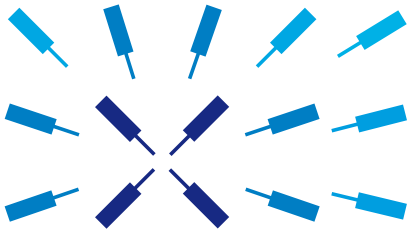
SHFSG User Manual



Zurich
Instruments

SHFSG User Manual

Zurich Instruments AG

Revision 22.08

Table of Contents

What's New in the SHFSG User Manual

Release 22.08

Release date: 31-Aug-2022

- AWG: Added support for 16 sample waveforms with the command table.
- AWG: Added a new sequencer command `playHold` to allow the AWG to hold waveform and marker data for a specified number of samples.
- AWG: Added ability of executeTableEntry to use variable arguments corresponding to qubit data received over ZSync.
- AWG: Fixed a bug in which the command table always required a waveform to make parameter changes.
- AWG: Improved the speed with which the AWGs can be enabled.
- LabOne: Added waveform and marker delay settings in the Digital Modulation and DIO Tabs, respectively.
- ZSync/DIO `/DEV*/SGCHANNELS/*/AWG/ELF/DATA` node accepts raw data as 8-, 16-, and 64-bit integer vectors in addition to 32-bit words.
- ZSync/DIO `/DEV*/DIOS/0/MODE` node changed keyword arguments to enable control of DIO values by the sequencer from `chanNseq` or `channelN_sequencer` to `sgchanNseq` or `sgchannelN_sequencer`.

Release 22.02

Release date: 28-Feb-2022

Highlights:

- Manual: Added new tutorials to the User Manual: Basic Sine Generation, Basic Waveform Playback, Triggering and Synchronization, Digital Modulation, Using the Python API, and Pulse-level Sequencing with the Command Table.
- Manual: Added LabOne UI descriptions for the Digital Modulation and DIO Tabs to the User Manual.
- AWG: Added support for fast sequencer-based IF sweeps via the SeqC commands configFreqSweep(), setSweepStep(), and setOscFreq().
- AWG: Added support for conditional triggering over DIO via SeqC commands: setDIO(), getDIO(), waitDIOTrigger(), getDIOTriggered(), playWaveDIO().
- AWG: Improved playWaveZSync() and getZSyncData() SeqC commands for fast feedback with PQSC.
- AWG: Added ability for each sequencer to read data from either ZSync or DIO.
- AWG: Added a built-in waiting time of wait(3) to the resetOscPhase() command, to prevent the oscillator from resetting during waveform playback.
- AWG: Added support for waitSineOscPhase() SeqC command.
- AWG: Fixed bug in which using too many triggers back to back would cause unpredictable waveform playback.
- LabOne: Added support for the SHFSG in zhinst-deviceutils in the Python API, to improve user programming experience.
- LabOne: Improved QCoDeS and Labber drivers and zhinst-toolkit to offer support of the SHFSG.
- LabOne: Added low-frequency path control to the Output Tab in the LabOne UI. Updated documentation in the User Manual accordingly.
- Outputs: Added ability to set the center frequency when using the low-frequency path. Accepted values: 0 - 2 GHz.
- Outputs: Marker and trigger outputs adjusted to be aligned with the RF output by default.
- Outputs: Default values of sine generator, AWG gain settings, and command table amplitude settings updated to automatically generate the upper sideband.
- Outputs: Replaced the two AWG output amplitude settings with a single global amplitude setting. Default value is 0.5 to prevent saturating the DAC.
- ZSync/DIO: Improved ZSync link stability.
- ZSync/DIO: Fixed bug in which DIO outputs could not be set properly.
- ZSync/DIO: Added the Real-Time Logger that logs history of incoming ZSync and DIO messages at /DEV*/SGCHANNELS/*/AWG/RTLOGGER/

Release 21.08

Release date: 31-Aug-2021

Highlights:

- Initial release of SHFSG user manual.

Declaration of Conformity

The manufacturer

Zurich Instruments
Technoparkstrasse 1
8005 Zurich Switzerland

declares that the product

SHFSG, Super High Frequency Signal Generator

fulfills the requirements of the European guidelines

- 2014/30/EU Electromagnetic Compatibility
- 2014/35/EU Low Voltage
- 2011/65/EU, 2015/863/EU, 2017/2102/EU Restriction of Hazardous Substances (RoHS)
- 1907/2006/EC Registration, Evaluation, Authorization, and Restriction of Chemicals (REACH)

The assessment was performed using the directives according to Table 1.

Table 1. Conformity table

EN 61326-1:2012 (ed.2.0)	Electrical equipment for measurement, control and laboratory use - EMC requirements - Part 1: General requirements
CISPR 11:2016-06 (EN 55011)	Industrial, scientific and medical (ISM) radio-frequency equipment – Electromagnetic disturbance characteristics – Limits and methods of measurement
EN 61010-1:2010 IEC61010-1:2010 + AMD1:2016 IEC61010-1 National Deviations for EU, US, CA	Safety requirements for electrical equipment for measurement, control and laboratory use

For the assessment of electromagnetic compatibility, the limits of radio interference for Class B equipment as well as the immunity to interference for operation in industry have been used as a basis.

CE

Chapter 1. Getting Started

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

Please refer to:

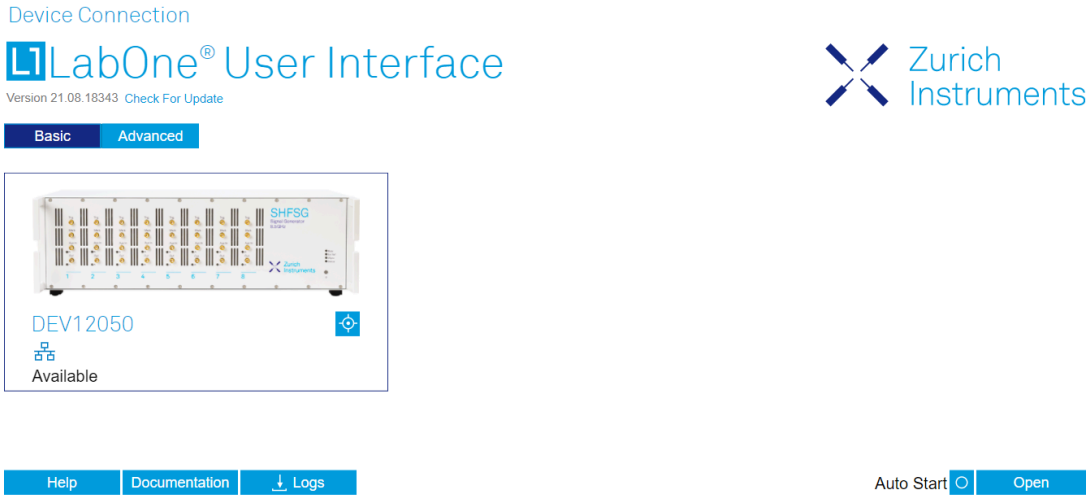
- Section 1.1 for a Quick Start Guide for the impatient.
- Section 1.2 for inspecting the package content and accessories.
- Section 1.3 for a list of essential handling and safety instructions.
- Section 1.4 - Section 1.6 for help connecting to the SHFSG Instrument with the LabOne software.
- Section 1.7 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 SHFSG User Manual.

1.1. 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 Section 1.3 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 Section 1.4.
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 Section 1.5.

If any problems occur while setting up the instrument and software, please see Section 1.7 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 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 Chapter 3. The functional description of the SHFSG can be found in Chapter 4 and provides a general introduction to the various tools and tables in each section describing every setting. In

the same section, Chapter 4 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.

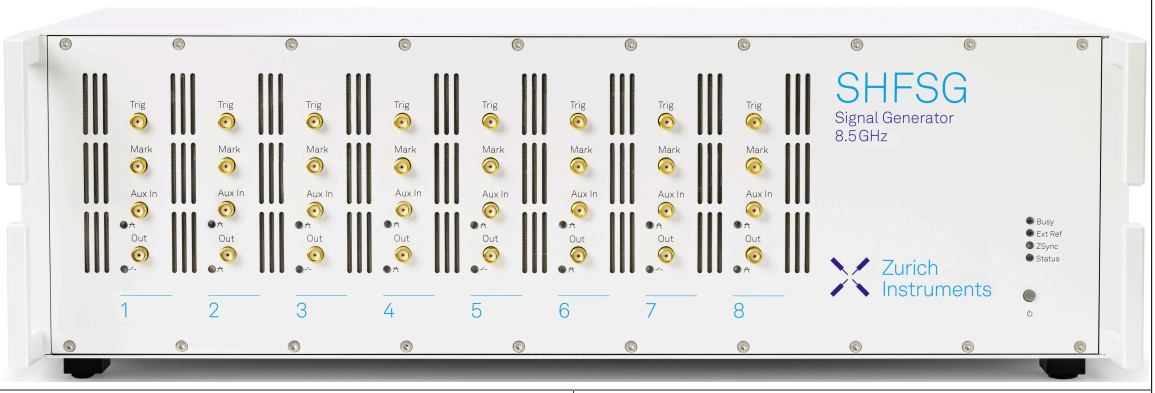
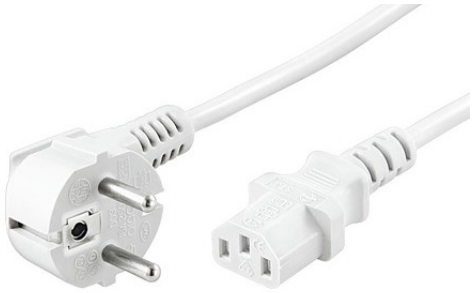
1.2. Inspect the Package Contents

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 SHFSG 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 1.1. Package contents

4 or 8-channel SHFSG instrument 	
the power cord (e.g. ETU norm) 	the power inlet, with power switch and fuse holder 

the USB 3.0 cable 	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 SHFSG 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.

1.3. Handling and Safety Instructions

The SHFSG 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 1.2. 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 groundloops, consider adding dc-blocks at the Inputs of the device.
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 Chapter 5 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.

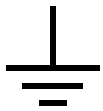
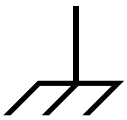
Notes	Instructions contained in the notes of this user manual are of essential importance for correctly interpreting the acquired measurement data.
Location and ventilation	This instrument or system is intended for indoor use in an installation category II and pollution degree 2 environment as per IEC 61010-1. Do not operate or store the instrument outside the ambient conditions specified in the Chapter 5 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 Chapter 5 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.

When 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 1.3. 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 1.4. Symbols

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

1.4. Software Installation

The SHFSG Instrument is operated from a host computer with the LabOne software. To install the LabOne software on a PC, administrator rights are 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 Section 1.6.

1.4.1. 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.

Windows LabOne Installation

1. The SHFSG 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.

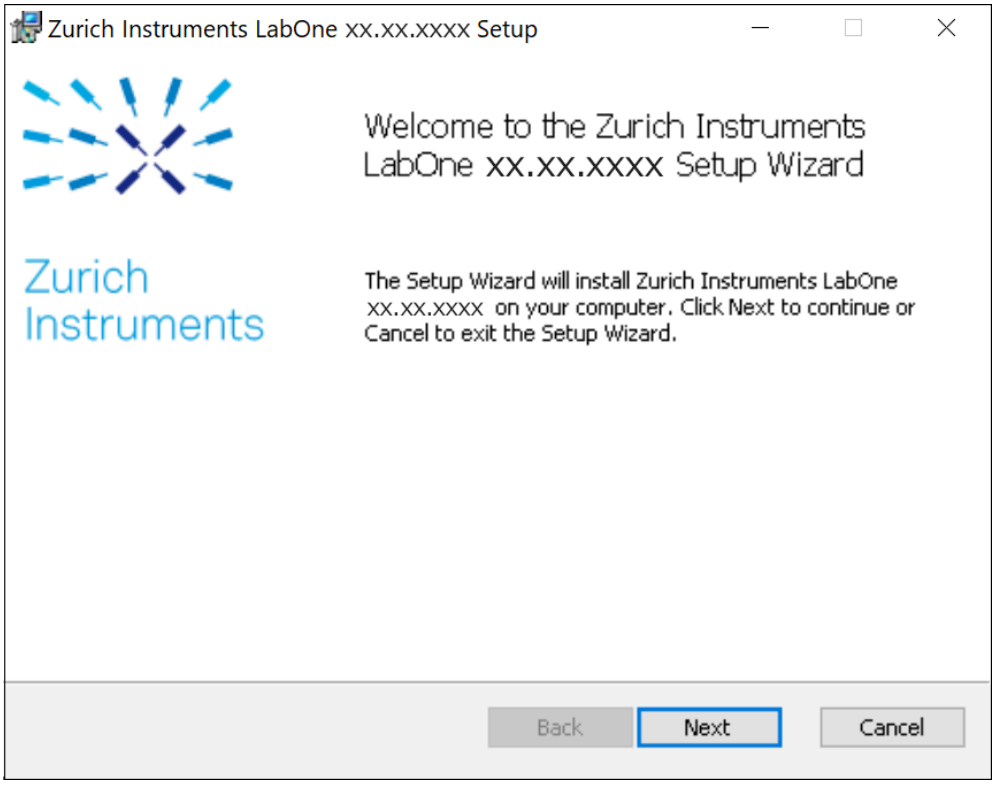


Figure 1.1. Installation welcome screen

   - 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 SHFSG Instrument the "SHF 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.

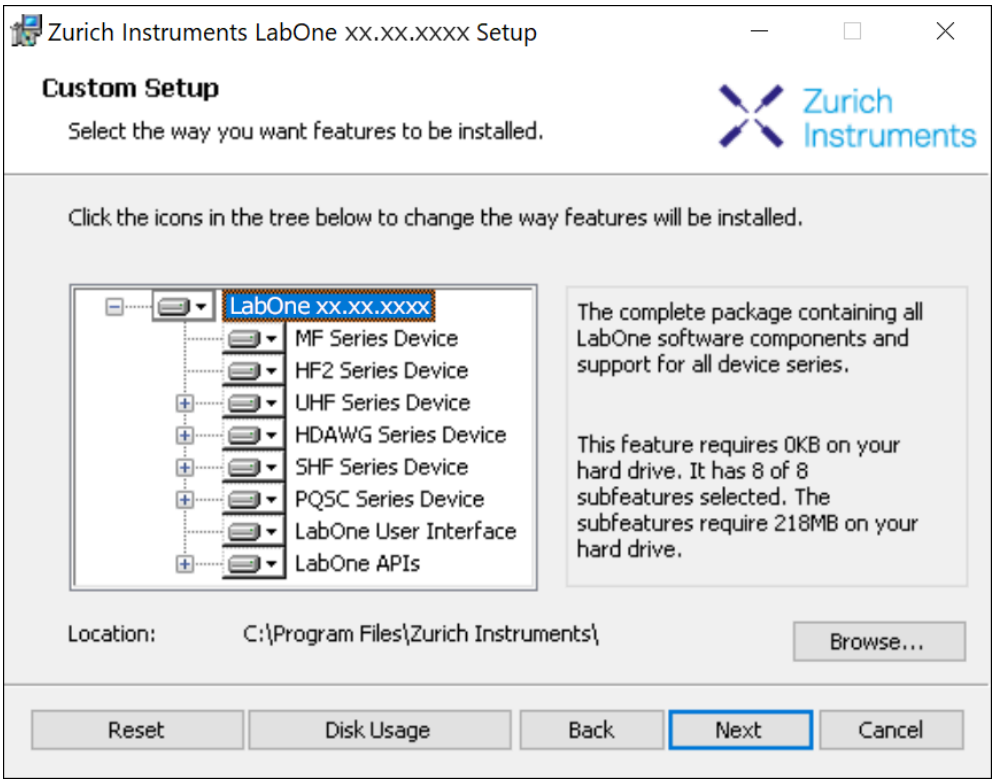


Figure 1.2. Custom setup screen

- 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.

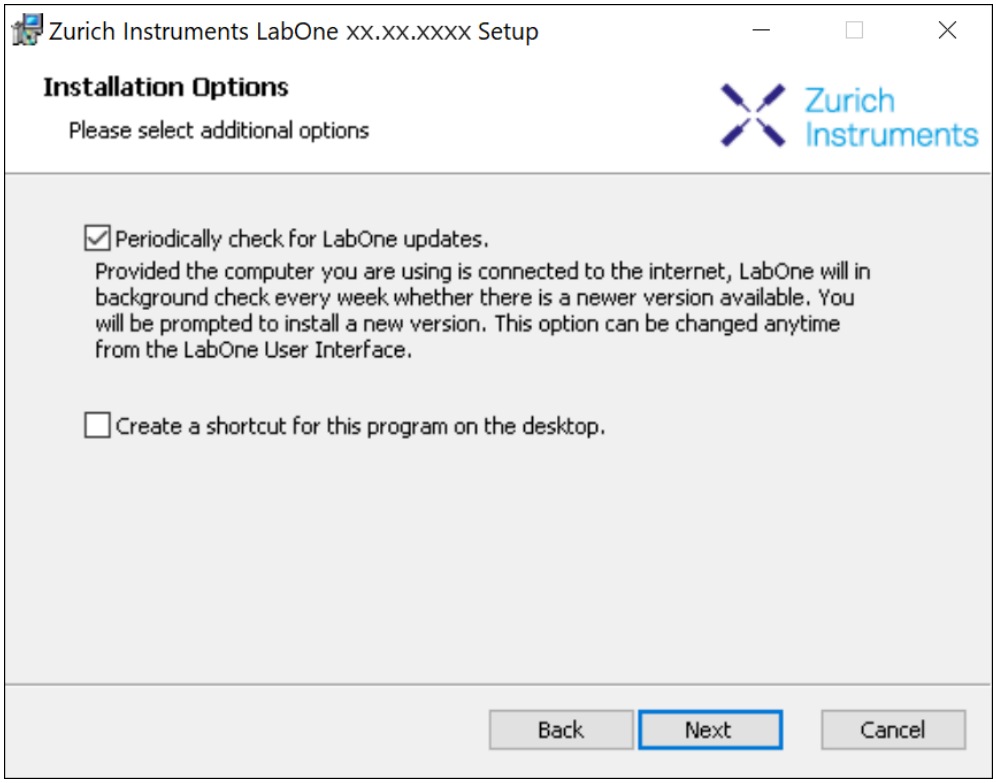


Figure 1.3. Automatic update check

- 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.

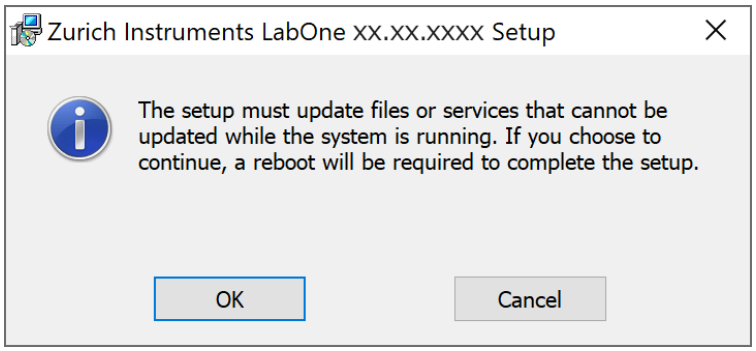


Figure 1.4. Installation reboot request

- 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**.

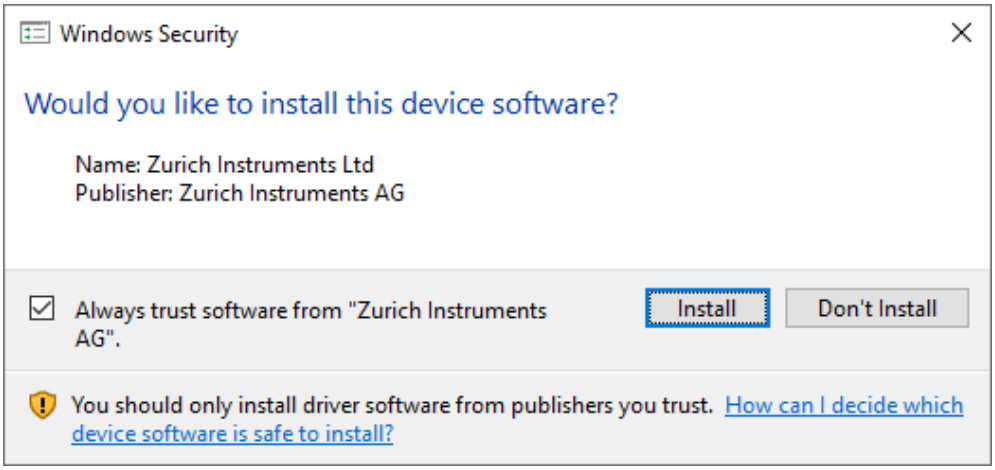


Figure 1.5. Installation driver acceptance

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

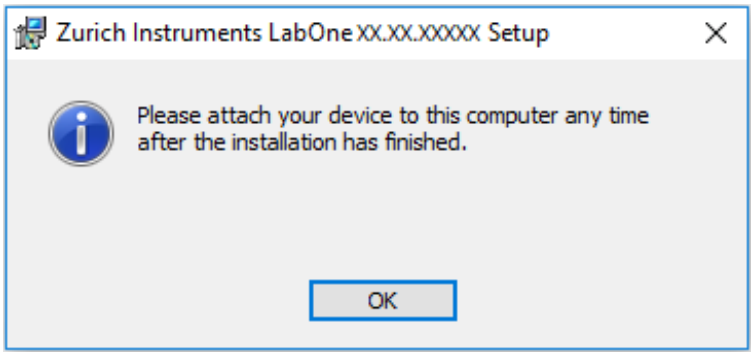


Figure 1.6. Installation completion screen

3. Click **Finish** to close the Zurich Instruments LabOne installer.
4. 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.

1.4.2. Installing LabOne on Linux

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.

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.

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 LabOne Data Server program at a command prompt:

   ```
   $ ziDataServer
   ```

   You should be able to access your instrument. In case of problems please consult the Section 1.7 at the end of this chapter.

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

   ```
   $ ziWebServer
   ```

3. 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.

4. 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.

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
   ```

1.4.3. Start LabOne Manually on 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. To start the Servers manually, open a command-line terminal (Command Prompt, PowerShell (Windows) or Bash (Linux)). For Windows, the current working directory needs to be the installation directory of the Web Server and Data Server. They are installed in the Program Files folder (usually: C:\Program Files) under \Zurich Instruments\LabOne in the WebServer and DataServer folders, respectively. The Web Server and Data Server (ziDataServer) are started by running the respective executable in each folder. Please be aware that only one instance of the Web Server can run at a time per computer. The behavior of the Servers can be changed by providing command line arguments. For a detailed list of all arguments see the command line help text:

```
$ ziWebServer --help
```

For the Data Server:

```
$ ziDataServer --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. Before running the Webserver from the terminal, the user needs to ensure there is no other instance of Webserver running in the background. This can be checked using the Tray Icon as shown below.

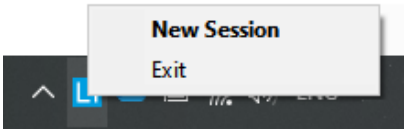


Figure 1.7. LabOne Tray Icon in Windows 10

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"
```

1.5. Connecting to the Instrument

The Zurich Instruments SHFSG is operated using the LabOne software. After installation of LabOne, the instrument can be connected to a PC by using either the Universal Serial Bus (USB) cable or the 1 Gbit/s Ethernet (1GbE) LAN cable supplied with the instrument. The LabOne software is controlled via a web browser after suitable physical and logical connections to the instrument have been made.

Note

The following web browsers are supported (latest versions).

- When using 1GbE, integrate the instrument physically into an existing local area network (LAN) by connecting the instrument to a switch in the LAN using an Ethernet cable. The instrument can then be accessed from a web browser running on any device in the same LAN. The Ethernet connection can also be point-to-point. This requires some adjustment of the network card settings of the host computer. Depending on the network configuration and the installed network card, one or the other connection scheme is better suited.
- Using the USB connection to physically connect to the instrument requires the installation of an RNDIS driver on the host computer. For PC users, this driver is included in the LabOne software installer. For Mac users, the driver is available online.

1.5.1. 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 1.8.

- 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.

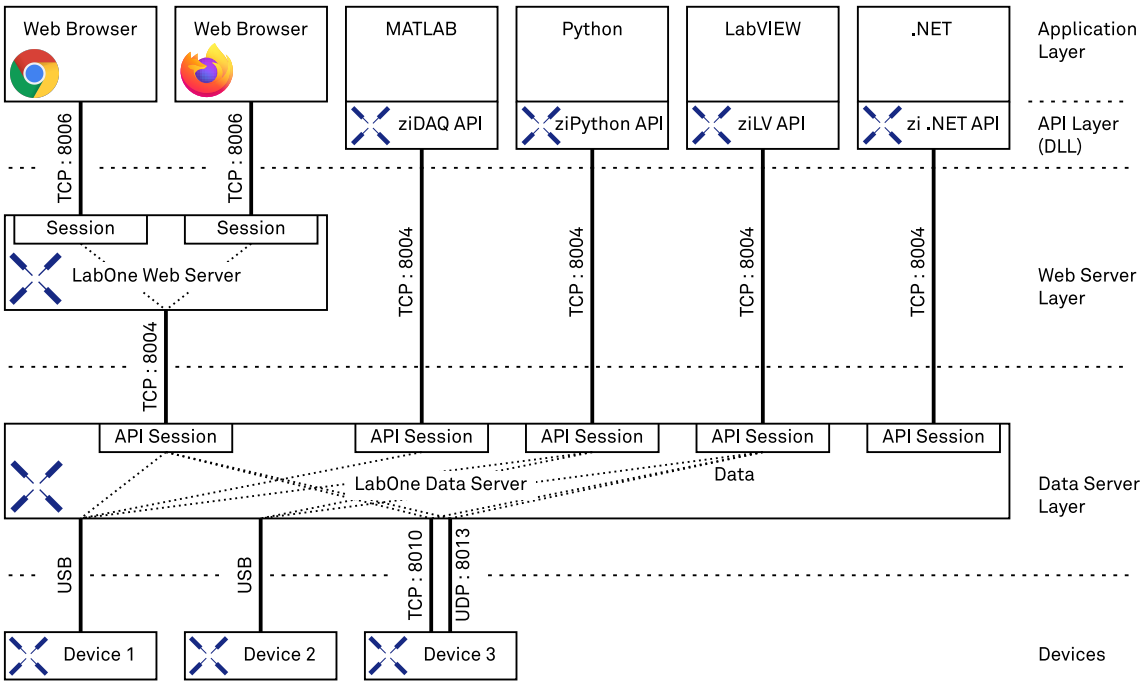


Figure 1.8. LabOne Software architecture

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. The Data Server is started automatically by a service when the PC is started. This service can be disabled if necessary, though the Data Server consumes only little resources when there is no active session. On a PC, only a single instance of a LabOne Data Server should be running.

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.

LabOne API Layer

The instrument can also be controlled via the application program interfaces (APIs) provided by Zurich Instruments.

The SHF-series comes with an extensive Python API and python-based drivers for the following frameworks:

- Zurich Instruments Toolkit
- QCoDeS
- 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.

1.5.2. LabOne Software Start-up

This section describes the start-up of the LabOne User Interface which is used to control the SHFSG Instrument. If the LabOne software is not yet installed on the PC please follow the instructions in Section 1.4. If the device is not yet connected please find more information in Section 1.5.3.

The LabOne User Interface start-up link can be found under the Windows 10 Start Menu (Under Windows 7 and 8, the LabOne User Interface start-up link can be found in `Start Menu → all programs / all apps → Zurich Instruments LabOne`). As shown in Figure 1.9, 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 Section 1.5.1.

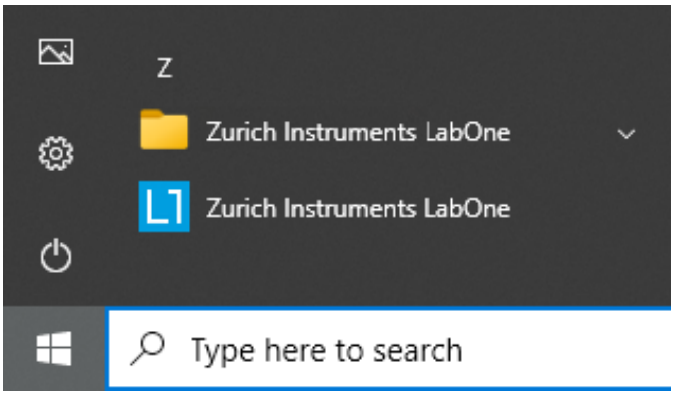


Figure 1.9. Link to the LabOne User Interface in the Windows 10 Start Menu

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 1.10 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 Section 1.5.1.

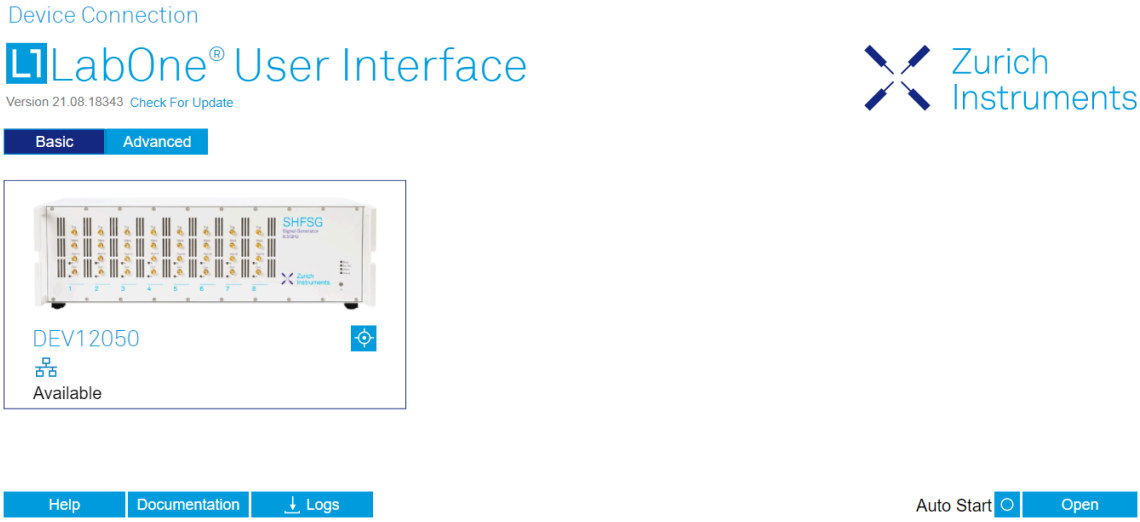


Figure 1.10. Device Connection dialog

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.

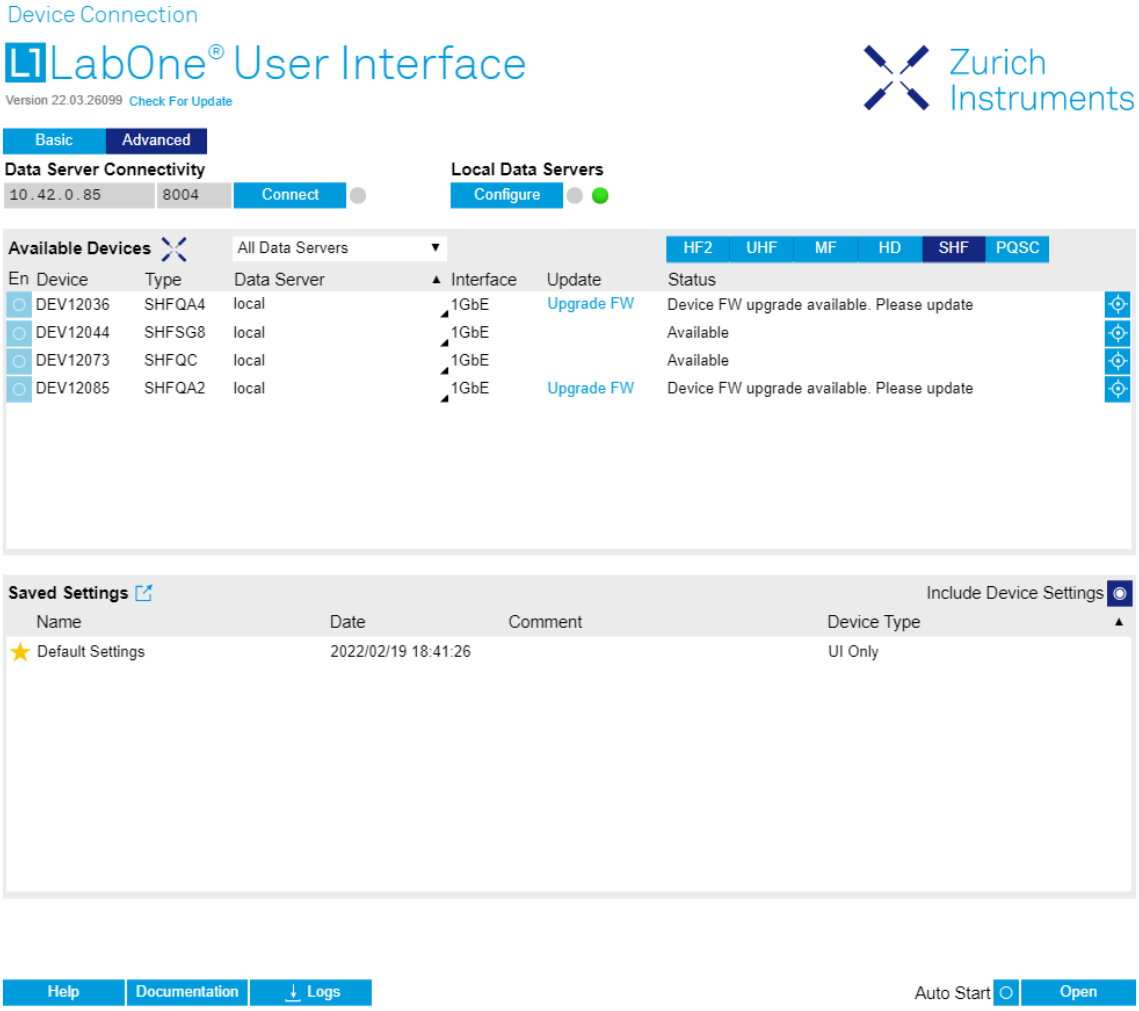


Figure 1.11. Device Connection dialog (Advanced view)

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, 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.

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 SHFSG 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, 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.

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 SHF 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. If an interface is physically connected but not visible please read Section 1.5.3.

If a firmware update of the instrument is available, the second to last column will show a button, and a simple click on it will update the instrument. The last column indicates the status of the device. Table 1.5 explains the meaning of some of the possible device statuses.

Table 1.5. Device Status Information

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.
Free	The device is not in use by any LabOne Data Server and can be connected by clicking the **Open** button.
In Use	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.
Device FW upgrade required/available	The firmware of the device is out of date. Please first upgrade the firmware as described in Section 1.6.
Device not yet ready	The device is visible and starting up.

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 1.6. 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 1.6. 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 .xml.
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 1.7.

Table 1.7. 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.

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.

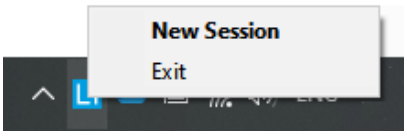


Figure 1.12. LabOne Tray Icon in Windows 10

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 1.10. The figure below shows an example of the Connection Lost dialog.

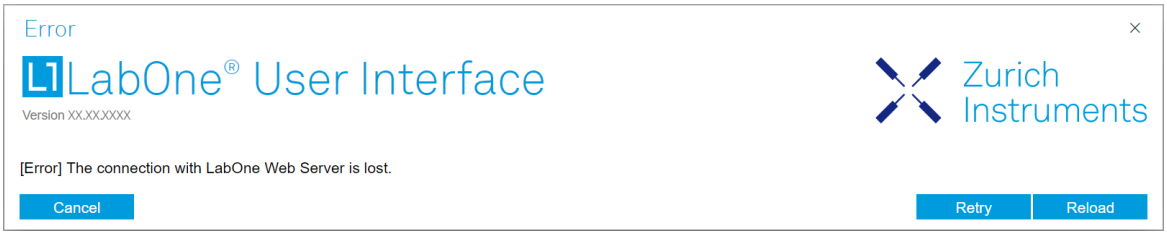


Figure 1.13. Dialog: Connection Lost

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 1.10. 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.

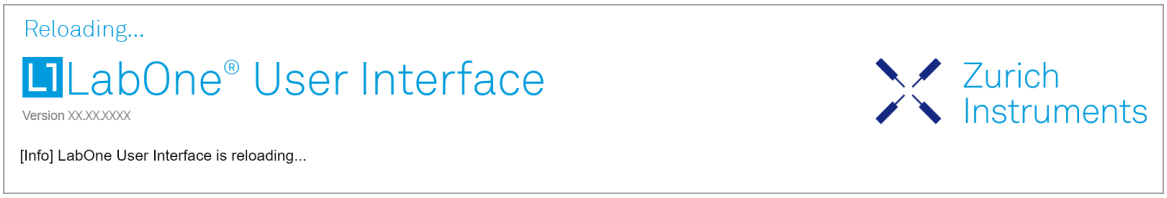


Figure 1.14. Dialog: Reloading

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 Section 1.5.3. The figure below shows an example of this dialog.

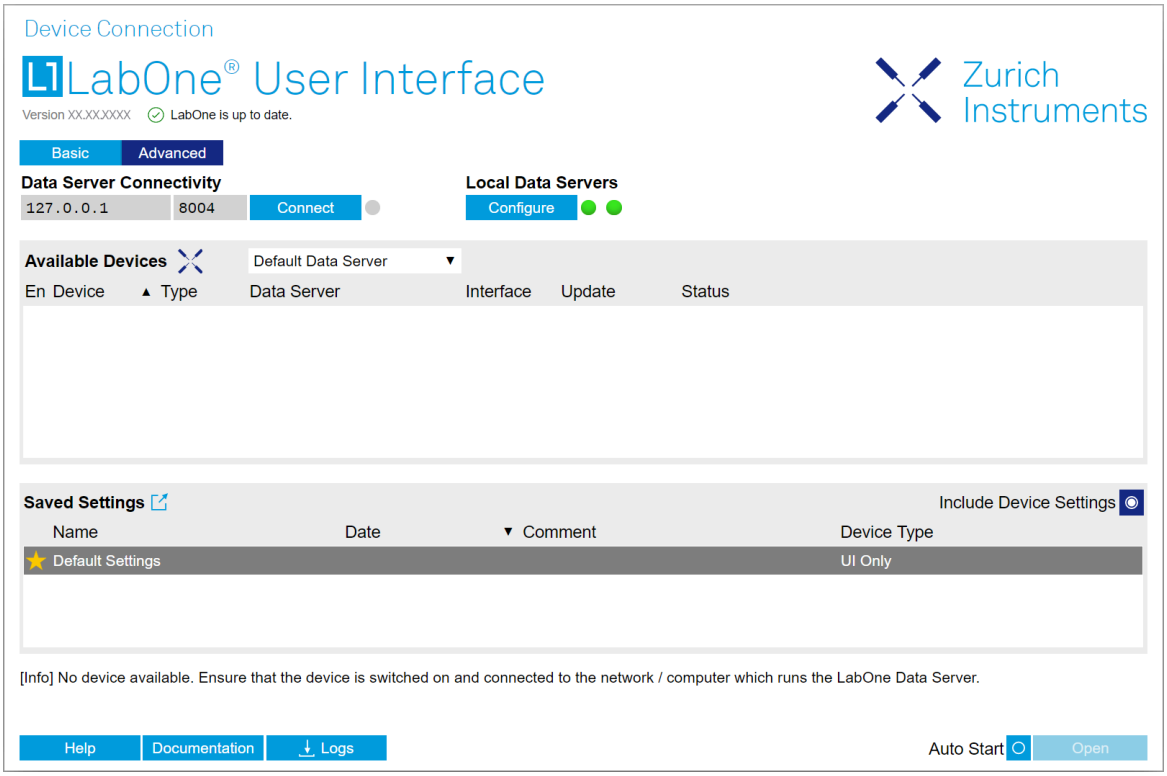


Figure 1.15. No Device Discovered

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 Section 1.6. If all the devices are in use, access is not possible until a connection is relinquished by another Data Server.

1.5.3. 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 connection 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.

Figure 1.16 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.

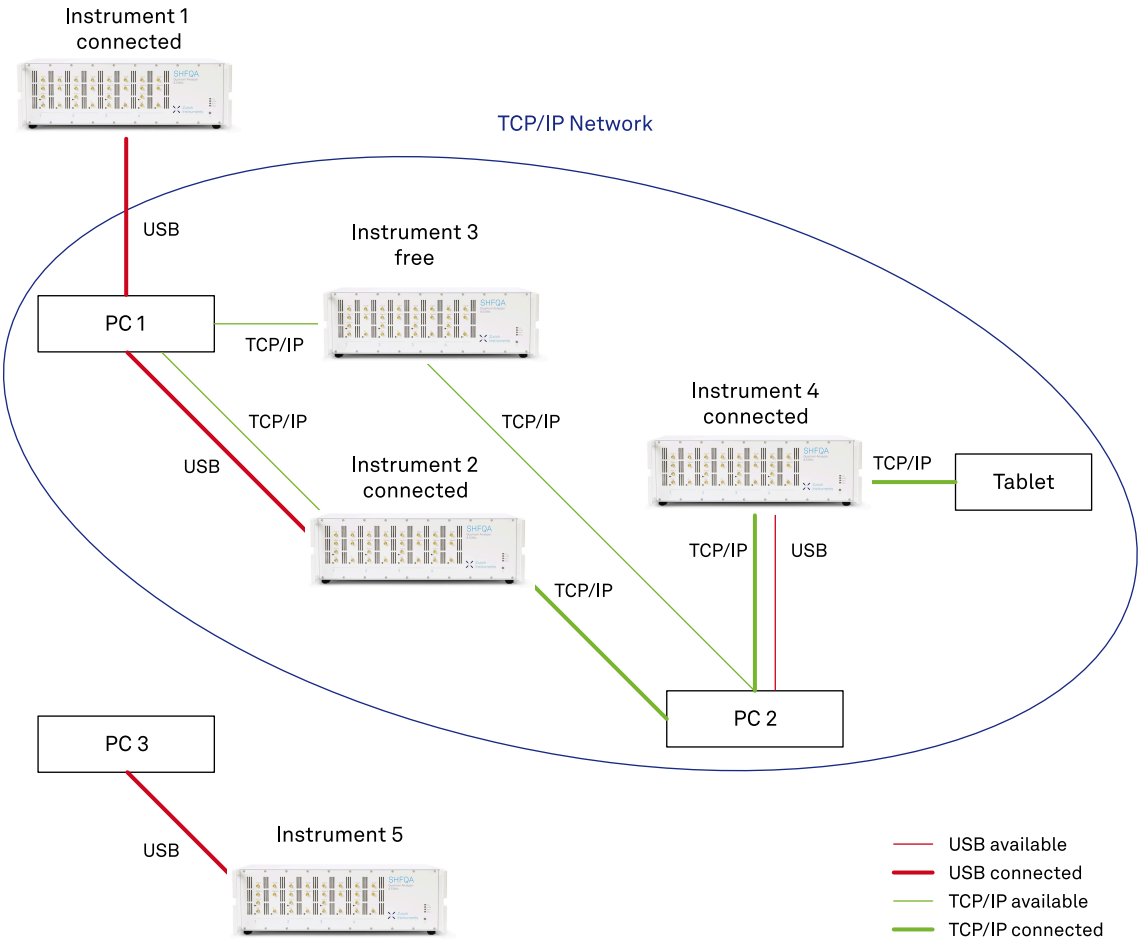


Figure 1.16. Connectivity Example

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.

1.5.4. 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 (LabOne 18.12 and later). 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 1.8 provides an overview of the two settings.

Table 1.8. 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:`--auto-connect=off`.

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 1.17.

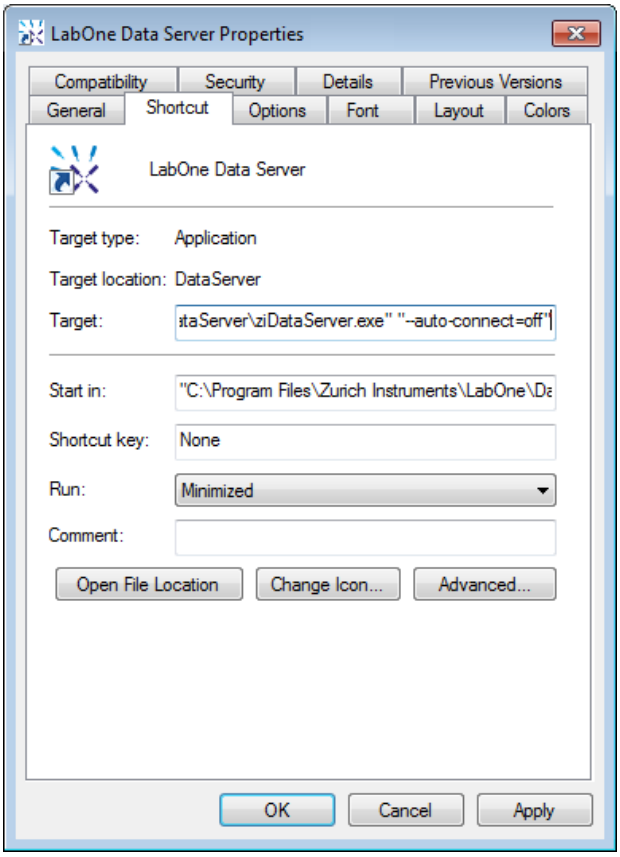


Figure 1.17. Setting auto-connect in Windows

1.5.5. 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`).

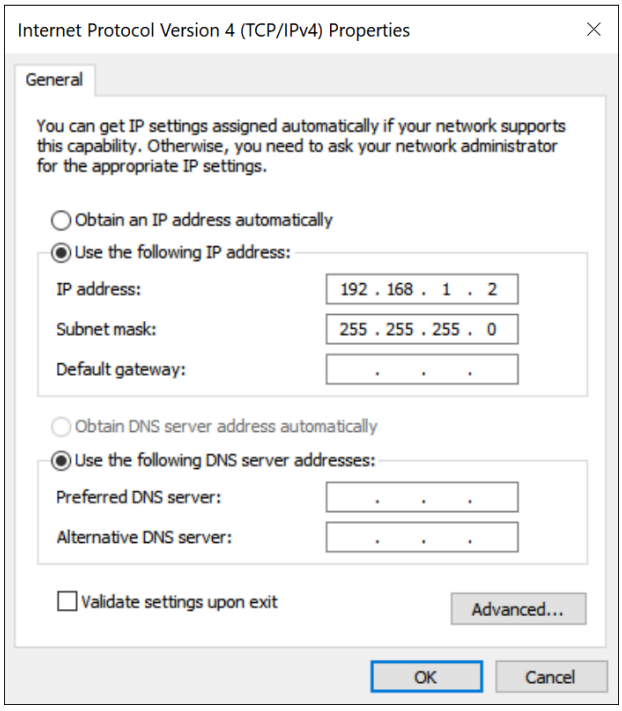


Figure 1.18. Static IP configuration for the host computer

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 the section called “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.

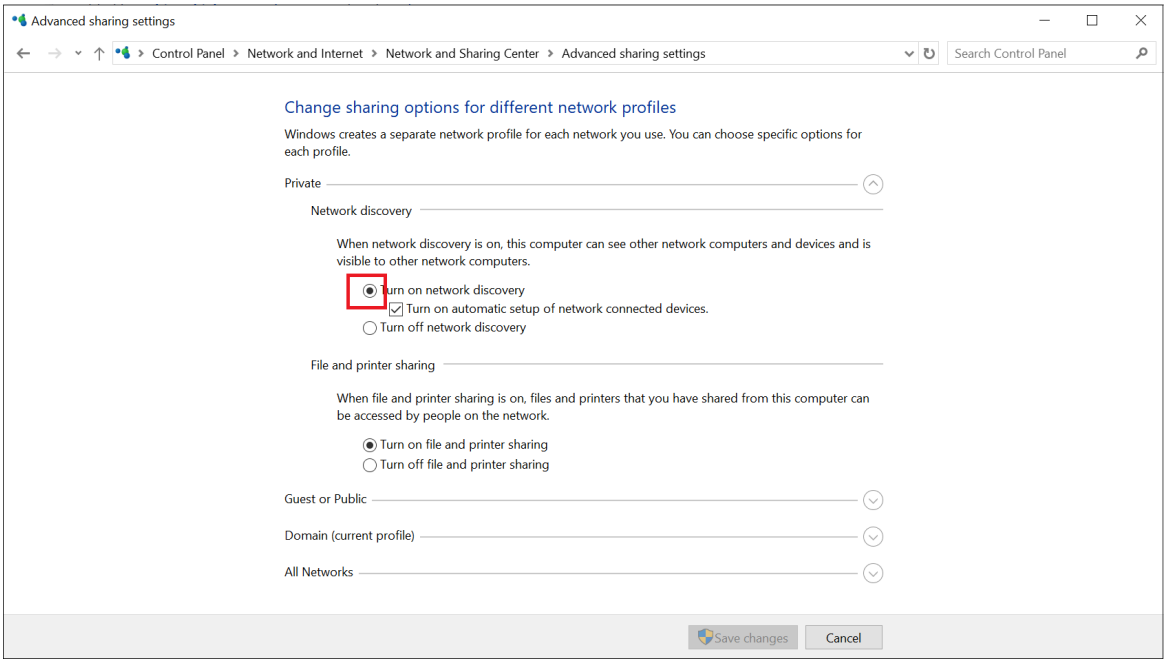


Figure 1.19. Turn on network discovery for Private P2P connection

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.

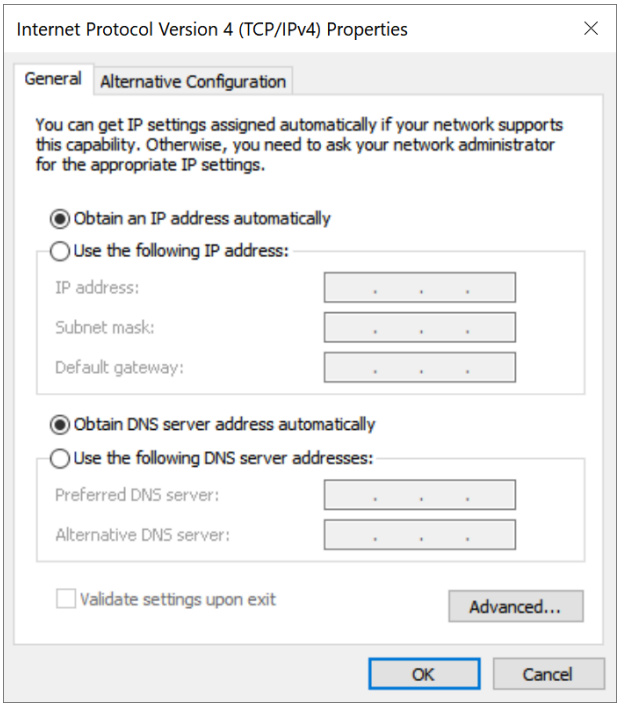


Figure 1.20. Dynamic IP configuration for the host computer

Static Device IP

Using a static IP address for the host computer is necessary to set up a point-to-point network. On top of that, a static device IP configuration can be necessary in the rare cases in which the network card does not support multicast.

1. Connect the 1GbE port of the network card that is dedicated for device connectivity directly to the 1GbE port of the instrument.
2. In the Device tab of the LabOne user interface, enable the setting "Static IP" with the IP Address 192.168.1.10, click on "Program", and restart the instrument using the soft power button.
3. On Windows, 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 field: `--device-ip 192.168.1.10`.

   The LabOne User Interface shortcut Target field should look like this:

   ```
   "C:\Program Files\Zurich Instruments\LabOne\WebServer
   \ziWebServer.exe" --auto-start=1 --server-port=8004 --resource-path
   "C:\Program Files\Zurich Instruments\LabOne\WebServer\html\\" --
   device-ip 192.168.1.10
   ```

   The LabOne Data Server shortcut Target field should look like this:

   ```
   "C:\Program Files\Zurich Instruments\LabOne\DataServer
   \ziDataServer.exe" --device-ip 192.168.1.10
   ```

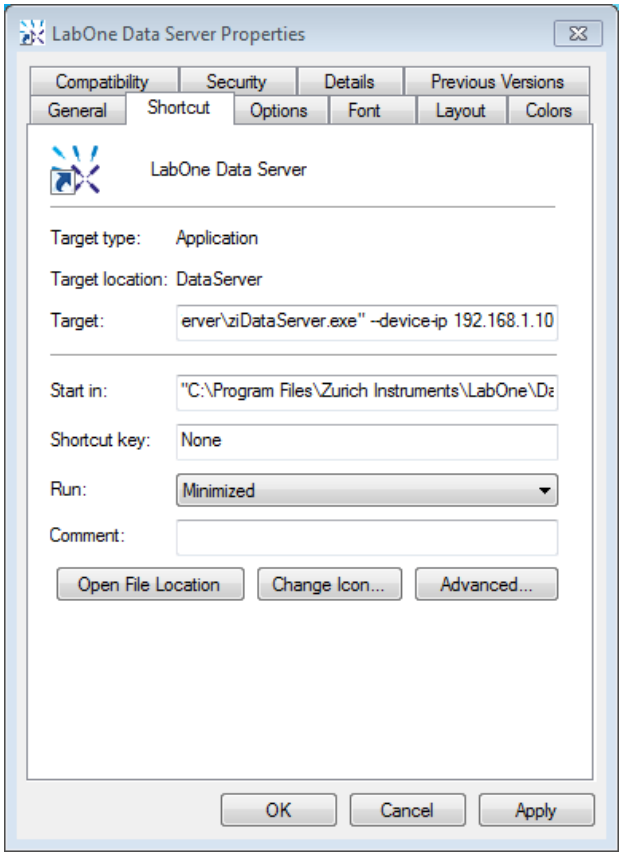


Figure 1.21. Static IP shortcut modification

4. (Optional) To verify the connection between the host computer and the UHF Instrument, open a DOS command window and ping the IP address entered above.

Requirements:

- Device IP must be known
- Needs network administrator support on networks with dynamic IP configuration

1.6. Software Update

1.6.1. Overview

It is recommended to regularly update the LabOne software on the SHFSG 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 Section 1.6.2. If you use one of the LabOne APIs with a separate installer, don't forget to update this part of the software, too.

1.6.2. 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 Section 1.4 . Second, the instrument firmware needs to be updated from the Device Connection dialog after starting up LabOne. This is shown in Section 1.6.3. 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 1.22 will start a download the latest LabOne installer for Windows or Linux, see Figure 1.23. After download, proceed as explained in Section 1.4 to update LabOne.

Figure 1.22. Device Connection dialog: LabOne update available

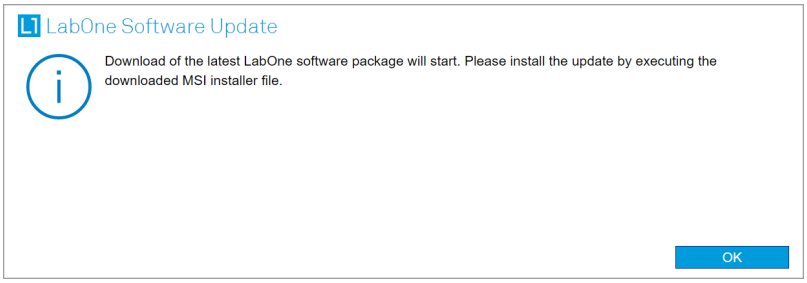


Figure 1.23. Download LabOne MSI using Automatic Update Check feature

1.6.3. 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 1.24. In the Advanced view, there will be a link "Upgrade FW" in the Update column of the Available Devices table. Click on **Upgrade FW**, respectively, to open the firmware update start-up dialog shown in Figure 1.25. The firmware upgrade takes approximately 2 minutes.

Figure 1.24. Device Connection dialog with available firmware update

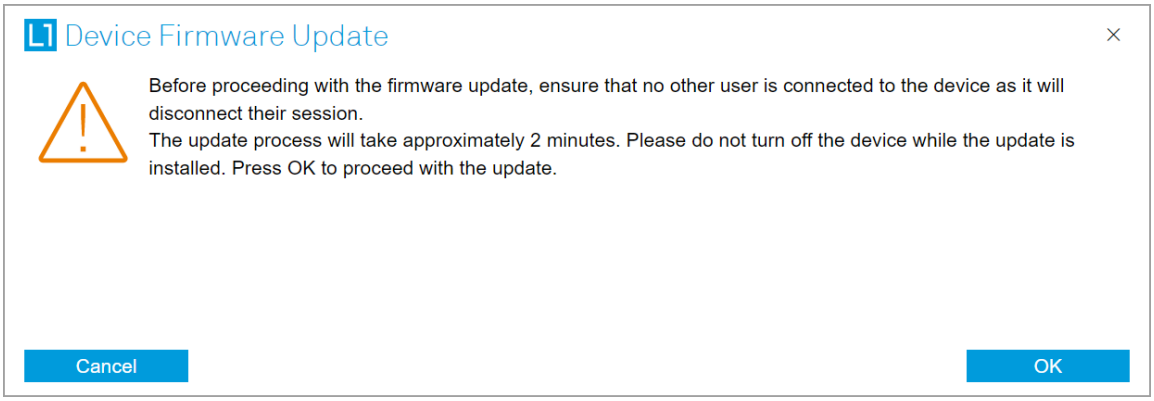


Figure 1.25. Device Firmware Update start-up dialog

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.

1.7. Troubleshooting

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

1.7.1. Common Problems

Your SHFSG 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 1.9. 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.
The Instrument can't be connected over USB	Please verify that the USB port labeled "Maintenance" is connected. The port labeled "USB" is not supported with LabOne 21.08 and will be enabled with a future LabOne release.
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.

Problem	Check item
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.

1.7.2. 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.

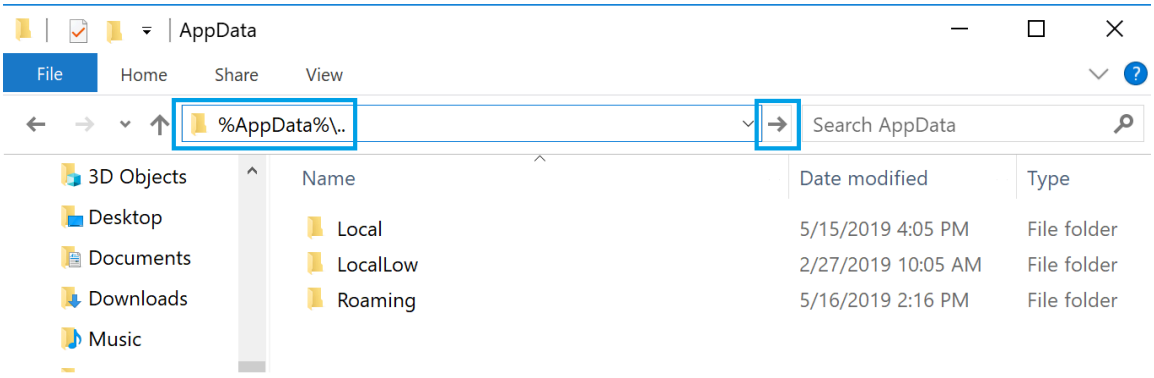


Figure 1.26. Using the `%AppData%\..` shortcut in Windows Explorer to access the hidden folder.

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]`

1.7.3. 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**.

Chapter 2. Functional Overview

This chapter provides the overview of the features provided by the SHFSG Instrument. 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.

2.1. Features

The SHFSG Instrument consists of several internal units that generate digital signals (light blue color) and several units that transform the digital signals into analog signals with the appropriate center frequency (dark blue color). The front panel is depicted on the left-hand side and the back panel is depicted on the right-hand side. The arrows between the panels and the interface units indicate selected physical connections and the data flow. The SHFSG 8.5 GHz Signal Generator comes in a 4-channel and an 8-channel variant, providing either 4 or 8 output channels, respectively. The ordering guide details the available upgrade options for each instrument type and whether the option can be upgraded directly in the field.

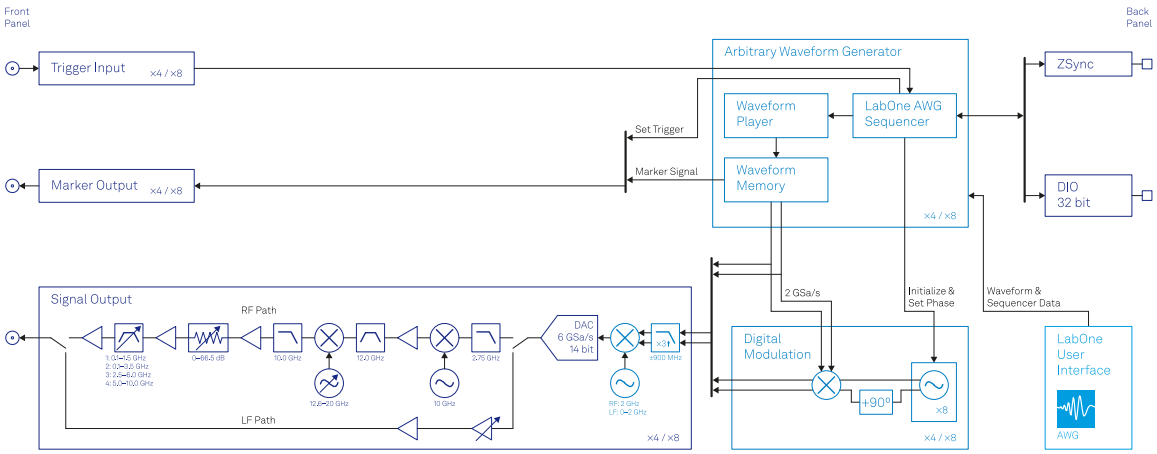


Figure 2.1. SHFSG instrument functional diagram

Each channel has signal generation (**AWG**, **Modulation**) functionality, as well as common functionality such as the shared communications (**32-bit DIO, ZSync**). The digital, complex-valued signal from the signal generation is up-converted to microwave frequencies in the analog domain using the **Signal Output** Module.

2.1.1. Super-high-frequency Signal Outputs

- Low-noise Outputs, DC - 8.5 GHz frequency range, 1 GHz modulation bandwidth
- Broadband double superheterodyne frequency upconversion
- 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

2.1.2. Advanced Pulse Sequencer

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

2.1.3. Hardware Trigger Engine

- 1 Trigger Engine per Channel
- 1 Marker Output and Trigger Input per Channel

2.1.4. 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
- 2 ZSync connectors for clock synchronization and fast data transfer
- Clock input/output connectors (10 MHz)

2.1.5. Software Features

- Web-based, high-speed user interface with multi-instrument control
- Data server with multi-client support
- APIs for C, LabVIEW, MATLAB, or Python-based instrument programming
- Turnkey software and firmware features for fast system tune-up

2.2. Front Panel Tour

The front panel SMA connectors and control LEDs are arranged as shown in Figure 2.2 and listed in Table 2.1.

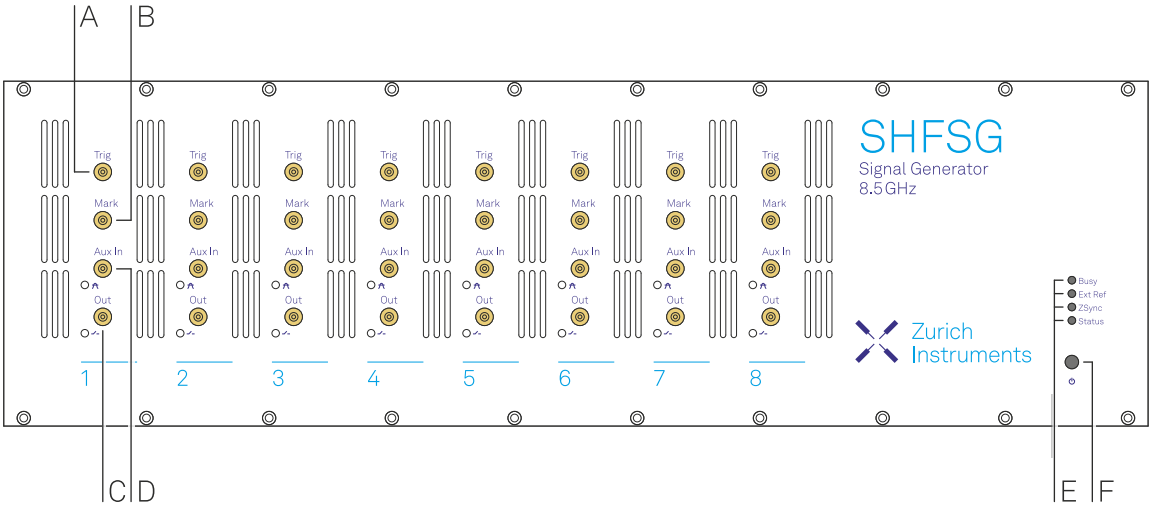


Figure 2.2. SHFSG 8.5 GHz Signal Generator front panel

Table 2.1. SHFSG Signal Generator front panel description

Position	Label / Name	Description	
A	Trig	TTL Trigger Input	
B	Mark	TTL Marker Output	
C	Out	single-ended waveform Signal Output, DC-8.5 GHz, max. 10 dBm	
D	Aux In	analog Auxiliary Input, max. 10 V	
E	multicolor LEDs	off	Instrument off or uninitialized
		blink	all LEDs blink for 5 seconds → indicator used by the Identify Device functionality
	Busy	unused	
	Ext Ref	off	External Reference Signal not present/detected
		blue	External Reference Signal is present and locked on to
		yellow	External Reference Signal present, but not locked on to
		red	External Reference Signal present, but lock failed
	ZSync	off	no connection
		blue	▬ steady: ZSync fully connected AND synchronized ▬ blinking: ZSync synchronized but not yet fully connected
		yellow	ZSync plugged in, but not connected
		red	ZSync interface error
	Status	off	Instrument off or uninitialized
		blue	Instrument is initialized and has no warnings or errors

Position	Label / Name	Description
		yellow Instrument has warnings red Instrument has errors
F	⏻ Soft power button	Power button with incorporated status LED off Instrument off and disconnected from mains power blue - 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 red - 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

2.3. Back Panel Tour

The back panel is the main interface for power, control, service and connectivity to other ZI instruments. Please refer to Figure 2.3 and Table 2.2 for the detailed description of the items.

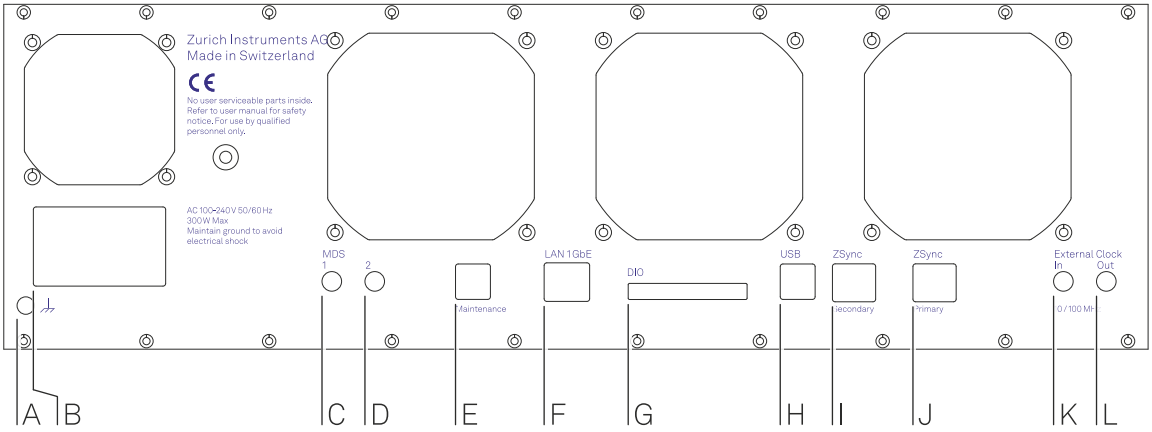


Figure 2.3. SHFSG Instrument back panel

Table 2.2. SHFSG 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	SMA: bidirectional TTL ports for multi-device synchronization
D	MDS 2	SMA: bidirectional TTL ports for multi-device synchronization
E	Maintenance	Universal Serial Bus (USB) 3.0 port for maintenance and instrument control
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 connector → do not use for standard operation
I	ZSync Secondary	Secondary inter-instrument synchronization bus connector - 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

2.4. Ordering Guide

Table 2.3 provides an overview of the available SHFSG products. Upgradeable features are options that can be purchased anytime without the need to send the Instrument back to Zurich Instruments.

Table 2.3. SHFSG Instrument product codes for ordering

Product code	Product name	Description	Field upgrade possible
SHFSG	SHFSG Signal Generator	Base 4-channel Super-High-Frequency Signal Generator (SHFSG)	-
SHFSG	SHFSG Signal Generator	Base 8-channel Super-High-Frequency Signal Generator (SHFSG)	-

Table 2.4. Product selector SHFSG

Feature	SHFSG 4 channels	SHFSG 8 channels
Number of Output Channels	4	8
Number of analog RF synthesizers	4	4
Digital oscillators per channel	8	8
Mixer-calibration-free analog frequency upconversion (double super-heterodyne)	yes	yes
Frequency range	DC-8.5 GHz	DC-8.5 GHz
Total number of Markers/Triggers (1 each per channel)	4/4	8/8
Vertical resolution Output	14 bit	14 bit
Digital IQ modulation	yes	yes
ZSync capability	yes	yes
Sequencing	yes	yes
USB 3.0	yes	yes
LAN 1 Gbit/s	yes	yes

Chapter 3. Tutorials

The tutorials in this chapter have been created to allow users to become more familiar with the operation of the SHFSG Signal Generator. Tutorials also provide tips and tricks to optimally program the instrument.

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

Note

In many tutorials, we use the Python API to control the instruments.

Note

For all tutorials, you must have LabOne installed as described in the chapter Getting Started.

Note

The documentation is frequently updated to match with the latest functionality of the LabOne software and to extend the number of example use cases covered. For the latest version of the documentation, please always refer to the online documentation.

3.1. Basic Sine Generation

Note

This tutorial is applicable to all SHFSG Instruments.

3.1.1. Goals and Requirements

The goal of this tutorial is to demonstrate basic sine generation with the SHFSG. 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.

3.1.2. 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.

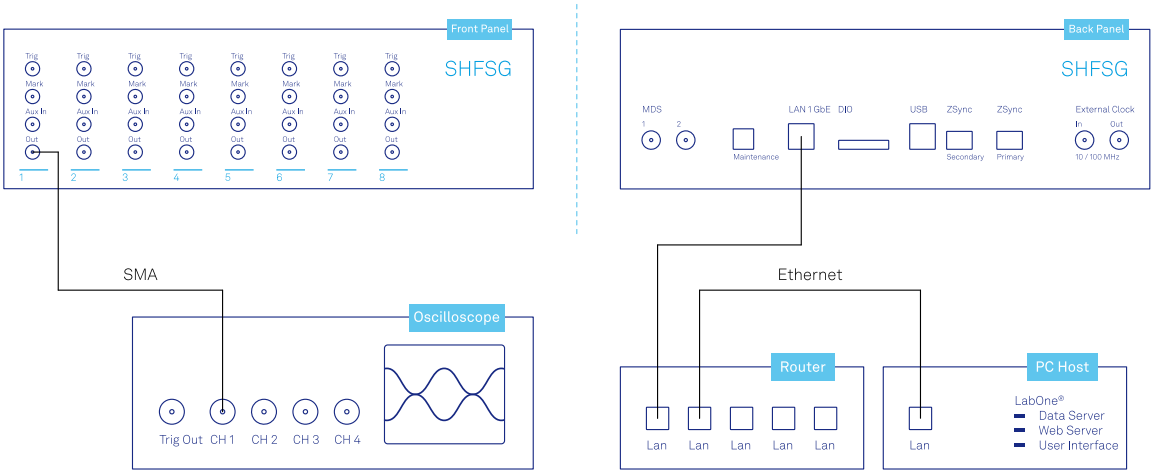


Figure 3.1. Connections for the basic sine generation tutorial

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).

3.1.3. 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 Output Tab to enable the Output of the SHFSG 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 3.1. Settings: enable the output

Tab	Section	Label	Setting / Value / State
Output	Signal Output 1	On	ON
Output	Signal Output 1	Range (dBm)	10
Output	Channel 1	Center Freq (Hz)	1.0 G
Output	Signal Output 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.

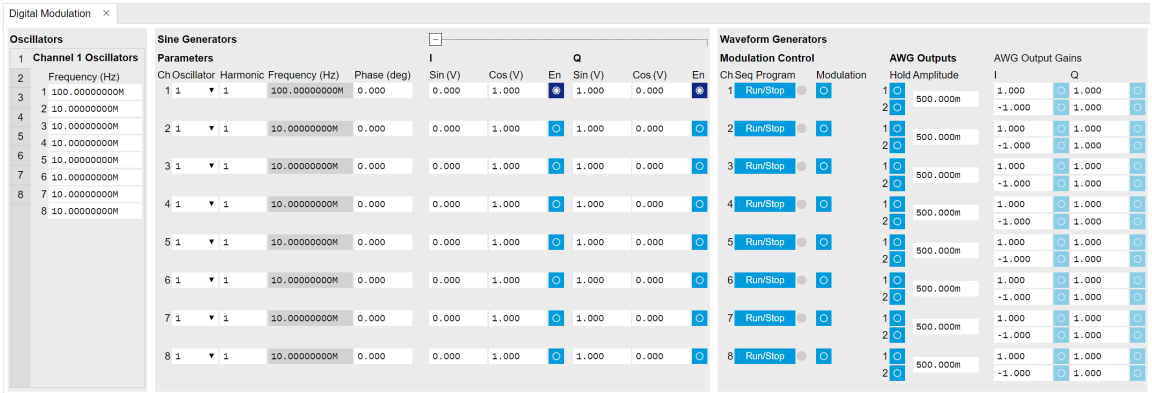


Figure 3.2. LabOne UI: Digital Modulation tab

Table 3.2. 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

Tab	Section	Sub-section	Label	#	Setting / Value / State
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.

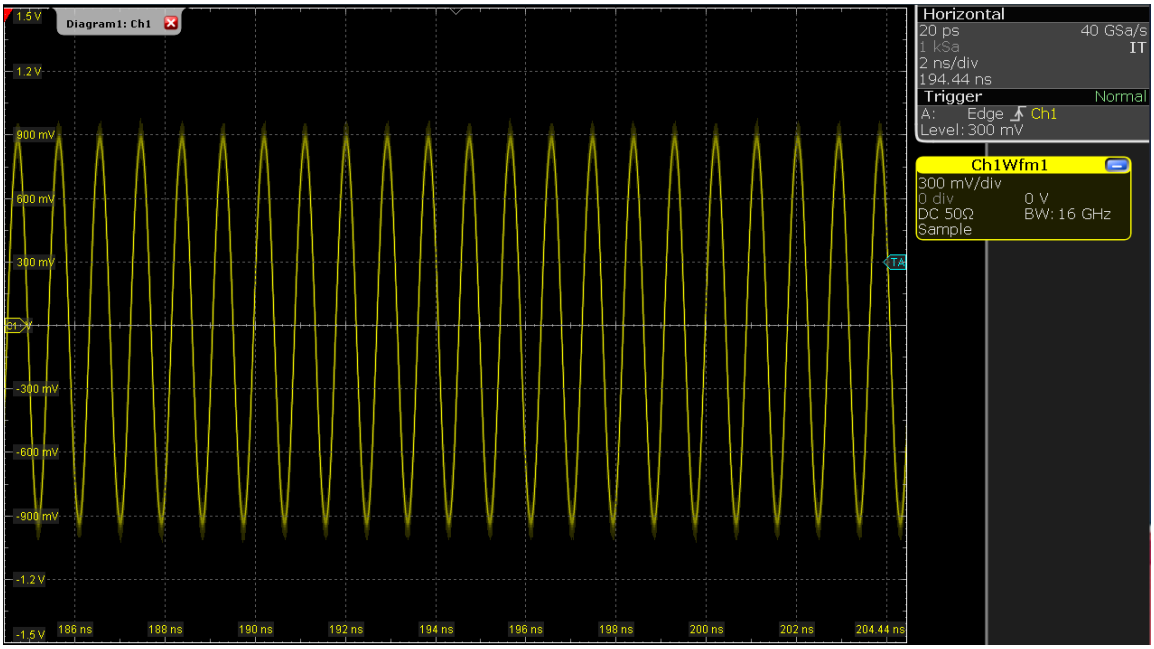


Figure 3.3. Scope trace of a 1.1-GHz signal

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.

3.2. Basic Waveform Playback

Note

This tutorial is applicable to all SHFSG Instruments.

3.2.1. Goals and Requirements

The goal of this tutorial is to demonstrate the basic use of the SHFSG, 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.

3.2.2. 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.

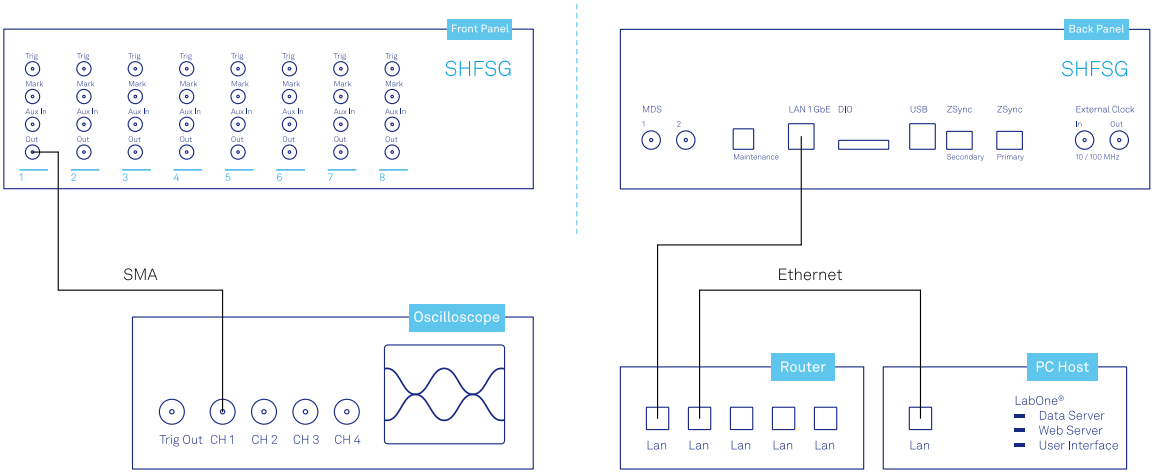


Figure 3.4. Connections for the arbitrary waveform generator basic playback tutorial

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).

3.2.3. 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 SHFSG 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 0.1 - 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 3.3. Settings: enable the output

Tab	Sub-tab	Label	Setting / Value / State
Output	Signal Output 1	On	ON
Output	Signal Output 1	Range (dBm)	10

Tab	Sub-tab	Label	Setting / Value / State
Output	Channel 1	Center Freq (Hz)	1.0 G
Output	Signal Output 1	Output Path	RF

Table 3.4. 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

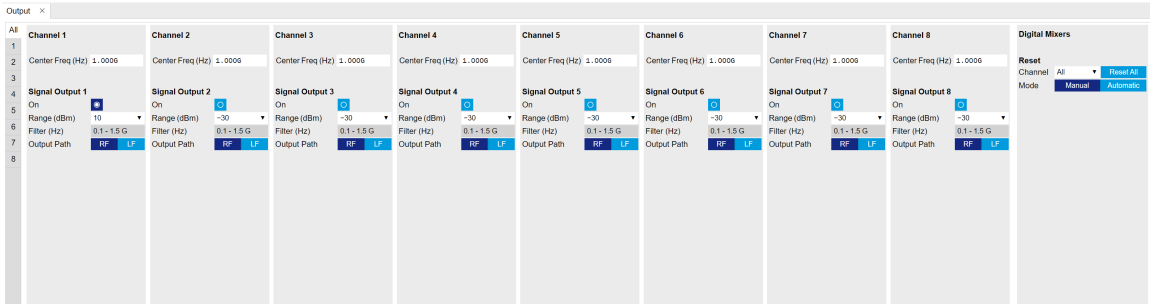


Figure 3.5. LabOne UI: Output tab

In the Output Tab, we configure the first 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 3.5. 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: $V_p = \sqrt{2 * 10^{P\max / 10} * 10^{-3}\,\mathrm{W} * 50\,\Omega}$, 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 $50\,\Omega$ load. To calculate he 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 Output 1. 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. 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. In this case, the hold functionality will hold 0 instead of the last finite value.

By clicking on Save, 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 µs and a carrier frequency of 1.0 GHz, as shown in Figure Figure 3.6.

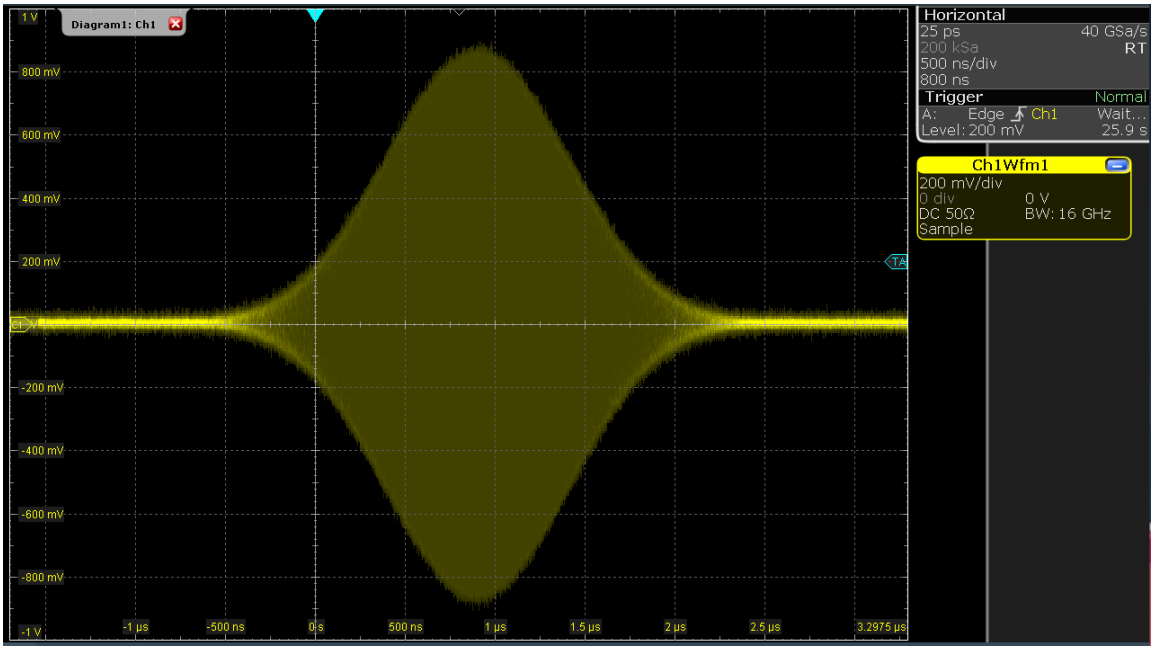


Figure 3.6. Scope shot of a Gaussian pulse generated by the AWG

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 3.7.

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

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

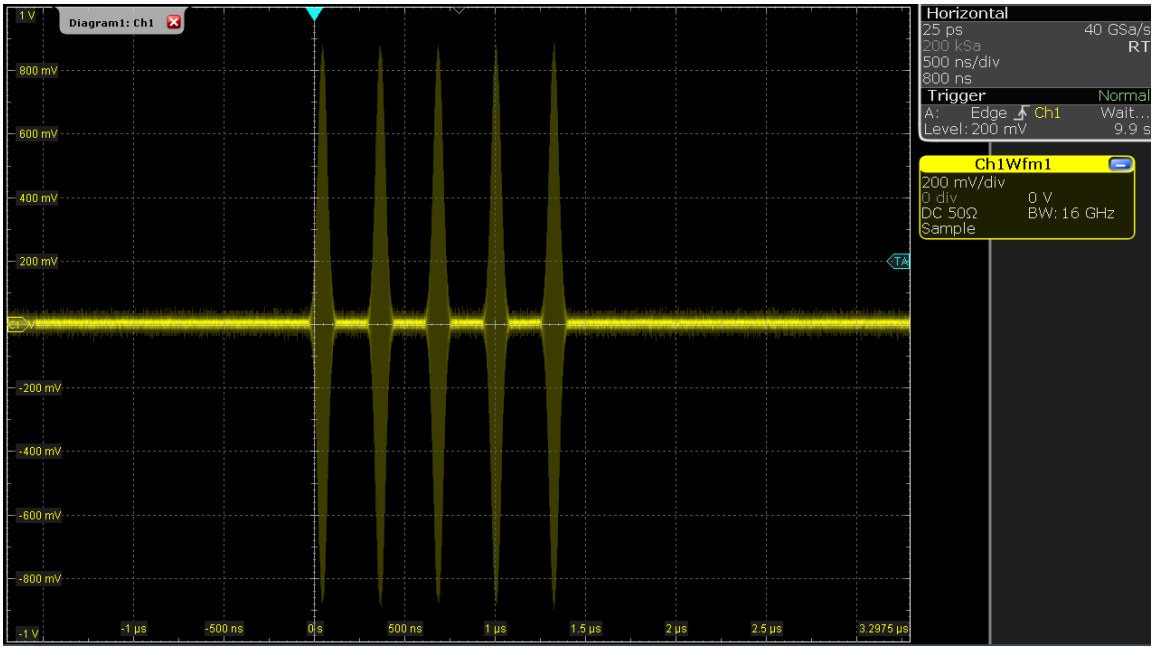


Figure 3.7. Burst of Gaussian pulses generated by the AWG and captured by the scope

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 Figure 3.8.

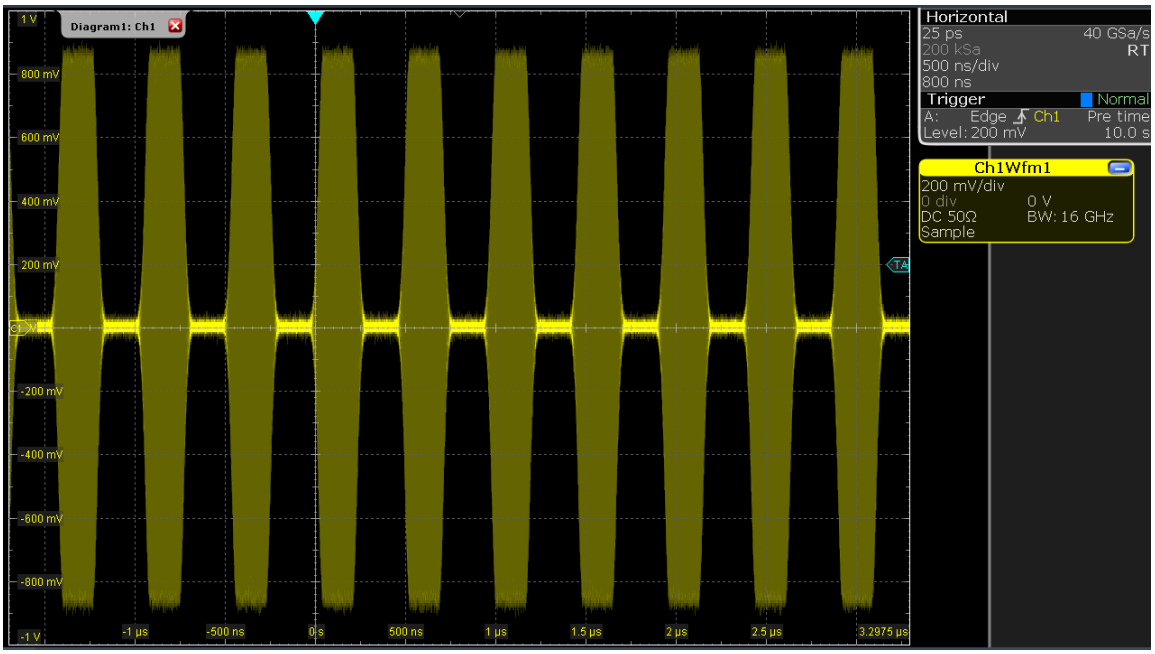


Figure 3.8. Scope shot of an infinite pulse series generated by the AWG

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.

3.3. Triggering and Synchronization

Note

This tutorial is applicable to all SHFSG Instruments.

3.3.1. Goals and Requirements

The goal of this tutorial is to show how to use the SHFSG as a trigger source, as well as how to configure the SHFSG to respond to an external trigger. In order to visualize the multi-channel signals, an oscilloscope with sufficient bandwidth and channel number is required.

3.3.2. 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.

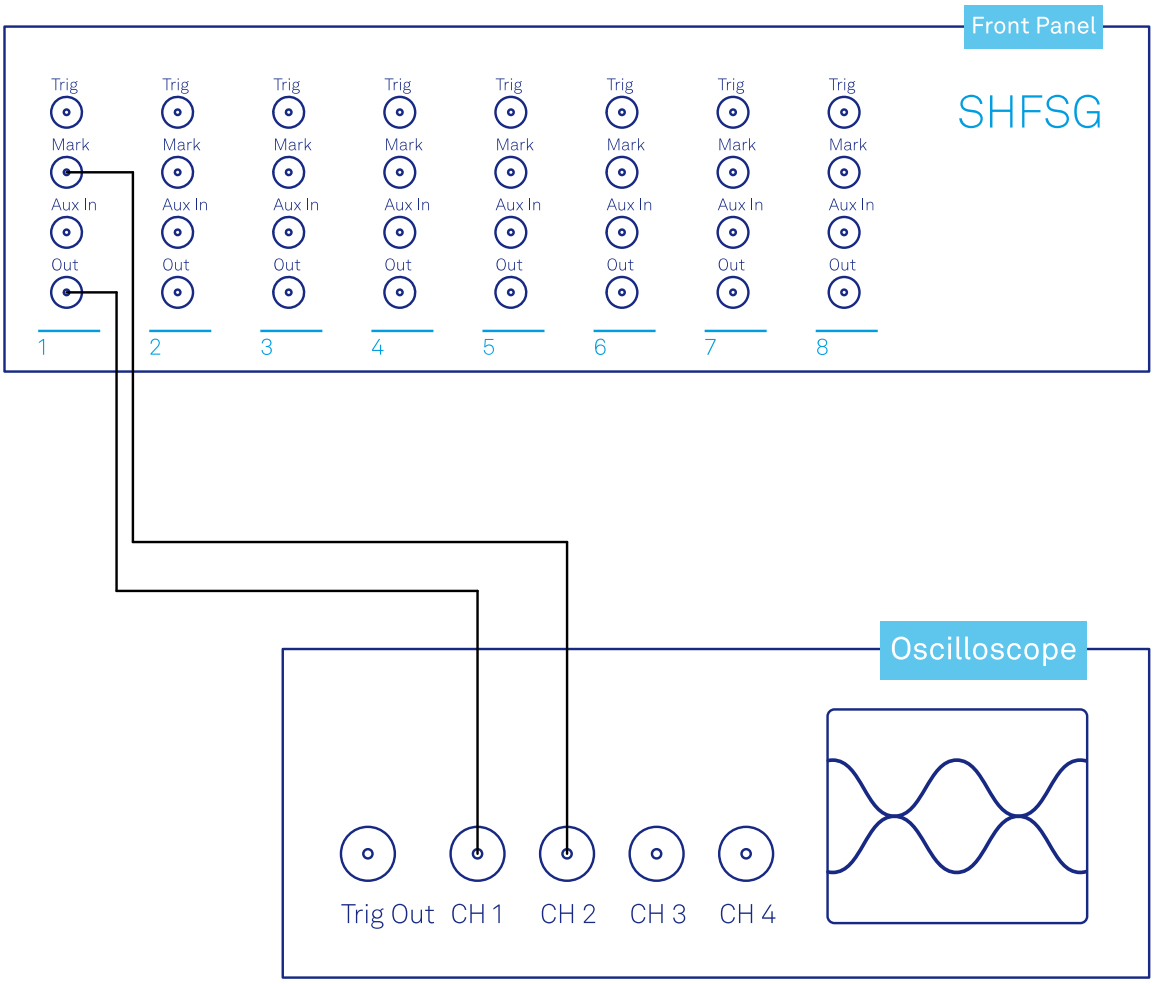


Figure 3.9. Connections for the arbitrary waveform generator triggering and synchronization tutorial

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).

3.3.3. 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 AWG with an external TTL signal

Generating Markers with the AWG

To begin with, we generate a trigger output with channel 1. As this tutorial is an extension of the Basic Waveform Playback Tutorial, configure the SHFSG as follows:

Table 3.6. Settings: configure the output

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

Table 3.7. 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	500 ns/div
Trigger source	Ch2
Trigger level	200 mV
Run / Stop	ON

After configuring the output using the table above, we use the SHFSG to generate a trigger output. There are two ways of generating trigger output signals with the 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 3.8. 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 Mark 1.

Table 3.9. 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 3.10 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.

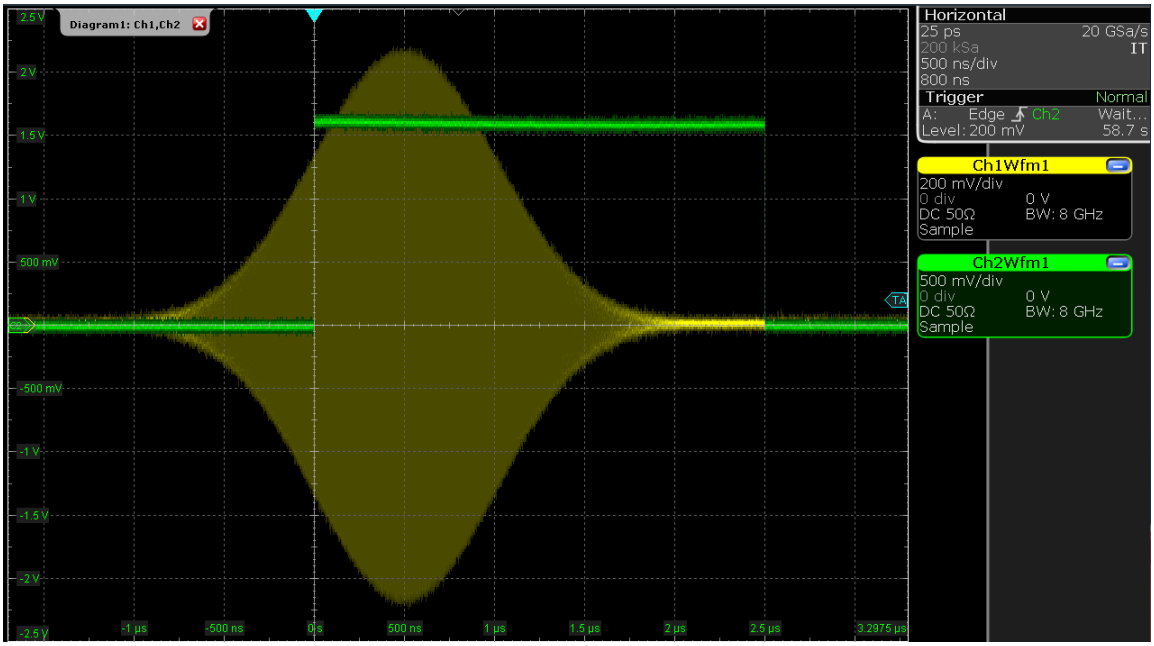


Figure 3.10. Gaussian pulse and square marker signal generated by the AWG and captured by the scope

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 3.10. Settings: configure the AWG trigger output

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

Figure 3.11 shows the AWG signal captured by the scope. This looks very similar to Figure 3.10 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.

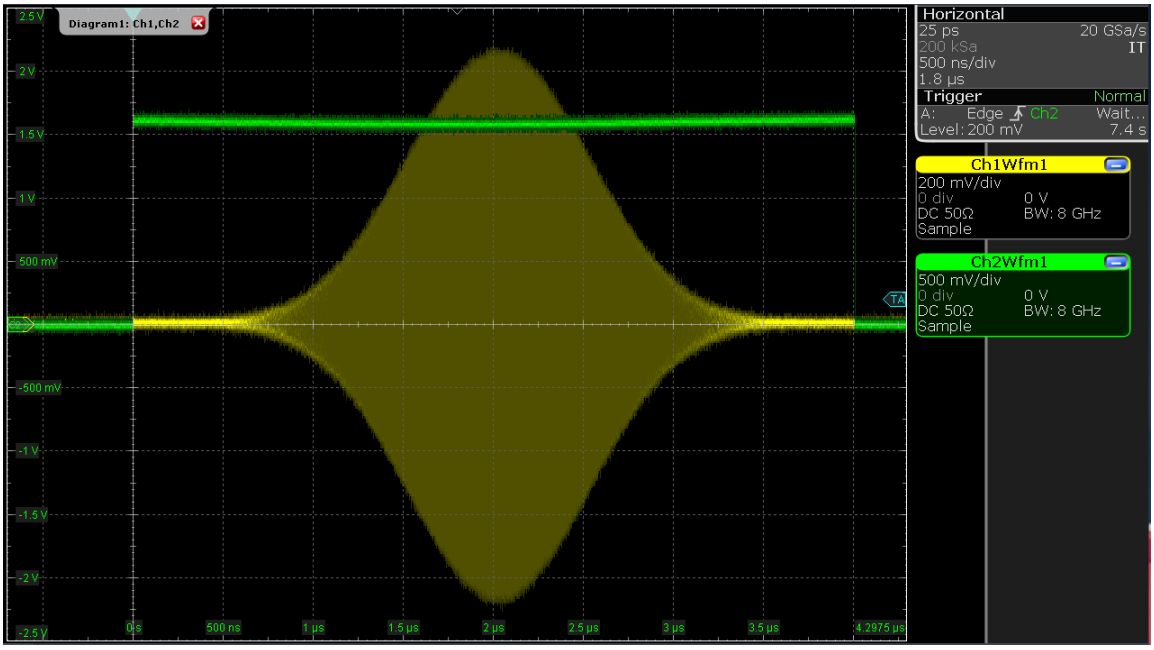


Figure 3.11. Gaussian pulse and trigger signal generated by the AWG and captured by the scope

Triggering the AWG

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

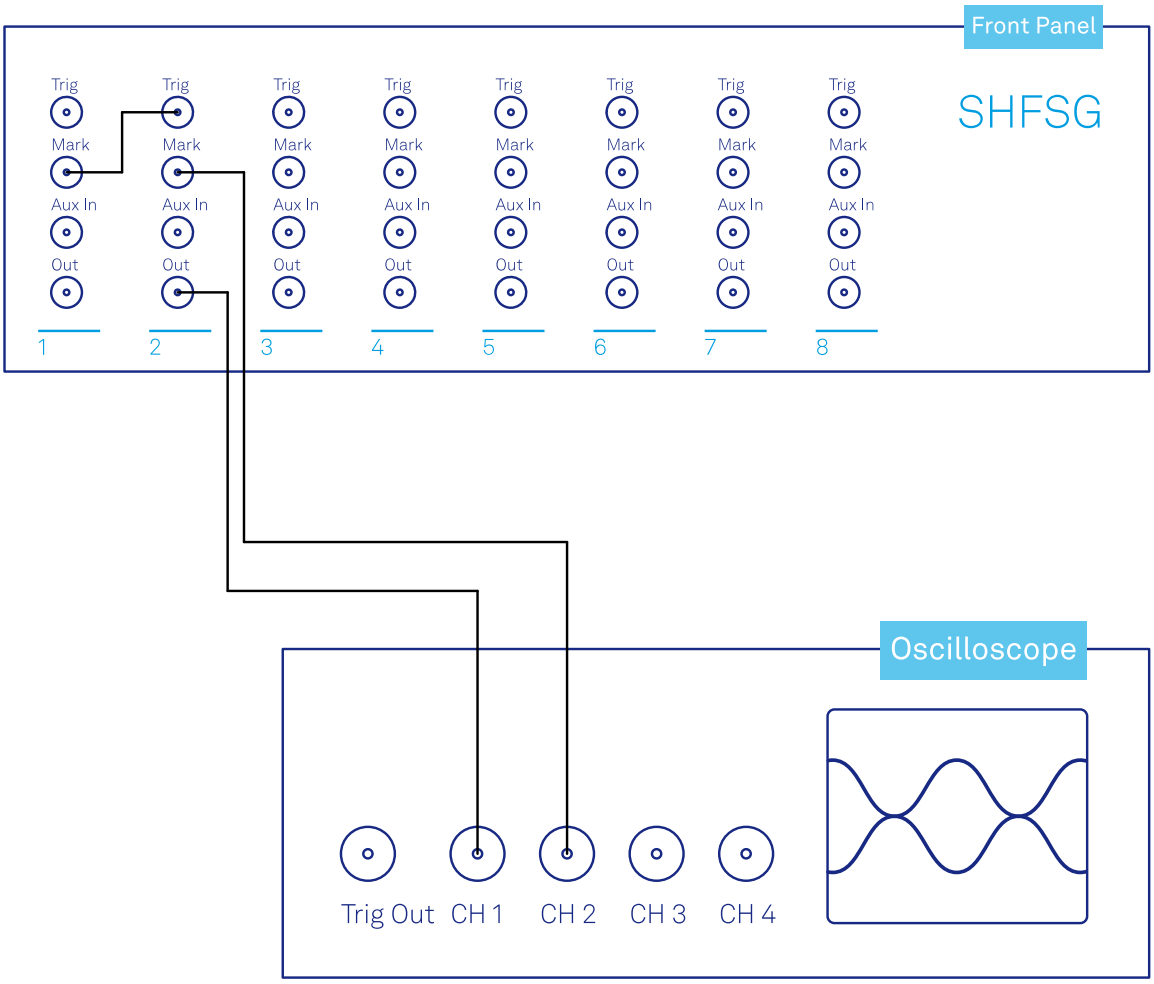


Figure 3.12. Connections for the arbitrary waveform generator basic playback tutorial

In this section we show how to trigger the AWG with an external TTL signal. We start by using Channel 1 of the SHFSG to generate a periodic TTL signal. As shown in Figure 3.12, 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 3.11. 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
DIO		Marker Out	2	Signal	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 3.12. 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 3.13 shows the pulse series as seen on the scope: the pulses are now spaced by the oscillator period of 8 µs, 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.

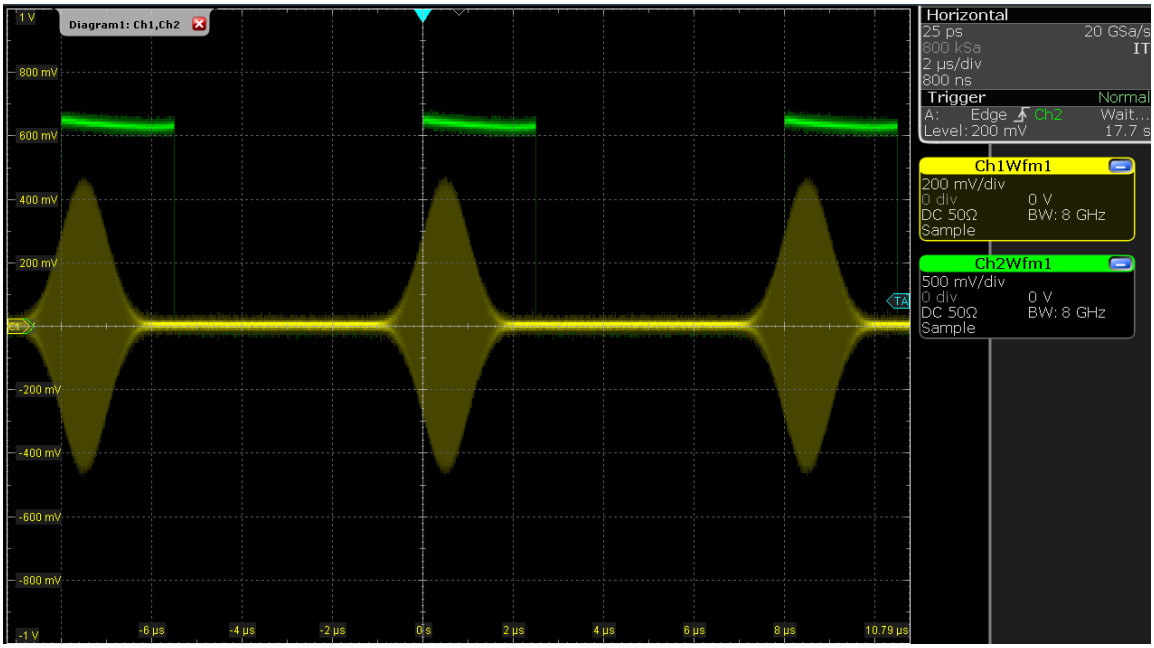


Figure 3.13. Externally triggered pulse series generated by the AWG.

3.4. Digital Modulation

Note

This tutorial is applicable to all SHFSG Instruments.

3.4.1. Goals and Requirements

The goal of this tutorial is to demonstrate the use of the digital modulation feature of the AWG. 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.

3.4.2. 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.

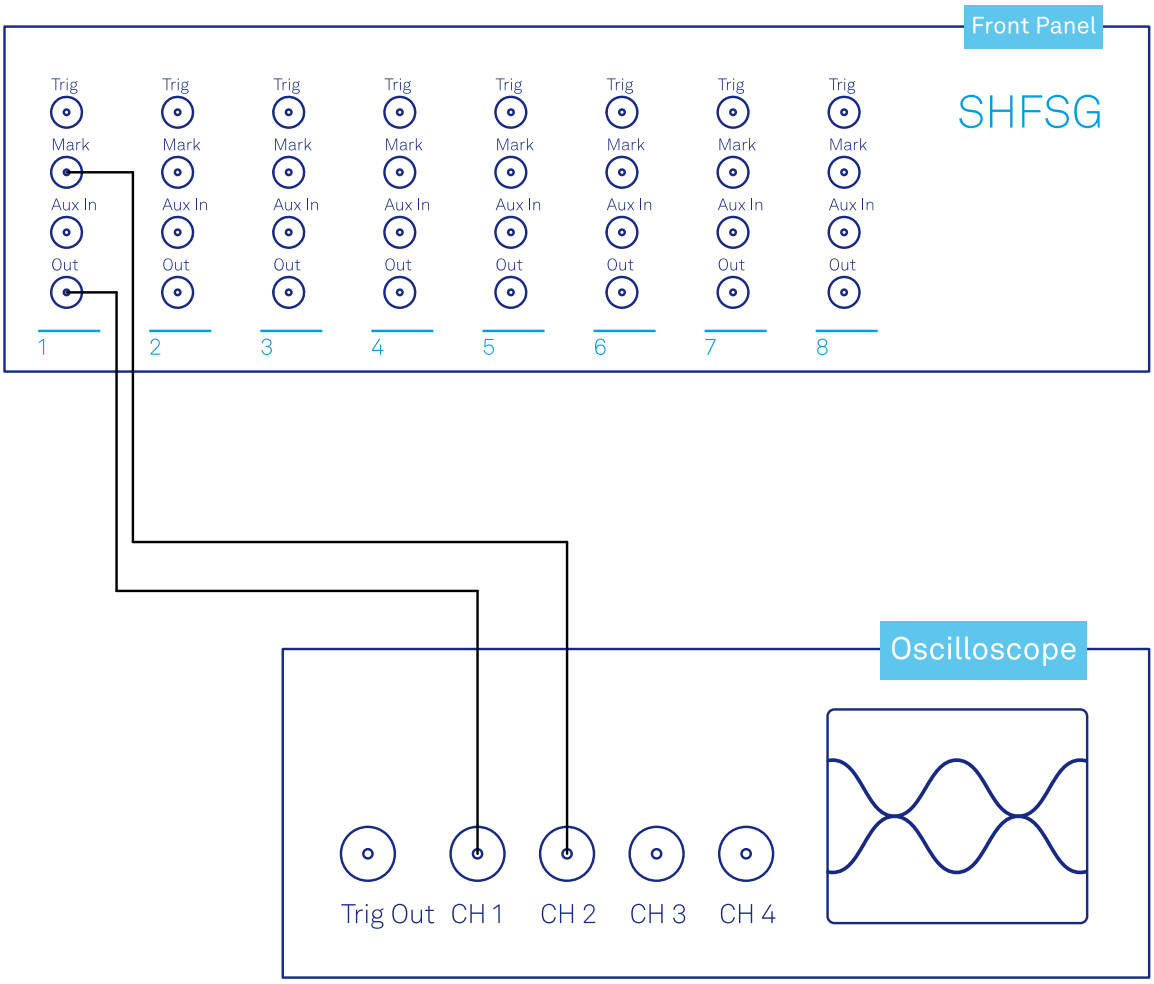


Figure 3.14. Connections for the arbitrary waveform generator digital modulation tutorial

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.

3.4.3. 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 SHFSG consists of a chain of several conversion steps, which can be summarized in a simple model for the generated RF voltage:

$$V_{\mathrm{RF}}(t) = V_0 \,\mathrm{Re}\Big[A(w_I(t) + i w_Q(t)) e^{+i\phi} e^{+i2\pi f_{\mathrm{Osc}} t} e^{+i2\pi f_{\mathrm{RF}} t}\Big], \qquad (1)$$

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 $V_0 = \sqrt{2 * 10^{\mathrm{Pmax}\,/\,10} * 10^{-3}\,\mathrm{W} * 50\,\Omega}$ is the maximum output voltage determined by the range setting P_{max} with a $50\,\Omega$ 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:

$$\begin{aligned} V_{\mathrm{RF}}(t) = V_0 A \big[w_I(t)\cos(2\pi f_{\mathrm{Osc}} t + \phi) - w_Q(t)\sin(2\pi f_{\mathrm{Osc}} t + \phi)\big]\cos(2\pi f_{\mathrm{RF}} t) \\ -\big[w_I(t)\sin(2\pi f_{\mathrm{Osc}} t + \phi) + w_Q(t)\cos(2\pi f_{\mathrm{Osc}} t + \phi)\big]\sin(2\pi f_{\mathrm{RF}} t). \end{aligned} \qquad (2)$$

We now look at how the SHFSG 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.

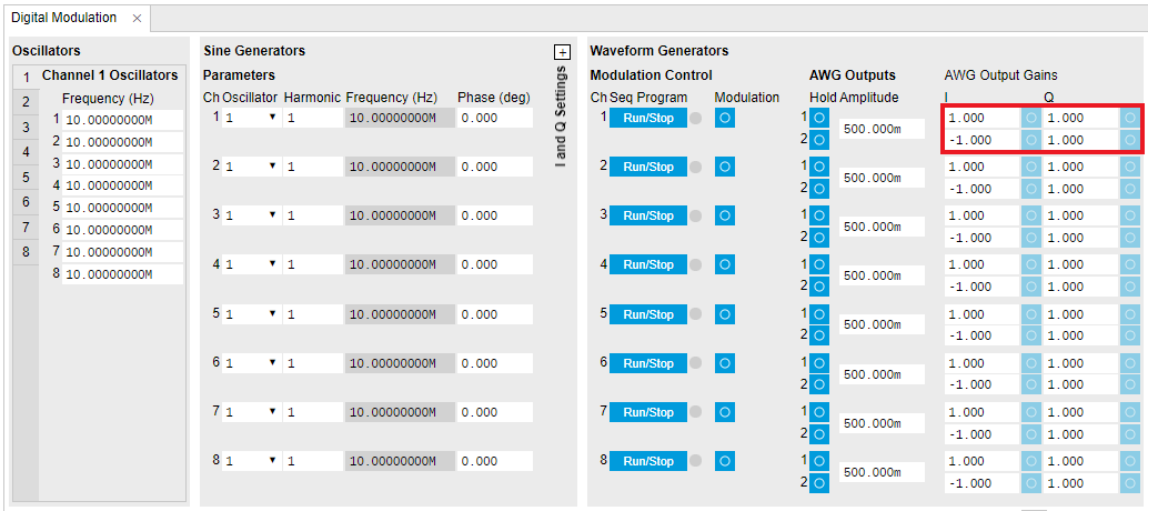


Figure 3.15. Digital Modulation tab in the LabOne UI

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 3.16 shows how the different parts of the `playWave` command map to the different gain nodes in the digital modulation process.

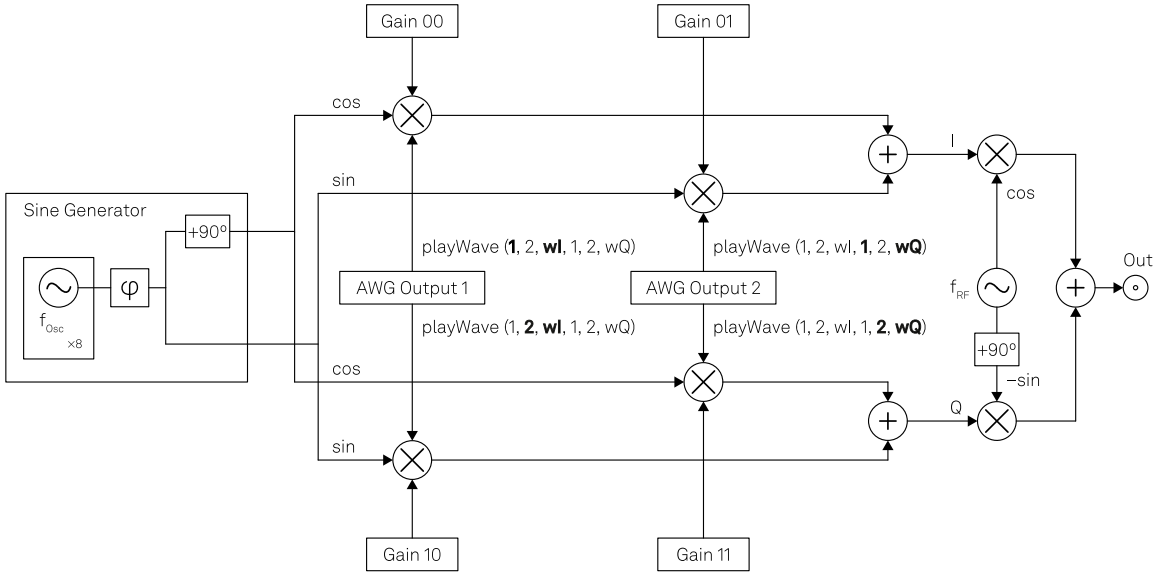


Figure 3.16. Diagram of digital modulation processing chain and settings. Bolded parts of the text show how the `playWave(1,2,wI,1,2,wQ)` command routes the waveform envelopes to different gain nodes and cosine/sine terms to generate a complex signal.

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

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

where $\mathrm{Gain}ij$ of Channel `n` corresponds to the node path `<dev>/SGCHANNELS/<chan>/AWG/OUTPUTS/<i>/GAINS/<j>`. The choice of $\mathrm{Gain00} = \mathrm{Gain10} = \mathrm{Gain11} = 1.0$ and $\mathrm{Gain01} = -1.0$ yields the same expression as in ??? 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 Section 4.2.1.

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 SHFSG 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 SHFSG 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 $\mathrm{Gain01}$ and $\mathrm{Gain10}$.

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 3.13. 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	$\mathrm{Gain}ij$	`<dev>/SGCHANNELS/<chan>/AWG/OUTPUTS/<i>/GAINS/<j>`
Global amplitude	A	`<dev>/SGCHANNELS/<chan>/AWG/OUTPUTAMPLITUDE`
Oscillator frequency	f_{osc}	`<dev>/SGCHANNELS/<chan>/OSCS/<osc>/FREQ`
Sine generator phase	ϕ	`<dev>/SGCHANNELS/<chan>/SINES/0/PHASESHIFT`
RF center frequency	f_{RF}	`<dev>/SYNTHESIZERS/<synth>/CENTERFREQ`
Output range	$P_{\max}$	`<dev>/SGCHANNELS/<chan>/OUTPUT/RANGE`

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 SHFSG outputs and configure the sine generator, as well as to configure the external scope.

Table 3.14. Settings: enable the output

Tab	Section	Sub-Section	Label	Setting / Value / State
Output	Signal Output 1		On	ON
Output	Signal Output 1		Range (dBm)	10

Tab	Section	Sub-Section	Label	Setting / Value / State
Output	Channel 1		Center Freq (Hz)	1.0 G
Output	Signal Output 1		Output Path	RF
Digital Modulation	Waveform Generators	Modulation	1	ON
Digital Modulation	AWG Outputs	Amplitude		0.5
Digital Modulation	AWG Output Gains	I	1	1.0
Digital Modulation	AWG Output Gains	I	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	I	En	OFF
Digital Modulation	Sine Generators	Q	En	OFF
DIO	Marker	Source	1	Output 1 Marker 1

Table 3.15. 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 SHFSG 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 output channel of the SHFSG 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 3.16. 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 3.17 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.

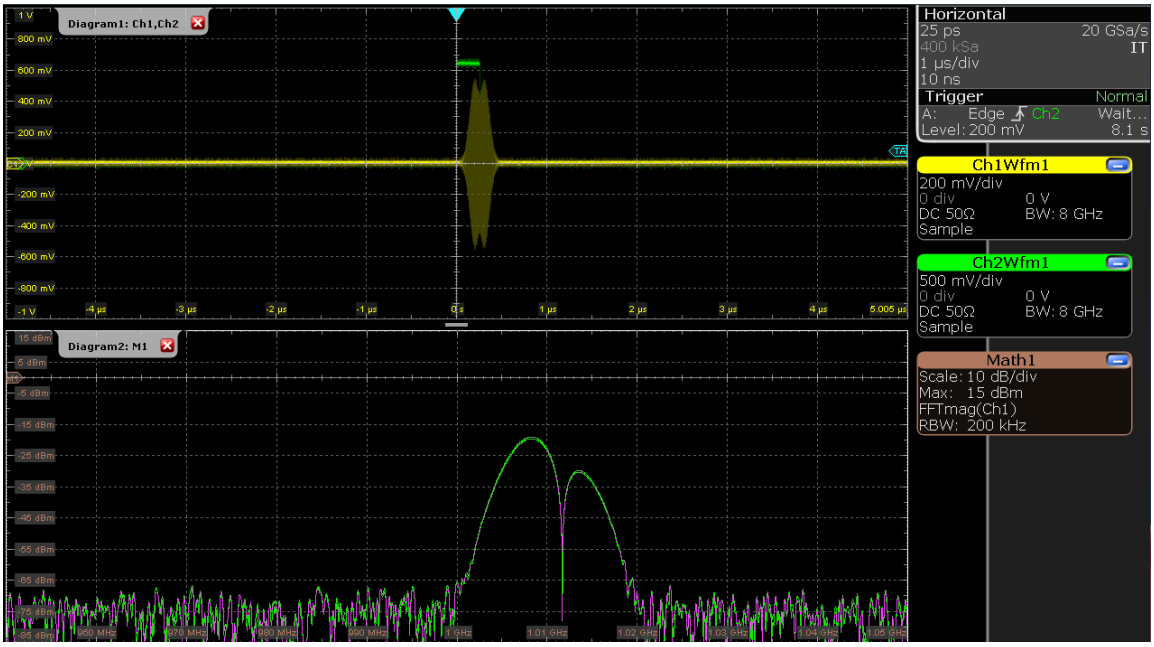


Figure 3.17. Dual-channel signal generated by the AWG and captured by the scope. The top half of the figure shows a pulse that is a combination of a Gaussian pulse and a derivative of a Gaussian pulse, modulated at 10 MHz and upconverted with an RF center frequency of 1.0 GHz. The bottom part of the 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, though `playWave(1,2,wI,1,2,wI)` can also be used. 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)` or `playWave(1,2,wI,1,2,wI)` 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 SHFSG 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)` or `assignWaveIndex(1,2,wI,1,2,wI,0)` for dual-channel signals. For more details, see the Command Table Tutorial.

3.4.4. Rapid Phase Changes

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

Configure the scope according to the following settings.

Table 3.17. 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	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 3.18. Settings: enable the output

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

Figure 3.18 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.

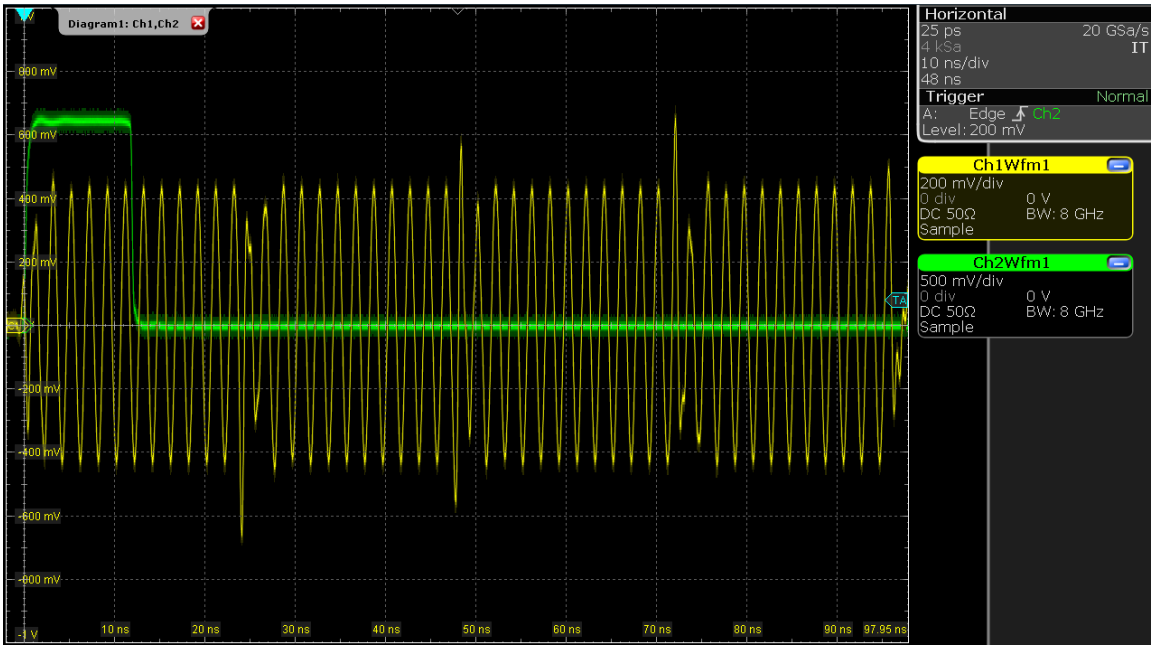


Figure 3.18. Amplitude-modulated dual-channel signal with rapid real-time phase increments generated by the SHFSG.

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.

3.4.5. 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, 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 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.

3.5. Using the Python API

Note

This tutorial is applicable to all SHFSG Instruments.

3.5.1. Goals and Requirements

The previous tutorials showed how to use the SHFSG with the LabOne user interface. However, APIs provide an important alternative method to controlling the SHFSG. In this tutorial, we focus on the Python API, showing how to use it to connect to the SHFSG, as well as how to upload and play a sequence that uses user-defined waveforms. 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

3.5.2. 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.

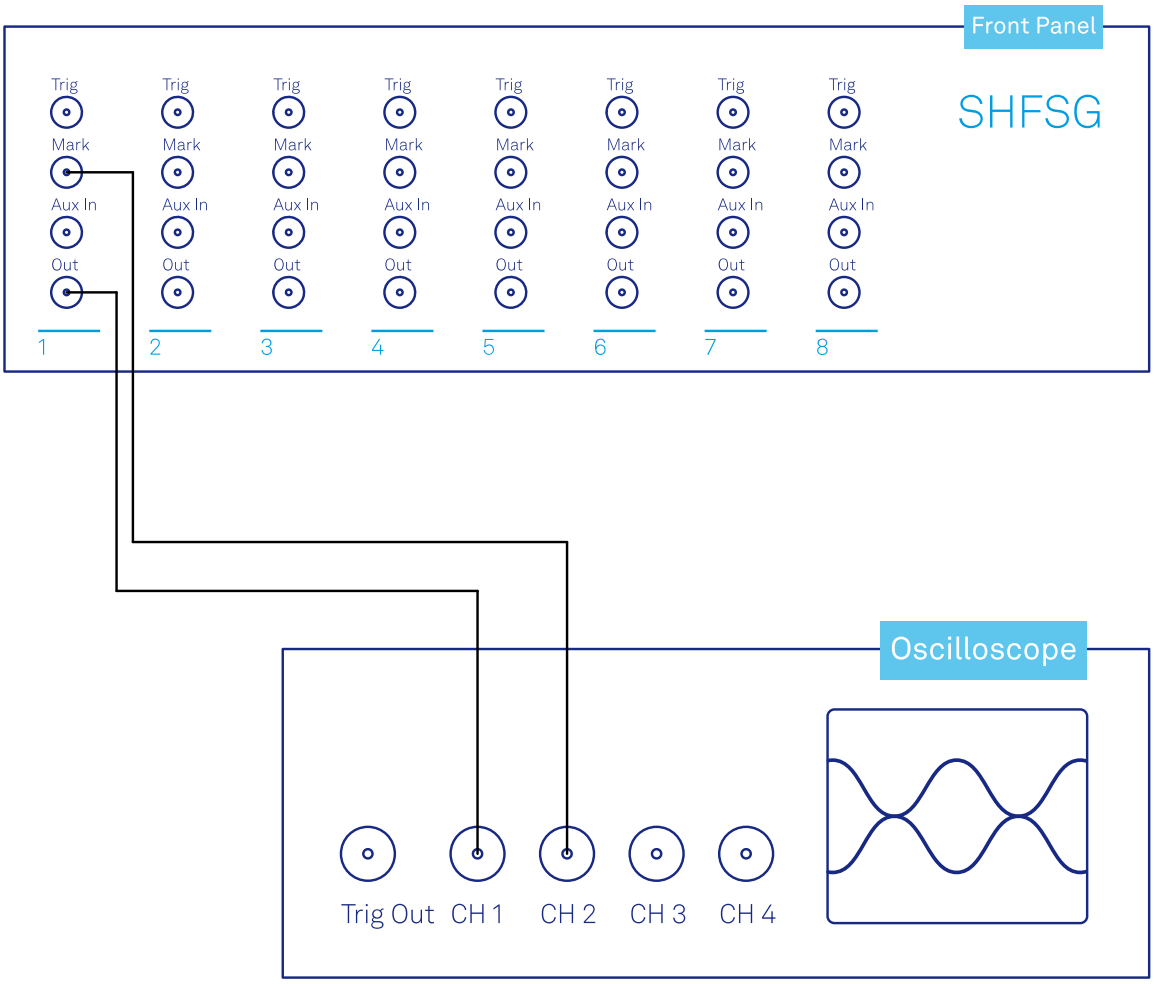


Figure 3.19. Connections for the arbitrary waveform generator Python tutorial

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.

3.5.3. Connecting to to the instrument

First we connect to the SHFSG using Python. For this we first open a **daq**-session and then connect to the instrument using the following code and by replacing **devXXXXX** with the id of our SHFSG instrument, e.g. **12050**:

```
# Load the LabOne API and other necessary packages
```

```
import zhinst.core
import zhinst.utils as utils
import time

device_id='devXXXXX'
apilevel_example=6

server_host = 'localhost'
server_port = 8004

## connect to data server
daq = zhinst.core.ziDAQServer(host=server_host, port=server_port, api_level=6)

## connect to device
device_interface = '1gbe'
daq.connectDevice(device_id, device_interface)
```

Defining the data server and port allows users to connect to the instrument in the local network when using `local_host` 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 version is compatible with the LabOne Python API version:

```
utils.api_server_version_check(daq)
```

If it returns `True`, we are now in the position to access the device. If it returns `False`, please update your API version.

Often the first parameters that need to be set for every experiment are the Center Frequency and Range of the Output Channel. To see the parameter updates that will be performed by the Python script, open the Output 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 Channel 1 to 6 GHz and the Output Range to 10 dBm.

```
import zhinst.deviceutils.shfsg as shfsg_utils

chan_index=0
synth=0

rf_frequency = 6.0 # GHz
daq.setDouble(f'/{device_name}/SYNTHESIZERS/{synth}/CENTERFREQ', rf_frequency*1e9)
output_range = 10.0
daq.setDouble(f'/{device_name}/SGCHANNELS/{chan_index}/OUTPUT/RANGE',
 output_range)
```

Observe how the corresponding GUI values in the first panel of the Output tab change their values correspondingly. Note that in both 4- and 8-channel variants of the SHFSG, there are 4 synthesizers. In the 8-channel variant, neighboring channels (1&2, 3&4, etc.) therefore share one synthesizer, which is why the synthesizer frequencies are set using the `/{device_name}/SYNTHESIZERS/{synth}/CENTERFREQ` node and not within the `/{device_name}/SGCHANNELS/` branch. To check which synthesizer is being used by a particular SGCHANNEL, you can use the `get` command:

```
daq.getInt(f'/{device_name}/SGCHANNELS/{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 `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

```
daq.setDouble(f'/{device_name}/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 through reading the node value in Python using the `get` command:

```
daq.getDouble(f'/{device_name}/SYNTHESIZERS/{synth}/CENTERFREQ')
```

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

```
# Base for node path
path = f"/{device_name}/sgchannels/{chan_index}/"

#determine which synthesizer is used by the desired channel
synth = daq.getInt(path + "synthesizer")

settings = [
    # RF output settings
    [path + "output/range", 10], #output range in dBm
    [path + "output/rflfpath", 1], #use RF path, not LF path
    #set the corresponding synthesizer frequency in Hz
    ["/%s/synthesizers/%d/centerfreq" % (device_id,synth), 6.0e9],
    [path + "output/on", 1], #enable output

    # Digital modulation settings
    [path + "awg/outputamplitude", 0.5], #set the amplitude for the AWG outputs
    [path + "oscs/0/freq", 10.0d6], #frequency of oscillator 1 in Hz
    [path + "oscs/1/freq", -500.0e6], #frequency of oscillator 2 in Hz
    [path + "awg/modulation/enable", 1], #enable digital modulation

    #use first marker bit of waveform as marker source
    [path + "marker/source", 4]
]

# Set the values
daq.set(settings) #use a transactional set to update all settings
```

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.

3.5.4. 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, 1, 2, w, 10);     // Create a wave table entry with
 placeholder waveform, index 10
playWave(1, 2, w, 1, 2, w);
```

We upload this sequence to the SHFSG using the following Python code:

```
# Import the device-utils for the SHFSG
import zhinst.deviceutils.shfsg as shfsg_utils

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

resetOscPhase(); //reset oscillator phase
playWave(1, 2, w, 1, 2, w);
""

# Upload the sequence
shfsg_utils.load_sequencer_program(daq, device_id, chan_index, awg_seqc)
```

We have made use of the `device-utils` for the SHFSG, which contains 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

#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

#Convert them and send to the instrument
wave_raw = utils.convert_awg_waveform(wave, markers=marker)
daq.set(f'/{device_name}/sgchannels/{chan_index}/awg/waveform/waves/10', wave_raw)
```

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

```
# Enable sequencer with single mode
single = 1
shfsg_utils.enable_sequencer(daq, device_id, chan_index, single)
```

After running the sequence, we observe the signal shown in Figure 3.20 on the scope.

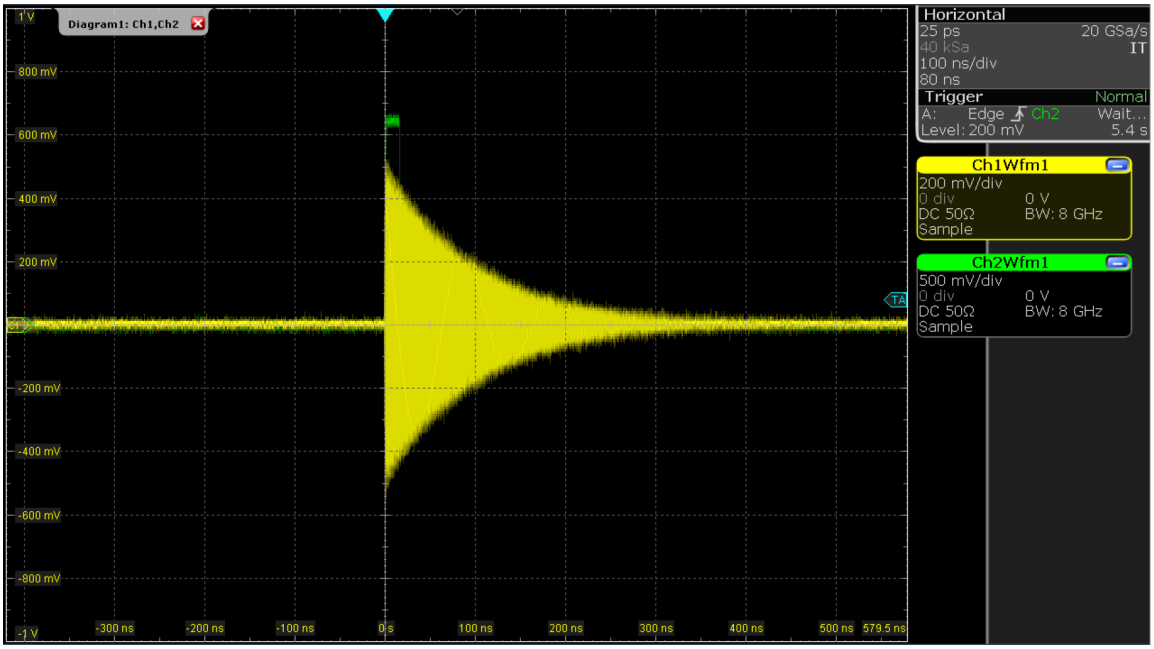


Figure 3.20. Waveform loaded by the API

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, true);
wave w_b = placeholder(LENGTH, true, true);
wave w_c = placeholder(LENGTH, true, true);
assignWaveIndex(1, 2, w_a, 10);
assignWaveIndex(1, 2, w_b, 1, 2, w_c, 11);

playWave(1, 2, w_a);
playWave(1, 2, w_b, 1, 2, w_c);
```

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 this example. The marker integer array encodes the available markers in its least significant bits.

```
#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([  0b11*np.ones(32), np.zeros(LENGTH-32)]).astype(int)
marker_bc = np.concatenate([0b1111*np.ones(32), np.zeros(LENGTH-32)]).astype(int)

#Convert and send them to the instrument
wave_raw_a = zhinst.utils.convert_awg_waveform(wave_a, markers=marker_a)
wave_raw_bc = zhinst.utils.convert_awg_waveform(wave_b, wave_c, markers=marker_bc)
settings = [
    [path + "awg/waveform/waves/10", wave_raw_a],
    [path + "awg/waveform/waves/11", wave_raw_bc]
    ]
daq.set(settings)
```

3.6. Pulse-level Sequencing with the Command Table

Note

This tutorial is applicable to all SHFSG Instruments.

3.6.1. 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 SHFSG.

3.6.2. 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.

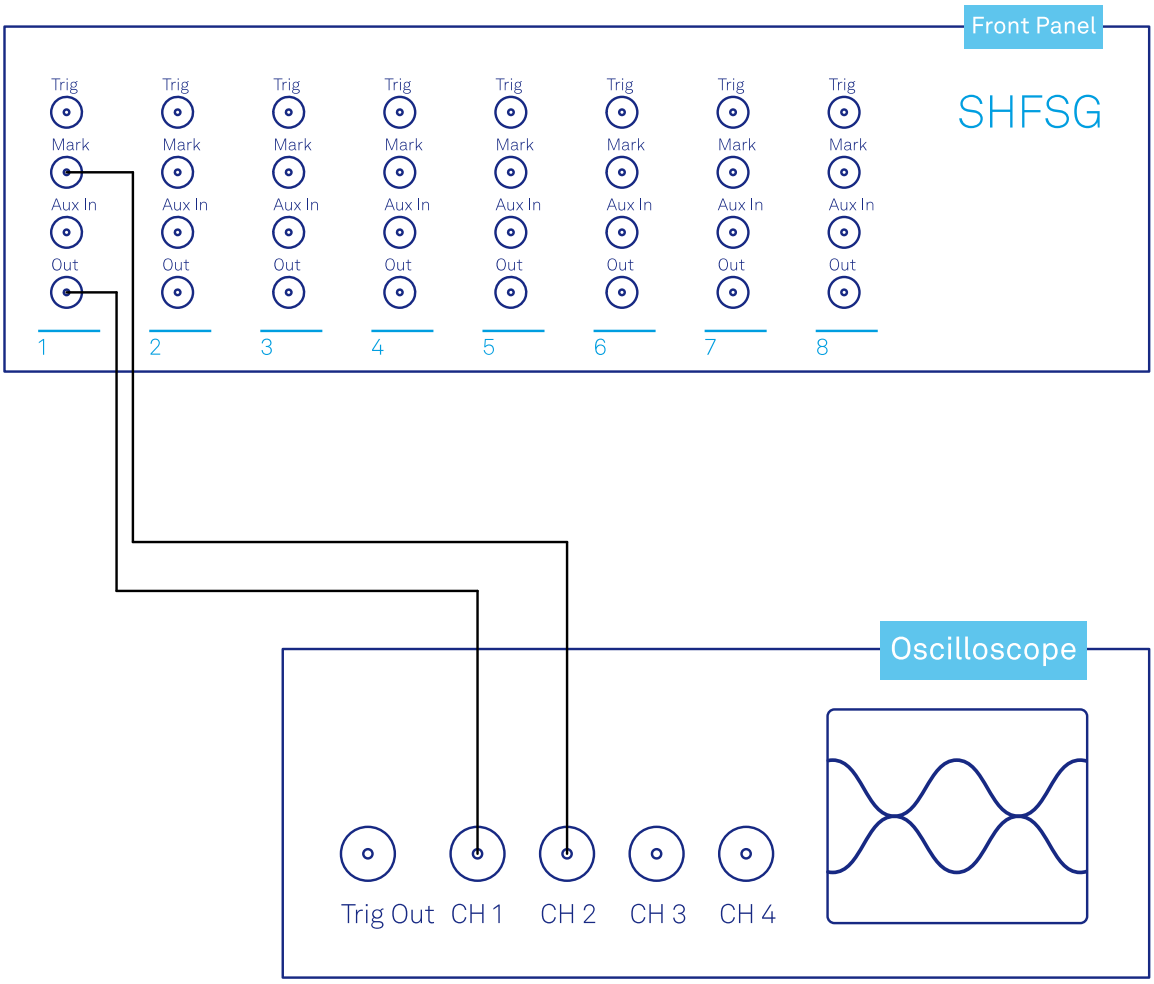


Figure 3.21. Connections for the arbitrary waveform generator command table tutorial

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.

3.6.3. Configure the Output

To begin with, we configure the output and digital modulation settings of the SHFSG, to be able to observe our signal on a scope. We use the Python API 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).

```
import zhinst.utils
import json

device_id='dev12050'

server_host = 'localhost'
server_port = 8004

## connect to data server
daq = zi.ziDAQServer(host=server_host, port=server_port, api_level=6)

## connect to device
device_interface = '1gbe'
daq.connectDevice(device_id, device_interface)

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

# Base for node path
path = f"/{device_id}/sgchannels/{chan_index}/"
#determine which synthesizer is used by the desired channel
synth = daq.getInt(path + "synthesizer")

settings = [
    # RF output settings
    [path + "output/range", 10], #output range in dBm
    [path + "output/rflfpath", 1], #use RF path, not LF path
    [f"/{device_id}/synthesizers/{synth}/centerfreq", 1.0e9], #set the
 corresponding synthesizer frequency in Hz
    [path + "output/on", 1], #enable output
    # Digital modulation settings
    [path + "awg/outputamplitude", 0.5], #set the amplitude for the AWG outputs
    [path + "oscs/0/freq", 10.0e6], #frequency of oscillator 1 in Hz
    [path + "oscs/1/freq", -150.0e6], #frequency of oscillator 2 in Hz
    [path + "awg/modulation/enable", 1], #enable digital modulation
    # Triggering settings
    [path + "marker/source", 0] #AWG trigger 1
    ]

# Set the values
daq.set(settings) #use a transactional set to update all settings
```

In this case, we will use 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.

3.6.4. 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 16384 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 `<dev>/SGCHANNELS/<n>/AWG/WAVEFORM/WAVES/<index>` Chapter 6
- 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 `<dev>/SGCHANNELS/<n>/AWG/COMMANDTABLE/DATA`

3.6.5. Basic command table use

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

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

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

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 entries using Python. It often comes in handy to define the command table as a Python dictionary, which is then converted into a JSON format at upload.

```
# index of wave table and command table entries
ct_index = 0
wave_index = 0
gain = 1.0

# command table as python dictionary
ct = [{"index": ct_index,
        "waveform": {"index": wave_index},
        "amplitude00": {
            "value": gain
        },
        "amplitude01": {
            "value": -gain
        },
        "amplitude10": {
            "value": gain
        },
        "amplitude11": {
            "value": gain
        },
        "phase": {
            "value": 0.0
        }
    },
    {"index": ct_index + 1,
        "waveform": {"index": wave_index},
        "amplitude00": {
            "value": gain/2
        },
        "amplitude01": {
            "value": -gain/2
        },
        "amplitude10": {
            "value": gain/2
        },
        "amplitude11": {
            "value": gain/2
        },
        "phase": {
            "value": 180.0
        }
    }
    ]

command_table = {"$schema": "https://docs.zhinst.com/shfsg/commandtable/v1_0/
schema",
                "header": {"version": "1.0"},
                "table": ct}
```

In this example, we generate a first command table entry with index "ct_index", which plays the dual-channel 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.

To upload the command table to the SHFSG, 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 - generate string from dictionary
daq.setVector(f"/{device_id}/SGCHANNELS/{chan_index}/AWG/COMMANDTABLE/DATA",
 json.dumps(command_table))
```

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:

```
settings = [
    [path + "awg/single", 1], #don't repeat sequence
    [path + "awg/enable",1] #enable sequencer
    ]
daq.set(settings)
```

The expected output is shown in Figure 3.22. 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.

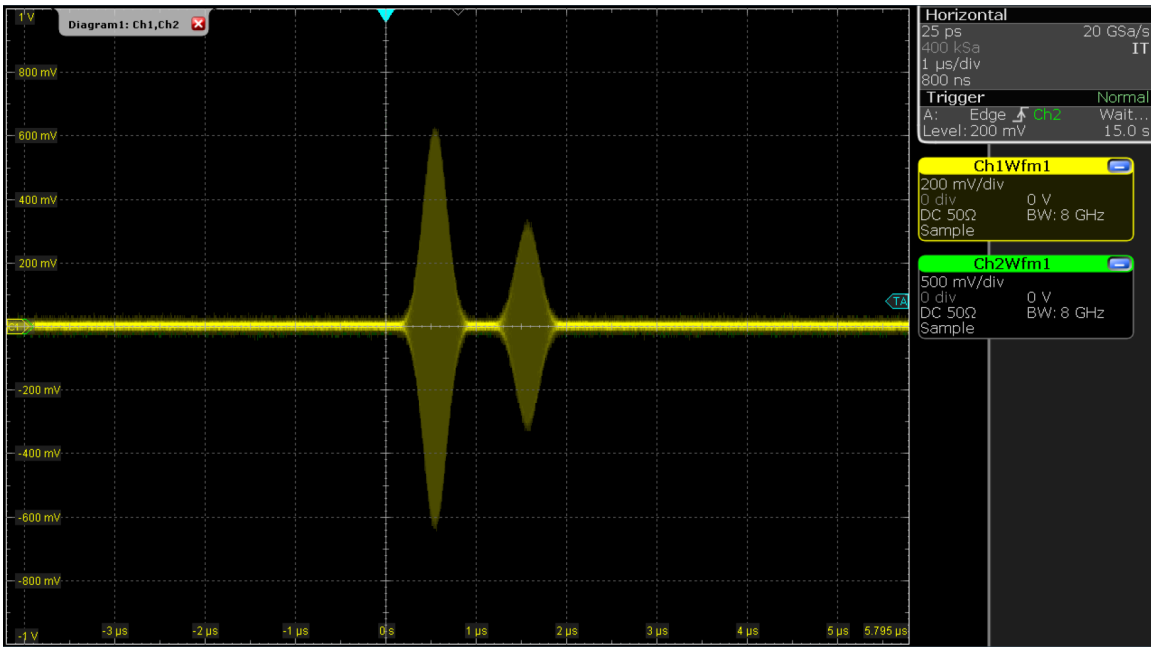


Figure 3.22. Output of the first channel from the basic command table example

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.

3.6.6. 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.

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

// Assign a dual channel waveform to wave table entry
assignWaveIndex(1,2,w_a, 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);
}
```

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.

```
# Define command table as dict
ct = [{"index": 0,
        "waveform": {"index": 0},
        "amplitude00": {
            "value": 0.0
        },
        "amplitude01": {
            "value": -0.0
        },
        "amplitude10": {
            "value": 0.0
        },
        "amplitude11": {
            "value": 0.0
        }
    },
    {"index": 1,
        "waveform": {"index": 0},
        "amplitude00": {
            "value": 0.05,
            "increment": True
        },
        "amplitude01": {
```

```
            "value": -0.05,
            "increment": True
        },
        "amplitude10": {
            "value": 0.05,
            "increment": True
        },
        "amplitude11": {
            "value": 0.05,
            "increment": True
        }
    }
    ]

command_table = {"$schema": "https://docs.zhinst.com/shfsg/commandtable/v1_0/
schema",
                 "header": {"version": "1.0"},
                 "table": ct}
```

After uploading the command table to the instrument and executing the sequencer program, the channel then produces the output shown in Figure 3.23. 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.

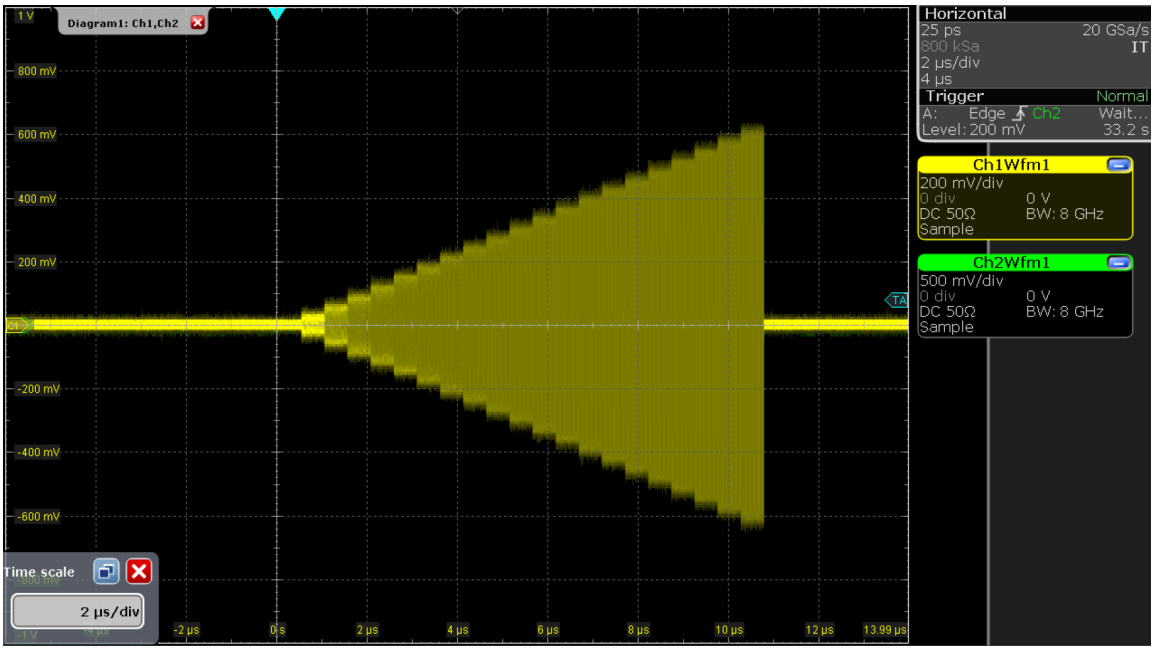


Figure 3.23. Incrementing waveform amplitudes using the command table increment functionality

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 SHFSG output channel.

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

```
# Define command table as dict
ct = [{"index": 0,
        "waveform": {"index": 0},
        "phase": {
            "value": 90.0
        }
    },
    {"index": 1,
        "waveform": {"index": 0},
        "phase": {
            "value": 0.1,
            "increment": True
        }
    }
    ]

command_table = {"$schema": "https://docs.zhinst.com/shfsg/commandtable/v1_0/
schema",
                "header": {"version": "1.0"},
                "table": ct}
```

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.

3.6.7. 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.

```
// Define two wave table entries through paceholders
assignWaveIndex(1,2,placeholder(32), 1,2,placeholder(32), 0);
assignWaveIndex(1,2,placeholder(64), 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);
```

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

# 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_1*np.exp(-x_1**2/width_1**2)],
    [amp_2*np.exp(-x_2**2/width_2**2), amp_2*np.exp(-x_2**2/width_2**2)]]

# upload waveforms to instrument
settings = []
for i, wave in enumerate(waves):
    wave_raw = zhinst.utils.convert_awg_waveform(wave[0],wave[1])
    settings.append((f'/{device_id}/sgchannels/0/awg/waveform/waves/{i}',
 wave_raw))
daq.set(settings)
```

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

```
# Define command table as dict
ct = [{"index": 0,
      "waveform": {
          "index": 0
          },
      "amplitude00": {
          "value": 1.0
          },
      "amplitude01": {
          "value": -1.0
          },
      "amplitude10": {
          "value": 1.0
          },
      "amplitude11": {
          "value": 1.0
          },
      "phase": {
          "value": 0
          },
      "oscillatorSelect": {
          "value": 0
          }
      },
      {"index": 1,
      "waveform": {
          "index": 1
          },
      "amplitude00": {
          "value": 1.0
          },
      "amplitude01": {
          "value": -1.0
          },
      "amplitude10": {
          "value": 1.0
          },
      "amplitude11": {
          "value": 1.0
          },
      "phase": {
          "value": 0
          },
      "oscillatorSelect": {
          "value": 1
          }
      },
      {"index": 2,
      "waveform": {
          "index": 0
```

```
        },
      "amplitude00": {
          "value": 0.5
          },
      "amplitude01": {
          "value": -0.5
          },
      "amplitude10": {
          "value": 0.5
          },
      "amplitude11": {
          "value": 0.5
          },
      "phase": {
          "value": 90
          },
      "oscillatorSelect": {
          "value": 0
          }
      }
    ]

command_table = {"$schema": "https://docs.zhinst.com/shfsg/commandtable/v1_0/
schema",
                 "header": {"version": "1.0"},
                 "table": ct}
```

The sequencer program can now be run and will produce output as shown in Figure 3.24.

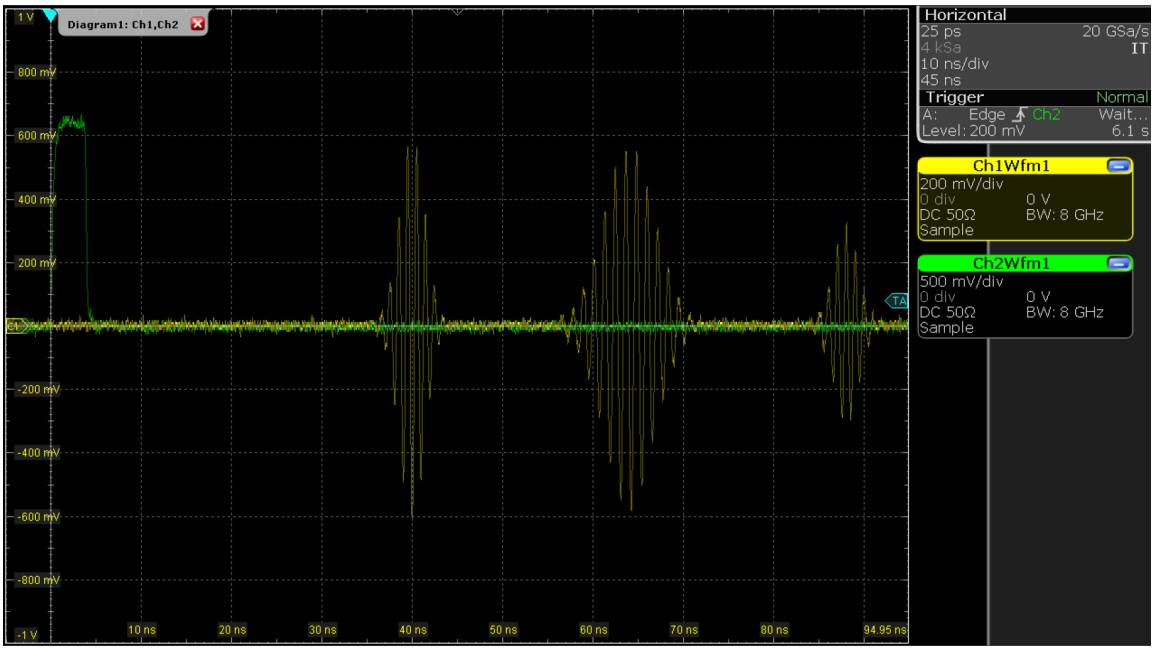


Figure 3.24. Advanced command table example output, including oscillator selection

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.

The documentation of all possible parameters in the command table JSON file is contained in the publicly hosted JSON Schema file. The URL of this schema file should be referenced from the JSON string when working on it in a JSON-enabled editor. In this way, you can easily access the documentation via auto complete and in-line help, as shown in the screenshot below for the example of Microsoft's Visual Studio Code.

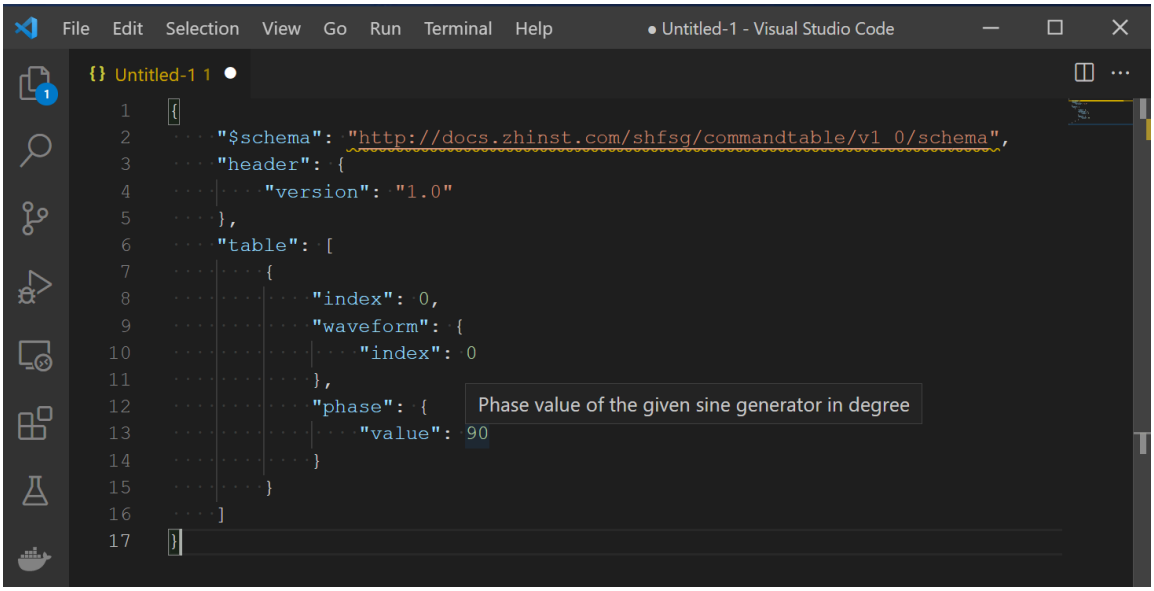


Figure 3.25. JSON editing with inline documentation in Visual Studio Code

3.7. Characterizing a Two-Qubit System

Note

This tutorial is applicable to all SHFSG Instruments. A PQSC instrument is used in this tutorial to synchronize the output channels of the SHFSG.

Note

Users can download all LabOne API Python example files from Github, https://github.com/zhinst/labone-api-examples.

3.7.1. Goals and Requirements

In this tutorial, the SHFSG is used to generate pulse sequences for characterizing a two-qubit system. We implement the pulse sequences using the Python API to measure the lifetimes of the two qubits, and to demonstrate how to perform Rabi, Ramsey and qubit spectroscopy measurements.

To visualize the generated output signals of the SHFSG, an oscilloscope with sufficient bandwidth and at least 3 channels is required.

3.7.2. Preparation

Connect the cables as illustrated below. The first step to enable the device synchronization is to connect the SHFSG to a PQSC using both a ZSync port and the reference clock of the PQSC, as shown in Figure 3.26. For this we need to enable the ZSync clock and triggers on the SHFSG.

Note

The synchronization and triggering of the SHFSG output channels can also be realized through an external trigger source connected to the trigger input connectors on the front panel of the SHFSG.

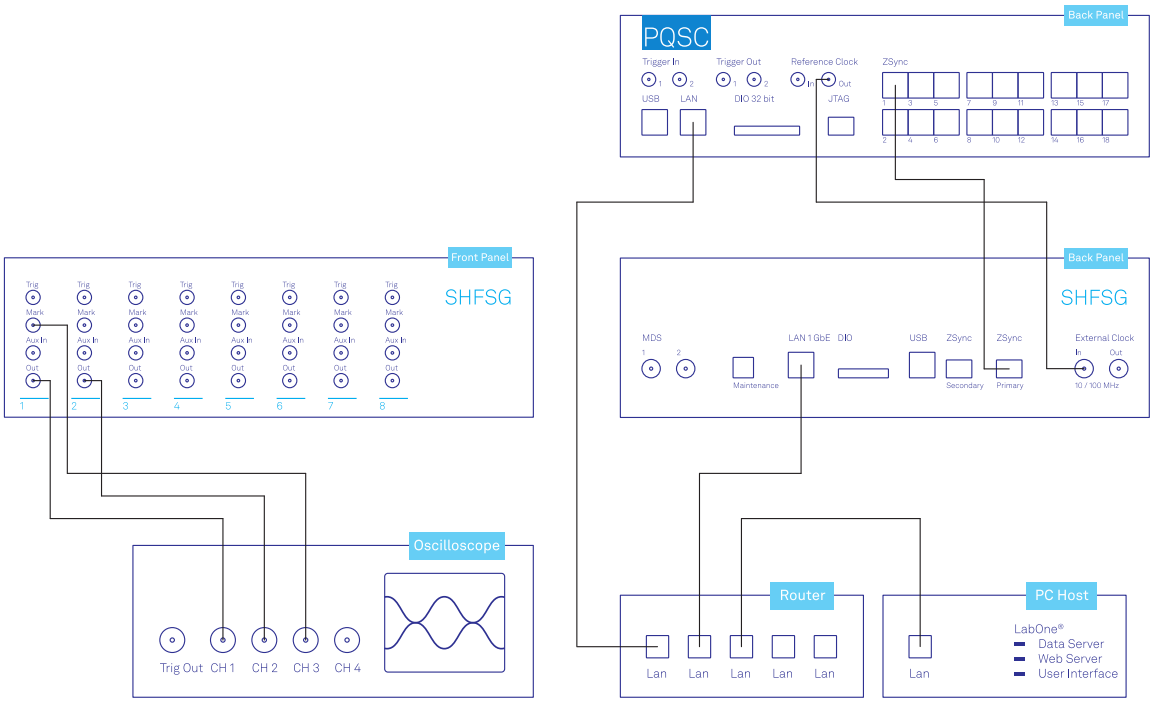


Figure 3.26. Connections for Characterizing a Two-Qubit System Tutorial.

The following tables summarizes the necessary settings for each instrument.

Table 3.19. Settings: enable the external reference clock on the PQSC

Tab	Section	Label	Setting / Value / State
Device	Configuration	Reference clock Input	Internal
Device	Configuration	Reference clock Output	Enable
Device	Configuration	Reference clock Output Frequency	100 MHz

Table 3.20. Settings: enable the ZSync and external clock on the SHFSG

Tab	Section	Label	Setting / Value / State
Device	Configuration	Input Reference Clock Set Source	External

We will also monitor the SHFSG outputs using two channels of an external scope, and use a third scope channel for visualizing the marker output. Connect the outputs of the SHFSG's outputs to the oscilloscope as follows:

The following table summarizes the settings used to configure the external scope.

Table 3.21. Settings: Configure the external scope

Scope Setting	Value / State
Ch1-3 enable	ON

Scope Setting	Value / State
Ch1-2 range	0.2 V/div
Timebase	1 μs/div
Trigger source	Ch3
Trigger level	100 mV
Run / Stop	ON

We also configure the FFT settings on the scope.

Table 3.22. Settings: Configure the external scope

Scope Setting	Value / State
Acquisition Time	10 us
FFT Center	1 GHz
FFT Span	2 GHz

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. 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 perform the two qubit characterization measurements. We advise the user to ensure that the version of the LabOne Python API, LabOne and Firmware of the SHFSG device are updated and compatible . The examples shown here use the Python API. Similar functionality is available also for C+, Matlab and .Net, among others.

3.7.3. Qubit characterization set up

This tutorial describes how to perform various characterization measurements for a two superconducting qubit system using the measurement set up shown in Figure 3.27. For this, two control lines (green) are used to generate two qubits control pulses at microwave frequencies using Outputs 1 & 2 of the SHFSG. To gain information about the state of the two qubits without directly interacting with them, dispersive readout resonators are used to probe the state of each qubit. Synchronization between the two instruments is ensured through the PQSC, which distributes its reference clock to both the SHFSG and the SHFQA through the ZSync link.

Upon a receiving the trigger signal from the PQSC, the two qubit control signals (green) are generated by the SHFSG, followed by the multiplexed readout of the two qubits (purple) which is performed by the SHFQA in the readout mode. The readout results are then fed back to the PQSC for further processing.

Note

Each qubit is dispersively coupled to a readout resonator. Upon detecting the rising edge of a trigger from the PQSC, the SHFQA determines the state of the qubits (|g> ground or |e> excited states) by detecting any changes in the amplitudes and/or phases of the multiplexed microwave readout pulses, which probe the transmissions of the two resonators at frequencies f_{Res1} and f_{Res2}.

Note

This tutorial can also be used to address other quantum systems, such as spin qubits, color centers, or neutral atoms, although some changes might be needed.

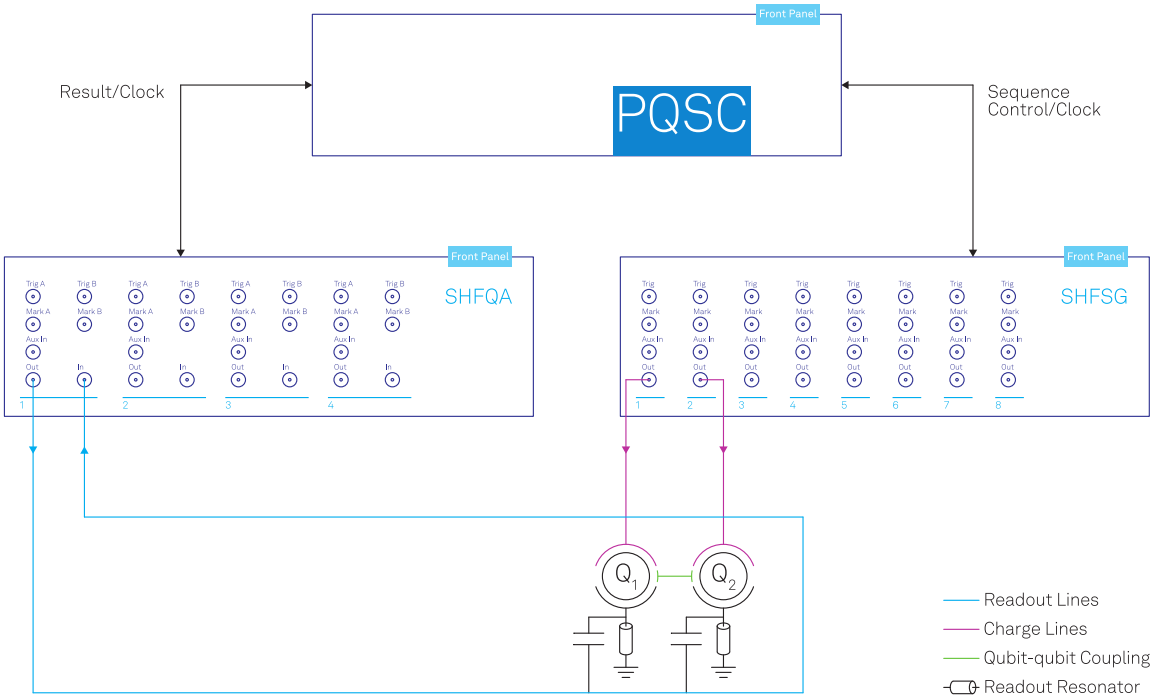


Figure 3.27. Experimental measurement set up. The elements labeled Q1 and Q2 denote the qubits 1 and 2, both of which are dispersively coupled to a resonator.

3.7.4. General Instruments configuration

In this section, we discuss the instrument configuration for the SHFSG channels using the LabOne Python API:

We connect to the instruments using the procedure describing in the Connecting to the Instrument Tutorial. To connect to the device **devXXXXX** should be replaced by the corresponding device ID (e.g. **dev12050**).

Multiple Python packages are used in this tutorial. The **numpy** package is a common mathematics package for scientific computing with Python. The **helper** package includes various helper functions for device configuration and qubit characterization. The **json** library is used to convert the command tables from a Python dictionary into a JSON formatted string. The **ziDAQServer** is used to create an API session (i.e. to connect to a Data Server).

```
import numpy as np
from helper import *
import json
from zhinst.core import ziDAQServer
```

Next we define the various output parameters of the SHFSG (e.g. center frequency, power range). The **number_of_qubits** parameter, which in this case corresponds to 2 qubits, defines

the number of channels used by the SHFSG. In this case, channel indices 0 & 1 are used, corresponding to channels 1& 2 on the front panel.

For both channels, the central frequency is set to 1 GHz, the output power range is fixed at 10 dBm, and all node settings are uploaded to the device using a transactional set.

```
# Define parameters for each SG channel
number_of_qubits = 2
sgchannel_number=[0,1]
sgchannel_center_frequency=[1e9,1e9]
sgchannel_power_out=[10,10]
sgchannel_trigger_input=1

# configure sg channels
for qubit in range(number_of_qubits):
    print(sgchannel_number[qubit],int(np.floor(sgchannel_number[qubit]/2))+1)
    exp_setting = [
        ["/%s/sgchannels/%d/output/on" % (device_id_SG, sgchannel_number[qubit]),
 1],
        ["/%s/sgchannels/%d/output/range" % (device_id_SG,
 sgchannel_number[qubit]), sgchannel_power_out[qubit]],
        ["/%s/synthesizers/%d/centerfreq" % (device_id_SG,
 sgchannel_number[qubit]), sgchannel_center_frequency[qubit]],
       ]
    daq.set(exp_setting)
# Ensure that all settings have taken effect on the device before continuing.
daq.sync()

awgModule = daq.awgModule()
awgModule.set("device", device_id_SG)
```

3.7.5. Qubit spectroscopy

The first experiment we describe is a qubit spectroscopy measurement, in which the frequencies of the control signals (green lines) for the two qubits are swept in parallel. A readout pulse (blue line) determines the qubit state for each frequency step of the control pulse.

In order to program the frequency sweep on both channels 1 & 2 of the SHFSG, we first need to define the parameters of the sweep by setting the number of frequency steps (`num_sweep_steps_qubit_spectroscopy`) and the integration time for the readout signal for each qubit. Then, we also define the minimum/maximum frequencies of the sweep to be -100/300 MHz and -100/200 MHz for channels 1 & 2, respectively. We set also the maximum drive strength to 1 for both channels.

```
# define parameters
num_sweep_steps_qubit_spectroscopy = 10000
integration_time_qubit_spectroscopy = 2e-3
# preparation
min_max_frequencies=[[-100e6,300e6],[100e6,200e6]]
max_drive_strength=[1,1]
```

In order to perform a frequency sweep on both channels 1 & 2 of the SHFSG, we need to program the sequencer of the AWG module using the sequencer command `configFreqSweep`, which allows us to configure the frequency sweep by defining the name of the oscillator `OSC0`, the starting frequency `FREQ_START` and the frequency step size `FREQ_STEP` of the sweep. Before starting to sweep the frequency, we trigger the external scope using the `setTrigger(1)` and `setTrigger(0)` commands. Then, after waiting for a trigger signal from the PQSC using the command `waitZSyncTrigger()`, the oscillator's frequency is swept using the `setSweepStep` command, which increments the oscillator's frequency by one frequency step. Both channels 1 & 2 of the SHFSG are configured and sent to the device in a single call using a transactional set: `daq.set(exp_setting)`.

```
seqc_program_sg=[ [] for _ in range(number_of_qubits) ]
```

```
for qubit in range(number_of_qubits):
    seqc_program_sg[qubit]=f""
    const OSC0 = 0;
    const FREQ_START = {min_max_frequencies[qubit][0]};
    const FREQ_STEP = {(min_max_frequencies[qubit][1]-min_max_frequencies[qubit]
[0])/num_sweep_steps_qubit_spectroscopy};

    configFreqSweep(OSC0, FREQ_START, FREQ_STEP);
    // Trigger the scope
    setTrigger(1);
    setTrigger(0);
    // Frequency sweep
    for(var i = 0; i < {num_sweep_steps_qubit_spectroscopy}; i++) {{
        waitZSyncTrigger(); //wait for PQCS trigger
        setSweepStep(OSC0, i);
    }}
    ""
    exp_setting = [
        ["/%s/sgchannels/%d/sines/0/i/sin/amplitude" % (device_id_SG,
 sgchannel_number[qubit]), 0.5*max_drive_strength[qubit]],
        ["/%s/sgchannels/%d/sines/0/i/cos/amplitude" % (device_id_SG,
 sgchannel_number[qubit]), 0.5*max_drive_strength[qubit]],
        ["/%s/sgchannels/%d/sines/0/q/sin/amplitude" % (device_id_SG,
 sgchannel_number[qubit]), 0.5*max_drive_strength[qubit]],
        ["/%s/sgchannels/%d/sines/0/q/cos/amplitude" % (device_id_SG,
 sgchannel_number[qubit]), -0.5*max_drive_strength[qubit]],
        ["/%s/sgchannels/%d/sines/0/i/enable" % (device_id_SG,
 sgchannel_number[qubit]), 1],
        ["/%s/sgchannels/%d/sines/0/q/enable" % (device_id_SG,
 sgchannel_number[qubit]), 1],
    ]
    daq.set(exp_setting)
    # Ensure that all settings have taken effect on the device before continuing.
    daq.sync()
```

Note

When using an external trigger source, the `waitZSyncTrigger()` command by the `waitDigTrigger(1)` command

The sequencer program is then uploaded AWG core of both channels of the SHFSG by using the `upload_seqc_program_sgchannel` function of the `helper` class. We then run the sequencer program.

```
# upload programs
for qubit in range(number_of_qubits):
    # Create an instance of the AWG Module
    awgModule=upload_seqc_program_sgchannel(awgModule,sgchannel_number[qubit],
 seqc_program_sg[qubit])
    awgModule.set('awg/enable',1)
```

After running the sequence, we measure the following pulse sequence with the scope set to a time base of 1 μs/div. The FFT of the scope trace shows that the frequency component of the signal sweeps in the frequency domain from 900 MHz to 1.3 GHz, which is basically the frequency sweep performed by the channel output 1 of the SHFSG.

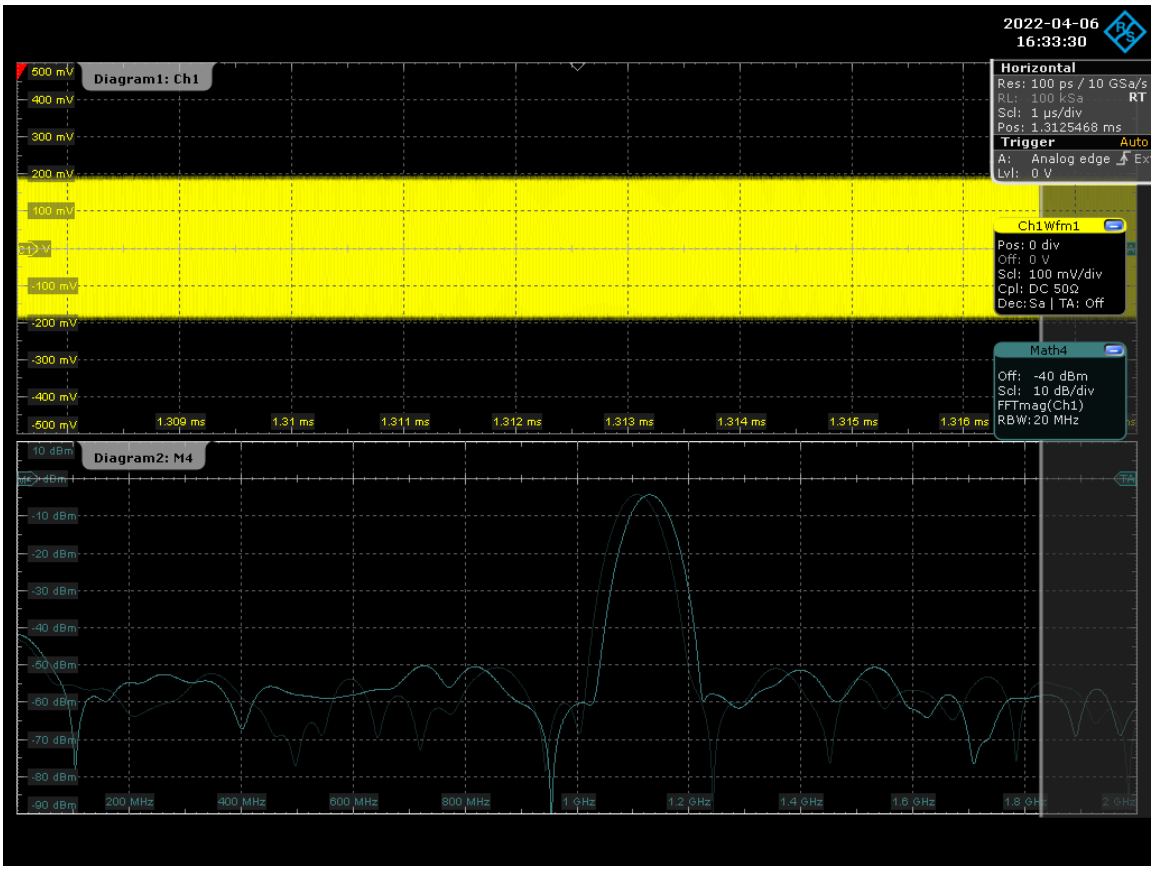


Figure 3.28. Signal generated by the AWG and captured by the scope. The top half of the figure shows a sine signal that is swept in frequency between 900 MHz to 1.3 GHz.

From this measurement, we can estimate the drive frequency needed for each of the two qubits by measuring the change in the amplitude and/or phase of the readout signal versus the drive frequency of the control signal.

3.7.6. Rabi Oscillation Measurement

The Rabi oscillation measurement is the second step to characterize our two qubits. In this case, the two qubits are driven using control pulses of fixed width and variable amplitude. In this example, we first start by defining the number of Rabi pulses, which we set to 5. We also set the number of averages for each Rabi sequence, the length of the Rabi pulses for each qubit, as well as the qubit drive frequencies.

First, we define the different parameters needed for the Rabi measurement:

```
# define parameters
num_steps_rabi_experiment = 5
num_averages_rabi_experiment = 2**5
qubit_single_gate_time=[50e-9,50e-9,50e-9,50e-9]
qubit_drive_frequency=[22e6,32e6]
sampling _rate = 2e9
single_qubit_pulse_time_seqSamples=int(np.max(qubit_single_gate_time)*SAMPLING_FREQUENCY)
single_qubit_pulse_time_samples=single_qubit_pulse_time_seqSamples*8
```

We start by defining gaussian pulses in a sequencer program, where we assign a dual-channel waveform to the wave index 0. We reset the oscillator phases to 0 using the command `resetOscPhase()`. Upon receiving the trigger signal from the PQSC, as shown in Figure 3.27, we execute the command table entry 0 (see the command table definition below) for each Rabi

measurement, which sets the starting value of 0 for all AWG output gains. Then, for each step of the Rabi sequences we execute the command table entry 1, which increments the AWG output gains and thereby increase the amplitudes of the gain of each sinusoid which is then multiplied by the AWG output, leading to an increase in the amplitude of the control pulses in both channels of the SHFSG.

Additionally, we configure both channels of the SHFSG and we finish by uploading the sequencer program to the AWG module for both SHFSG outputs channels 1 & 2.

```
for qubit in range(number_of_qubits):
    seqc_program_sg[qubit]=f""
// Define a single waveform
wave rabi_pulse=gauss({single_qubit_pulse_time_samples}, 1,
 {single_qubit_pulse_time_samples/2},
 {qubit_single_gate_time[qubit]*sampling_rate});

// Assign a dual channel waveform to wave table entry
assignWaveIndex(1,2,rabi_pulse, 1,2,rabi_pulse, 0);
resetOscPhase();
// Trigger the scope
setTrigger(1);
setTrigger(0);
repeat ({num_averages_rabi_experiment}) {{
    waitZSyncTrigger();
    executeTableEntry(0);
    repeat ({num_steps_rabi_experiment}-1) {{
        waitZSyncTrigger();
        executeTableEntry(1);
        waitWave();
        setTrigger(1);
        playZero(1024);
        waitWave();
        setTrigger(0);
        wait(5*{qubit_lifetime_samples});
         }}
}}
    ""

    # Upload command table - generate string from dictionary

    exp_setting = [
        ["/%s/sgchannels/%d/sines/0/i/enable" % (device_id_SG,
 sgchannel_number[qubit]), 0],
        ["/%s/sgchannels/%d/sines/0/q/enable" % (device_id_SG,
 sgchannel_number[qubit]), 0],
        ["/%s/sgchannels/%d/awg/modulation/enable" % (device_id_SG,
 sgchannel_number[qubit]), 1],
        ["/%s/sgchannels/%d/oscs/0/freq" % (device_id_SG,
 sgchannel_number[qubit]), qubit_drive_frequency[qubit]],
    ]
    daq.set(exp_setting)
    # Ensure that all settings have taken effect on the device before continuing.
    daq.sync()
# upload programs
for qubit in range(number_of_qubits):
    # Create an instance of the AWG Module
    awgModule=upload_seqc_program_sgchannel(awgModule,sgchannel_number[qubit],
 seqc_program_sg[qubit])
```

Here, we have define the command table with 2 entries with indices 0 and 1, both of which play the dual-channel waveform referenced in the wave table at index 0, which corresponds to the Gaussian waveform defined previously. The four amplitude settings of the command table have the same effect as the four gain settings of the Digital Modulation Tutorial. The first command table entry sets the starting amplitudes of the sweep. The second command table entry increments the current amplitude by +/- 0.5 increment_value each time the entry is called.

```
# create and upload command table
for qubit in range(number_of_qubits):
    increment_value=max_drive_strength[qubit]/num_steps_rabi_experiment

    # define command table
    # Define command table as dict
    ct = [{"index": 0,
            "waveform": {"index": 0},
            "amplitude00": {
                "value": 0
            },
            "amplitude01": {
                "value": -0
            },
            "amplitude10": {
                "value": 0
            },
            "amplitude11": {
                "value": 0
            }
            }
        },
        {"index": 1,
            "waveform": {"index": 0},
            "amplitude00": {
                "value": 0.5*increment_value,
                "increment": True
            },
            "amplitude01": {
                "value": -0.5*increment_value,
                "increment": True
            },
            "amplitude10": {
                "value": 0.5*increment_value,
                "increment": True
            },
            "amplitude11": {
                "value": 0.5*increment_value,
                "increment": True
            }
        }
        ]

    command_table = {"$schema": "https://json-schema.org/draft-04/schema",
                     "header": {"version": "1.0"},
                     "table": ct}

    daq.setVector(f"/{device_id_SG}/sgchannels/{sgchannel_number[qubit]}/awg/
commandtable/data", json.dumps(command_table))

    awgModule.set("index", sgchannel_number[qubit])
    awgModule.set('awg/enable',1)
```

After uploading the command table as a vector to the correct node of the SHFSG, we run the sequence, and start by measuring the following pulse sequences with the scope set to a time base of 700 ns/div.

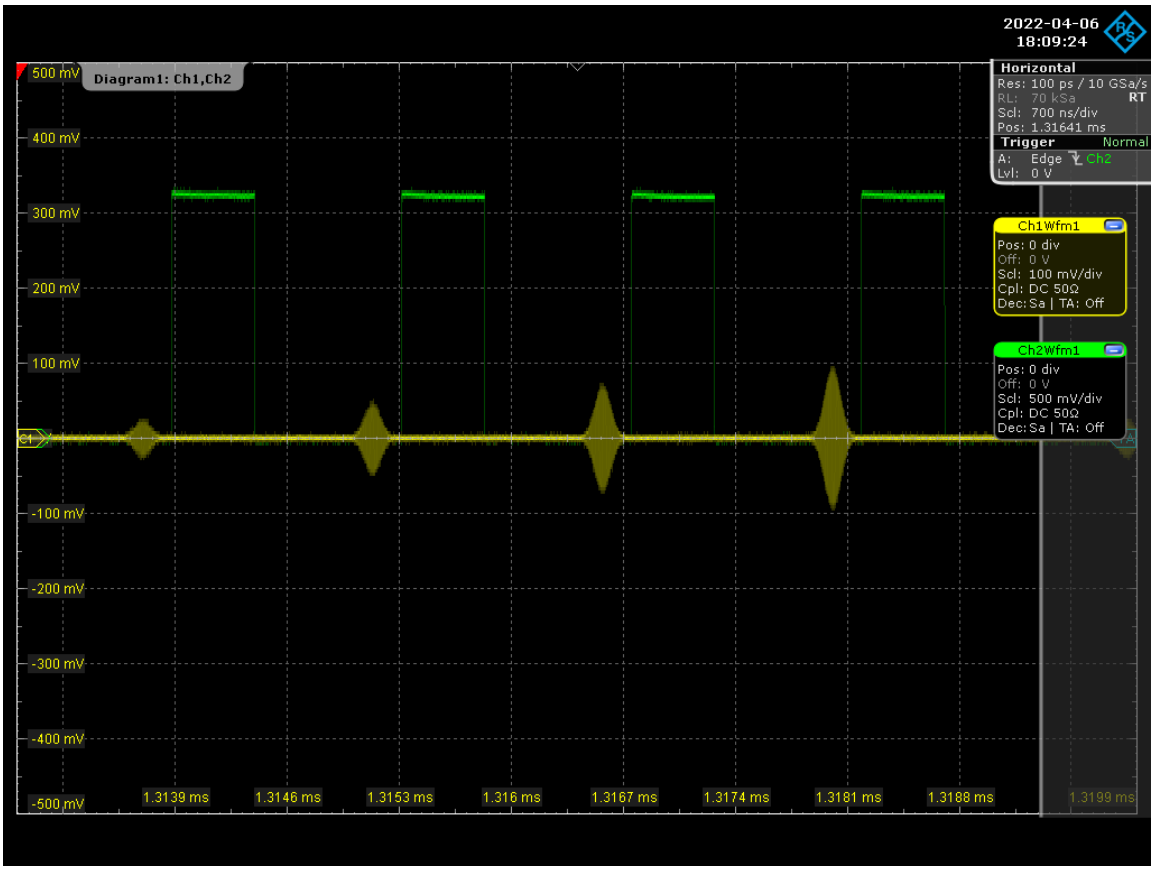


Figure 3.29. Rabi Oscillation measurement pulses generated (yellow), with a readout pulse (green) generated by the AWG.

From the results of this measurement, we can determine the control pulse amplitudes needed for rotations of π or π/2. These amplitudes are parametrized as `qubit_pi_amplitude[qubit]` and `qubit_pi/2_amplitude[qubit]` in our LabOne Python API program, respectively.

3.7.7. Ramsey Fringe Measurement

The Ramsey fringe measurement is also used to characterize our 2 qubits. The measurement starts by using a π/2 pulse to create an equal superposition of the excited and ground states. After waiting for a certain evolution time, we apply a second π/2 pulse, followed by the readout of the 2 qubits. Sweeping the evolution time between the two π/2 pulses gives us information about the coherence time of the qubit. Resonantly, which results in the exponential decay of the signal with respect to the coherence time, or we can drive the 2 qubits off-resonantly, which yields an oscillation of the signal at the detuning frequency.

This experiment starts by defining the different parameters needed for the Ramsey measurement. We set the parameter `num_steps_ramsey_experiment` to 4, and add `num_averages_ramsey_experiment = 1, ramsey_offset_frequency=1e6`:

```
# define parameters
num_steps_ramsey_experiment = 4
num_averages_ramsey_experiment = 1
ramsey_offset_frequency=1e6
max_wait_time_target=1e-6
wait_time_steps_samples=int(round(max_wait_time_target/
(num_steps_ramsey_experiment-1)*sampling_rate/16)*16)
qubit_pi_2_amplitude = [1,1]
```

Similarly to what we have seen before. We then define the waveforms and assigning them to the right index

After this, we define the variable `evolution_time_samples` which sets the time separation between two π/2 pulses in the Ramsey experiment. After receiving the trigger signal from the PQSC, the sequence executes command table entry 0, which applies π/2 pulse to the qubit. A `playZero` command follows the π/2 pulse to account for the evolution time `evolution_time_samples`. Then we play a second π/2 pulse by executing the command table entry 0 again. After applying a first Ramsey sequence, we increment the time separation by `+ wait_time_steps_samples` for 4 times and we repeat this experiment one time since `num_averages_ramsey_experiment = 1`.

Additionally, we enable the digital modulation on both channels 1&2 of the SHFSG and set the frequency of the oscillators to the sum of the qubit frequency and the offset frequency, which is given by `ramsey_offset_frequency=1e6`. After configuring the outputs of both channels, we finish by uploading the sequencer program to channels 1 & 2 of the SHFSG.

```
for qubit in range(number_of_qubits):
    seqc_program_sg[qubit]=f""
// Define a single waveform
wave rabi_pulse=gauss({single_qubit_pulse_time_samples}, 1,
 {single_qubit_pulse_time_samples/2},
 {qubit_single_gate_time[qubit]*sampling_rate);

// Assign a dual channel waveform to wave table entry
assignWaveIndex(1,2,rabi_pulse, 1,2,rabi_pulse, 0);
resetOscPhase();
var i =0;
// Trigger the scope
setTrigger(1);
setTrigger(0);
repeat ({num_averages_ramsey_experiment}) {{
    const evolution_time_samples = 0
    for (i = 0; i < {num_steps_ramsey_experiment}; i++) {{
        waitZSyncTrigger(1);
        executeTableEntry(0);
        playZero(evolution_time_samples);
        executeTableEntry(0);
        waitWave();
        setTrigger(1);
        playZero(1024);
        waitWave();
        setTrigger(0);
        wait(5*{qubit_lifetime_samples});
        evolution_time_samples = evolution_time_samples + wait_time_steps_samples;


    }}
}}
    ""

    # Upload command table - generate string from dictionary

    exp_setting = [
        ["/%s/sgchannels/%d/awg/modulation/enable" % (device_id_SG,
 sgchannel_number[qubit]), 1],
        ["/%s/sgchannels/%d/oscs/0/freq" % (device_id_SG,
 sgchannel_number[qubit]), qubit_drive_frequency[qubit]+ramsey_offset_frequency],
    ]
    daq.set(exp_setting)
    # Ensure that all settings have taken effect on the device before continuing.
    daq.sync()
# upload programs
for qubit in range(number_of_qubits):
    # Create an instance of the AWG Module
```

```
    awgModule=upload_seqc_program_sgchannel(awgModule,sgchannel_number[qubit],
 seqc_program_sg[qubit])
```

In this case, the command table contains single entry and plays the dual-channel waveform referenced in the wave table at index 0. We specify the amplitude and the phase settings. The four amplitude settings of the command table are defined so that the output signal on both the channel 1&2 of the SHFSG plays a π/2 pulse. Then, we upload the command table as a vector to the correct node to the SHFSG. After uploading the command table and the sequencer program, we run the sequence.

```
# create and upload command table
for qubit in range(number_of_qubits):

    # define command table
    # Define command table as dict
    ct = [{"index": 0,
            "waveform": {"index": 0},
            "amplitude00": {
                "value": 0.5*qubit_pi_2_amplitude[qubit]
            },
            "amplitude01": {
                "value": -0.5*qubit_pi_2_amplitude[qubit]
            },
            "amplitude10": {
                "value": 0.5*qubit_pi_2_amplitude[qubit]
            },
            "amplitude11": {
                "value": 0.5*qubit_pi_2_amplitude[qubit]
            },

        }
        ]

    command_table = {"$schema": "https://json-schema.org/draft-04/schema",
                     "header": {"version": "1.0"},
                     "table": ct}

    daq.setVector(f"/{device_id_SG}/sgchannels/{sgchannel_number[qubit]}/awg/
commandtable/data", json.dumps(command_table))

    awgModule.set("index", sgchannel_number[qubit])
    awgModule.set('awg/enable',1)
```

We show the expected result of the sequence in the Figure 3.30. As expected, we observe 5 pulse sequences of two consecutive π/2 pulses with increasing temporal separation. The `playZero` command between two consecutive Gaussian control pulses does not contribute to the memory use. Therefore, the evolution time between the π/2 pulses can be extended to several seconds without using waveform memory or increasing compilation and upload time.

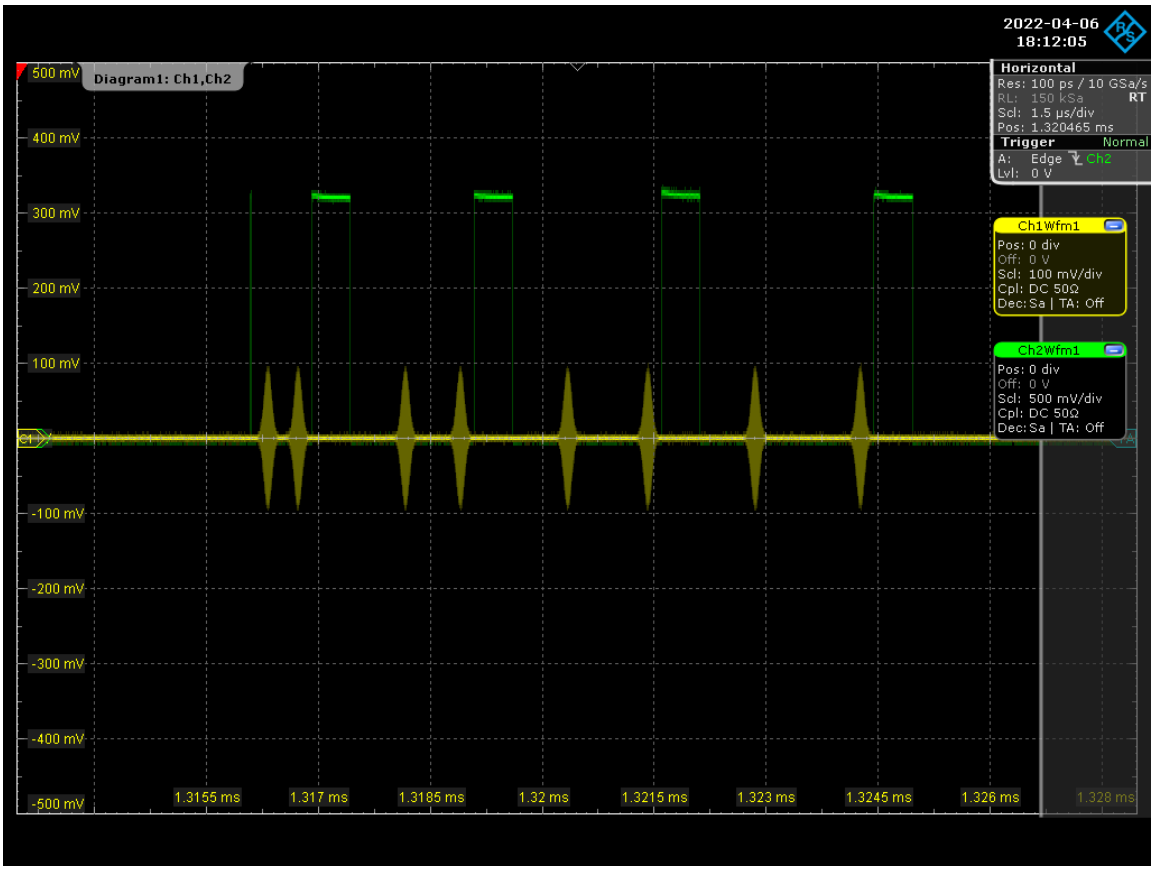


Figure 3.30. Ramsey interference experiment pulses using the AWG.

Therefore, the evolution time between the π/2 pulses can be extended to several seconds without using waveform memory or increasing compilation and upload time.

3.7.8. Qubit Lifetime Measurement

In order to measure the lifetimes of the two qubits, we perform a T1 measurement on both qubits. The measurement starts after waiting for sufficiently long time $\tau(\tau\text{Unknown character}\text{Unknown character}T_1)$, so that the 2 qubits are in the ground state. Then, we excite both qubits by applying a π pulse to each qubit and wait for a variable amount of time before reading out the qubit state. The measured signal decays exponentially with respect to the delay time between control and readout pulses, with the time constant determined by the qubit's lifetime.

We define the different parameters needed for the T1 measurement. We start by setting the parameter `num_steps_T1_experiment` to 5 and add `num_averages_T1_experiment = 2**4`:

```
# define parameters
num_steps_T1_experiment = 5
num_averages_T1_experiment = 2**4
qubit_pi_amplitude = [2,2]
```

Similarly to what we have seen before, we define a variable `time_ind`, which in this case sets the time separation between the control and readout pulses. After receiving the trigger signal from the PQSC, we execute the command table entry 0, which applies a π pulse to each qubit. This is followed by a `playZero` command, which sets the time separation between the control pulse and the readout signal.

The control channels are configured in the same way as in the Ramsey experiment. After configuring both channels of the SHFSG, we finish by uploading the sequencer program to the AWG module for both SHFSG channels 1 & 2.

```
for qubit in range(number_of_qubits):
    seqc_program_sg[qubit]=f""
// Define a single waveform
wave rabi_pulse=gauss({single_qubit_pulse_time_samples}, 1,
 {single_qubit_pulse_time_samples/2},
 {qubit_single_gate_time[qubit]*SAMPLING_FREQUENCY});

// Assign a dual channel waveform to wave table entry
assignWaveIndex(1,2,rabi_pulse, 1,2,rabi_pulse, 0);
resetOscPhase();
//Trigger the scope
setTrigger(1);
setTrigger(0);
var time_ind = 0;
repeat ({num_averages_T1_experiment}) {{
    for (time_ind = 0; time_ind < {num_steps_T1_experiment}; time_ind++) {{
        waitZSyncTrigger(1);
        executeTableEntry(0);
        playZero({wait_time_steps_samples}*time_ind);
        waitWave();
        setTrigger(1);
        playZero(1024);
        waitWave();
        setTrigger(0);
        wait(5*{qubit_lifetime_samples});

    }}
}}
    ""

    # Upload command table - generate string from dictionary

    exp_setting = [
        ["/%s/sgchannels/%d/awg/modulation/enable" % (device_id_SG,
 sgchannel_number[qubit]), 1],
        ["/%s/sgchannels/%d/oscs/0/freq" % (device_id_SG,
 sgchannel_number[qubit]), qubit_drive_frequency[qubit]+ramsey_offset_frequency],
    ]
    daq.set(exp_setting)
    # Ensure that all settings have taken effect on the device before continuing.
    daq.sync()
# upload programs
for qubit in range(number_of_qubits):
    # Create an instance of the AWG Module
    awgModule=upload_seqc_program_sgchannel(awgModule,sgchannel_number[qubit],
 seqc_program_sg[qubit])
```

Here, the command table contains a single entry with an index 0. The chosen amplitude settings allow us to play a π pulse for both qubits. We upload the command table as a vector to the correct node of the SHFSG. After uploading the command table and the sequencer program, we run the sequence.

```
# create and upload command table
for qubit in range(number_of_qubits):
    # define command table
    # Define command table as dict
    ct = [{"index": 0,
            "waveform": {"index": 0},
            "amplitude00": {
                "value": 0.5*qubit_pi_amplitude[qubit]
            },
            "amplitude01": {
```

```
                "value": -0.5*qubit_pi_amplitude[qubit]
            },
            "amplitude10": {
                "value": 0.5*qubit_pi_amplitude[qubit]
            },
            "amplitude11": {
                "value": 0.5*qubit_pi_amplitude[qubit]
            },

        }
        ]

    command_table = {"$schema": "https://json-schema.org/draft-04/schema",
                     "header": {"version": "1.0"},
                     "table": ct}

    daq.setVector(f"/{device_id_SG}/sgchannels/{sgchannel_number[qubit]}/awg/
commandtable/data", json.dumps(command_table))

    awgModule.set("index", sgchannel_number[qubit])
    awgModule.set('awg/enable',1)
```

As expected, we observe 5 pulses with the π pulse amplitude, amplitude, with an increasing delay between the pi pulse and the readout pulse, as shown in the Figure 3.31.

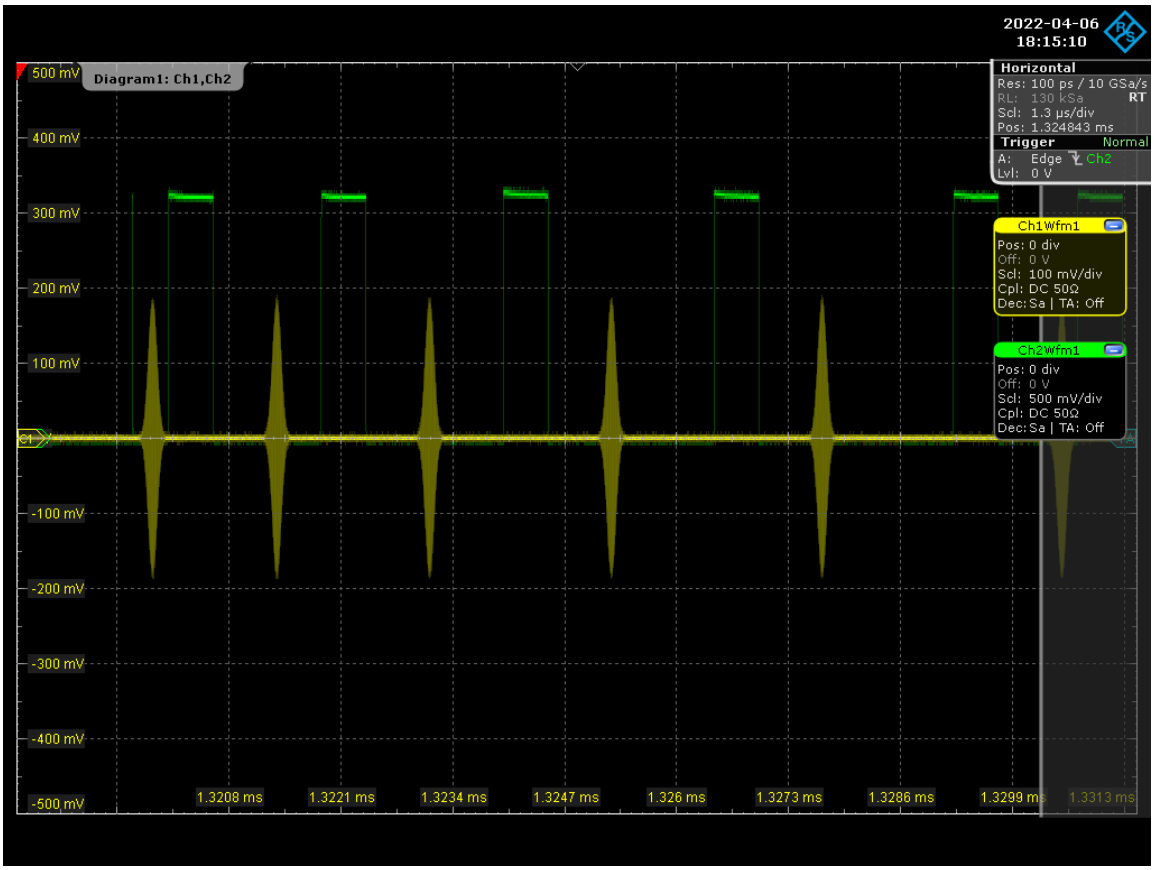


Figure 3.31. Life time measurement sequence generated by the AWG.

Chapter 4. Functional Description

This chapter gives a detailed description of the setup and measurement functionality of the Zurich Instruments SHFSG. 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.

4.1. 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 SHFSG-specific functionality of the GUI is provided in the Measurement Functionality.

Note

Some of the pictures in the following sections may not show SHFSG-specific nodes, functionality or pictures.

4.1.1. 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 SHFSG 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.

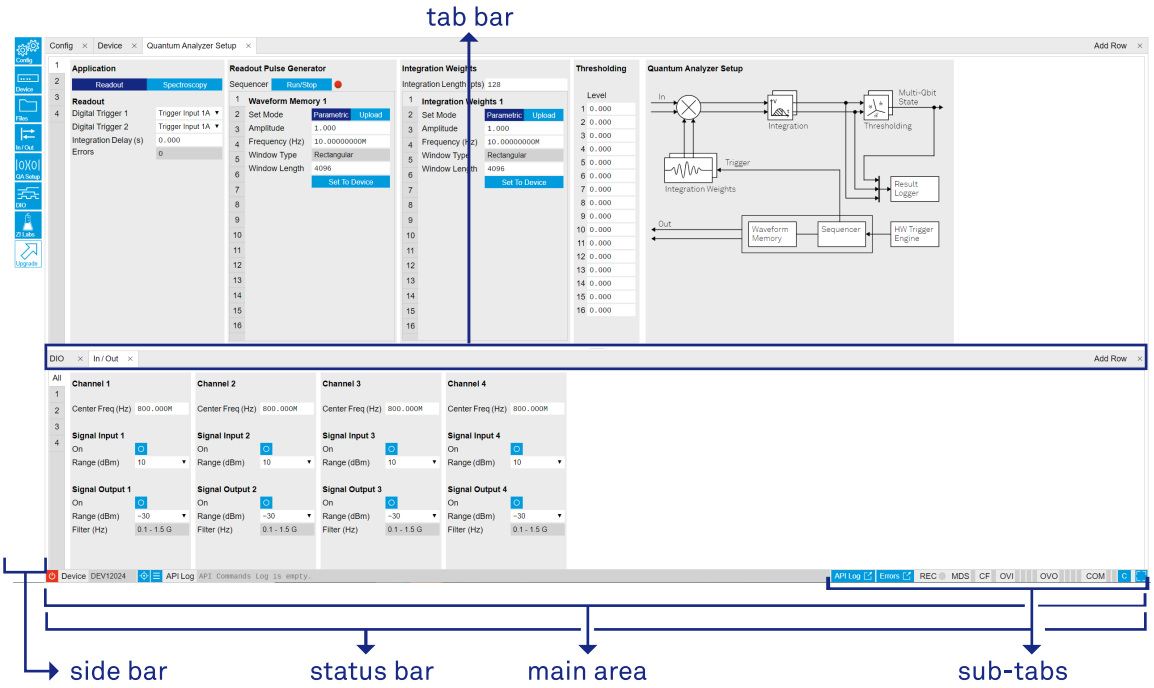


Figure 4.1. LabOne User Interface (default view)

The Figure 4.1[LabOne User Interface] automatically opens some tabs by default after a new UI session has been started. The UI is by default 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 displaced by drag-and-drop within a row or across rows.

Table 4.1 gives brief descriptions and naming conventions for the most important UI items.

Table 4.1. 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 indicators, warning lamps, 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 4.2.

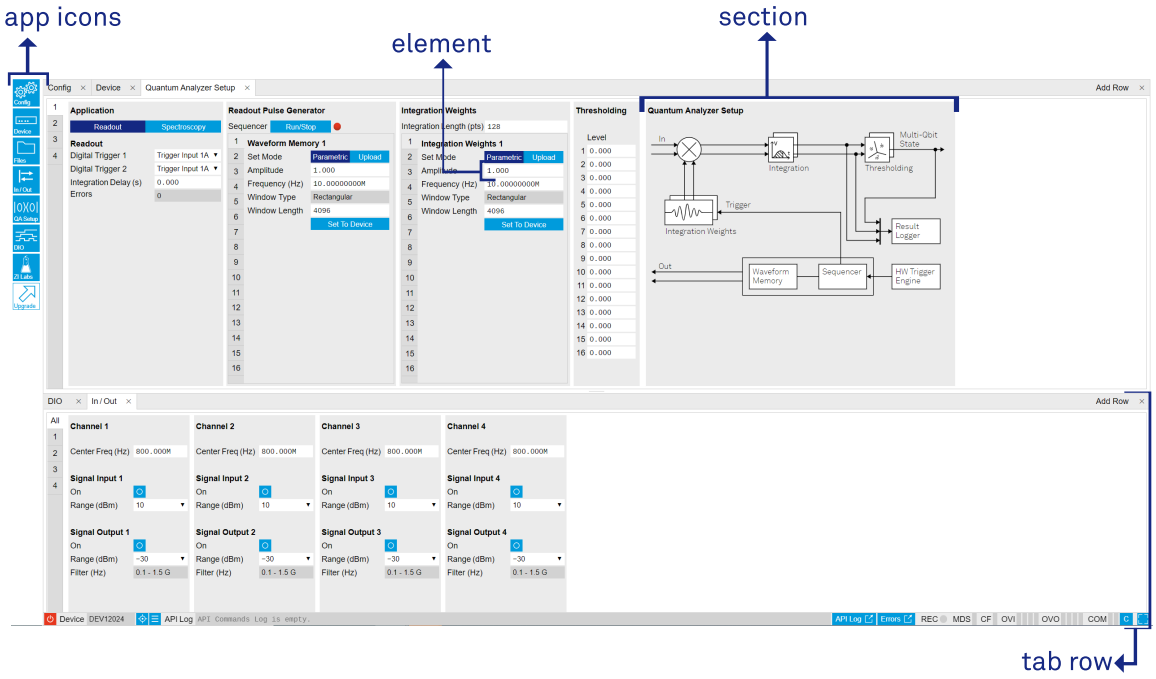


Figure 4.2. LabOne User Interface (more items)

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 4.2. 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.
DIO		Gives access to all controls relevant for the digital inputs and outputs including DIO, Trigger Inputs, and Marker Outputs.
AWG		Generate arbitrary signals using sequencing and sample-by-sample definition of waveforms.
ZI Labs		Experimental settings and controls.

Table 4.3 provides a quick overview over the different status bar elements along with a short description.

Table 4.3. Status bar description

Control/Tool	Option/Range	Description
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.
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
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).
CF	grey/yellow/red	Clock Failure - Red: present malfunction of the external 10 MHz reference oscillator. Yellow: indicates a malfunction 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.
C		Reset status flags: Clear the current state of the status flags
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 4.4. 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 4.5.

Note

Plot data can be conveniently exported to other applications such as Excel or Matlab by using LabOne's Net Link functionality, see LabOne Net Link for more information.

Table 4.5. Plot control description

Control/Tool	Option/Range	Description
Axis scaling mode		Selects between automatic, full scale and manual axis scaling.

Control/Tool	Option/Range	Description
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.
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 4.6 gives an overview of all the elements and their functionality. The chosen Signals and Operations are applied to the currently active trace only.

Note

Cursor data can be conveniently exported to other applications such as Excel or MATLAB by using LabOne's Net Link functionality, see LabOne Net Link for more information.

Table 4.6. 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.

Control/Tool	Option/Range	Description
	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.
	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

Control/Tool	Option/Range	Description
		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	Add the selected math function to the result table below.
Add All	Add All	Add all operations for the selected signal to the result table below.
Clear Selected	Clear	Clear selected lines from the result table above.
Clear All	Clear All	Clear all lines from the result table above.
Copy	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	CSV	Values of the current result table are saved as a text file into the download folder.
Net Link	Link	Provides a LabOne Net Link to use the data in tools like Excel, MATLAB, etc.
Help	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 above formula 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 signal 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.

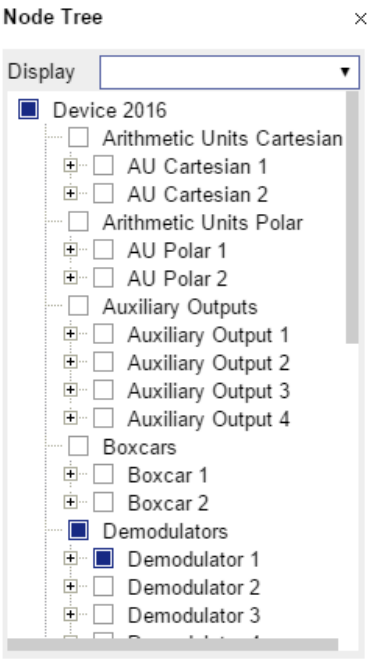


Figure 4.3. Tree selector with Display drop-down menu

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.

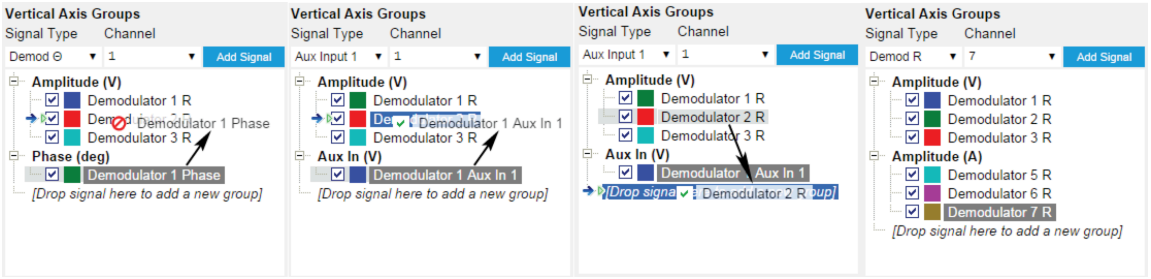


Figure 4.4. Vertical Axis Group typical drag and drop moves.

Demodulator data are only available when using a Zurich Instruments lock-in amplifier from the UHF, HF, or MF series.

Table 4.7. 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	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

Control/Tool	Option/Range	Description
		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 the section called “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.

4.1.2. Config Tab

The Config tab provides access to all major LabOne settings and is available on all SHFSG 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 4.8. App icon and short description

Control/Tool	Option/Range	Description
Config		Provides access to software configuration.

The Config tab (see Figure 4.5) is divided into five sections to control connections, sessions, settings, user interface appearance and data recording.

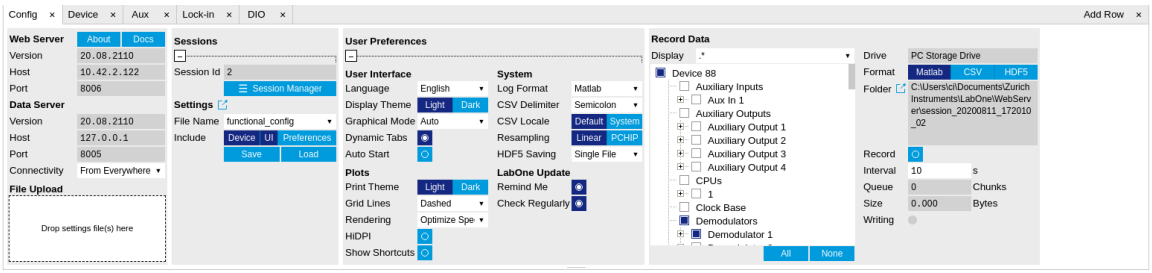


Figure 4.5. LabOne UI: Config tab

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 SHFSG instrument is used from the same computer account.

For low ambient light conditions the use of the dark display theme is recommended (see Figure 4.6).

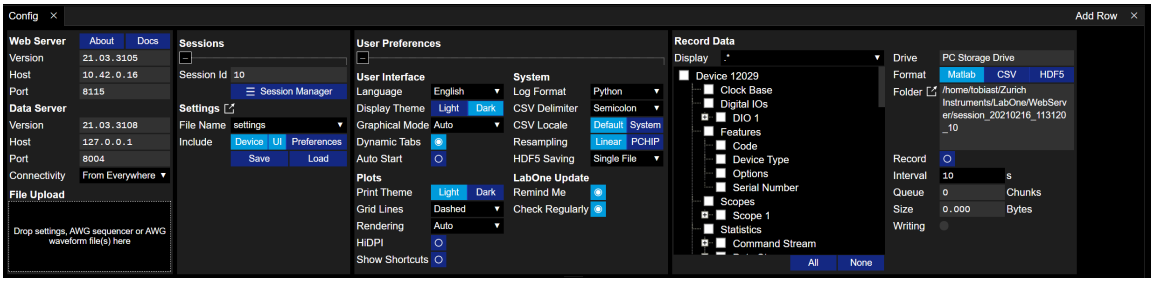


Figure 4.6. LabOne UI: Config tab - dark theme

Functional Elements

Table 4.9. 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.
Connectivity	Localhost Only	Forbid/Allow to connect to this Data Server from other computers.
	From Everywhere	
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.
Session Id	integer number	Session identifier. A session is a connection between a client and LabOne Data Server.
Session Manager	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		Enable Save/Load of particular settings.
	No Include Settings	Please enable settings type to be included during Save/Load.
	Include Device	Enable Save/Load of Device settings.
	Include UI	Enable Save/Load of User Interface settings.
	Include UI and Device	Enable Save/Load of User Interface and Device settings.
	Include Preferences	Enable loading of User Preferences from settings file.
	Include UI, Device and Preferences	Enable Save/Load of User Interface, Device and User Preferences.
Save	Save	Save the user interface and device setting to a file.
Load	Load	Load the user interface and device setting from a file.
Language		Choose the language for the tooltips.
Display Theme	Light	Choose theme of the user interface.
	Dark	
Plot Print Theme	Light	Choose theme for printing SVG plots.
	Dark	

Control/Tool	Option/Range	Description
Plot Grid	Dashed	Select active grid setting for all SVG plots.
	Solid	
	None	
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.
	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	Auto	Select the display mode for the graphical elements. Auto format will select the format which fits best the current window width.
	Expanded	
	Collapsed	
Log Format	Telnet	Choose the command log format. See status bar and [User]\Documents\Zurich Instruments \LabOne\WebServer\Log
	MATLAB	
	Python	
	.NET	

Control/Tool	Option/Range	Description
CSV Delimiter	Comma	Select which delimiter to insert for CSV files.
	Semicolon	
	Tab	
CSV Locale	Default locale. Dot for the decimal point and no digit grouping, e.g. 1005.07	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.
	System locale. Use the symbols set in the language and region settings of the computer	
HDF5 Saving	Single file. All measurements go in one 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.
	Multiple files. Each measurement goes in a separate 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	MATLAB	File format of recorded and saved data.
	CSV	
	SXM (Nanonis)	
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

Control/Tool	Option/Range	Description
		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.

4.1.3. Device Tab

The Device tab is the main settings tab for the connected instrument and is available on all SHFSG 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 4.10. 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.

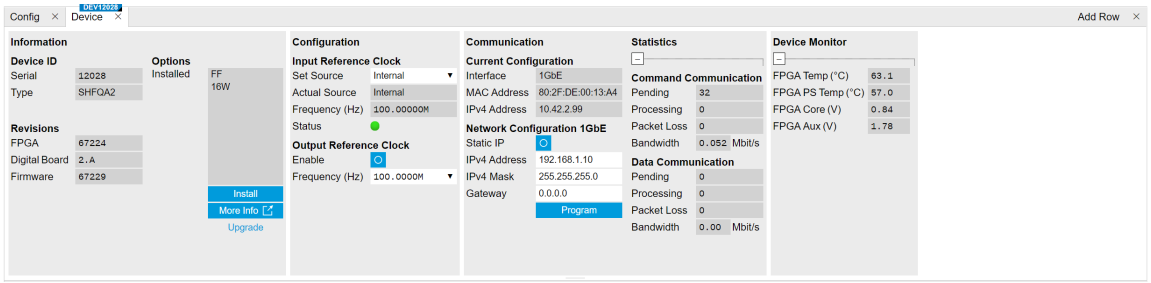


Figure 4.7. LabOne UI: Device tab

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 4.11. Device tab

Control/Tool	Option/Range	Description
Serial	1-4 digit number	Device serial number
Type	string	Device type
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.
	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.

Control/Tool	Option/Range	Description
Actual Input Reference Clock Source		Currently active clock source. This might differ from the Set Source choice if the set clock is not available.
	ZSync	ZSync clock is actually used as the frequency and time base reference.
	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.
		Indicates the frequency of the input reference clock.
		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.
Interface		Active interface between device and data server. In case multiple options are available, the priority as indicated on the left applies.

4.1.4. 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 4.12. 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 Figure 4.8) 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.

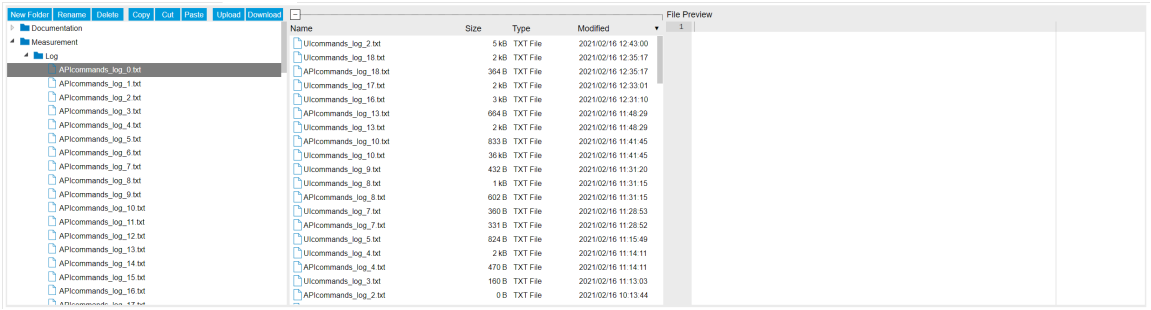


Figure 4.8. LabOne UI: File Manager tab

Functional Elements

Table 4.13. 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.
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).

4.1.5. Saving and Loading Data

Overview

In this section we discuss how to save and record measurement data with the SHFSG 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 of 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 Section 4.1.2 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.

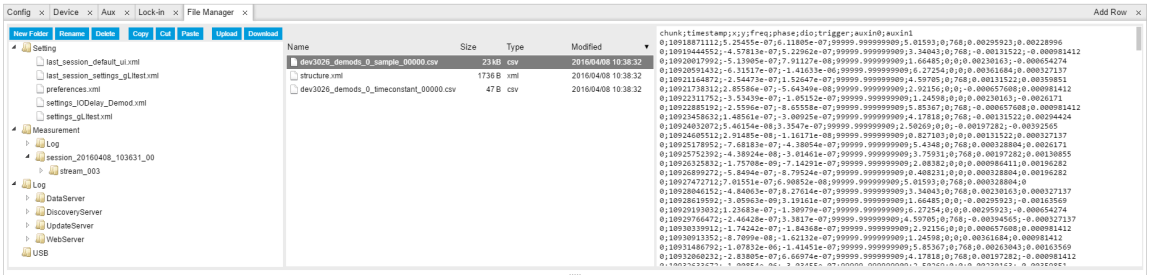


Figure 4.9. Browsing and inspecting files in the LabOne File Manager tab

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 Section 4.1.4 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 4.9 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 support feature saving, autosaving, and loading functionality. By default, the plot area in those tools displays the last 100 measurements (i.e., depending on the tool, sweep traces, scope shots, DAQ data sets, or spectra), and each measurement is represented in as a list 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 4.8 illustrates some of these features. Figure 4.10 illustrates the data loading feature.

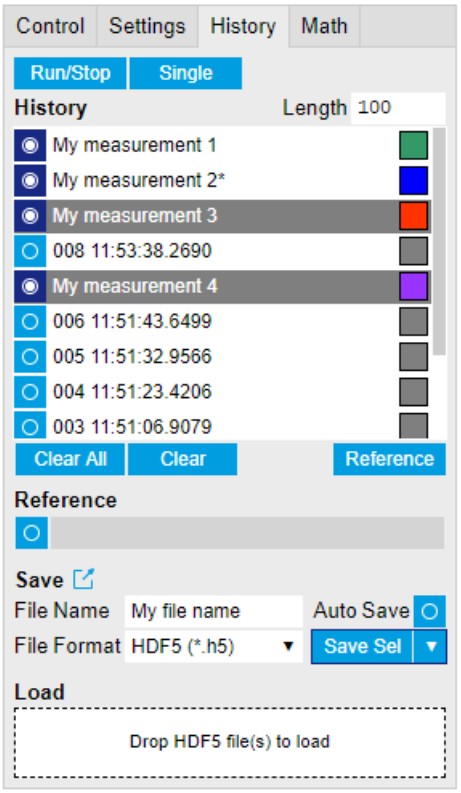


Figure 4.10. History sub-tab features. The entries "My measurement 1" etc. were renamed by the user. Measurement 1, 2, 3, 4 are currently displayed in the plot because their respective the left-hand-side button is enabled. Clicking on Save Sel would save "My measurement 3" and "My measurement 4" to a 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.

[1]Among the mentioned tools, the Scope is exceptional: it displays the most recent acquisition, and its display color is fixed. However, the Persistence feature represents a more specialized functionality for multi-trace display.

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 4.11 illustrates the data loading feature.

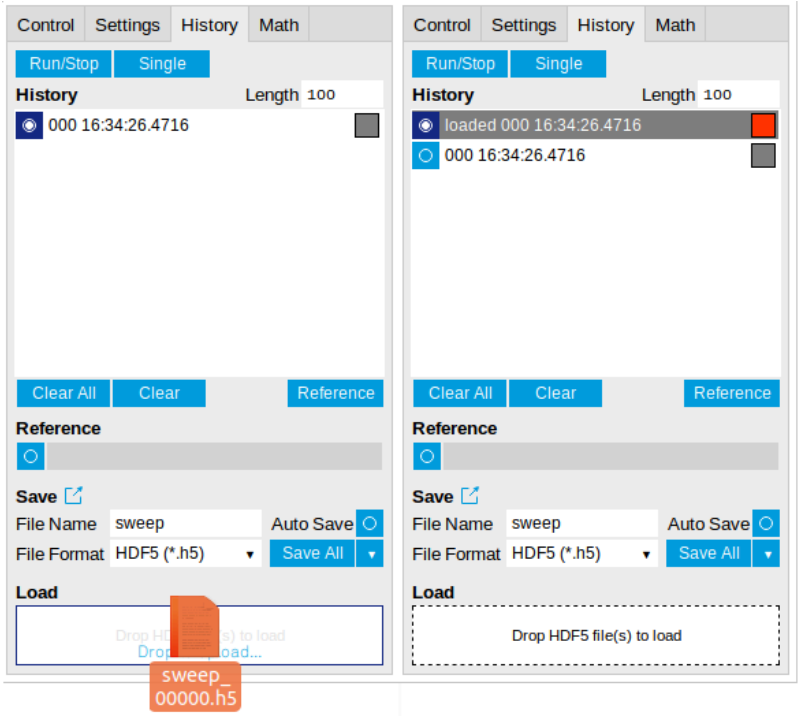


Figure 4.11. History data loading feature. Here, the file sweep_00000.h5 is loaded by drag-and-drop. The loaded data are added to the measurements in the history list.

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.

LabOne Net Link

Measurement and cursor data can be downloaded from the browser as CSV data. This allows for further processing in any application that supports CSV file formats. As the data is stored internally on the web server it can be read by direct server access from other applications. Most up-to-date software supports data import from web pages or CSV files over the internet. This allows for automatic import and refresh of data sets in many applications. To perform the import the application needs to know the address from where to load the data. This link is supplied by the LabOne User Interface. The following chapter lists examples of how to import data into some commonly used applications.

The CSV data sent to the application is a snap-shot of the data set on the web server at the time of the request. Many applications support either manual or periodic refresh functionality.

Since tabs can be instantiated several times within the same user interface, the link is specific to the tab that it is taken from. Changing the session on the LabOne User Interface or removing tabs may invalidate the link.

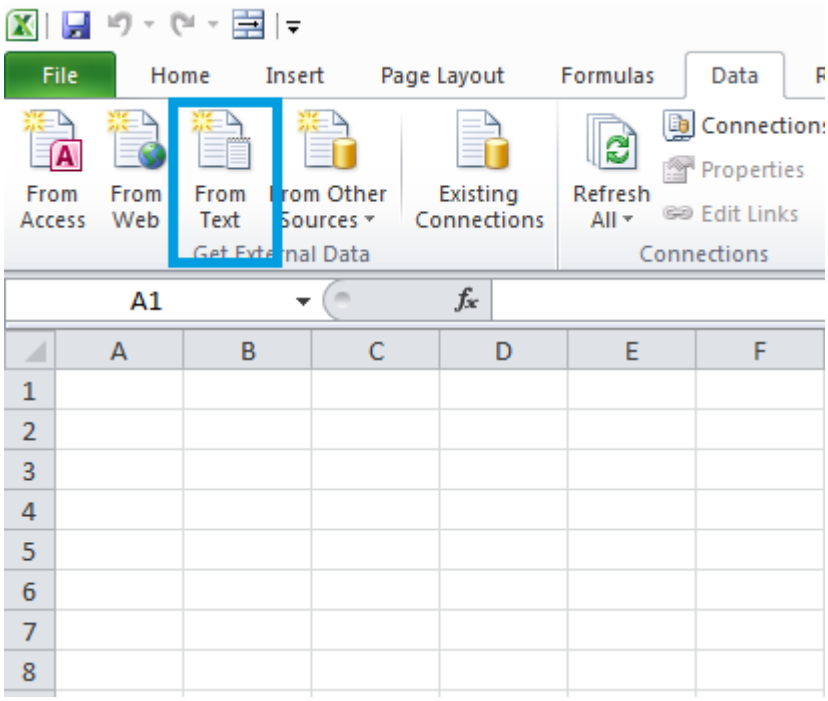
Supported applications:

- the section called “Excel”
- the section called “MATLAB”
- the section called “Python”
- the section called “C#.NET”
- the section called “Igor Pro”
- the section called “Origin”

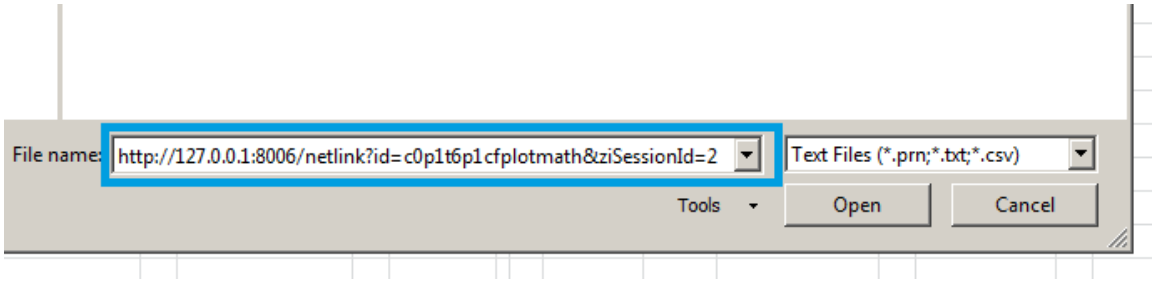
Excel

These instructions are for Excel 2010 (English). The procedure for other versions may differ.

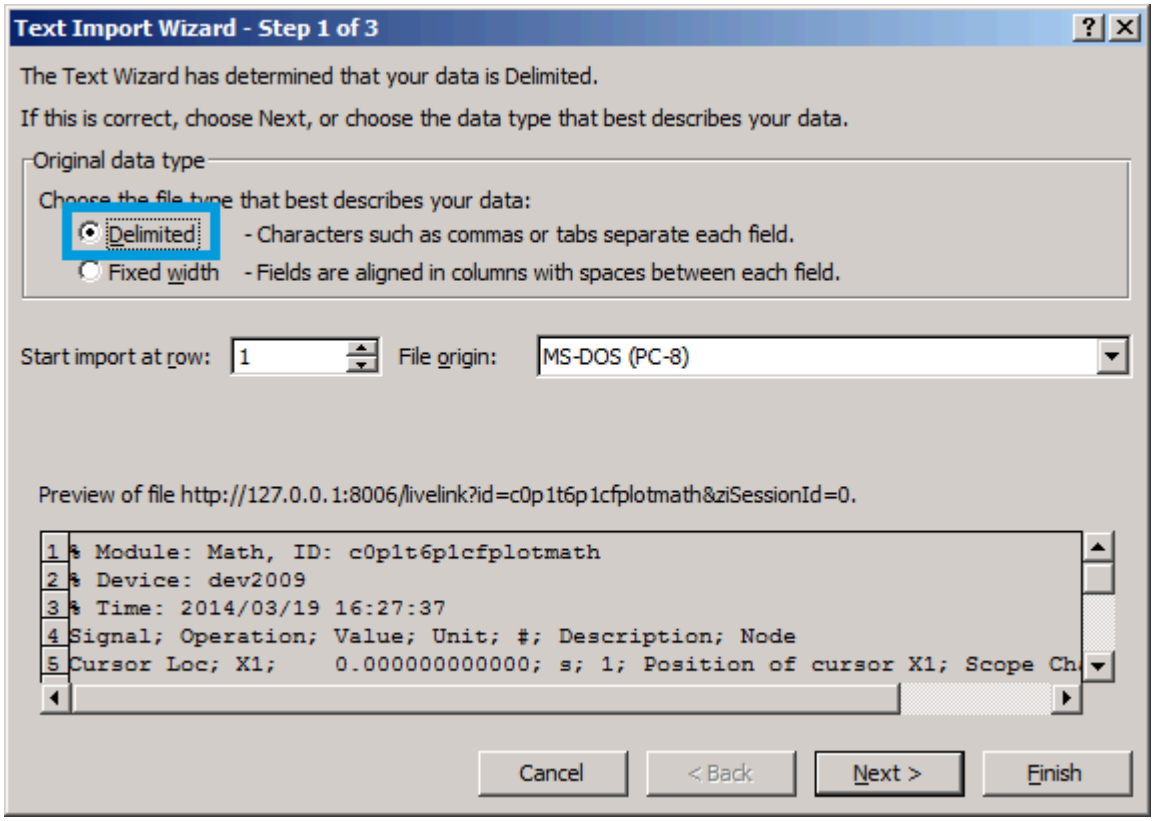
1. In Excel, click on the cell where the data is to be placed. From the Data ribbon, click the "From Text" icon. The "Import Text File" dialog will appear.



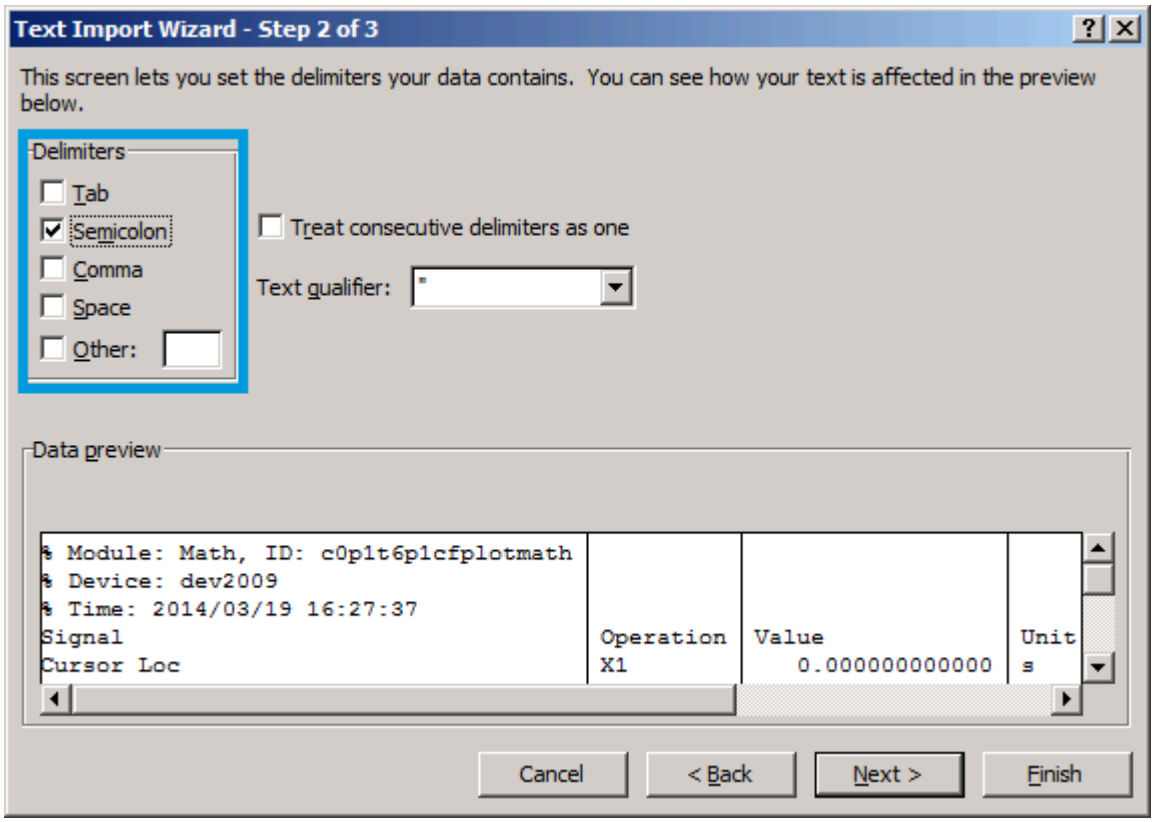
2. In LabOne, click the "Link" button of the appropriate Math tab. Copy the selected text from the "LabOne Net Link" dialog to the clipboard (either with Ctrl-C or by right clicking and selecting "Copy").



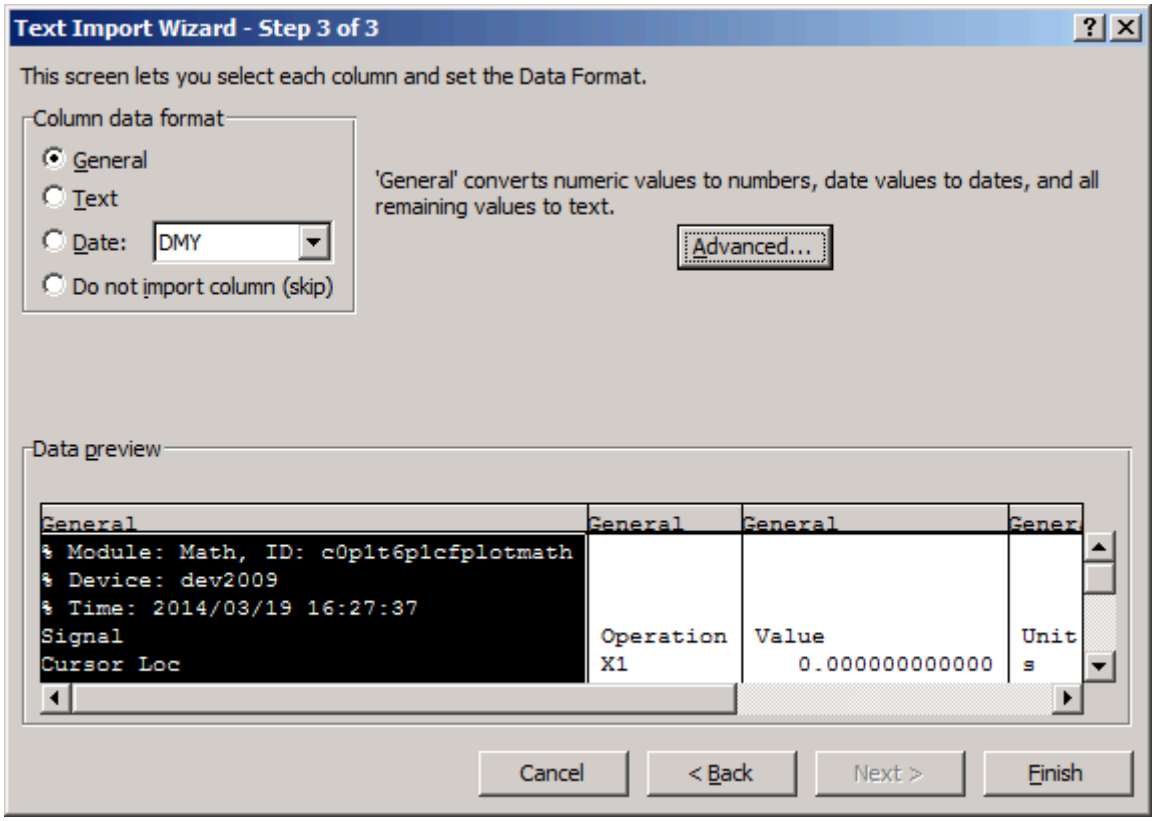
3. In Excel, paste the link into the "File name" entry field of the "Import Text File" dialog and click the "Open" button. This will start the text import wizard. Ensure that the "Delimited" button is checked before clicking the "Next" button.



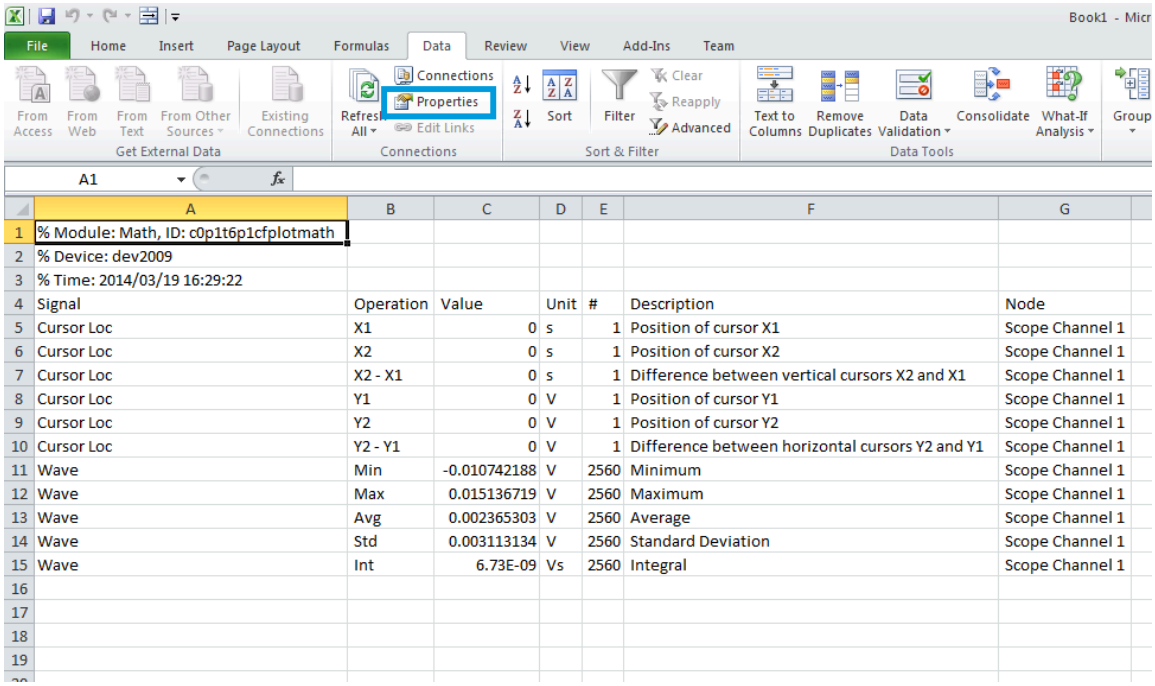
4. In the next dialog, select the delimiter character corresponding to that selected in LabOne (this can be found in the "Sessions" section of the Config tab). The default is semicolon. Click the "Next" button.



5. In the next dialog, click on "Finish" and then "OK" in the "Import Data" dialog. The data from the Math tab will now appear in the Excel sheet.

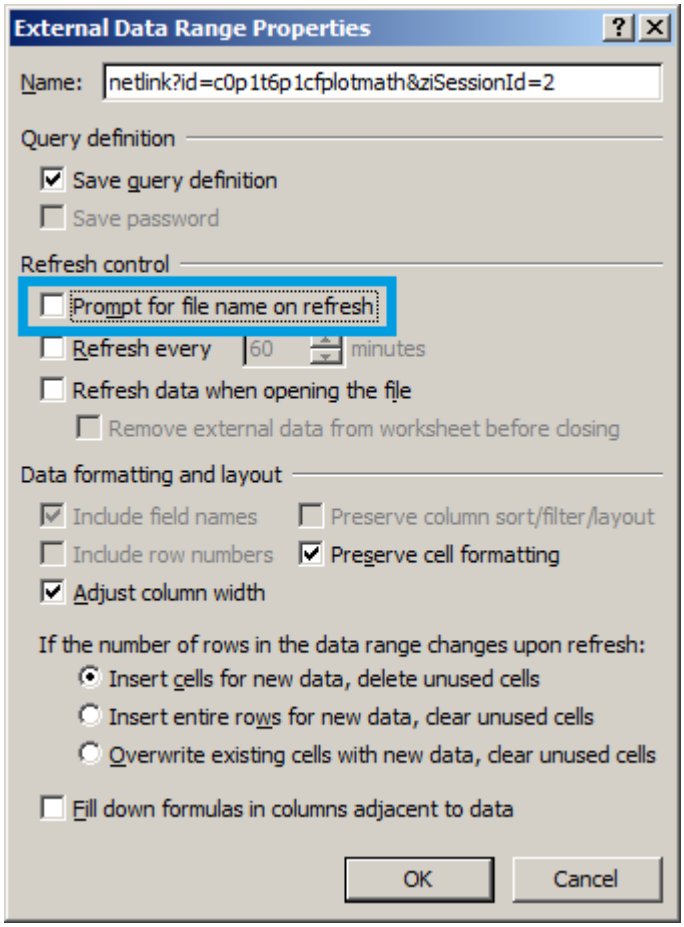


6. The data in the sheet can be updated by clicking the "Refresh All" icon. To make updating the data easier, the "Import text file" dialog can be suppressed by clicking on "Properties".



	A	B	C	D	E	F	G
1	% Module: Math, ID: c0p1t6p1cfplotmath						
2	% Device: dev2009						
3	% Time: 2014/03/19 16:29:22						
4	Signal	Operation	Value	Unit	#	Description	Node
5	Cursor Loc	X1	0	s	1	Position of cursor X1	Scope Channel 1
6	Cursor Loc	X2	0	s	1	Position of cursor X2	Scope Channel 1
7	Cursor Loc	X2 - X1	0	s	1	Difference between vertical cursors X2 and X1	Scope Channel 1
8	Cursor Loc	Y1	0	V	1	Position of cursor Y1	Scope Channel 1
9	Cursor Loc	Y2	0	V	1	Position of cursor Y2	Scope Channel 1
10	Cursor Loc	Y2 - Y1	0	V	1	Difference between horizontal cursors Y2 and Y1	Scope Channel 1
11	Wave	Min	-0.010742188	V	2560	Minimum	Scope Channel 1
12	Wave	Max	0.015136719	V	2560	Maximum	Scope Channel 1
13	Wave	Avg	0.002365303	V	2560	Average	Scope Channel 1
14	Wave	Std	0.003113134	V	2560	Standard Deviation	Scope Channel 1
15	Wave	Int	6.73E-09	Vs	2560	Integral	Scope Channel 1
16							
17							
18							
19							

7. Deactivate the check box "Prompt for file name on refresh".



MATLAB

By copying the link text from the "LabOne Net Link" dialog to the clipboard, the following code snippet can be used in MATLAB to read the data.

```
textscan(urlread(clipboard('paste')),'%s%s%f%s%d%s%s','Headerlines',
 4,'Delimiter', ';')
```

Python

The following code snippet can be used in Python 2 to read the LabOne Net Link data, where "url" is assigned to the text copied from the "LabOne Net Link" dialog.

```
import csv
import urllib2
url = "http://127.0.0.1:8006/netlink?id=c0p5t6p1cfplotmath&ziSessionId=0"
webpage = urllib2.urlopen(url)
datareader = csv.reader(webpage)
data = []
for row in datareader:
    data.append(row)
```

C#.NET

The .NET Framework offers a WebClient object which can be used to send web requests to the LabOne WebServer and download LabOne Net Link data. The string with comma separated content can be parsed by splitting the data at comma borders.

```
using System;
```

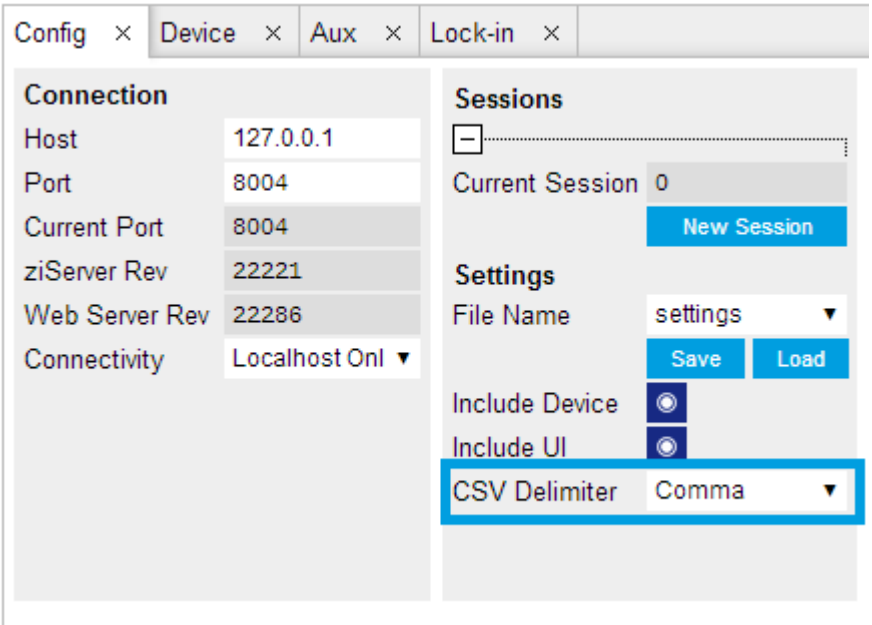
```
using System.Text;
using System.Net;

namespace ExampleCSV
{
  class Program
  {
    static void Main(string[] args)
    {
      try
      {
        WebClient wc = new WebClient();
        byte[] buffer = wc.DownloadData("http://127.0.0.1:8006/netlink?
id=c0p1t6p1cfplotmath&ziSessionId=0");
        String doc = Encoding.ASCII.GetString(buffer);
        // Parse here CSV lines and extract data
        // ...
        Console.WriteLine(doc);
      } catch (Exception e) {
        Console.WriteLine("Caught exception: " + e.Message);
      }
    }
  }
 }
```

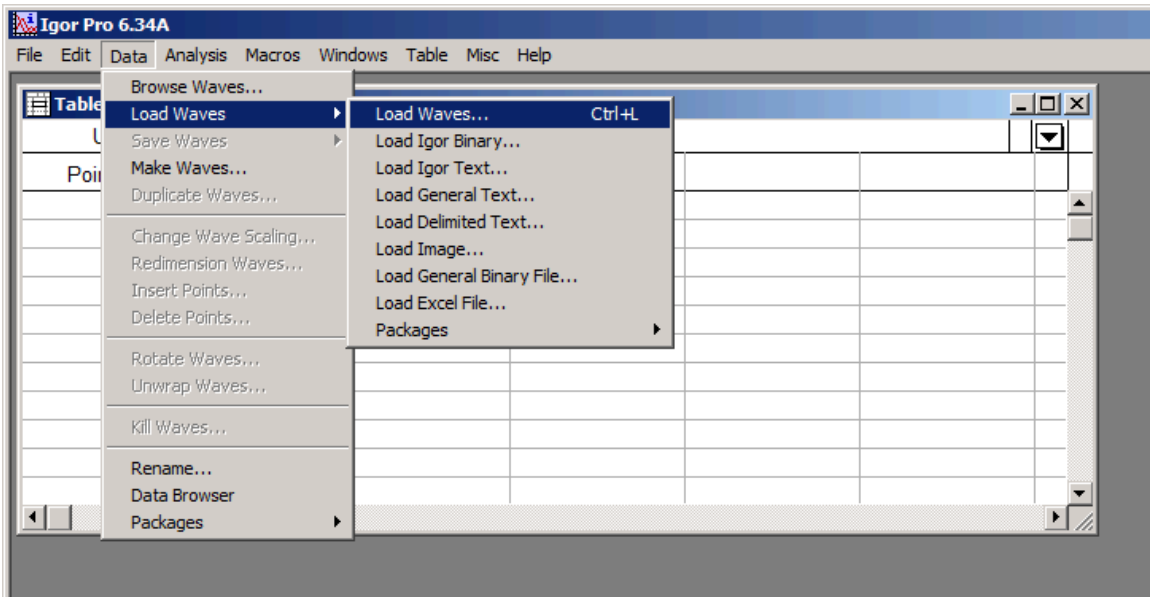
Igor Pro

These instructions are for Igor Pro 6.34A English. The procedure for other versions may differ.

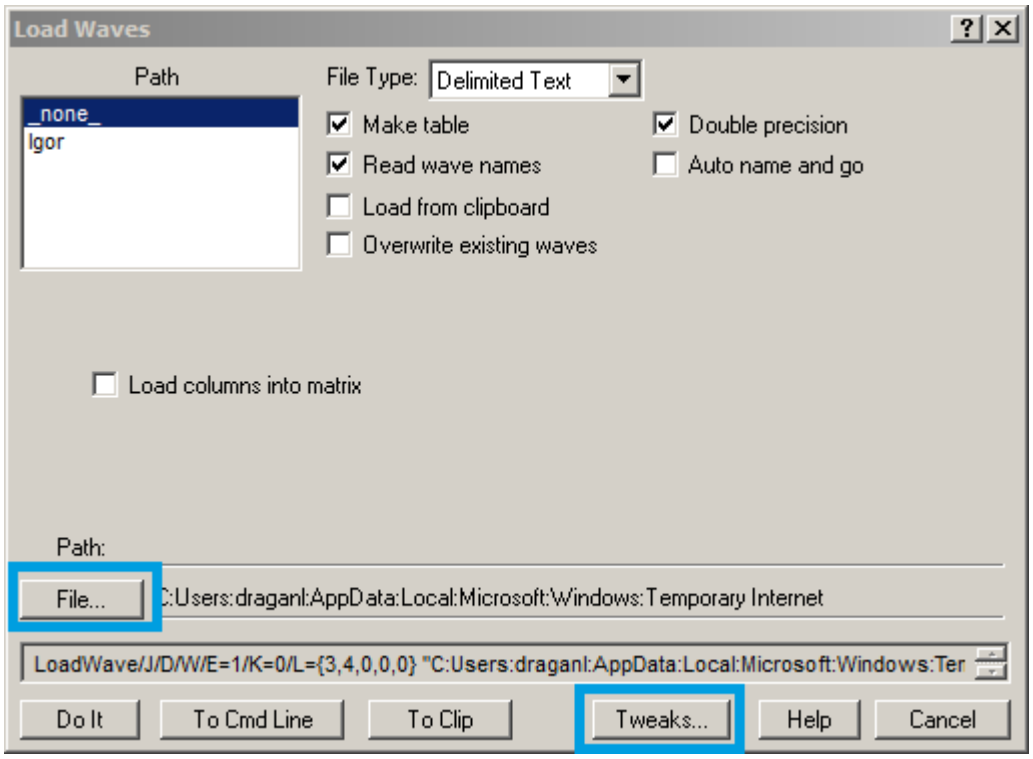
1. For Igor Pro, the CSV separator has to be the comma. Set this in the LabOne Config tab as follows:



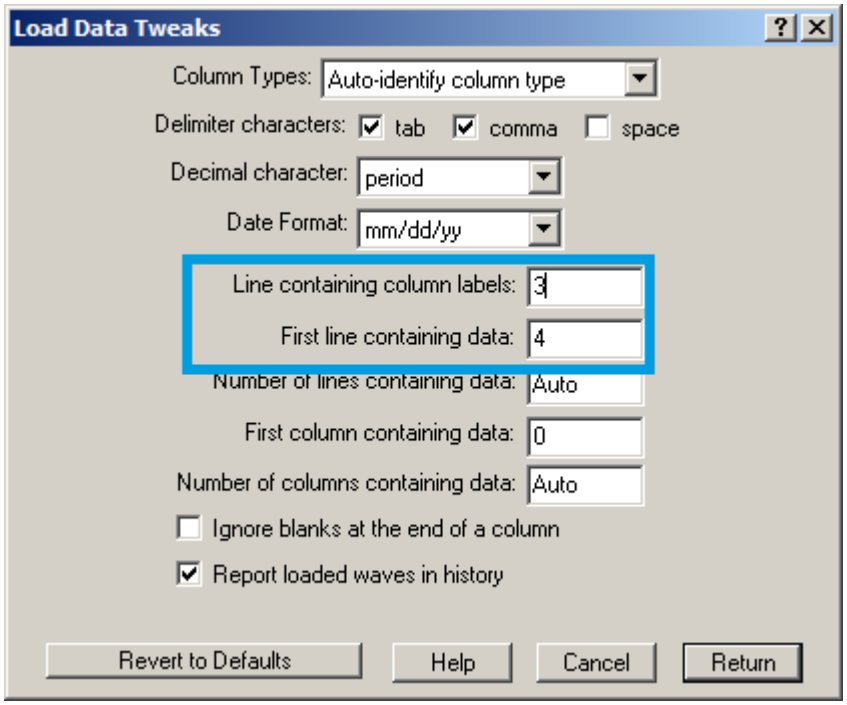
2. In Igor Pro, select the menu "Data→Load Waves→Load Waves...".



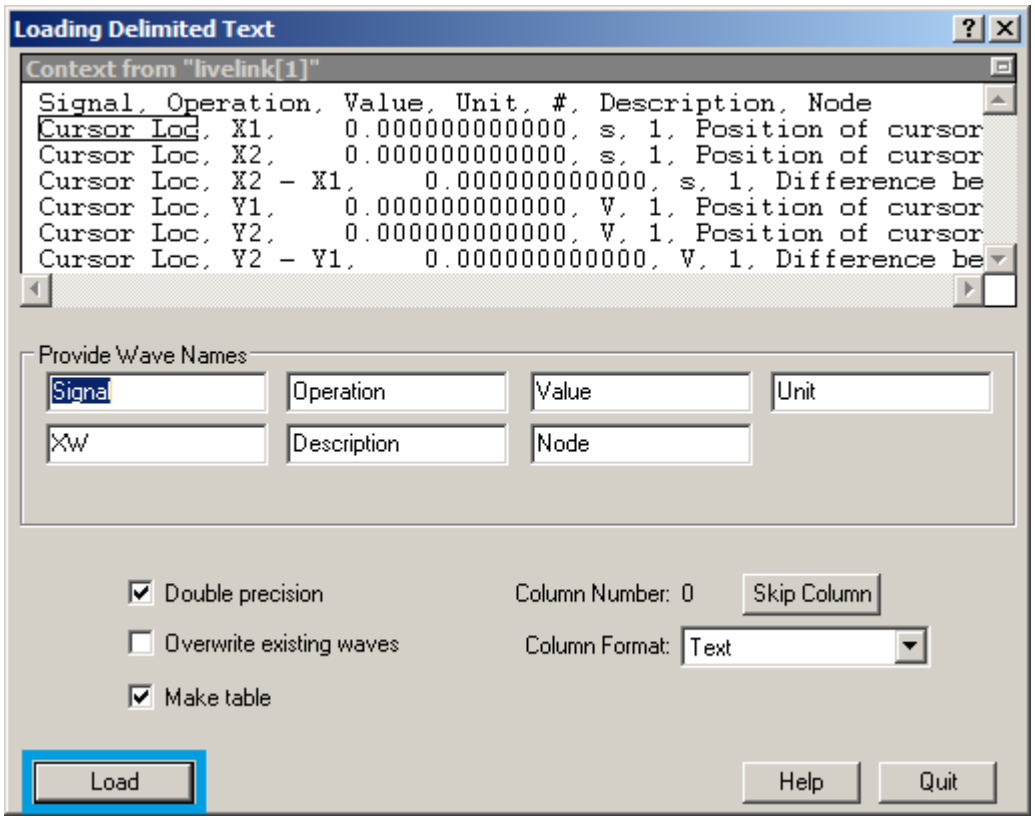
3. In the "Load Waves" dialog, click the "File..." button and paste the link text from the "LabOne Net Link" dialog into the entry field. Then click the "Tweaks..." button to open the "Load Data Tweaks" dialog.



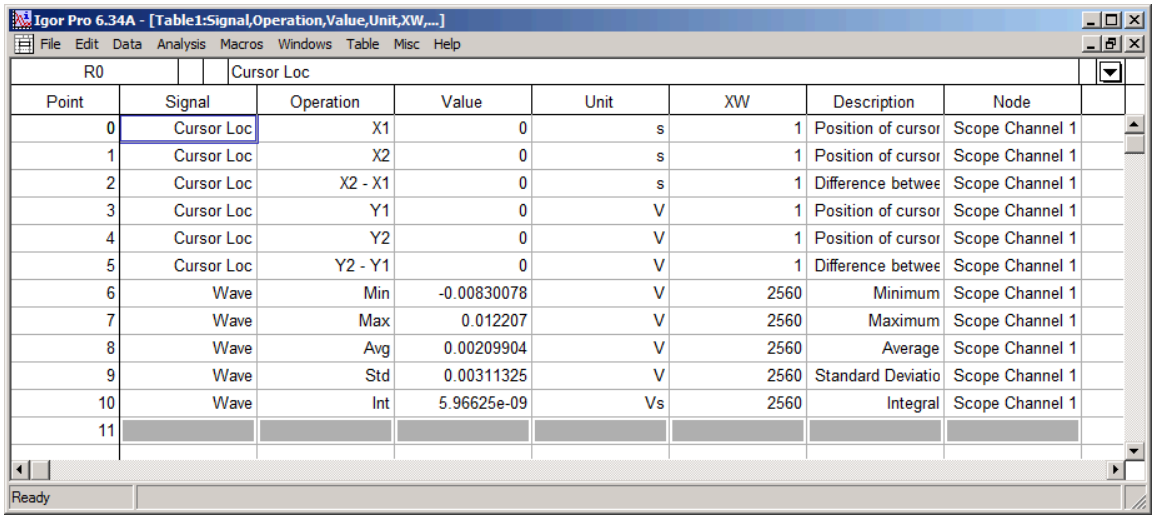
4. Adjust values as highlighted below and click "Return". The "Loading Delimited Data" dialog will appear.



5. Click the "Load" button to read the data.



6. The data will appear in the Igor Pro main window.

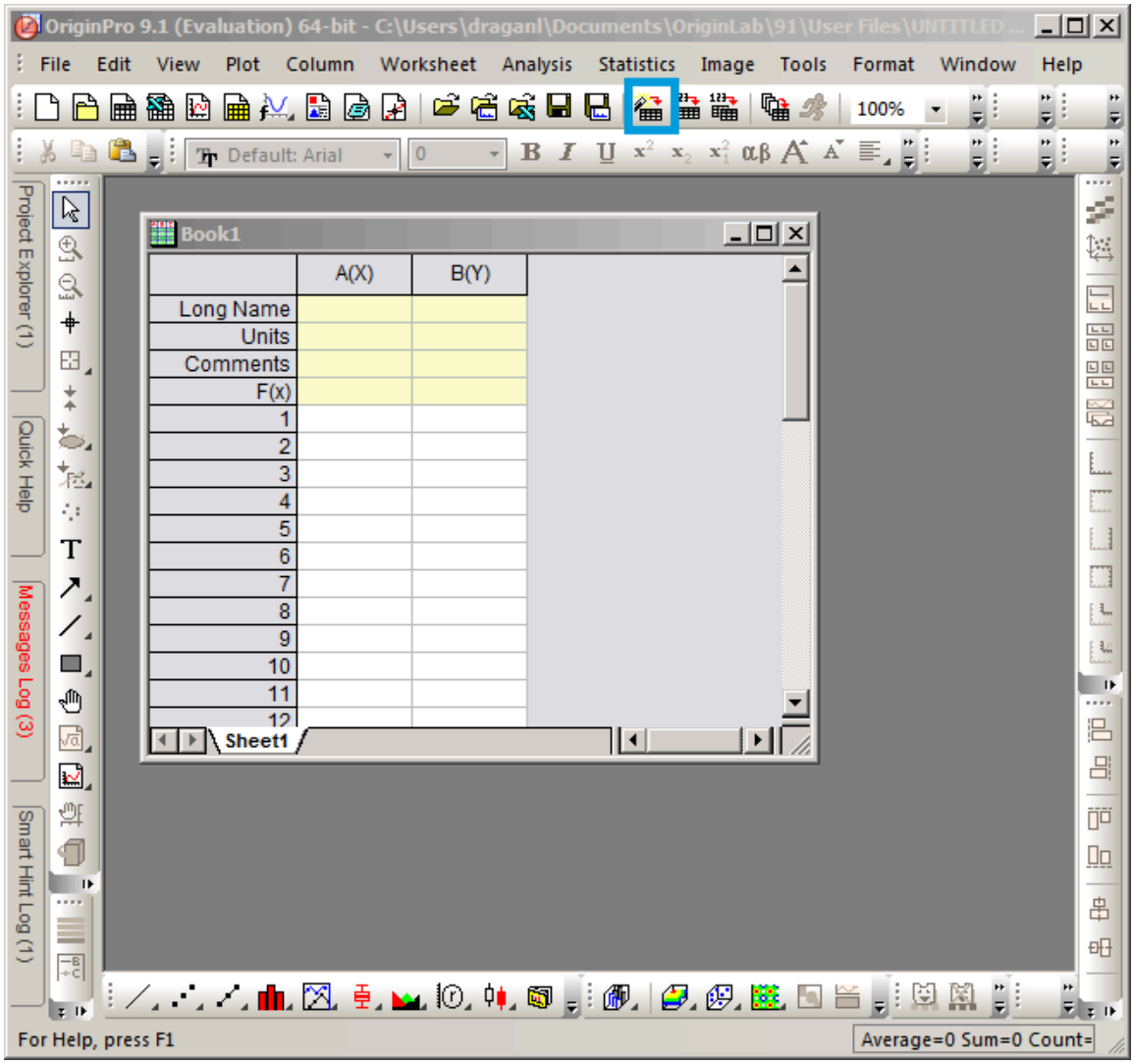


Point	Signal	Operation	Value	Unit	XW	Description	Node
0	Cursor Loc	X1	0	s	1	Position of cursor	Scope Channel 1
1	Cursor Loc	X2	0	s	1	Position of cursor	Scope Channel 1
2	Cursor Loc	X2 - X1	0	s	1	Difference betwee	Scope Channel 1
3	Cursor Loc	Y1	0	V	1	Position of cursor	Scope Channel 1
4	Cursor Loc	Y2	0	V	1	Position of cursor	Scope Channel 1
5	Cursor Loc	Y2 - Y1	0	V	1	Difference betwee	Scope Channel 1
6	Wave	Min	-0.00830078	V	2560	Minimum	Scope Channel 1
7	Wave	Max	0.012207	V	2560	Maximum	Scope Channel 1
8	Wave	Avg	0.00209904	V	2560	Average	Scope Channel 1
9	Wave	Std	0.00311325	V	2560	Standard Deviatio	Scope Channel 1
10	Wave	Int	5.96625e-09	Vs	2560	Integral	Scope Channel 1
11							

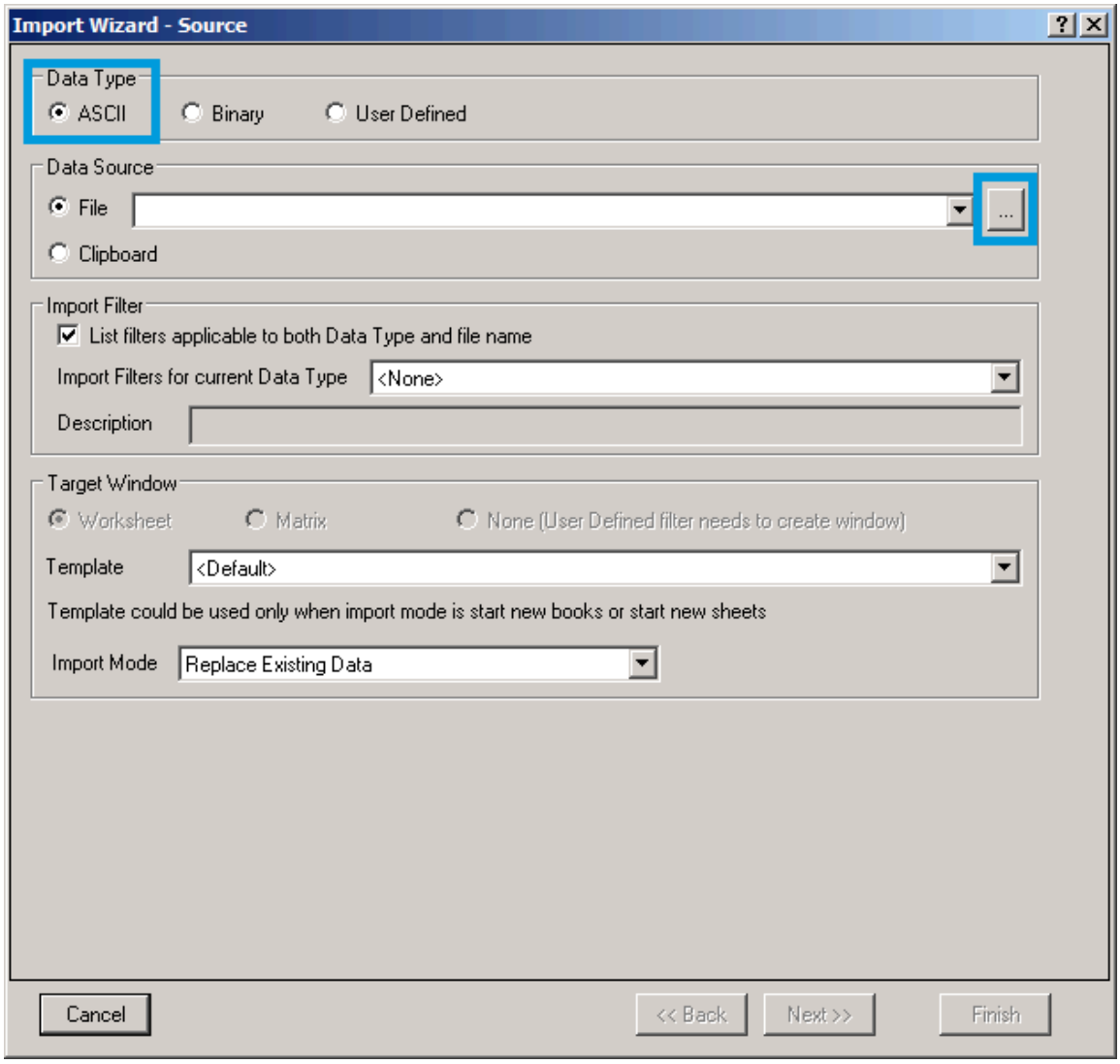
Origin

These instructions are for Origin 9.1 English. The procedure for other versions may differ.

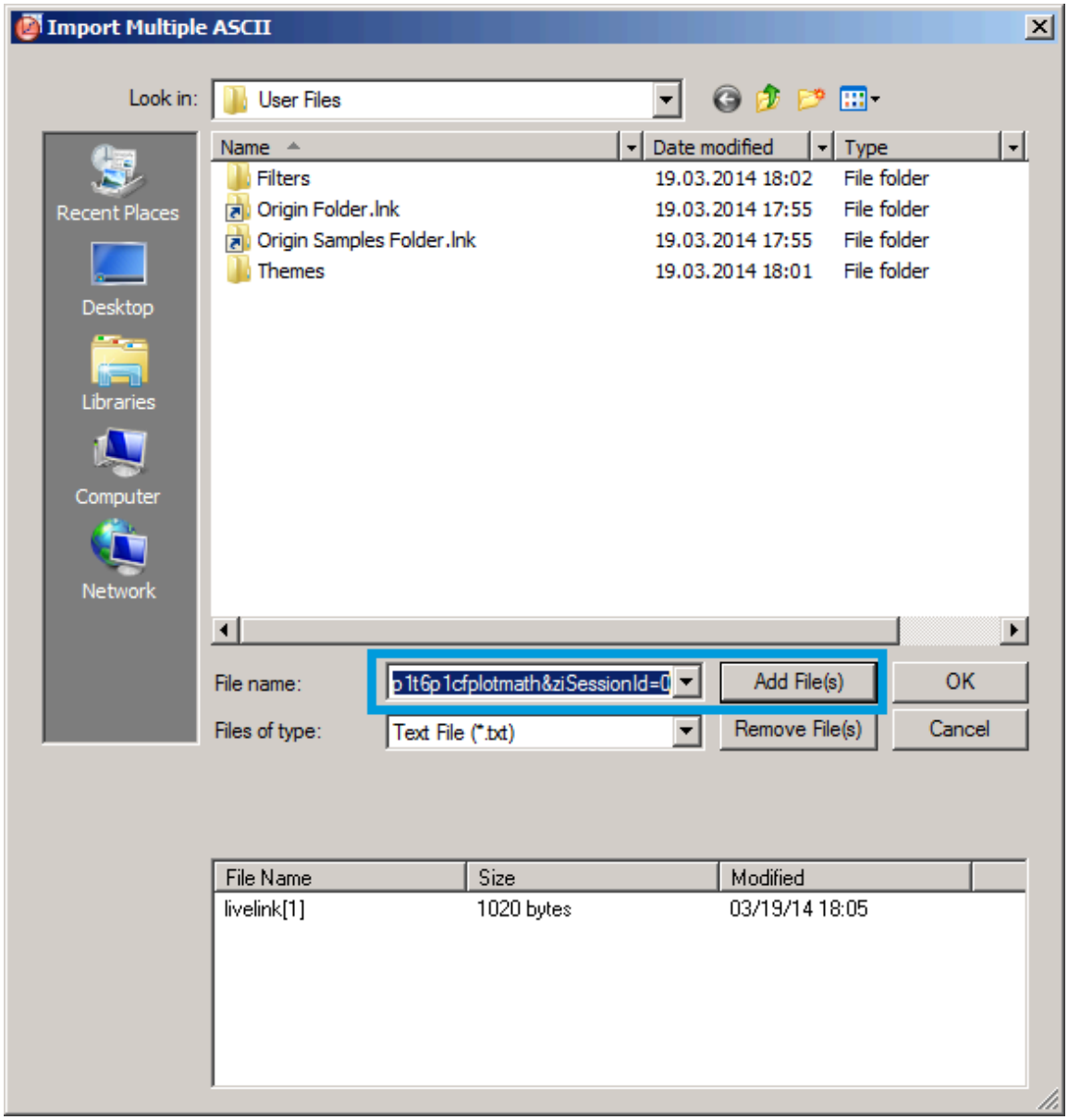
1. Open the import wizard by clicking on the icon highlighted below.



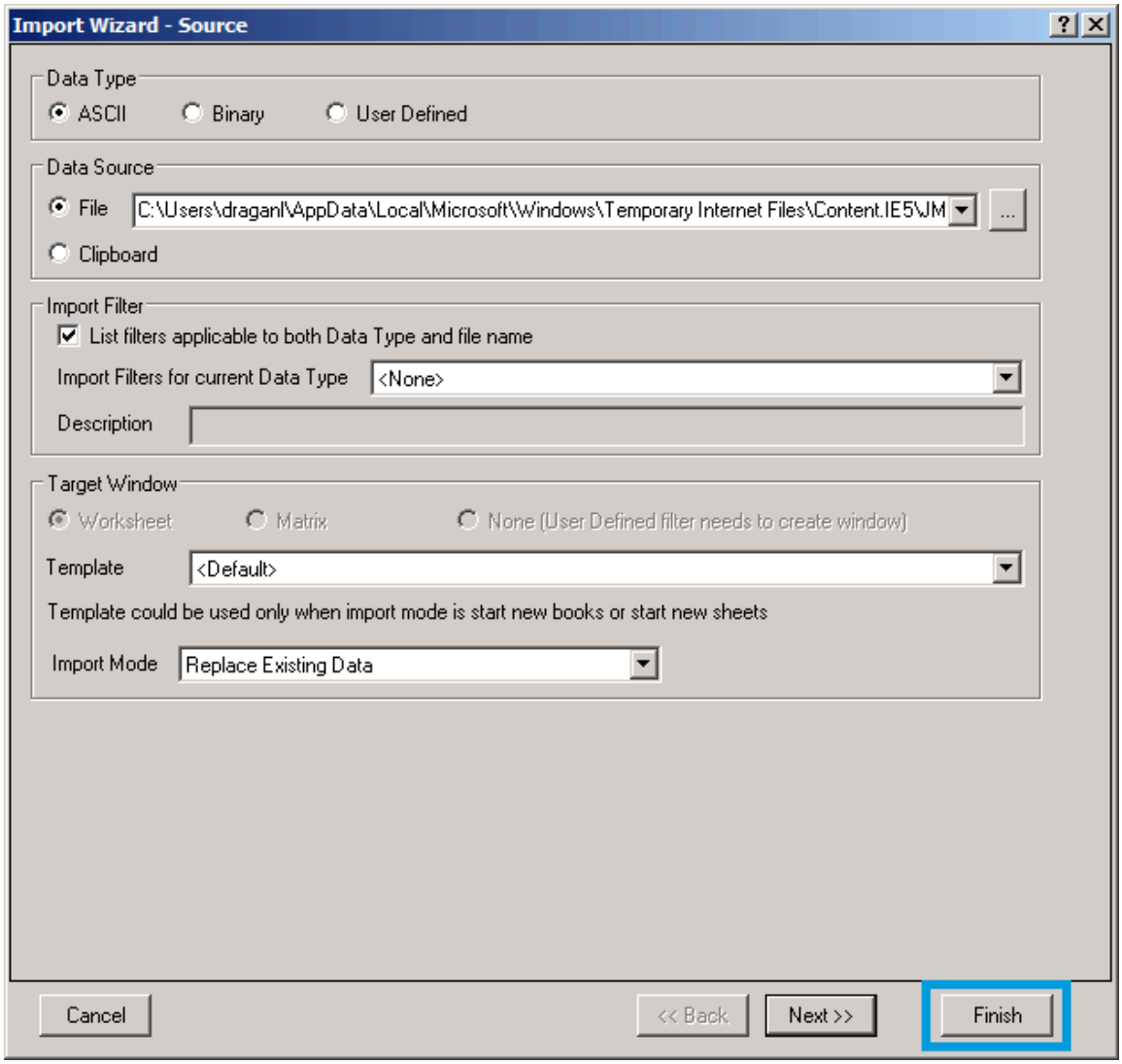
2. Ensure that the ASCII button is selected. Click the "..." button. See screenshot below. The "Import Multiple ASCII" dialog will appear.



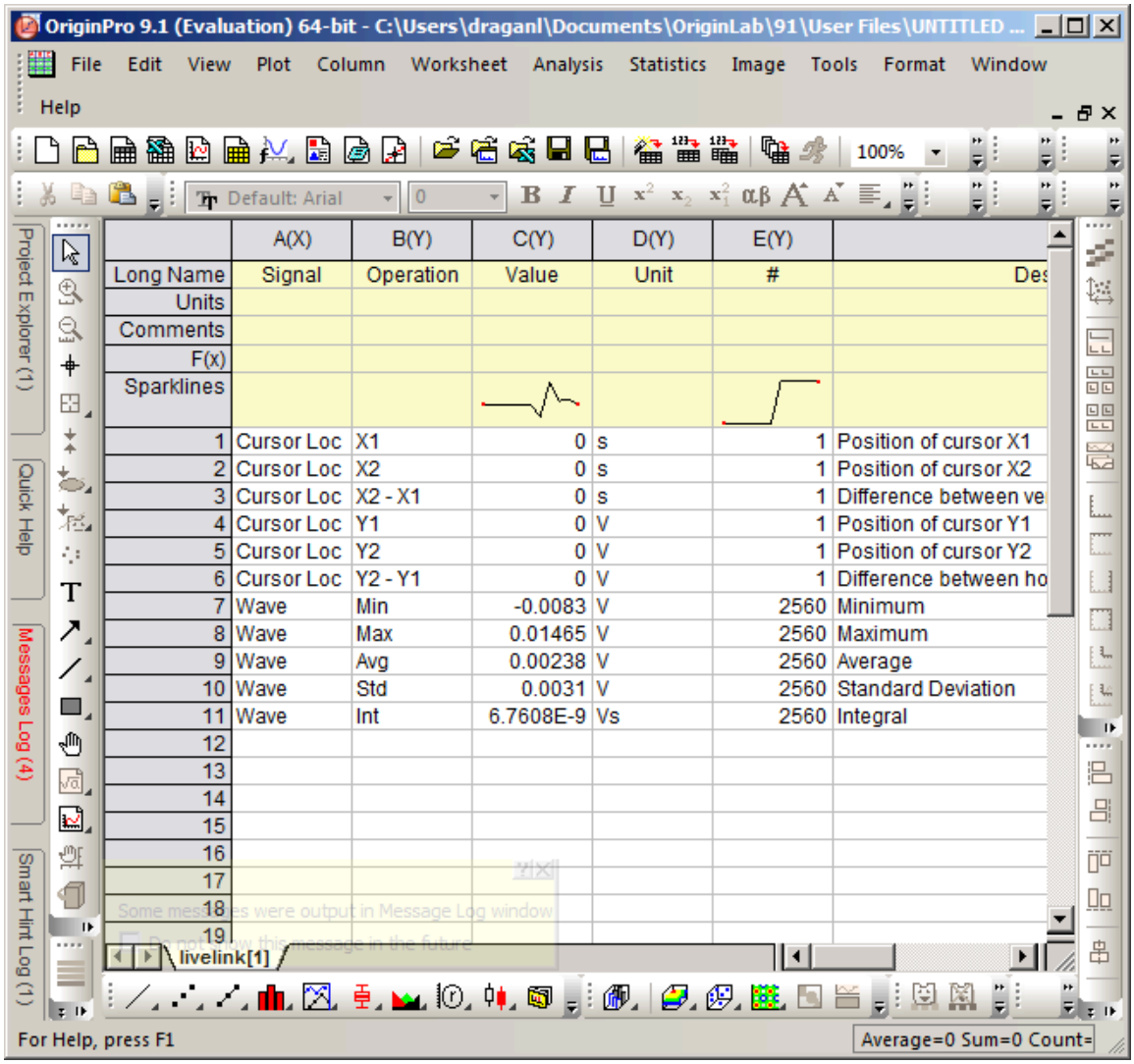
3. Paste the link text from the "LabOne Net Link" dialog into the entry field highlighted below. Then click "Add File(s)" followed by "OK".



4. Back in the "Import Wizard - Source" dialog click "Finish".



5. The data will appear in the Origin main window.



	A(X)	B(Y)	C(Y)	D(Y)	E(Y)	
Long Name	Signal	Operation	Value	Unit	#	Des
Units						
Comments						
F(x)						
Sparklines						
1	Cursor Loc	X1	0	s	1	Position of cursor X1
2	Cursor Loc	X2	0	s	1	Position of cursor X2
3	Cursor Loc	X2 - X1	0	s	1	Difference between ve
4	Cursor Loc	Y1	0	V	1	Position of cursor Y1
5	Cursor Loc	Y2	0	V	1	Position of cursor Y2
6	Cursor Loc	Y2 - Y1	0	V	1	Difference between ho
7	Wave	Min	-0.0083	V	2560	Minimum
8	Wave	Max	0.01465	V	2560	Maximum
9	Wave	Avg	0.00238	V	2560	Average
10	Wave	Std	0.0031	V	2560	Standard Deviation
11	Wave	Int	6.7608E-9	Vs	2560	Integral
12						
13						
14						
15						
16						
17						
18						
19						

4.2. Measurement Functionality

In this section, the measurement functionality of the SHFSG 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 can be represented by a node tree. Each node can either set, read or poll settings or data from the device. Most of the functionality resides within the different output Channels of the SHFSG, each of which is represented by its own version of the SGChannels branch: /DEV..../SGCHANNELS/n/.... 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

Because of the recent launch of the SHFSG instrument, 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.

4.2.1. Output Tab

The Output tab can be used to configure the center frequency and maximum output power of the generated signals. It is available on all SHFSG instruments.

Features Overview

- Enable/disable output
- Define the Center Frequency of the modulation band
- Define the output power range
- Switch between RF and LF paths

Description

Table 4.14. 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 SHFSG uses the double super-heterodyne frequency upconversion technique to generate its RF output frequencies, and each Output Channel has its own frequency upconversion chain. Each Output 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 0.1 GHz. Both variants of the SHFSG contain 4 synthesizers. In the 4-channel variant, each Output Channel therefore has its own synthesizer, whereas in the 8-channel variant, there is 1 synthesizer per Output Channel pair. This means that Output Channels 1 and 2 must share the same RF center frequency in the 8-channel variant of the SHFSG Instrument when using the RF path. To achieve different output frequencies on

Output Channels 1 and 2 in the 8-channel variant, digital modulation must be employed (see the Modulation Tab). When using the LF path, the center frequencies of each channel can be set independently of the other channels in both variants of the SHFSG Instrument.

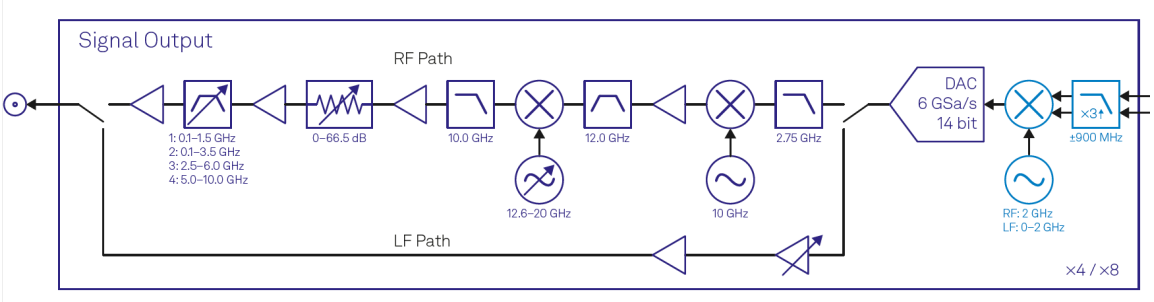


Figure 4.12. Analog Signal Output Stage

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). 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 SHFSG, such that only a few settings need to be set in the Output band parameters of the SHFSG: Center Frequency, Output Range, and Output On.

Output Tab in the LabOne GUI

The Output settings can be accessed through the Output tab of the SHFSG's LabOne general user interface. After clicking on the tab, an overview subtab opens that displays all settings for all available Output Channels.

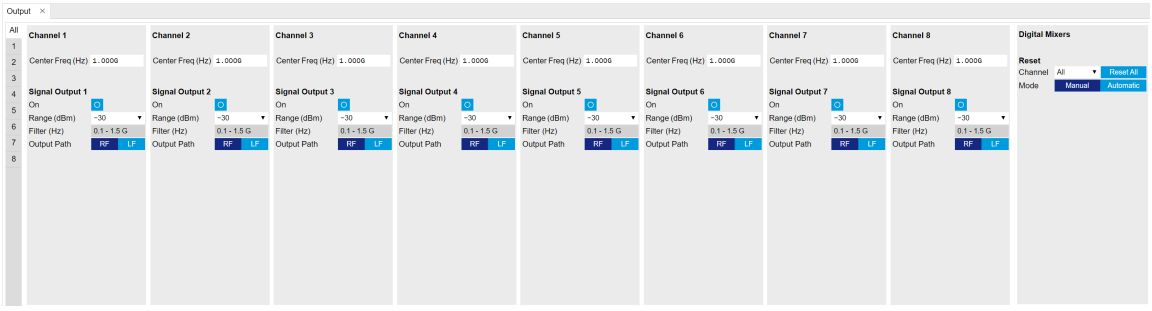


Figure 4.13. The Overview Output Tab of the GUI

With the selectors at the left side of the Output tab, the detailed view of the up-conversion chain for the different Output Channels can be displayed. Each detailed view shows the available settings in the first, leftmost panel. In the second panel, a graphical representation of the currently selected parameters of the up-conversion chain is displayed.

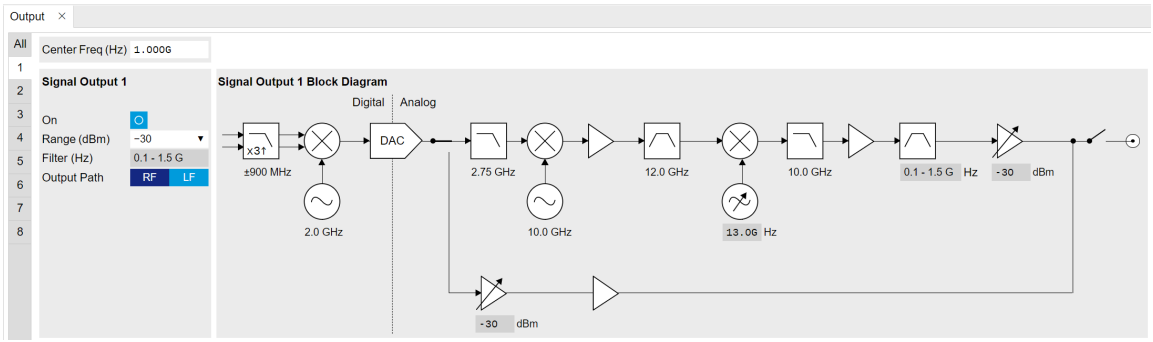


Figure 4.14. A detailed Output Tab of the GUI

Functional Elements

Table 4.15. Output tab

Control/Tool	Option/Range	Description
Center Frequency		Center frequency of the output band at the output of the instrument. A copy of the displayed value is also contained in the read-only node '/{device}/sgchannels/{n}/centerfreq'.
Center Frequency		Set center frequency of digital mixer.
Center Frequency		Center frequency of the detection band at the input/output of the instrument.
Variable Local Oscillator Frequency		This local oscillator converts between the fixed signal band around 12 GHz and the variable readout band at the In/Out connector. Shared between the Signal Input/Output modules of the same channel, its value is given by the user-determined Center Frequency value + 12 GHz.
Variable Local Oscillator Frequency		This local oscillator converts between the fixed signal band around 12 GHz and the variable output band at the Out connector. Its value is given by the user-determined Center Frequency value + 12 GHz.
Range		Maximal power at the input of the instrument.
Range		Maximal power at the output of the instrument.
Selectable RF Output Filter		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	LF path is used.	Switch between RF and LF output path.

Control/Tool	Option/Range	Description
	RF path is used.	
Delay (s)		This value adds a delay to both the signal and trigger/marker outputs.
Channel Select		Select which channel is to be cleared.
Reset All		Reset all the channels.
Reset Channel		Reset only the selected channel.
Mode	In manual mode the instrument does not automatically reset NCOs when switching a channel from LF to RF mode.	Configure the NCO reset 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.	

4.2.2. 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 SHFSG 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 4.16. 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 4.15) is divided into three sections: Oscillators, Sine Generators, and Waveform Generators.

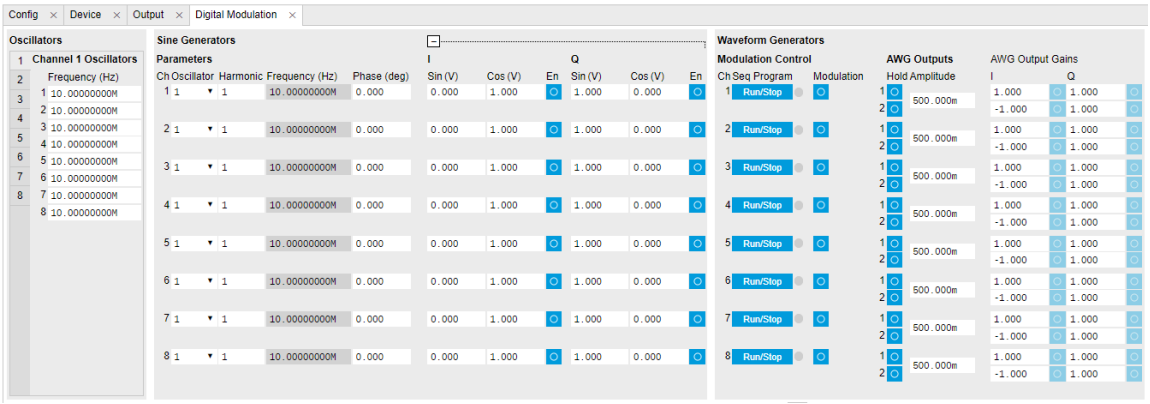


Figure 4.15. LabOne UI: Digital Modulation tab

The purpose of the Digital Modulation tab is to configure the digital sine generator of each SHFSG 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 4.17. 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.

Control/Tool	Option/Range	Description
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	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.
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).

4.2.3. AWG Tab

The AWG tab is available on all SHFSG Signal Generator instruments.

Features

- 4- or 8-channel arbitrary waveform generator
- 100 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 4.18. 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 4.16) 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.

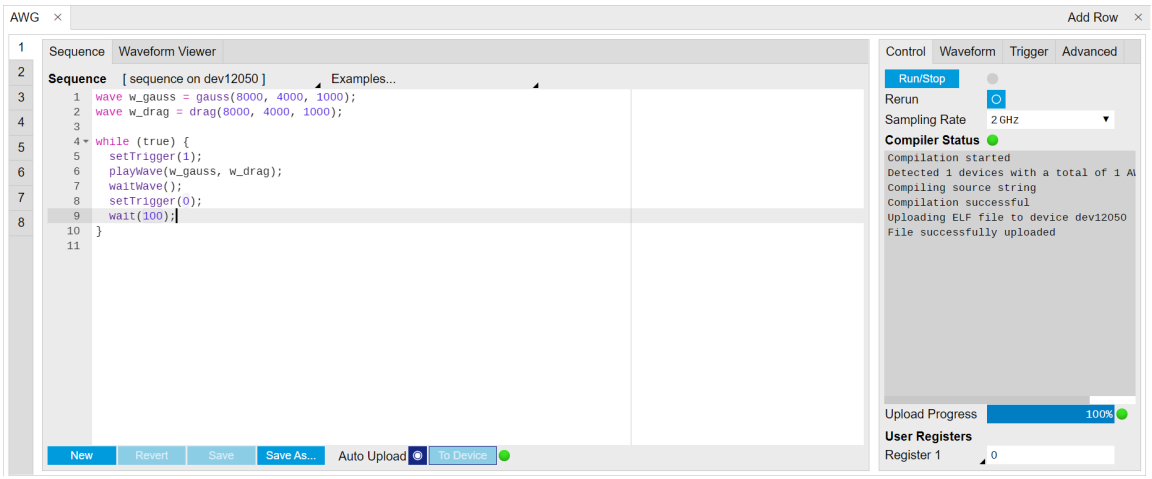


Figure 4.16. LabOne UI: AWG tab

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 Chapter 3 . The LabOne sequence programming language is specified in detail in the section called “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 4.17. 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.

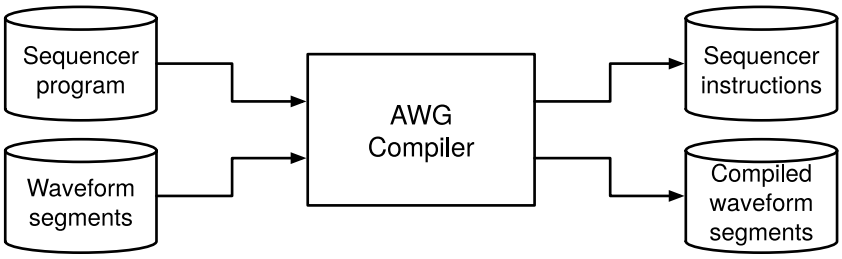


Figure 4.17. AWG sequence program compilation process

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 Section 4.1.2 so they become accessible from the user interface. The files can be managed in the Section 4.1.4 and their location in the directory structure is shown in Table 4.19. 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 4.19. Sequence program and waveform file location

File type	Location
Waveform files (Windows)	`C:\Users\<user name>\Documents\Zurich Instruments \LabOne\WebServer\awg\waves`
Sequence programs (Windows)	`C:\Users\<user name>\Documents\Zurich Instruments \LabOne\WebServer\awg\src`
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 4.20. 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-Delete	Remove to line end
Alt-Backspace	Remove to line start
Ctrl-Backspace	Remove word left
Ctrl-Delete	Remove word right

Table 4.21. 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

Shortcut	Action
Alt-Shift-Left	Select to line start
Shift-Up	Select up
Shift-Down	Select down
Shift-PageUp	Select page up
Shift-PageDown	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 4.22. 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
PageUp	Go to page up
PageDown	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 4.23. Find/Replace

Shortcut	Action
Ctrl-F	Find
Ctrl-H	Replace
Ctrl-K	Find next
Ctrl-Shift-K	Find previous

Table 4.24. Folding

Shortcut	Action
Alt-L	Fold selection
Alt-Shift-L	Unfold

Table 4.25. 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
Insert	Overwrite
Ctrl-Shift-E	Macros replay
Ctrl-Alt-E	Macros recording
Delete	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 Chapter 3 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);
  // Wait until waveform playback has ended
  waitWave();
  // 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 4.26. Programming keywords

Keyword	Description
`const`	Constant declaration
`var`	Integer variable declaration
`cvar`	Compile-time variable declaration
`string`	Constant string declaration
`true`	Boolean true constant

Keyword	Description
`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 (baseSamplingClock)/2^n. The AWG rates constants are specified for the sampling clock of 2.0 GHz of the SHFSG.

Table 4.27. Predefined Constants

Name	Value	Description
commandTableEntries	{4096}	
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>	
ZSYNC_DATA_RAW	commandTableEntries + 0	Constant to use as argument to getZSyncData/getFeedback. Return the data received on the ZSync as-is without parsing.
ZSYNC_DATA_PQSC_REGISTER	commandTableEntries + 1	Constant to use as argument to getZSyncData/getFeedback. Get last readout register forwarded by the PQSC.
ZSYNC_DATA_PQSC_DECODER	commandTableEntries + 2	Constant to use as argument to getZSyncData/getFeedback. Get last output of the decoder received from the PQSC.
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.
AWG_USERREG_SWEEP_COUNT0	50	Constant for the sweep count register 0.
AWG_USERREG_SWEEP_COUNT1	51	Constant for the sweep count register 1.

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.

$$h(x) = 0 \qquad (4)$$

Parameter
- `samples:` Number of samples in the waveform

Return resulting waveform

wave ones(const samples)

Constant amplitude of 1 over the defined number of samples.

$$h(x) = 1 \qquad (5)$$

Parameter
- `samples:` Number of samples in the waveform

Return 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).

$$h(x) = a \cdot \sin(2\pi f \frac{x}{N} + p) \qquad (6)$$

Parameter
- `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

Return 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).

$$h(x) = a \cdot \cos(2\pi f \frac{x}{N} + p) \qquad (7)$$

Parameter
- `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

Return 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 (β) and number of samples (N).

$$h(x) = \begin{cases} a & \text{if } x = p \\ a \cdot \dfrac{\sin(2\pi \cdot beta \cdot \frac{x-p}{N})}{2\pi \cdot beta \cdot \frac{x-p}{N}} & \text{else} \end{cases} \qquad (8)$$

Parameter
- `amplitude:` Amplitude of the signal (optional)
- `beta:` Width of the function
- `position:` Peak position of the function
- `samples:` Number of samples in the waveform

Return 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).

$$h(x) = s + \frac{x(e-s)}{N-1} \qquad (9)$$

Parameter
- `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

Return 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.

Parameter
- `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

Return 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.

Parameter
- `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

Return 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).

$$h(x) = a \cdot e^{-\frac{(x-p)^2}{2 \cdot w^2}} \qquad (10)$$

Parameter
- `amplitude:` Amplitude of the signal (optional)
- `position:` Peak position of the pulse
- `samples:` Number of samples in the waveform
- `width:` Width of the pulse

Return 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.

$$h(x) = a \cdot \frac{\sqrt{e}(p-x)}{w} \cdot e^{-\frac{(x-p)^2}{2 \cdot w^2}} \qquad (11)$$

Parameter
- `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

Return 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).

$$h(x) = -a \cdot (\alpha_0 - \alpha_1 \cos(\frac{2\pi x}{N-1}) + \alpha_2 \cos(\frac{4\pi x}{N-1})) \quad (12)$$
$$\alpha_0 = \frac{1-\alpha}{2}; \quad \alpha_1 = \frac{1}{2}; \quad \alpha_2 = \frac{\alpha}{2};$$

Parameter
- `alpha:` Width of the function
- `amplitude:` Amplitude of the signal (optional)
- `samples:` Number of samples in the waveform

Return resulting waveform

wave hamming(const samples, const amplitude=1.0)

Hamming window function with arbitrary amplitude (a) and number of samples (N).

$$h(x) = a \cdot (\alpha - \beta \cos(\frac{2\pi x}{N-1})) \quad (13)$$
$$\text{with} \alpha = 0.54 \text{and} \beta = 0.46$$

Parameter
- `amplitude:` Amplitude of the signal (optional)
- `samples:` Number of samples in the waveform

Return resulting waveform

wave hann(const samples, const amplitude=1.0)

Hann window function with arbitrary amplitude (a) and number of samples (N).

$$h(x) = a \cdot 0.5 \cdot (1 - \cos(\frac{2\pi x}{N-1})) \quad (14)$$

Parameter
- `amplitude:` Amplitude of the signal
- `samples:` Number of samples in the waveform

Return resulting waveform

wave rect(const samples, const amplitude)

Rectangle function, constants amplitude (a) over the defined number of samples.

$$h(x) = \mathrm{a} \qquad (15)$$

Parameter
- `amplitude:` Amplitude of the signal
- `samples:` Number of samples in the waveform

Return 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.

Parameter
- `markerValue:` Value of the marker bits
- `samples:` Number of samples in the waveform

Return resulting waveform

wave rand(const samples, const amplitude=1.0, const mean, const stdDev)

White noise with arbitrary amplitude, power and standard deviation.

Parameter
- `amplitude:` Amplitude of the signal
- `mean:` Average signal level
- `samples:` Number of samples in the waveform
- `stdDev:` Standard deviation of the noise signal

Return resulting waveform

wave randomGauss(const samples, const amplitude=1.0, const mean, const stdDev)

White noise with arbitrary amplitude, power and standard deviation.

Parameter
- `amplitude:` Amplitude of the signal
- `mean:` Average signal level
- `samples:` Number of samples in the waveform
- `stdDev:` Standard deviation of the noise signal

Return resulting waveform

wave randomUniform(const samples, const amplitude=1.0)

Random waveform with uniform distribution.

Parameter
- `amplitude:` Amplitude of the signal
- `samples:` Number of samples in the waveform

Return 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)".

Parameter
- `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

Return 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.

Parameter
- `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

Return 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 (β) and width (w) and number of samples (N).

$$h(y) = a\frac{\sin(y\pi(1-\beta)) + 4y\beta\cos(y\pi(1+\beta))}{y\pi(1-(4y\beta)^2)} \quad \text{with}\, y(x) = 2w\frac{x-p}{N} \tag{16}$$

Parameter
- `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

Return 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.

Parameter
- `value:` Waveform amplitude at the respective sample

Return 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.

Parameter
- `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

Return waveform object

The following table contains the definition of functions for waveform editing.

wave join(wave wave1, wave wave2, const interpolLength=0)

Connect two or more waveforms with optional linear interpolation between the waveforms.

Parameter
- `interpolLength:` Number of samples to interpolate between waveforms (optional, default 0)
- `wave1:` Input waveform
- `wave2:` Input waveform

Return joined waveform

wave join(wave wave1, wave wave2,...)

Connect two or more waveforms.

Parameter
- `wave1:` Input waveform
- `wave2:` Input waveform

Return joined waveform

wave interleave(wave wave1, wave wave2,...)

Interleave two or more waveforms sample by sample.

Parameter
- `wave1:` Input waveform
- `wave2:` Input waveform

Return interleaved waveform

wave add(wave wave1, wave wave2,...)

Add two or more waveforms sample by sample. Alternatively, the "+" operator may be used for waveform adddition.

Parameter
- `wave1:` Input waveform
- `wave2:` Input waveform

Return 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.

Parameter
- `wave1:` Input waveform
- `wave2:` Input waveform

Return 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.

Parameter
- `factor:` Scaling factor
- `waveform:` Input waveform

Return scaled waveform

wave flip(wave waveform)

Flip the input waveform back to front and return the flipped waveform. The input waveform remains unchanged.

Parameter
- `waveform:` Input waveform

Return 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".

Parameter
- `from:` First sample of the cut waveform
- `to:` Last sample of the cut waveform
- `waveform:` Input waveform

Return 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.

$$y(n) = \frac{1}{a_0}\left(\sum_{i=0}^{M} b_i x_{n-i} - \sum_{i=1}^{N} a_i y_{n-i}\right) \quad (17)$$
$$\text{with} M = \text{length}(b) - 1$$
$$\text{and} N = \text{length}(a) - 1$$

Parameter
- `a:` Denominator coefficients
- `b:` Numerator coefficients
- `x:` Input waveform

Return filtered waveform

wave circshift(wave a, const n)

Circularly shifts a 1D waveform and returns it.

Parameter
- `n:` Number of elements to shift
- `waveform:` Input waveform

Return 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.

Parameter
- `value:` The value to write to the DIO (const or var)

var getDIO()

Reads a 32-bit value from the DIO bus.

Return var containing the read value

var getDIOTriggered()

Reads a 32-bit value from the DIO bus as recorded at the last DIO trigger position.

Return 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.

Parameter
- `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 ns = 20 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 ns = 12 ns.

Parameter
- `cycles:` number of cycles to wait

void waitTrigger(const mask, const value)

Waits until the masked trigger input is equal to the given value.

Parameter
- `mask:` mask to be applied to the input signal
- `value:` value to be compared with the trigger input

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.

Parameter
- `index:` index of the Digital Trigger input to be read; can be either 1 or 2

Return trigger state, either 0 or 1

void error(string msg,...)

Throws the given error message when reached.

Parameter
- `msg:` Message to be displayed

void info(string msg,...)

Returns the specified message when reached.

Parameter
- `msg:` Message to be displayed

void setID(var id)

void setSeqIndex(var id)

void waitWave()

Waits until the AWG is done playing the current waveform.

void setRate(const rate)

Overwrites the default Sampling Rate for the following playWave commands.

Parameter
- `rate:` New default sampling rate

void randomSeed()

Generate a new seed for the subsequent random vector commands.

void assignWaveIndex(const output, wave waveform, const index)

Assigns an index to the specified waveform(s). Channels/waveforms specification must match that of the corresponding playWave. Once program is loaded to AWG, the waveforms can be r/w accessed via the "<dev>/AWGS/<n>/WAVEFORM/WAVES/<index>" node, and addressed from the command table using the assigned index.

Parameter
- `index:` index to assign
- `output:` defines on which output the following waveform to be played
- `waveform:` waveform to be played

void assignWaveIndex(wave waveform, const index)

Assigns an index to the specified waveform(s). Once a sequencer program is uploaded to the AWG, the waveforms can be written and read via the "<dev>/AWGS/<n>/WAVEFORM/WAVES/

<index>" API node, and addressed from the command table using the assigned index. Channels/ waveforms specification must match that of the corresponding playWave.

Parameter
- `index:` index to assign
- `waveform:` waveform to be played

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.

Parameter
- `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.

Parameter
- `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.

Parameter
- `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.

Parameter
- `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.

Parameter
- `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.

Parameter
- `register:` The register to be read (0 to 15)

Return 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.

Parameter
- `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.

Parameter
- `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.

Parameter
- `index:` Index of the digital trigger input; can be either 1 or 2.

void executeTableEntry(var index)

Execute the entry of the AWG command/waveform table with the given index. An entry of the command/waveform table contains a waveform playback instruction as well as additional space for instructions for real-time setting of Sine Generator phases and AWG amplitude. The entries of the command/waveform table can be specified by direct upload of a JSON string to the "<dev>/SGCHANNELS/<n>/AWG/COMMANDTABLE/DATA" API node.

Parameter
- `data_type:` ZSYNC_DATA_RAW: The raw ZSYNC data is the table entry to execute ZSYNC_DATA_PQSC_REGISTER: The last readout register forwarded by the PQSC is the table entry to execute ZSYNC_DATA_PQSC_DECODER: The last output of the decoder received from the PQSC is the table entry to execute
- `index:` table entry that shall be executed

- `wait_cycles:` Wait for the specified number of cycles after the most recent waitZSyncTrigger() or waitDigTrigger() instruction

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 unsigned int value. Zero is invalid as seed.

Parameter
- `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 AND 1; lsfr = lsfr >> 1; if (lsb == 1) then: lsfr = 0xb400 XOR lsfr; rand = ((lsfr * (upper-lower+1) >> 16) + lower) AND 0xffff;.

Return Random value rand

void setPRNGRange(var lower, var upper)

Configures the range of the psuedo random number generator (PRNG) to generate output in range [lower, upper].

Parameter
- `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.

Parameter
- `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.

Parameter
- `phase:` Phase value [degree]

void resetOscPhase(const mask)

Reset the phase of the oscillators specified by the binary mask argument. Each AWG core can access the oscillators of its SGCHANNEL.

Parameter
- `mask:` one-hot encoding to reset phase of individual oscillators

void resetOscPhase()

Reset the phase of all oscillators controllable by the AWG core.

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.

Parameter
- `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 samples. Behaves as if same length constant waveform is played using playWave, but without consuming waveform memory.

Parameter
- `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 setWaveDIO instruction. 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.

Parameter
- `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 setSweepStep(const oscillator_index, var sweep_index)

Executes a step within a frequency sweep.

Parameter
- `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.

Parameter
- `freq:` Frequency to be set [Hz]
- `oscillator_index:` Index of oscillator

var getZSyncData(const data_type)

Read the last received message on ZSync. The argument specify which data the function should return.

Parameter
- `data_type:` Specifies which data the function should return: ZSYNC_DATA_RAW: Return the data received on the ZSync as-is without parsing. The structure of the message can change across different LabOne releases. ZSYNC_DATA_PQSC_REGISTER: Get last readout register forwarded by the PQSC ZSYNC_DATA_PQSC_DECODER: Get last output of the decoder received from the PQSC.
- `wait_cycles:` Wait for the specified number of cycles after the most recent waitZSyncTrigger() instruction

Return var containing the read value

var getZSyncData(const data_type, var wait_cycles)

var getFeedback(const data_type)

Read the last received feedback message. The argument specify which data the function should return.

Parameter
- `data_type:` Specifies which data the function should return: ZSYNC_DATA_RAW: Return the data received on the ZSync as-is without parsing. The structure of the message can change across different LabOne releases. ZSYNC_DATA_PQSC_REGISTER: Get last readout register forwarded by the PQSC ZSYNC_DATA_PQSC_DECODER: Get last output of the decoder received from the PQSC.
- `wait_cycles:` Wait for the specified number of cycles after the most recent waitZSyncTrigger() instruction

Return var containing the read value

var getFeedback(const data_type, var wait_cycles)

void playWaveZSync(const data_type)

void playWaveZSync(const data_type, var wait_cycles)

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 4.28. 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
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

Function	Description
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 4.29. Mathematical Constants

Name	Value	Description
M_E	2.71828182845904523536028747135266250	e
M_LOG2E	1.44269504088896340735992468100189214	log2(e)
M_LOG10E	0.43429448190325182765112891891660508 2	log10(e)
M_LN2	0.69314718055994530941723212145817656 8	loge(2)
M_LN10	2.30258509299404568401799145468436421	loge(10)
M_PI	3.14159265358979323846264338327950288	pi
M_PI_2	1.57079632679489661923132169163975144	pi/2
M_PI_4	0.78539816339744830961566084581987572 1	pi/4
M_1_PI	0.31830988618379067153776752674502872 4	1/pi
M_2_PI	0.63661977236758134307553505349005744 8	2/pi
M_2_SQRTPI	1.12837916709551257389615890312154517	2/sqrt(pi)
M_SQRT2	1.41421356237309504880168872420969808	sqrt(2)
M_SQRT1_2	0.70710678118654752440084436210484903 9	1/sqrt(2)

Table 4.30. 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

Operator	Description	Priority
!=	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 4.31. 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
+	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);
}
```

Functional Elements

Table 4.32. 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.
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.

Control/Tool	Option/Range	Description
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 4.33. AWG tab: Waveform sub-tab

Control/Tool	Option/Range	Description
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.
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.

Table 4.34. AWG tab: Trigger sub-tab

Control/Tool	Option/Range	Description
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.
Display Format		Select PQSC Register and Decoder view format.
	Hexadecimal	ZSync parameters view format is hexadecimal.
	Decimal	ZSync parameters view format is decimal.
	Binary	ZSync parameters view format is binary.
PQSC Register Shift		The bit shift applied to the message received on ZSync interface coming from the PQSC readout registers.
PQSC Register Mask		4-bit value to select the bits of the message received on ZSync interface coming from the PQSC readout registers.
PQSC Register Offset		The additive offset applied to the message received on ZSync interface coming from the PQSC readout registers.

Control/Tool	Option/Range	Description
PQSC Decoder Shift		The bit shift applied to the message received on ZSync interface coming from the PQSC error decoder.
PQSC Decoder Mask		8-bit value to select the bits of the message received on ZSync interface coming from the PQSC error decoder.
PQSC Decoder Offset		The additive offset applied to the message received on ZSync interface coming from the PQSC error decoder.
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.

Table 4.35. AWG tab: Advanced sub-tab

Control/Tool	Option/Range	Description
Assembly	Text display	Displays the current sequence program in compiled form. Every line corresponds to one hardware instruction.
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.
Sequence Editor		Display and edit the sequence program.
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.
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.

4.2.4. DIO Tab

The DIO tab provides access to the settings and controls of the digital I/O as well as the Marker outputs and Trigger inputs and is available on all SHFSG instruments.

Features

- Monitor and control of digital I/O connectors
- Control settings for marker outputs and trigger inputs

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 4.36. 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, and Marker Outputs.

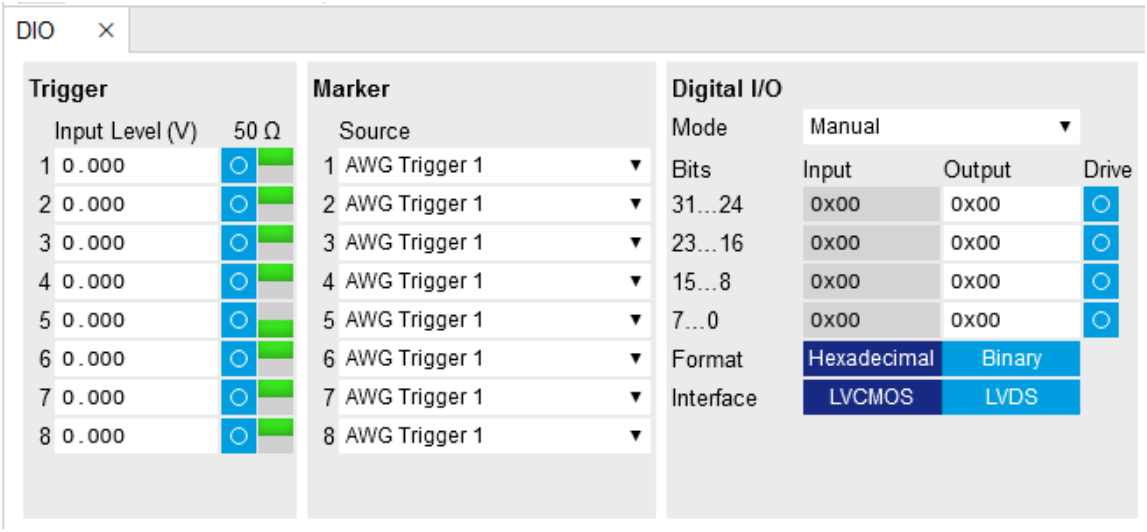


Figure 4.18. LabOne UI: DIO tab

The Digital I/O section provides numerical monitors to observe the states of the digital inputs and outputs. Moreover, with the values set in the Output column and the Drive button activated the states can also be actively set in different numerical formats.

The Trigger section shows the settings for the 4 or 8 Trig inputs on the front panel. 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 4 or 8 Mark outputs on the front panel. Alternatively, the outputs can be set to static high or low values.

Functional Elements

Table 4.37. Digital input and output channels, reference and trigger

Control/Tool	Option/Range	Description
DIO mode		Select DIO mode
	Sequencer	Enables control of DIO values by the Sequencer.
	Result	Sends discriminated Readout Results to the DIO.
	QA Results QCCS	Enables setting of DIO output values by QA results compatible with the QCCS
	QA Sequencer 1	Enables setting of DIO output values by QA Sequencer 1 commands.
	QA Sequencer 2	Enables setting of DIO output values by QA Sequencer 2 commands.
	QA Sequencer 3	Enables setting of DIO output values by QA Sequencer 3 commands.
	QA Sequencer 4	Enables setting of DIO output values by QA Sequencer 4 commands.
	Manual	Enables manual control of the DIO output bits.
DIO mode		Select DIO mode
	Manual	Enables manual control of the DIO output bits.
	QA Results	Sends discriminated readout results to the DIO.
	QA Sequencer 1	Enables control of DIO values by the sequencer of QA channel 1.
	SG Sequencer 1	Enables control of DIO values by the sequencer of SG channel 1.
	SG Sequencer 2	Enables control of DIO values by the sequencer of SG channel 2.
	SG Sequencer 3	Enables control of DIO values by the sequencer of SG channel 3.
	SG Sequencer 4	Enables control of DIO values by the sequencer of SG channel 4.
	SG Sequencer 5	Enables control of DIO values by the sequencer of SG channel 5.
	SG Sequencer 6	Enables control of DIO values by the sequencer of SG channel 6.
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.
QA Result Overflow	grey/yellow/red	Red: present overflow condition on the DIO interface during readout. Yellow: indicates an

Control/Tool	Option/Range	Description
		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.
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 kΩ	Trigger input impedance: When on, the trigger input impedance is 50 Ω, when off 1 kΩ.
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.
Marker output signal		Select the signal assigned to the marker output.
Run/Stop	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.
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.

4.2.5. DIO Interface

The figure below shows the architecture of the DIO input/output. The DIO port 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.

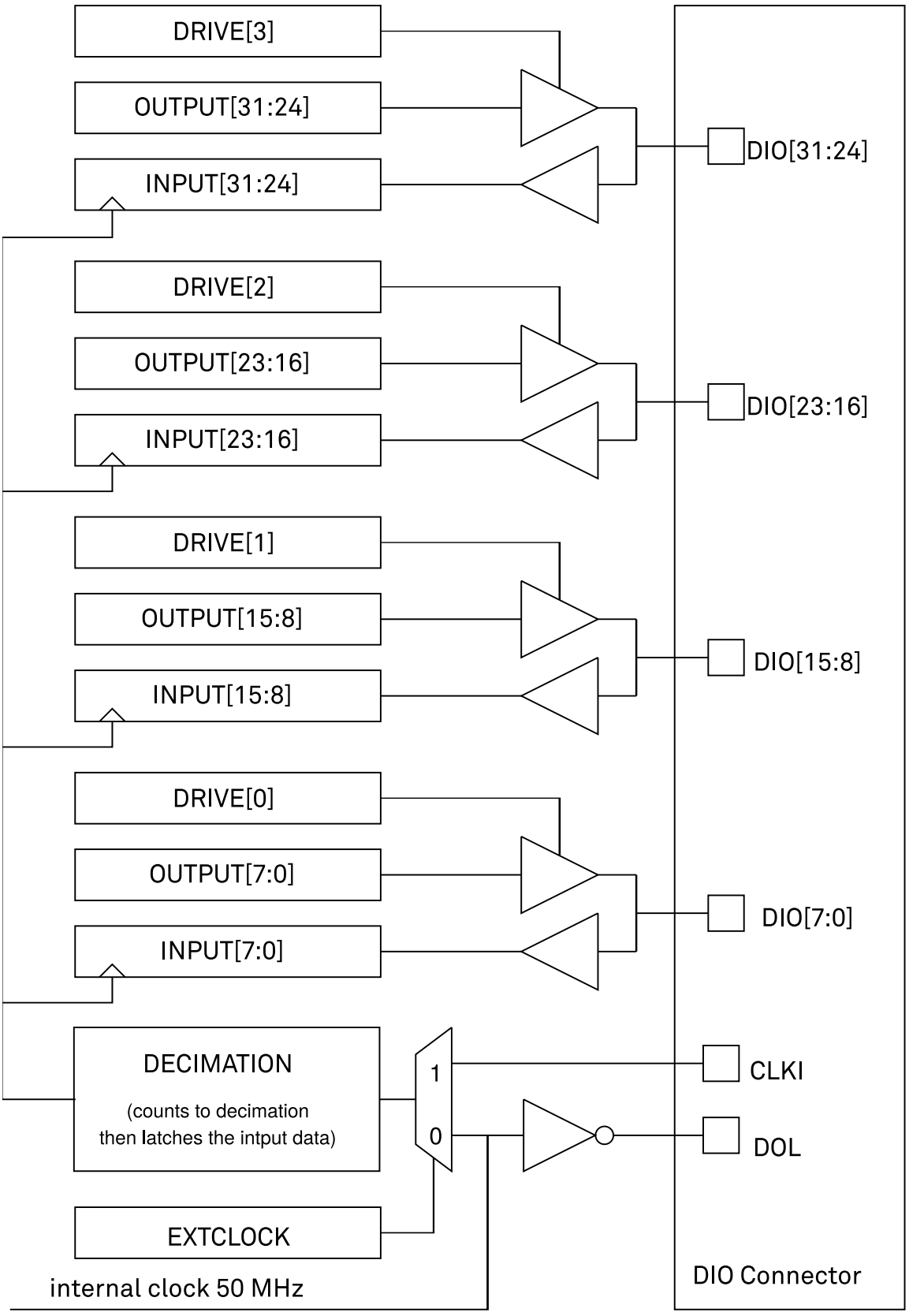


Figure 4.19. DIO input/output architecture

Readout Result communication via DIO

The table below 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 SHFSG instrument. In order to use these signals, the Digital I/O Mode and Drive setting have to be chosen accordingly.

Figure 4.20 belows 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.

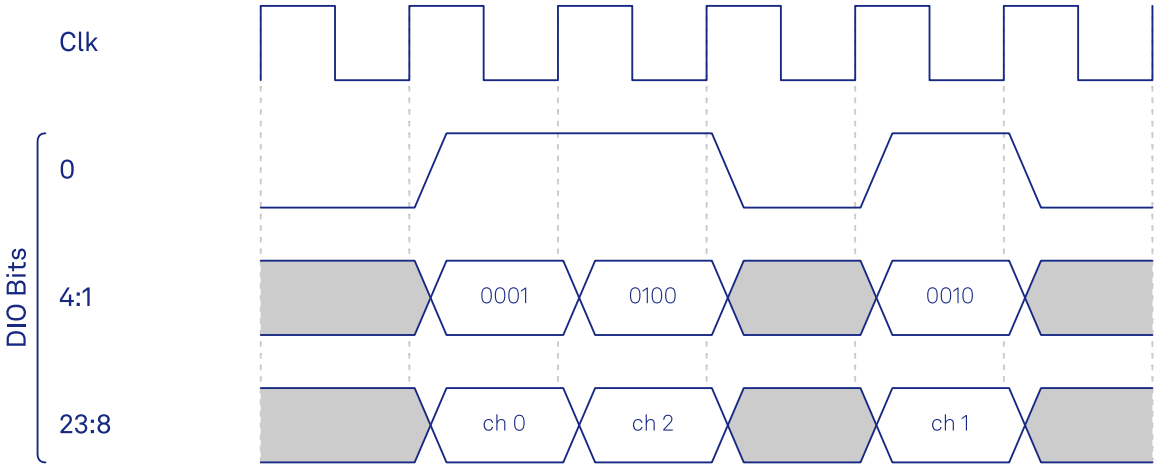


Figure 4.20. DIO Result communication

ZSync Interface

The ZSync link of the Zurich Instruments' Quantum Computing Control System (QCCS) enables instrument synchronization and communication on the system level. With the PQSC Programmable Quantum System Controller as the central controller, the different instruments for qubit control and readout interface in a scalable architecture.

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 SHFSG, the link is optimized for maximum data transfer to the central controller. For example, twice the bandwidth is reserved for results being transfered to the PQSC with respect to the allocated bandwith 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 SHFSG with the QCCS can be found in the user manual of the PQSC Programmable Quantum System Controller.

When combined with an SHFQA for qubit readout and the PQSC for relaying and processing of the readout results, the SHFSG can help perform feedback and error correction protocols.

Using the **`startQA`**- command, the SHFQA 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 4.21. 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.

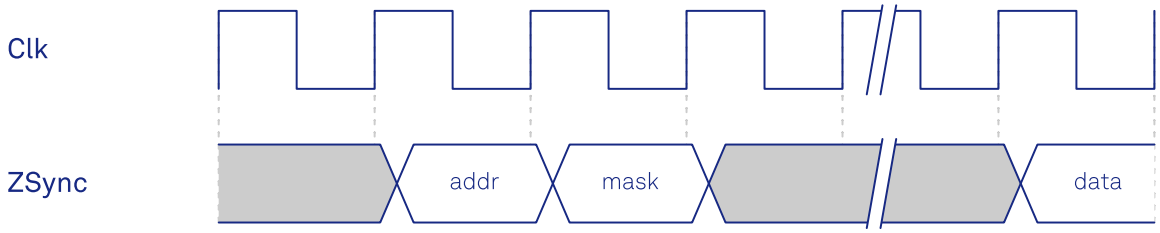


Figure 4.21. Readout Result communication via ZSync

Chapter 5. 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 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.

5.1. General Specifications

Table 5.1. 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 (±10%), 50/60 Hz		
dimensions (width x depth x height)	45.0 × 39.7 × 13.2 cm (no handle), 17.7 × 15.6 × 5.2 inch, 19 inch rack compatible		
weight	15 kg (33 lb)		
recommended calibration interval	2 years		

Table 5.2. Maximum ratings

Parameter	Min	Typ	Max
damage threshold Out	-	-	+30 dBm
damage threshold Mark Out	–0.7 V	-	+4 V
damage threshold Trig In (1 kΩ 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
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 5.3. Host computer requirements

Parameter	Description
supported Windows versions	Windows 10 (x86-64)
supported macOS versions	macOS 10.11+ (x86-64)
supported GNU/Linux distributions	Ubuntu 16.04+ (x86-64)
minimum host computer requirements	▬ Windows 10 64-bit ▬ Dual Core CPU ▬ 8 GB DRAM ▬ 1 Gbit/s Ethernet controller
recommended host computer requirements	▬ Windows 10 or GNU/Linux, 64-bit ▬ Quad Core CPU or better ▬ 16 GB DRAM or better ▬ 1 Gbit/s Ethernet controller ▬ Solid-state drive (SSD) ▬ USB 3.0 connection
supported processors	all 64-bit Intel and AMD processors

5.2. Analog Interface Specifications

Parameter	Details		Min	Typ	Max
connectors	-		SMA, front panel single-ended		
output impedance	-		-	50 Ω	-
output coupling	-		AC		
synthesizer frequency range	1 synthesizer/channel in SHFSG4, 1 synthesizer/channel pair in SHFSG8		0.6-8 GHz		
output range	into 50 Ω		-30 dBm	-	+10 dBm
output level accuracy	into 50 Ω		-	±(1 dBm of setting)	-
output level temperature drift	-		-	0.15 dB/C	-
D/A converter vertical resolution	-		14 bit		
D/A converter sampling rate	after internal x3 interpolation		6 GSa/s		
output phase noise	10 dBm, 1 GHz	offset 0.1 kHz	-	-83 dB/Hz	-
		offset 1 kHz	-	-92 dBc/Hz	-
		offset 10 kHz	-	-100 dBc/Hz	-
		offset 100 kHz	-	-102 dBc/Hz	-
	10 dBm, 4 GHz	offset 0.1 kHz	-	-82 dBc/Hz	-
		offset 1 kHz	-	-91 dBc/Hz	-
		offset 10 kHz	-	-100 dBc/Hz	-
		offset 100 kHz	-	-101 dBc/Hz	-
	10 dBm, 6 GHz	offset 0.1 kHz	-	-81 dBc/Hz	-
		offset 1 kHz	-	-90 dBc/Hz	-
		offset 10 kHz	-	-98 dBc/Hz	-
		offset 100 kHz	-	-100 dBc/Hz	-
	10 dBm, 8 GHz	offset 0.1 kHz	-	-80 dBc/Hz	-
		offset 1 kHz	-	-88 dBc/Hz	-
		offset 10 kHz	-	-96 dBc/Hz	-

Parameter	Details		Min	Typ	Max
		offset 100 kHz	-	-98 dBc/ Hz	-
output voltage noise density	10 dBm,1 GHz	offset > 200 kHz, into 50 Ω	-	-135 dBm/ Hz	-
	10 dBm, 4 GHz		-	-140 dBm/ Hz	-
	10 dBm, 6 GHz		-	-144 dBm/ Hz	-
	10 dBm, 8 GHz		-	-144 dBm/ Hz	-
output spurious free dynamic range (excluding harmonics)	0 dBm, 1 GHz	-800 to 800 MHz	-	74 dBc	-
	0 dBm, 4 GHz		-	66 dBc	-
	0 dBm, 6 GHz		-	60 dBc	-
	0 dBm, 8 GHz		-	65 dBc	-
output worst harmonic component	10 dBm, 1 GHz	-800 to 800 MHz	-	-40 dBc	-
	10 dBm, 4 GHz		-	-40 dBc	-
	10 dBm, 6 GHz		-	-38 dBc	-
	10 dBm, 8 GHz		-	-36 dBc	-

Table 5.4. Time Domain Output Characteristics

Parameter	Details	Min	Typ	Max
skew adjustment resolution	1 sample clock period	-	<0.5 ns	-

Table 5.5. Marker Outputs & Trigger Inputs

Parameter	Details	Min	Typ	Max
marker outputs	-	1 per channel, SMA output on front panel		
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	-
marker output period jitter	square wave, 100 MHz	-	60 ps p-p	-
trigger inputs	-	1 per channel, 1 SMA on front panel		
trigger input impedance	-	50 Ω / 1 kΩ		
trigger input voltage range	50 Ω impedance	–5 V	-	5 V
	1 kΩ impedance	–10 V	-	10 V
trigger input threshold range	50 Ω impedance	–5 V	-	5 V
	1 kΩ impedance	–10 V	-	10 V

Parameter	Details	Min	Typ	Max
trigger input threshold resolution	-	-	< 0.4 mV	-
trigger input threshold hysteresis	-	-	> 60 mV	-
trigger input min. pulse width	-	-	5 ns	-
trigger input max. operating frequency	-	-	300 MHz	-

Table 5.6. 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 5.7. 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	-

5.3. Digital Signal Generation Specifications

Table 5.8. Waveform Generation

Parameter	Details	Specification
number of AWG cores	-	1 per channel
AWG sampling rate	dual-channel	2 GSa/s
waveform memory per output channel	-	98 kSa
sequence length	-	32768 instructions per core
waveform granularity	-	16 samples
minimum waveform length	-	32 samples
sequencer clock frequency	-	250 MHz

5.4. Digital Interface Specifications

Table 5.9. 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 (2 times)

5.4.1. 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.

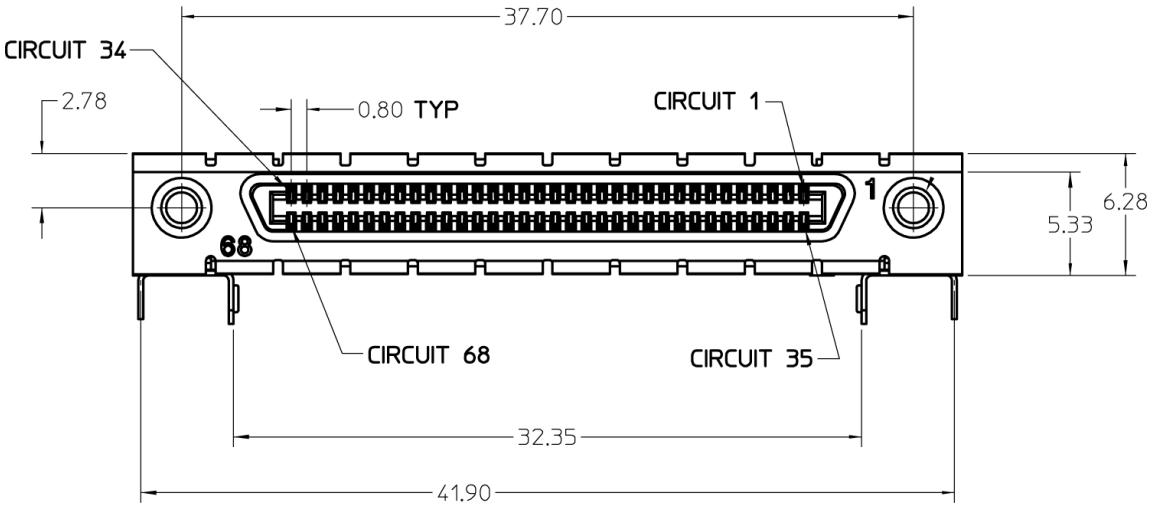


Figure 5.1. DIO HD 68 pin connector

Table 5.10. Electrical Specifications

Parameter	Details	Min	Typ	Max
supported DIO interface standards	-	LVCMOS/LVTTL (single-ended, 3.3 V); LVDS (differential)		
high-level input voltage VIH	LVCMOS/LVTTL	2.0 V	-	-
low-level input voltage VIL	LVCMOS/LVTTL	-	-	0.8 V
high-level output voltage VOH	LVCMOS/LVTTL at IOH < 12 mA	2.6 V	-	-
low-level output voltage VOL	LVCMOS/LVTTL at IOL < 12 mA	-	-	0.4 V
high-level output current IOH (sourcing)	LVCMOS/LVTTL	-	-	12 mA

Parameter	Details	Min	Typ	Max
low-level output current IOL (sinking)	LVCMOS/LVTTL	-	-	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 5.11. DIO Pin Assignment in LVCMOS/LVTTL 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 5.12. 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)

Pin	Name	Description
8 .. 1	DIO–[7:0]	digital input or output byte (set by user)

Chapter 6. Node Documentation

6.1. 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 "dev2006" are located in the tree in the branch:

/DEV2006/AUXOUTS/

In turn, each individual auxiliary output channel has its own branch underneath the "AUXOUTS" branch.

/DEV2006/AUXOUTS/0/
/DEV2006/AUXOUTS/1/
/DEV2006/AUXOUTS/2/
/DEV2006/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:

/DEV2006/AUXOUTS/0/DEMODSELECT
/DEV2006/AUXOUTS/0/LIMITLOWER
/DEV2006/AUXOUTS/0/LIMITUPPER
/DEV2006/AUXOUTS/0/OFFSET
/DEV2006/AUXOUTS/0/OUTPUTSELECT
/DEV2006/AUXOUTS/0/PREOFFSET
/DEV2006/AUXOUTS/0/SCALE
/DEV2006/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 Section 6.1.1.

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 Section 6.1.2.

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.

6.1.1. Node Properties and Data Types

A node may have one or more of the following properties:

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.

A node may contain data of the following types:

Integer	Integer data.
Double	Double precision floating point data.
String	A string array.
Enumerated (integer)	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

6.1.2. Exploring the Node Tree

6.1.3. 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. For example, the status bar 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.

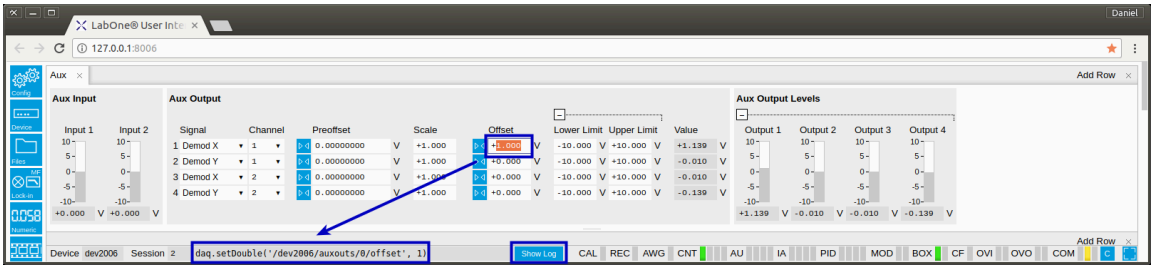


Figure 6.1. When a device's configuration is modified in the LabOne User Interface, the Status Bar displays the equivalent command to perform the same configuration via a LabOne programming interface. Here, the MATLAB code to modify auxiliary output 1's offset value is provided. When "Show Log" is clicked the entire configuration history is displayed in a new browser tab.

6.1.4. In the Instrument-specific User Manual

Each instrument user manual has a Device Node Tree chapter that contains complete reference documentation of every node available on the device. This documentation may be explored by branch to obtain a complete overview of which settings are available on the instrument.

6.1.5. 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 below 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('/dev2006/auxouts/0/offset')
# Out:
# /DEV2006/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
```

6.1.6. 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

6.2. Reference Node Documentation

This section describes all the nodes in the data server's node tree organized by branch.

6.2.1. CLOCKBASE

/DEV..../CLOCKBASE

Properties:	Read
Type:	Double
Unit:	Hz

Returns the internal clock frequency of the device.

6.2.2. DIOS (Digital I/O)

/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.
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.

/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.

6.2.3. 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.

6.2.4. 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

/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, Setting
Type: Integer (64 bit)
Unit: None

Writing to this node clears all data previously loaded to the command table of the device.

/DEV..../SGCHANNELS/n/AWG/COMMANDTABLE/DATA

Properties: Read, Write
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
Type: Integer (64 bit)
Unit: None

Status of the command table on the instrument. 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/DATA

Properties: Read
Type: ZIVectorData
Unit: None

A vector of 32-bit integers representing the values on the DIO interface.

/DEV..../SGCHANNELS/n/AWG/DIO/DELAY/INDEX

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Index of the bit on the DIO interface for which the delay should be changed.

/DEV..../SGCHANNELS/n/AWG/DIO/DELAY/VALUE

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

Corresponding delay value to apply to the given bit of the DIO interface in units of 150 MHz clock cycles. Valid values are 0 to 3.

/DEV..../SGCHANNELS/n/AWG/DIO/ERROR/TIMING

Properties: Read, Write, Setting

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, Write, Setting
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, Write, Setting
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, Write, Setting
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, Write, Setting
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
Type: Integer (64 bit)
Unit: None

Checksum of the uploaded ELF file.

/DEV..../SGCHANNELS/n/AWG/ELF/DATA

Properties: Write
Type: ZIVectorData
Unit: None

Accepts the data of the sequencer ELF file.

/DEV..../SGCHANNELS/n/AWG/ELF/LENGTH

Properties: Read, Write
Type: Integer (64 bit)
Unit: None

Length of the compiled ELF file.

/DEV..../SGCHANNELS/n/AWG/ELF/MEMORYUSAGE

Properties: Read
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
Type: ZIVectorData
Unit: None

The name of the uploaded ELF file.

/DEV..../SGCHANNELS/n/AWG/ELF/PROGRESS

Properties: Read

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

Enables the driving of the given AWG output channel.

/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
Type: Integer (64 bit)
Unit: None

AWG has a compiled wave form and is ready to be enabled.

/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, Setting
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 data part of the log.

/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 (64 bit)
Unit:	None

Selects the timestamp-triggered operation mode.

/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.

5 `erasing`

Erasing: Log is being erased

/DEV..../SGCHANNELS/n/AWG/RTLOGGER/TIMEBASE

Properties: Read
Type: Double
Unit: s

Minimal time difference between two timestamps. The value is equal to 1/(awg sampling rate).

/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/CONTINUE

Properties: Read, Write
Type: Integer (64 bit)
Unit: None

Reserved for future use.

/DEV..../SGCHANNELS/n/AWG/SEQUENCER/MEMORYUSAGE

Properties: Read
Type: Double
Unit: None

Size of the current sequence program relative to the device cache memory. The cache memory provides space for 8192 instructions.

/DEV..../SGCHANNELS/n/AWG/SEQUENCER/NEXT

Properties:	Read, Write
Type:	Integer (64 bit)
Unit:	None

Reserved for future use.

/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) waveform playback to finish, b) prefetch to finish, or c) trigger to arrive; Bit 3: AWG has detected an error

/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 cache memory. The cache memory provides space for 512 kSa of 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
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/AWG/ZSYNC/DECODER/MASK

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

8-bit value to select the bits of the message received on ZSync interface coming from the PQSC error decoder.

/DEV..../SGCHANNELS/n/AWG/ZSYNC/DECODER/OFFSET

Properties: Read, Write, Setting

Type: Integer (64 bit)
Unit: None

The additive offset applied to the message received on ZSync interface coming from the PQSC error decoder.

/DEV..../SGCHANNELS/n/AWG/ZSYNC/DECODER/SHIFT

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

The bit shift applied to the message received on ZSync interface coming from the PQSC error decoder.

/DEV..../SGCHANNELS/n/AWG/ZSYNC/REGISTER/MASK

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

4-bit value to select the bits of the message received on ZSync interface coming from the PQSC readout registers.

/DEV..../SGCHANNELS/n/AWG/ZSYNC/REGISTER/OFFSET

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

The additive offset applied to the message received on ZSync interface coming from the PQSC readout registers.

/DEV..../SGCHANNELS/n/AWG/ZSYNC/REGISTER/SHIFT

Properties: Read, Write, Setting
Type: Integer (64 bit)
Unit: None

The bit shift applied to the message received on ZSync interface coming from the PQSC readout registers.

/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/MARKER/SOURCE

Properties:	Read, Write
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/ON

Properties: Read, Write
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
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/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/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
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
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.

6.2.5. STATS

/DEV..../STATS/CMDSTREAM/BANDWIDTH

Properties: Read
Type: Double
Unit: Mbit/s

Command streaming bandwidth usage on the physical network connection between device and data server.

/DEV..../STATS/CMDSTREAM/BYTESRECEIVED

Properties: Read

Type: Integer (64 bit)
Unit: B

Number of bytes received on the command stream from the device since session start.

/DEV..../STATS/CMDSTREAM/BYTESSENT

Properties: Read
Type: Integer (64 bit)
Unit: B

Number of bytes sent on the command stream from the device since session start.

/DEV..../STATS/CMDSTREAM/PACKETSLOST

Properties: Read
Type: Integer (64 bit)
Unit: None

Number of command packets lost since device start. Command packets contain device settings that are sent to and received from the device.

/DEV..../STATS/CMDSTREAM/PACKETSRECEIVED

Properties: Read
Type: Integer (64 bit)
Unit: None

Number of packets received on the command stream from the device since session start.

/DEV..../STATS/CMDSTREAM/PACKETSSENT

Properties: Read
Type: Integer (64 bit)
Unit: None

Number of packets sent on the command stream to the device since session start.

/DEV..../STATS/CMDSTREAM/PENDING

Properties: Read
Type: Integer (64 bit)
Unit: None

Number of buffers ready for receiving command packets from the device.

/DEV..../STATS/CMDSTREAM/PROCESSING

Properties: Read
Type: Integer (64 bit)
Unit: None

Number of buffers being processed for command packets. Small values indicate proper performance. For a TCP/IP interface, command packets are sent using the TCP protocol.

/DEV..../STATS/DATASTREAM/BANDWIDTH

Properties: Read
Type: Double
Unit: Mbit/s

Data streaming bandwidth usage on the physical network connection between device and data server.

/DEV..../STATS/DATASTREAM/BYTESRECEIVED

Properties: Read
Type: Integer (64 bit)
Unit: B

Number of bytes received on the data stream from the device since session start.

/DEV..../STATS/DATASTREAM/PACKETSLOST

Properties: Read
Type: Integer (64 bit)
Unit: None

Number of data packets lost since device start. Data packets contain measurement data.

/DEV..../STATS/DATASTREAM/PACKETSRECEIVED

Properties: Read
Type: Integer (64 bit)
Unit: None

Number of packets received on the data stream from the device since session start.

/DEV..../STATS/DATASTREAM/PENDING

Properties: Read

Type: Integer (64 bit)
Unit: None

Number of buffers ready for receiving data packets from the device.

/DEV..../STATS/DATASTREAM/PROCESSING

Properties: Read
Type: Integer (64 bit)
Unit: None

Number of buffers being processed for data packets. Small values indicate proper performance. For a TCP/IP interface, data packets are sent using the UDP protocol.

/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/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.

6.2.6. STATUS

/DEV..../STATUS/ADC0MAX

Properties: Read
Type: Integer (64 bit)
Unit: None

The maximum value on Signal Input 1 (ADC0) during 100 ms.

/DEV..../STATUS/ADC0MIN

Properties: Read
Type: Integer (64 bit)
Unit: None

The minimum value on Signal Input 1 (ADC0) during 100 ms

/DEV..../STATUS/ADC1MAX

Properties: Read
Type: Integer (64 bit)
Unit: None

The maximum value on Signal Input 2 (ADC1) during 100 ms.

/DEV..../STATUS/ADC1MIN

Properties: Read
Type: Integer (64 bit)
Unit: None

The minimum value on Signal Input 2 (ADC1) during 100 ms

/DEV..../STATUS/FIFOLEVEL

Properties: Read
Type: Double
Unit: None

USB FIFO level: Indicates the USB FIFO fill level inside the device. When 100%, data is lost

/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/FLAGS/PACKETLOSSTCP

Properties: Read
Type: Integer (64 bit)
Unit: None

Flag indicating if tcp packages have been lost.

/DEV..../STATUS/FLAGS/PACKETLOSSUDP

Properties: Read
Type: Integer (64 bit)
Unit: None

Flag indicating if udp packages have been lost.

/DEV..../STATUS/TIME

Properties: Read
Type: Integer (64 bit)
Unit: None

The current timestamp.

6.2.7. SYNTHESIZERS

/DEV..../SYNTHESIZERS/n/CENTERFREQ

Properties: Read, Write
Type: Double
Unit: Hz

The Center Frequency of the synthesizer.

6.2.8. 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 Reference clock has been locked on.
1 There was an error locking onto the reference clock signal.
2 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. When the frequency is changed, all the instruments connected with ZSync links will be disconnected. The connection should be re-established manually.

/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/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/INTERFACESPEED

Properties:	Read
Type:	String
Unit:	None

Speed of the currently active interface (USB only).

/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/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/OWNER

Properties: Read
Type: String
Unit: None

Returns the current owner of the device (IP).

/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/MAXDEMODRATE

Properties: Read
Type: Double
Unit: 1/s

The maximum demodulator rate that can be set. Only relevant for lock-in amplifiers.

/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/UPDATE

Properties:	Read, Write
Type:	Integer (64 bit)
Unit:	None

Requests update of the device firmware and bitstream from the dataserver.

Glossary

This glossary provides easy to understand descriptions for many terms related to measurement instrumentation including the abbreviations used inside this user manual.

A

A/D	Analog to Digital. See also ADC.
AC	Alternate Current
ADC	Analog to Digital Converter
AM	Amplitude Modulation
Amplitude Modulated AFM (AM-AFM)	AFM mode where the amplitude change between drive and measured signal encodes the topography or the measured AFM variable. See also Atomic Force Microscope.
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
Atomic Force Microscope (AFM)	Microscope that scans surfaces by means an oscillating mechanical structure (e.g. cantilever, tuning fork) whose oscillating tip gets so close to the surface to enter in interaction because of electrostatic, chemical, magnetic or other forces. With an AFM it is possible to produce images with atomic resolution. See also Amplitude Modulated AFM, Frequency Modulated AFM, Phase Modulated AFM.
AVAR	Allen Variance

B

Bandwidth (BW)	The signal bandwidth represents the highest frequency components of interest in a signal. For filters the signal bandwidth is the cut-off point, where the transfer function of a system shows 3 dB attenuation versus DC. In this context the bandwidth is a synonym of cut-off frequency $f_{cut-off}$ or 3dB frequency f_{-3dB}. The concept of bandwidth is used when the dynamic behavior of a signal is important or separation of different signals is required. In the context of a open-loop or closed-loop system, the bandwidth can be used to indicate the fastest speed of the system, or the highest signal update change rate that is possible with the system. Sometimes the term bandwidth is erroneously used as synonym of frequency range. See also Noise Equivalent Power Bandwidth.
BNC	Bayonet Neill-Concelman Connector

C

CF	Clock Fail (internal processor clock missing)

Common Mode Rejection Ratio (CMRR)	Specification of a differential amplifier (or other device) indicating the ability of an amplifier to obtain the difference between two inputs while rejecting the components that do not differ from the signal (common mode). A high CMRR is important in applications where the signal of interest is represented by a small voltage fluctuation superimposed on a (possibly large) voltage offset, or when relevant information is contained in the voltage difference between two signals. The simplest mathematical definition of common-mode rejection ratio is: CMRR = 20 * log(differential gain / common mode gain).
CSV	Comma Separated Values

D

D/A	Digital to Analog
DAC	Digital to Analog Converter
DC	Direct Current
DDS	Direct Digital Synthesis
DHCP	Dynamic Host Configuration Protocol
DIO	Digital Input/Output
DNS	Domain Name Server
DSP	Digital Signal Processor
DUT	Device Under Test
Dynamic Reserve (DR)	The measure of a lock-in amplifier's capability to withstand the disturbing signals and noise at non-reference frequencies, while maintaining the specified measurement accuracy within the signal bandwidth.

E

XML	Extensible Markup Language. See also XML.

F

FFT	Fast Fourier Transform
FIFO	First In First Out
FM	Frequency Modulation
Frequency Accuracy (FA)	Measure of an instrument's ability to faithfully indicate the correct frequency versus a traceable standard.
Frequency Modulated AFM (FM-AFM)	AFM mode where the frequency change between drive and measured signal encodes the topography or the measured AFM variable. See also Atomic Force Microscope.
Frequency Response Analyzer	Instrument capable to stimulate a device under test and plot the frequency response over a selectable frequency range with a fine granularity.

Frequency Sweeper	See also Frequency Response Analyzer.

G

Gain Phase Meter	See also Vector Network Analyzer.
GPIB	General Purpose Interface Bus
GUI	Graphical User Interface

I

I/O	Input / Output
Impedance Spectroscope (IS)	Instrument suited to stimulate a device under test and to measure the impedance (by means of a current measurement) at a selectable frequency and its amplitude and phase change over time. The output is both amplitude and phase information referred to the stimulus signal.
Input Amplitude Accuracy (IAA)	Measure of instrument's capability to faithfully indicate the signal amplitude at the input channel versus a traceable standard.
Input voltage noise	Total noise generated by the instrument and referred to the signal input, thus expressed as additional source of noise for the measured signal.
IP	Internet Protocol

L

LAN	Local Area Network
LED	Light Emitting Diode
Lock-in Amplifier (LI, LIA)	Instrument suited for the acquisition of small signals in noisy environments, or quickly changing signal with good signal to noise ratio - lock-in amplifiers recover the signal of interest knowing the frequency of the signal by demodulation with the suited reference frequency - the result of the demodulation are amplitude and phase of the signal compared to the reference: these are value pairs in the complex plane (X, Y), (R, Θ).

M

Media Access Control address (MAC address)	Refers to the unique identifier assigned to network adapters for physical network communication.
Multi-frequency (MF)	Refers to the simultaneous measurement of signals modulated at arbitrary frequencies. The objective of multi-frequency is to increase the information that can be derived from a measurement which is particularly important for one-time, non-repeating events, and to increase the speed of a measurement since different frequencies do not have to be applied one after the other. See also Multi-harmonic.
Multi-harmonic (MH)	Refers to the simultaneous measurement of modulated signals at various harmonic frequencies. The objective of multi-frequency is to increase the information that can be derived from a measurement which is particularly

	important for one-time, non-repeating events, and to increase the speed of a measurement since different frequencies do not have to be applied one after the other. See also Multi-frequency.

N

Noise Equivalent Power Bandwidth (NEPBW)	Effective bandwidth considering the area below the transfer function of a low-pass filter in the frequency spectrum. NEPBW is used when the amount of power within a certain bandwidth is important, such as noise measurements. This unit corresponds to a perfect filter with infinite steepness at the equivalent frequency. See also Bandwidth.
Nyquist Frequency (NF)	For sampled analog signals, the Nyquist frequency corresponds to two times the highest frequency component that is being correctly represented after the signal conversion.

O

Output Amplitude Accuracy (OAA)	Measure of an instrument's ability to faithfully output a set voltage at a given frequency versus a traceable standard.
OV	Over Volt (signal input saturation and clipping of signal)

P

PC	Personal Computer
PD	Phase Detector
Phase-locked Loop (PLL)	Electronic circuit that serves to track and control a defined frequency. For this purpose a copy of the external signal is generated such that it is in phase with the original signal, but with usually better spectral characteristics. It can act as frequency stabilization, frequency multiplication, or as frequency recovery. In both analog and digital implementations it consists of a phase detector, a loop filter, a controller, and an oscillator.
Phase modulation AFM (PM-AFM)	AFM mode where the phase between drive and measured signal encodes the topography or the measured AFM variable. See also Atomic Force Microscope.
PID	Proportional-Integral-Derivative
PL	Packet Loss (loss of packets of data between the instruments and the host computer)

R

RISC	Reduced Instruction Set Computer
Root Mean Square (RMS)	Statistical measure of the magnitude of a varying quantity. It is especially useful when variates are positive and negative, e.g., sinusoids, sawtooth, square waves. For a sine wave the following relation holds between the

	amplitude and the RMS value: $U_{RMS} = U_{PK} / \sqrt{2} = U_{PK} / 1.41$. The RMS is also called quadratic mean.
RT	Real-time

S

Scalar Network Analyzer (SNA)	Instrument that measures the voltage of an analog input signal providing just the amplitude (gain) information. See also Spectrum Analyzer, Vector Network Analyzer.
SL	Sample Loss (loss of samples between the instrument and the host computer)
Spectrum Analyzer (SA)	Instrument that measures the voltage of an analog input signal providing just the amplitude (gain) information over a defined spectrum. See also Scalar Network Analyzer.
SSH	Secure Shell

T

TC	Time Constant
TCP/IP	Transmission Control Protocol / Internet Protocol
Thread	An independent sequence of instructions to be executed by a processor.
Total Harmonic Distortion (THD)	Measure of the non-linearity of signal channels (input and output)
TTL	Transistor to Transistor Logic level

U

UHF	Ultra-High Frequency
UHS	Ultra-High Stability
USB	Universal Serial Bus

V

VCO	Voltage Controlled Oscillator
Vector Network Analyzer (VCO)	Instrument that measures the network parameters of electrical networks, commonly expressed as s-parameters. For this purpose it measures the voltage of an input signal providing both amplitude (gain) and phase information. For this characteristic an older name was gain phase meter. See also Gain Phase Meter, Scalar Network Analyzer.

X

XML	Extensible Markup Language: Markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.

Z

ZCtrl	Zurich Instruments Control bus
ZoomFFT	This technique performs FFT processing on demodulated samples, for instance after a lock-in amplifier. Since the resolution of an FFT depends on the number of point acquired and the spanned time (not the sample rate), it is possible to obtain very highly resolution spectral analysis.
ZSync	Zurich Instruments Synchronization bus

Index