-sample definition of waveforms.

The AWG tab (see [Figure 5.30](#)) consists of a settings section on the right side and the Sequence and Waveform Viewer sub-tabs on the left side. The settings section is further divided into Control, Waveform, Trigger, and Advanced sub-tabs. The **Sequence** sub-tab is used for displaying, editing and compiling a LabOne sequence program. The sequence program defines which waveforms are played and in which order. The Sequence Editor is the main tool for operating the AWG.

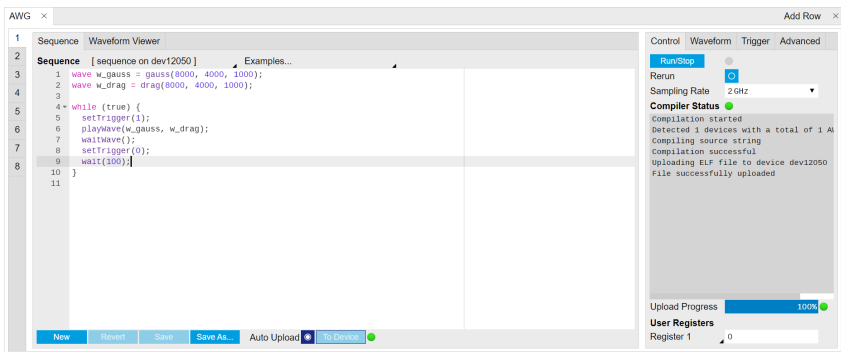


Figure 5.30: LabOne UI: AWG tab

A number of sequence programming examples are available through a drop-down menu at the top of the Sequence Editor, and additional ones can be found in [Tutorials](#). The LabOne sequence programming language is specified in detail in [LabOne Sequence Programming](#). The language comes with a number of predefined waveforms, such as Gaussian, Blackman, sine, or square functions. By combining those predefined waveforms using the waveform editing tools (add, multiply, cut, concatenate, etc), signals with a high level of complexity can be generated directly from the Sequence Editor window. Sample-by-sample definition of the output signal is possible by using comma-separated value (CSV) files specified by the user.

The AWG features a compiler which translates the high-level sequence program into machine instructions and waveform data to be stored in the instrument memory as shown in [Figure 5.31](#). The sequence program is written using high-level control structures and syntax that are inspired by human language, whereas machine instructions reflect exactly what happens on the hardware level. Writing the sequence program using a high-level language represents a more natural and efficient way of working in comparison to writing lists of machine instructions, which is the traditional way of programming AWGs. Concretely, the improvements rely on features such as:

- combination of waveform generation, editing, and playback sequence in a single script
- easily readable syntax and naming for run-time variables and constants
- optimized waveform memory management, reduced transfers upon waveform changes
- definition of user functions and procedures for advanced structuring
- syntax validation

By design, there is no one-to-one link between the list of statements in the high-level language and the list of instructions executed by the Sequencer. In order to understand the execution timing, it's helpful to consider the internal architecture of the AWG, consisting of the Sequencer itself, the Waveform Player, and the Waveform Memory.

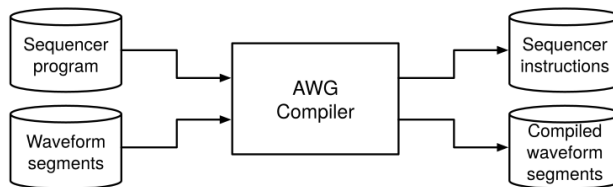


Figure 5.31: AWG sequence program compilation process

The **Sequence Editor** provides the editing, compilation, and transfer functionality for sequence programs. A program typed into the Editor is compiled upon clicking **Save**. If the compilation is successful and Automatic Upload is enabled, the program including all necessary waveform data is transferred to the device. If the compilation fails, the Status field will display debug messages. Clicking on **Save as...** allows you to choose a new name for the program. The name of the program that is currently edited is displayed at the top of the editor. External program files as well as waveform data files can be transferred to the right location easily using the file drag-and-drop zone in the **Config Tab** so they become accessible from the user interface. The files can be managed in the **File Manager Tab** and their location in the directory structure is shown in Table 5.35. The program name is displayed in a drop-down box. The box allows quick access to all programs in the standard sequence program location. It is possible to quickly switch between programs using the box. Changes made in one program will be preserved when switching to a different program. The file name of a program will be postfixed by an asterisk in case there are unsaved changes in the source file. Note that switching programs in the editor is not sufficient to also update the program in the instrument. In order to send a newly selected program to the instrument, the **To Device** button must be clicked.

Table 5.35: Sequence program and waveform file location

File type	Location
Waveform files (Windows)	C:\Users\ <user name="">\Documents\Zurich Instruments\LabOne\WebServer\awg\waves</user>
Sequence programs (Windows)	C:\Users\ <user name="">\Documents\Zurich Instruments\LabOne\WebServer\awg\src</user>
Waveform files (Linux)	~/Zurich Instruments/LabOne/WebServer/awg/waves
Sequence programs (Linux)	~/Zurich Instruments/LabOne/WebServer/awg/src

In the **Control sub-tab** the user configures signal parameters and controls the execution of the AWG. The AWG can be started in by clicking on **Start**. When enabling the Rerun button, the Sequencer will be restarted automatically when its program completes. The continuous mode is a simple way to create an infinite loop, but it results in a considerable timing jitter. To avoid this jitter, it is recommended to specify infinite loops directly in the sequence program.

The Sampling Rate field is used to control the default playback sampling rate of the AWG. The sampling rate is dynamic, i.e., can be specified for each waveform by using an optional argument in the waveform playback instructions in the sequence program. This allows for considerably reducing waveform upload time for signals that contain both fast and slow components.

The **Waveform sub-tab** displays information about the waveforms that are used by the current sequence program, such as their length and channel number. Together with the **Waveform viewer sub-tab**, it is a useful tool to visualize the waveforms used in the sequence program.

On the **Trigger sub-tab** you can configure the trigger inputs of the AWG. Each AWG core has two internal trigger input channels which can be configured to probe any of the Trig inputs on the instrument front panel. The **Advanced sub-tab** displays the compiled list of sequencer instructions and the current state of the sequencer on the instrument. This can help an advanced user in debugging a sequence program and understanding its execution.

Sequence Editor Keyboard Shortcuts

The tables below list a number of helpful keyboard shortcuts that are applicable in the LabOne Sequence Editor.

Table 5.36: Line Operations

Shortcut	Action
Ctrl+D	Remove line
Alt+Shift+Down	Copy lines down
Alt+Shift+Up	Copy lines up
Alt+Down	Move lines down
Alt+Up	Move lines up
Alt+Del	Remove to line end
Alt+Backspace	Remove to line start
Ctrl+Backspace	Remove word left
Ctrl+Del	Remove word right

Table 5.37: Selection

Shortcut	Action
Ctrl+A	Select all
Shift+Left	Select left
Shift+Right	Select right
Ctrl+Shift+Left	Select word left
Ctrl+Shift+Right	Select word right
Shift+Home	Select line start
Shift+End	Select line end
Alt+Shift+Right	Select to line end
Alt+Shift+Left	Select to line start
Shift+Up	Select up
Shift+Down	Select down
Shift+Page Up	Select page up
Shift+Page Down	Select page down
Ctrl+Shift+Home	Select to start
Ctrl+Shift+End	Select to end
Ctrl+Shift+D	Duplicate selection
Ctrl+Shift+P	Select to matching bracket

Table 5.38: Go to

Shortcut	Action
Left	Go to left
Right	Go to right
Ctrl+Left	Go to word left
Ctrl+Right	Go to word right
Up	Go line up
Down	Go line down
Alt+Left, Home	Go to line start
Alt+Right, End	Go to line end

Shortcut	Action
Page Up	Go to page up
Page Down	Go to page down
Ctrl+Home	Go to start
Ctrl+End	Go to end
Ctrl+L	Go to line
Ctrl+Down	Scroll line down
Ctrl+Up	Scroll line up
Ctrl+P	Go to matching bracket

Table 5.39: Find/Replace

Shortcut	Action
Ctrl+F	Find
Ctrl+H	Replace
Ctrl+K	Find next
Ctrl+Shift+K	Find previous

Table 5.40: Folding

Shortcut	Action
Alt+L	Fold selection
Alt+Shift+L	Unfold

Table 5.41: Other

Shortcut	Action
Tab	Indent
Shift+Tab	Outdent
Ctrl+Z	Undo
Ctrl+Shift+Z, Ctrl+Y	Redo
Ctrl+/**	Toggle comment
Ctrl+Shift+U	Change to lower case
Ctrl+U	Change to upper case
Ins	Overwrite
Ctrl+Shift+E	Macros replay
Ctrl+Alt+E	Macros recording
Del	Delete

LabOne Sequence Programming

A Simple Example

The syntax of the LabOne AWG Sequencer programming language is based on C, but with a few simplifications. Each statement is concluded with a semicolon, several statements can be grouped

with curly brackets, and comment lines are identified with a double slash. The following example shows some of the fundamental functionalities: waveform generation, repeated playback, triggering, and single/dual-channel waveform playback. See [Tutorials](#) for a step-by-step introduction with more examples.

```
// Define an integer constant
const N = 4096;
// Create two Gaussian pulses with length N points,
// amplitude +1.0 (-1.0), center at N/2, and a width of N/8
wave gauss_pos = 1.0*gauss(N, N/2, N/8);
wave gauss_neg = -1.0*gauss(N, N/2, N/8);
// execute playback sequence 100 times
repeat (100) {
    // Play pulse on AWG channel 1
    playWave(gauss_pos);
    // Play pulses simultaneously on both AWG channels
    playWave(gauss_pos, gauss_neg);
}
```

Keywords and Comments

The following table lists the keywords used in the LabOne AWG Sequencer language.

Table 5.42: Programming keywords

Keyword	Description
const	Constant declaration
var	Integer variable declaration
cvar	Compile-time variable declaration
string	Constant string declaration
true	Boolean true constant
false	Boolean false constant
for	For-loop declaration
while	While-loop declaration
repeat	Repeat-loop declaration
if	If-statement
else	Else-part of an if-statement
switch	Switch-statement
case	Case-statement within a switch
default	Default-statement within a switch
return	Return from function or procedure, optionally with a return value

The following code example shows how to use comments.

```
const a = 10; // This is a line comment. Everything between the double
              // slash and the end of the line will be ignored.

/* This is a block comment. Everything between the start-of-block-comment
and end-of-block-comment markers is ignored.

For example, the following statement will be ignored by the compiler.
const b = 100;
*/
```

Constants and Variables

Constants may be used to make the program more readable. They may be of integer or floating-point type. It must be possible for the compiler to compute the value of a constant at compile time, i.e., on the host computer. Constants are declared using the **const** keyword.

Compile-time variables may be used in computations and loop iterations during compile time, e.g. to create large numbers of waveforms in a loop. They may be of integer or floating-point type. They are used in a similar way as constants, except that they can change their value during compile time operations. Compile-time variables are declared using the **cvar** keyword.

Variables may be used for making simple computations during run time, i.e., on the instrument. The Sequencer supports integer variables, addition, and subtraction. Not supported are floating-point variables, multiplication, and division. Typical uses of variables are to step waiting times or to tag digital measurement data with a numerical identifier. Variables are declared using the **var** keyword.

The following code example shows how to use variables.

```
var b = 100; // Create and initialize a variable

// Repeat the following block of statements 100 times
repeat (100) {
  b = b + 1; // Increment b
  wait(b);   // Wait 'b' cycles
}
```

The following table shows the predefined constants. These constants are intended to be used as arguments in certain run-time evaluated functions that encode device parameters with integer numbers. For example, the AWG Sampling Rate is specified as an integer exponent n in the expression $(\text{baseSamplingClock})/2^n$. The AWG rates constants are specified for the sampling clock of 2.0 GHz of the SHFQC+.

Constants whose value is marked as "opaque" are meant to always be used instead of their numerical value.

Table 5.43: Predefined Constants

Name	Value	Description
AWG_RATE_2000MHZ	0	Constant to set Sampling Rate to 2.0 GHz.
AWG_RATE_1000MHZ	1	Constant to set Sampling Rate to 1.0 GHz.
AWG_RATE_500MHZ	2	Constant to set Sampling Rate to 500 MHz.
AWG_RATE_250MHZ	3	Constant to set Sampling Rate to 250 MHz.
AWG_RATE_125MHZ	4	Constant to set Sampling Rate to 125 MHz.
AWG_RATE_62P5MHZ	5	Constant to set Sampling Rate to 62.5 MHz.
AWG_RATE_31P25MHZ	6	Constant to set Sampling Rate to 31.25 MHz.
AWG_RATE_15P63MHZ	7	Constant to set Sampling Rate to 15.63 MHz.
AWG_RATE_7P81MHZ	8	Constant to set Sampling Rate to 7.81 MHz.
AWG_RATE_3P9MHZ	9	Constant to set Sampling Rate to 3.9 MHz.
AWG_RATE_1P95MHZ	10	Constant to set Sampling Rate to 1.95 MHz.
AWG_RATE_976KHZ	11	Constant to set Sampling Rate to 976 kHz.
AWG_RATE_488KHZ	12	Constant to set Sampling Rate to 488 kHz.
AWG_RATE_244KHZ	13	Constant to set Sampling Rate to 244 kHz.
DEVICE_SAMPLE_RATE	<actual device sample rate>	

Name	Value	Description
ZSYNC_DATA_RAW	opaque	Constant to use as argument to getFeedback or executeTableEntry. Respectively, returns the last ZSync message received as-is without processing or execute the command table entry with index equal to the last raw ZSync message.
ZSYNC_DATA_PROCESSED_A	opaque	Constant to use as argument to configureFeedbackProcessing, getFeedback or executeTableEntry. Respectively, configure the processing of ZSync messages, returns the last ZSync message received with processing or execute the command table entry with index equal the last ZSync message received with processing.
ZSYNC_DATA_PROCESSED_B	opaque	Constant to use as argument to configureFeedbackProcessing, getFeedback or executeTableEntry. Respectively, configure the processing of ZSync messages, returns the last ZSync message received with processing or execute the command table entry with index equal the last ZSync message received with processing.
QA_DATA_RAW	opaque	Constant to use as argument to getFeedback. Returns the last readout data received from the QA channel as-is.
QA_DATA_PROCESSED	opaque	Constant to use as argument to configureFeedbackProcessing, getFeedback or executeTableEntry. Respectively, configure the processing of readout data received from the QA channel, returns the last readout data received from the QA channel with processing or execute the command table entry with index equal the last readout data received from the QA channel with processing.
AWG_CHAN1	1	Constant to select channel 1.
AWG_CHAN2	2	Constant to select channel 2.
AWG_MARKER1	1	Constant to select marker 1.
AWG_MARKER2	2	Constant to select marker 2.
AWG_OSC_PHASE_START	1	Constant to trigger the oscillator phase on the positive edge.
AWG_OSC_PHASE_MIDDLE	0	Constant to trigger the oscillator phase on the negative edge.

Numbers can be expressed using either of the following formatting.

```
const a = 10;           // Integer notation
const b = -10;          // Negative number
const h = 0xdeadbeef;   // Hexadecimal integer
const bin = 0b10101;    // Binary integer
const f = 0.1e-3;       // Floating point number.
const not_float = 10e3; // Not a floating point number
```

Booleans are specified with the keywords **true** and **false**. Furthermore, all numbers that evaluate to a nonzero value are considered true. All numbers that evaluate to zero are considered false.

Strings are delimited using "" and are interpreted as constants. Strings may be concatenated using the + operator.

```
string AWG_PATH = "awgs/0/";
string AWG_GAIN_PATH = AWG_PATH + "gains/0/";
```

Waveform Generation and Editing

The following table contains the definition of functions for waveform generation.

wave zeros(const samples)

Constant amplitude of 0 over the defined number of samples.

$$h(x) = 0$$

Args:

- **samples**: Number of samples in the waveform

Returns:

resulting waveform

wave ones(const samples)

Constant amplitude of 1 over the defined number of samples.

$$h(x) = 1$$

Args:

- **samples**: Number of samples in the waveform

Returns:

resulting waveform

wave sine(const samples, const amplitude=1.0, const phaseOffset, const nrOfPeriods)

Sine function with arbitrary amplitude (a), phase offset in radians (p), number of periods (f) and number of samples (N).

$$h(x) = a \cdot \sin(2\pi f \frac{x}{N} + p)$$

Args:

- **amplitude**: Amplitude of the signal (optional)
- **nrOfPeriods**: Number of Periods within the defined number of samples
- **phaseOffset**: Phase offset of the signal in radians
- **samples**: Number of samples in the waveform

Returns:

resulting waveform

wave cosine(const samples, const amplitude=1.0, const phaseOffset, const nrOfPeriods)

Cosine function with arbitrary amplitude (a), phase offset in radians (p), number of periods (f) and number of samples (N).

$$h(x) = a \cdot \cos(2\pi f \frac{x}{N} + p)$$

Args:

- **amplitude**: Amplitude of the signal (optional)
- **nrOfPeriods**: Number of Periods within the defined number of samples
- **phaseOffset**: Phase offset of the signal in radians
- **samples**: Number of samples in the waveform

Returns:

resulting waveform

wave sinc(const samples, const amplitude=1.0, const position, const beta)

Normalized sinc function with control of peak position (p), amplitude (a), width (\beta) and number of samples (N).

$$h(x) = \begin{cases} a & \text{if } x = p \\ a \cdot \frac{\sin(2\pi \cdot \beta \cdot \frac{x-p}{N})}{2\pi \cdot \beta \cdot \frac{x-p}{N}} & \text{else} \end{cases}$$

Args:

- **amplitude**: Amplitude of the signal (optional)
- **beta**: Width of the function
- **position**: Peak position of the function
- **samples**: Number of samples in the waveform

Returns:

resulting waveform

wave ramp(const samples, const startLevel, const endLevel)

Linear ramp from the start (s) to the end level (e) over the number of samples (N).

$$h(x) = s + \frac{x(e - s)}{N - 1}$$

Args:

- **endLevel**: level at the last sample of the waveform
- **samples**: Number of samples in the waveform
- **startLevel**: level at the first sample of the waveform

Returns:

resulting waveform

wave sawtooth(const samples, const amplitude=1.0, const phaseOffset, const nrOfPeriods)

Sawtooth function with arbitrary amplitude, phase in radians and number of periods.

Args:

- **amplitude**: Amplitude of the signal
- **nrOfPeriods**: Number of Periods within the defined number of samples
- **phaseOffset**: Phase offset of the signal in radians
- **samples**: Number of samples in the waveform

Returns:

resulting waveform

wave triangle(const samples, const amplitude=1.0, const phaseOffset, const nrOfPeriods)

Triangle function with arbitrary amplitude, phase in radians and number of periods.

Args:

- **amplitude**: Amplitude of the signal
- **nrOfPeriods**: Number of Periods within the defined number of samples
- **phaseOffset**: Phase offset of the signal in radians
- **samples**: Number of samples in the waveform

Returns:

resulting waveform

wave gauss(const samples, const amplitude=1.0, const position, const width)

Gaussian pulse with arbitrary amplitude (a), position (p), width (w) and number of samples (N).

$$h(x) = a \cdot e^{-\frac{(x-p)^2}{2 \cdot w^2}}$$

Args:

- **amplitude**: Amplitude of the signal (optional)
- **position**: Peak position of the pulse
- **samples**: Number of samples in the waveform
- **width**: Width of the pulse

Returns:

resulting waveform

wave drag(const samples, const amplitude=1.0, const position, const width)

Derivative of Gaussian pulse with arbitrary amplitude (a), position (p), width (w) and number of samples (N) normalized to a maximum amplitude of 1.

$$h(x) = a \cdot \frac{\sqrt{e}(p - x)}{w} \cdot e^{-\frac{(x-p)^2}{2 \cdot w^2}}$$

Args:

- **amplitude**: Amplitude of the signal (optional)
- **position**: Center point position of the pulse
- **samples**: Number of samples in the waveform
- **width**: Width of the pulse

Returns:

resulting waveform

wave blackman(const samples, const amplitude=1.0, const alpha)

Blackman window function with arbitrary amplitude (a), alpha parameter and number of samples (N).

$$h(x) = a \cdot (\alpha_0 - \alpha_1 \cos(\frac{2\pi x}{N-1}) + \alpha_2 \cos(\frac{4\pi x}{N-1}))$$

$$\alpha_0 = \frac{1-\alpha}{2}; \quad \alpha_1 = \frac{1}{2}; \quad \alpha_2 = \frac{\alpha}{2};$$

Args:

- **alpha**: Width of the function
- **amplitude**: Amplitude of the signal (optional)
- **samples**: Number of samples in the waveform

Returns:

resulting waveform

wave hamming(const samples, const amplitude=1.0)

Hamming window function with arbitrary amplitude (a) and number of samples (N).

$$h(x) = a \cdot (\alpha - \beta \cos(\frac{2\pi x}{N-1}))$$

with $\alpha = 0.54$ and $\beta = 0.46$

Args:

- **amplitude**: Amplitude of the signal (optional)
- **samples**: Number of samples in the waveform

Returns:

resulting waveform

wave hann(const samples, const amplitude=1.0)

Hann window function with arbitrary amplitude (a) and number of samples (N).

$$h(x) = a \cdot 0.5 \cdot (1 - \cos(\frac{2\pi x}{N-1}))$$

Args:

- **amplitude**: Amplitude of the signal
- **samples**: Number of samples in the waveform

Returns:

resulting waveform

wave rect(const samples, const amplitude)

Rectangle function, constants amplitude (a) over the defined number of samples.

$$h(x) = a$$

Args:

- **amplitude**: Amplitude of the signal
- **samples**: Number of samples in the waveform

Returns:

resulting waveform

wave marker(const samples, const markerValue)

Generate a waveform with marker bits set to the specified value. The analog part of the waveform is zero.

Args:

- **markerValue:** Value of the marker bits
- **samples:** Number of samples in the waveform

Returns:

resulting waveform

wave rand(const samples, const amplitude=1.0, const mean, const stdDev)

White noise with arbitrary amplitude, power and standard deviation.

Args:

- **amplitude:** Amplitude of the signal
- **mean:** Average signal level
- **samples:** Number of samples in the waveform
- **stdDev:** Standard deviation of the noise signal

Returns:

resulting waveform

wave randomGauss(const samples, const amplitude=1.0, const mean, const stdDev)

White noise with arbitrary amplitude, power and standard deviation.

Args:

- **amplitude:** Amplitude of the signal
- **mean:** Average signal level
- **samples:** Number of samples in the waveform
- **stdDev:** Standard deviation of the noise signal

Returns:

resulting waveform

wave randomUniform(const samples, const amplitude=1.0)

Random waveform with uniform distribution.

Args:

- **amplitude:** Amplitude of the signal
- **samples:** Number of samples in the waveform

Returns:

resulting waveform

wave lfsrGaloisMarker(const samples, const markerBit, const polynomial, const initial)

Generate a waveform with specified marker bit set to the Galois LFSR (linear-feedback shift register) generated sequence. The analog part of the waveform is zero. The LFSR characteristic polynomial is a member of the Galois Field of two elements and represented in binary form. See wikipedia entries for "Finite field arithmetic" and "Linear-feedback shift register (Galois LFSR)".

Args:

- **initial**: LFSR initial state, any nonzero value will work, usually 0x1
- **markerBit**: Marker bit to set (1 or 2)
- **polynomial**: LFSR characteristic polynomial in binary representation (max shift length 32), use 0x90000 for QRSS / PRBS-20
- **samples**: Number of samples in the waveform

Returns:

resulting waveform

wave chirp(const samples, const amplitude=1.0, const startFreq, const stopFreq, const phase=0)

Frequency chirp function with arbitrary amplitude, start and stop frequency, initial phase in radians and number of samples. Start and stop frequency are expressed in units of the AWG Sampling Rate. The amplitude can only be defined if the initial phase is defined as well.

Args:

- **amplitude**: Amplitude of the signal (optional)
- **phase**: Initial phase of the signal (optional)
- **samples**: Number of samples in the waveform
- **startFreq**: Start frequency of the signal
- **stopFreq**: Stop Frequency of the signal

Returns:

resulting waveform

wave rrc(const samples, const amplitude=1.0, const position, const beta, const width)

Root raised cosine function with arbitrary amplitude (a), position (p), roll-off factor (\beta) and width (w) and number of samples (N).

$$h(y) = a \frac{\sin(y\pi(1 - \beta)) + 4y\beta \cos(y\pi(1 + \beta))}{y\pi(1 - (4y\beta)^2)}$$

$$\text{with } y(x) = 2w \frac{x - p}{N}$$

Args:

- **amplitude**: Amplitude of the signal
- **beta**: Roll-off factor
- **position**: Center point position of the pulse
- **samples**: Number of samples in the waveform
- **width**: Width of the pulse

Returns:

Resulting waveform

wave vect(const value,...)

Specify a waveform sample by sample. Each sample is defined by one of an arbitrary number of input arguments. Only recommended for short waveforms that consist of less than 100 samples. Larger waveforms may be defined in a CSV file.

Args:

- **value:** Waveform amplitude at the respective sample

Returns:

resulting waveform

wave placeholder(const samples, const marker0=false, const marker1=false)

Creates space for a single-channel waveform, optionally with markers, without actually generating any waveform data when compiling the sequence program. Actual waveform data needs to be uploaded separately via the "<dev>/AWGS/<n>/WAVEFORM/WAVES/<index>" API nodes after the sequence compilation and upload. The waveform index can be explicitly assigned to the generated placeholder wave using the assignWaveIndex instruction.

Args:

- **marker0:** true if marker bit 0 must be used (default false)
- **marker1:** true if marker bit 1 must be used (default false)
- **samples:** Number of samples in the waveform

Returns:

waveform object

The following table contains the definition of functions for waveform editing.

wave join(wave wave1, wave wave2, const interpollLength=0)

Connect two or more waveforms with optional linear interpolation between the waveforms.

Args:

- **interpollLength:** Number of samples to interpolate between waveforms (optional, default 0)
- **wave1:** Input waveform
- **wave2:** Input waveform

Returns:

joined waveform

wave join(wave wave1, wave wave2,...)

Connect two or more waveforms.

Args:

- **wave1:** Input waveform
- **wave2:** Input waveform

Returns:

joined waveform

wave interleave(wave wave1, wave wave2,...)

Interleave two or more waveforms sample by sample.

Args:

- **wave1:** Input waveform
- **wave2:** Input waveform

Returns:

interleaved waveform

wave add(wave wave1, wave wave2,...)

Add two or more waveforms sample by sample. Alternatively, the "+" operator may be used for waveform addition.

Args:

- **wave1:** Input waveform
- **wave2:** Input waveform

Returns:

sum waveform

wave multiply(wave wave1, wave wave2,...)

Multiply two or more waveforms sample by sample. Alternatively, the "*" operator may be used for waveform multiplication.

Args:

- **wave1:** Input waveform
- **wave2:** Input waveform

Returns:

product waveform

wave scale(wave waveform, const factor)

Scale the input waveform with the factor and return the scaled waveform. The input waveform remains unchanged.

Args:

- **factor:** Scaling factor
- **waveform:** Input waveform

Returns:

scaled waveform

wave flip(wave waveform)

Flip the input waveform back to front and return the flipped waveform. The input waveform remains unchanged.

Args:

- **waveform:** Input waveform

Returns:

flipped waveform

wave cut(wave waveform, const from, const to)

Cuts a segment out of the input waveform and returns it. The input waveform remains unchanged. The segment is flipped in case that "from" is larger than "to".

Args:

- **from**: First sample of the cut waveform
- **to**: Last sample of the cut waveform
- **waveform**: Input waveform

Returns:

cut waveform

wave filter(wave b, wave a, wave x)

Filter generates a rational transfer function with the waveforms a and b as numerator and denominator coefficients. The transfer function is normalized by the first element of a, which has to be non-zero. The filter is applied to the input waveform x and returns the filtered waveform.

$$y(n) = \frac{1}{a_0} \left(\sum_{i=0}^M b_i x_{n-i} - \sum_{i=1}^N a_i y_{n-i} \right)$$

with $M = \text{length}(b) - 1$
and $N = \text{length}(a) - 1$

Args:

- **a**: Denominator coefficients
- **b**: Numerator coefficients
- **x**: Input waveform

Returns:

filtered waveform

wave circshift(wave a, const n)

Circularly shifts a 1D waveform and returns it.

Args:

- **n**: Number of elements to shift
- **waveform**: Input waveform

Returns:

circularly shifted waveform

Waveform Playback and Predefined Functions

The following table contains the definition of functions for waveform playback and other purposes.

void setDIO(var value)

Writes the value as a 32-bit value to the DIO bus.

The value can be either a const or a var value. Configure the Mode setting in the DIO tab when using this command. The DIO interface speed of 50 MHz limits the rate at which the DIO output value is updated.

Args:

- **value**: The value to write to the DIO (const or var)

var getDIO()

Reads a 32-bit value from the DIO bus.

Returns:

var containing the read value

var getDIOTriggered()

Reads a 32-bit value from the DIO bus as recorded at the last DIO trigger position.

Returns:

var containing the read value

void setTrigger(var value)

Sets the AWG Trigger output signals.

The state of all four AWG Trigger output signals is represented by the bits in the binary representation of the integer value. Binary notation of the form 0b0000 is recommended for readability.

Args:

- ─ **value**: to be written to the trigger output lines

void wait(var cycles)

Waits for the given number of Sequencer clock cycles (4 ns per cycle). The execution of the instruction adds an offset of 2 clock cycles, i.e., the statement wait(3) leads to a waiting time of $5 * 4 \text{ ns} = 20 \text{ ns}$.

Note: the minimum waiting time amounts to 3 cycles, which means that wait(0) and wait(1) will both result in a waiting time of $3 * 4 \text{ ns} = 12 \text{ ns}$.

Args:

- ─ **cycles**: number of cycles to wait

void waitDIOTrigger()

Waits until the DIO interface trigger is active. The trigger is specified by the Strobe Index and Strobe Slope settings in the AWG Sequencer tab.

var getDigTrigger(const index)

Gets the state of the indexed Digital Trigger input (1 or 2).

The physical signal connected to the AWG Digital Trigger input is to be configured in the Trigger sub-tab of the AWG tab.

Args:

- ─ **index**: index of the Digital Trigger input to be read; can be either 1 or 2

Returns:

trigger state, either 0 or 1

void error(string msg,...)

Throws the given error message when reached.

Args:

- ─ **msg**: Message to be displayed

void info(string msg,...)

Returns the specified message when reached.

Args:

- **msg**: Message to be displayed

void waitWave()

Waits until the AWG is done playing the current waveform.

void randomSeed()

Generate a new seed for the subsequent random vector commands.

void assignWaveIndex(const output, wave waveform, const index)**void assignWaveIndex(wave waveform, const index)****void playWave(const output, wave waveform, const rate=AWG_RATE_DEFAULT)**

Starts to play the given waveforms on the defined output channels. The playback begins as soon as the previous waveform playback is finished.

Args:

- **output**: defines on which output the following waveform is played
- **rate**: sample rate with which the AWG plays the waveforms (default set in the user interface).
- **waveform**: waveform to be played

void playWave(const output, wave waveform,...)

Starts to play the given waveforms on the defined output channels. It can contain multiple waveforms with an output definition. The playback begins as soon as the previous waveform playback is finished.

Args:

- **output**: defines on which output the following waveform is played
- **waveform**: waveform to be played

void playWave(wave waveform, const rate=AWG_RATE_DEFAULT)

Starts to play the given waveforms, output channels are assigned automatically depending on the number of input waveforms. The playback begins as soon as the previous waveform playback is finished.

Args:

- **rate**: sample rate with which the AWG plays the waveforms (default set in the user interface).
- **waveform**: waveform to be played

void playWave(wave waveform,...)

Starts to play the given waveforms, output channels are assigned automatically depending on the number of input waveforms. The playback begins as soon as the previous waveform playback is finished.

Args:

- **waveform**: waveform to be played

void setUserReg(const register, var value)

Writes a value to one of the User Registers (indexed 0 to 15).

The User Registers may be used for communicating information to the LabOne User Interface or a running API program.

Args:

- **register**: The register index (0 to 15) to be written to
- **value**: The integer value to be written

var getUserReg(const register)

Reads the value from one of the User Registers (indexed 0 to 15). The User Registers may be used for communicating information to the LabOne User Interface or a running API program.

Args:

- **register**: The register to be read (0 to 15)

Returns:

current register value

void playZero(var samples)

Starts to play zeros on all channels for the specified number of samples. Behaves as if same length all-zeros waveform is played using playWave, but without consuming waveform memory.

Args:

- **samples**: Number of samples to be played. The same min length and granularity applies as for regular waveforms.

void playZero(var samples, const rate)

Starts to play zeros on all channels for the specified number of samples. Behaves as if same length all-zeros waveform is played using playWave, but without consuming waveform memory.

Args:

- **rate**: Sample rate with which the AWG plays zeros (default set in the user interface).
- **samples**: Number of samples to be played. The same min length and granularity applies as for regular waveforms.

void waitDigTrigger(const index)

Waits for the reception of a trigger signal on the indexed Digital Trigger (index 1 or 2). The physical signals connected to the two AWG Digital Triggers are to be configured in the Trigger sub-tab of the AWG Sequencer tab. The Digital Triggers are configured separately for each AWG Core.

Args:

- **index**: Index of the digital trigger input; can be either 1 or 2.

void executeTableEntry(var index)

Execute the entry of the command table with the given index. An entry of the command table contains a waveform playback instruction as well as instructions for real-time setting of sine generators phases, oscillator select and output amplitude.

Args:

- **index**: table entry that shall be executed.

void setPRNGSeed(var value)

Sets the seed for the linear-shift feedback register lsfr of the pseudo random number generator (PRNG).

The seed is a 16 bit int32_t value. Zero is invalid as seed.

Args:

- **value:** seed value to be configured

var getPRNGValue()

Returns a random value from the pseudo-random number generator (PRNG). The PRNG is implemented as a Galois linear-shift feedback register according to the pseudo code below. The feedback register lsfr is initialized to a seed value using the function setPRNGSeed. The values lower and upper are set using the function setPRNGRange. The feedback register lsfr is stored from one call of the function getPRNGValue to the next, which renders the pseudo code recursive. In the pseudo code, XOR and AND are bitwise logical operators, and >> is the right bit shift operator. Pseudo code: $lsb = lsfr \text{ AND } 1$; $lsfr = lsfr \gg 1$; if $(lsb == 1)$ then: $lsfr = 0xb400 \text{ XOR } lsfr$; $rand = ((lsfr * (upper - lower + 1) \gg 16) + lower) \text{ AND } 0xffff$;

Returns:

Random value rand

void setPRNGRange(var lower, var upper)

Configures the range of the pseudo random number generator (PRNG) to generate output in range [lower, upper].

Args:

- **lower:** lower bound of range, 0 ... $2^{16}-1$
- **upper:** upper bound of range, 0 ... $2^{16}-1$

void setSinePhase(const phase)

Set the phase in units of degree of the sine generator of the AWG core in use. The phase is reset to 0 after execution of the sequence program.

Args:

- **phase:** Phase value [degree]

void incrementSinePhase(const phase)

Increment the phase in units of degree of the sine generator of the AWG core in use.

Args:

- **phase:** Phase value [degree]

void playHold(var samples)

Hold the last played value for the specified number of samples samples. Behaves as if same length constant waveform is played using playWave, but without consuming waveform memory.

Args:

- **samples:** Number of samples to be played. The same min length and granularity applies as for regular waveforms.

void playHold(var samples, const rate)

Hold the last played value for the specified number of samples. Behaves as if same length constant waveform is played using playWave, but without consuming waveform memory.

Args:

- **rate**: Sample rate with which the AWG plays zeros (default set in the user interface).
- **samples**: Number of samples to be played. The same min length and granularity applies as for regular waveforms.

void playWaveDIO()

Starts to play a waveform from the table defined by the command table. The waveform is selected according to the integer codeword currently read on the DIO interface. The codeword is specified by the Codeword Mask and Codeword Shift settings in the AWG Sequencer tab.

void waitSineOscPhase()

Waits until the oscillator phase of the sine generator reaches a zero crossing (negative -> positive, start of sine period) of I component.

void configFreqSweep(const oscillator_index, const freq_start, const freq_increment)

Configures a frequency sweep.

Args:

- **freq_increment**: Specify how much to increment the frequency for each step of the sweep [Hz]
- **freq_start**: Specify the start frequency value for the sweep [Hz]
- **oscillator_index**: Index of the oscillator that will be used for the sweep

void resetOscPhase(const mask)**void resetOscPhase()****void setSweepStep(const oscillator_index, var sweep_index)**

Executes a step within a frequency sweep.

Args:

- **oscillator_index**: Index of the oscillator that will be used for the sweep
- **sweep_index**: Sets the step index, from which the frequency is set

void setOscFreq(const oscillator_index, const freq)

Configures the frequency of an oscillator.

Args:

- **freq**: Frequency to be set [Hz]
- **oscillator_index**: Index of oscillator

var getFeedback(const data_type)

Read the last received feedback message. The argument specify which data the function should return.

Args:

- **data_type:** Specifies which data the function should return: ZSYNC_DATA_RAW: Returns the last ZSync message received as-is without processing. ZSYNC_DATA_PROCESSED_A: Returns the last ZSync message received with processing. ZSYNC_DATA_PROCESSED_B: Returns the last ZSync message received with processing. QA_DATA_RAW: Returns the last readout data received from the QA channel as-is. QA_DATA_PROCESSED: Returns the last readout data received from the QA channel with processing.

Returns:

var containing the read value

var getFeedback(const data_type, var wait_cycles)

Read the last received feedback message. The argument specify which data the function should return.

Args:

- **data_type:** Specifies which data the function should return: ZSYNC_DATA_RAW: Returns the last ZSync message received as-is without processing. ZSYNC_DATA_PROCESSED_A: Returns the last ZSync message received with processing. ZSYNC_DATA_PROCESSED_B: Returns the last ZSync message received with processing. QA_DATA_RAW: Returns the last readout data received from the QA channel as-is. QA_DATA_PROCESSED: Returns the last readout data received from the QA channel with processing.
- **wait_cycles:** Wait for the specified number of cycles after the most recent waitZSyncTrigger() or waitDigTrigger() instruction.

Returns:

var containing the read value

void waitZSyncTrigger()

Waits for a trigger over ZSync.

void resetRTLoggerTimestamp()

Reset the timestamp counter of the Real-Time Logger.

Expressions

Expressions may be used for making computations based on mathematical functions and operators. There are two kinds of expressions: those evaluated at compile time (the moment of clicking "Save" or "Save as..." in the user interface), and those evaluated at run time (after clicking "Run/Stop" or "Start"). Compile-time evaluated expressions only involve constants (**const**) or compile-time variables (**cvar**) and can be computed at compile time by the host computer. Such expressions can make use of standard mathematical functions and floating point arithmetic. Run-time evaluated expressions involve variables (**var**) and are evaluated by the Sequencer on the instrument. Due to the limited computational capabilities of the Sequencer, these expressions may only operate on integer numbers and there are less operators available than at compile time.

The following table contains the list of mathematical functions supported at compile time.

Table 5.44: Mathematical Functions

Function	Description
const abs(const c)	absolute value
const acos(const c)	inverse cosine

Function	Description
const acosh(const c)	hyperbolic inverse cosine
const asin(const c)	inverse sine
const asinh(const c)	hyperbolic inverse sine
const atan(const c)	inverse tangent
const atanh(const c)	hyperbolic inverse tangent
const cos(const c)	cosine
const cosh(const c)	hyperbolic cosine
const exp(const c)	exponential function
const ln(const c)	logarithm to base e (2.71828...)
const log(const c)	logarithm to the base 10
const log2(const c)	logarithm to the base 2
const log10(const c)	logarithm to the base 10
const sign(const c)	sign function -1 if x<0; 1 if x>0
const sin(const c)	sine
const sinh(const c)	hyperbolic sine
const sqrt(const c)	square root
const tan(const c)	tangent
const tanh(const c)	hyperbolic tangent
const ceil(const c)	smallest integer value not less than the argument
const round(const c)	round to nearest integer
const floor(const c)	largest integer value not greater than the argument
const avg(const c1, const c2,...)	mean value of all arguments
const max(const c1, const c2,...)	maximum of all arguments
const min(const c1, const c2,...)	minimum of all arguments
const pow(const base, const exp)	first argument raised to the power of second argument
const sum(const c1, const c2,...)	sum of all arguments

The following table contains the list of predefined mathematical constants. These can be used for convenience in compile-time evaluated expressions.

Table 5.45: Predefined Constants

Name	Value	Description
M_E	2.71828182845904523536028747135266250	e
M_LOG2E	1.44269504088896340735992468100189214	log ₂ (e)
M_LOG10E	0.434294481903251827651128918916605082	log ₁₀ (e)
M_LN2	0.693147180559945309417232121458176568	log _e (2)
M_LN10	2.30258509299404568401799145468436421	log _e (10)
M_PI	3.14159265358979323846264338327950288	pi
M_PI_2	1.57079632679489661923132169163975144	pi/2
M_PI_4	0.785398163397448309615660845819875721	pi/4
M_1_PI	0.318309886183790671537767526745028724	1/pi
M_2_PI	0.636619772367581343075535053490057448	2/pi
M_2_SQRTPI	1.12837916709551257389615890312154517	2/sqrt(pi)
M_SQRT2	1.41421356237309504880168872420969808	sqrt(2)

Name	Value	Description
M_SQRT1_2	0.707106781186547524400844362104849039	1/sqrt(2)

Table 5.46: Operators supported at compile time

Operator	Description	Priority
=	assignment	-1
+=, -=, *=, /=, %=, &=, =, <<=, >>=	assignment by sum, difference, product, quotient, remainder, AND, OR, left shift, and right shift	-1
	logical OR	1
&&	logical AND	2
	bit-wise logical OR	3
&	bit-wise logical AND	4
!=	not equal	5
==	equal	5
<=	less or equal	6
>=	greater or equal	6
>	greater than	6
<	less than	6
<<	arithmetic left bit shift	7
>>	arithmetic right bit shift	7
+	addition	8
-	subtraction	8
*	multiplication	9
/	division	9
~	bit-wise logical negation	10

Table 5.47: Operators supported at run time

Operator	Description	Priority
=	assignment	-1
+=, -=, *=, /=, %=, &=, =, <<=, >>=	assignment by sum, difference, product, quotient, remainder, AND, OR, left shift, and right shift	-1
	logical OR	1
&&	logical AND	2
	bit-wise logical OR	3
&	bit-wise logical AND	4
==	equal	5
!=	not equal	5
<=	less or equal	6
>=	greater or equal	6
>	greater than	6
<	less than	6
<<	left bit shift	7
>>	right bit shift	7

Operator	Description	Priority
+	addition	8
-	subtraction	8
~	bit-wise logical negation	9

Control Structures

Functions may be declared using the **var** keyword. **Procedures** may be declared using the **void** keyword. Functions must return a value, which should be specified using the **return** keyword. Procedures can not return values. Functions and procedures may be declared with an arbitrary number of arguments. The **return** keyword may also be used without arguments to return from an arbitrary point within the function or procedure. Functions and procedures may contain variable and constant declarations. These declarations are local to the scope of the function or procedure.

```
var function_name(argument1, argument2, ...) {
    // Statements to be executed as part of the function.
    return constant-or-variable;
}

void procedure_name(argument1, argument2, ...) {
    // Statements to be executed as part of the procedure.
    // Optional return statement
    return;
}
```

An **if-then-else** structure is used to create a conditional branching point in a sequencer program.

```
// If-then-else statement syntax
if (expression) {
    // Statements to execute if 'expression' evaluates to 'true'.
} else {
    // Statements to execute if 'expression' evaluates to 'false'.
}
```

```
// If-then-else statement short syntax
(expression)?(statement if true):(statement if false)
```

```
// If-then-else statement example
const REQUEST_BIT    = 0x0001;
const ACKNOWLEDGE_BIT = 0x0002;
const IDLE_BIT       = 0x8000;
var dio = getDIO();
if (dio & REQUEST_BIT) {
    dio = dio | ACKNOWLEDGE_BIT;
    setDIO(dio);
} else {
    dio = dio | IDLE_BIT;
    setDIO(dio);
}
```

A **switch-case** structure serves to define a conditional branching point similarly to the **if-then-else** statement, but is used to split the sequencer thread into more than two branches. Unlike the **if-then-else** structure, the switch statement is synchronous, which means that the execution time is the same for all branches and determined by the execution time of the longest branch. If no default case is provided and no case matches the condition, all cases will be skipped. The case arguments need to be of type **const**.

```
// Switch-case statement syntax
switch (expression) {
    case const-expression:
        expression;
```

```

...
default:
    expression;
}

// Switch-case statement example
switch (getDIO()) {
    case 0:
        playWave(gauss(1024,1.0,512,64));
    case 1:
        playWave(gauss(1024,1.0,512,128));
    case 2:
        playWave(drag(1024,1.0,512,64));
    default:
        playWave(drag(1024,1.0,512,128));
}

```

The **for loop** is used to iterate through a code block several times. The **initialization** statement is executed before the loop starts. The **end-expression** is evaluated at the start of each iteration and determines when the loop should stop. The loop is executed as long as this expression is true. The **iteration-expression** is executed at the end of each loop iteration.

Depending on how the **for** loop is set up, it can be either evaluated at compile time or at run time. Run-time evaluation is typically used to play series of waveforms. Compile-time evaluation is typically used for advanced waveform generation, e.g. to generate a series of waveforms with varying amplitude. For a run-time evaluated **for** loop, use the **var** data type as a loop index. To ensure that a loop is evaluated at compile time, use the **cvar** data type as a loop index. Furthermore, the compile-time **for** loop should only contain waveform generation/editing operations and it can't contain any variables of type **var**. The following code example shows both versions of the loop.

```

// For loop syntax
for (initialization; end-expression; iteration-expression) {
    // Statements to execute while end-expression evaluates to true
}

// FOR loop example to assemble a train of pulses into
// a single waveform (compile-time execution)
cvar gain_factor; // CVAR: integer or float values allowed
wave w_pulse_series;
for (gain_factor = 0; gain_factor < 1.0; gain_factor = gain_factor + 0.1) {
    w_pulse_series = join(w_pulse_series, gain_factor*gauss(1008, 504, 100));
}

// Playback of waveform defined using compile-time FOR loop
playWave(w_pulse_series);

// FOR loop example to vary waiting time between
// waveform playbacks (run-time execution)
var i; // VAR: integer values allowed
for (i = 0; i < 1000; i = i + 100) {
    playWave(gauss(1008, 504, 100));
    waitWave();
    wait(i);
}

```

The **while** loop is a simplified version of the **for** loop. The **end-expression** is evaluated at the start of each loop iteration. The contents of the loop are executed as long as this expression is true. Like the **for** loop, this loop comes in a compile-time version (if the end-expression involves only **cvar** and **const**) and in a run-time version (if the end-expression involves also **var** data types).

```

// While loop syntax
while (end-expression) {
    // Statements to execute while end-expression evaluates to true
}

```

```
// While loop example
const STOP_BIT = 0x8000;
var run = 1;
var i = 0;
var dio = 0;
while (run) {
    dio = getDIO();
    run = dio & STOP_BIT;

    dio = dio | (i & 0xff);
    setDIO(dio);
    i = i + 1;
}
```

The **repeat** loop is a simplified version of the **for** loop. It repeats the contents of the loop a fixed number of times. In contrast to the **for** loop, the repetition number of the **repeat** loop must be known at compile time, i.e., **const-expression** can only depend on constants and not on variables. Unlike the **for** and the **while** loop, this loop comes only in a run-time version. Thus, no **cvar** data types may be modified in the loop body.

```
// Repeat loop syntax
repeat (constant-expression) {
    // Statements to execute
}
```

```
// Repeat loop example
repeat (100) {
    setDIO(0x1);
    wait(10);
    setDIO(0x0);
    wait(10);
}
```

Usage of playZero and playHold commands

The functionalities of **playHold** and **playZero** are both available either through sequencer commands or through the command table. To use within a sequence, only the length in samples must be specified, as in **playHold(32)** or **playZero(128)**. The sequencer commands also accept a sampling rate as a second optional argument, which reduces the sampling rate only for the duration of the command. For example, **playZero(128, AWG_RATE_1000MHZ)** will play 128 samples of zeros at a sampling rate of 1.0 GSa/s, corresponding to 128 ns.

To use **playZero** or **playHold** within the command table, a command table entry must be made. See [Pulse-level Sequencing with the Command Table](#) for more information on using **playZero** within the command table. Similar syntax applies for using **playHold** within the command table. The table entries can be used within a sequence by adding the appropriate **executeTableEntry** command to the sequencer code.

Depending on the experiment being performed, it can make sense to use the **playZero** sequencer command in some cases and the command table version in other cases (and similarly for **playHold**). Generally speaking, the sequencer commands should be used when the length is variable, when the length is $2^{20} - 1$ or fewer samples, or when the optional sampling rate argument is used. When using a variable argument, such as when performing a sweep of the evolution time between two pulses with **playZero** or of the length of a pulse with **playHold**, the sequencer command must be used, as the **playZero** and **playHold** functionality within the command table cannot support variable arguments. A similar restriction applies to the optional sampling rate argument.

When the length is $2^{20} - 1$ or fewer samples, the sequencer commands map to a single assembly instruction. Once the length is more than or equal to 2^{20} samples, however, the sequencer commands map to at least two assembly instructions instead. Additionally, when using the optional sampling rate divider argument of the sequencer commands, **playZero** and **playHold** always map to at least three assembly instructions, regardless of the length in samples. When using the command table to perform **playZero** or **playHold** functionality, the corresponding **executeTableEntry** command always maps to a single assembly instruction, regardless of the length of the **playZero** or **playHold**, at the cost of using a command table entry.

Using Qubit Feedback Data in a Sequence

The AWG can make decisions depending on the feedback data received over ZSync or internal feedback. There are two primary ways to use the feedback data received: by using the command **getFeedback** and storing the result in a variable, or by using the feedback data directly as the argument of **executeTableEntry**. To directly make decisions about which pulse to play, it is recommended to use the feedback arguments of the **executeTableEntry**. For example, active reset in which the qubit data is passed to an SG Channel over a ZSync connection to a PQSC could involve a snippet of code like the following:

```
configureFeedbackProcessing(ZSYNC_DATA_PROCESSED_A, RESULT_INDEX, RESULT_SIZE,
OFFSET);
waitZSyncTrigger();
executeTableEntry(ZSYNC_DATA_PROCESSED_A, feedback_time);
```

The first instruction, **configureFeedbackProcessing** is used to configure the processing of the feedback data. The message will be reduced and an optional offset could be added. If the instruction is omitted, no trimming of the message will be done and the offset will be zero. Alternatively, the constant **ZSYNC_DATA_RAW** could be used in the following instructions, and no processing will be performed. The first argument of **executeTableEntry** determines which command table entry should be played, and the second argument accounts for the time between when the ZSync trigger is received and when the updated qubit readout data is available for use. The exact value of **feedback_time** (specified in number of sequencer clock cycles) depends on the combination of equipment being used as well as the experiment being performed and must be characterized by the user. For this example, the command table has been defined to play no pulse if the appropriate bit of the trimmed message is 0 or to play a pi-pulse if is 1:

```
## Qubit was in state 0
table[0].waveform.playZero = True
table[0].waveform.length = PI_PULSE_LENGTH

## Qubit was in state 1
table[1].waveform.index = 0
table[1].amplitude00.value = PI_AMPLITUDE
table[1].amplitude01.value = -PI_AMPLITUDE
table[1].amplitude10.value = PI_AMPLITUDE
table[1].amplitude11.value = PI_AMPLITUDE
```

In other cases, storing the results of **getFeedback** in a variable is the recommended route. For example, repeat until success requires repeated checking of the qubit readout data, but does not require a pulse to be played until the success criterion is met. Such an experiment might include sequencer code snippet like the following:

```
configureFeedbackProcessing(ZSYNC_DATA_PROCESSED_A, RESULT_INDEX, RESULT_SIZE,
OFFSET);
waitZSyncTrigger();

do {
    // preceding code
    failure = getFeedback(ZSYNC_DATA_PROCESSED_A, feedback_time); // check for
failure
    // following code
} while (failure)

// Success pulse
playWave(1, 2, w_success);
```

The success pulse is played only once the success condition has been met, and the type of pulse played does not directly depend on the feedback data received.

When testing a new sequence, it can also be useful to store the feedback message, as the value of the variable can be monitored by writing to a user register:

```
waitZSyncTrigger();
feedback_data = getFeedback(ZSYNC_DATA_RAW, feedback_time);
setUserReg(0, feedback_data);
```

The above code will write the feedback data available at **feedback_time** sequencer clock cycles after the ZSync trigger is received. The data is written to user register 0.


To use internal feedback data from the QA channel of the {dev_class_abbrev}, the terms **QA_DATA_RAW** and **QA_DATA_PROCESSED** can be used as arguments of the **getFeedback**. Alternatively, **QA_DATA_PROCESSED** can be used as an argument of the **executeTableEntry** commands. This functionality is analogous to the corresponding terms for qubit readout data received over ZSync (**ZSYNC_DATA_PROCESSED_A** and **ZSYNC_DATA_PROCESSED_B**).

Synchronizing Multiple AWG Cores

In many cases, using a common start trigger at an appropriate point in the sequence is enough to ensure that the start of the output signals of the AWG cores are aligned in time. In some cases, however, actions with non-deterministic timing can cause the AWG cores to become out of sync with each other. To ensure that the AWG cores start their waveform playback in sync, even in the presence of actions with non-deterministic timing, it is possible to enable a synchronization check between the different AWG cores. Each SG channel has its own synchronization node (**/device/sgchannels/[SG_CHAN_INDEX]/synchronization/enable**, see [Node Documentation](#)). If the synchronization node of a channel is enabled (i.e. set to 1), the AWG core of that channel will participate in a synchronization check, which can be useful when executing actions with a non-deterministic timing such as loading of new sequences. This check is performed after the prefetch step of a sequence (i.e. after the sequencer instruction data, command table data, and waveform data have been transferred from the external to the internal memory of the AWG module but before the first sequencer instruction is executed), and each participating AWG core will wait until all of the participants in the synchronization check have returned a “ready” status. It is possible to even synchronize an entire QCCS setup in this way by setting the node **/device/system/synchronization/source** to external (numerical value of 1). When the source is external, the instrument will report its ready status to the PQSC to which the instrument is connected, and the PQSC will wait for all instruments connected to it to report a ready status. If the source is set to internal (i.e. a numerical value of 0), then only the AWG cores on the instrument that have synchronization enabled participate in the synchronization check. Also note that if the source of the SHFQC+ is set to internal, the PQSC will not consider the instrument as participating in the synchronization check and will ignore the ready status of that instrument. Finally, to ensure that trigger generation is coordinated with the synchronization check of the AWG cores, it is possible to have the internal trigger unit participate in the synchronization check by setting the node **/device/system/internaltrigger/synchronization/enable** to **True** (i.e. to 1). A similar node is available on the PQSC. When the trigger synchronization is enabled, the same number of trigger events with the same holdoff time will be generated between subsequent synchronization checks.

Functional Elements

Table 5.48: AWG tab: Control sub-tab

Control/Tool	Option/Range	Description
Start		Runs the AWG.
Sampling Rate		AWG sampling rate. This value is used by default and can be overridden in the Sequence program. The numeric values are rounded for display purposes. The exact values are equal to the base sampling rate divided by 2^n , where n is an integer between 0 and 13.
Round oscillator frequencies.		Round oscillator frequencies to nearest commensurable with 225 MHz.
Status		Display compiler errors and warnings.
Compile Status	grey/green/yellow/red	Sequence program compilation status. Grey: No compilation started yet. Green: Compilation successful. Yellow: Compiler warnings (see status field). Red: Compilation failed (see status field).
Upload Progress	0% to 100%	The percentage of the sequencer program already uploaded to the device.
Upload Status	grey/yellow/green	Indicates the upload status of the compiled AWG sequence. Grey: Nothing has been uploaded. Yellow: Upload in progress. Green: Compiled sequence has been uploaded.

Control/Tool	Option/Range	Description
Register selector		Select the number of the user register value to be edited.
Register	0 to 2^{32}	Integer user register value. The sequencer has reading and writing access to the user register values during run time.
Input File		External source code file to be compiled.
Example File		Load pre-installed example sequence program.
New	New	Create a new sequence program.
Revert	Revert	Undo the changes made to the current program and go back to the contents of the original file.
Save (Ctrl+S)	Save	Compile and save the current program displayed in the Sequence Editor. Overwrites the original file.
Save as... (Ctrl+Shift+S)	Save as...	Compile and save the current program displayed in the Sequence Editor under a new name.
Automatic upload	ON / OFF	If enabled, the sequence program is automatically uploaded to the device after clicking Save and if the compilation was successful.
To Device	To Device	Sequence program will be compiled and, if the compilation was successful, uploaded to the device.
Multi-Device	ON / OFF	Compile the program for use with multiple devices. If enabled, the program will be compiled for and uploaded to the devices currently synchronized in the Multi-Device Sync tab.
Sync Status	grey/green/yellow	Sequence program synchronization status. Grey: No program loaded on device. Green: Program in sync with device. Yellow: Sequence program in editor differs from the one running on the device.

Table 5.49: AWG tab: Waveform sub-tab

Control/Tool	Option/Range	Description
Wave Selection		Select wave for display in the waveform viewer. If greyed out, the corresponding wave is too long for display.
Waveforms		Lists all waveforms used by the current sequence program.
Mem Usage (%)	0 to 100	Amount of the used waveform data relative to the device cache memory. The cache memory provides space for 64 kSa of waveform data per AWG core.

Table 5.50: AWG tab: Trigger sub-tab

Control/Tool	Option/Range	Description
Trigger State	grey/green	State of the Trigger. Grey: No trigger detected. Green: Trigger detected.
Slope		Select the signal edge that should activate the trigger. The trigger will be level sensitive when the Level option is selected.
Level (V)	numeric value	Defines the analog trigger level.
Auxiliary Trigger State	grey/green	State of the Auxiliary Trigger. Grey: No trigger detected. Green: Trigger detected.
Signal		Selects the digital trigger source signal.
DIO/Zsync Trigger state	grey/green	Indicates that triggers are generated from the DIO or ZSync interface to the AWG.
Read DIO/ZSync		Each AWG can be configured to either receive DIO data or ZSync data.

Control/Tool	Option/Range	Description
Valid Index	16 to 31	Selects the index n of the DIO interface bit (notation DIO[n] in the Specification chapter of the User Manual) to be used as a VALID signal input, i.e. a qualifier indicating that a valid codeword is available on the DIO interface.
Valid Polarity		Polarity of the VALID bit that indicates that a codeword is available on the DIO interface.
	None	VALID bit is ignored.
	Low	VALID bit must be logical low.
	High	VALID bit must be logical high.
	Both	VALID bit may be logical high or logical low.
Codeword Mask	0 to 1023	10-bit value to select the bits of the DIO interface input state (notation DIO[n] in the Specification chapter of the User Manual) to be used as a codeword in connection with the playWaveDIO sequencer instruction. The Codeword Mask is combined with the DIO interface input state by a bitwise AND operation after applying the Codeword Shift.
Codeword Shift	0 to 31	Defines the integer bit shift to be applied to the input state of the DIO interface (notation DIO[n] in the Specification chapter of the User Manual) to be used as a codeword for waveform selection in connection with the playWaveDIO sequencer instruction.
High bits		32-bit value indicating which bits on the DIO interface are detected as logic high.
Low bits		32-bit value indicating which bits on the DIO interface are detected as logic low.
Timing Error	grey/red	Indicates a timing error. A timing error is defined as an event where either the VALID or any of the data bits on the DIO interface change value at the same time as the STROBE bit.
Synchronization Enable		Enable multi-channel synchronization for this channel. The program will only execute once all channels with enabled synchronization are ready.

Table 5.51: AWG tab: Advanced sub-tab

Control/Tool	Option/Range	Description
Sequence Editor		Display and edit the sequence program.
Assembly	Text display	Displays the current sequence program in compiled form. Every line corresponds to one hardware instruction.
Counter		Current position in the list of sequence instructions during execution.
Status	Running, Idle, Waiting	Displays the status of the sequencer on the instrument. Off: Ready, not running. Green: Running, not waiting for any trigger event. Yellow: Running, waiting for a trigger event. Red: Not ready (e.g., pending elf download, no elf downloaded)
Rerun	ON / OFF	Reruns the Sequencer program continuously. This way of looping a program results in timing jitter. For a jitter free signal implement a loop directly in the sequence program.
Mem Usage (%)	0 to 100	Size of the current sequence program relative to the device cache memory. The cache memory provides space for a maximum of 16384 instructions.
Clear		Clears the command table description for the selected AWG Core.
Status	grey/green/red	Displays the status of the command table of the selected AWG Core. Grey: no table description uploaded, Green: table description successfully uploaded, Red: Error occurred during uploading of the table description.

5.2.7. Scope Tab

The Scope is a powerful time domain and frequency domain measurement tool as introduced in [Unique Set of Analysis Tools](#) and is available on all SHFQC+ Instruments.


Features

- Display complex signal in time domain
- 2 or 4 input channels with total memory of 64 kSa
- 14 bit nominal resolution
- Fast Fourier Transform (FFT) of complex signal: +/-500 MHz, spectral density and power conversion, choice of window functions
- Hardware Averaging up to 65536
- Segmented memory for up to 1024 scope shots
- Access internal triggers

Description

The Scope tab serves as the graphical display for time domain data. Whenever the tab is closed or an additional one of the same type is needed, clicking the following icon will open a new instance of the tab.

Table 5.52: App icon and short description

Control/Tool	Option/Range	Description
Scope		Displays shots of data samples in time and frequency domain (FFT) representation.

The Scope tab consists of a plot section on the left and a configuration section on the right. The configuration section is further divided into a number of sub-tabs. It gives access to a single-channel oscilloscope that can be used to monitor a choice of signals in the time or frequency domain. Hence the X axis of the plot area is time (for time domain display, [Figure 5.32](#)) or frequency (for frequency domain display, [Figure 5.33](#)). It is possible to display the time trace and the associated FFT simultaneously by opening a second instance of the Scope tab.

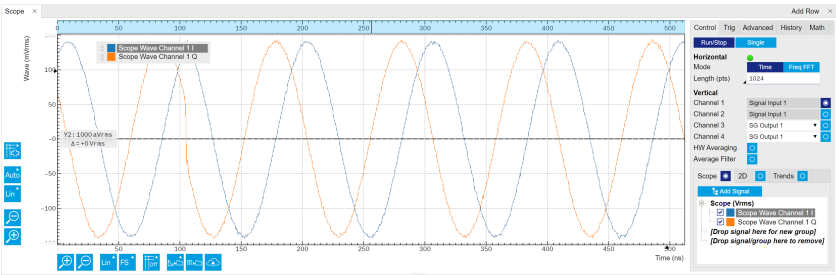


Figure 5.32: The overview Scope tab time domain of the GUI

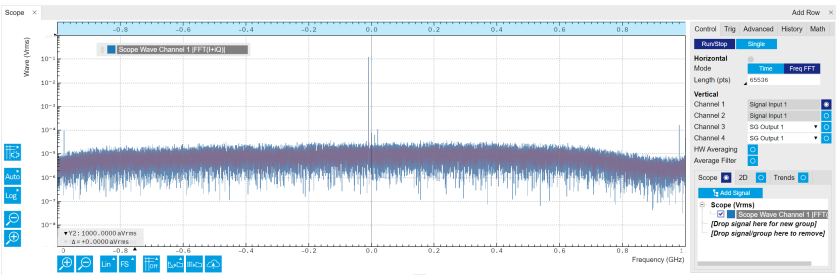


Figure 5.33: The overview Scope tab Freq FFT of the GUI

The raw complex data recorded by the Scope with a sampling rate of 2.0 GSa/s is from frequency down-conversion of the input signal, the Center Frequency is set in the [Inputs/Outputs Tab](#). In the time-domain, the complex data is decomposed into 2 traces representing the real and imaginary components, respectively. With LF path, the imaginary components of the raw data is 0. For detailed data processing, please refer to [Quantum Analyzer Setup Tab](#). The signal displayed in the FFT mode

is calculated as $|\text{FFT}(\mathbf{I} + \mathbf{iQ})|$, where \mathbf{I} is the real component of the complex data, and \mathbf{Q} is the imaginary component. The frequency of input signal can be read as $\mathbf{f}_0 + \mathbf{f}_{\text{offset}}$, where \mathbf{f}_0 is the center frequency of the channel, and $\mathbf{f}_{\text{offset}}$ is the offset frequency in the FFT plot. When using LF path, the FFT of input signal shows 2 sidebands $\pm \mathbf{f}_{\text{offset}}$ around $\mathbf{f}_0 = 0$ since the input signal has real components only.

The Averaging function of the Scope mode is performed in the hardware level. This means if the number of averages is > 1 , the recorded raw data is the data after the averaging, i.e. in the time-domain, the result is $(\mathbf{I} + \mathbf{iQ})$, in FFT mode, the result is $|\text{FFT}((\mathbf{I} + \mathbf{iQ}))|$.

Functional Elements

Table 5.53: Scope tab functional elements

Control/Tool	Option/Range	Description
Run/Stop		Runs the scope/FFT with a default internal trigger (200 ms) if Trigger mode is disabled, or a configured trigger if Trigger mode is enabled.
Single		Acquires a single shot of samples.
Horizontal Mode	Time Domain / Freq Domain (FFT)	Switches between time and frequency domain display.
Shown Trigger	grey/green/yellow	When flashing, indicates that new scope shots are being captured and displayed in the plot area. The Trigger must not necessarily be enabled for this indicator to flash. A disabled trigger is equivalent to continuous acquisition. Scope shots with data loss are indicated by yellow. Such an invalid scope shot is not processed.
Length Mode & Value		Switches between length and duration display.
	Length (pts) 16 to 2^{16} / 2^{15} / 2^{14} Sa for 1 / 2 / 3 or 4 channels;	The scope shot length is defined in number of samples. The duration is given by the number of samples divided by the sampling rate 2.0 GSa/s. The granularity is 16 Samples.
	Duration (s) 8 ns to 32 / 16 / $8 \mu\text{s}$ for 1 / 2 / 3 or 4 channels;	The scope shot length is defined as a duration. The number of samples is given by the duration times the sampling rate 2.0 GSa/s. The resolution is 8 ns.
Channel N	Signal Input m (m is 1 to 2 or 1 to 4)	Select input source of the Scope channels.
Enable	ON / OFF	Activates the display of the corresponding scope channel.
Enable Hardware Averaging	ON / OFF	Enable hardware averaging where results are available and displayed only once all necessary shots have been acquired. As opposed to the EMA filter, the source data for hardware averaging is always the time trace before any postprocessing such as FFT.
Averages (HW)	1 to 65536	Number of shots to average on the Instrument before returning the data
Enable Average Filter	ON / OFF	Enable Exponential Moving Average (EMA) filter that is applied when the average of several scope shots is computed and displayed. Depending on the mode, the source data for averaging is either the Time or the Freq FFT trace.
Averages (EMA)	Integer from 1	Number of shots required to reach 63% setting. Twice the number of shots yields 86% setting.
Reset (EMA)		Resets the averaging filter.
Scope Display	ON / OFF	Display traces in a 1D plot.

Control/Tool	Option/Range	Description
2D Display	ON / OFF	Display traces in a 2D plot. In segment mode, the vertical axis shows the number of segments, the horizontal axis shows the length/duration of the shots. It can be used together with Scope Display.
Trends Display	ON / OFF	Display trends of monitored traces. It can be used together with the Scope Display.

Table 5.54: Scope Tab: Trigger Sub-Tab

Control/Tool	Option/Range	Description
Enable	ON / OFF	When triggering is enabled scope shots are acquired every time the defined trigger condition is met. If disabled, scope shots are acquired continuously (every 200 ms).
Signal	Trigger Input nA and nB, Sequencer n Trigger Out, Sequencer n Monitor, Software Trigger, internal trigger. n is from 1 to 2 or 1 to 4	Selects the trigger source signal.
Delay (s)	-4 μ s to 131 μ s	Trigger position relative to reference. A positive delay results in less data being acquired before the trigger point, a negative delay results in more data being acquired before the trigger point. The delay resolution is 2 ns.
Enable Segments	ON / OFF	Enable segmented scope recording. This allows for full bandwidth recording of scope shots with a minimum dead time between individual shots.
Segments	1 to 1024	Specifies the number of segments to be recorded in device memory. The maximum scope length size is given by the available memory divided by the number of segments.
Shown Trigger	1 to 1024	Displays the number of triggered events since last start.

Table 5.55: Scope Tab: Advanced Sub-Tab

Control/Tool	Option/Range	Description
FFT Window	Rectangular Hann Hamming Blackman Harris Exponential (ring-down) Cosine (ring-down) Cosine squared (ring-down)	Seven different FFT windows to choose from. Each window function results in a different trade-off between amplitude accuracy and spectral leakage. Please check the literature to find the window function that best suits your needs.
Resolution (Hz)	8 / 15 / 30 kHz to 125 MHz for 1 / 2 / 3 or 4 channels	Spectral resolution defined by the reciprocal acquisition time (sample rate, number of samples recorded).
Spectral Density	ON / OFF	Calculate and show the spectral density. If power is enabled the power spectral density value is calculated. The spectral density is used to analyze noise.
Power	ON / OFF	Calculate and show the power value. To extract power spectral density (PSD) this button should be enabled together with Spectral Density.
Persistence	ON / OFF	Keeps previous scope shots in the display. The color scheme visualizes the number of occurrences at certain positions in the time and amplitude by a multi-color scheme.

Table 5.56: Scope Tab: History Sub-Tab

Control/Tool	Option/Range	Description
History	History	Each entry in the list corresponds to a single trace in the history. The number of traces displayed in the plot is limited to 20. Use the toggle buttons to hide or show individual traces. Use the color picker to change the color of a trace in the plot. Double click on a list entry to edit its name.
Length	0 to 4294967295	Maximum number of records in the history. The number of entries displayed in the list is limited to the 100 most recent ones.
Clear All		Remove all records from the history list.
Clear		Remove selected records from the history list.
Save		Save the traces in the history to a file accessible in the File Manager tab. The file contains the signals in the Vertical Axis Groups of the Control sub-tab. The data that is saved depends on the selection from the pull-down list. Save All: All traces are saved. Save Sel: The selected traces are saved.
File Name		Enter a name which is used as the head of the folder name to save the history into. An additional three-digit counter is added as the rest of the folder name automatically to identify consecutive files.
File Format		Select the file format in which to save the data.
Auto Save		Activate autosaving. When activated, any measurements already in the history are saved. Each subsequent measurement is then also saved. The autosave directory is identified by the text "autosave" in the name, e.g. "sweep_autosave_001". If autosave is active during continuous running of the module, each successive measurement is saved to the same directory. For single shot operation, a new directory is created containing all measurements in the history. Depending on the file format, the measurements are either appended to the same file, or saved in individual files. For HDF5 and ZView formats, measurements are appended to the same file. For MATLAB and SXM formats, each measurement is saved in a separate file.
Load file		Load data from a file into the history. Loading does not change the data type and range displayed in the plot, this has to be adapted manually if data is not shown.

5.2.8. DIO Tab

The DIO tab provides access to the settings and controls of the digital inputs and outputs. It is available on all SHFQC+ Instruments.

Features

- Monitor and control of 32-bit DIO port
- Configure Trigger Inputs and Marker Outputs
- Configure the Internal Trigger settings

Description

The DIO tab is the main panel to control the digital inputs and outputs as well as the trigger levels. Whenever the tab is closed or an additional one of the same type is needed, clicking the following icon will open a new instance of the tab.

Table 5.57: App icon and short description

Control/Tool	Option/Range	Description
DIO		Gives access to all controls relevant for the digital inputs and outputs including DIO, Trigger Inputs, Trigger Outputs, and Marker Outputs.

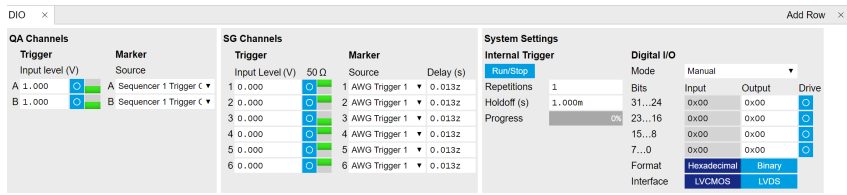


Figure 5.34: LabOne UI: DIO tab

The QA Channels section shows the settings for the two Trig inputs on the front panel of the Quantum Analyzer channel. The LED status indicator helps in monitoring the input signal state and selecting the threshold. The Marker section allows users to assign internal marker bits to the Mark outputs on the front panel. Alternatively, the outputs can be set to static high or low values. The SG Channels section shows similar settings for the 2, 4, or 6 Signal Generator channels. Additionally, the marker outputs of the Signal Generator channels have a configurable delay, with a resolution of 1 ns. In the System Settings section, the number of repetitions and the holdoff time of the Internal Trigger can be configured. The Internal Trigger is useful for synchronizing the outputs of different channels on the same instrument.

Digital I/O

The Digital I/O has 3 operation modes: Manual means controlled manually, QA Sequencer n means controlled by QA Sequencer n, QA there are the 32-bit DIO port is in use.

Figure 5.35 shows the architecture of the DIO port. It features 32 bits that can be configured byte-wise as inputs or outputs by means of a drive signal. The digital output data is latched synchronously with the falling edge of the internal clock, which is running at 50 MHz. The internal sampling clock is available at the DOL pin of the DIO connector. Digital input data can either be sampled by the internal clock or by an external clock provided through the CLKI pin. A decimated version of the input clock is used to sample the input data. The Decimation unit counts the clocks to decimation and then latches the input data. The default decimation is 5625000, corresponding to a digital input sampling rate of 1 sample per second.

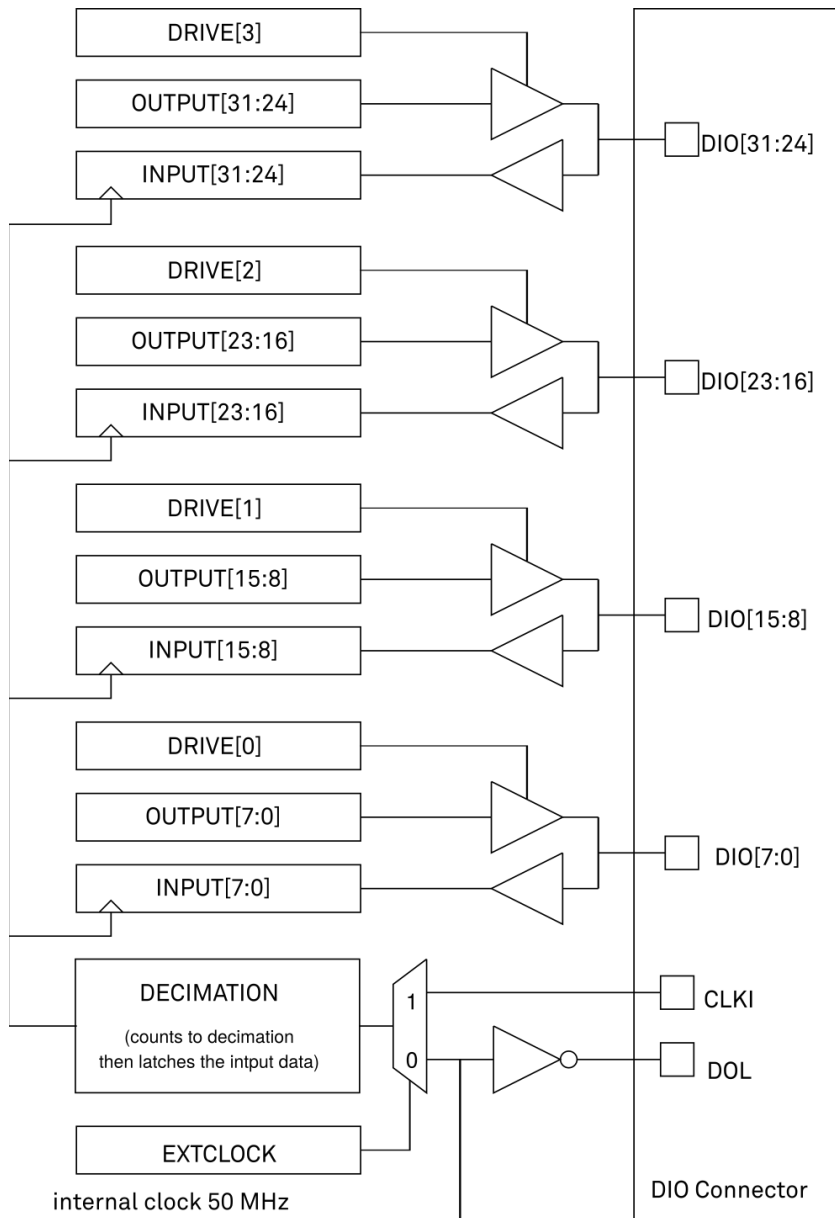


Figure 5.35: DIO input/output architecture

In **Manual** mode, each DIO pin can be controlled manually according to [Figure 5.35](#) and the DIO interface specification is detailed in [Specifications](#).

In **QA Sequencer N** mode, each DIO pin output can be controlled in the SHFQC+ [Readout Pulse Generator Tab](#) Sequencer N (N indicates which Channel) by a SeqC command **setDIO**.

In **QA Results** mode, DIO pins are configured to send Readout Result after Thresholding. [Table 5.58](#) shows the mapping between DIO pins and input/output signals available when using the DIO connector to output qubit state measurement results. The direction is as seen from the SHFQC+ Instrument. In order to use these signals, the Digital I/O Mode and Drive setting have to be chosen accordingly.

Table 5.58: DIO signal assignment in QA Results Mode

DIOLink signal	DIO pin	Direction	Description
VALID	DIO[0]	OUT	valid bit
CW	DIO[4:1]	OUT	one-hot encoding of Readout Channel
CW	DIO[8:23]	OUT	quantized results for a maximum of 16 State Discriminations
reserved	DIO[24:31]	IN	incoming communication

DIO Result communication below shows an example of multiple channel readout transmissions through DIO. Every readout is sent in a single message. A one-hot encoding of the readout channel is sent along with the readout on dedicated bits. The valid bit is set for every valid DIO transaction.

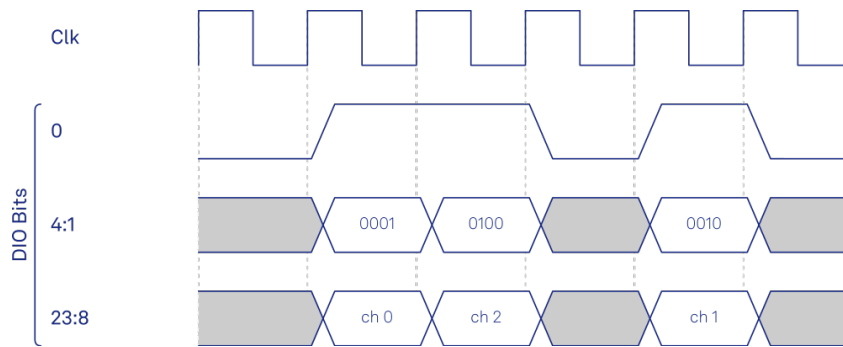


Figure 5.36: DIO Result communication

The QA Results QCCS mode of the DIO interface provides one way of communicating discriminated qubit states between 2 Instruments. For more than 2 Instruments, both qubit states and synchronization becomes essential. The following section explains how the ZSync interface works for both Instrument synchronization and feedback. Please note that ZSync settings are under the [Device Tab](#).

ZSync Interface

The ZSync link of the Zurich Instruments' [Quantum Computing Control System \(QCCS\)](#) enables Instrument synchronization and communication on the system level through the Zurich Instruments' [PQSC Programmable Quantum System Controller](#). This architecture is able to support quantum algorithms run in scalable quantum processors.

In particular, the ZSync links distribute the system clock to all Instruments and synchronize all Instruments to sub-nanosecond levels. Besides status monitoring to ensure quality and reliability of qubit tune-up routines, it provides a bidirectional data interface to send readout results to, or obtain sequence instructions from the PQSC.

The ZSync links adhere to strict real-time behavior: all data transfers are predictable to single clock cycle precision. In the SHFQC+, the link is optimized for maximum data transfer bandwidth to the central controller. For example, twice the bandwidth is reserved for results being transferred to the PQSC with respect to the allocated bandwidth for instructions that are received from the PQSC. This enables global feedback and error correction through centralized syndrome decoding and synchronized actions on the global QCCS system level.

Feedback through the PQSC

Note

More information on the ZSync, and how to properly link the SHFQC+ with the QCCS can be found in the user manual of the PQSC Programmable Quantum System Controller.

Using the **startQA-** command, the SHFQA+ or the Quantum Analyzer Channel of the SHFQC+ generates a readout result and forwards it to the PQSC over the ZSync. Depending on the address provided, the PQSC stores it in the register bank - the center of the feedback in the QCCS system. After processing, the PQSC then forwards the results to other devices in the QCCS, such as the SHFSG+.

The register bank requires a readout to have an address and a mask along with the readout data. Each component is sent in a separate ZSync message. The address is sent first, followed by the mask, and then the data, see [Figure 5.37](#). To reduce latency, the address and the mask are sent during the readout, and the data is then sent as soon as the discriminated qubit results are ready.

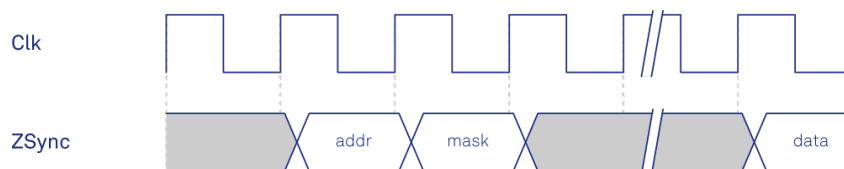



Figure 5.37: Readout Result communication via ZSync

Functional Elements

Table 5.59: Digital input and output channels, reference and trigger

Control/Tool	Option/Range	Description
DIO mode		Select DIO mode
	Manual	Enables manual control of the DIO output bits.
	Sequencer	Enables control of DIO values by the Sequencer.
	Result	Sends discriminated Readout Results to the DIO.
DIO mode		Select DIO mode
	Manual	Enables manual control of the DIO output bits.
QA Result Overflow	grey/yellow/red	Red: present overflow condition on the DIO interface during readout. Yellow: indicates an overflow occurred in the past. An overflow can happen if readouts are triggered faster than the maximum possible data-rate of the DIO interface.
DIO bits	label	Partitioning of the 32 bits of the DIO into 4 buses of 8 bits each. Each bus can be used as an input or output.
DIO input	numeric value in either Hex or Binary format	Current digital values at the DIO input port.
DIO output	numeric value in either hexadecimal or binary format	Digital output values. Enable drive to apply the signals to the output.
DIO drive	ON / OFF	When on, the corresponding 8-bit bus is in output mode. When off, it is in input mode.
Format		Select DIO view format.
	Hexadecimal	DIO view format is hexadecimal.
	Binary	DIO view format is binary.
Clock		Select DIO internal or external clocking.
Interface		Selects the interface standard to use on the 32-bit DIO interface. This setting is persistent across device reboots.
	LVCMOS	A single-ended, 3.3V CMOS interface is used.
	LVDS	A differential, LVDS compatible interface is used.
Trigger level		Trigger voltage level at which the trigger input toggles between low and high. Use 50% amplitude for digital input and consider the trigger hysteresis.
50 Ω	50 Ω /1 k Ω	Trigger input impedance: When on, the trigger input impedance is 50 Ω , when off 1 k Ω .
Trigger Input Low status		Indicates the current low level trigger state.
	Off	A low state is not being triggered.
	On	A low state is being triggered.
Trigger Input High status		Indicates the current high level trigger state.
	Off	A high state is not being triggered.
	On	A high state is being triggered.

Control/Tool	Option/Range	Description
Marker output signal		Select the signal assigned to the marker output.
Delay (s)		This delay adds an offset that acts only on the trigger/marker output. The total delay to the trigger/marker output is the sum of this value and the value of the output delay node.
Run/Stop		Enable internal trigger generator.
Repetitions		Number of triggers to be generated.
Holdoff		Hold-off time between generated triggers.
Progress		The fraction of the triggers generated so far.
Synchronization		Enable synchronization. Trigger generation will only start once all synchronization participants have reported a ready status. Synchronization checks will be repeated with the same trigger generation settings (holdoff and repetitions) until synchronization is disabled.

5.2.9. Output Router and Adder

The Output Router and Adder is a software upgrade option for the SHFQC+. The option can be installed in the field.

Features

- Signals from up to three additional Digital Signal Units can be routed and added to any Output of an SG Channel
- Independent amplitude and phase control for each routed signal
- Ability to enable each routed signal separately
- Overflow counter to indicate if the added signals saturate the DAC
- Gain access to additional Digital Signal Units for extended signal generation capabilities on some instruments

Description

The Output Router and Adder is a feature that allows the user to flexibly route the signals generated by different Digital Signal Units to any front panel Output of an SG Channel. The same signal can be routed to multiple Outputs at once, and the user has completely independent amplitude and phase control of each routed instance of a signal. This can have uses in crosstalk compensation on the RF lines for superconducting qubits, but it can also be used to simultaneously drive multiple hyperfine transitions in color centers, to perform state transfer protocols in quantum optics, or to perform other experiments where frequency multiplexing is needed.

The Output Router and Adder works by introducing additional signal line connections between the different digital signal pathways that lead to the analog upconversion chains of the Outputs. The functional diagram below highlights how a signal could be routed from the Digital Signal Unit of SG Channel 2 to the Output of SG Channel 1.

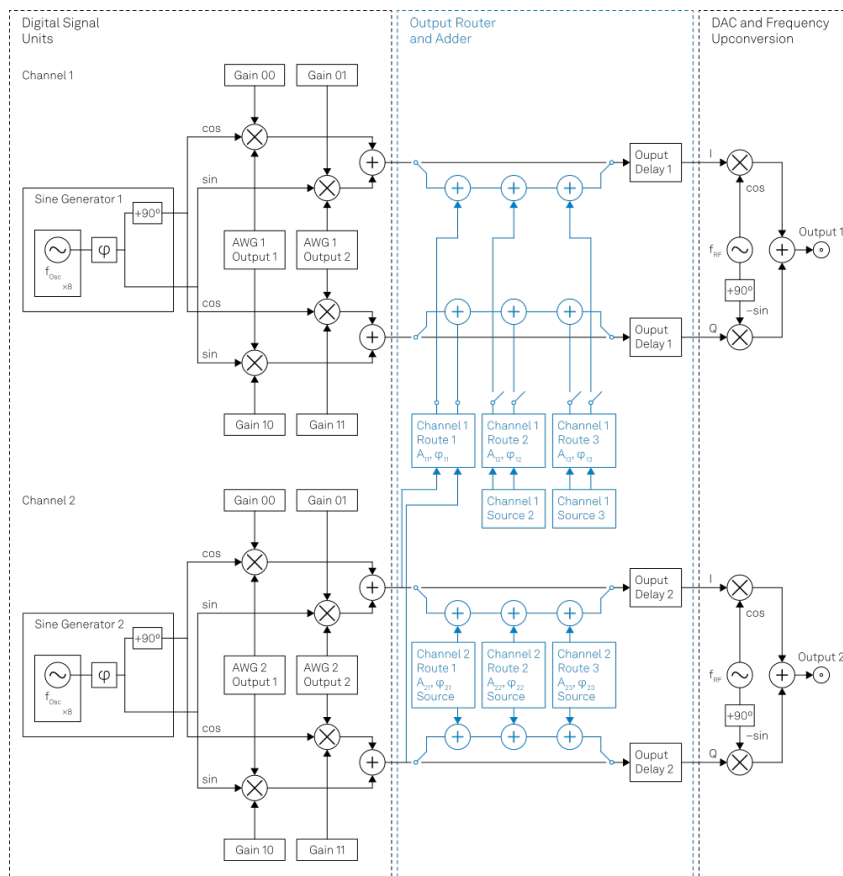


Figure 5.38: Diagram showing how the digital signals are routed between SGChannels. The default signal pathways are shown in black, whereas the additional pathways and controls are highlighted in light blue.

In this case, the digital signal generated by AWG 2 and modulated by the Digital Modulation settings of SG Channel 2 is routed to the digital signal line that leads to the analog upconversion chain of SG Channel Output 1, with the routed signal indicated by the light blue connections leading from SG Channel 2 to SG Channel 1. The user has the ability to add an additional amplitude scaling factor and phase shift to the routed signal, indicated by A_{11} and ϕ_{11} in the diagram. The signal that is generated at SG Channel Output 1 will therefore be a linear combination of the signal normally generated from AWG core 1, as well as an amplitude-scaled and phase-shifted version of the signal generated by AWG core 2 and Digital Modulation settings of SGChannel 2. For detailed information on how the signals from the Digital Signal Units are generated and how the Digital Modulation settings are applied, please refer to the [Digital Modulation Tutorial](#). Similarly, for more information on how the DAC and the Frequency Upconversion Chain convert the digital signals into the final analog signal at the desired RF center frequency, please refer to the [In/Out Tab](#).

Note

It is not possible to route marker or trigger output data between Channels: Only waveform information is routed.

Each SG Channel has three routes, corresponding to the three additional digital signals that can be added to the default signal pathway of that channel. Each route has its own enable/disable switch, source (to select which channel the signal should be drawn from), amplitude scaling factor, and phase shift. All of the signal routing, signal addition, amplitude scaling, and phase shifting happens digitally, before the digital-to-analog conversion and associated analog upconversion chain. The node `device.sgchannels[n].outputrouter.overflowcount()` can be queried to determine whether the total signal, comprising the default signal for that SG Channel Output as well as all routed signals that are added to it, has produced an overflow at the DAC, indicating that the signal was clipped. If the Output Router and Adder has detected an overflow in the past, it automatically clamps the signal to be within the range $[-1, 1]$, possibly distorting the signal. Both the amplitude scaling factor and the phase shift of each routed signal are floating numbers that are serialized to 16 bits before being written to the FPGA. The table below summarizes each of the different node settings that are part of the Output Router and Adder.

Table 5.60: Output Router and Adder node settings

Name	Node	Description
Channel n Output Router Enable	<code>device.sgchannels[n].outputrouter.enable()</code>	Enables (1) or disables (0) the Output Router of Channel n
Channel n Output Router Overflow Indicator	<code>device.sgchannels[n].outputrouter.overflowcount()</code>	Indicates the number of overflow events for the Output Router of Channel n
Channel n Route m Enable	<code>device.sgchannels[n].outputrouter.routes[m].enable()</code>	Enables (1) or disables (0) Route m of the Output Router Adder of Channel n
Channel n Route m Source	<code>device.sgchannels[n].outputrouter.routes[m].source()</code>	Index of the Channel from which the signal originates. Note that it is not possible to have both <code>device.sgchannels[0].outputrouter.routes[0].source()</code> and <code>device.sgchannels[0].outputrouter.routes[1].source()</code> simultaneously. Similarly, the Output Router Adder of Channel n cannot accept its own index as a Source. For example, <code>device.sgchannels[n].outputrouter.routes[n].source()</code> is invalid.
Channel n Route m Amplitude	<code>device.sgchannels[n].outputrouter.routes[m].amplitude()</code>	Amplitude scaling factor (between 0 and 1) applied to the routed signal. Indicated by ϕ_{nm} in the equations.
Channel n Route m Phase	<code>device.sgchannels[n].outputrouter.routes[m].phase()</code>	Phase shift (any real value, serially added to the routed signal. Indicated by ϕ_{nm} in the equations.

Based on the figure and table above, we can write down the total I and Q signals that are present on SG Channel 1. Before reaching the Output Router and Adder, SG Channels 1 and 2 create the following I and Q signals:

$$\begin{aligned}
 V_{I,Ch1}(t) &= \text{Gain00}_{Ch1} \times w_{I,Ch1}(t) \cos(2\pi f_{Osc,Ch1}t + \phi_{Ch1}) + \text{Gain01}_{Ch1} \times w_{Q,Ch1}(t) \sin(2\pi f_{Osc,Ch1}t + \phi_{Ch1}) \\
 V_{Q,Ch1}(t) &= \text{Gain10}_{Ch1} \times w_{I,Ch1}(t) \sin(2\pi f_{Osc,Ch1}t + \phi_{Ch1}) + \text{Gain11}_{Ch1} \times w_{Q,Ch1}(t) \cos(2\pi f_{Osc,Ch1}t + \phi_{Ch1}) \\
 V_{I,Ch2}(t) &= \text{Gain00}_{Ch2} \times w_{I,Ch2}(t) \cos(2\pi f_{Osc,Ch2}t + \phi_{Ch2}) + \text{Gain01}_{Ch2} \times w_{Q,Ch2}(t) \sin(2\pi f_{Osc,Ch2}t + \phi_{Ch2}) \\
 V_{Q,Ch2}(t) &= \text{Gain10}_{Ch2} \times w_{I,Ch2}(t) \sin(2\pi f_{Osc,Ch2}t + \phi_{Ch2}) + \text{Gain11}_{Ch2} \times w_{Q,Ch2}(t) \cos(2\pi f_{Osc,Ch2}t + \phi_{Ch2})
 \end{aligned}$$

where all of the symbols are as defined in the [Digital Modulation Tutorial](#), and the subscript **Chn** indicates from which SG Channel the signal or setting originates. At this stage, these are digital I and Q signals, not analog voltages, that will be sent to the double-superheterodyne upconversion scheme explained in the [In/Out Tab](#). It is these digitally modulated I and Q signals that are routed between channels. In this case, we are using the default signal of SG Channel 1 as well as the first route of the Output Router and Adder, with SG Channel 2 as the source for that route. The total I and Q signals that are sent to the DAC and Output of SG Channel 1 are therefore given by the following equation:

$$\begin{aligned}
 V_{I,Ch1,Total}(t) &= V_{I,Ch1}(t) + A_{00} (V_{I,Ch2}(t) \cos(\phi_{00}) - V_{Q,Ch2}(t) \sin(\phi_{00})) \\
 V_{Q,Ch1,Total}(t) &= V_{Q,Ch1}(t) + A_{00} (V_{I,Ch2}(t) \sin(\phi_{00}) + V_{Q,Ch2}(t) \cos(\phi_{00}))
 \end{aligned}$$

where A_{00} and ϕ_{00} stand for the node settings

`device.sgchannels[0].outputrouter.routes[0].amplitude()` and

`device.sgchannels[0].outputrouter.routes[0].phase()`, as described in the table above.

Because all of the signal routing and adding happens digitally, all frequency components of the total signal should lie within the 1 GHz bandwidth of the analog upconversion chain, or there is a risk of attenuating parts of the generated RF signal. For use cases in which frequency multiplexing in a bandwidth exceeding 1 GHz is needed, it is recommended to combine the desired RF signals external to the instrument. This also means that all channels between which crosstalk compensation is being performed must share the same RF center frequency, such that the compensation pulse appears at the correct frequency at the Output.

Note that enabling the Output Router on a given channel increases the output latency by 26 ns (52 samples), as the signal pathway is extended to include additional signal addition stages. It is highly recommended to enable the Output Router on all channels that are sharing signals in any way, to ensure that the signals at the SG Outputs on the front panel remain synchronized. For example, if AWG core 2 is generating a signal that is routed to SG Output 1, it is recommended to enable the

Output Router on SG Channel 2 as well, even though it is not adding signals from other channels to its own output. This is to ensure that the relative timing of the signal from SG Channel 2 played on SG Channel Output 2 lines up with the SG Channel 2 signal played on SG Output 1.

For instruments that have fewer than the maximum possible number of AWG cores for the instrument class, additional Digital Signal Units become available. This means that an SHFQC2 (SHFQC4) will have access to 6 AWG cores in total, as well as the corresponding Digital Modulation settings and other digital signal settings, such as the digital trigger and the mask, shift, and offset settings for processing data received over DIO or ZSync. The first 2 (4) AWG cores, with indices 0 – 1 (0 – 3), and corresponding Digital Modulation settings come with the SHFQC2 (SHFQC4) by default. The 4 (2) additional Digital Signal Units, with indices 2 – 5 (4 – 5), grant access to both another AWG core and Digital Modulation settings with which to modulate the AWG signals, as well as other several other settings needed to configure the digital signal pathways. The additional Digital Signal Units have all the same settings and abilities as the channels that come with the base version of the instrument, but they are not associated with an SG Channel Output on the front panel by default and therefore do not generate an output signal unless their signals are intentionally routed to an Output using the Output Router and Adder. The table below lists the functionality that is NOT available on the additional Digital Signal Units.

Table 5.61: Node settings that are NOT available on the additional Digital Signal Units made available with the Output Router and Adder

Name	Node
Channel n Digitalmixer Center Frequency	<code>device.sgchannels[n].digitalmixer.centerfreq()</code>
Channel n Marker Source	<code>device.sgchannels[n].marker.source()</code>
Channel n Output Center Frequency	<code>device.sgchannels[n].centerfreq()</code>
Channel n Output Delay	<code>device.sgchannels[n].output.delay()</code>
Channel n Output Filter	<code>device.sgchannels[n].output.filter()</code>
Channel n Output On	<code>device.sgchannels[n].output.on()</code>
Channel n Output Over Range Counter	<code>device.sgchannels[n].output.overrangecount()</code>
Channel n Output Range	<code>device.sgchannels[n].output.range()</code>
Channel n Output RFLF Path	<code>device.sgchannels[n].output.rflfpath()</code>
Channel n Synthesizer	<code>device.sgchannels[n].synthesizer()</code>
Channel n Trigger Delay	<code>device.sgchannels[n].trigger.delay()</code>
Channel n Trigger Impedance (50 Ohm)	<code>device.sgchannels[n].trigger.imp50()</code>
Channel n Trigger Level	<code>device.sgchannels[n].trigger.level()</code>
Channel n Trigger Value	<code>device.sgchannels[n].trigger.value()</code>

All of the above settings involve analog settings or other settings involved in the upconversion chain that do not apply to the additional Digital Signal Units. Although the node `device.sgchannels[n].output.delay()` is a digital delay, its effects are applied after the Output Router and Adder but before the DAC and is therefore not needed for the additional Digital Signal Units. Having the delay node implemented after the Output Router and Adder also ensures that the delay on a given Output is common to all signals applied on that line.

How-To: Route signals between Channels 1, 2, 4, and 6

For motivation, consider a superconducting qubit chip in which SG Channels 1 – 6 (e.g. of an SHFQC6) are connected to Qubits 1 – 6 and in which Qubit 4 experiences strong crosstalk to Qubits 1, 2, and 6. Assuming the amplitude of the crosstalk has been characterized, as has the necessary phase shift for the compensation pulse, we can use the following node settings to enable the instrument to automatically play compensation pulses, such that the net effect of the crosstalk at the qubit is negated.

We assume that the instrument has already been connected to and that all of the AWGs and Digital Modulation settings have been programmed or set up.

```
# Define paths for Output Routers of each channel used
ch1_rtr = device.sgchannels[0].outputrouter
```

```

ch2_rtr = device.sgchannels[1].outputrouter
ch4_rtr = device.sgchannels[3].outputrouter
ch6_rtr = device.sgchannels[5].outputrouter

with sg.set_transaction():
    ## Signals routed to Output 1
    ch1_rtr.enable(1) # allow other signals to be added to the output of this
channel
    ch1_rtr.routes[0].enable(1) # enable route 1
    ch1_rtr.routes[0].source(3) # for route 1, use SG channel 4 as the source
    ch1_rtr.routes[0].amplitude(AMP_Q4_TO_Q1) # apply an amplitude scaling factor
of AMP_Q4_TO_Q1, corresponding to the amount of leakage from charge line 4 into
qubit 1
    ch1_rtr.routes[0].phase(PHASE_Q4_TO_Q1)
# apply a phase shift of PHASE_Q4_TO_Q1 degrees to the routed signal,
corresponding to the phase shift needed to cancel out the leakage from charge
line 4 into qubit 1

    ## Signals routed to Output 2
    ch2_rtr.enable(1) # allow other signals to be added to the output of this
channel
    ch2_rtr.routes[0].enable(1) # enable route 1
    ch2_rtr.routes[0].source(3) # for route 1, use SG channel 4 as the source
    ch2_rtr.routes[0].amplitude(AMP_Q4_TO_Q2) # apply an amplitude scaling factor
of AMP_Q4_TO_Q6 to the routed signal
    ch2_rtr.routes[0].phase(PHASE_Q4_TO_Q2) # apply a phase shift of PHASE_Q4_TO_Q2

    ## Signals routed to Output 4
    ch4_rtr.enable(1) # allow other signals to be added to the output of this
channel
    # Route 1
    ch4_rtr.routes[0].enable(1) # enable route 1
    ch4_rtr.routes[0].source(0) # for route 1, use SG channel 1 as the source
    ch4_rtr.routes[0].amplitude(AMP_Q1_TO_Q4) # apply an amplitude scaling factor
of AMP_Q1_TO_Q4 to the routed signal
    ch4_rtr.routes[0].phase(PHASE_Q1_TO_Q4) # apply a phase shift of PHASE_Q1_TO_Q4

    # Route 2
    ch4_rtr.routes[1].enable(1) # enable route 2
    ch4_rtr.routes[1].source(1) # for route 2, use SG channel 2 as the source
    ch4_rtr.routes[1].amplitude(AMP_Q2_TO_Q4) # apply an amplitude scaling factor
of AMP_Q2_TO_Q4 to the routed signal
    ch4_rtr.routes[0].phase(PHASE_Q2_TO_Q4) # apply a phase shift of PHASE_Q2_TO_Q4

    # Route 3
    ch4_rtr.routes[2].enable(1) # enable route 1
    ch4_rtr.routes[2].source(5) # for route 3, use SG channel 6 as the source
    ch4_rtr.routes[2].amplitude(AMP_Q6_TO_Q4) # apply an amplitude scaling factor
of AMP_Q6_TO_Q4 to the routed signal
    ch4_rtr.routes[2].phase(PHASE_Q6_TO_Q4) # apply a phase shift of PHASE_Q6_TO_Q4

    ## Signals routed to Output 6
    ch6_rtr.enable(1) # allow other signals to be added to the output of this
channel
    ch6_rtr.routes[0].enable(1) # enable route 1
    ch6_rtr.routes[0].source(3) # for route 1, use SG channel 4 as the source
    ch6_rtr.routes[0].amplitude(AMP_Q4_TO_Q6) # apply an amplitude scaling factor
of AMP_Q4_TO_Q6 to the routed signal
    ch6_rtr.routes[0].phase(PHASE_Q4_TO_Q6)
# apply a phase shift of PHASE_Q4_TO_Q6 degrees to the routed signal

```

With these settings, any sequence played on SG Channel 4 will automatically play compensation pulses on SG Channel Outputs 1, 2, and 6. Similarly, any sequence played on SG Channels 1, 2, or 6

will automatically play a compensation pulse on SG Channel Output 4 (so long as the corresponding AWG cores and analog Outputs are all enabled).

Since we are adding many signals together on SG Channel Output 4, there is a risk that the total signal could saturate the DAC at some points. To check whether this is the case, a user can query the node `device.sgchannels[3].outputrouter.overflowcount()`. The result is the number of overflows that have occurred so far, and a non-zero result means that an overflow has occurred in the past, and the digital signals must be scaled down. This can be accomplished, for example, by increasing the range setting of the SG Channel Output and reducing the digital amplitudes of all signals to compensate. A result of `0` means that no overflow has occurred.

6. Specifications

Important

Unless otherwise stated, all specifications apply after 30 minutes of instrument warm-up.

For measurements **in which high gate fidelity** is crucial, it is highly recommended to enable all required outputs and inputs and wait for 2 hours after powering on the instrument.

Important

Important changes in the specification parameters are explicitly mentioned in the revision history of this document.

6.1. General Specifications

Table 6.1: General and storage

Parameter	Min	Typ	Max
storage temperature	-25 °C	-	65 °C
storage relative humidity (non-condensing)	-	-	95%
operating temperature	5 °C	-	40 °C
operating relative humidity (non-condensing)	-	-	90%
specification temperature	18 °C	-	28 °C
power consumption	-	-	300 W
operating environment	IEC61010, indoor location, installation category II, pollution degree 2		
operating altitude	up to 2000 meters		
power inlet fuses	250 V, 2 A, fast acting, 5 x 20 mm		
power supply AC line	100-240 V ($\pm 10\%$), 50/60 Hz		
dimensions (width x depth x height)	45.0 x 39.7 x 13.2 cm (no handle), 17.7 x 15.6 x 5.2 inch, 19 inch rack compatible		
weight	15 kg (33 lb)		
recommended calibration interval	2 years		

Table 6.2: Maximum ratings

Parameter	Min	Typ	Max
damage threshold Out	-	-	+30 dBm
damage threshold In	-	-	+20 dBm
damage threshold Mark Out	-0.7 V	-	+4 V
damage threshold Trig In (1 k Ω input impedance)	-11 V	-	+11 V
damage threshold Trig In (50 Ω input impedance)	-6 V	-	+6 V
damage threshold Aux In (DC)	-10 V	-	+10 V
damage threshold Aux In (AC)	-	-	+20 dBm
damage threshold External Clk In (DC)	-3 V	-	+3 V

Parameter	Min	Typ	Max
damage threshold External Clk In (AC, with DC offset 0 V)	-	-	+13.5 dBm
damage threshold External Clk Out (DC)	-3 V	-	+3 V
MDS In / Out	-0.7 V	-	+4 V
DIO In / Out in default configuration 3.3 V CMOS/TTL	-0.7 V	-	+4 V
torque limit front panel SMA connectors	-	-	0.5 Nm
torque limit back panel SMA connectors	-	-	1.0 Nm

Table 6.3: Host computer requirements

Parameter	Description
supported Windows operating systems	Windows 10, 11 on x86-64
supported macOS operating systems	macOS 10.11+ on x86-64 and ARMv8
supported Linux distributions	GNU/Linux (Ubuntu 14.04+, CentOS 7+, Debian 8+) on x86-64 and ARMv8
supported processors	x86-64 (Intel, AMD), ARMv8 (e.g., Raspberry Pi 4 and newer, Apple M-series)

6.2. Analog Interface Specifications

Table 6.4: Signal Outputs

Parameter	Details	Min	Typ	Max
connectors	-	SMA, front panel single-ended		
impedance	-	-	50 Ω	-
coupling	LF path	DC		
	RF path	AC		
synthesizers	Quantum Analyzer channel	One per channel shared with Signal Input of the same Channel		
	Signal Generator channels	One per channel pair		
synthesizer frequency range	Quantum Analyzer channel	1-8 GHz		
	Signal Generator channels	0.6-8 GHz		
instantaneous bandwidth (-3dB)	RF path	± 500 MHz		
	LF path, Quantum Analyzer channel	DC - 800 MHz		
	LF path, Signal Generator channels	DC - 2500 MHz		
total frequency range		DC	-	8.5 GHz
range	RF path, into 50 Ω	-30 dBm	-	+10 dBm
	LF path, into 50 Ω	-30 dBm	-	+5 dBm
output level accuracy	into 50 Ω	-	$\pm(1$ dBm of setting)	-
output level temperature drift	direct	-	0.15 dB/°C	-
	when looped with Signal Input	-	0.25 dB/°C	-
D/A converter vertical resolution	-	14 bit		

Parameter	Details		Min	Typ	Max
D/A converter sampling rate	after internal x3 interpolation		6 GSa/s		
voltage spectral noise density	RF path, 10 dBm range		-	-143 dBm/Hz	-
phase noise	RF path, 5 GHz, 10 kHz offset frequency		-	-110 dBc/Hz	-
	RF path, 5 GHz, 10 MHz offset frequency		-	-138 dBc/Hz	-
spurious free dynamic range (excluding harmonics)	RF path, 10 dBm range, CW tone, signal amplitude 10 dBm (0 dBFS)		-	48 dBc	-
worst harmonic component	10 dBm range	1 GHz	-	-40 dBc	-
		4 GHz	-	-40 dBc	-
		6 GHz	-	-38 dBc	-
		8 GHz	-	-36 dBc	-

Table 6.5: Time Domain Output Characteristics

Parameter	Details	Min	Typ	Max
skew adjustment resolution	Quantum Analyzer channel	2 ns		
	Signal Generator channels	500 ps		

Table 6.6: Signal Inputs

Parameter	Details	Min	Typ	Max
connectors	-	SMA, front panel single-ended		
impedance	-	-	50 Ω	-
coupling	RF path	AC		
synthesizers	-	One per channel shared with Signal Output of the same Channel		
synthesizer frequency range	-	1-8 GHz		
instantaneous bandwidth (-3dB)	RF path	± 500 MHz		
	LF path	DC - 800 MHz		
total frequency range	-	DC	-	8.5 GHz
range	RF path, into 50 Ω	-50 dBm	-	+10 dBm
	LF path, into 50 Ω	-30 dBm	-	+10 dBm
input level accuracy at carrier frequency and 23°C	$\leq 1 V_{pp}$ and <4 GHz	-	± 0.5 dB	-
	$\leq 1 V_{pp}$ and <8 GHz	-	± 1 dB	-
	>8 GHz	-	± 3 dB	-
input level temperature drift	direct	-	0.15 dB/C	-
	when looped with Signal Output	-	0.25 dB/C	-
A/D converter vertical resolution	-	14 bit		
A/D converter sampling rate	before internal x2 decimation	4 GSa/s		

Parameter	Details		Min	Typ	Max
voltage noise density	in-band noise measured with room-temperature 50 Ω cap, 1 GHz	10 dBm	-	-130 dBm/Hz (71 nV/ $\sqrt{\text{Hz}}$)	-
		0 dBm	-	-134 dBm/Hz (45 nV/ $\sqrt{\text{Hz}}$)	-
		-10 dBm	-	-145 dBm/Hz (13 nV/ $\sqrt{\text{Hz}}$)	-
		-20 dBm	-	-142 dBm/Hz (18 nV/ $\sqrt{\text{Hz}}$)	-
		-30 dBm	-	-162 dBm/Hz (1.78 nV/ $\sqrt{\text{Hz}}$)	-
		-40 dBm	-	-165 dBm/Hz (1.26 nV/ $\sqrt{\text{Hz}}$)	-
		-50 dBm	-	-166 dBm/Hz (1.12 nV/ $\sqrt{\text{Hz}}$)	-
spurious free dynamic range (excluding harmonics)	signal at center frequency, max. amplitude, -750 to 750 MHz	10 dBm	-	54 dBc	-
		0 dBm	-	54 dBc	-
		-10 dBm	-	55 dBc	-
		-20 dBm	-	50 dBc	-
		-30 dBm	-	58 dBc	-
		-40 dBm	-	53 dBc	-
		-50 dBm	-	45 dBc	-
3rd order intermodulation distortion	dual tone with -7 dBFS of range with 150 MHz Splitting from 1 GHz to 8 GHz	10 dBm	-	45 dBc	-
		0 dBm	-	54 dBc	-
		-10 dBm	-	54 dBc	-
		-20 dBm	-	56 dBc	-
		-30 dBm	-	54 dBc	-
		-40 dBm	-	50 dBc	-
		-50 dBm	-	40 dBc	-

Table 6.7: Marker Outputs & Trigger Inputs

Parameter	Details	Min	Typ	Max
marker outputs	Quantum Analyzer channel	2		
	Signal Generator channels	1 per channel		

Parameter	Details	Min	Typ	Max
marker outputs connector	-	SMA, front panel single-ended		
marker output high voltage	-	-	3.3 V	-
marker output low voltage	-	-	0 V	-
marker output impedance	-	-	50 Ω	-
marker output rise time 20% to 80%	-	-	300 ps	-
trigger inputs	Quantum Analyzer channel	2		
	Signal Generator channels	1 per channel		
trigger inputs connector	-	SMA, front panel single-ended		
trigger input impedance	-	50 Ω / 1 k Ω		
trigger input voltage range	50 Ω impedance	-5 V	-	5 V
	1 k Ω impedance	-10 V	-	10 V
trigger input threshold range	50 Ω impedance	-5 V	-	5 V
	1 k Ω impedance	-10 V	-	10 V
trigger input threshold resolution	-	-	< 0.4 mV	-
trigger input threshold hysteresis	-	-	> 60 mV	-

Table 6.8: Other Inputs and Outputs

Parameter	Details	min	Typ	Max
reference clock input	-	SMA on back panel		
reference clock input impedance	-	50 Ω , AC coupled		
reference clock input frequency	-	10 / 100 MHz		
reference clock input amplitude	10 MHz	-4 dBm	-	+13 dBm
	100 MHz	-5 dBm	-	+13 dBm
reference clock output	-	SMA on back panel		
reference clock output impedance	-	50 Ω , AC coupled		
reference clock output amplitude	into 50 Ω	2 Vpp	-	5 Vpp
reference clock output frequency	-	10/100 MHz		
reference clock output jitter	derived from integrated phase noise measurement (12 kHz to 20 MHz offset frequency)	-	280 fs RMS	-

Table 6.9: Oscillator and Clocks

Parameter	Details	Min	Typ	Max
internal clock type	-	OCXO		
internal clock long term accuracy / aging	-	-	-	± 0.3 ppm/year
internal clock short term stability (1 s)	-	-	-	± 0.05 ppm
internal clock initial accuracy	-	-	-	± 0.5 ppm
internal clock temperature stability	-20°C to 70°C	-	-	± 0.5 ppm
internal clock phase noise	offset 100 Hz	-	-135 dBc/Hz	-
	offset 1 kHz	-	-157 dBc/Hz	-

6.3. Digital Waveform Generation of Signal Generator Channel

Table 6.10: Waveform Generation

Parameter	Details	Specification
number of AWG cores	for SHFQC+ base version	1 per channel
	for SHFQC+ with SHFQC-RTR	6 total
AWG sampling rate	dual-channel	2 GSa/s
waveform memory per output channel	-	96 kSa ¹
sequence length	-	32 kInstructions instructions per core ²
waveform granularity	-	16 samples
minimum waveform length	-	16 samples (with command table)
sequencer clock frequency	-	250 MHz

6.4. Digital Signal Processing of Quantum Analyzer Channel

Table 6.11: Readout Pulse Generator

Parameter	Details	Specification
number of readout pulse generators	-	1
waveform memory per channel	for SHFQC+ base version	32 kSa total memory, 8 memory blocks with 4 kSa for arbitrary waveform storage, freely configurable and triggerable ¹
	for SHFQC+ with SHFQC-16W	64 kSa total memory, 16 memory blocks with 4 kSa for arbitrary waveform storage, freely configurable and triggerable ¹
sequence length	-	16 kInstructions instructions ²
waveform granularity	-	4 samples
minimum waveform length	-	4 samples
sequencer clock frequency	-	250 MHz

Table 6.12: Qubit Measurement Unit

Parameter	Details	Specification
number of Qubit Measurement Units	-	1
number of integration weights of Qubit Measurement Unit	for SHFQC+ base version	8
	for SHFQC+ with SHFQC-16W	16
integration weight time resolution	-	0.5 ns
integration weight length	-	4 kSa ¹
minimum weight length	-	4 samples
integration weight granularity	-	4 samples

Parameter	Details	Specification
multistate discrimination	-	yes
number of distinguishable states	-	up to 4

Table 6.13: Scope and Trigger Engine

Parameter	Details	Specification
Scope	memory	64 kSa ¹
	max. averages	2 ¹⁶

6.5. Digital Interface Specifications

Table 6.14: Digital Interfaces

Parameter	Description
host computer connection	USB 3.0, 1.6 Gbit/s (1 communication, 1 maintenance)
	1GbE, LAN / Ethernet, 1 Gbit/s
DIO port	4 x 8 bit, general purpose digital input/output port, 3.3 V TTL specification
ZSync peripheral port	connector for ZI proprietary bus to communicate with external peripherals

6.5.1. DIO Port

The DIO port is a VHDCI 68 pin connector as introduced by the SPI-3 document of the SCSI-3 specification. It is a female connector that requires a 32 mm wide male connector. The interface standard is switchable between LVDS (low-voltage differential signalling) and LVCMOS/LVTTL. The DIO port features 32 user-controlled bits that can all be configured byte-wise as inputs or outputs in LVCMOS/LVTTL mode, whereas in LVDS mode, half of the bits are always configured as inputs. For more specifics on how the user-definable pins can be set.

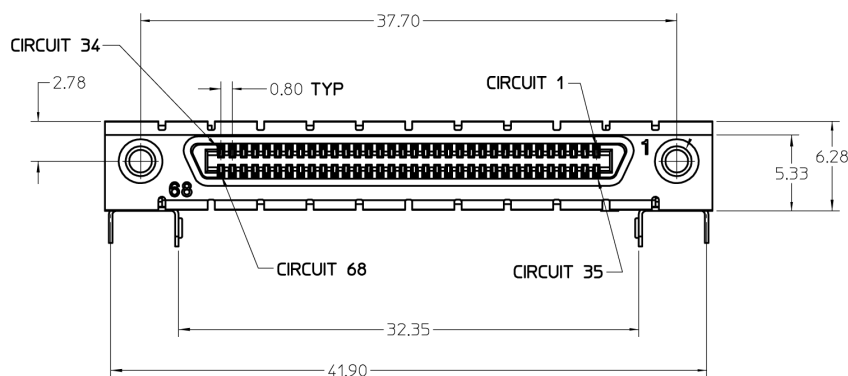


Figure 6.1: DIO HD 68 pin connector

Table 6.15: Electrical Specifications

Parameter	Details	Min	Typ	Max
supported DIO interface standards	-	LVCMOS/LVTTL (single-ended, 3.3 V); LVDS (differential)		
high-level input voltage V _{IH}	LVCMOS/LVTTL	2.0 V	-	-
low-level input voltage V _{IL}	LVCMOS/LVTTL	-	-	0.8 V
high-level output voltage V _{OH}	LVCMOS/LVTTL at I _{OH} < 12 mA	2.6 V	-	-

Parameter	Details	Min	Typ	Max
low-level output voltage VOL	LVC MOS/ LV TTL at IOL < 12 mA	-	-	0.4 V
high-level output current IOH (sourcing)	LVC MOS/ LV TTL	-	-	12 mA
low-level output current IOL (sinking)	LVC MOS/ LV TTL	-	-	12 mA
input differential voltage VID	LVDS	100 mV	-	600 mV
input common-mode voltage VICM	LVDS	0.3 V	-	2.35 V
output differential voltage VOD	LVDS	247 mV	-	454 mV
output common-mode voltage VOCM	LVDS	1.125 V	-	1.375 V

Table 6.16: DIO Pin Assignment in LVC MOS/LV TTL Mode

Pin	Name	Description
68	CLKI	digital input
67	unused	leave unconnected
66 .. 59	DIO[31:24]	digital input or output byte (set by user)
58 .. 51	DIO[23:16]	digital input or output byte (set by user)
50 .. 43	DIO[15:8]	digital input or output byte (set by user)
42 .. 35	DIO[7:0]	digital input or output byte (set by user)
34	GND	digital ground
33	unused	leave unconnected
32 .. 1	GND	digital ground

Table 6.17: DIO Pin Assignment in LVDS Mode

Pin	Name	Description
68	CLKI+	digital input
67	unused	leave unconnected
66 .. 59	DI+[31:24]	digital input byte
58 .. 51	DI+[23:16]	digital input byte
50 .. 43	DIO+[15:8]	digital input or output byte (set by user)
42 .. 35	DIO+[7:0]	digital input or output byte (set by user)
34	CLKI-	digital input
33	unused	leave unconnected
32 .. 25	DI-[31:24]	digital input byte
24 .. 17	DI-[23:16]	digital input byte
16 .. 9	DIO-[15:8]	digital input or output byte (set by user)
8 .. 1	DIO-[7:0]	digital input or output byte (set by user)

1. With binary prefix: 1 kSa = 1024 Sa ↩↩↩↩↩
2. With binary prefix: 1 kInstructions = 1024 Instructions ↩↩

7. Device Node Tree

This chapter contains reference documentation for the settings and measurement data available on SHFQC+ Instruments. Whilst [Functional Description](#) describes many of these settings in terms of the features available in the LabOne User Interface, this chapter describes them on the device level and provides a hierarchically organized and comprehensive list of device functionality.

Since these settings and data streams may be written and read using the LabOne APIs (Application Programming Interfaces) this chapter is of particular interest to users who would like to perform measurements programmatically via LabVIEW, Python, MATLAB, .NET or C.

Please see:

- [Introduction](#) for an introduction of how the instrument's settings and measurement data are organized hierarchically in the Data Server's so-called "Node Tree".
- [Reference Node Documentation](#) for a reference list of the settings and measurement data available on SHFQC+ Instruments, organized by branch in the Node Tree.

7.1. Introduction

This chapter provides an overview of how an instrument's configuration and output is organized by the Data Server.

All communication with an instrument occurs via the Data Server program the instrument is connected to (see [LabOne Software Architecture](#) for an overview of LabOne's software components). Although the instrument's settings are stored locally on the device, it is the Data Server's task to ensure it maintains the values of the current settings and makes these settings (and any subscribed data) available to all its current clients. A client may be the LabOne User Interface or a user's own program implemented using one of the LabOne Application Programming Interfaces, e.g., Python.

The instrument's settings and data are organized by the Data Server in a file-system-like hierarchical structure called the node tree. When an instrument is connected to a Data Server, its device ID becomes a top-level branch in the Data Server's node tree. The features of the instrument are organized as branches underneath the top-level device branch and the individual instrument settings are leaves of these branches.

For example, the auxiliary outputs of the instrument with device ID "dev1000" are located in the tree in the branch:

```
/dev1000/auxouts/
```

In turn, each individual auxiliary output channel has its own branch underneath the "AUXOUTS" branch.

```
/dev1000/auxouts/0/  
/dev1000/auxouts/1/  
/dev1000/auxouts/2/  
/dev1000/auxouts/3/
```

Whilst the auxiliary outputs and other channels are labelled on the instrument's panels and the User Interface using 1-based indexing, the Data Server's node tree uses 0-based indexing. Individual settings (and data) of an auxiliary output are available as leaves underneath the corresponding channel's branch:

```
/dev1000/auxouts/0/demodselect  
/dev1000/auxouts/0/limitlower  
/dev1000/auxouts/0/limitupper  
/dev1000/auxouts/0/offset  
/dev1000/auxouts/0/outputselect  
/dev1000/auxouts/0/preoffset  
/dev1000/auxouts/0/scale  
/dev1000/auxouts/0/value
```

These are all individual node paths in the node tree; the lowest-level nodes which represent a single instrument setting or data stream. Whether the node is an instrument setting or data-stream and

which type of data it contains or provides is well-defined and documented on a per-node basis in the Reference Node Documentation section in the relevant instrument-specific user manual. The different properties and types are explained in [Node Properties and Data Types](#).

For instrument settings, a Data Server client modifies the node's value by specifying the appropriate path and a value to the Data Server as a (path, value) pair. When an instrument's setting is changed in the LabOne User Interface, the path and the value of the node that was changed are displayed in the Status Bar in the bottom of the Window. This is described in more detail in [Exploring the Node Tree](#).

Module Parameters

LabOne Core Modules, such as the Sweeper, also use a similar tree-like structure to organize their parameters. Please note, however, that module nodes are not visible in the Data Server's node tree; they are local to the instance of the module created in a LabOne client and are not synchronized between clients.

7.1.1. Node Properties and Data Types

A node may have one or more of the following properties:

Property	Description
Read	Data can be read from the node.
Write	Data can be written to the node.
Setting	The node corresponds to a writable instrument configuration. The data of these nodes are persisted in snapshots of the instrument and stored in the LabOne XML settings files.
Streaming	A node with the read attribute that provides instrument data, typically at a user-configured rate. The data is usually a more complex data type, for example demodulator data is returned as ZIDemodSample . A full list of streaming nodes is available in the Programming Manual in the Chapter Instrument Communication. Their availability depends on the device class (e.g. MF) and the option set installed on the device.
Pipelined	If the sequence pipeliner mode is off the value set to the node is applied immediately. Otherwise, it goes to the staging area of the sequence pipeliner instead. Multiple pipelined nodes can be programmed as part of a job definition, that is finalized by writing a one to the relevant commit node.

A node may contain data of the following types:

Integer	Integer data.
Double	Double precision floating point data.
String	A string array.
Integer (enumerated)	As for Integer, but the node only allows certain values.
Composite data type	For example, ZIDemodSample . These custom data types are structures whose fields contain the instrument output, a timestamp and other relevant instrument settings such as the demodulator oscillator frequency. Documentation of custom data types is available in

7.1.2. Exploring the Node Tree

In the LabOne User Interface

A convenient method to learn which node is responsible for a specific instrument setting is to check the Command Log history in the bottom of the LabOne User Interface. The command in the Status Bar gets updated every time a configuration change is made. [Figure 7.1](#) shows how the equivalent

MATLAB command is displayed after modifying the value of the auxiliary output 1's offset. The format of the LabOne UI's command history can be configured in the Config Tab (MATLAB, Python and .NET are available). The entire history generated in the current UI session can be viewed by clicking the "Show Log" button.

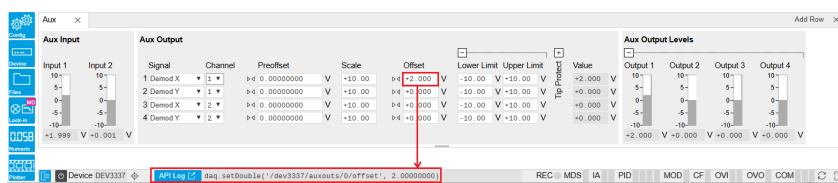


Figure 71: When a device's configuration is modified in the LabOne User Interface, the Status Bar displays the equivalent command to perform the same configuration via a LabOne programming interface. Here, the MATLAB code to modify auxiliary output 1's offset value is provided. When "Show Log" is clicked the entire configuration history is displayed in a new browser tab.

In a LabOne Programming Interface

A list of nodes (under a specific branch) can be requested from the Data Server in an API client using the `listNodes` command (MATLAB, Python, .NET) or `ziAPIListNodes()` function (C API). Please see each API's command reference for more help using the `listNodes` command. To obtain a list of all the nodes that provide data from an instrument at a high rate, so-called streaming nodes, the `streamingonly` flag can be provided to `listNodes`. More information on data streaming and streaming nodes is available in the LabOne Programming Manual.

The detailed descriptions of nodes that is provided in [Reference Node Documentation](#) is accessible directly in the LabOne MATLAB or Python programming interfaces using the "help" command. The `help` command is `daq.help(path)` in Python and `ziDAQ('help', path)` in MATLAB. The command returns a description of the instrument node including access properties, data type, units and available options. The "help" command also handles wildcards to return a detailed description of all nodes matching the path. An example is provided below.

```
daq = zhinst.core.ziDAQServer('localhost', 8004, 6)
daq.help('/dev1000/auxouts/0/offset')
# Out:
# /dev1000/auxouts/0/OFFSET#
# Add the specified offset voltage to the signal after scaling. Auxiliary
Output
# Value = (Signal+Preoffset)*Scale + Offset
# Properties: Read, Write, Setting
# Type: Double
# Unit: V
```

7.1.3. Data Server Nodes

The Data Server has nodes in the node tree available under the top-level `/zi/` branch. These nodes give information about the version and state of the Data Server the client is connected to. For example, the nodes:

- `/zi/about/version`
- `/zi/about/revision`

are read-only nodes that contain information about the release version and revision of the Data Server. The nodes under the `/zi/devices/` list which devices are connected, discoverable and visible to the Data Server.

The nodes:

- `/zi/config/open`
- `/zi/config/port`

are settings nodes that can be used to configure which port the Data Server listens to for incoming client connections and whether it may accept connections from clients on hosts other than the localhost.

Nodes that are of particular use to programmers are:

- `/zi/debug/logpath` - the location of the Data Server's log in the PC's file system,
- `/zi/debug/level` - the current log-level of the Data Server (configurable; has the Write attribute),
- `/zi/debug/log` - the last Data Server log entries as a string array.

The Global nodes of the LabOne Data Server are listed in the Instrument Communication chapter of the LabOne Programming Manual

7.2. Reference Node Documentation

This section describes all the nodes in the data server's node tree organized by branch.

7.2.1. CLOCKBASE

`/dev....clockbase`

Properties:	Read
Type:	Double
Unit:	Hz

Returns the internal clock frequency of the device.

7.2.2. DIOS

`/dev....dios/n/drive`

Properties:	Read, Write, Setting
Type:	Integer (64 bit)
Unit:	None

When on (1), the corresponding 8-bit bus is in output mode. When off (0), it is in input mode. Bit 0 corresponds to the least significant byte. For example, the value 1 drives the least significant byte, the value 8 drives the most significant byte.

`/dev....dios/n/input`

Properties:	Read
Type:	Integer (64 bit)
Unit:	None

Gives the value of the DIO input for those bytes where drive is disabled.

`/dev....dios/n/interface`

Properties:	Read, Write, Setting
Type:	Integer (64 bit)
Unit:	None

Selects the interface standard to use on the 32-bit DIO interface. A value of 0 means that a 3.3 V CMOS interface is used. A value of 1 means that an LVDS compatible interface is used.

/dev..../dios/n/mode

Properties: Read, Write, Setting
Type: Integer (enumerated)
Unit: None

Select DIO mode

0	"manual": Enables manual control of the DIO output bits.
16	"qa_result": Sends discriminated readout results to the DIO.
32	"qachan0seq", "qachannel0_sequencer": Enables control of DIO values by the sequencer of QA channel 1.
48	"sgchan0seq", "sgchannel0_sequencer": Enables control of DIO values by the sequencer of SG channel 1.
49	"sgchan1seq", "sgchannel1_sequencer": Enables control of DIO values by the sequencer of SG channel 2.
50	"sgchan2seq", "sgchannel2_sequencer": Enables control of DIO values by the sequencer of SG channel 3.
51	"sgchan3seq", "sgchannel3_sequencer": Enables control of DIO values by the sequencer of SG channel 4.
52	"sgchan4seq", "sgchannel4_sequencer": Enables control of DIO values by the sequencer of SG channel 5.
53	"sgchan5seq", "sgchannel5_sequencer": Enables control of DIO values by the sequencer of SG channel 6.

/dev..../dios/n/output

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Sets the value of the DIO output for those bytes where 'drive' is enabled.

7.2.3. FEATURES**/dev..../features/code**

Properties: Write
Type: String
Unit: None

Node providing a mechanism to write feature codes.

/dev..../features/devtype

Properties: Read
Type: String
Unit: None

Returns the device type.

/dev..../features/options

Properties: Read
Type: String
Unit: None

Returns enabled options.

/dev..../features/serial

Properties: Read
Type: String
Unit: None

Device serial number.

7.2.4. QACHANNELS

/dev..../qachannels/n/busy

Properties: Read
Type: Integer (64 bit)
Unit: None

Indicates that the channel is busy applying settings, e.g., center-frequency or analog output settings.

/dev..../qachannels/n/centerfreq

Properties: Read, Write, Setting
Type: Double
Unit: Hz

The Center Frequency of the analysis band.

/dev..../qachannels/n/generator/auxtriggers/n/channel

Properties: Read, Write, Setting, Pipelined
Type: Integer (enumerated)
Unit: None

Selects the source of the digital Trigger.

0	"chan0trigin0", "channel0_trigger_input0": QA Channel 1, Trigger Input A.
1	"chan0trigin1", "channel0_trigger_input1": QA Channel 1, Trigger Input B.
2	"trigin0", "trigger_input0": SG Channel Trigger In 1.
3	"trigin1", "trigger_input1": SG Channel Trigger In 2.
4	"trigin2", "trigger_input2": SG Channel Trigger In 3.
5	"trigin3", "trigger_input3": SG Channel Trigger In 4.
6	"trigin4", "trigger_input4": SG Channel Trigger In 5.
7	"trigin5", "trigger_input5": SG Channel Trigger In 6.
8	"inttrig", "internal_trigger": Internal Trigger
32	"chan0seqtrig0", "channel0_sequencer_trigger0": Deprecated.
98	Deprecated.
99	Deprecated.
100	Deprecated.
101	Deprecated.
102	Deprecated.
103	"awg_marker0": Deprecated.
128	"chan0rod", "channel0_readout_done": Deprecated.
160	"awg_trigger0": Deprecated.
161	"awg_trigger1": Deprecated.
162	"awg_trigger2": Deprecated.
163	"awg_trigger3": Deprecated.
164	"awg_trigger4": Deprecated.
165	"awg_trigger5": Deprecated.
1024	"swtrig0", "software_trigger0": Software Trigger 1.

/dev..../qachannels/n/generator/clearwave

Properties: Read, Write, Pipelined
Type: Integer (64 bit)
Unit: None

Clears all waveforms in each slot and resets their length to 0.

/dev....qachannels/n/generator/delay

Properties: Read, Write, Setting, Pipelined
Type: Double
Unit: s

Sets a common delay for the start of the playback for all Waveform Memories. The resolution is 2 ns.

/dev....qachannels/n/generator/dio/valid/index

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Select the DIO bit to use as the VALID signal to indicate that a valid input is available.

/dev....qachannels/n/generator/dio/valid/polarity

Properties: Read, Write, Setting
Type: Integer (enumerated)
Unit: None

Polarity of the VALID bit that indicates that a valid input is available.

0	"none": None: VALID bit is ignored.
1	"low": Low: VALID bit must be logical zero.
2	"high": High: VALID bit must be logical high.
3	"both": Both: VALID bit may be logical high or zero.

/dev....qachannels/n/generator/elf/data

Properties: Write, Pipelined
Type: ZIVectorData
Unit: None

Accepts the data of the sequencer ELF file. If the sequence pipeliner mode is not off, the data of the ELF file goes to the staging area of the sequence pipeliner instead.

/dev....qachannels/n/generator/elf/length

Properties: Read, Pipelined
Type: Integer (64 bit)
Unit: None

Length of the compiled ELF file.

/dev....qachannels/n/generator/elf/name

Properties: Read, Pipelined
Type: ZIVectorData
Unit: None

Name of the uploaded ELF file.

/dev....qachannels/n/generator/elf/progress

Properties: Read, Pipelined
Type: Double
Unit: %

Percentage of the Sequencer program already uploaded to the device.

/dev....qachannels/n/generator/enable

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Enables the Sequencer.

/dev....qachannels/n/generator/ready

Properties: Read, Pipelined
Type: Integer (64 bit)
Unit: None

The Sequencer has a compiled program and is ready to be enabled.

/dev....qachannels/n/generator/reset

Properties: Read, Write
Type: Integer (64 bit)
Unit: None

Clears the configured Sequencer program and resets the state to not ready.

/dev....qachannels/n/generator/rtlogger/clear

Properties: Read, Write
Type: Integer (64 bit)
Unit: None

Clears the logger data.

/dev....qachannels/n/generator/rtlogger/data

Properties: Read
Type: ZIVectorData
Unit: None

Vector node with the logged events.

/dev....qachannels/n/generator/rtlogger/enable

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Activates the Real-time Logger.

/dev....qachannels/n/generator/rtlogger/input

Properties: Read, Write, Setting
Type: Integer (enumerated)
Unit: None

Select input data of logger.

0	"dio": DIO interface will be used as input.
1	"zsync": ZSync interface will be used as input.

/dev....qachannels/n/generator/rtlogger/mode

Properties: Read, Write, Setting
Type: Integer (enumerated)
Unit: None

Selects the operation mode.

- | | |
|---|--|
| 0 | "normal": Normal: Logger starts with the AWG and overwrites old values as soon as the memory limit of 1024 entries is reached. |
| 1 | "timestamp": Timestamp-triggered: Logger starts with the AWG, waits for the first valid trigger, and only starts recording data after the time specified by the starttime. Recording stops as soon as the memory limit of 1024 entries is reached. |

/dev....qachannels/n/generator/rtlogger/starttimestamp

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Timestamp at which to start logging for timestamp-triggered mode.

/dev....qachannels/n/generator/rtlogger/status

Properties: Read
Type: Integer (enumerated)
Unit: None

Operation state.

- | | |
|---|---|
| 0 | "idle": Idle: Logger is not running. |
| 1 | "normal": Normal: Logger is running in normal mode. |
| 2 | "ts_wait": Wait for timestamp: Logger is in timestamp-triggered mode and waits for start timestamp. |
| 3 | "ts_active": Active: Logger is in timestamp-triggered mode and logging. |
| 4 | "ts_full": Log Full: Logger is in timestamp-triggered mode and has stopped logging because log is full. |

/dev....qachannels/n/generator/rtlogger/timebase

Properties: Read
Type: Double
Unit: s

Minimal time difference between two timestamps. The value matches the AWG sequencer execution rate (4 ns).

/dev....qachannels/n/generator/sequencer/assembly

Properties: Read
Type: ZIVectorData
Unit: None

Displays the current sequence program in compiled form. Every line corresponds to one hardware instruction.

/dev....qachannels/n/generator/sequencer/memoryusage

Properties: Read
Type: Double
Unit: None

Size of the current Sequencer program relative to the available instruction memory of 16 kInstructions (16'384 instructions).

/dev....qachannels/n/generator/sequencer/program

Properties: Read
Type: ZIVectorData
Unit: None

Displays the source code of the current Sequencer program.

/dev....qachannels/n/generator/sequencer/status

Properties: Read
Type: Integer (64 bit)
Unit: None

Status of the Sequencer on the instrument. Bit 0: Sequencer is running; Bit 1: reserved; Bit 2: Sequencer is waiting for a trigger to arrive; Bit 3: Sequencer has detected an error; Bit 4: sequencer is waiting for synchronization with other channels.

/dev....qachannels/n/generator/sequencer/triggered

Properties: Read
Type: Integer (64 bit)
Unit: None

When at value 1, indicates that the Sequencer has been triggered.

/dev....qachannels/n/generator/single

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Puts the Sequencer into single-shot mode.

/dev....qachannels/n/generator/userregs/n

Properties: Read, Write, Setting, Pipelined
Type: Integer (64 bit)
Unit: None

Integer user register value. The sequencer has read and write access to the user register values during runtime.

/dev....qachannels/n/generator/waveforms/n/length

Properties: Read, Pipelined
Type: Integer (64 bit)
Unit: None

Length of the uploaded waveform in number of complex samples.

/dev....qachannels/n/generator/waveforms/n/wave

Properties: Read, Write, Pipelined
Type: ZIVectorData
Unit: None

Contains the generators waveforms as a vector of complex samples

/dev....qachannels/n/input/adcoverrangecount

Properties: Read
Type: Integer (64 bit)
Unit: None

Indicates the number of times the analog-to-digital converter (ADC) of the Signal Input was in an overrange condition within intervals of 200 ms. Note that this condition always occurs together with the condition of the "overrangecount" node. The overrange condition can cause potential damage of the Signal Input processing electronics, in particular the ADC units.

/dev....qachannels/n/input/digitalmixer/centerfreq

Properties: Read
Type: Double
Unit: Hz

The Center Frequency of the digital mixer for the Signal Input.

/dev....qachannels/n/input/on

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Enables the Signal Input.

/dev....qachannels/n/input/overrangecount

Properties: Read
Type: Integer (64 bit)
Unit: None

Indicates the number of times the Signal Input was in an overrange condition within intervals of 200 ms. The overrange condition can cause potential damage of the Signal Input processing electronics of the instrument.

/dev....qachannels/n/input/range

Properties: Read, Write, Setting
Type: Double
Unit: dBm

Sets the maximal Range of the Signal Input power. The instrument selects the closest available Range with a resolution of 5 dBm.

/dev....qachannels/n/input/rflfpath

Properties: Read, Write, Setting
Type: Integer (enumerated)
Unit: None

Switch between RF and LF path for the QA channel input.

0 "lf": LF path is used.
 1 "rf": RF path is used.

/dev..../qachannels/n/markers/n/source

Properties: Read, Write, Setting, Pipelined
Type: Integer (enumerated)
Unit: None

Selects the source for the marker output.

32	"chan0seqtrig0", "channel0_sequencer_trigger0": QA Channel, Sequencer Trigger Output 1.
36	"chan0seqtrig1", "channel0_sequencer_trigger1": QA Channel, Sequencer Trigger Output 2.
128	"chan0rod", "channel0_readout_done": QA Channel, Readout done.
1024	"swtrig0", "software_trigger0": Software Trigger 1.

/dev..../qachannels/n/mode

Properties: Read, Write, Setting
Type: Integer (enumerated)
Unit: None

Selects between Spectroscopy and Qubit Readout mode.

0	"spectroscopy": In Spectroscopy mode, the Signal Output is connected to the Oscillator, with which also the measured signals are correlated.
1	"readout": In Qubit Readout mode, the Signal Output is connected to the Readout Pulse Generator, and the measured signals are correlated with the Integration Weights before state discrimination.

/dev..../qachannels/n/oscs/n/freq

Properties: Read, Write, Setting, Pipelined
Type: Double
Unit: Hz

Controls the frequency of each digital Oscillator.

/dev..../qachannels/n/oscs/n/gain

Properties: Read, Write, Setting, Pipelined
Type: Double
Unit: None

Controls the gain of each digital Oscillator. The gain is defined relative to the Output Range of the Readout Channel.

/dev..../qachannels/n/output/digitalmixer/centerfreq

Properties: Read
Type: Double
Unit: Hz

The Center Frequency of the digital mixer for the Signal Output.

/dev....qachannels/n/output/filter

Properties: Read
Type: Integer (enumerated)
Unit: None

Reads the selected analog filter before the Signal Output.

0	"lowpass_1500": Low-pass filter of 1.5 GHz.
1	"lowpass_3000": Low-pass filter of 3 GHz.
2	"bandpass_3000_6000": Band-pass filter between 3 GHz - 6 GHz
3	"bandpass_6000_10000": Band-pass filter between 6 GHz - 10 GHz

/dev....qachannels/n/output/muting/enable

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Enables the muting functionality on the signal output. If enabled (1), the output noise is suppressed when the marker output is high. The default value is disabled (0).

/dev....qachannels/n/output/on

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Enables the Signal Output.

/dev....qachannels/n/output/overrangecount

Properties: Read
Type: Integer (64 bit)
Unit: None

Indicates the number of times the Signal Output was in an overrange condition within the last 200 ms. It is checked for an overrange condition every 10 ms.

/dev....qachannels/n/output/range

Properties: Read, Write, Setting
Type: Double
Unit: dBm

Sets the maximal Range of the Signal Output power. The instrument selects the closest available Range with a resolution of 5 dBm.

/dev....qachannels/n/output/rflfinterlock

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Enables the RF/LF path interlock between input and output. If enabled (1), the output path is always configured according to the input. The default value is disabled (0).

/dev....qachannels/n/output/rflfpath

Properties: Read, Write, Setting
Type: Integer (enumerated)
Unit: None

Switch between RF and LF path for the QA channel output.

0 "lf": LF path is used.
 1 "rf": RF path is used.

/dev....qachannels/n/pipeliner/availableslots

Properties: Read
Type: Integer (64 bit)
Unit: None

Number of free slots in the sequence pipeliner queue. Sequence upload is blocked if this node is 0.

/dev....qachannels/n/pipeliner/commit

Properties: Read, Write
Type: Integer (64 bit)
Unit: None

Commit node data in staging area to queue of sequence pipeliner.

/dev....qachannels/n/pipeliner/enable

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Enable execution of sequences in pipeline.

/dev....qachannels/n/pipeliner/idcurrent

Properties: Read
Type: Integer (64 bit)
Unit: None

ID of sequence in staging area.

/dev....qachannels/n/pipeliner/idrunning

Properties: Read
Type: Integer (64 bit)
Unit: None

ID of executed sequence.

/dev....qachannels/n/pipeliner/maxslots

Properties: Read
Type: Integer (64 bit)
Unit: None

Maximum number of available slots in the sequence pipeliner queue.

/dev..../qachannels/n/pipeliner/mode

Properties: Read, Write, Setting
Type: Integer (enumerated)
Unit: None

Selects the sequence pipeliner mode: off (default), batch, or queue mode. Changing the mode will reset both the sequence pipeliner and the normal AWG.

- 0 "off": Off: The sequence pipeliner is turned off.
- 1 "batch": Batch: The sequence pipeliner operates in batch mode. All sequences must be committed before the pipeliner is enabled. A batch can be executed once or multiple times.
- 2 "queue": Queue: The sequence pipeliner operates in queue mode. Sequences can be committed while the pipeliner is enabled. Every sequence is executed only once and the slot in the queue is then available for a new sequence.

/dev..../qachannels/n/pipeliner/ready

Properties: Read
Type: Integer (64 bit)
Unit: None

Indicates whether a sequence can be committed to the pipeliner.

/dev..../qachannels/n/pipeliner/repetitions/remaining

Properties: Read
Type: Integer (64 bit)
Unit: None

Number of remaining batch repetitions. This node is fixed to 1 if the sequence pipeliner is not in batch mode.

/dev..../qachannels/n/pipeliner/repetitions/value

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Number of batch repetitions (1 to 4e6). This node is fixed to 1 if the sequence pipeliner is not in batch mode.

/dev..../qachannels/n/pipeliner/reset

Properties: Read, Write
Type: Integer (64 bit)
Unit: None

Clears all sequences previously added to the sequence pipeliner and disables the pipeliner if it has been running before.

/dev..../qachannels/n/pipeliner/status

Properties: Read
Type: Integer (enumerated)
Unit: None

Status of the sequence pipeliner (0: idle, 1: executing sequence, 2: waiting for next sequence to be committed (queue mode only)

0	"idle": Idle: The sequence pipeliner is idle.
1	"exec": Executing sequence: The sequence pipeliner is executing a sequence.
2	"waiting": Waiting: The sequence pipeliner is waiting for the next sequence to be committed (queue mode only).
3	"done": Done: The sequence pipeliner is still enabled but all sequences have been executed (batch mode only).

/dev..../qachannels/n/pipeliner/timeout

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Maximal execution time per sequence in milliseconds. The execution of a sequence is aborted if the maximal execution time is reached. A value of 0 means infinity.

/dev..../qachannels/n/readout/discriminators/n/threshold

Properties: Read, Write, Setting, Pipelined
Type: Double
Unit: None

Sets the threshold level for the 2-state discriminator on the real signal axis in Vs.

/dev..../qachannels/n/readout/integration/clearweight

Properties: Read, Write, Pipelined
Type: Integer (64 bit)
Unit: None

Clears all integration weights by setting them to 0.

/dev..../qachannels/n/readout/integration/delay

Properties: Read, Write, Setting, Pipelined
Type: Double
Unit: s

Sets a common delay for the start of the readout integration for all Integration Weights with respect to the time when the trigger is received. The resolution is 2 ns.

/dev..../qachannels/n/readout/integration/length

Properties: Read, Write, Setting, Pipelined
Type: Integer (64 bit)
Unit: None

Sets the length of all Integration Weights in number of samples. A maximum of 4096 samples can be integrated, which corresponds to 2.05 μ s.

/dev..../qachannels/n/readout/integration/weights/n/wave

Properties: Read, Write, Pipelined
Type: ZIVectorData
Unit: None

Contains the complex-valued waveform of the Integration Weight. The valid range is between -1.0 and +1.0 for both the real and imaginary part.

/dev..../qachannels/n/readout/multistate/clear

Properties: Read, Write, Pipelined
Type: Integer (64 bit)
Unit: None

Clears all settings related to the multi-state qudit discrimination mode.

/dev..../qachannels/n/readout/multistate/dio/bits/n/source

Properties: Read, Write, Setting, Pipelined
Type: Integer (enumerated)
Unit: None

Sets qudit discrimination bit source to transmit at this DIO bit position.

0	"qudit_0_bit_0": Qudit 0, bit 0
1	"qudit_0_bit_1": Qudit 0, bit 1
2	"qudit_1_bit_0": Qudit 1, bit 0
3	"qudit_1_bit_1": Qudit 1, bit 1
4	"qudit_2_bit_0": Qudit 2, bit 0
5	"qudit_2_bit_1": Qudit 2, bit 1
6	"qudit_3_bit_0": Qudit 3, bit 0
7	"qudit_3_bit_1": Qudit 3, bit 1
8	"qudit_4_bit_0": Qudit 4, bit 0
9	"qudit_4_bit_1": Qudit 4, bit 1
10	"qudit_5_bit_0": Qudit 5, bit 0
11	"qudit_5_bit_1": Qudit 5, bit 1
12	"qudit_6_bit_0": Qudit 6, bit 0
13	"qudit_6_bit_1": Qudit 6, bit 1
14	"qudit_7_bit_0": Qudit 7, bit 0
15	"qudit_7_bit_1": Qudit 7, bit 1
16	"qudit_8_bit_0": Qudit 8, bit 0 (requires SHFQC-16W option)
17	"qudit_8_bit_1": Qudit 8, bit 1 (requires SHFQC-16W option)
18	"qudit_9_bit_0": Qudit 9, bit 0 (requires SHFQC-16W option)
19	"qudit_9_bit_1": Qudit 9, bit 1 (requires SHFQC-16W option)
20	"qudit_10_bit_0": Qudit 10, bit 0 (requires SHFQC-16W option)
21	"qudit_10_bit_1": Qudit 10, bit 1 (requires SHFQC-16W option)
22	"qudit_11_bit_0": Qudit 11, bit 0 (requires SHFQC-16W option)
23	"qudit_11_bit_1": Qudit 11, bit 1 (requires SHFQC-16W option)
24	"qudit_12_bit_0": Qudit 12, bit 0 (requires SHFQC-16W option)
25	"qudit_12_bit_1": Qudit 12, bit 1 (requires SHFQC-16W option)
26	"qudit_13_bit_0": Qudit 13, bit 0 (requires SHFQC-16W option)
27	"qudit_13_bit_1": Qudit 13, bit 1 (requires SHFQC-16W option)
28	"qudit_14_bit_0": Qudit 14, bit 0 (requires SHFQC-16W option)
29	"qudit_14_bit_1": Qudit 14, bit 1 (requires SHFQC-16W option)
30	"qudit_15_bit_0": Qudit 15, bit 0 (requires SHFQC-16W option)
31	"qudit_15_bit_1": Qudit 15, bit 1 (requires SHFQC-16W option)
32	"fixed_0": Fixed 0
34	"fixed_1": Fixed 1

/dev..../qachannels/n/readout/multistate/dio/packed

Properties: Read, Write, Setting, Pipelined
Type: Integer (64 bit)
Unit: None

Enables compact bit packing for multi-state qudit discrimination values over DIO, where DIO bits are assigned to qudit bits by increasing qudit index using only one bit for qubits or two bits for qutrits and ququads. The **source** node becomes read-only in packed mode.

/dev..../qachannels/n/readout/multistate/enable

Properties: Read, Write, Setting, Pipelined
Type: Integer (64 bit)
Unit: None

Enables multi-state qudit discrimination mode.

/dev..../qachannels/n/readout/multistate/integratorcount

Properties: Read, Pipelined
Type: Integer (64 bit)
Unit: None

Number of integrators used by the current multi-state discrimination qudit configuration.

/dev..../qachannels/n/readout/multistate/integratorusage

Properties: Read, Pipelined
Type: Double
Unit: None

Percentage of integrators used by the current multi-state discrimination qudit configuration.

/dev..../qachannels/n/readout/multistate/qudits/n/assignmentvec

Properties: Read, Write, Pipelined
Type: ZIVectorData
Unit: None

Assignment matrix for qudit discrimination. The vector should contain $2^{(d * (d - 1) / 2)}$ elements, where d is the maximum number of states for the qudit.

/dev..../qachannels/n/readout/multistate/qudits/n/available

Properties: Read, Pipelined
Type: Integer (64 bit)
Unit: None

Indicates whether enough integrators are available to allow enabling the qudit in its current configuration.

/dev..../qachannels/n/readout/multistate/qudits/n/enable

Properties: Read, Write, Setting, Pipelined
Type: Integer (64 bit)
Unit: None

Enables the multi-state qudit.

/dev..../qachannels/n/readout/multistate/qudits/n/integrator/indexvec

Properties: Read, Pipelined
Type: ZIVectorData
Unit: None

List of integrator indices used by the qudit.

/dev..../qachannels/n/readout/multistate/qudits/n/integrator/startindex

Properties: Read, Pipelined
Type: Integer (64 bit)
Unit: None

Index of the first integrator used by the qudit.

/dev..../qachannels/n/readout/multistate/qudits/n/numstates

Properties: Read, Write, Setting, Pipelined
Type: Integer (64 bit)
Unit: None

Maximum number of states for the qudit. The value must be within the range [2, 4] (inclusive).

/dev..../qachannels/n/readout/multistate/qudits/n/thresholds/n/value

Properties: Read, Write, Setting, Pipelined
Type: Double
Unit: None

Sets the threshold used to discriminate the qudit. Only the first $(d * (d - 1) / 2)$ thresholds will be used, where d is the maximum number of states for the qudit.

/dev..../qachannels/n/readout/multistate/qudits/n/weights/n/wave

Properties: Read, Write, Pipelined
Type: ZIVectorData
Unit: None

Contains the complex-valued waveform of the Integration Weights for the qudit. The valid range is between -1.0 and +1.0 for both the real and imaginary part. Only the first $d - 1$ waves are used, where d is the maximum number of states for the qudit.

/dev..../qachannels/n/readout/multistate/valid

Properties: Read, Pipelined
Type: Integer (64 bit)
Unit: None

Indicates whether the current multi-state qudit configuration allows for all qudits that are set to be enabled to actually be enabled, i.e. whether the enabled qudits require at most the available number of integrators.

/dev..../qachannels/n/readout/multistate/zsync/bits/n/source

Properties: Read, Write, Setting, Pipelined
Type: Integer (enumerated)
Unit: None

Sets qudit discrimination bit source to transmit at this ZSync bit position.

0	"qudit_0_bit_0": Qudit 0, bit 0
1	"qudit_0_bit_1": Qudit 0, bit 1
2	"qudit_1_bit_0": Qudit 1, bit 0
3	"qudit_1_bit_1": Qudit 1, bit 1
4	"qudit_2_bit_0": Qudit 2, bit 0
5	"qudit_2_bit_1": Qudit 2, bit 1
6	"qudit_3_bit_0": Qudit 3, bit 0
7	"qudit_3_bit_1": Qudit 3, bit 1
8	"qudit_4_bit_0": Qudit 4, bit 0
9	"qudit_4_bit_1": Qudit 4, bit 1
10	"qudit_5_bit_0": Qudit 5, bit 0
11	"qudit_5_bit_1": Qudit 5, bit 1
12	"qudit_6_bit_0": Qudit 6, bit 0
13	"qudit_6_bit_1": Qudit 6, bit 1
14	"qudit_7_bit_0": Qudit 7, bit 0
15	"qudit_7_bit_1": Qudit 7, bit 1
16	"qudit_8_bit_0": Qudit 8, bit 0 (requires SHFQC-16W option)
17	"qudit_8_bit_1": Qudit 8, bit 1 (requires SHFQC-16W option)
18	"qudit_9_bit_0": Qudit 9, bit 0 (requires SHFQC-16W option)
19	"qudit_9_bit_1": Qudit 9, bit 1 (requires SHFQC-16W option)
20	"qudit_10_bit_0": Qudit 10, bit 0 (requires SHFQC-16W option)
21	"qudit_10_bit_1": Qudit 10, bit 1 (requires SHFQC-16W option)
22	"qudit_11_bit_0": Qudit 11, bit 0 (requires SHFQC-16W option)
23	"qudit_11_bit_1": Qudit 11, bit 1 (requires SHFQC-16W option)
24	"qudit_12_bit_0": Qudit 12, bit 0 (requires SHFQC-16W option)
25	"qudit_12_bit_1": Qudit 12, bit 1 (requires SHFQC-16W option)
26	"qudit_13_bit_0": Qudit 13, bit 0 (requires SHFQC-16W option)
27	"qudit_13_bit_1": Qudit 13, bit 1 (requires SHFQC-16W option)
28	"qudit_14_bit_0": Qudit 14, bit 0 (requires SHFQC-16W option)
29	"qudit_14_bit_1": Qudit 14, bit 1 (requires SHFQC-16W option)
30	"qudit_15_bit_0": Qudit 15, bit 0 (requires SHFQC-16W option)
31	"qudit_15_bit_1": Qudit 15, bit 1 (requires SHFQC-16W option)
32	"fixed_0": Fixed 0. Note: this bit will not be forwarded by the PQSC since it is marked as invalid result.
34	"fixed_1": Fixed 1. Note: this bit will not be forwarded by the PQSC since it is marked as invalid result.

/dev..../qachannels/n/readout/multistate/zsync/packed

Properties: Read, Write, Setting, Pipelined
Type: Integer (64 bit)
Unit: None

Enables compact bit packing for multi-state qudit discrimination values over ZSync, where zSync bits are assigned to qudit bits by increasing qudit index using only one bit for qubits or two bits for qutrits and ququads. The **source** node becomes read-only in packed mode.

/dev..../qachannels/n/readout/result/acquired

Properties: Read, Pipelined
Type: Integer (64 bit)
Unit: None

Indicates the index of the acquisition that will be performed on the next trigger.

/dev....qachannels/n/readout/result/averages

Properties: Read, Write, Setting, Pipelined
Type: Integer (64 bit)
Unit: None

Number of measurements that are averaged.

/dev....qachannels/n/readout/result/data/n/wave

Properties: Read, Pipelined
Type: ZIVectorData
Unit: None

Acquired result data. Depending on the source of the data, the data can be complex- or integer-valued.

/dev....qachannels/n/readout/result/enable

Properties: Read, Write, Setting, Pipelined
Type: Integer (64 bit)
Unit: None

Enables the acquisition of readout results.

/dev....qachannels/n/readout/result/error/clear

Properties: Read, Write, Pipelined
Type: Integer (64 bit)
Unit: None

Writing to this node clears the hold-off error count.

/dev....qachannels/n/readout/result/error/count

Properties: Read, Pipelined
Type: Integer (64 bit)
Unit: None

Number of hold-off errors detected.

/dev....qachannels/n/readout/result/error/jobidx

Properties: Read, Pipelined
Type: Integer (64 bit)
Unit: None

Job index for which last hold-off errors were detected.

/dev....qachannels/n/readout/result/length

Properties: Read, Write, Setting, Pipelined
Type: Integer (64 bit)
Unit: None

Number of data points to record. One data point corresponds to a single averaged result value of the selected source.

/dev....qachannels/n/readout/result/mode

Properties: Read, Write, Setting, Pipelined
Type: Integer (enumerated)
Unit: None

Selects the averaging order of the result.

- | | |
|---|--|
| 0 | "cyclic": Cyclic averaging: a sequence of multiple results is recorded first, then averaged with the next repetition of the same sequence. |
| 1 | "sequential": Sequential averaging: each result is recorded and averaged first, before the next result is recorded and averaged. |

/dev....qachannels/n/readout/result/overdio

Properties: Read
Type: Integer (64 bit)
Unit: None

Number of DIO interface overflows during readout. This can happen if readouts are triggered faster than the maximum possible data-rate of the DIO interface.

/dev....qachannels/n/readout/result/source

Properties: Read, Write, Setting, Pipelined
Type: Integer (enumerated)
Unit: None

Selects the signal source of the Result Logger.

- | | |
|---|--|
| 1 | "result_of_integration": Complex-valued integration results of the Weighted Integration in Qubit Readout mode. |
| 3 | "result_of_discrimination": The results after state discrimination. |

/dev....qachannels/n/spectroscopy/delay

Properties: Read, Write, Setting, Pipelined
Type: Double
Unit: s

Sets the delay of the integration in Spectroscopy mode with respect to the Trigger signal. The resolution is 2 ns.

/dev....qachannels/n/spectroscopy/envelope/delay

Properties: Read, Write, Setting, Pipelined
Type: Double
Unit: s

Sets the delay of the envelope waveform in Spectroscopy mode with respect to the Trigger signal. The resolution is 2 ns.

/dev....qachannels/n/spectroscopy/envelope/enable

Properties: Read, Write, Setting, Pipelined
Type: Integer (64 bit)
Unit: None

Enables the modulation of the oscillator signal with the complex envelope waveform.

/dev....qachannels/n/spectroscopy/envelope/length

Properties: Read, Pipelined
Type: Integer (64 bit)
Unit: None

Length of the uploaded envelope waveform in number of complex samples.

/dev....qachannels/n/spectroscopy/envelope/wave

Properties: Read, Write, Pipelined
Type: ZIVectorData
Unit: None

Contains the envelope waveform as a vector of complex samples.

/dev....qachannels/n/spectroscopy/length

Properties: Read, Write, Setting, Pipelined
Type: Integer (64 bit)
Unit: None

Sets the integration length in Spectroscopy mode in number of samples. Up to 33.5 MSa (2^{25} samples) can be recorded, which corresponds to 16.7 ms.

/dev....qachannels/n/spectroscopy/psd/enable

Properties: Read, Write, Setting, Pipelined
Type: Integer (64 bit)
Unit: None

Enable the power spectral density mode of the spectroscopy mode.

/dev....qachannels/n/spectroscopy/psd/error/clear

Properties: Read, Write, Pipelined
Type: Integer (64 bit)
Unit: None

Clears power spectral density error flags.

/dev....qachannels/n/spectroscopy/psd/error/overflow

Properties: Read, Pipelined
Type: Integer (64 bit)
Unit: None

Indicates whether overflow happened while calculating the power spectral density.

/dev....qachannels/n/spectroscopy/result/acquired

Properties: Read, Pipelined
Type: Integer (64 bit)
Unit: None

Indicates the index of the acquisition that will be performed on the next trigger.

/dev....qachannels/n/spectroscopy/result/averages

Properties: Read, Write, Setting, Pipelined
Type: Integer (64 bit)
Unit: None

Number of measurements that are averaged.

/dev....qachannels/n/spectroscopy/result/data/wave

Properties: Read, Pipelined
Type: ZIVectorData
Unit: None

Acquired complex spectroscopy result data.

/dev....qachannels/n/spectroscopy/result/enable

Properties: Read, Write, Setting, Pipelined
Type: Integer (64 bit)
Unit: None

Enables the acquisition of spectroscopy results.

/dev....qachannels/n/spectroscopy/result/error/clear

Properties: Read, Write, Pipelined
Type: Integer (64 bit)
Unit: None

Writing to this node clears the hold-off error count.

/dev....qachannels/n/spectroscopy/result/error/count

Properties: Read, Pipelined
Type: Integer (64 bit)
Unit: None

Number of hold-off errors detected.

/dev....qachannels/n/spectroscopy/result/error/jobidx

Properties: Read, Pipelined
Type: Integer (64 bit)
Unit: None

Job index for which last hold-off errors were detected.

/dev....qachannels/n/spectroscopy/result/length

Properties: Read, Write, Setting, Pipelined
Type: Integer (64 bit)
Unit: None

Number of data points to record. One data point corresponds to a single averaged result value of the selected source.

/dev....qachannels/n/spectroscopy/result/mode

Properties: Read, Write, Setting, Pipelined
Type: Integer (enumerated)
Unit: None

Selects the averaging order of the result.

- 0 "cyclic": Cyclic averaging: a sequence of multiple results is recorded first, then averaged with the next repetition of the same sequence.
- 1 "sequential": Sequential averaging: each result is recorded and averaged first, before the next result is recorded and averaged.

/dev....qachannels/n/spectroscopy/trigger/channel

Properties: Read, Write, Setting, Pipelined
Type: Integer (enumerated)
Unit: None

Selects the source of the trigger for the integration and envelope in Spectroscopy mode.

0	"chan0trigin0", "channel0_trigger_input0": Channel 1, Trigger Input A.
1	"chan0trigin1", "channel0_trigger_input1": Channel 1, Trigger Input B.
8	"inttrig", "internal_trigger": Internal Trigger
32	"chan0seqtrig0", "channel0_sequencer_trigger0": Channel 1, Sequencer Trigger Output 1.
36	"chan0seqtrig1", "channel0_sequencer_trigger1": Channel 1, Sequencer Trigger Output 2.
1024	"swtrig0", "software_trigger0": Software Trigger 1.

/dev....qachannels/n/synchronization/enable

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Enable multi-channel synchronization for this channel. The program will only execute once all channels with enabled synchronization are ready.

/dev....qachannels/n/triggers/n/imp50

Properties: Read, Write, Setting
Type: Integer (enumerated)
Unit: None

Trigger Input impedance: When on, the Trigger Input impedance is 50 Ohm; when off, 1 kOhm.

0	"1_kOhm": OFF: 1 k Ohm
1	"50_Ohm": ON: 50 Ohm

/dev....qachannels/n/triggers/n/level

Properties: Read, Write, Setting
Type: Double
Unit: V

Defines the analog Trigger level.

/dev....qachannels/n/triggers/n/value

Properties: Read
Type: Integer (64 bit)
Unit: None

Shows the value of the digital Trigger Input. The value is integrated over a period of 100 ms. Values are: 1: low; 2: high; 3: was low and high in the period.

7.2.5. SCOPES

/dev....scopes/n/averaging/count

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Configures the number of Scope measurements to average.

/dev..../scopes/n/averaging/enable

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Enables averaging of Scope measurements.

/dev..../scopes/n/channels/n/enable

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: Dependent

Enables recording for this Scope channel.

/dev..../scopes/n/channels/n/inputselect

Properties: Read, Write, Setting
Type: Integer (enumerated)
Unit: None

Selects the scope input signal.

0	"chan0signin", "channel0_signal_input": Signal Input Channel 1.
48	"sgchan0sigout", "sgchannel0_signal_output": Signal Output SG Channel 1.
49	"sgchan1sigout", "sgchannel1_signal_output": Signal Output SG Channel 2.
50	"sgchan2sigout", "sgchannel2_signal_output": Signal Output SG Channel 3.
51	"sgchan3sigout", "sgchannel3_signal_output": Signal Output SG Channel 4.
52	"sgchan4sigout", "sgchannel4_signal_output": Signal Output SG Channel 5.
53	"sgchan5sigout", "sgchannel5_signal_output": Signal Output SG Channel 6.

/dev..../scopes/n/channels/n/wave

Properties: Read
Type: ZIVectorData
Unit: Dependent

Contains the acquired Scope measurement data.

/dev..../scopes/n/enable

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Enables the acquisition of Scope shots. Goes back to 0 (disabled) after the scope shot has been acquired.

/dev..../scopes/n/length

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Defines the length of the recorded Scope shot in number of samples.

/dev..../scopes/n/segments/count

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Specifies the number of segments to be recorded in device memory. The maximum Scope shot size is given by the available memory divided by the number of segments.

/dev..../scopes/n/segments/enable

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Enable segmented Scope recording. This allows for full bandwidth recording of Scope shots with a minimum dead time between individual shots.

/dev..../scopes/n/single

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Puts the Scope into single shot mode.

/dev..../scopes/n/time

Properties: Read, Write, Setting
Type: Integer (enumerated)
Unit: None

Defines the time base of the Scope.

0 2 GHz

/dev..../scopes/n/trigger/channel

Properties: Read, Write, Setting
Type: Integer (enumerated)
Unit: None

Selects the trigger source signal.

0 "chan0trigin0", "channel0_trigger_input0": Channel 1, Trigger Input A.
 1 "chan0trigin1", "channel0_trigger_input1": Channel 1, Trigger Input B.
 8 "intrig", "internal_trigger": Internal Trigger
 32 "chan0seqtrig0", "channel0_sequencer_trigger0": Channel 1, Sequencer Trigger
 Output 1.
 36 "chan0seqtrig1", "channel0_sequencer_trigger1": Channel 1, Sequencer Trigger
 Output 2.
 64 "chan0seqmon0", "channel0_sequencer_monitor0": Channel 1, Sequencer
 Monitor Trigger.
 1024 "swtrig0", "software_trigger0": Software Trigger 1.

/dev..../scopes/n/trigger/delay

Properties: Read, Write, Setting
Type: Double
Unit: s

The delay of a Scope measurement. A negative delay results in data being acquired before the trigger point. The resolution is 2 ns.

/dev..../scopes/n/trigger/enable

Properties: Read, Write, Setting
Type: Integer (enumerated)
Unit: None

When triggering is enabled scope data are acquired every time the defined trigger condition is met.

0 "off": OFF: Continuous scope shot acquisition
 1 "on": ON: Trigger based scope shot acquisition

7.2.6. SGCHANNELS

/dev..../sgchannels/n/awg/auxtriggers/n/channel

Properties: Read, Write, Setting
Type: Integer (enumerated)
Unit: None

Selects the digital trigger source signal.

0	"trigin0", "trigger_input0": Trigger In 1
1	"trigin1", "trigger_input1": Trigger In 2
2	"trigin2", "trigger_input2": Trigger In 3
3	"trigin3", "trigger_input3": Trigger In 4
4	"trigin4", "trigger_input4": Trigger In 5
5	"trigin5", "trigger_input5": Trigger In 6
6	"trigin6", "trigger_input6": Trigger In 7
7	"trigin7", "trigger_input7": Trigger In 8
8	"inttrig", "internal_trigger": Internal Trigger
32	"chan0seqtrig0", "channel0_sequencer_trigger0": QA Channel 1, Sequencer Trigger Output 1.
36	"chan0seqtrig1", "channel0_sequencer_trigger1": QA Channel 1, Sequencer Trigger Output 2.
128	"chan0rod", "channel0_readout_done": QA Channel 1, Readout done.

/dev..../sgchannels/n/awg/auxtriggers/n/slope

Properties: Read, Write, Setting
Type: Integer (enumerated)
Unit: None

Select the signal edge that should activate the trigger. The trigger will be level sensitive when the Level option is selected.

0	"level_sensitive": Level sensitive trigger
1	"rising_edge": Rising edge trigger
2	"falling_edge": Falling edge trigger
3	"both_edges": Rising or falling edge trigger

/dev..../sgchannels/n/awg/auxtriggers/n/state

Properties: Read
Type: Integer (64 bit)
Unit: None

State of the Auxiliary Trigger: No trigger detected/trigger detected.

/dev..../sgchannels/n/awg/commandtable/clear

Properties: Read, Write
Type: Integer (64 bit)
Unit: None

Writing to this node clears all data previously loaded to the command table of the device. If the sequence pipeliner mode is not off, the command table in the sequence pipeliner staging area is cleared instead.

/dev..../sgchannels/n/awg/commandtable/data

Properties: Read, Write, Pipelined
Type: ZIVectorData
Unit: None

Data contained in the command table in JSON format.

/dev..../sgchannels/n/awg/commandtable/schema

Properties: Read
Type: ZIVectorData
Unit: None

JSON schema describing the command table JSON format (read-only).

/dev..../sgchannels/n/awg/commandtable/status

Properties: Read, Pipelined
Type: Integer (64 bit)
Unit: None

Status of the command table on the instrument. If the sequence pipeliner mode is not off, the status of the command table in the sequence pipeliner staging area is shown instead. Bit 0: data uploaded to the command table; Bit 1, Bit 2: reserved; Bit 3: uploading of data to the command table failed due to a JSON parsing error.

/dev..../sgchannels/n/awg/dio/error/timing

Properties: Read
Type: Integer (64 bit)
Unit: None

A 32-bit value indicating which bits on the DIO interface may have timing errors. A timing error is defined as an event where either the VALID or any of the data bits on the DIO interface change value at the same time as the STROBE bit.

/dev..../sgchannels/n/awg/dio/error/width

Properties: Read
Type: Integer (64 bit)
Unit: None

Indicates a width (i.e. jitter) error on either the STROBE (bit 0 of the value) or VALID bit (bit 1 of the result). A width error indicates that there was jitter detected on the given bit, meaning that an active period was either shorter or longer than the configured expected width.

/dev..../sgchannels/n/awg/dio/highbits

Properties: Read
Type: Integer (64 bit)
Unit: None

32-bit value indicating which bits on the 32-bit interface are detected as having a logic high value.

/dev..../sgchannels/n/awg/dio/lowbits

Properties: Read
Type: Integer (64 bit)
Unit: None

32-bit value indicating which bits on the 32-bit interface are detected as having a logic low value.

/dev..../sgchannels/n/awg/dio/mask/shift

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Defines the amount of bit shifting to apply for the DIO wave selection in connection with `playWaveDIO()`.

/dev..../sgchannels/n/awg/dio/mask/value

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Selects the DIO bits to be used for waveform selection in connection with playWaveDIO().

/dev..../sgchannels/n/awg/dio/state

Properties: Read
Type: Integer (64 bit)
Unit: None

When asserted, indicates that triggers are generated from the DIO interface to the AWG.

/dev..../sgchannels/n/awg/dio/strobe/index

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Select the DIO bit to use as the STROBE signal.

/dev..../sgchannels/n/awg/dio/strobe/slope

Properties: Read, Write, Setting
Type: Integer (enumerated)
Unit: None

Select the signal edge of the STROBE signal for use in timing alignment.

0	"off": Off
1	"rising_edge": Rising edge trigger
2	"falling_edge": Falling edge trigger
3	"both_edges": Rising or falling edge trigger

/dev..../sgchannels/n/awg/dio/strobe/width

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Specifies the expected width of active pulses on the STROBE bit.

/dev..../sgchannels/n/awg/dio/valid/index

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Select the DIO bit to use as the VALID signal to indicate a valid input is available.

/dev..../sgchannels/n/awg/dio/valid/polarity

Properties: Read, Write, Setting
Type: Integer (enumerated)
Unit: None

Polarity of the VALID bit that indicates that a valid input is available.

0	"none": None: VALID bit is ignored.
1	"low": Low: VALID bit must be logical zero.
2	"high": High: VALID bit must be logical high.
3	"both": Both: VALID bit may be logical high or zero.

/dev..../sgchannels/n/awg/dio/valid/width

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Expected width of an active pulse on the VALID bit.

/dev..../sgchannels/n/awg/diozsyncswitch

Properties: Read, Write, Setting
Type: Integer (enumerated)
Unit: None

Defines which interface input to use with this AWG

0 "dio": DIO interface will be used as input.
 1 "zsync": ZSync interface will be used as input.

/dev..../sgchannels/n/awg/elf/checksum

Properties: Read, Pipelined
Type: Integer (64 bit)
Unit: None

Checksum of the uploaded ELF file.

/dev..../sgchannels/n/awg/elf/data

Properties: Write, Pipelined
Type: ZIVectorData
Unit: None

Accepts the data of the sequencer ELF file. If the sequence pipeliner mode is not off, the data of the ELF file goes to the staging area of the sequence pipeliner instead.

/dev..../sgchannels/n/awg/elf/length

Properties: Read, Pipelined
Type: Integer (64 bit)
Unit: None

Length of the compiled ELF file.

/dev..../sgchannels/n/awg/elf/memoryusage

Properties: Read, Pipelined
Type: Double
Unit: None

Size of the uploaded ELF file relative to the size of the main memory.

/dev..../sgchannels/n/awg/elf/name

Properties: Read, Pipelined
Type: ZIVectorData
Unit: None

The name of the uploaded ELF file.

/dev..../sgchannels/n/awg/elf/progress

Properties: Read, Pipelined
Type: Double
Unit: %

The percentage of the sequencer program already uploaded to the device.

/dev..../sgchannels/n/awg/enable

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Activates the AWG.

/dev..../sgchannels/n/awg/modulation/enable

Properties: Read, Write, Setting
Type: Integer (enumerated)
Unit: None

Enable or disable digital modulation.

0 "off": Modulation off
 1 "on": Modulation on

/dev..../sgchannels/n/awg/outputamplitude

Properties: Read, Write, Setting
Type: Double
Unit: None

Amplitude scale factor applied to both AWG outputs.

/dev..../sgchannels/n/awg/outputs/n/enables/n

Properties: Read
Type: Integer (64 bit)
Unit: None

Indicates the routing of the AWG signal (k index) to the digital mixer input (m index).

/dev..../sgchannels/n/awg/outputs/n/gains/n

Properties: Read, Write, Setting
Type: Double
Unit: None

Gain factor applied to the AWG Output at the given output multiplier stage. The final signal amplitude is proportional to the Range voltage setting of the Wave signal outputs.

/dev..../sgchannels/n/awg/outputs/n/hold

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Keep the last sample (constant) on the output even after the waveform program finishes.

/dev..../sgchannels/n/awg/ready

Properties: Read, Pipelined
Type: Integer (64 bit)
Unit: None

A value of True means that the AWG has a compiled waveform and is ready to be enabled. If the sequence pipeliner is not off, a value of True means that the sequence in the staging area is ready to be committed to the pipeline.

/dev..../sgchannels/n/awg/reset

Properties: Read, Write
Type: Integer (64 bit)
Unit: None

Clears the configured AWG program and resets the state to not ready.

/dev..../sgchannels/n/awg/rtlogger/clear

Properties: Read, Write
Type: Integer (64 bit)
Unit: None

Clears the logger data.

/dev..../sgchannels/n/awg/rtlogger/data

Properties: Read
Type: ZIVectorData
Unit: None

Vector node with the logged events.

/dev..../sgchannels/n/awg/rtlogger/enable

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Activates the Real-time Logger.

/dev..../sgchannels/n/awg/rtlogger/input

Properties: Read, Write, Setting
Type: Integer (enumerated)
Unit: None

Select input data of logger.

0	"dio": DIO interface will be used as input.
1	"zsync": ZSync interface will be used as input.

/dev..../sgchannels/n/awg/rtlogger/mode

Properties: Read, Write, Setting
Type: Integer (enumerated)
Unit: None

Selects the operation mode.

- 0 "normal": Normal: Logger starts with the AWG and overwrites old values as soon as the memory limit of 1024 entries is reached.
- 1 "timestamp": Timestamp-triggered: Logger starts with the AWG, waits for the first valid trigger, and only starts recording data after the time specified by the starttime. Recording stops as soon as the memory limit of 1024 entries is reached.

/dev..../sgchannels/n/awg/rtlogger/starttimestamp

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Timestamp at which to start logging for timestamp-triggered mode.

/dev..../sgchannels/n/awg/rtlogger/status

Properties: Read
Type: Integer (enumerated)
Unit: None

Operation state.

- 0 "idle": Idle: Logger is not running.
- 1 "normal": Normal: Logger is running in normal mode.
- 2 "ts_wait": Wait for timestamp: Logger is in timestamp-triggered mode and waits for start timestamp.
- 3 "ts_active": Active: Logger is in timestamp-triggered mode and logging.
- 4 "ts_full": Log Full: Logger is in timestamp-triggered mode and has stopped logging because log is full.

/dev..../sgchannels/n/awg/rtlogger/timebase

Properties: Read
Type: Double
Unit: s

Minimal time difference between two timestamps. The value matches the AWG sequencer execution rate (4 ns)

/dev..../sgchannels/n/awg/sequencer/assembly

Properties: Read
Type: ZIVectorData
Unit: None

Displays the current sequence program in compiled form. Every line corresponds to one hardware instruction.

/dev..../sgchannels/n/awg/sequencer/memoryusage

Properties: Read
Type: Double
Unit: None

Size of the current Sequencer program relative to the available instruction memory of 32 kInstructions (32'768 instructions).

/dev..../sgchannels/n/awg/sequencer/pc

Properties: Read
Type: Integer (64 bit)
Unit: None

Current position in the list of sequence instructions during execution.

/dev..../sgchannels/n/awg/sequencer/program

Properties: Read
Type: ZIVectorData
Unit: None

Displays the source code of the current sequence program.

/dev..../sgchannels/n/awg/sequencer/status

Properties: Read
Type: Integer (64 bit)
Unit: None

Status of the sequencer on the instrument. Bit 0: sequencer is running; Bit 1: reserved; Bit 2: sequencer is waiting for a trigger to arrive; Bit 3: AWG has detected an error; Bit 4: sequencer is waiting for synchronization with other channels.

/dev..../sgchannels/n/awg/sequencer/triggered

Properties: Read
Type: Integer (64 bit)
Unit: None

When 1, indicates that the AWG Sequencer has been triggered.

/dev..../sgchannels/n/awg/single

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Puts the AWG into single shot mode.

/dev..../sgchannels/n/awg/time

Properties: Read, Write, Setting
Type: Integer (enumerated)
Unit: None

AWG sampling rate. The numeric values here are equal to the base sampling rate of 2.0 GHz divided by 2^n , where n is the node value. This value is used by default and can be overridden in the Sequence program.

0	2.0 GHz
1	1.0 GHz
2	500 MHz
3	250 MHz
4	125 MHz
5	62.50 MHz
6	31.25 MHz
7	15.63 MHz
8	7.81 MHz
9	3.91 MHz
10	1.95 MHz
11	976.56 kHz
12	488.28 kHz
13	244.14 kHz

/dev..../sgchannels/n/awg/userregs/n

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Integer user register value. The sequencer has reading and writing access to the user register values during run time.

/dev..../sgchannels/n/awg/waveform/descriptors

Properties: Read
Type: ZIVectorData
Unit: None

JSON-formatted string containing a dictionary of various properties of the current waveform: name, filename, function, channels, marker bits, length, timestamp.

/dev..../sgchannels/n/awg/waveform/memoryusage

Properties: Read
Type: Double
Unit: %

Amount of the used waveform data relative to the device memory. The memory provides space for 96 kSa (98'304 Sa) of dual-channel waveform data.

/dev..../sgchannels/n/awg/waveform/playing

Properties: Read
Type: Integer (64 bit)
Unit: None

When 1, indicates if a waveform is being played currently.

/dev..../sgchannels/n/awg/waveform/waves/n

Properties: Read, Write, Pipelined
Type: ZIVectorData
Unit: None

The waveform data in the instrument's native format for the given playWave waveform index. This node will not work with subscribe as it does not push updates. For short vectors get may be used. For long vectors (causing get to time out) getAsEvent and poll can be used. The index of the waveform to be replaced can be determined using the Waveform sub-tab in the AWG tab of the LabOne User Interface.

/dev..../sgchannels/n/busy

Properties: Read
Type: Integer (64 bit)
Unit: None

Indicates that the channel is busy applying settings, e.g., center-frequency or analog output settings.

/dev..../sgchannels/n/centerfreq

Properties: Read
Type: Double
Unit: Hz

The Center Frequency of signal generation band. This value is read-only. Frequency is set through synthesizer node.

/dev..../sgchannels/n/digitalmixer/centerfreq

Properties: Read, Write, Setting
Type: Double
Unit: Hz

Set center frequency of digital mixer.

/dev..../sgchannels/n/internal

Properties: Read
Type: Integer (64 bit)
Unit: None

Indicates if an SG-channel is internal 1, otherwise 0. An internal channel is not equipped with an upconverter and needs to be routed to a channel with an upconverter.

/dev..../sgchannels/n/marker/source

Properties: Read, Write, Setting
Type: Integer (enumerated)
Unit: None

Assign a signal to a marker.

0	"awg_trigger0": Trigger output is assigned to AWG Trigger 1, controlled by AWG sequencer commands.
1	"awg_trigger1": Trigger output is assigned to AWG Trigger 2, controlled by AWG sequencer commands.
2	"awg_trigger2": Trigger output is assigned to AWG Trigger 3, controlled by AWG sequencer commands.
3	"awg_trigger3": Trigger output is assigned to AWG Trigger 4, controlled by AWG sequencer commands.
4	"output0_marker0": Output is assigned to I component Marker 1.
5	"output0_marker1": Output is assigned to I component Marker 2.
6	"output1_marker0": Output is assigned to Q component Marker 1.
7	"output1_marker1": Output is assigned to Q component Marker 2.
8	"trigin0", "trigger_input0": Output is assigned to Trigger Input 1.
9	"trigin1", "trigger_input1": Output is assigned to Trigger Input 2.
10	"trigin2", "trigger_input2": Output is assigned to Trigger Input 3.
11	"trigin3", "trigger_input3": Output is assigned to Trigger Input 4.
12	"trigin4", "trigger_input4": Output is assigned to Trigger Input 5.
13	"trigin5", "trigger_input5": Output is assigned to Trigger Input 6.
14	"trigin6", "trigger_input6": Output is assigned to Trigger Input 7.
15	"trigin7", "trigger_input7": Output is assigned to Trigger Input 8.
16	"low": Output is set to low.
17	"high": Output is set to high.

/dev..../sgchannels/n/oscs/n/freq

Properties: Read, Write, Setting
Type: Double
Unit: Hz

Frequency control for each oscillator.

/dev..../sgchannels/n/output/delay

Properties: Read, Write, Setting
Type: Double
Unit: s

This value adds a delay to both the signal and trigger/marker outputs.

/dev..../sgchannels/n/output/filter

Properties: Read
Type: Integer (enumerated)
Unit: None

Reads the selected analog filter before the Signal Output.

0	"lowpass_1500": Low-pass filter of 1.5 GHz.
1	"lowpass_3000": Low-pass filter of 3 GHz.
2	"bandpass_3000_6000": Band-pass filter between 3 GHz - 6 GHz
3	"bandpass_6000_10000": Band-pass filter between 6 GHz - 10 GHz

/dev..../sgchannels/n/output/muting/enable

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Enables the muting functionality on the signal output. If enabled (1), the output noise is suppressed when the marker output is high. The default value is disabled (0).

/dev..../sgchannels/n/output/on

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Enables the Signal Output.

/dev..../sgchannels/n/output/overrangecount

Properties: Read
Type: Integer (64 bit)
Unit: None

Indicates the number of times the Signal Output was in an overrange condition within the last 200 ms. It is checked for an overrange condition every 10 ms.

/dev..../sgchannels/n/output/range

Properties: Read, Write, Setting
Type: Double
Unit: dBm

Sets the maximal Range of the Signal Output power. The instrument selects the closest available Range with a resolution of 5 dBm.

/dev..../sgchannels/n/output/rflfpath

Properties: Read, Write, Setting
Type: Integer (enumerated)
Unit: None

Switch between RF and LF output path.

0	"lf": LF path is used.
1	"rf": RF path is used.

/dev..../sgchannels/n/outputrouter/enable

Properties: Read, Write, Setting
Type: Integer (enumerated)
Unit: None

Enable outputrouter module

0 "off": Output-router disabled
 1 "on": Output-router enabled

/dev..../sgchannels/n/outputrouter/overflowcount

Properties: Read
Type: Integer (64 bit)
Unit: None

Indicates the number of overflow events in the output-router of the corresponding channel within intervals of 200 ms. An overflow condition results in clipping of the output signal.

/dev..../sgchannels/n/outputrouter/routes/n/amplitude

Properties: Read, Write, Setting
Type: Double
Unit: None

Configure amplitude of route. Selected signal (source) is multiplied with amplitude and phase, and summed with other routes on SG-channel's output.

/dev..../sgchannels/n/outputrouter/routes/n/enable

Properties: Read, Write, Setting
Type: Integer (enumerated)
Unit: None

Enable/disable route.

0 "off": OFF: Route inactive
 1 "on": ON: Route active

/dev..../sgchannels/n/outputrouter/routes/n/phase

Properties: Read, Write, Setting
Type: Double
Unit: deg

Configure phase of route. Selected signal (source) is multiplied with amplitude and phase, and summed with other routes on SG-channel's output.

/dev..../sgchannels/n/outputrouter/routes/n/source

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Select AWG channel source as input to the outputrouter.

/dev..../sgchannels/n/pipeliner/availableslots

Properties: Read
Type: Integer (64 bit)
Unit: None

Number of free slots in the sequence pipeliner queue. Sequence upload is blocked if this node is 0.

/dev..../sgchannels/n/pipelinier/commit

Properties: Read, Write
Type: Integer (64 bit)
Unit: None

Commit node data in staging area to queue of sequence pipelinier.

/dev..../sgchannels/n/pipelinier/enable

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Enable execution of sequences in pipeline.

/dev..../sgchannels/n/pipelinier/idcurrent

Properties: Read
Type: Integer (64 bit)
Unit: None

ID of sequence in staging area.

/dev..../sgchannels/n/pipelinier/idrunning

Properties: Read
Type: Integer (64 bit)
Unit: None

ID of executed sequence.

/dev..../sgchannels/n/pipelinier/maxslots

Properties: Read
Type: Integer (64 bit)
Unit: None

Maximum number of available slots in the sequence pipelinier queue.

/dev..../sgchannels/n/pipelinier/mode

Properties: Read, Write, Setting
Type: Integer (enumerated)
Unit: None

Selects the sequence pipelinier mode: off (default), batch, or queue mode. Changing the mode will reset both the sequence pipelinier and the normal AWG.

- 0 "off": Off: The sequence pipelinier is turned off.
- 1 "batch": Batch: The sequence pipelinier operates in batch mode. All sequences must be committed before the pipelinier is enabled. A batch can be executed once or multiple times.
- 2 "queue": Queue: The sequence pipelinier operates in queue mode. Sequences can be committed while the pipelinier is enabled. Every sequence is executed only once and the slot in the queue is then available for a new sequence.

/dev..../sgchannels/n/pipelinier/ready

Properties: Read
Type: Integer (64 bit)
Unit: None

Indicates whether a sequence can be committed to the pipelinier.

/dev..../sgchannels/n/pipeliner/repetitions/remaining

Properties: Read
Type: Integer (64 bit)
Unit: None

Number of remaining batch repetitions. This node is fixed to 1 if the sequence pipeliner is not in batch mode.

/dev..../sgchannels/n/pipeliner/repetitions/value

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Number of batch repetitions (1 to 4e6). This node is fixed to 1 if the sequence pipeliner is not in batch mode.

/dev..../sgchannels/n/pipeliner/reset

Properties: Read, Write
Type: Integer (64 bit)
Unit: None

Clears all sequences previously added to the sequence pipeliner and disables the pipeliner if it has been running before.

/dev..../sgchannels/n/pipeliner/status

Properties: Read
Type: Integer (enumerated)
Unit: None

Status of the sequence pipeliner (0: idle, 1: executing sequence, 2: waiting for next sequence to be committed (queue mode only))

- 0 "idle": Idle: The sequence pipeliner is idle.
- 1 "exec": Executing sequence: The sequence pipeliner is executing a sequence.
- 2 "waiting": Waiting: The sequence pipeliner is waiting for the next sequence to be committed (queue mode only).
- 3 "done": Done: The sequence pipeliner is still enabled but all sequences have been executed (batch mode only).

/dev..../sgchannels/n/pipeliner/timeout

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Maximal execution time per sequence in milliseconds. The execution of a sequence is aborted if the maximal execution time is reached. A value of 0 means infinity.

/dev..../sgchannels/n/sines/n/freq

Properties: Read
Type: Double
Unit: Hz

Indicates the frequency of the sines generator.

/dev..../sgchannels/n/sines/n/harmonic

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Multiplies the sine signals's reference frequency with the integer factor defined by this field.

/dev..../sgchannels/n/sines/n/i/cos/amplitude

Properties: Read, Write, Setting
Type: Double
Unit: None

Sets the peak amplitude of the cosine component on the I signal path.

/dev..../sgchannels/n/sines/n/i/enable

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Enables the sine signal to the I signal path.

/dev..../sgchannels/n/sines/n/i/sin/amplitude

Properties: Read, Write, Setting
Type: Double
Unit: None

Sets the peak amplitude of the sine component on the I signal path.

/dev..../sgchannels/n/sines/n/oscselect

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Select oscillator for generation of this sine signal.

/dev..../sgchannels/n/sines/n/phaseshift

Properties: Read, Write, Setting
Type: Double
Unit: None

Phase shift applied to sine signal.

/dev..../sgchannels/n/sines/n/q/cos/amplitude

Properties: Read, Write, Setting
Type: Double
Unit: None

Sets the peak amplitude of the cosine component on the Q signal path.

/dev..../sgchannels/n/sines/n/q/enable

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Enables the sine signal to the Q signal path.

/dev..../sgchannels/n/sines/n/q/sin/amplitude

Properties: Read, Write, Setting
Type: Double
Unit: None

Sets the peak amplitude of the sine component on the Q signal path.

/dev..../sgchannels/n/synchronization/enable

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Enable multi-channel synchronization for this channel. The program will only execute once all channels with enabled synchronization are ready.

/dev..../sgchannels/n/synthesizer

Properties: Read
Type: Integer (64 bit)
Unit: None

Index of synthesizer mapped to this channel.

/dev..../sgchannels/n/trigger/delay

Properties: Read, Write, Setting
Type: Double
Unit: s

This delay adds an offset that acts only on the trigger/marker output. The total delay to the trigger/marker output is the sum of this value and the value of the output delay node.

/dev..../sgchannels/n/trigger/imp50

Properties: Read, Write, Setting
Type: Integer (enumerated)
Unit: None

Trigger Input impedance: When on, the Trigger Input impedance is 50 Ohm; when off, 1 kOhm.

0	"1_kOhm": OFF: 1 k Ohm
1	"50_Ohm": ON: 50 Ohm

/dev..../sgchannels/n/trigger/level

Properties: Read, Write, Setting
Type: Double
Unit: V

Defines the analog Trigger level.

/dev..../sgchannels/n/trigger/value

Properties: Read
Type: Integer (64 bit)
Unit: None

Shows the value of the digital Trigger Input. The value is integrated over a period of 100 ms. Values are: 1: low; 2: high; 3: was low and high in the period.

7.2.7. STATS

/dev..../stats/physical/currents/n

Properties: Read
Type: Double
Unit: mA

Provides internal current readings for monitoring.

/dev..../stats/physical/fanspeeds/n

Properties: Read
Type: Integer (64 bit)
Unit: RPM

Speed of the internal cooling fans for monitoring.

/dev..../stats/physical/fpga/aux

Properties: Read
Type: Double
Unit: V

Supply voltage of the FPGA.

/dev..../stats/physical/fpga/core

Properties: Read
Type: Double
Unit: V

Core voltage of the FPGA.

/dev..../stats/physical/fpga/pstemp

Properties: Read
Type: Double
Unit: °C

Internal temperature of the FPGA's processor system.

/dev..../stats/physical/fpga/temp

Properties: Read
Type: Double
Unit: °C

Internal temperature of the FPGA.

/dev..../stats/physical/overtemperature

Properties: Read
Type: Integer (64 bit)
Unit: None

This flag is set to a value greater than 0 when the internal temperatures are reaching critical limits.

/dev..../stats/physical/power/currents/n

Properties: Read
Type: Double
Unit: A

Currents of the main power supply.

/dev..../stats/physical/power/temperatures/n

Properties: Read
Type: Double
Unit: °C

Temperatures of the main power supply.

/dev..../stats/physical/power/voltages/n

Properties: Read
Type: Double
Unit: V

Voltages of the main power supply.

/dev..../stats/physical/sigins/n/currents/n

Properties: Read
Type: Double
Unit: A

Provides internal current readings on the Signal Input board for monitoring.

/dev..../stats/physical/sigins/n/temperatures/n

Properties: Read
Type: Double
Unit: °C

Provides internal temperature readings on the Signal Input board for monitoring.

/dev..../stats/physical/sigins/n/voltages/n

Properties: Read
Type: Double
Unit: V

Provides internal voltage measurement on the Signal Input board for monitoring.

/dev..../stats/physical/sigouts/n/currents/n

Properties: Read
Type: Double
Unit: A

Provides internal current readings on the Signal Output board for monitoring.

/dev..../stats/physical/sigouts/n/temperatures/n

Properties: Read
Type: Double
Unit: °C

Provides internal temperature readings on the Signal Output board for monitoring.

/dev..../stats/physical/sigouts/n/voltages/n

Properties: Read
Type: Double
Unit: V

Provides internal voltage readings on the Signal Output board for monitoring.

/dev..../stats/physical/synthesizer/currents/n

Properties: Read
Type: Double
Unit: A

Provides internal current readings on the Synthesizer board for monitoring.

/dev..../stats/physical/synthesizer/temperatures/n

Properties: Read
Type: Double
Unit: °C

Provides internal temperature readings on the Synthesizer board for monitoring.

/dev..../stats/physical/synthesizer/voltages/n

Properties: Read
Type: Double
Unit: V

Provides internal voltage readings on the Synthesizer board for monitoring.

/dev..../stats/physical/temperatures/n

Properties: Read
Type: Double
Unit: °C

Provides internal temperature readings for monitoring.

/dev..../stats/physical/voltages/n

Properties: Read
Type: Double
Unit: V

Provides internal voltage readings for monitoring.

7.2.8. STATUS

/dev..../status/flags/binary

Properties: Read
Type: Integer (64 bit)
Unit: None

A set of binary flags giving an indication of the state of various parts of the device. Reserved for future use.

/dev..../status/time

Properties:	Read
Type:	Integer (64 bit)
Unit:	None

The current timestamp.

7.2.9. SYNTHESIZERS

/dev..../synthesizers/n/centerfreq

Properties:	Read, Write, Setting
Type:	Double
Unit:	Hz

The Center Frequency of the synthesizer.

7.2.10. SYSTEM

/dev..../system/activeinterface

Properties:	Read
Type:	String
Unit:	None

Currently active interface of the device.

/dev..../system/boardrevisions/n

Properties:	Read
Type:	String
Unit:	None

Hardware revision of the motherboard containing the FPGA.

/dev..../system/clocks/referenceclock/in/freq

Properties:	Read
Type:	Double
Unit:	Hz

Indicates the frequency of the reference clock.

/dev..../system/clocks/referenceclock/in/source

Properties:	Read, Write, Setting
Type:	Integer (enumerated)
Unit:	None

The intended reference clock source. When the source is changed, all the instruments connected with ZSync links will be disconnected. The connection should be re-established manually.

- | | |
|---|--|
| 0 | "internal": The internal clock is intended to be used as the frequency and time base reference. |
| 1 | "external": An external clock is intended to be used as the frequency and time base reference. Provide a clean and stable 10 MHz or 100 MHz reference to the appropriate back panel connector. |
| 2 | "zsync": The ZSync clock is intended to be used as the frequency and time base reference. |

/dev..../system/clocks/referenceclock/in/sourceactual

Properties: Read
Type: Integer (enumerated)
Unit: None

The actual reference clock source.

0 "internal": The internal clock is used as the frequency and time base reference.
 1 "external": An external clock is used as the frequency and time base reference.
 2 "zsync": The ZSync clock is used as the frequency and time base reference.

/dev..../system/clocks/referenceclock/in/status

Properties: Read
Type: Integer (enumerated)
Unit: None

Status of the reference clock.

0 "locked": Reference clock has been locked on.
 1 "error": There was an error locking onto the reference clock signal.
 2 "busy": The device is busy trying to lock onto the reference clock signal.

/dev..../system/clocks/referenceclock/out/enable

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Enable clock signal on the reference clock output. When the clock output is turned on or off, all the instruments connected with ZSync links will be disconnected. The connection should be re-established manually.

/dev..../system/clocks/referenceclock/out/freq

Properties: Read, Write, Setting
Type: Double
Unit: Hz

Select the frequency of the output reference clock. Only 10 MHz and 100 MHz are allowed.

/dev..../system/digitalmixer/reset/all

Properties: Read, Write
Type: Integer (64 bit)
Unit: None

Writing to this node clears all digital mixer NCOs of the instrument.

/dev..../system/digitalmixer/reset/mode

Properties: Read, Write, Setting
Type: Integer (enumerated)
Unit: None

Configure the NCO reset mode.

0 "manual": In manual mode the instrument does not automatically reset NCOs when switching a channel from LF to RF mode.
 1 "auto": In automatic mode the instrument automatically resets the NCOs of all channels whenever a channel is switched from LF to RF, in order to restore alignment.

/dev..../system/digitalmixer/reset/select

Properties: Read, Write
Type: Integer (64 bit)
Unit: None

Writing a bit mask to this node triggers a digital mixer NCO reset of the selected (bit value: 1) channels.

/dev..../system/fpgarevision

Properties: Read
Type: Integer (64 bit)
Unit: None

HDL firmware revision.

/dev..../system/fwlog

Properties: Read
Type: String
Unit: None

Returns log output of the firmware.

/dev..../system/fwlogenable

Properties: Read, Write
Type: Integer (64 bit)
Unit: None

Enables logging to the fwlog node.

/dev..../system/fwrevision

Properties: Read
Type: Integer (64 bit)
Unit: None

Revision of the device-internal controller software.

/dev..../system/fx3revision

Properties: Read
Type: String
Unit: None

USB firmware revision.

/dev..../system/identify

Properties: Read, Write
Type: Integer (64 bit)
Unit: None

Setting this node to 1 will cause all frontpanel LEDs to blink for 5 seconds, then return to their previous state.

/dev..../system/internaltrigger/enable

Properties: Read, Write
Type: Integer (enumerated)
Unit: None

Enable internal trigger generator.

0 "off": Generator off
 1 "on": Generator on

/dev..../system/internaltrigger/holdoff

Properties: Read, Write, Setting
Type: Double
Unit: s

Hold-off time between generated triggers.

/dev..../system/internaltrigger/progress

Properties: Read
Type: Double
Unit: None

The fraction of the triggers generated so far.

/dev..../system/internaltrigger/repetitions

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Number of triggers to be generated.

/dev..../system/internaltrigger/synchronization/enable

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Enable synchronization. Trigger generation will only start once all synchronization participants have reported a ready status. Synchronization checks will be repeated with the same trigger generation settings (holdoff and repetitions) until synchronization is disabled.

/dev..../system/kerneltype

Properties: Read
Type: String
Unit: None

Returns the type of the data server kernel (mdk or hpk).

/dev..../system/nics/n/defaultgateway

Properties: Read, Write
Type: String
Unit: None

Default gateway configuration for the network connection.

/dev..../system/nics/n/defaultip4

Properties: Read, Write
Type: String
Unit: None

IPv4 address of the device to use if static IP is enabled.

/dev..../system/nics/n/defaultmask

Properties: Read, Write
Type: String
Unit: None

IPv4 mask in case of static IP.

/dev..../system/nics/n/gateway

Properties: Read
Type: String
Unit: None

Current network gateway.

/dev..../system/nics/n/ip4

Properties: Read
Type: String
Unit: None

Current IPv4 of the device.

/dev..../system/nics/n/mac

Properties: Read
Type: String
Unit: None

Current MAC address of the device network interface.

/dev..../system/nics/n/mask

Properties: Read
Type: String
Unit: None

Current network mask.

/dev..../system/nics/n/saveip

Properties: Read, Write
Type: Integer (64 bit)
Unit: None

If written, this action will program the defined static IP address to the device.

/dev..../system/nics/n/static

Properties: Read, Write
Type: Integer (64 bit)
Unit: None

Enable this flag if the device is used in a network with fixed IP assignment without a DHCP server.

/dev..../system/powerconfigdate

Properties: Read
Type: Integer (64 bit)
Unit: None

Contains the date of power configuration (format is: (year << 16) | (month << 8) | day)

/dev..../system/preset/busy

Properties: Read
Type: Integer (64 bit)
Unit: None

Indicates if presets are currently loaded.

/dev..../system/preset/error

Properties: Read
Type: Integer (64 bit)
Unit: None

Indicates if the last operation was illegal. Successful: 0, Error: 1.

/dev..../system/preset/load

Properties: Read, Write
Type: Integer (64 bit)
Unit: None

Load the selected preset.

/dev..../system/properties/freqresolution

Properties: Read
Type: Integer (64 bit)
Unit: None

The number of bits used to represent a frequency.

/dev..../system/properties/freqscaling

Properties: Read
Type: Double
Unit: None

The scale factor to use to convert a frequency represented as a freqresolution-bit integer to a floating point value.

/dev..../system/properties/maxfreq

Properties: Read
Type: Double
Unit: None

The maximum oscillator frequency that can be set.

/dev..../system/properties/maxtimeconstant

Properties: Read
Type: Double
Unit: s

The maximum demodulator time constant that can be set. Only relevant for lock-in amplifiers.

/dev..../system/properties/minfreq

Properties: Read
Type: Double
Unit: None

The minimum oscillator frequency that can be set.

/dev..../system/properties/mintimeconstant

Properties: Read
Type: Double
Unit: s

The minimum demodulator time constant that can be set. Only relevant for lock-in amplifiers.

/dev..../system/properties/negativefreq

Properties: Read
Type: Integer (64 bit)
Unit: None

Indicates whether negative frequencies are supported.

/dev..../system/properties/timebase

Properties: Read
Type: Double
Unit: s

Minimal time difference between two timestamps. The value is equal to 1/(maximum sampling rate).

/dev..../system/shutdown

Properties: Read, Write
Type: Integer (64 bit)
Unit: None

Sending a '1' to this node initiates a shutdown of the operating system on the device. It is recommended to trigger this shutdown before switching the device off with the hardware switch at the back side of the device.

/dev..../system/stall

Properties: Read, Write
Type: Integer (64 bit)
Unit: None

Indicates if the network connection is stalled.

/dev..../system/swtriggers/n/single

Properties: Read, Write
Type: Integer (64 bit)
Unit: None

Issues a single software trigger event.

/dev..../system/synchronization/source

Properties: Read, Write, Setting
Type: Integer (enumerated)
Unit: None

Selects the source for synchronization of channels: internal (default) or external

- | | |
|---|--|
| 0 | "internal": Internal: Synchronization of all channels of a device that have the corresponding synchronization setting enabled. |
| 1 | "external": External: Same as internal plus synchronization to other devices via ZSync. |

/dev..../system/triggerdelays/automatic

Properties: Read, Write, Setting
Type: Integer (enumerated)
Unit: None

Enables the instrument to automatically adjust trigger delays to maintain output alignment

- | | |
|---|---|
| 0 | "off": No trigger delays are tuned automatically. The user has to manually align the channel outputs. |
| 1 | "on": The instrument will set the required trigger delay based on configuration |

/dev..../system/update

Properties: Read, Write
Type: Integer (64 bit)
Unit: None

Requests update of the device firmware and bitstream from the dataserver.