ngscopeclient Operator Manual

Andrew D. Zonenberg

September 13, 2023

Contents

Chapter 1

Introduction

1.1 Introduction

ngscopeclient is a high performance, GPU accelerated remote user interface, signal processing, protocol analysis, and automation tool for test and measurement equipment. It runs on all major operating systems and can interoperate with a broad and continuously growing range of T&M products from many vendors.

As of this writing, ngscopeclient is under active development but has not had a formal v0.1 release and should be considered alpha quality.

This is free software: you are free to change and redistribute it. There is NO WARRANTY, to the extent permitted by law.

1.2 Key Concepts

1.2.1 User Interface

Most UI elements can be interacted with by left clicking (select), left dragging (move), using the scroll wheel (zoom), double clicking (open properties dialog), or right clicking (context menu). Hovering the mouse over a main window UI element, or a (?) help marker in a dialog box, displays a tooltip explaining the purpose of the element

Most text fields allow SI prefixes for scaling values (mV, μs, GHz, etc). Lowercase 'u' is interpreted as "micro", equivalent to the Greek letter μ. The unit is automatically added if not specified, for example typing "2.4G" in a frequency input field will be interpreted as meaning 2.4 GHz.

1.2.2 Design Philosophy

Users familiar with conventional benchtop oscilloscopes will notice some important distinctions between ngscopeclient and classical DSO user interfaces. While there is an initial learning curve getting used to the different ways of doing things, these changes allow for greater productivity and more complex analysis.

Legacy DSO user interfaces largely still imitate the front panel controls of analog CRT instruments dating back to the mid 1940s. A single view of each waveform shows the entire acquisition on a grid with a fixed number of divisions (emulating an etched graticule on a CRT) and both time and voltage scales are defined in terms of these divisions. While more recent DSOs do allow math functions, protocol decodes, zooms, and so on, this archaic concept has remained.

In ngscopeclient, the acquisition record length is completely decoupled from the X axis scale of the viewport, and there is no concept of a “zoom" waveform or measuring time in “divisions". Arbitrarily many views of a channel may be created, and each may be scaled and zoomed independently. Acquisition record length and duration are controlled separately, from the timebase properties dialog.

Similarly, vertical scale for waveforms is defined in terms of full-scale range, a far more intuitive and useful metric than arbitrary “divisions". While horizontal grid lines are still displayed in waveform views for convenience, their number, spacing, and locations may change. Tall plots will have more scale divisions than short ones, and the divisions are always located at round numbers even if this requires the grid to not be centered in the plot (Fig. 1.1)

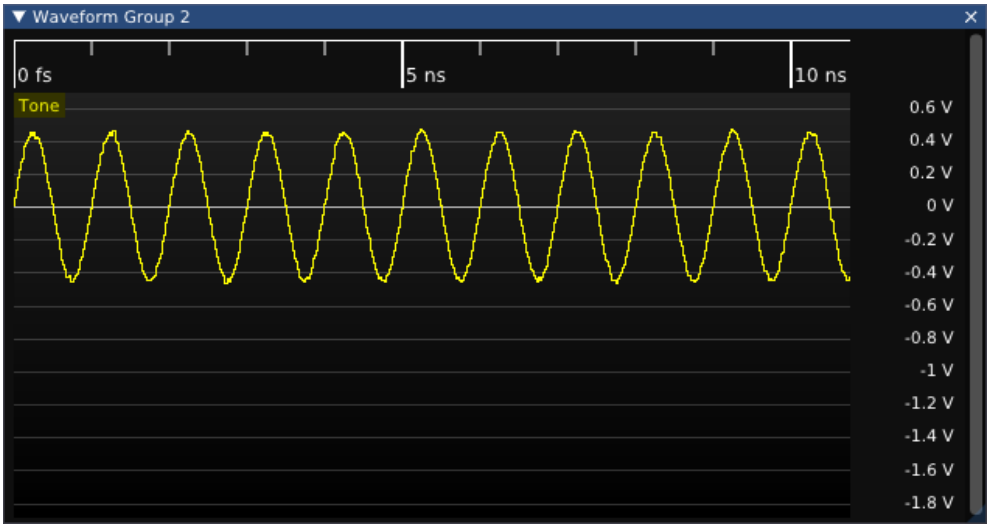

Figure 1.1: Example waveform showing off-center grid and round-numbered grid lines

Rather than optimizing for a touch screen (as is common for benchtop oscilloscopes), ngscopeclient’s UI is heavily mouse driven and context based. Space used by always-visible buttons, sliders, etc is kept to a minimum in order to keep as much screen real estate as possible usable for waveform display. Additional controls are displayed in menus or pop-up dialogs which can be closed, moved out of view, or docked as needed.

1.2.3 Terminology

The overall software package consists of *ngscopeclient* (graphical user interface frontend), *libscopehal* (C++ library for core APIs and instrument drivers), and *libscopeprotocols* (filter graph blocks). End users will normally use ngscopeclient, however it is possible to interface with libscopehal and libscopeprotocols directly from C++ code for writing low level test automation tools or even a fully custom application-specific user interface.

Data consists of two fundamental types: *scalars* and *waveforms*. A scalar is a single numeric value with an associated unit, for example “500 mV". A waveform is a sequence of *samples* plotted against another quantity, for example voltage versus time for an oscilloscope waveform or amplitude versus frequency for a spectrum analyzer waveform.

Samples may be of arbitrary type (analog value, digital bit, SPI bus event, etc.), but all samples in a single waveform must be of the same data type. Waveforms may be either *uniform* (sampled at constant rate with no gaps between samples) or *sparse* (sampled at arbitrary intervals, possibly with gaps between samples).

An *instrument* is a physical piece of hardware [1] which can be remote controlled and interacted with. The connection between ngscopeclient and an instrument is provided by a *transport*, such as a USBTMC interface, a GPIB data stream, or a TCP socket. A *driver* is a software component, either supplied as part of the libscopehal core or a third party plugin, which controls an instrument.

Each instrument has one or more[2] *channels*. A channel corresponds to a single logical "piece" of an instrument and may consist of one or more physical connectors: a typical oscilloscope channel has a single BNC input while a typical power supply output has two banana jacks.

Each channel may provide features associated with one or more instrument *types*, and not all channels on an instrument are guaranteed to be the same type(s). For example, an oscilloscope may consist of several channels providing both waveform acquisition (oscilloscope) and scalar acquisition (multimeter) capabilities, one channel providing only trigger input capability, and one channel providing function generator output capability.

All channels, triggers, and math / protocol decode blocks are considered *nodes* within the *filter graph*. The filter graph is a directed acyclic graph (a set of nodes and connections between them, with no loops permitted) connecting all of the various data inputs and outputs of the experimental setup together.

Each node may have zero or more *inputs*, of either scalar or waveform type, and zero or more output *streams*. A stream is a data source which may or may not have an associated scalar or waveform value; for example a math block with missing inputs or an instrument which has not yet triggered do not have a meaningful value. A typical oscilloscope channel might have one waveform output stream, while a typical power supply channel might have two scalar output streams for measured current and voltage. A sink block for writing a waveform to a CSV file would have one input for each column in the generated file.

Instrument hardware limitations or the particular math/decode block's design will impose various restrictions on legal connections in the filter graph. For example, a trigger can normally only accept signals from hardware input channels on the same instrument. An FFT filter can only accept uniformly sampled analog waveforms. A UART protocol decode can only accept digital waveforms, so analog waveforms must be converted to digital by a thresholding filter before they can be decoded.

1.3 Revision History

- September 13, 2023: [in progress] Initial draft

[1] Or a simulated mock-up of one, such as the "demo" oscilloscope driver used for testing

[2] Zero channels is legal in the API, however such an instrument would be of little practical use!

Chapter 2

Legal Notices

2.1 Introduction

ngscopeclient, libscopehal, and the remainder of the project are all released under the 3-clause BSD license (reproduced below). This is a permissive license, explicitly chosen to encourage integration with third-party open source and commercial projects.

2.2 License Agreement

2.3 Trademarks

This document frequently mentions the names of various test equipment vendors and products in order to discuss ngscopeclient's compatibility with said products. The reader should assume that these are all trademarks of their respective owners.

2.4 Third Party Licenses

TODO: go through full dependency list and update this

- Dear ImGui (static, MIT license)
- FFTS (shared, BSD-3)
- imgui-node-editor (static, MIT license)
- liblxi (shared, BSD-3/EPICS)
- vkFFT (static, MIT license)
- yaml-cpp (shared, MIT license)

2.4.1 avx_mathfun.h (zlib license)

AVX implementation of sin, cos, sincos, exp and log

Based on "sse_mathfun.h", by Julien Pommier http://gruntthepeon.free.fr/ssemath/

Copyright (C) 2012 Giovanni Garberoglio Interdisciplinary Laboratory for Computational Science (LISC) Fondazione Bruno Kessler and University of Trento via Sommarive, 18 I-38123 Trento (Italy)

This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.

Chapter 3

Getting Started

3.1 Documentation Conventions

Items to be selected from a menu are displayed in `monospace font`.

Multilevel menu paths are separated by a / character. For example, `Attenuation / 1x` means to open the `Attenuation` submenu and select the `1x` item.

If there are multiple options for a menu or configuration option, they are displayed in square brackets and separated by a | character. For example, `Move waveform to / Waveform Group [1|2]` means to select either `Waveform Group 1` or `Waveform Group 2` from the `Move waveform to` menu.

This project is under active development and is not anywhere near feature complete! As a result, this document is likely to refer to active bug or feature request tickets on the GitHub issue trackers. Issues are referenced as repository:ticket, for example scopehal-apps:3.

3.2 Host System Requirements

The majority of development is performed on Linux operating systems (primarily Debian and Arch) so this is the most well tested platform, however Windows and Mac OS are also supported.

Any 64-bit Intel or AMD processor, or Apple Silicon Mac, should be able to run ngscopeclient. If AVX2 and/or AVX512F support is present ngscopeclient will use special optimized versions of some signal processing functions, however neither instruction set is required. Other (non Apple Silicon) ARM64 platforms may work if a compatible GPU is available, but have not been tested. 32-bit platforms are not supported due to the significant RAM requirements.

A mouse with scroll wheel, or touchpad with scroll gesture support, is mandatory to enable full use of the UI. We may explore alternative input methods for some UI elements in the future.

Any GPU with Vulkan support should be able to run ngscopeclient, however Vulkan 1.2 will deliver better performance. The minimum supported GPUs are:

- AMD: GCN based (Radeon HD 7000 and newer, January 2012)
- Apple: all Apple Silicon devices (M1 and newer)
- Intel: Iris Plus 540 or HD Graphics 520 (Skylake, August 2015)
- NVIDIA: Maxwell architecture (GeForce GTX 700 series and newer, February 2014)

The minimum RAM requirement to actually launch ngscopeclient is relatively small, however actual memory consumption is heavily dependent on workload and can easily reach into the tens of gigabytes when doing complex analysis on many channels with deep history.

Typical RAM consumption examples:

- Default configuration with demo scope (4 channels 100K points, 10 waveforms of history, no analysis): 250 MB
- 4M point live streaming with 10 waveforms of history, eye pattern, 8B/10B decode, and jitter histogram: 650 MB
- Single 512M point waveform, no analysis or history: 2.1 GB
- 512M point P/N channel waveforms with CDR and eye pattern, no history: 8.3 GB

Large amounts of GPU RAM are required for working with deep waveforms, especially if you intend to perform complex analysis on them. Analog waveforms are stored in 32-bit floating point format internally, so a single 256 megapoint waveform will consume 1GB of GPU memory. Intermediate results in multi-step filter pipelines require GPU memory as well, even if not displayed.

3.3 Instrument Support

ngscopeclient uses the libscopehal library to communicate with instruments, so any libscopehal-compatible hardware should work with ngscopeclient. See the Oscilloscope Drivers section for more details on which hardware is supported and how to configure specific drivers.

3.4 Compilation

ngscopeclient can be compiled on Linux, macOS, and Windows, but the specific steps that have to be taken differ quite a lot between these the three.

TODO: rewrite and simplify this section once glscopeclient is deprecated and we've removed all remaining vestiges of GTK dependencies

TODO: update URLs for most recent tested Vulkan SDK revision

3.4.1 Linux

1. Install dependencies.

   On Debian/Ubuntu:

   ```
   sudo apt install build-essential cmake pkg-config libglm-dev \
       libgtkmm-3.0-dev libsigc++-2.0-dev libyaml-cpp-dev \
       liblxi-dev texlive texlive-fonts-extra libglew-dev \
       catch2 libvulkan-dev glslang-dev libglfw3-dev
   ```

   On Fedora(this section is out of date):

   ```
   sudo dnf install gtkmm30-devel cmake pkg-config glm-devel \
       texlive libyaml-devel yaml-cpp-devel glew-devel \
       catch-devel vulkan-devel
   ```

If you are using an older stable release (such as Debian Buster), you may need to install catch2 from source (https://github.com/catchorg/Catch2). Alternatively, you may pass -DBUILD_TESTING=OFF to CMake to disable unit testing.

2. Install FFTS library:

```
cd ~
git clone https://github.com/anthonix/ffts.git
cd ffts
mkdir build
cd build
cmake .. -DENABLE_SHARED=ON -DCMAKE_INSTALL_PREFIX=/usr
make -j4
sudo make install
```

3. Install Vulkan SDK:

```
cd ~
mkdir vulkan
cd vulkan
wget https://sdk.lunarg.com/sdk/download/1.3.224.1/linux/vulkansdk-
    linux-x86_64-1.3.224.1.tar.gz
tar xf vulkansdk-linux-x86_64-1.3.224.1.tar.gz
rm -f vulkansdk-linux-x86_64-1.3.224.1.tar.gz
export VULKAN_SDK=~/vulkan/1.3.224.1/x86_64
sudo cp -r $VULKAN_SDK/include/vulkan/ /usr/local/include/
sudo cp -P $VULKAN_SDK/lib/libvulkan.so* /usr/local/lib/
sudo cp $VULKAN_SDK/lib/libVkLayer_*.so /usr/local/lib/
sudo mkdir -p /usr/local/share/vulkan/explicit_layer.d
sudo cp $VULKAN_SDK/etc/vulkan/explicit_layer.d/VkLayer_*.json /usr/
    local/share/vulkan/explicit_layer.d
sudo ldconfig
```

4. Build scopehal and scopehal-apps:

```
export VULKAN_SDK=~/vulkan/1.3.224.1/x86_64
export PATH=$VULKAN_SDK/bin:$PATH
export LD_LIBRARY_PATH=$VULKAN_SDK/lib${LD_LIBRARY_PATH:+:
    $LD_LIBRARY_PATH}
export VK_LAYER_PATH=$VULKAN_SDK/etc/vulkan/explicit_layer.d

cd ~
git clone -{}-recursive https://github.com/glscopeclient/scopehal-apps
    .git
cd scopehal-apps
mkdir build
cd build
cmake ../ -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=/usr
make -j4
```

5. Install scopehal and scopehal-apps:

```
cd ~/scopehal-apps/build
sudo make install
```

3.4.2 macOS

1. Install dependencies.

   With Homebrew (brew.sh):

```
brew install pkg-config gtk+3 gtkmm3 glfw cmake yaml-cpp glew catch2
    libomp
```

2. Install Vulkan SDK:

   Download and install the Vulkan SDK from sdk.lunarg.com/sdk/download/1.3.231.1/mac/vulkansdk-macos-1.3.231.1.dmg.

3. Build scopehal and scopehal-apps:

```
export VULKAN_SDK=~/VulkanSDK/1.3.231.1/macOS
export PATH=${PATH}:${VULKAN_SDK}/bin:/opt/homebrew/bin
cd ~
git clone -{}-recursive https://github.com/glscopeclient/scopehal-apps
    .git
cd scopehal-apps
mkdir build
cd build
cmake ../ -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=/usr -
    DCMAKE_PREFIX_PATH="/opt/homebrew;/opt/homebrew/opt/libomp"
make -j4
```

4. Install scopehal and scopehal-apps:

   Installation on macOS is not yet complete. The binaries can be found in the build directory, such as ngscopeclient in $HOME/scopehal-apps/build/src/ngscopeclient.

3.4.3 Windows

On Windows, we make use of the MSYS2 development environment, which gives us access to the MingGW-w64 toolchain. Since this toolchain allows ngscopeclient to be compiled as a native Windows application, the project might be run outside of MSYS2.

Installing from the package manager

```
pacman -Sy
pacman -S mingw-w64-x86_64-scopehal-apps
```

See also `mingw-w64-x86_64-eda`.

Building from source

1. Download and install MSYS2. You can download it from msys2.org or github.com/msys2/msys2-installer/releases
   It is of utmost importance that **all** steps outlined on the website are followed precisely, even if they might seem unnecessary. This will avoid lots of hard to diagnose problems later on in the build.

   All following steps are to be done in a MinGW64 shell (**not** in a MSYS, MINGW32, CLANG64, CLANG32 or UCRT64 shell, which also get installed by the MSYS2 installer).

2. Install WiX Toolset v3.11 (required to build the Windows x64 MSI) You shall download and install WiX Toolset v3.11 in

   `C:\Program Files (x86)\WiX Toolset v3.11`

https://wixtoolset.org/docs/wix3/

3. Install git and the toolchain:

```
pacman -S git wget mingw-w64-x86_64-cmake mingw-w64-x86_64-toolchain
```

4. Build glslang tags/sdk-1.3.224.1:

   Launch MSYS2 or MINGW64 as Administrator only for this step (it is mandatory to do the install in default path C:
   VulkanSDK ...)

```
# Windows mingw64 glslang build (as it is not fully integrated in
    VulkanSDK-1.3.224.1 for Windows and built with Visual Studio 2017)
cd ~
git clone https://github.com/KhronosGroup/glslang.git
cd glslang
git checkout tags/sdk-1.3.224.1
git clone https://github.com/google/googletest.git External/googletest
cd External/googletest
git checkout 0c400f67fcf305869c5fb113dd296eca266c9725
cd ../..
./update_glslang_sources.py

SOURCE_DIR=~/glslang
BUILD_DIR=$SOURCE_DIR/build
mkdir -p $BUILD_DIR
cd $BUILD_DIR
cmake -DCMAKE_BUILD_TYPE=Release -G"MinGW Makefiles" $SOURCE_DIR -
    DCMAKE_INSTALL_PREFIX="$(pwd)/install"
cmake -{}-build . -{}-config Release -{}-target install
```

5. Install Vulkan SDK:

   Launch MSYS2 or MINGW64 as Administrator only for this step (it is mandatory to do the install in default path C:
   VulkanSDK ...)

```
cd ~
wget https://sdk.lunarg.com/sdk/download/1.3.224.1/windows/VulkanSDK
    -1.3.224.1-Installer.exe
./VulkanSDK-1.3.224.1-Installer.exe -{}-accept-licenses -{}-default-
    answer -{}-confirm-command install
rm -f VulkanSDK-1.3.224.1-Installer.exe
```

6. Check out the code

```
cd ~
git clone -{}-recursive https://github.com/glscopeclient/scopehal-apps
```

7. Execute makepkg-mingw in subdir MSYS2:

```
cd ~/scopehal-apps/msys2
export VK_SDK_PATH=/c/VulkanSDK/1.3.224.1
export PATH=$VK_SDK_PATH/Bin:$PATH
export VULKAN_SDK=$VK_SDK_PATH
export GLSLANG_BUILD_PATH=~/glslang/build/install

MINGW_ARCH=mingw64 makepkg-mingw -{}-noconfirm -{}-noprogressbar -sCLf
```

The unit tests will not be run as part of this build - if you would like to run them, you will need to provide catch2 (https://github.com/catchorg/Catch2), and remove the -DBUILD_TESTING=OFF flag from the PKGBUILD recipe in subdir msys2.

8. Installing, copying binaries and running ngscopeclient.

   Since ngscopeclient is built using the MinGW toolchain, it depends on a rather large number of dynamic libraries. The recommended procedure is to install the package generated by makepkg-mingw on a MinGW64 shell:

   ```
   cd ~
   cd msys2
   pacman -U *.zst
   ```

   This is equivalent to the package installed through `pacman -S`, but it's built from the checked out commit, instead of the pinned version available from MSYS2 repositories.

   The `*.zst` package includes metadata about the dependencies. Therefore, when installed through `pacman`, those will be installed automatically. However, some users might want to use ngscopeclient outside of MSYS2. In those cases, it needs to be installed first, and then a tarball/zipfile can be created by collecting all the dependencies. This last approach is not officially supported yet.

3.5 Running ngscopeclient

When running ngscopeclient with no arguments, an empty session (Fig. 3.1) is created. To perform useful work, you can:

- Open a saved session and reconnect to the instruments (`File | Open Online`)
- Open a saved session without reconnecting to the instruments (`File | Open Offline`)
- Open a recently used session (`File | Recent Files`)
- Import waveforms from a third party file format(`Add | Import`)
- Connect to an instrument (`Add | Oscilloscope`, `Add | Multimeter`, etc.)
- Generate a synthetic waveform (`Add | Generate`)

3.5.1 Console verbosity arguments

ngscopeclient takes standard liblogtools arguments for controlling console debug verbosity.

If no verbosity level is specified, the default is "notice" (3). (We suggest using `--debug` for routine use until the v1.0 release to aid in troubleshooting.)

- `--debug`
  Sets the verbosity level to "debug" (5).
- `-l [file]`, `--logfile [file]`
  Writes a copy of all log messages to `file`. This is preferred over simply redirecting output with pipes, as console escape sequences are stripped from the file log output.
- `-L [file]`, `--logfile-lines [file]`
  Same as `--logfile` except line buffering is turned on.

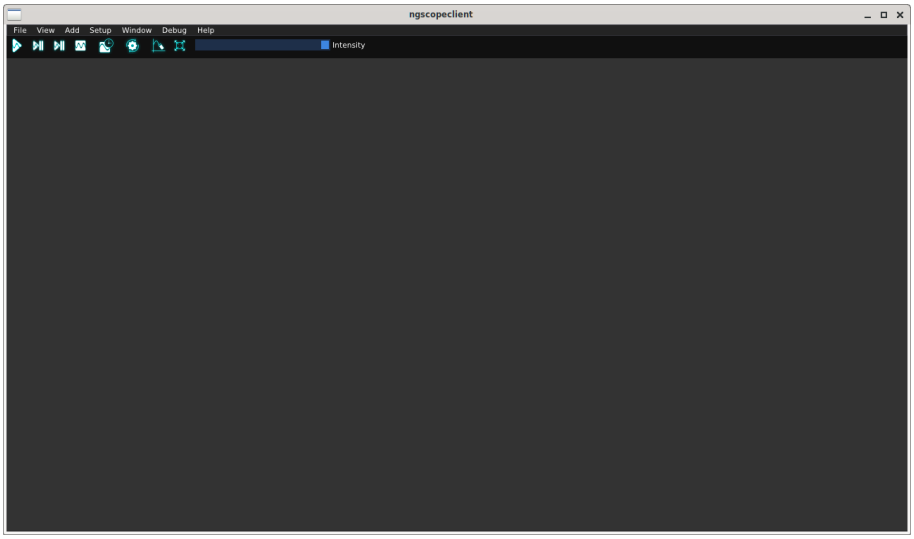

Figure 3.1: Empty ngscopeclient session

- `-q`, `--quiet`
  Reduces the verbosity level by one. Can be specified more than once to lower verbosity by several steps.

- `--trace [class]`, `--trace [class::function]`
  Enables extra debug output from the class `class` or the function `class::function`. Has no effect unless `--debug` is also specified.

- `--stdout-only`
  Sends all logging output to stdout. By default, error (level 1) and warning (level 2) messages go to stderr.

- `--verbose`
  Sets the verbosity level to “verbose" (4).

Chapter 4

Main Window

The only fixed UI elements in ngscopeclient are the main menu and toolbar at the top of the window. All remaining space may be filled with waveform plots, properties dialogs, protocol analyzers, and other dockable windows as required for a given experimental setup. This flexibility allows almost the entire screen to be dedicated to waveform views, or more space allocated to controls and protocol decodes.

4.1 Menu

4.1.1 File

This menu contains commands for saving and loading session files.

- `Open Online...`
  Loads a session file and reconnects to the instrument(s) to continue existing work. Settings from the saved session will be applied and overwrite the current channel and timebase configuration of the instrument, if different.
- `Open Offline...`
  Loads a session file in offline mode, allowing you to work with saved waveform data without connecting to the instrument(s) the data was captured from.
- `Recent Files`
  Displays a list of recently accessed session files and allows them to be opened online or offline.
- `Save`
  Saves UI configuration and waveform data (including history) to a session file for future use.

  A session consists of a YAML file called *filename*.scopesession containing instrument and UI configuration, as well as a directory called *filename*_data which contains waveform metadata and sample values for all enabled instrument channels, including history.

  Note that both the .scopesession and the _data directory must be copied if moving the session to a new location in order to preserve waveform data. If you only wish to restore the filter graph and UI configuration without waveform content, the _data directory is not required.
- `Save As...`
  Saves the session to a new file, rather than the current one.
- `Close`
  Close the current session without exiting ngscopeclient.

- `Quit`
  Exits the application

4.1.2 View

- `Fullscreen`
  Toggles full-screen mode
- `Persistence Setup`
  Opens the Persistence Setup dialog, allowing you to control the decay coefficient for persistence maps.

4.1.3 Add

This menu allows new waveforms views or instrument connections to be created.

- `Load`
  Connect to a new, or recently used, electronic load
- `Generator`
  Connect to a new, or recently used, function generator
- `Multimeter`
  Connect to a new, or recently used, multimeter
- `Oscilloscope`
  Connect to a new, or recently used, oscilloscope
- `Power Supply`
  Connect to a new, or recently used, power supply
- `RF Generator`
  Connect to a new, or recently used, RF signal generator
- `Channels`
  Displays a list of filters and instrument channels which can be opened in a new waveform view
- `Generate`
  Allows synthetic waveforms to be generated for testing, simulation, and channel design applications
- `Import`
  Allows waveforms to be loaded from external data files in various interchange formats

4.1.4 Setup

- `Timebase...`
  Opens the Timebase Properties dialog, allowing sample rate and memory depth of each connected oscilloscope to be adjusted
- `Trigger...`
  Configures trigger settings
- `Preferences...`
  Opens the preferences dialog

4.1.5 Window

This menu provides access to various utility windows.

- `Analyzer`
  Opens protocol analyzer dialogs for active protocol decodes
- `Generator`
  Opens the properties dialog for a currently connected function generator
- `Multimeter`
  Opens the properties dialog for a currently connected multimeter
- `SCPI Console`
  Opens a console window allowing you to send raw SCPI commands to a currently connected instrument.

  This is a low level debug tool primarily intended for use by driver developers. The console is interlocked with background threads polling the instrument, so that replies to commands typed in the console will not be mixed with replies which the instrument driver is expecting to its own commands. However, commands sent in the console will bypass any caching in the driver and can easily lead to the driver and instrument firmware states becoming mutually inconsistent.
- `Log Viewer`
  Opens a console window allowing you to see debug log messages generated by the application. This is the same log stream which is normally written to stdout, but this dialog allows it to be accessed even when the application is not launched from a shell session.
- `Measurements`
  Opens the Measurements window, displaying scalar-valued measurements coming from instrument channels or filter blocks.
- `Performance Metrics`

  Opens the Performance Metrics window, which provides access to debug information which can be helpful when debugging slow application performance, optimizing the code, or benchmarking instruments.

  Three categories of information are displayed:

  - Rendering: render loop framerate, monitor refresh rate, total time spent last frame in the rasterization and tone mapping shaders, and the number of vertices and indices drawn as Vulkan geometry. Note that waveforms are drawn by a compute shader and do not contribute towads the vertex/index totals, other than a single textured rectangle used for displaying the shader output.
  - Filter graph: number of filter blocks in the current graph, and run time for the most recent evaluation of the filter graph.
  - Acquisition: streaming data rate, in waveforms per second.
- `History`

  Opens the History dialog, which allows access to a rolling buffer of recently acquired waveforms.
- `Filter Graph`
  Opens the filter graph editor (see Chapter 8)

4.1.6 Debug

Provides access to GUI toolkit test dialogs and other features intended only for developers.

4.1.7 Help

Nothing here yet, we should add at least an About dialog at some point...

Chapter 5

Waveform Groups

A waveform group is a collection of one or more waveform views stacked vertically under a common timeline. All waveform views within a group share the same timeline and vertical cursor(s), but may have independent vertical range and offset settings.

When a new oscilloscope is added to an empty ngscopeclient session, all enabled channels on the attached instrument(s) are displayed in a single waveform group (Figure 5.1). If no channels are enabled at connection time, the first channel will be enabled and displayed.

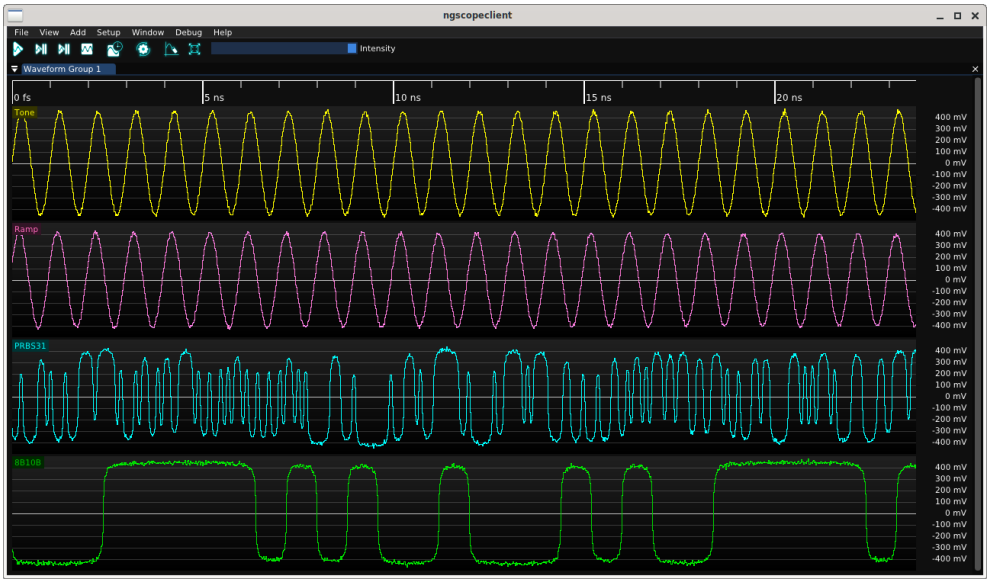

Figure 5.1: Top level ngscopeclient window with a single waveform group

As you add protocol decodes or look at different parts of a waveform, it may be helpful to create additional waveform groups. Typical reasons for creating additional groups include:

- Zooming into one set of signals to see detail on short time scales while maintaining a high level overview of others
- Viewing signals with incompatible horizontal units. For example, a FFT has horizontal units of frequency while an analog waveform has horizontal units of time. Eye patterns also have horizontal units of time, but are always displayed as two UIs wide and cannot be zoomed.

5.1 Managing Groups

New waveform groups are automatically created when adding a channel which is not compatible with any existing group. For example, if your session has a single group containing time-domain waveforms, adding a FFT filter block will result in a new waveform group being created to contain the FFT. Additional frequency-domain waveforms will then be added to this group by default.

A new group may also be created at any time by clicking on a channel name and dragging it to the top, bottom, left, or right edge of an existing group. An overlay (Fig. 5.2) will be displayed showing the resulting split. For example, dropping the channel on the right side of the window produces the layout shown in Fig. 5.3.

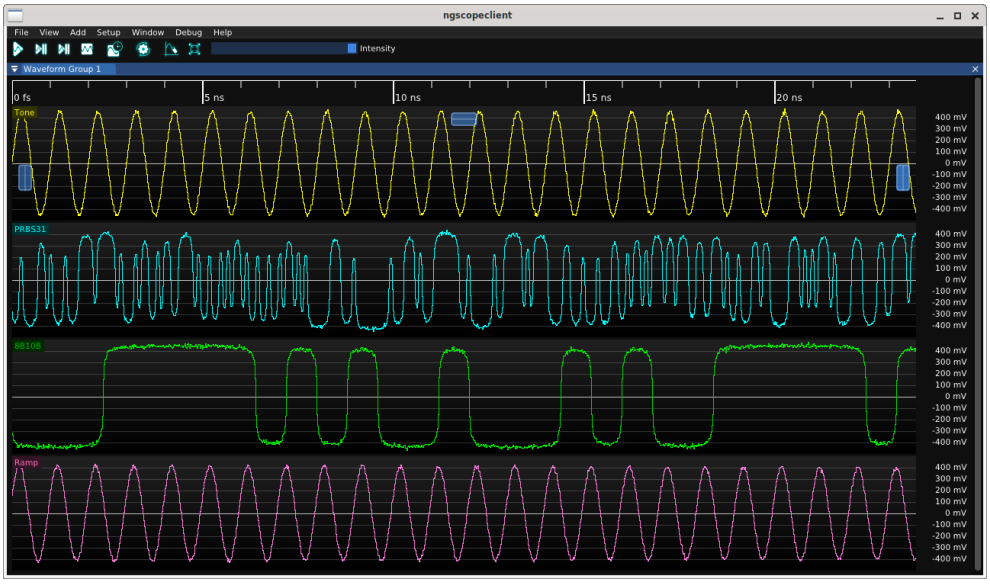

Figure 5.2: Overlays showing drag-and-drop locations for splitting waveform groups

Figure 5.3: Result of dropping a waveform to the right side split area

Waveform groups may be resized arbitrarily by dragging the separator between them. The title bar of a group may also be dragged, allowing the entire group to be undocked as a floating window. Floating windows can be re-docked by dragging the title bar back into the main ngscopeclient

window (or another floating window), creating new tabs or splitting existing groups as desired (Fig. 5.4).

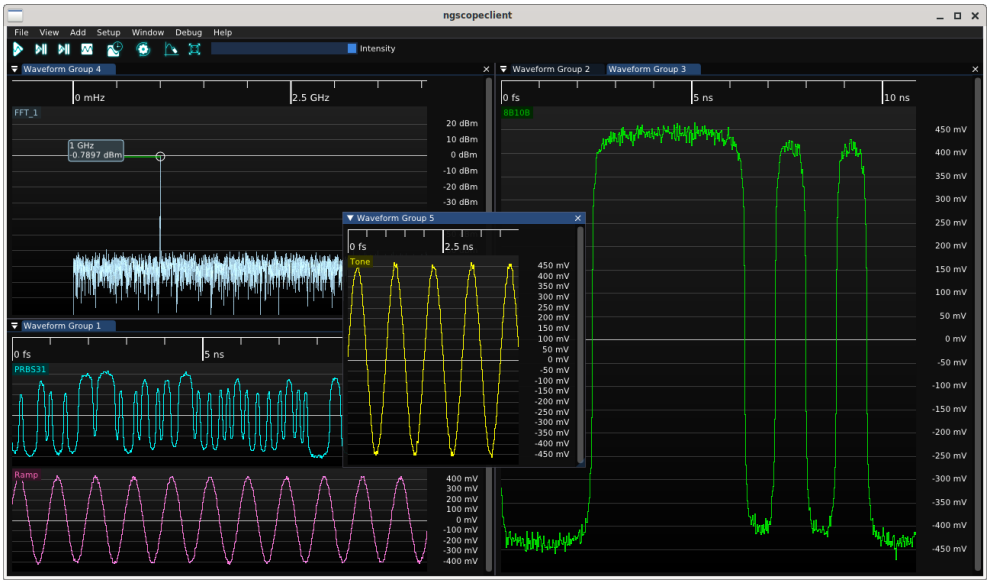

Figure 5.4: Example of complex window layout with multiple tabs, splitters between docked waveform groups, and a group in a floating window

Chapter 6

Waveform Views

A waveform view is a 2D graph of a signal or protocol decode within a waveform group.

Arbitrarily many channels of data may be displayed within a single view, however all analog channels within a single view share the same Y axis unit, gain, and offset. Digital channels and protocol decodes can be overlaid on analog waveforms or displayed in their own dedicated views.

2D density plots, such as eye patterns, spectrograms, and waterfall plots, cannot share a waveform view with any other channel.

6.1 Navigation

Scrolling with the mouse wheel adjusts the horizontal scale of the current waveform group, zooming in or out centered on the position of the mouse cursor.

6.2 Plot Area

The plot area shows the waveform being displayed. The horizontal grid lines line up with the voltage scale markings on the Y axis. If the plot area includes Y=0, the grid line for zero is slightly brighter.

The waveform is drawn as a semi-transparent line so that when zoomed out, the density of voltage at various points in the graph may be seen as lighter or darker areas. This is referred to as “intensity grading".

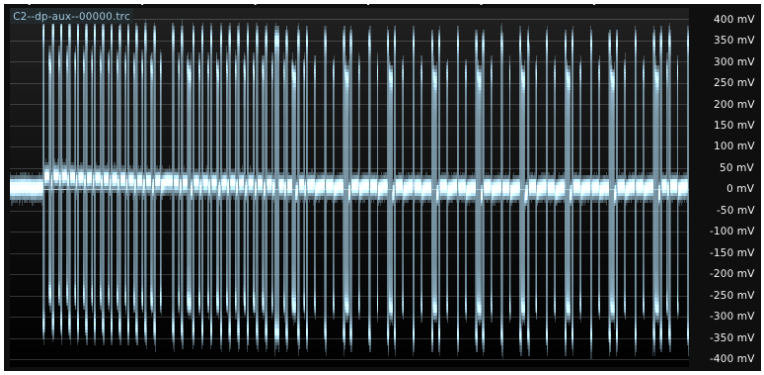

Figure 6.1: Intensity-graded waveform

6.3 Y Axis Scale

Each waveform view has its own Y axis scale, which is locked to the ADC range of the instrument.

Channel gain may be configured by scrolling with the mouse wheel, and offset may be adjusted by dragging the scale with the left mouse button. Pressing the middle mouse button on the Y axis will auto-scale the vertical gain and offset to show the full span of all channels in the view with 5% of vertical margin.

If a left-pointing arrow (as seen in Fig. 6.2) is visible, one of the channels in the view is selected as a trigger source. Click on the arrow and drag up or down to select the trigger level. Some trigger types, such as window triggers, have two arrows for upper and lower levels.

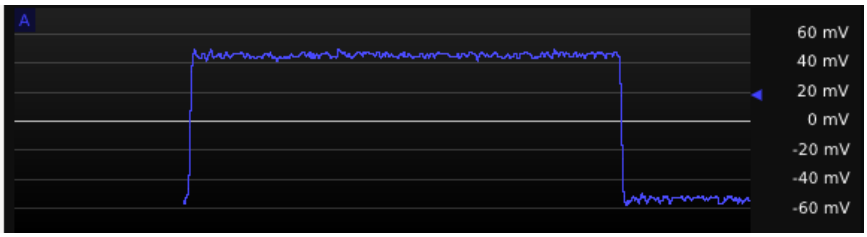

Figure 6.2: Waveform view showing trigger arrow on Y axis

6.4 Channel Label

The top left corner of each waveform view contains a legend with a label for each channel being displayed in the view.

Mousing over the channel name displays a tooltip (Fig. 6.3) with some helpful information about the waveform. The exact information displayed in the tooltip depends on the type of data being displayed, for example analog waveforms display sample rate and record length while eye patterns display the number of integrated UIs.

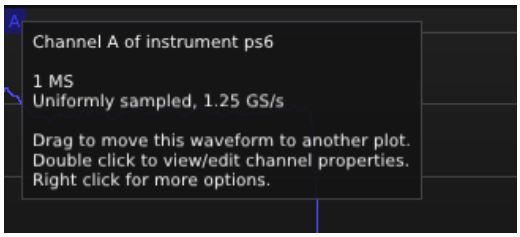

Figure 6.3: Example tooltip on channel label

The label may be dragged with the left mouse button to move the waveform to a different location. Dragging to the left or right edge of a waveform view, or the top or bottom edge of the topmost or bottommost waveform in a group, will split the group. Dragging to the left half of another waveform view, whether in the same group or a different group, moves the channel to that view. Dragging to the right half of the view adds a new view within the same group containing only the dragged waveform.

Double-clicking the label opens the channel properties dialog (Fig. 6.4). As with all dialogs in ngscopeclient, the properties dialog may be left in the default floating state or docked.

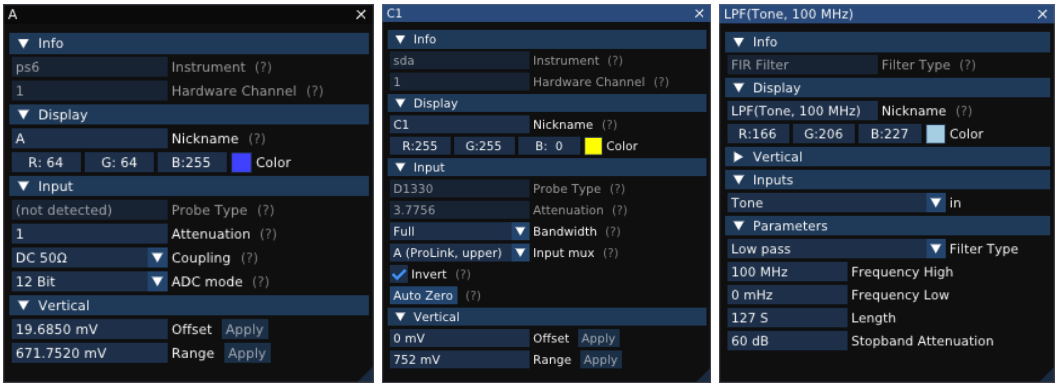

Figure 6.4: Example of properties dialogs for three different channels

The properties dialog will always contain an editable nickname for the channel, a color chooser, and some basic information about the instrument channel or filter block sourcing the data. Additional settings may be available but will vary depending on the type of instrument or filter. In Fig. 6.4, the left dialog shows a direct coaxial input to a Pico PicoScope 6824E, which has variable ADC resolution. The center dialog shows an active differential probe with auto-zero capability, connected to a Teledyne LeCroy SDA816Zi-A which has a mux for selecting between two input connectors for each channel. The right dialog shows a FIR filter with several configurable settings.

Right clicking on the label opens a context menu. The context menu allows setting of persistence mode, deleting the waveform, and creating new filter blocks or protocol decodes with the selected waveform as an input.

6.5 Cursors and Markers

Cursors are movable annotations which can be used to temporarily mark points of interest in a waveform and examine data values. Markers are similar to cursors but intended for long-term marking of specific points in a single acquisition and do not provide readout functionality.

6.5.1 Vertical Cursors

A vertical cursor describes a point in time *relative to the start of the acquisition.* When new waveforms are acquired, the cursor remains at the same offset in the new waveform. When the view is panned horizontally, the cursor scrolls with the waveform and remains at the same point in the waveform.

To add a vertical cursor (Fig. 6.5), right click in the view and select a single or double cursor from the `Cursors | X Axis` menu.

Vertical cursors are attached to a waveform group and will span all views within the group. Multiple groups may have independent vertical cursors active simultaneously.

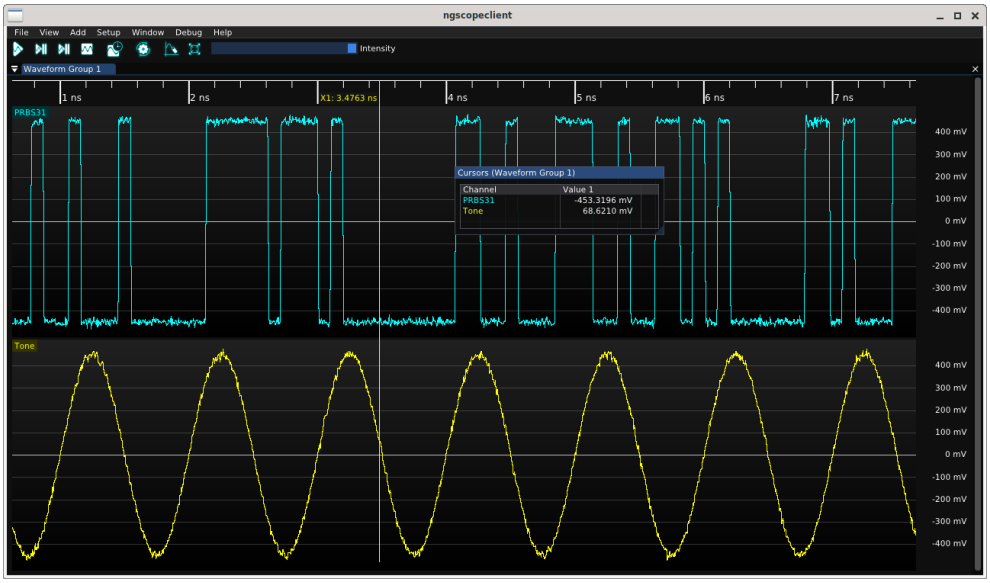

Figure 6.5: Single vertical cursor

Figure 6.6: Double vertical cursor

To place a single cursor, click on the waveform at the desired location. To place double cursors, click at the starting location to place the first cursor then drag to the ending location and release the mouse to place the second cursor. Once placed, either cursor can be moved by clicking on it and dragging to the new location.

In the timeline each cursor will display its X-axis position. If both cursors are active, the delta between them is shown. If the X axis uses time units, the frequency with period equal to the cursor spacing is also shown.

When a cursor is active, a dockable pop-up dialog appears displaying the value of each waveform in the group at the cursor location. If two cursors are active, both values as well as the difference between them is shown (Fig. 6.6)

6.5.2 Markers

A marker is a named location in *absolute* time intended for marking specific events (such as protocol packets or glitches) which may need to be re-examined in the future. When new waveforms are acquired, the marker remains attached to the same point in the old waveform and will disappear until the old waveform is re-loaded from the history window. In Fig. 6.7, two of the three markers are visible while the third is in a prior waveform.

Unlike vertical cursors, which are local to a single waveform group, cursors are global and will appear at the same timestamp in all waveform groups. This allows an event of interest to be examined in detail in one view, while a different view provides a global overview of the entire acquisition or examines another event (Fig. 6.8).

Creating a marker automatically pins the active waveform so it will not be removed from history as new data is acquired. The waveform cannot be un-pinned unless all markers are deleted first, or the waveform itself is manually deleted.

Newly created markers will have default numeric names such as M1, M2, etc. This name can be changed from the history window.

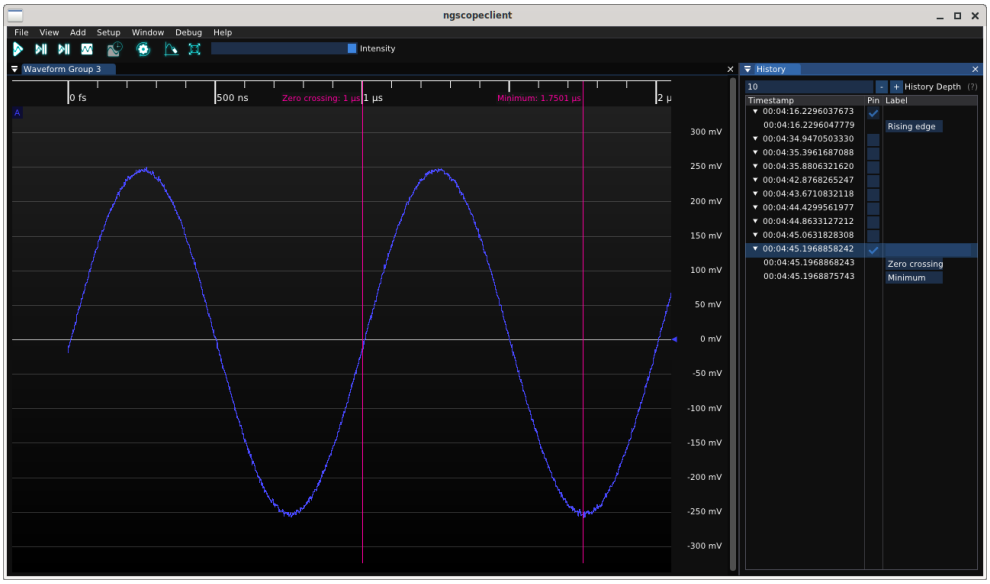

Figure 6.7: Session with three markers, two on the currently displayed waveform and one on a prior waveform

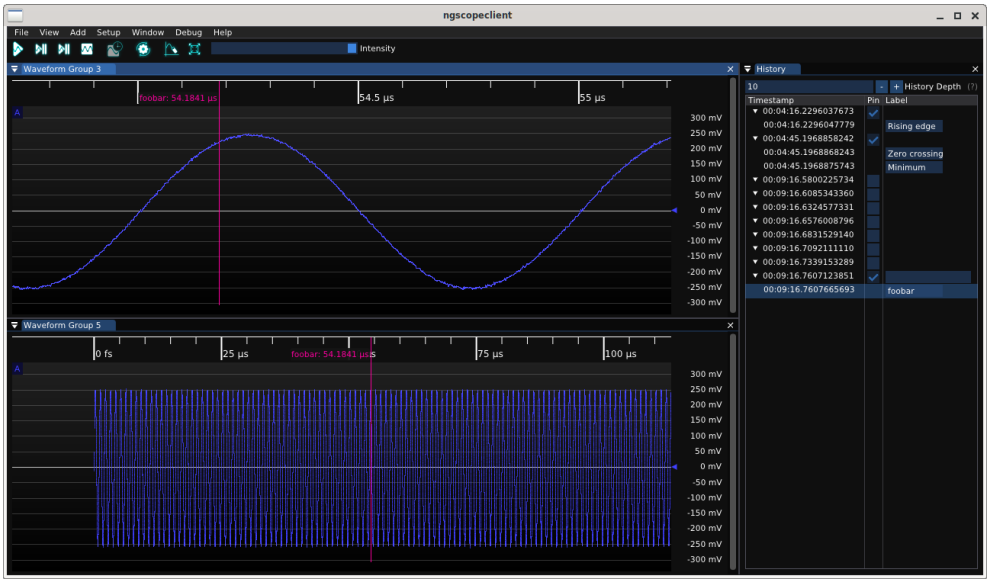

Figure 6.8: A single marker seen at multiple time scales in different views

Chapter 7

History

ngscopeclient saves a rolling buffer of previous waveforms in memory, allowing you to go back in time and see previous state of the system being debugged. Clicking on a timestamp in the history view (Fig. 7.1) pauses acquisition and loads the historical waveform data for analysis. History is captured regardless of whether the window is visible or not.

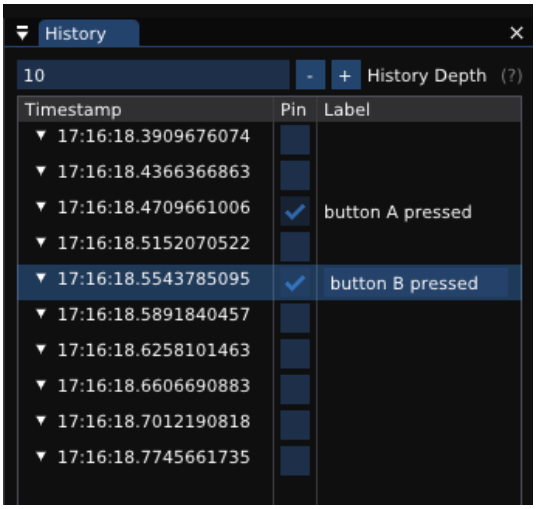

Figure 7.1: Waveform history view

The history depth defaults to 10 waveforms, but can be set arbitrarily within the limits of available RAM. Older waveforms beyond the history limit are deleted as new waveforms are acquired. Any single waveform in history may also be deleted by right clicking on the line and selecting “delete" from the menu.

7.1 Pinning

Interesting waveforms may be “pinned" in the history by checking the box in the “pin" column of the history view. Pinned waveforms are guaranteed to remain in the history buffer even when new waveforms arrive; only unpinned waveforms are eligible for automatic deletion to make space for incoming data.

If a waveform contains markers (6.5.2), it is automatically pinned and cannot be unpinned unless the marker (or entire waveform) is deleted. This prevents accidental loss of an important waveform: if the event was important enough to mark and name, it is probably worth keeping around.

7.2 Labeling

Arbitrary text names may be assigned to a waveform by clicking the corresponding cell in the “label" column. Waveforms with a label are automatically pinned, since assigning a label implies the waveform is important.

Chapter 8

Filter Graph Editor

The filter graph editor allows complex signal processing pipelines to be developed in a graphical fashion. It may be accessed from the `Window / Filter Graph` menu item.

NOTE: While the graph editor can be floated or docked just like any other window in ngscopeclient, there is a clipping bug (scopehal-app:572) causing the graph view to be drawn incorrectly or not at all when not docked. The workaround is to only interact with the filter graph when it is in a docked state.

The graph editor view (Fig. 8.1) shows nodes for every instrument channel, trigger, and filter block in the current session. As new instruments, channels, and filter blocks are added to the session, new nodes will automatically appear in the graph editor view. Nodes cannot overlap and will automatically move out of the way if another node is dragged on top of them.

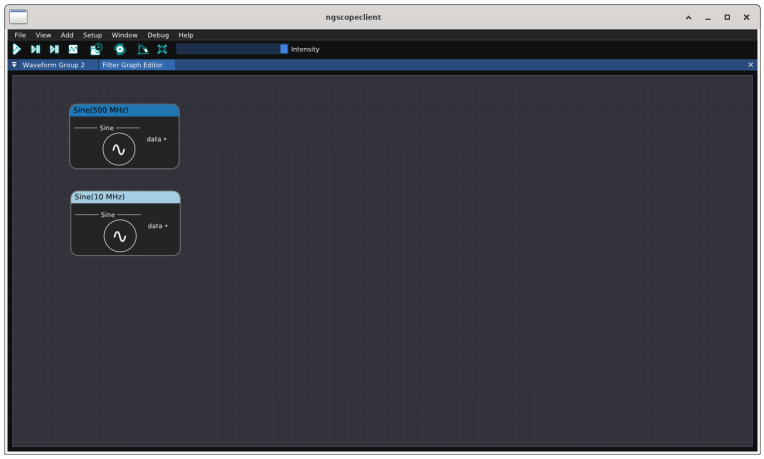

Figure 8.1: Filter graph editor showing two sinewave sources

The view may be zoomed with the mouse wheel, or panned by dragging with the right mouse button, to navigate large filter graphs which do not fit on a single screen. Right clicking on a node opens a pop-up properties view (Fig. 8.2).

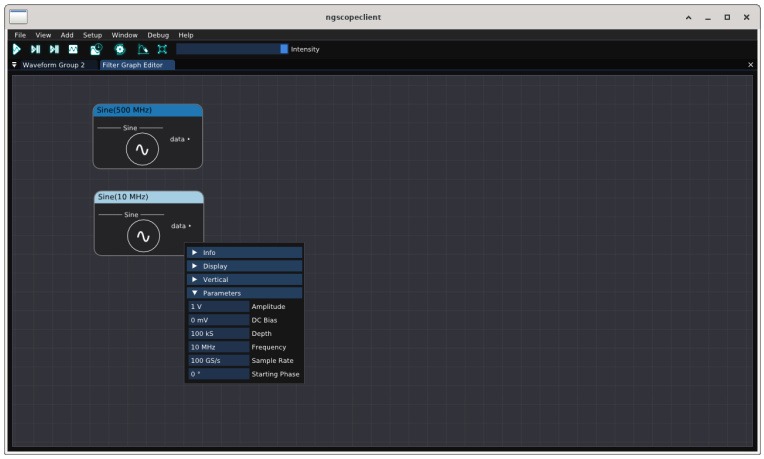

Figure 8.2: Filter graph editor showing properties popup

Nodes display inputs at left and outputs at right. To connect two existing nodes, click on an input or output port and drag to the port you wish to connect it to. An input can only connect to one output at a time; if the destination already is connected to a different signal the previous connection will be removed and replaced with the new one.

A tooltip with a green plus sign is displayed during dragging if the proposed connection is valid. If the tooltip displays a red X instead, the connection is invalid (connecting two inputs, two outputs, or an input and output of incompatible data types).

To create a new node, click on an input or output port and drag to an empty area of the canvas (Fig. 8.3). A context menu will appear, presenting a list of filters which can accept (if dragging from an output) or produce (if dragging from an input) the desired data type.

When a new block is added to the filter graph, each output channel will be automatically added to an existing waveform view if a compatible one is present. If no compatible view is available, a new view and/or group will be created.

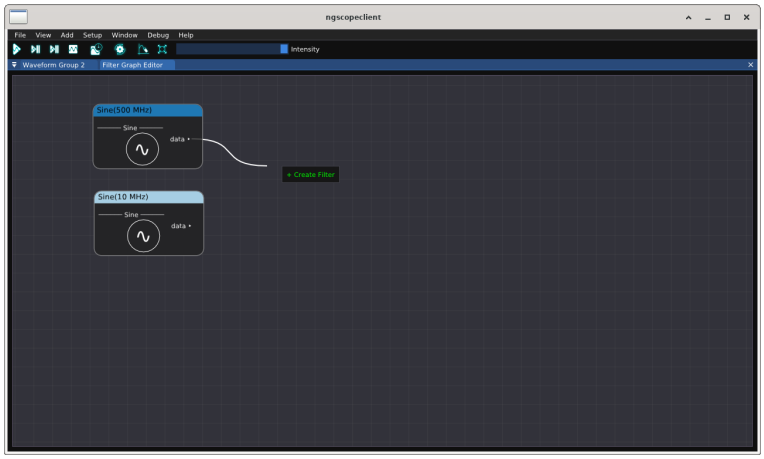

Figure 8.3: Filter graph editor dragging to an empty area of the canvas

Node title bars are color-coded to match the display color of the waveform trace, allowing easy navigation between waveform views and the graph editor.

Basic math blocks show icons in the graph editor view, allowing easier understanding of complex signal processing pipelines. Icons are gradually being added to more protocol decodes and filters to improve legibility.

Chapter 9

Transports

Libscopehal uses a “transport" driver in order to pass commands and data to instruments, in order to decouple the specifics of LXI, USBTMC, etc. from individual instrument drivers. This section describes the supported transports and their usage and limitations.

Not all transports are possible to use with any given driver due to hardware limitations or software/firmware quirks. For details on which transport(s) are usable with a particular instrument, consult the documentation for that device's driver.

9.1 gpib

SCPI over GPIB.

This transport takes up to four arguments: GPIB board index, primary address, secondary address, and timeout value. Only board index and primary address are required.

NOTE: The current implementation of this driver only works on Linux, using the linux-gpib library.

Example:

```
glscopeclient myscope:keysightdca:gpib:0:7
```

9.2 lan

SCPI over TCP with no further encapsulation.

This transport takes two arguments: hostname/IP and port number.

If port number is not specified, uses TCP port 5025 (IANA assigned) by default. Note that Rigol oscilloscopes use the non-standard port 5555, not 5025, so the port number must always be specified when using a Rigol instrument.

Example:

```
glscopeclient myscope:rigol:lan:192.0.2.9:5555
```

9.3 lxi

SCPI over LXI VXI-11.

Note that due to the remote procedure call paradigm used by LXI, it is not possible to batch multiple outstanding requests to an instrument when using this transport. Some instruments may experience reduced performance when using LXI as the transport. Drivers which require command batching may not be able to use LXI VXI-11 as the transport even if the instrument supports it.

Example:

```
glscopeclient myscope:tektronix:lxi:192.0.2.9
```

9.4 null

This transport does nothing, and is used as a placeholder for development simulations or non-SCPI instruments.

NOTE: Due to limitations of the current command line argument parsing code, an argument must be provided to all transports, including this one, when connecting via the command line. Since the argument is ignored, any non-empty string may be used.

Example:

```
glscopeclient sim:demo:null:blah
```

9.5 twinlan

This transport is used by some Antikernel Labs oscilloscopes, as well as most of the bridge servers used for interfacing libscopehal with USB oscilloscopes' SDKs. It takes three arguments: hostname/IP and two port numbers.

It uses two TCP sockets on different ports. The first carries SCPI text (as in the "lan" transport), and the second is for binary waveform data.

If port numbers are not specified, the SCPI port defaults to the IANA standard of 5025, and the data port defaults to 5026. If the SCPI port but not the data port is specified, the data port defaults to the SCPI port plus one.

9.6 uart

SCPI over RS-232 or USB-UART.

This transport takes two arguments: device path (required) and baud rate (optional). If baud rate is not specified, it defaults to 115200.

Example:

```
glscopeclient myscope:rigol:uart:/dev/ttyUSB0:115200
```

9.7 usbtmc

SCPI over USB Test & Measurement Class protocol.

This transport takes one argument: the path to the usbtmc kernel device object.

NOTE: The current implementation of this driver only works on Linux. There is currently no support for USBTMC on Windows (see scopehal:301)

Example:

```
glscopeclient myscope:siglent:usbtmc:/dev/usbtmc0
```

9.8 vicp

SCPI over Teledyne LeCroy Virtual Instrument Control Protocol.

This transport takes two arguments: hostname/IP and port number.

If port number is not specified, uses TCP port 1861 (IANA assigned) by default.

Example:

```
glscopeclient myscope:lecroy:vicp:192.0.2.9
```

Chapter 10

BERT Drivers

This chapter describes all of the available drivers for bit error rate testers (BERTs)

10.1 MultiLANE

Device Family	Driver	Transport	Notes
ML4039	mlbert	lan	Use scopehal-mlbert-bridge

10.1.1 mlbert

This driver is intended to connect via the scopehal-mlbert-bridge server for network transparency and does not directly link to the MultiLANE SDK or talk directly to the instrument. The bridge requires a Windows PC since MultiLANE's SDK is Windows only, however the libscopehal clientside driver can run on any supported OS.

It was developed using a ML4039 but may work with other similar models as well.

Chapter 11

Function Generator Drivers

This chapter describes all of the available drivers for standalone function generators.

Function generators which are part of an oscilloscope are described in the Oscilloscope Drivers section.

11.1 Rigol

Device Family	Driver	Transport	Notes
DG4000 series	rigol_awg	lan	Only tested via lan transport, but USBTMC and serial are available too

11.1.1 rigol_awg

This driver supports all DG4000 series function / arbitrary waveform generators.

Chapter 12

Electronic Load Drivers

This chapter describes all of the available drivers for electronic loads.

12.1 Siglent

Device Family	Driver	Transport	Notes
DSL1000X/X-E series	siglent_load	lan	Only tested via lan transport, but USBTMC and serial are available too

12.1.1 siglent_load

This driver supports all SDL1000 family loads (SDL1020X-E, SDL1020X, SDL1030X-E, SDL1030X).

Chapter 13

Multimeter Drivers

This chapter describes all of the available drivers for multimeters.

Multimeters which are part of an oscilloscope are described in the Oscilloscope Drivers section.

13.1 Rohde & Schwarz

Device Family	Driver	Transport	Notes
HMC8012	rs_hmc8012	lan	Only tested via lan transport, but USBTMC and serial are available too

13.1.1 rs_hmc8012

This driver supports the HMC8012 multimeter, which is the only device in the family.

Chapter 14

Oscilloscope Drivers

This chapter describes all of the available drivers for oscilloscopes and logic analyzers.

14.1 Agilent

Agilent devices support a similar similar SCPI command set across most device families.

Please see the table below for details of current hardware support:

Device Family	Driver	Transport	Notes
DSO5000 series	agilent	lan	Not recently tested, but should work.
DSO6000 & MSO6000 series	agilent	lan	Working. No support for digital channels yet.
DSO7000 & MSO7000 series	agilent	lan	Untested, but should work. No support for digital channels yet.
MSOX-2000 series	agilent	lan	
MSOX-3000 series	agilent	lan	

14.1.1 agilent

Typical Performance (MSO6034A, LAN)

Interestingly, performance sometimes gets better with more channels or deeper memory. Not sure why.

Channels	Memory depth	WFM/s
1	1K	66
4	1K	33
4	4K	33
1	40K	33
1	4K	22
1	20K	22
4	20K	22
1	100K	22
4	10K	17
4	40K	12
1	200K	11
1	400K	8
4	100K	6.5
4	200K	4
1	1M	3.7
4	400K	2.3
1	1M	1
4	1M	1
4	4M	0.2

Typical Performance (MSOX3104T, LAN)

Channels	Memory depth	WFM/s
1	2.5K	3.3
4	2.5K	2.5
1	2.5M	1.0
4	2.0M	0.5

14.2 Antikernel Labs

Device Family	Driver	Notes
Internal Logic Analyzer IP	akila	
BLONDEL Oscilloscope Prototype	aklabs	

14.2.1 akila

This driver uses a raw binary protocol, not SCPI.

Under-development internal logic analyzer analyzer core for FPGA design debug. The ILA uses a UART interface to a host system. Since there's no UART support in scopehal yet, socat must be used to bridge the UART to a TCP socket using the "lan" transport.

14.2.2 aklabs

This driver uses two TCP sockets. Port 5025 is used for SCPI control plane traffic, and port 50101 is used for waveform data using a raw binary protocol.

14.3 Demo

The “demo" driver is a simulation-only driver for development and training purposes, and does not connect to real hardware.

It ignores any transport provided, and is normally used with the “null" transport.

The demo instrument is intended to illustrate the usage of ngscopeclient for various types of analysis and to aid in automated testing on computers which do not have a connection to a real oscilloscope, and is not intended to accurately model the response or characteristics of real world scope frontends or signals.

It supports memory depths of 10K, 100K, 1M, and 10M points per waveform at rates of 1, 5, 10, 25, 50, and 100 Gsps. Four test signals are provided, each with 10 mV of Gaussian noise and a 5 GHz low-pass filter added (although this can be disabled under the channel properties)

Test signals:

- 1.000 GHz tone
- 1.000 GHz tone mixed with a second tone, which sweeps from 1.100 to 1.500 GHz
- 10.3125 Gbps PRBS-31
- 1.25 Gbps repeating two 8B/10B symbols (K28.5 D16.2)

Device Family	Driver	Transport	Notes
Simulator	demo	null	

14.4 Digilent

Digilent oscilloscopes using the WaveForms SDK are all supported using the “digilent" driver in libscopehal. This driver connects using the “twinlan" transport to a socket server which links against the Digilent WaveForms SDK. This provides network transparency, and allows the Digilent bridge server to be packaged separately for distribution and only installed by users who require it.

As of 2022-03-09, analog input channels on the Analog Discovery Pro and Analog Discovery 2 have been tested and are functional, however only basic edge triggering is implemented so far. Analog inputs on other devices likely work, however only these two have been tested to date.

Analog outputs, digital inputs, and digital outputs are currently unimplemented, but are planned to be added in the future.

14.4.1 digilent

Device Family	Driver	Transport	Notes
Electronics Explorer	digilent	twinlan	Not tested, but probably works
Analog Discovery	digilent	twinlan	Not tested, but probably works
Analog Discovery 2	digilent	twinlan	No digital channel support No analog output support
Analog Discovery Pro	digilent	twinlan	No digital channel support No analog output support
Digital Discovery	digilent	twinlan	No digital channel support, so pretty useless for now

Typical Performance (ADP3450, USB -> LAN)

Channels	Memory depth	WFM/s
4	64K	25.8
2	64K	32.3
1	64K	33.0

14.5 DreamSource Lab

DreamSourceLabs oscilloscopes and logic analyzers supported in their fork of sigrok ("libsigrok4DSL" distributed as part of their "DSView" software package) are supported through the "dslabs" driver in libscopehal. This driver connects using the "twinlan" transport to a socket server which links against libsigrok4DSL. This provides network transparency, and allows the DSLabs bridge server to be packaged separately for distribution and only installed by users who require it.

As of 2022-03-22, a DSCope U3P100 and a DSLogic U3Pro16has been tested and works adequately. Other products may work also, but are untested.

On DSCope: Only edge triggers are supported. 'Any' edge is not supported. "Ch0 && Ch1" and "Ch0 || Ch1" trigger modes are not supported.

On DSLogic: Only edge triggers are supported. All edges are supported. There is currently no way to configure a trigger on more than one channel. Serial / multi-stage triggers are not supported.

Known issues pending fixes/refactoring:

- Interleaved sample rates are not correctly reported in the timebase dialog (but are in the waveform display)
- Trigger position is quantized to multiples of 1% of total capture
- Non-localhost performance, and responsiveness in general may suffer as a result of hacky flow control on waveform capture
- DSLogic depth configuration is confusing and performance could be improved (currently only buffered more is supported)
- DSLogic devices trigger even if pre-trigger buffer has not been filled, leading to a small pre-trigger waveform in some cases

14.5.1 dslabs

Family / Device	Driver	Transport	Notes
DSCope U3P100	dslabs	twinlan	Tested, works
DSLogic U3P16	dslabs	twinlan	Tested, works
DSCope (others)	dslabs	twinlan	Not tested, but probably works
DSLogic (others)	dslabs	twinlan	Not tested, but probably works

Typical DSCope Performance (DSCope U3P100, USB3, localhost)

Channels	Memory depth	Sample Rate	WFM/s	UI-unconstrained WFM/s
2	1M	100MS/s	14	50
2	5M	500MS/s	4.5	14
1	5M	1GS/s	8.3	32

Typical DSLogic Performance (DSLogic U3Pro16, USB3, localhost)

Channels	Memory depth	Sample Rate	WFM/s	UI-unconstrained WFM/s
16	500k	100MS/s	16	44
16	500k	500MS/s	16	55

14.6 EEVengers

TODO: document WIP ThunderScope driver

14.7 Enjoy Digital

TODO (scopehal:79)

14.8 Hantek

TODO (scopehal:26)

14.9 Keysight

Keysight devices support a similar similar SCPI command set across most device families. Many Keysight devices were previously sold under the Agilent brand and use the same SCPI command set, so they are supported by the “agilent" driver.

Please see the table below for details of current hardware support:

14.9.1 agilent

Device Family	Driver	Notes
MSOX-2000 series	agilent	
MSOX-3000 series	agilent	
MSOX-3000T series	agilent	

14.9.2 keysightdca

A driver for the Keysight/Agilent/HP DCA series of equivalent-time sampling oscilloscopes.

Device Family	Driver	Notes
86100A	keysightdca	

14.10 Pico Technologies

Pico oscilloscopes all have slightly different command sets, but are supported using the “pico" driver in libscopehal. This driver connects via a TCP socket to a socket server scopehal-pico-bridge which connects to the appropriate instrument using Pico’s binary SDK.

Device Family	Driver	Notes
3000D series	pico	Early development, incomplete
6000E series	pico	

14.10.1 pico

Typical Performance (6824E, LAN)

Channels	Memory depth	WFM/s
8	1M	15.2
4	1M	30.5
2	1M	64.4
1	10M	12.2
1	50M	3.03

14.11 Rigol

Rigol oscilloscopes have subtle differences in SCPI command set, but this is implemented with quirks handling in the driver rather than needing different drivers for each scope family.

Device Family	Driver	Notes
DS1100D/E	rigol	
DS1000Z	rigol	
MSO5000	rigol	

14.11.1 rigol

Typical Performance (MSO5000 series, LAN)

Channels	Memory depth	WFM/s
4	10K	0.96
4	100K	0.91
4	1M	0.59
4	10M	0.13
1	100M	0.0601
4	25M	0.0568
2	50M	0.0568

14.12 Rohde & Schwarz

14.12.1 rs

There is partial support for RTM3000 (and possibly others, untested) however it appears to have bitrotted.

TODO (scopehal:59)

14.12.2 rs_rto6

This driver supports the newer RTO6 family scopes (and possibly others, untested).

14.13 Saleae

TODO (scopehal:16)

14.14 Siglent

A driver for SDS2000X+ is available in the codebase which has been developed according to Siglent offical documentation (Programming Guide PG01-E11A). This driver should be functional across the 'next generation' SDS2000X+, SDS5000X and SDS6000X scopes . It has been primarily developed using the SDS2000X+. Some older generation scopes are supported as well.

Digital channels are not supported on any scope yet, due to lack of an MSO probe to test with. Many trigger types are not yet supported.

Device Family	Driver	Transport	Notes
SDS1000X-E series	siglent	lan	Initialises, triggers and downloads waveforms. More testing needed
SDS2000X-E series	siglent	lan	Initialises, triggers and downloads waveforms. More testing needed
SDS2000X+ series	siglent	lan	Basic functionality complete.
SDS2000X HD series	siglent	lan	Tested and works well on SDS2354x HD.
SDS5000X series	siglent	lan	Initialises, triggers and downloads waveforms. More testing needed
SDS6000A series	siglent	lan	Tested and works well on SDS6204A. 10/12 bit models NOT supported, but unavailable for dev (not sold in western markets).

Typical Performance (SDS2104X+, LAN)

Channels	Memory depth	WFM/s
1	5-100K	2.3
2	5-100K	1.6
3	5-100K	1.2
4	5-100K	1
1	10M	0.5
2-4	10M	0.15

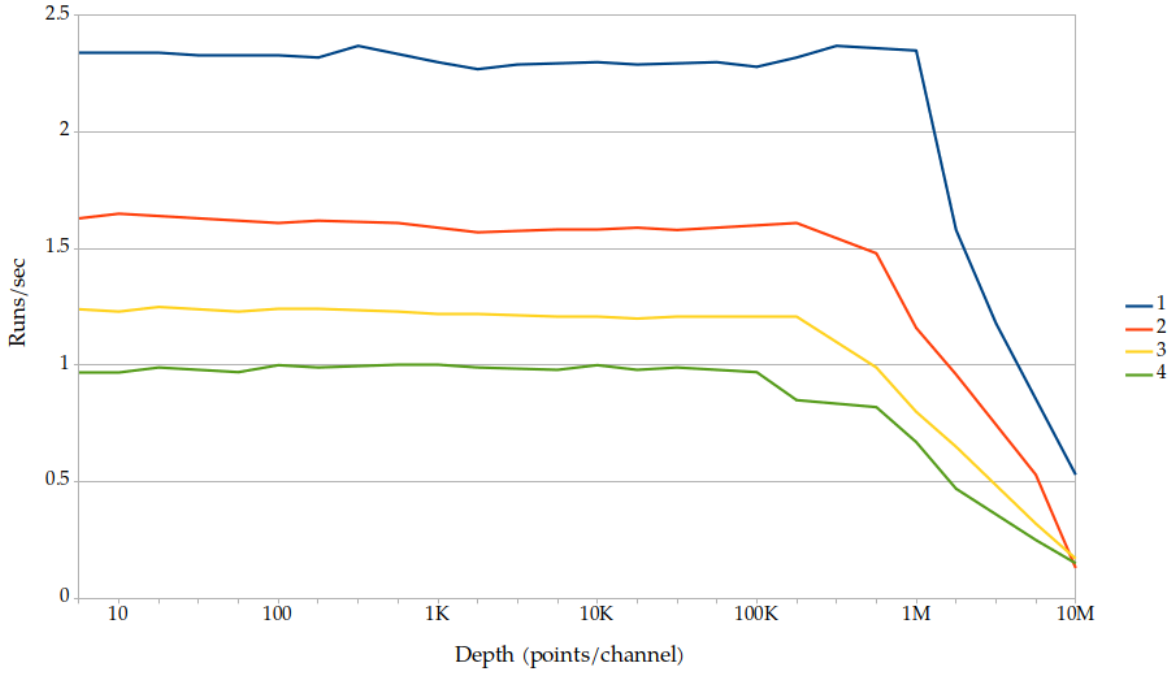

Figure 14.1: Siglent sample speed for various combinations of depth and channels

These figures were obtained from a SDS2104X+ running firmware version 1.3.7R5. Different scopes and software revisions may vary. This series of scopes support sample depths up to 100MPoints, but depths beyond 10MPoints require a different software interface and are likely to be extremely slow, so have not yet been implemented.

14.15 Teledyne LeCroy / LeCroy

Teledyne LeCroy (and older LeCroy) devices use the same driver, but two different transports for LAN connections.

While all Teledyne LeCroy / LeCroy devices use almost identical SCPI command sets, Windows based devices running XStream or MAUI use a custom framing protocol (“vicp") around the SCPI data while the lower end RTOS based devices use raw SCPI over TCP (“lan").

Please see the table below for details on which configuration to use with your hardware.

Device Family	Driver	Transport	Notes
DDA	lecroy	vicp	Tested on DDA5000A series
HDO	lecroy	vicp	Tested on HDO9000 series
LabMaster	lecroy	vicp	Untested, but should work for 4-channel setups
MDA	lecroy	vicp	Untested, but should work
SDA	lecroy	vicp	Tested on SDA 8Zi and 8Zi-A series
T3DSO	???	???	Untested
WaveAce	???	???	Untested
WaveJet	???	???	Untested
WaveMaster	lecroy	vicp	Same hardware as SDA/DDA
WaveRunner	lecroy	vicp	Tested on WaveRunner Xi, 8000, and 9000 series
WaveSurfer	lecroy	vicp	Tested on WaveSurfer 3000 series

14.15.1 lecroy

This is the primary driver for MAUI based Teledyne LeCroy / LeCroy devices.

This driver has been tested on a wide range of Teledyne LeCroy / LeCroy hardware. It should be compatible with any Teledyne LeCroy or LeCroy oscilloscope running Windows XP or newer and the MAUI or XStream software.

Typical Performance (HDO9204, VICP)

Channels	Memory depth	WFM/s
1	100K	>50
1	400K	29 - 35
2	100K	30 - 40
4	100K	17 - 21
1	2M	9 - 11
1	10M	2.2 - 2.6
4	1M	5.2 - 6.5
1	64M	0.41 - 0.42
2	64M	0.21 - 0.23
4	64M	0.12 - 0.13

Typical Performance (WaveRunner 8404M-MS, VICP)

Channels	Memory depth	WFM/s
1	80K	35 - 45
2	80K	35 - 45
2	800K	16 - 17
2	8M	3.1 - 3.2

14.15.2 lecroy_fwp

This is a special performance-enhanced extension of the base “lecroy" driver which takes advantage of the FastWavePort feature of the instrument to gain high speed access to waveform data via shared memory. Waveforms are pulled from shared memory when a synchronization event fires, then pushed to the client via a separate TCP socket on port 1862.

On low latency LANs, typical performance increases observed with SDA 8Zi series instruments are on the order of 2x throughput vs using the base driver downloading waveforms via SCPI. On higher latency connections such as VPNs, the performance increase is likely to be even higher because the push-based model eliminates the need for polling (which performs increasingly poorly as latency increases).

To use this driver, your instrument must have the XDEV software option installed and the scopehal-fwp-bridge server application running. If the bridge or option are not detected, the driver falls back to SCPI waveform download and will behave identically to the base “lecroy" driver.

There are some limitations to be aware of with this driver:

- Maxmimum memory depth is limited to no more than 40M samples per channel, regardless of installed instrument memory. This is an architectural limitation of the FastWavePort API;

the next generation FastMultiWavePort API eliminates this restriction however scopehal-fwp-bridge does not yet support it due to poor documentation.

- MSO channels are not supported, because neither FastWavePort nor FastMultiWavePort provide shared memory access to digital channel data. There is no known workaround for this given current instrument APIs.
- A maximum of four analog channels are supported even if the instrument actually has eight. There are no major technical blockers to fixing this under FastWavePort however no 8-channel instruments are available to the developers as of this writing, so there is no way to test potential fixes. FastMultiWavePort has a limit of four channels per instance, but it may be possible to instantiate multiple copies of the FastMultiWavePort block to work around this.
- Math functions F9-F12 are used by the FastWavePort blocks and cannot be used for other math functions.

14.16 Tektronix

This driver is being primarily developed on a MSO64. It supports SCPI over LXI VXI-11 or TCP sockets.

The hardware supports USBTMC, however waveform download via USBTMC does not work with libscopehal for unknown reasons.

Device Family	Driver	Transport	Notes
MSO5 series	tektronix	lan, lxi	
MSO6 series	tektronix	lan, lxi	

14.16.1 Note regarding “lan" transport on MSO5/6

The default settings for raw SCPI access on the MSO6 series use a full terminal emulator rather than raw SCPI commands. To remove the prompts and help text, go to Utility | I/O, then under the Socket Server panel select protocol “None" rather than the default of “Terminal".

Typical Performance (MSO64, LXI, embedded OS)

Channels	Memory depth	WFM/s
1	50K	10.3 - 11.4
2	50K	6.7 - 7.2
4	50K	5.1 - 5.3
1	500K	8.7 - 9.5
4	500K	3.8 - 3.9

14.17 Xilinx

TODO (scopehal:40)

Chapter 15

Power Supply Drivers

This chapter describes all of the available drivers for power supplies.

15.1 GW Instek

Device Family	Driver	Transport	Notes
GPD-X303S series	gwinstek_gpdx303s	uart	9600 Baud default. Tested with GPD-3303S. No support for tracking modes yet.

15.1.1 gwinstek_gpdx303s

Supported models should include GPD-2303S, GPD-3303S, GPD-4303S, and GPD-3303D.

15.2 Rohde & Schwarz

Device Family	Driver	Transport	Notes
HMC804x series	rs_hmc804x	uart, usbtmc, lan	No support for tracking modes yet.

15.2.1 rs_hmc804x

This driver should support the HMC8041, HMC8042, and HMC8043 but has only been tested on the HMC8042.

15.3 Siglent

Device Family	Driver	Transport	Notes
SPD3303X series	siglent_spd	lan	Tested with SPD3303X-E

15.3.1 siglent_spd

Supported models should include SPD3303X, SPD3303X-E.

NOTE: Channel 3 of the SPD3303x series does not support software voltage/current adjustment. It has a fixed current limit of 3.2A, and output voltage selectable to 2.5, 3.3, or 5V via a mechanical switch. While channel 3 can be turned on and off under software control, there is no readback capability whatsoever for channel 3 in the SCPI API.

As a result - regardless of actual hardware state - the driver will report channel 3 as being in constant voltage mode. Additionally, the driver will report channel 3 as being off until it is turned on by software. Once the output has been turned on, the driver will track the state and report a correct on/off state as long as no front panel control buttons are touched.

Chapter 16

RF Generator Drivers

This chapter describes all of the available drivers for RF synthesizers, vector signal generators, and similar devices.

16.1 Siglent

Device Family	Driver	Transport	Notes
SSG3000X	Unknown	Unknown	May be compatible with the siglent_ssg driver, but not tested
SSG5000A	Unknown	Unknown	May be compatible with the siglent_ssg driver, but not tested
SSG5000X	siglent_ssg	lan	Only tested via lan transport, but USBTMC and serial are available too

16.1.1 siglent_ssg

This driver was developed on a SSG5060X-V and should support the other models in the SSG5000X family (SSG5040X, SSG5060X, and SSG5040X-V). It is unknown whether it will function at all with the SSG3000X or 5000A families in its current state; additional development will likely be needed for full support.

Chapter 17

VNA Drivers

This chapter describes all of the available drivers for vector network analyzers.

17.1 Copper Mountain

Device Family	Driver	Transport	Notes
Planar	coppermt	lan	Not tested, but docs say same command set
S5xxx	coppermt	lan	Tested on S5180B
S7530	coppermt	lan	Not tested, but docs say same command set
SC50xx	coppermt	lan	Not tested, but docs say same command set
C1xxx	coppermt	lan	Not tested, but docs say same command set
C2xxx	coppermt	lan	Not tested, but docs say same command set
C4xxx	coppermt	lan	Not tested, but docs say same command set
M5xxx	coppermt	lan	Not tested, but docs say same command set

17.1.1 coppermt

This driver supports the S2VNA and S4VNA software from Copper Mountain.

As of this writing, only 2-port VNAs are supported. 4-port VNAs will probably work using only the first two ports, but this has not been tested.

17.2 Pico Technology

Device Family	Driver	Transport	Notes
PicoVNA 106	picovna	lan	
PicoVNA 108	picovna	lan	

17.2.1 picovna

This driver supports the PicoVNA 5 software from Pico Technology. The older PicoVNA 3 software does not provides a SCPI interface and is not compatible with this driver.

Chapter 18

Triggers

18.1 Trigger Properties

The `Setup / Trigger` menu opens the trigger properties dialog (Fig. 18.1).

The Trigger Type box allows the type of trigger to be chosen. The list of available triggers depends on the instrument model and installed software options.

The Trigger Offset field specifies the time from the *start* of the waveform to the trigger point. Positive values move the trigger later into the waveform, negative values introduce a delay between the trigger and the start of the waveform. [1]

Figure 18.1: Trigger properties dialog

The remaining settings in the trigger properties dialog depend on the specific trigger type chosen.

18.2 Serial Pattern Triggers

All serial pattern triggers take one or two pattern fields, a radix, and a condition.

For conditions like “between" or “not between" both patterns are used, and no wildcards are allowed. For other conditions, only the first pattern is used.

Patterns may be specified as ASCII text, hex, or binary. “Don’t care" nibbles/bits may be specified in hex/binary patterns as “X", for example “3fx8" or “1100010xxx1".

[1] This is a different convention than most oscilloscopes, which typically measure the trigger position from the *midpoint* of the waveform. Since glscopeclient decouples the acquisition length from the UI zoom setting, measuring from the midpoint makes little sense as there are no obvious visual cues to the midpoint’s location.

18.3 Dropout

Triggers when a signal stops toggling for a specified amount of time.

18.3.1 Inputs

Signal name	Type	Description
din	Analog or digital	Input signal

18.3.2 Parameters

Parameter name	Type	Description
Edge	Enum	Specifies the polarity of edge to look for (rising or falling)
Dropout Time	Int	Dropout time needed to trigger
Level	Float	Voltage threshold
Reset Mode	Enum	Specifies whether to reset the timer on the opposite edge

18.4 Edge

Triggers on edges in the signal.

Edge types “rising" and “falling" are self-explanatory. “Any" triggers on either rising or falling edges. “Alternating" is a unique trigger mode only found on certain Agilent/Keysight oscilloscopes, which alternates each waveform between rising and falling edge triggers.

18.4.1 Inputs

Signal name	Type	Description
din	Analog or digital	Input signal

18.4.2 Parameters

Parameter name	Type	Description
Edge	Enum	Specifies the polarity of edge to look for
Level	Float	Voltage threshold

18.5 Glitch

TODO: This is supported on at least LeCroy hardware, but it’s not clear how it differs from pulse width.

18.6 Pulse Width

Triggers when a high or low pulse meeting specified width criteria is seen.

Signal name	Type	Description
din	Analog or digital	Input signal

18.6.1 Parameters

Parameter name	Type	Description
Condition	Enum	Match condition (greater, less, between, or not between)
Edge	Enum	Specifies the polarity of edge to look for
Level	Float	Voltage threshold
Lower Bound	Int	Lower width threshold
Upper Bound	Int	Upper width threshold

18.7 Runt

Triggers when a pulse of specified width crosses one threshold, but not a second.

Signal name	Type	Description
din	Analog	Input signal

18.7.1 Parameters

Parameter name	Type	Description
Condition	Enum	Match condition (greater, less, between, or not between)
Edge Slope	Enum	Specifies the polarity of edge to look for
Lower Interval	Int	Lower width threshold
Lower Level	Float	Lower voltage threshold
Upper Interval	Int	Upper width threshold
Upper Level	Float	Upper voltage threshold

18.8 Slew Rate

Triggers when an edge is faster or slower than a specified rate.

Signal name	Type	Description
din	Analog	Input signal

18.8.1 Parameters

Parameter name	Type	Description
Condition	Enum	Match condition (greater, less, between, or not between)
Edge Slope	Enum	Specifies the polarity of edge to look for
Lower Interval	Int	Lower width threshold
Lower Level	Float	Lower voltage threshold
Upper Interval	Int	Upper width threshold
Upper Level	Float	Upper voltage threshold

18.9 UART

Triggers when a byte or byte sequence is seen on a UART.

18.9.1 Inputs

Signal name	Type	Description
din	Analog or digital	Input signal

18.9.2 Parameters

Parameter name	Type	Description
Bit Rate	Int	Baud rate
Condition	Enum	Match condition
Level	Float	Voltage threshold
Parity Mode	Enum	Odd, even, or no parity
Pattern	String	First match pattern
Pattern 2	String	Second match pattern
Polarity	Enum	Idle high (normal UART) or idle low (RS232)
Radix	Enum	Radix for the patterns
Stop Bits	Float	Number of stop bits
Trigger Type	Enum	Match data pattern or parity error

18.10 Window

Triggers when a signal goes above or below specified thresholds.

The available configuration settings for this trigger vary from instrument to instrument.

Signal name	Type	Description
din	Analog	Input signal

18.10.1 Parameters

Parameter name	Type	Description
Condition	Enum	Specifies whether to trigger on entry or exit from the window, and whether to trigger immediately or after a time limit.
Edge	Enum	Specifies which edge of the window to trigger on
Lower Level	Float	Lower voltage threshold
Upper Level	Float	Upper voltage threshold

Chapter 19

Filters

19.1 Introduction

19.1.1 Key Concepts

ngscopeclient and libscopehal are based on a “filter graph" architecture internally. The filter graph is a directed acyclic graph with a set of source nodes (waveforms captured from hardware, loaded from a saved session, or generated numerically) and sink nodes (waveform views, protocol analyzer views, and statistics) connected by edges representing data flow.

A filter is simply an intermediate node in the graph, which takes input from zero or more waveform nodes and outputs a waveform which may be displayed, used as input to other filters, or both. A waveform is a series of data points which may represent voltages, digital samples, or arbitrarily complex protocol data structures.

As a result, there is no internal distinction between math functions, measurements, and protocol decodes, and it is possible to chain them arbitrarily. Consider the following example:

- Two analog waveforms representing serial data and clock are acquired
- Each analog waveform is thresholded, producing a digital waveform
- The two digital waveforms are decoded as I^2C, producing a series of packets
- The I^2C packets are decoded as writes to a serial DAC, producing an analog waveform
- A moving average filter is applied to the analog waveform
- A measurement filter finds the instantaneous frequency of each cycle of the DAC output

In this document we use the term “filter" consistently to avoid ambiguity.

19.1.2 Conventions

A filter can take arbitrarily many inputs (vector inputs), arbitrarily many parameters (scalar inputs), and outputs a signal (vector output).

If the output signal is a multi-field type (as opposed to a single scalar, e.g. voltage, at each sample) the “Output Signal" section will include a table describing how various types of output data are displayed. Printf-style format codes maybe used for clarity. For example, “%02x" means data is formatted as hexadecimal bytes with leading zeroes.

All filters with complex output use a standardized set of colors to display various types of data fields in a consistent manner. These colors are configurable under the `Appearance / Decodes` preferences category.

Color name	Use case	Default Color
Address	Memory addresses	#ffff00
Checksum Bad	Incorrect CRC/checksum	#ff0000
Checksum OK	Valid CRC/checksum	#00ff00
Control	Miscellaneous control data	#c000a0
Data	User data	#336699
Error	Malformed/unreadable data	#ff0000
Idle	Inter-frame gaps	#404040
Preamble	Preamble/sync words	#808080

19.2 128b/130b

Decodes the 128b/130b line code used by PCIe gen 3/4/5. This filter performs block alignment and descrambling, but no decoding of block contents.

128b/130b, as a close relative of 64b/66b, is a serial line code which divides transmitted data into 128-bit blocks and scrambles them with a LFSR, then appends a 2-bit type field (which is not scrambled) to each block for synchronization. Block synchronization depends on always having an edge in the type field so types 2'b00 and 2'b11 are disallowed.

For PCIe over 128b/130b, block type 2'b01 contains 128 bits of upper layer protocol data while block type 2'b10 contains an ordered set.

Note that this filter only performs block alignment and descrambling. No decoding or parsing is applied to the 128-bit blocks, other than searching for skip ordered sets (beginning with 0xaa) and using them for scrambler synchronization.

19.2.1 Inputs

Signal name	Type	Description
data	1-bit digital	Serial 128b/130b data line
clk	1-bit digital	DDR bit clock, typically generated by use of the Clock Recovery (PLL) filter on the input data.

19.2.2 Parameters

This filter takes no parameters.

19.2.3 Output Signal

The 128B/130B filter outputs a time series of 128B/130B sample objects. These consist of a control/data flag and a 128-bit data block.

Type	Description	Color	Format
Ordered set	Block with type 2'b10	Control	%032x
Data	Block with type 2'b01	Data	%032x
Error	Block with type 2'b00 or 2'b11	Error	%032x

19.3 64b/66b

Decodes the 64/66b line code used by 10Gbase-R and other serial protocols, as originally specified in IEEE 802.3 clause 49.2.

64b/66b is a serial line code which divides transmitted data into 64-bit blocks and scrambles them with a LFSR, then appends a 2-bit type field (which is not scrambled) to each block for synchronization. Block synchronization depends on always having an edge in the type field so types 2'b00 and 2'b11 are disallowed.

Note that this filter only performs block alignment and descrambling. No decoding is applied to the 64-bit blocks, as different upper-layer protocols assign different meaning to them. In 10Gbase-R, type 2'b01 denotes "64 bits of upper layer data" and type 2'b10 denotes "8-bit type field and 56 bits of data whose meaning depends on the type", however this is not universal.

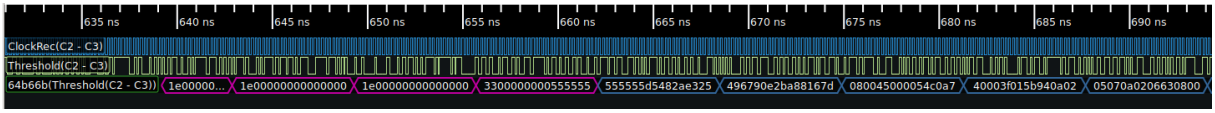

Figure 19.1: Example 64b/66b decode

19.3.1 Inputs

Signal name	Type	Description
data	1-bit digital	Serial 64b/66b data line
clk	1-bit digital	DDR bit clock, typically generated by use of the Clock Recovery (PLL) filter on the input data.

19.3.2 Parameters

This filter takes no parameters.

19.3.3 Output Signal

The 64B/66B filter outputs a time series of 64B/66B sample objects. These consist of a control/data flag and a 64-bit data block.

Type	Description	Color	Format
Control	Block with type 2'b10	Control	%016x
Data	Block with type 2'b01	Data	%016x
Error	Block with type 2'b00 or 2'b11	Error	%016x

19.4 8B/10B (IBM)

Decodes the standard 8b/10b line code used by SGMII, 1000base-X, DisplayPort, JESD204, PCIe gen 1/2, SATA, USB 3.0, and many other common serial protocols.

8b/10b is a dictionary based code which converts each byte of message data to a ten-bit code. In order to maintain DC balance and limit run length to a maximum of five identical bits in a row, all legal codes have one of:

- One legal coding, with exactly five zero bits
- Two legal codings, one with four zero bits and one with six

The transmitter maintains a “running disparity" counter and chooses the appropriate coding for each symbol to ensure DC balance. There are twelve legal codes which are not needed for encoding data values; these are used to encode frame boundaries, idle/alignment sequences, and other control information.

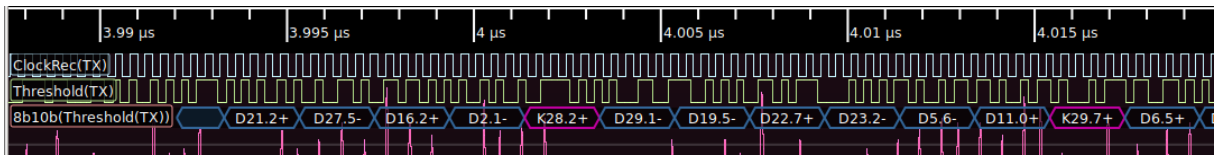

Figure 19.2: Example 8b/10b decode

19.4.1 Inputs

Signal name	Type	Description
data	1-bit digital	Serial 8b/10b data line
clk	1-bit digital	DDR bit clock, typically generated by use of the Clock Recovery (PLL) filter on the input data.

19.4.2 Parameters

Parameter name	Type	Description
Display Format	Enum	**Dotted (K28.5 D21.5)**: displays the 3b4b and 5b6b code blocks separately, with K or D prefix. **Hex (K.bc b5)**: displays data as hex byte values and control codes with a K prefix.

19.4.3 Output Signal

The 8B/10B filter outputs a time series of 8B/10B sample objects. These consist of a control/data flag and a byte of data.

Type	Description	Color	Format
Control	Control codes	Control	K%d.%d+ or K%02x
Data	Upper layer protocol data	Data	D%d.%d+ or %02x
Error	Malformed data	Error	ERROR

19.5 8B/10B (TMDS)

Decodes the 8-to-10 Transition Minimized Differential Signalling line code used in DVI and HDMI.

Like the 8B/10B (IBM) line code, TMDS is an 8-to-10 bit serial line code. TMDS, however, is designed to *minimize* the number of toggles in the data stream for EMC reasons, rendering it difficult to synchronize a CDR PLL to. As a result, HDMI and DVI provide a reference clock at the pixel clock rate (1/10 the serial data bit rate) along with the data stream to provide synchronization.

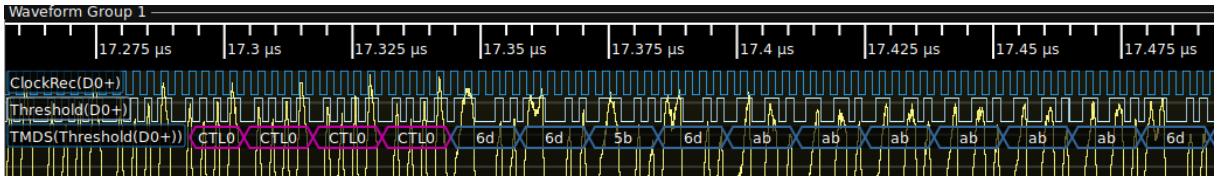

Figure 19.3: Example TMDS decode

19.5.1 Inputs

Signal name	Type	Description
data	1-bit digital	Serial TMDS data line
clk	1-bit digital	DDR *bit* clock, typically generated by use of the Clock Recovery (PLL) filter on the input data. Note that this is 5x the rate of the pixel clock signal.

19.5.2 Parameters

Parameter name	Type	Description
Lane Number	Integer	Lane number within the link (0-3)

19.5.3 Output Signal

The TMDS filter outputs a time series of TMDS sample objects. These consist of a type field and a byte of data.

The output of the TMDS decode is commonly fed to the DVI or HDMI protocol decoders.

Type	Description	Color	Format
Control	Control codes (H/V sync)	Control	CTL%d
Data	Pixel/island data	Data	%02x
Error	Malformed data	Error	ERROR
Guard band	HDMI data/video guard band	Preamble	GB

19.6 AC Couple

Automatically removes a DC offset from an analog waveform by subtracting the average of all samples from each sample.

This filter should only be used in postprocessing already acquired data, or other situations in which AC coupling in the hardware (via an AC coupled probe, or coaxial DC block) is not possible.

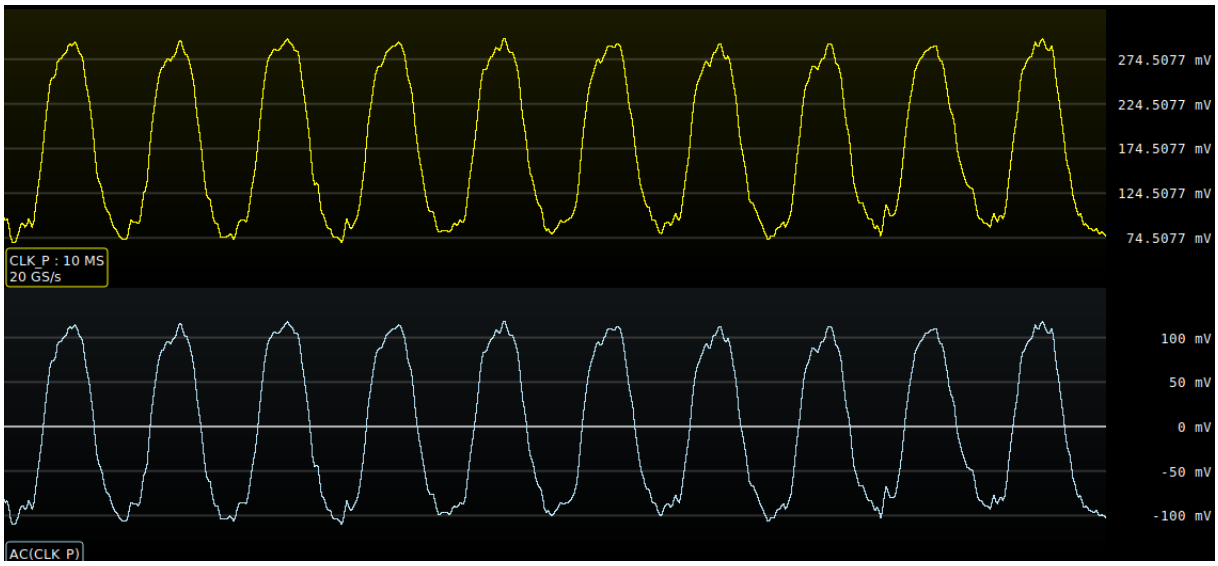

Figure 19.4: Example AC coupling

19.6.1 Inputs

Signal name	Type	Description
din	Analog	Input waveform

19.6.2 Parameters

This filter takes no parameters.

19.6.3 Output Signal

This filter outputs an analog waveform with identical sample rate to the input, vertically shifted to center the signal at zero volts.

19.7 AC RMS

Measures the Root Mean Square value of the waveform after removing any DC offset.

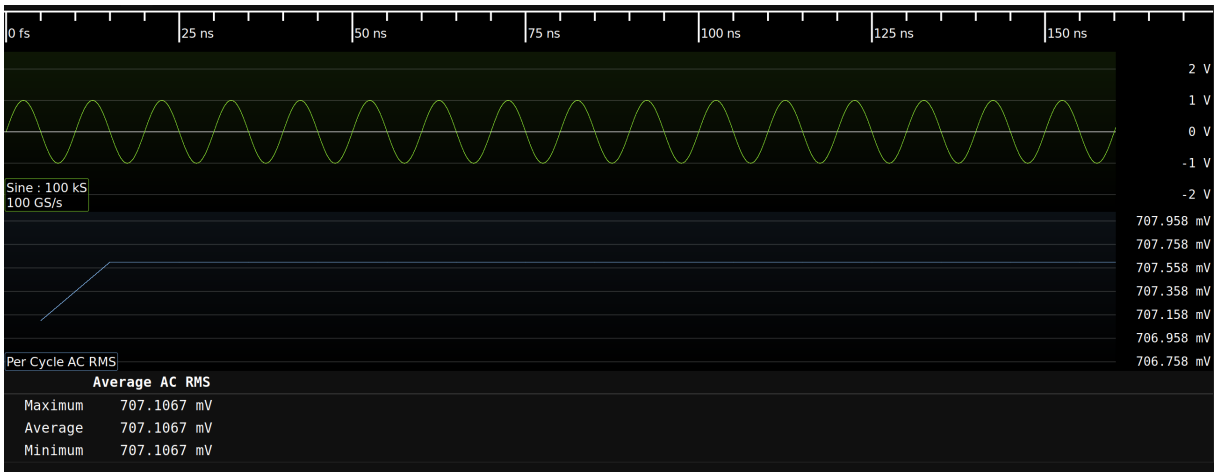

Figure 19.5: Example of an AC RMS Measurement of a Sinewave with 1V peak voltage

19.7.1 Inputs

Signal name	Type	Description
din	Analog	Input waveform

19.7.2 Parameters

This filter takes no parameters.

19.7.3 Output Signal

This filter has two output streams.

Stream name	Type	Description
trend	Sparse analog	One sample per cycle of the input waveform containing the RMS value across that cycle
avg	Scalar	RMS value across the entire waveform

19.8 Add

This filter adds two inputs. Either input may be a vector (waveform) or scalar.

19.8.1 Inputs

Signal name	Type	Description
a	Analog waveform or scalar	First input waveform
b	Analog waveform or scalar	Second input waveform

19.8.2 Parameters

This filter takes no parameters.

19.8.3 Output Signal

If both inputs are vectors, this filter outputs a waveform containing the pairwise sum; i.e. sample i of the output is $a[i] + b[i]$. No resampling is performed on the inputs so incorrect or unexpected results may occur if they do not share the same timebase.

If both inputs are scalars, this filter outputs their sum.

If one input is a vector and the other is a scalar, this filter outputs the sum of the scalar and each element of the waveform, i.e. sample i of the output is $a + b[i]$ for the scalar + vector case and $a[i] + b$ for the vector + scalar.

19.9 Area Under Curve

TODO: needs to be updated when we port to scalar interface

Measures the area under the curve by integrating the data points. By default, area measured above ground is considered as positive and area measured below the ground is considered negative. The negative area can also be considered as positive by changing a filter parameter. The measurement can be performed on the full record or on each cycle.

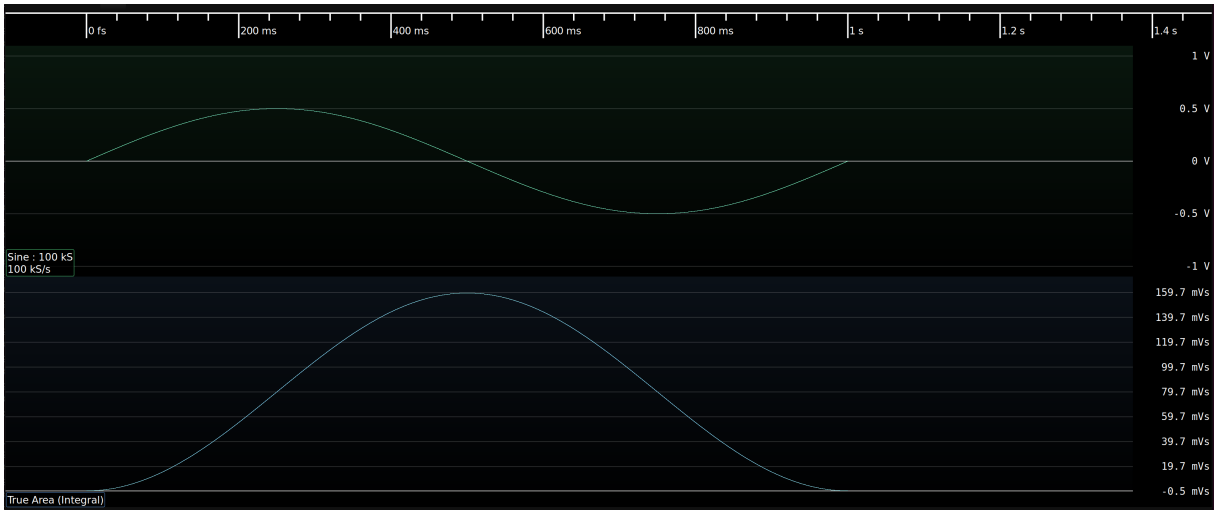

Figure 19.6: Example of true area under the curve measurement (Integral)

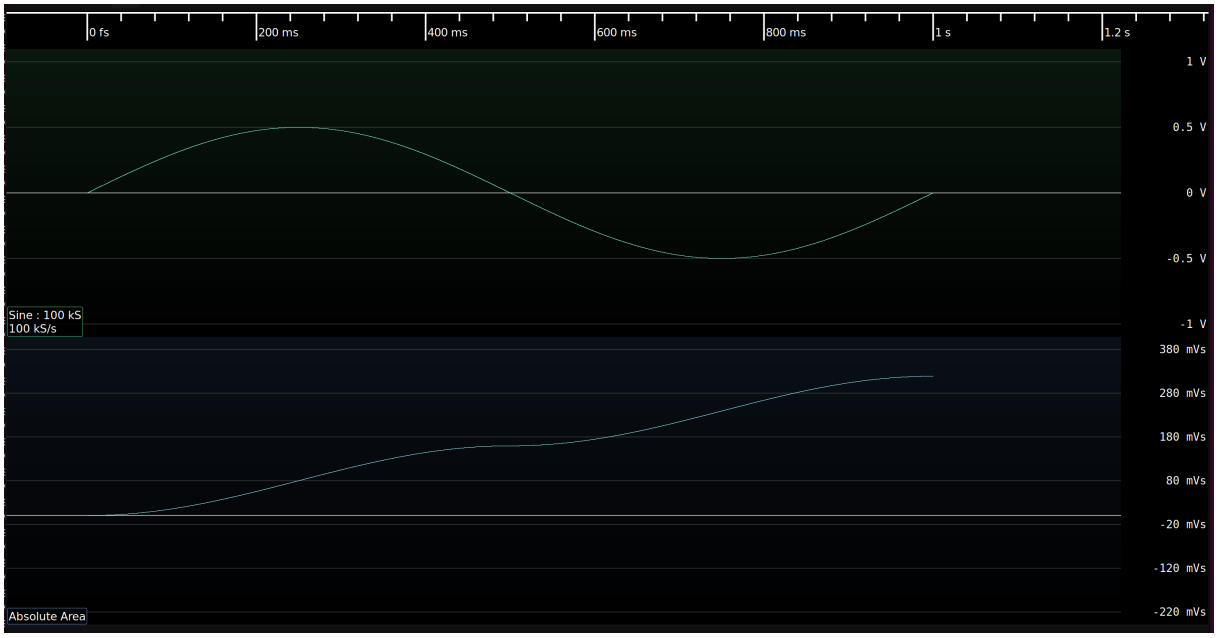

Figure 19.7: Example of absolute area under the curve measurement

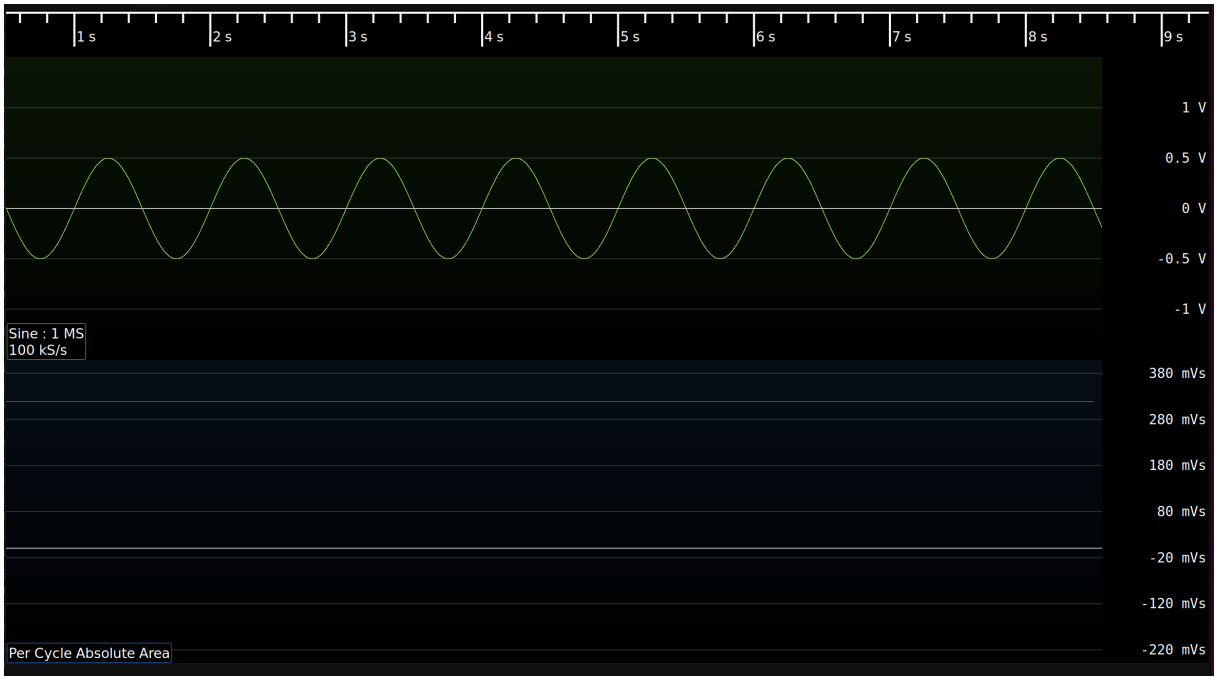

Figure 19.8: Example of per-cycle absolute area under the curve measurement

19.9.1 Inputs

Signal name	Type	Description
din	Analog	Input waveform

19.9.2 Parameters

Parameter name	Type	Description
Measurement Type	Enum	**Full Record**: Measure the area of entire waveform **Per Cycle**: Measure the area of each cycle in the waveform
Area Type	Enum	**True Area**: Consider area below ground as negative **Absolute Area**: Consider area below ground as positive

19.9.3 Output Signal

For full record measurement, this filter outputs a waveform indicating total area measured till the time on the waveform. For per cycle measurement, this filter outputs waveform representing area of each cycle.

19.10 ADL5205

Decodes SPI data traffic to one half of an ADL5205 variable gain amplifier.

TODO: Screenshot

19.10.1 Inputs

Signal name	Type	Description
spi	SPI bus	The SPI data bus

19.10.2 Parameters

This filter takes no parameters.

19.10.3 Output Signal

This filter outputs one ADL5205 sample object for each write transaction, formatted as “write: FA=2 dB, gain=8 dB".

19.11 Autocorrelation

This filter calculates the autocorrelation of an analog waveform. Autocorrelation is a measure of self-similarity calculated by multiplying the signal with a time-shifted copy of itself. In Fig. 19.9, strong peaks can be seen at multiples of the 8b/10b symbol rate.

For best performance, it is crucial to keep the maximum offset as low as possible, since filter run time grows linearly with offset range.

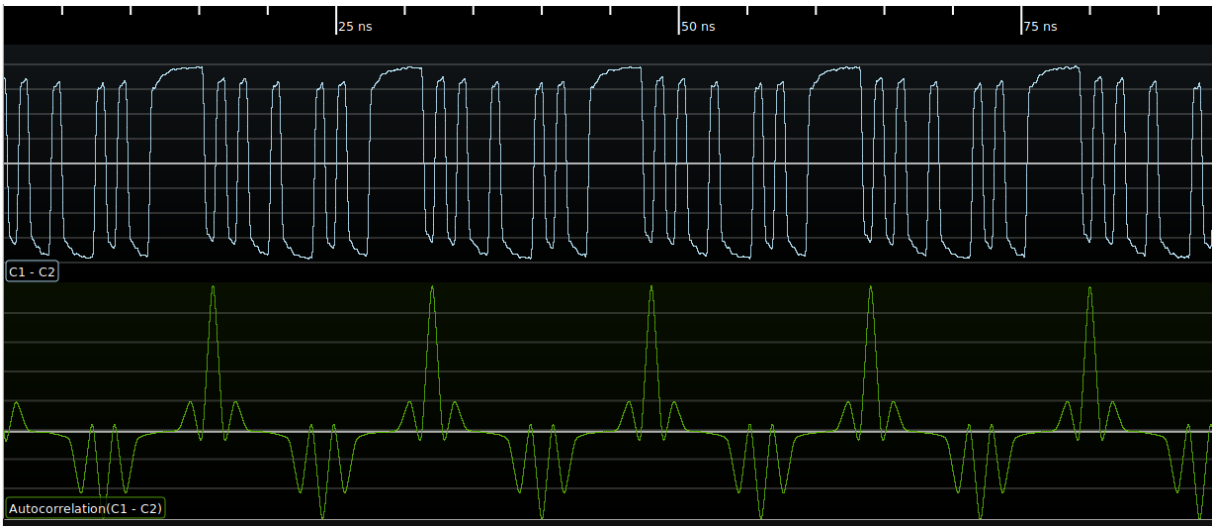

Figure 19.9: Example of autocorrelation on a serial data stream

19.11.1 Inputs

Signal name	Type	Description
din	Analog	Input waveform

19.11.2 Parameters

Parameter name	Type	Description
Max offset	Integer	Maximum shift (in samples)

19.11.3 Output Signal

This filter outputs an analog waveform with the same timebase as the input, one sample for each correlation offset.

19.12 Bandwidth

Calculates the -3 dB bandwidth of a network, given the insertion loss magnitude.

The bandwidth is measured relative to a user-specified reference level; for example the bandwidth of a -20 dB attenuator can be measured by setting the reference level to -20 dB.

19.12.1 Inputs

Signal name	Type	Description
din	Analog	Input waveform (typically S21)

19.12.2 Parameters

Parameter name	Type	Description
Reference Level	Float	Nominal (DC / mid band) insertion loss of the network

19.12.3 Output Signal

This filter outputs a scalar containing the first frequency in the network which is at least -3 dB below the reference level. If the input waveform is entirely below this level, the lowest frequency in the input is returned. If the input waveform is entirely above this level, the highest frequency in the input is returned.

19.13 Base

TODO: needs to be updated when we port to scalar interface

Calculates the base (logical zero level) of each cycle in a digital waveform.

It is most commonly used as an input to statistics, to view the average base of the entire waveform. At times, however, it may be useful to view the base waveform. For example, in Fig. 19.10, the vertical eye closure caused by channel ISI is readily apparent.

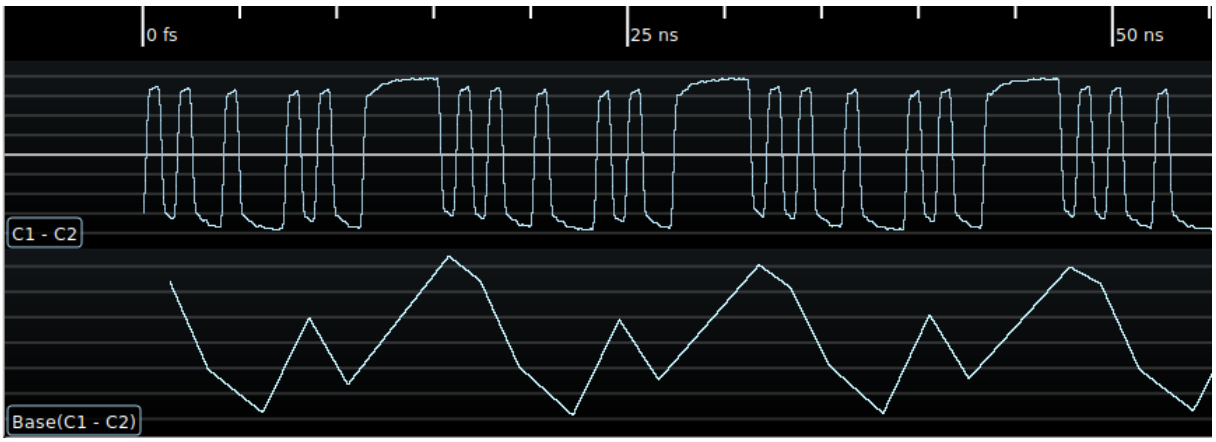

Figure 19.10: Example of base measurement on a serial data stream

19.13.1 Inputs

Signal name	Type	Description
din	Analog	Input waveform

19.13.2 Parameters

This filter takes no parameters.

19.13.3 Output Signal

This filter outputs an analog waveform with one sample for each group of logical zeroes in the input signal, containing the average value of the zero level for the middle 50% of the low period.

19.14 BIN Import

Loads an Agilent / Keysight / Rigol binary waveform file.

19.14.1 Inputs

This filter takes no inputs.

19.14.2 Parameters

Parameter name	Type	Description
BIN File	Filename	Path to the file being imported

19.14.3 Output Signal

This filter outputs a uniformly sampled analog waveform for each channel in the file. The number of output streams is variable based on how many channels are present in the file.

19.15 Burst Width

Measures the burst width of each burst in a waveform. A Burst is a sequence of adjacent crossings of the mid level reference of the waveform. Burst width is the duration of this sequence. Bursts are separated by a user-defined idle time that can be provided as a parameter to this filter. The measurement is made on each burst in the waveform.

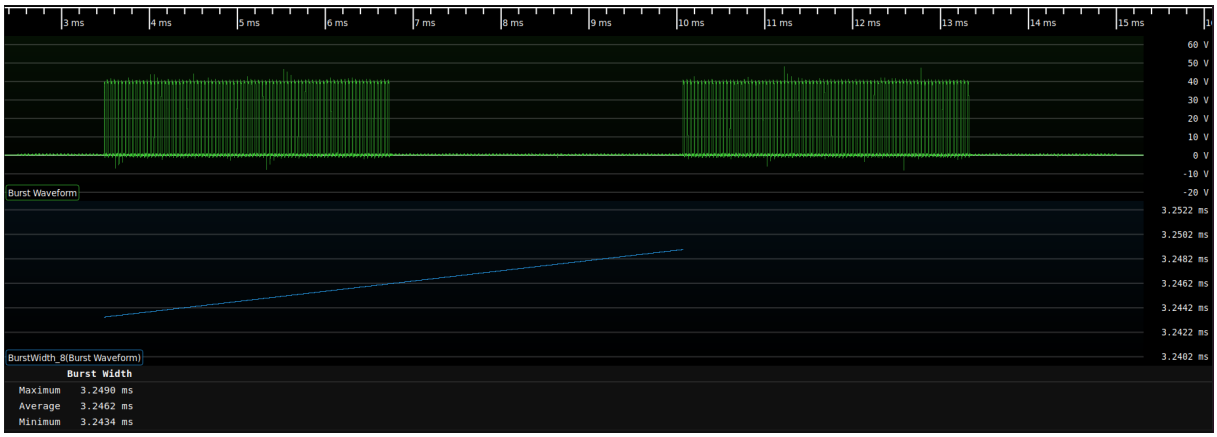

Figure 19.11: Example of burst width measurement

19.15.1 Inputs

Signal name	Type	Description
din	Analog	Input waveform

19.15.2 Parameters

Parameter name	Type	Description
Idle Time	Integer	Minimum idle time with no toggles, before declaring start of a new burst

19.15.3 Output Signal

This filter outputs an analog waveform with one sample for each burst in the input signal.

19.16 CAN

Decodes the Control Area Network (CAN) bus, commonly used in vehicle control systems. Both standard (11 bit) and extended (29 bit) IDs are supported.

CAN-FD frames are detected and flagged as such, but the current decode cannot parse them fully. Full support is planned (scopehal:334).

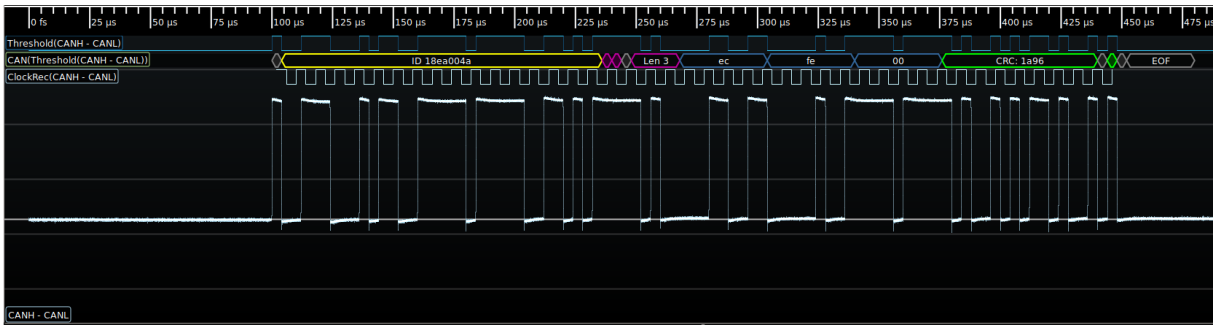

Figure 19.12: Example of CAN bus protocol decode

19.16.1 Inputs

Signal name	Type	Description
CANH	Digital	Thresholded CANH (or CANH-CANL) signal

19.16.2 Parameters

Parameter name	Type	Description
Bit Rate	Integer	Bit rate of the bus (most commonly 250 or 500 Kbps)

19.16.3 Output Signal

The CAN bus decode outputs a time series of CAN sample objects. These consist of a type field and a byte of data.

Type	Description	Color	Format
Control	Start of frame	Preamble	SOF
ID	CAN ID	Address	ID %x
RTR	Remote Transmission Request	Control	DATA | REQ
FD mode	CAN-FD mode	Control	FD | STD
R0	Reserved bits	Preamble	RSVD
DLC	Data Length Code	Control	Len 3
Data	Payload data	Data	%02x
Valid CRC	Good checksum	Checksum OK	CRC: %04x
Invalid CRC	Bad checksum	Checksum Bad	CRC: %04x
CRC delimiter	Bus turnaround	Preamble	CRC DELIM
ACK	Acknowledgement	Checksum OK	ACK
NAK	Missing acknowledgement	Checksum Bad	NAK
ACK delimiter	Bus turnaround	Preamble	ACK DELIM
EOF	End of frame	Preamble	EOF

19.17 Channel Emulation

This filter models the effects of applying an arbitrary channel, described via a single path of a set of S-parameters, to a waveform. Fig. 19.13 shows the result of passing a 1.25 Gbps serial data pattern through S21 of a 10x oscilloscope probe with approximately 500 MHz bandwidth. The ISI, attenuation, and phase shift introduced by the channel can all be seen.

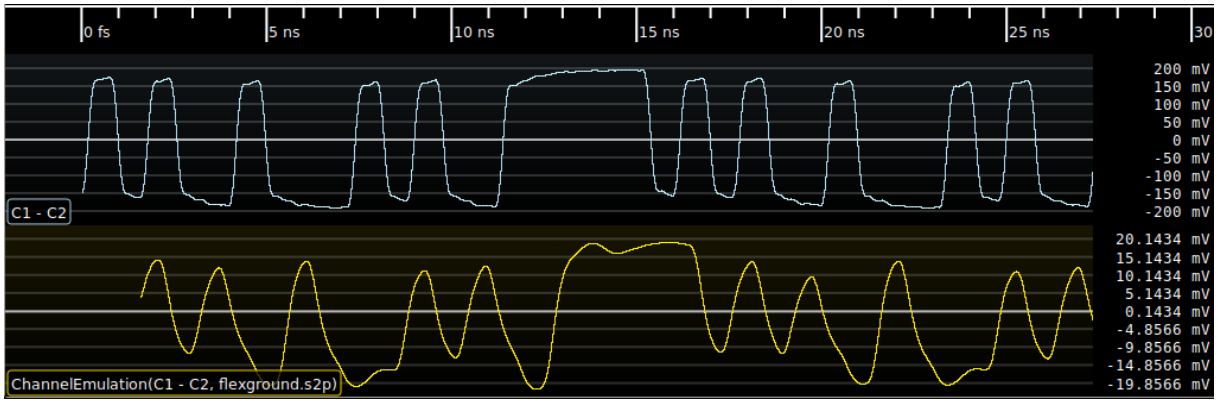

Figure 19.13: Example of channel emulation on a serial data stream

The channel model works in the frequency domain. An FFT is performed on the input, then each complex point is scaled by the interpolated magnitude and rotated by the phase, then an inverse FFT is used to transform the signal back into the time domain.

The group delay of the channel is then estimated and samples are discarded from the beginning of the waveform to prevent causality violations. For example, when performing channel emulation using a network with a 1ns group delay, the output waveform will begin 1ns after the input (since the channel output before this depends on input samples before the start of the waveform). Note that the automatic group delay estimation uses points from roughly the center of the S-parameter dataset in the current implementation; channels which do not have a significant passband around this frequency will give incorrect group delay estimates. The “Group Delay Truncation Mode" parameter can be set to manual in this case, selecting the “Group Delay Truncation" parameter instead of the automatically estimated value.

By choosing appropriate stimulus waveforms and S-parameter paths, many different kinds of analysis can be performed. For example, given a 4-port network describing two transmission lines (with ports 1 and 3 as input, and 2 and 4 as output):

- Applying S_{11} to a step or impulse waveform gives TDR response of the port 1-2 channel.
- Applying S_{21} to an impulse waveform gives impulse response of the port 1-2 channel
- Applying S_{21} to a serial data stream gives the port 1-2 signal as it would be seen by a receiver
- Applying S_{31} to a serial data stream gives the NEXT between the port 1-2 and 3-4 channels
- Applying S_{41} to a serial data stream gives the FEXT between the port 1-2 and 3-4 channels

Note that only the single S-parameter path provided is considered, and reflections elsewhere in the system are not modeled. As a result, multiple applications of this filter to emulate a large circuit piecewise (for example, a cable followed by a fixture) may give inaccurate results since reflections between the two networks are not considered. In this situation, it is preferable to use a circuit simulator or the S-Parameter Cascade filter to calculate combined S-parameters of the entire circuit and then perform the channel emulation once.

19.17.1 Inputs

Signal name	Type	Description
signal	Analog	Input waveform
mag	Analog	S-parameter magnitude channel
ang	Analog	S-parameter angle channel

19.17.2 Parameters

Parameter name	Type	Description
Max Gain	Float	Maximum gain to apply
Group Delay Truncation	Int	Group delay override for manual mode
Group Delay Truncation Mode	Enum	Specifies manual or automatically estimated group delay

19.17.3 Output Signal

This filter outputs an analog waveform with the same timebase as the input, with the emulated channel applied.

19.18 Clip

This filter limits the maximum or minimum value of a waveform to a given value. It can be configured to clip “above" in which case it imposes an upper limit or “below" in which case it imposes a lower limit.

19.18.1 Inputs

Signal name	Type	Description
din	Analog	Input waveform

19.18.2 Parameters

Parameter name	Type	Description
Behavior	Enum	Select between clipping values above or below selected value
Level	Float	Maximum/minimum signal level

19.18.3 Output Signal

This filter outputs an analog waveform with the same timebase as the input, clipped as specified by the parameters.

19.19 Clock Recovery (D-PHY HS Mode)

Extracts a double-rate clock from a MIPI D-PHY clock+data stream, which is gated to only toggle when the data input is in HS mode. This can be used for generating eye patterns of the HS-mode data.

19.20 Clock Recovery (PLL)

This filter uses a PLL to recover a clock from a serial data stream. The recovered clock is double-rate and phased 90°with respect to the data, such that the data can be sampled directly by both edges of the PLL output clock.

When the optional clock gating input is low, the output does not toggle and any edges in the input signal are ignored. As soon as the gate goes high, the PLL will phase shift the internal NCO to align with the next transition in the input signal and then begin running closed-loop.

NOTE: The current edge detector uses a single threshold suitable for NRZ inputs. When using a multi-level modulation such as PAM-4 or MLT-3, set the threshold to the highest or lowest crossing level. This will work fine for MLT-3 but introduces some data-dependent jitter in PAM signals (since the slew rate for an 00-11 transition is different than that for a 10-11 transition). The resulting recovered clock should still be adequate for protocol decoding, however a better edge detector will need to be implemented in order to do adequate jitter measurements on PAM waveforms. An edge detector suitable for PAM is planned (scopehal:77).

The current implementation of this filter uses a simple bang-bang control loop which is fast and provides reasonable jitter transfer performance (passing high frequency jitter but rejecting spread spectrum modulation), but does not precisely match the jitter transfer characteristics of any particular serial data standard. In the future, several standard PLL responses including the Fibre Channel golden PLL (scopehal:163) will be supported as options.

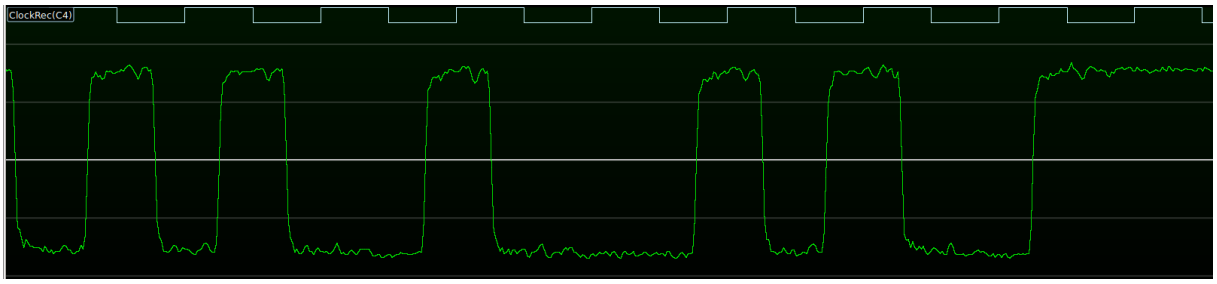

Figure 19.14: Example of CDR PLL on a serial data stream

19.20.1 Inputs

Signal name	Type	Description
IN	Analog	Input waveform
Gate	Digital	Clock enable signal, or NULL to disable gating

19.20.2 Parameters

Parameter name	Type	Description
Symbol rate	Float	Symbol rate, in Hz
Threshold	Float	Decision threshold for the edge detector, in volts

19.20.3 Output Signal

This filter outputs an digital waveform with one sample per transition of the recovered clock.

19.21 Clock Recovery (UART)

Simple DLL suitable for displaying eye patterns of RS232 and similar protocols.

19.22 Complex Import

Loads waveform data from a raw binary file containing I/Q samples in one of several formats. Regardless of sample format, the samples must be in I-Q-I-Q order.

Supported formats (native endianness, no byte swapping is performed):

- Signed int8
- Unsigned int8
- Signed int16
- Float32
- Float64

19.22.1 Inputs

This filter takes no inputs.

19.22.2 Parameters

Parameter name	Type	Description
Complex File	String	Path to the input file
File Format	Enum	Data type of the samples
Sample Rate	Int	Sampling frequency

19.22.3 Output Signal

This filter outputs two streams named “I" and “Q" containing the I/Q waveform data.

19.23 Constant

This filter outputs a scalar with a constant value, which may be used as input to other filter graph blocks.

19.23.1 Inputs

This filter takes no inputs.

19.23.2 Parameters

Parameter name	Type	Description
Value	Float	The value to output
Unit	Enum	Data type of the constant value

19.23.3 Output Signal

This filter outputs a single scalar with a constant value.

19.24 CSV Export

Saves waveform data to a comma-separated-value file.

The Update Mode parameter specifies how and when the file is modified:

- **Append (continuous):** Every time the filter graph runs, the inputs are appended to the end of the file.
- **Append (manual):** When the “Export" button in the filter properties box is clicked, the inputs are appended to the end of the file.
- **Overwrite (continuous):** Every time the filter graph runs, the input waveforms replace the current contents of the file.
- **Overwrite (manual):** When the “Export" button in the filter properties box is clicked, the input waveforms replace the current contents of the file.

19.24.1 Inputs

This filter takes a variable number of inputs, named “column1", “column2", etc, which may be of analog, digital, or arbitrary protocol type. 2D persistence maps are not supported.

19.24.2 Parameters

Parameter name	Type	Description
File name	String	Path to the CSV file
Update mode	Enum	Specifies how and when to update the file)

19.24.3 Output Signal

This filter stores its output to a file and has no filter graph output ports.

19.25 CSV Import

Loads waveform data from a comma-separated-value file.

19.26 Current Shunt

Converts a voltage waveform acquired across a known resistance into a current waveform.

19.27 DDJ

Calculates the peak-to-peak data-dependent jitter for a serial data stream.

This filter uses the non-repeating-pattern method, which allows DDJ to be computed for arbitrary waveforms rather than requiring a short, repeating PRBS. In this method, per-UI jitter (TIE) measurements are split across 2^n histogram bins, one for each possible combination of the preceding n bits. The jitter samples for each bin are then averaged to remove the effects of other jitter, leaving only the DDJ. The final DDJ value is reported as the difference between the minimum and maximum histogram bins.

The current implementation uses a fixed window size of $n = 8$ UI. If the channel has significant memory effects or reflections with delays of more than 8 UI, DDJ maybe underestimated.

The current implementation only supports NRZ signals and cannot measure DDJ for MLT3 or PAM waveforms.

19.27.1 Inputs

Signal name	Type	Description
TIE	Analog	TIE waveform computed by the TIE filter
Threshold	Digital	Thresholded digital sample values
Clock	Digital	Double rate, center aligned sampling clock for threshold values

19.27.2 Parameters

This filter takes no parameters.

19.27.3 Output Signal

This filter outputs an analog waveform with a single sample containing the computed DDJ value.

Additionally, the raw DDJ histogram is stored internally and may be accessed by other filters via the C++ API. There is currently no way to display the histogram content.

19.28 DDR1 Command Bus

Decodes the command bus for first-generation DDR SDRAM.

19.29 DDR3 Command Bus

Decodes the command bus for third-generation DDR SDRAM.

19.30 De-Embed

Applies the inverse of a channel (described by a single path in an S-parameter dataset, normally S_{21}) to a signal, in order to calculate what the waveform would have looked like at the input to a cable, fixture, etc. given the signal seen at the output.

The channel model works in the frequency domain. An FFT is performed on the input, then each complex point is scaled by the interpolated magnitude and rotated by the phase, then an inverse FFT is used to transform the signal back into the time domain.

The group delay of the channel is then estimated and samples are discarded from the end of the waveform to prevent causality violations. For example, when performing a de-embed using a network with a 1ns group delay, the output waveform will end 1ns before the input does (since the channel output after this depends on input samples after the end of the stimulus waveform). Note that the automatic group delay estimation uses points from roughly the center of the S-parameter dataset in the current implementation; channels which do not have a significant passband around this frequency will give incorrect group delay estimates. The “Group Delay Truncation Mode" parameter can be set to manual in this case, selecting the “Group Delay Truncation" parameter instead of the automatically estimated value.

Note that only the single S-parameter path provided is considered, and reflections elsewhere in the system are not modeled. As a result, multiple applications of this filter to de-embed a large circuit piecewise (for example, a cable followed by a probe) may give inaccurate results since reflections between the two networks are not considered. In this situation, it is preferable to use a circuit simulator or the S-Parameter Cascade filter to calculate combined S-parameters of the entire circuit and then perform a single de-embed.

The maximum gain the de-embed applies is capped (default 20 dB) in order to prevent amplifying noise outside the passband of the network being de-embedded.

19.30.1 Inputs

Signal name	Type	Description
signal	Analog	Input waveform
mag	Analog	S-parameter magnitude channel
ang	Analog	S-parameter angle channel

19.30.2 Parameters

Parameter name	Type	Description
Max Gain	Float	Maximum gain to apply
Group Delay Truncation	Int	Group delay override for manual mode
Group Delay Truncation Mode	Enum	Specifies manual or automatically estimated group delay

19.30.3 Output Signal

This filter outputs an analog waveform with the same timebase as the input, with the emulated channel applied.

19.31 Deskew

Moves an analog waveform earlier or later in time to compensate for trigger offsets, probe length mismatch, etc. It is generally preferable to deskew using the skew adjustment on the channel during acquisition; this filter is provided for correction in postprocessing.

19.31.1 Inputs

Signal name	Type	Description
din	Analog	Input waveform

19.31.2 Parameters

Parameter name	Type	Description
Skew	Float	Time offset to shift the waveform

19.31.3 Output Signal

This filter outputs an analog waveform with one sample for each sample in the input, phase shifted by the requested offset.

19.32 Digital to NRZ

Convert a digital signal (and associated clock) to an analog NRZ waveform. This filter uses a simplistic piecewise linear rise/fall time model: the output stays at the logic low/high voltage until the input changes, then ramps at a constant rate to then new value. For more accurate modeling of edge shape use the IBIS Driver filter with the appropriate IBIS model for your DUT.

19.32.1 Inputs

Signal name	Type	Description
data	Digital	Digital data to send
clk	Digital	Clock for data

19.32.2 Parameters

Parameter name	Type	Description
Level 0	Float	Voltage to send when the input is a logic 0
Level 1	Float	Voltage to send when the input is a logic 1
Sample Rate	Int	Sample rate for the generated waveform
Transition Time	Int	Rising and falling edge time

19.32.3 Output Signal

This filter outputs an analog NRZ version of the provided digital input, sampled uniformly at the specified rate.

19.33 Digital to PAM4

Convert a digital signal (and associated clock) to an analog PAM-4 waveform. This filter uses a simplistic piecewise linear rise/fall time model: the output stays at the current symbol's voltage until the input changes, then ramps at a constant rate to then new value. For more accurate modeling of edge shape use the IBIS Driver filter with the appropriate IBIS model for your DUT.

The input data is a digital serial bit stream at twice the PAM4 symbol rate. Two consecutive input bits map to a single PAM-4 output sample.

19.33.1 Inputs

Signal name	Type	Description
data	Digital	Serial digital data to send
clk	Digital	Clock for data

19.33.2 Parameters

Parameter name	Type	Description
Level 00	Float	Voltage to send when the input is a logic 0-0
Level 01	Float	Voltage to send when the input is a logic 0-1
Level 10	Float	Voltage to send when the input is a logic 1-0
Level 11	Float	Voltage to send when the input is a logic 1-1
Sample Rate	Int	Sample rate for the generated waveform
Transition Time	Int	Rising and falling edge time

19.33.3 Output Signal

This filter outputs an analog PAM-4 version of the provided digital input, sampled uniformly at the specified rate.

19.34 DisplayPort - Aux Channel

Decodes the Auxiliary Channel of DisplayPort

19.35 Divide

Divides one waveform by another.

19.36 Downconvert

Performs digital downconversion by mixing a directly sampled RF signal with a two-phase local oscillator, then outputs the downconverted signal. No LO rejection filtering or decimation is performed.

19.37 Downsample

Low-pass filters a signal to prevent aliasing, then decimates by an integer factor.

19.38 DRAM Clocks

Given a DRAM command bus and a DQS strobe, produce separate gated DQ clock streams for read and write bursts.

19.39 DRAM Trcd

Calculates T_{rcd} (RAS-to-CAS delay) for each newly opened row in a DRAM command bus stream.

19.40 DRAM Trfc

Calculates T_{rfc} (refresh-to-refresh delay) for each refresh operation in a DRAM command bus stream.

19.41 Duty Cycle

Calculates the duty cycle of a bimodal waveform. The duty cycle is defined as the percentage of time spent in the high state divided by the period.

19.42 DVI

Decodes Digital Visual Interface (DVI) video signals.

19.43 Emphasis

Adds pre/de emphasis to a signal.

19.44 Emphasis Removal

Removes pre/de emphasis from a signal.

19.45 Enhanced Resolution

Applies a FIR low-pass filter to a signal to increase the vertical resolution and reduce noise at the cost of reduced bandwidth. This technique assumes a small amount of Gaussian noise is present in the input waveform, such that a signal whose true value is midway between two ADC codes will randomly fluctuate between the two quantized values, with an average equal to the true value.

Each half bit of resolution reduces the bandwidth by an additional factor of two beyond the Nyquist limit. For example, a 1.5 bit resolution improvement reduces the bandwith to Fnyquist / 8. The filter properties dialog displays the calculated -3 dB bandwidth based on the current input sample rate.

19.45.1 Inputs

Signal name	Type	Description
in	Analog	Input signal

19.45.2 Parameters

Parameter name	Type	Description
Bits	Enum	Number of additional bits of resolution to add

19.46 Envelope

Finds the minimum and maximum of each sample in the input over time, and outputs them as separate streams.

19.47 Ethernet - 10baseT

Decodes the 10base-T Ethernet PCS/PMA as specified in IEEE 802.3-2018 clause 14.

19.48 Ethernet - 100baseTX

Decodes the 100base-TX Ethernet PMA/PCS as specified in IEEE 802.3-2018 clause 24 and 25, and the ANSI X3T12 FDDI PHY.

19.49 Ethernet - 1000baseX

Decodes the 1000base-X Ethernet PCS as specified in IEEE 802.3-2018 clause 36.

Signal name	Type	Description
data	8b/10b	Output of 8b/10b protocol decode

19.49.1 Parameters

This filter takes no parameters.

19.49.2 Output Signal

The 1000base-X filter outputs a series of Ethernet frame segment objects.

Type	Description	Color	Format
Preamble	Preamble	Preamble	PREAMBLE
Preamble	Start of frame delimiter	Preamble	SFD
Address	Src/dest MAC	Address	From 02:00:11:22:33:44
Control	Ethertype	Control	Type: IPv4 Type: 0xbeef
Control	VLAN tag	Control	VLAN 10, PCP 0
Data	Frame data	Data	a5
Checksum OK	Valid FCS	Checksum OK	CRC: 0xdeadbeef
Checksum Bad	Invalid FCS	Checksum Bad	CRC: 0xbaadc0de
Error	Malformed data	Error	ERROR

TODO: Document protocol analyzer output

19.50 Ethernet - GMII

Decodes the Gigabit Media Independent Interface as specified in IEEE 802.3-2018 clause 35.

19.51 Ethernet - QSGMII

Converts a Quad SGMII data stream into four separate SGMII data streams which can be independently decoded.

19.52 Ethernet - RGMII

Decodes the Reduced Gigabit Media Independent Interface as specified in the RGMII 2.0 specification.

19.53 Ethernet - RMII

Decodes the Reduced Media Independent Interface as specified in the RMII specification.

19.54 Ethernet - SGMII

Decodes Serial GMII data at 10, 100, or 1000 Mbps rates to Ethernet frames.

19.55 Ethernet Autonegotiation

Decodes the Base-T autonegotiation signaling for Ethernet as specified in IEEE 802.3-2018 clause 28.

This filter outputs a stream of 16-bit negotiation codewords, which is typically fed to the Ethernet Autonegotiation Page filter.

19.56 Ethernet Autonegotiation Page

Decodes a stream of 16-bit negotiation codewords to ability values, as specified in IEEE 802.3-2018 annex 28A, 28B, and 28C.

Note that the autonegotiation protocol is stateful, so it is not possible to definitively decode a single code word or small group of them in isolation. For accurate decoding, the input waveform should start with the Base Page (sent during the link-down state before a link partner has been detected).]

19.57 Ethernet Base-X Autonegotiation

Decodes the Base-X autonegotiation signaling for Ethernet as specified in IEEE 802.3-2018 clause 37.

Also supports the extended autonegotiation used by SGMII.

19.58 Eye Bit Rate

Measures the bit rate of an eye pattern.

19.59 Eye Height

Measures the vertical opening of an eye pattern.

19.60 Eye P-P Jitter

Measures the peak-to-peak jitter of an eye pattern.

19.61 Eye Pattern

Calculates an eye pattern.

19.62 Eye Period

Measures the UI width of an eye pattern.

19.63 Eye Width

Measures the horizontal opening of an eye pattern.

19.64 Fall

Measures the fall time of each falling edge in a waveform.

19.65 FFT

Calculates a Fast Fourier Transform and displays the magnitude response.

19.66 FIR

Applies a finite-impulse-response filter to a signal.

19.67 Frequency

Measures the frequency of each cycle in a waveform.

19.68 FSK

Converts a frequency-vs-time waveform (typically generated by the Vector Frequency filter either directly or through a denoising filter) to a digital waveform. As of now, only BFSK is supported.

The filter calculates a histogram of the input signal each waveform, expecting a bimodal distribution. The two highest histogram peaks are selected as the nominal logic 0 and 1 levels, with the higher frequency assigned to logic 1 and the lower to logic 0.

Thresholding is performed at the midpoint of the nominal 0 and 1 levels, with hysteresis equal to 20% of the difference between the nominal levels. Using adaptive thresholds allows the filter to automatically track frequency-hopping systems as long as only one packet is present in each waveform.

TODO: re-histogram any time we break squelch?

19.69 Full Width at Half Maximum

Calculates the full width at the half of maximum value of all peaks in a signal.

Figure 19.15: Example of full width at half maximum of a Sinewave input waveform.

19.69.1 Inputs

Signal name	Type	Description
din	Analog	Input waveform

19.69.2 Parameters

Parameter name	Type	Description
Peak Threshold	Float	Pulses with peak values below this threshold are not considered

19.69.3 Output Signal

This filter outputs two analog waveforms. One shows the value of full width at half maximum value of all the peaks in the signal. Another output waveform shows the amplitude of all the corresponding peaks.

19.70 Glitch Removal

This filter removes 'glitches' from a digital waveform. A Minimum Width is specified, and any 'pulse' (period during which the waveform has the same value) shorter than that pulse is ignored, the previous pulse continuing. Common use is to remove glitches from a f Hz signal by filtering pulses shorter than $\frac{1}{1.1f}$ s.

19.70.1 Inputs

Signal name	Type	Description
data	Digital	Input data.

19.70.2 Parameters

Parameter name	Type	Description
Minimum Width	Float	Minimum width of a pulse allowed through.

19.70.3 Output Signal

This filter outputs a digital waveform which has no samples shorter than Minimum Width. The output waveform does not have any samples until the first pulse of at least Minimum Width, and the last state continues to the end of the waveform.

19.71 Group Delay

Calculates the group delay of a phase-vs-frequency waveform, $\frac{d\phi}{d\omega}$.

19.71.1 Inputs

Signal name	Type	Description
Phase	Analog	Phase angle vs frequency

19.71.2 Parameters

This filter takes no parameters.

19.71.3 Output Signal

This filter outputs an analog waveform with one sample per frequency point, containing the group delay at that frequency.

19.72 Histogram

Computes a histogram from incoming data. Histogram counts are accumulated across multiple processed waveforms and cleared on "Clear Sweeps." Number of histogram bins is determined from the bin size parameter and the max/min values configured. Default behavior is to autorange the input and have 100fs bins. Samples outside a configured manual range will fall into the highest/lowest bin and the "CLIPPING" flag will be set on the output waveform.

19.72.1 Inputs

Signal name	Type	Description
data	Analog	Input data. Usually in units of fs.

19.72.2 Parameters

Parameter name	Type	Description
Autorange	Bool	If the filter should automatically range the maximum and minimum bins
Min Value	Float	Lower end of the lowest bin when Autorange disabled
Max Value	Float	Higher end of the highest bin when Autorange disabled
Bin Size	Float	Size of a bin. Number of bins is determined from this and max/min values

19.72.3 Output Signal

This filter outputs an analog waveform with one sample per bin and a value in counts. The "CLIPPING" flag on a waveform indicates that input samples fell outside the configured range of bins (when not using Autoranging.)

19.73 Horizontal Bathtub

Calculates a bathtub curve across a horizontal slice through an eye pattern.

19.74 HDMI

Decodes HDMI

19.75 I^2C

Decodes the Phillips I^2C bus protocol.

19.76 I^2C EEPROM

Decodes common I^2C EEPROM memory devices

19.77 I^2C Register

Decodes low level I^2C bus traffic into a series of register read-write transactions targeting a specific device address.

This filter assumes that the device has a fixed sized address pointer. Register writes consist of a write to the device's address, the register address, then write data. Reads consist of a write to the device's address, the register address, a read from the device's address, and read data.

19.78 IBIS Driver

Converts a digital waveform and double-rate clock to an analog waveform using the rising and falling edge waveforms from an IBIS model.

This filter assumes a perfect 50Ω load or other matched load as specified in the IBIS model; clamp behavior of the driver in response to channels with significant reflection is not currently modeled.

IBIS-AMI is not currently supported, however this is planned (scopehal:192).

Model name and termination conditions are dynamically created enumerations; the set of legal values for these fields depends on the specific .ibs file loaded.

Note that IBIS corners specify minimum, typical, or maximum *output voltage*, not timing or other properties.

19.78.1 Inputs

Signal name	Type	Description
data	Digital	Digital waveform to transmit
clk	Digital	Transmit clock (double rate)

19.78.2 Parameters

Parameter name	Type	Description
Corner	Enum	Name of the corner to use
File Path	String	Filesystem path to the IBIS model
Model Name	Enum	Name of the I/O cell model within the IBIS model to use
Sample Rate	Int	Sample rate to use for the output waveform
Termination	Enum	Name of the termination condition to use

19.78.3 Output Signal

This filter outputs an analog waveform containing uniformly spaced samples at the specified rate.

19.79 Invert

Inverts an analog waveform by negating each sample.

19.80 Intel eSPI

Decodes the Enhanced Serial Peripheral Interface protocol, used between Intel CPUs and peripherals such as baseboard management controllers (BMCs) and embedded controllers (ECs).

19.81 IPv4

Internet Protocol version 4

19.82 IQ Squelch

Gates I/Q data to eliminate noise between packets. Signal regions with amplitude below the squelch threshold are replaced with an equal number of zero-valued samples.

19.83 Jitter

Adds random and/or periodic jitter to a digital waveform by displacing each sample.

Random jitter is unbounded and has a Gaussian distribution with a user-specified standard deviation. Periodic jitter is sinusoidal and has a bounded range of -1 to +1 times the specified amplitude. Only a single frequency of Pj is supported, however several instances of this filter may be chained in order to inject Pj at multiple frequencies. The starting phase of the Pj sinusoid is random.

19.83.1 Inputs

Signal name	Type	Description
din	Digital	Input waveform

19.83.2 Parameters

Parameter name	Type	Description
Rj Stdev	Float	Standard deviation of random jitter
Pj Frequency	Float	Frequency of periodic jitter
Pj Amplitude	Float	Amplitude of periodic jitter

19.83.3 Output Signal

This filter outputs a digital waveform with one sample per sample in the input waveform, with sample time shifted by the sum of random and periodic jitter terms. The output waveform will have 1fs timebase resolution and not be dense packed, regardless of the input timebase configuration.

19.84 Jitter Spectrum

Calculates an FFT of a TIE waveform.

19.85 JTAG

Joint Test Action Group

19.86 Magnitude

Calculates the magnitude of a complex valued signal

19.87 MDIO

Decodes the Management Data Input/Output interface on Ethernet PHYs. At the moment, only Clause 22 format is supported.

19.88 Memory

Takes a snapshot of the input which remains “frozen" until manually updated. Typically used for comparing past and present values of a signal on the same plot.

19.89 MIL-STD-1553

Decodes the MIL-STD-1553 avionics data bus.

19.90 MIPI D-Phy Data

Converts two streams of D-Phy Symbols (one data and one clock) into bytes and control events.

Only a single data lane is supported at the moment, but multi-lane support will be added in the future.

This filter only supports high speed data; escape mode data is handled by the D-PHY Escape Mode filter.

19.91 MIPI D-Phy Escape Mode

Converts a stream of D-PHY Symbols for a data lane into low-power data.

19.92 MIPI D-Phy Symbol

Decodes one or two analog channels to MIPI D-PHY symbols (HS/LS line states). Either the positive half, or both positive and negative, of the pair may be provided.

If only the positive half is provided, it is possible to decode HS data and clocks, but not the LP-01 and LP-10 states, as these are indistinguishable from LP-00 and LP-11. This prevents proper decoding of Escape Mode data, although Start-Of-Transmission sequences may be inferred from context.

19.93 MIPI DSI Frame

Converts a MIPI DSI Packet stream into video scanlines.

19.94 MIPI DSI Packet

Converts two streams of D-Phy Symbol's (one data and one clock) into MIPI DSI packets.

19.95 Moving Average

Calculates a moving average (box filter) over an analog waveform.

19.96 Multiply

Multiplies one waveform by another. No resampling is performed; both inputs must have identical sample rates.

Unit conversions are performed, for example the product of a voltage and current waveform is a power waveform.

19.97 Noise

Adds Gaussian noise with a specified standard deviation to a waveform.

19.98 OFDM Demodulator

NOTE: this filter is still under development and not suitable for general use.

19.99 Overshoot

19.100 PAM4 Demodulator

Converts an analog PAM4 waveform and recovered clock into a digital serial waveform and recovered clock at twice the symbol rate. This allows conventional NRZ protocol decodes to be applied to a PAM4 data stream.

Gray coding is assumed, as used by all major PAM-4 networking standards.

19.101 Parallel Bus

19.102 PCIe Data Link

Decodes the Data Link layer of PCI Express. At this layer DLLPs are fully decoded. TLP sequence numbers are visible and CRC16s are checked, however TLP content is displayed as hex dumps.

19.103 PCIe Gen 1/2 Logical

Decodes the Logical Sub-Block of the PCI Express 1.0 and 2.0 PHY. This layer decodes 8B/10B symbols and the LFSR scrambler. TLP and DLLP start/end markers are identified but no packet decoding is performed.

19.104 PCIe Gen 3/4/5 Logical

Decodes the Logical Sub-Block of the PCI Express 3.0, 4.0, and 5.0 PHY. This layer converts 128b/130b symbols into a stream of protocol packets and content. TLP and DLLP start/end markers are identified but no packet decoding is performed.

19.105 PCIe Link Training

Decodes the initial PCIe gen1/2 link training sequence

19.106 PCIe Transport

Decodes the Transport layer of PCI Express. At this layer TLPs are fully decoded, however only a unidirectional view of the system is visible (only TX or only RX).

19.107 Peak Hold

19.108 Peak-to-Peak

19.109 Period

19.110 Phase

Displays the relative phase of a signal as a function of time. Typically used for visualizing PSK modulations.

19.111 Phase Nonlinearity

Given a phase angle waveform, outputs the difference between the actual phase and linear phase. A perfectly linear network will be displayed as a horizontal line at Y=0; leading or lagging phase will show up as spikes above or below zero.

The nominal linear phase response is calculated based on the averge group delay between two user-supplied frequencies. Moving the reference frequencies further apart reduces the impact of phase noise in the data (since more points are being averaged) however both points must be located well within the linear region of the network in order to give accurate results.

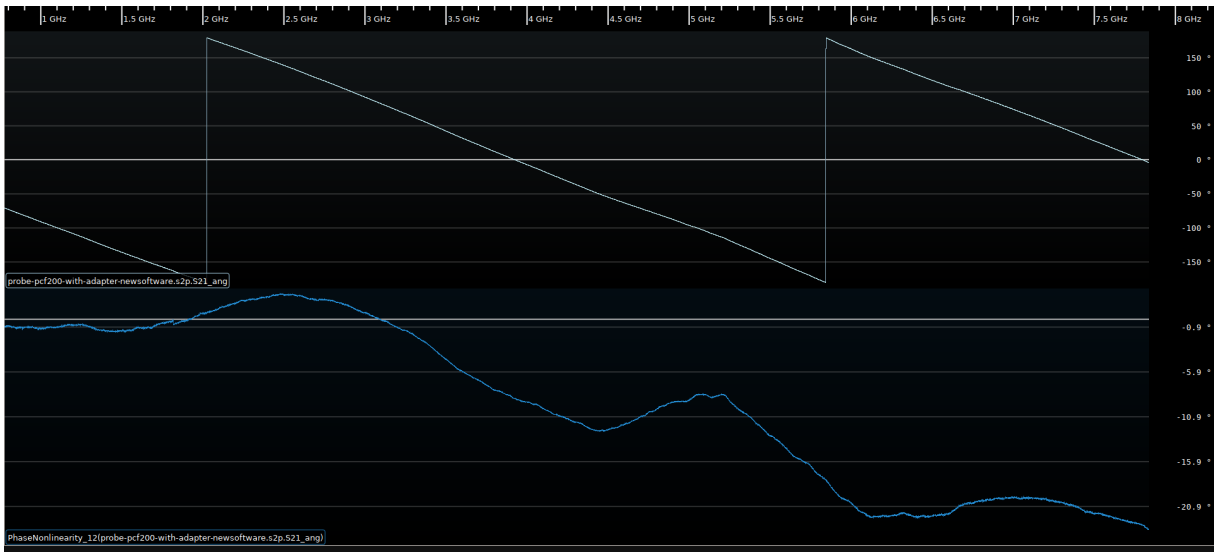

Figure 19.16: Example of nonlinear phase of a filter in the stopband

19.111.1 Inputs

Signal name	Type	Description
Phase	Analog	Input waveform

19.111.2 Parameters

Parameter name	Type	Description
Ref Freq Low	Float	Lower reference frequency
Ref Freq High	Float	Upper reference frequency

19.111.3 Output Signal

This filter outputs an analog waveform with one sample for each sample in the input, containing the deviation from linear phase.

19.112 PRBS

Generates a pseudorandom bit sequence, and double rate bit clock, with a specified bit rate from a list of standard polynomials.

19.113 Pulse Width

This filter measures the length of pulses and outputs that as a waveform. It auto-thresholds analog inputs at 50%.

19.113.1 Inputs

Signal name	Type	Description
din	Analog	Input waveform

19.113.2 Output Signal

This filter outputs an sparse analog waveform with the same timebase as the input, containing one sample per pulse with a duration and value equal to the length of the pulse.

19.114 Reference Plane Extension

Given a set of S-parameters, shifts the reference plane on one or two ports and outputs a new set of S-parameters.

19.115 Rj + BUj

Removes data-dependent jitter (DDJ) from a TIE waveform, leaving uncorrelated jitter (Rj and BUj).

19.116 QSPI

Quad SPI as used in serial Flash. Note that this filter *only* decodes quad mode streams, not x1 SPI.

19.117 Quadrature

Quadrature pulses from a rotary encoder

19.118 Rise

Calculates the rise time for each cycle of a waveform

19.119 SNR

Computes simple $\frac{\mu}{\sigma}$ (mean over standard deviation) signal-to-noise ratio for the input signal.

19.119.1 Inputs

Signal name	Type	Description
in	Analog	Input Waveform

19.119.2 Parameters

This filter takes no parameters.

19.119.3 Output Signal

This filter outputs a scalar value representing the $\frac{\mu}{\sigma}$ SNR for the whole waveform. For sparse waveforms samples are weighted by length and gaps are not considered.

19.120 S-Parameter Cascade

Cascades two two-port networks and outputs a two-port network equivalent to the two input networks in series.

19.121 S-Parameter De-Embed

Given a two port network equal to the cascade of two others, plus S-parameters for one of the two sub-networks, output S-parameters for the other.

19.122 Scalar Stairstep

Outputs a scalar value which ramps from a starting value to an ending value in a stairstep pattern, with configurable step duration and spacing.

19.123 Scale

Multiplies a waveform by a scalar.

19.124 SD Card Command

Decodes the Secure Digital card command bus protocol

19.125 Sine

Generates a pure sine wave with specified frequency, amplitude, sample rate, and DC bias.

19.126 Spectrogram

Displays a 2D plot of frequency vs time using configurable FFT length.

19.127 SPI

Serial Peripheral Interface.

19.128 SPI Flash

Flash memory attached to a SPI or quad SPI bus. Typically these chips have part numbers that start with “25".

19.129 Squelch

Detects periods with no signal.

19.130 Step

Generates a single step from one voltage level to another. Typically used for measuring step response of a channel or doing TDR transforms on S-parameters.

19.131 Subtract

Subtracts one waveform from another. No resampling is performed; both inputs must have identical sample rates.

19.131.1 Inputs

Signal name	Type	Description
IN+	Analog	Positive input waveform
IN-	Analog	Negative input waveform

19.131.2 Parameters

This filter takes no parameters.

19.131.3 Output Signal

This filter outputs an analog waveform with one sample for each sample in the input, containing the difference of the two input waveforms.

19.132 SWD

The Serial Wire Debug protocol between a Debug Probe and an ARM Microcontroller, typically from the CORTEX-M family. This decode recognises all SWD frame elements and validates type and parity of both incoming and outgoing messages. It also identifies line resets and line protocol change messages.

The SWD Protocol defines that the target will read and write on the rising edge of SWCLK. It does not place any constraint on when the probe reads and writes. For the purposes of graphical depiction each protocol element starts at a falling edge and continues to be valid until the next falling edge, following the graphical convention established in the ARM documentation.

Reference: ARM Debug Interface v5 Architecture Specification, Chapter 4.

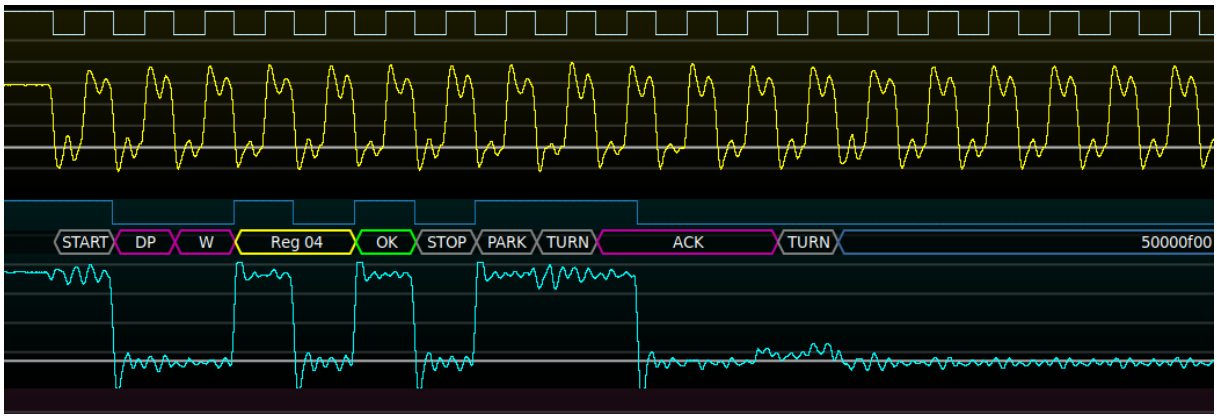

Figure 19.17: Example of SWD protocol decode

19.132.1 Inputs

Signal name	Type	Description
SWDIO	Digital	Serial Wire Data In/Out (To/From target)
SWCLK	Digital	Serial Wire Clock In (To Target from Debug Probe)

19.132.2 Parameters

No parameters are required for configuration of SWD. The protocol is clocked by SWCLK.

19.132.3 Output Signal

The SWD bus decode outputs a time series of SWD message elements, each of which may be one or a number of bits long. Each message element consist of a type and optional numeric content.

Type	Description	Color	Format
Line Control	Line Reset	Preamble	LINE RESET
Line Mode	Line Mode Change to SWD	Control	JTAG TO SWD
Line Mode	Line Mode Change to JTAG	Control	SWD TO JTAG
Line Mode	Line Mode Change to Dormant	Control	SWD TO DORMANT
Line Mode	Leave Dormant Mode	Control	LEAVE DORMANT
Start	Start of frame	Preamble	START
APnDP	Selection between AP and DP	Control	AP|DP
RnW	Read or Write mode	Control	R|W
ADDR	AP or DP Address	Address	Reg %02x
Parity	Good Header Parity	Control	OK
Parity	Bad Header Parity	Control	BAD
Stop	End of Header	Preamble	STOP
Park	Line Release	Preamble	PARK
Turnaround	Line Direction Change	Preamble	TURN
Acknowledge	Good Response from target to request	Control	ACK|WAIT
Acknowledge	Bad Response from target to request	Control	FAULT|ERROR
Data	Payload to/From Target	Data	%08x

19.133 SWD MEM-AP

Converts SWD accesses to MEM-AP registers into memory read-write transactions.

Reference: ARM Debug Interface v5 Architecture Specification, chapter 8.

19.134 Tachometer

Converts pulses from a tachometer to shaft speed

19.135 Tapped Delay Line

Generic FIR filter with arbitrary tap values and delays. Can be used as-is for testing FIR filter coefficients calculated by hand, but most commonly used as a base class for more specialized filters.

19.136 TCP

Decodes the Transmission Control Protocol (RFC 675). As of this writing, only IPv4 is supported as a network layer protocol. IPv6 support is planned once an IPv6 protocol decode has been written.

19.137 TDR

Converts a TDR waveform from volts to reflection coefficient or impedance.

19.138 TDR Step De-Embed

Given a waveform of a fast rising step, calculate the frequency response of a de-embedding network to convert the measured waveform into an ideal unit step. The resulting data can be exported to a Touchstone file.

The calculated response is typically used as input to the de-embed filter and applied to a TDR/TDT waveform generated with the same pulse generator. This correction allows for overshoot, ringing, and other artifacts on the pulse to be removed from the TDR/TDT response.

It is important that the input contain a single rising edge, and is reasonably stable before and after the edge. If multiple cycles of the test step, or falling edges, are present inaccurate results may be obtained.

NOTE: this filter is still under development and not suitable for general use.

19.139 Time Outside Level

Measures the total integrated time a signal remains above a high reference level or below a low reference level or both.

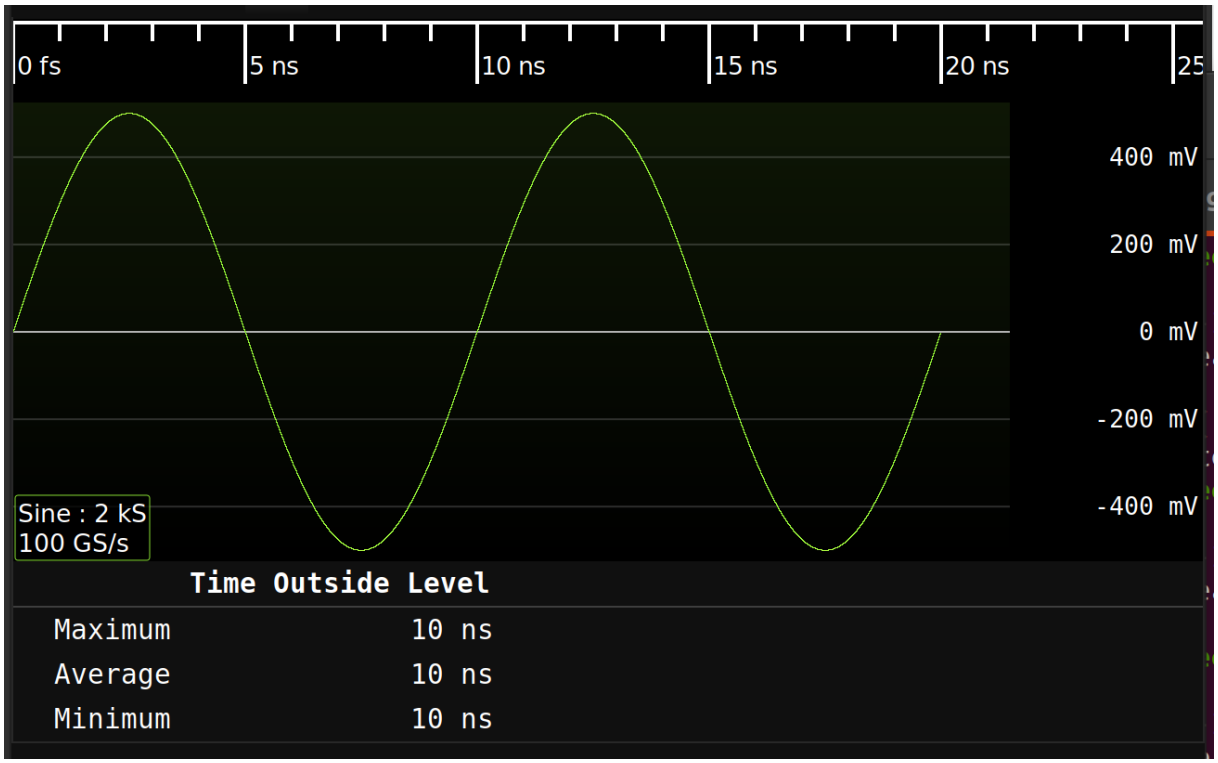

Figure 19.18: Example of time outside high level measurement with a high level threshold of 0mV

19.139.1 Inputs

Signal name	Type	Description
din	Analog	Input waveform

19.139.2 Parameters

Parameter name	Type	Description
High Level	Float	High level reference voltage
Low Level	Float	Low level reference voltage
Measurement Type	Enum	**High Level**: Measure the total time the signal is above high level reference voltage **Low Level**: Measure the total time the signal is below low level reference voltage **Both**: Measure the total time the signal is both above and below high level and low level reference voltages respectively

19.140 Thermal Diode

Converts an analog voltage measurement of a thermal diode to a temperature value

19.141 Threshold

Converts an analog waveform to digital by thresholding at a constant level (no hysteresis).

19.141.1 Inputs

Signal name	Type	Description
din	Analog	Input waveform

19.141.2 Parameters

Parameter name	Type	Description
Threshold	Float	Decision threshold

19.141.3 Output Signal

This filter outputs an digital waveform with one sample for each sample in the input, which is true if the corresponding input sample is above the threshold and false if less than or equal.

19.142 TIE

Calculates the time interval error of a data or clock signal with respect to an ideal “golden" clock (typically obtained from a CDR PLL).

19.143 Top

Calculates the top (logical one level) of each cycle in a digital waveform. It is most commonly used as an input to statistics, to view the average top of the entire waveform.

19.143.1 Inputs

Signal name	Type	Description
din	Analog	Input waveform

19.143.2 Parameters

This filter takes no parameters.

19.143.3 Output Signal

This filter outputs an analog waveform with one sample for each group of logical ones in the input signal, containing the average value of the one level.

19.144 Touchstone Export

Saves S-parameter data to a Touchstone file.

19.145 Touchstone Import

Loads a Touchstone file and displays the complex data in magnitude/angle format

19.146 Trend

Plots a trend of a scalar value over time

19.147 TRC Import

Loads waveform data from a Teledyne LeCroy TRC waveform file.

19.148 UART

19.149 Unwrapped Phase

Given a phase angle waveform which wraps within the interval $[-180°, +180°]$, unwrap the phase angle.

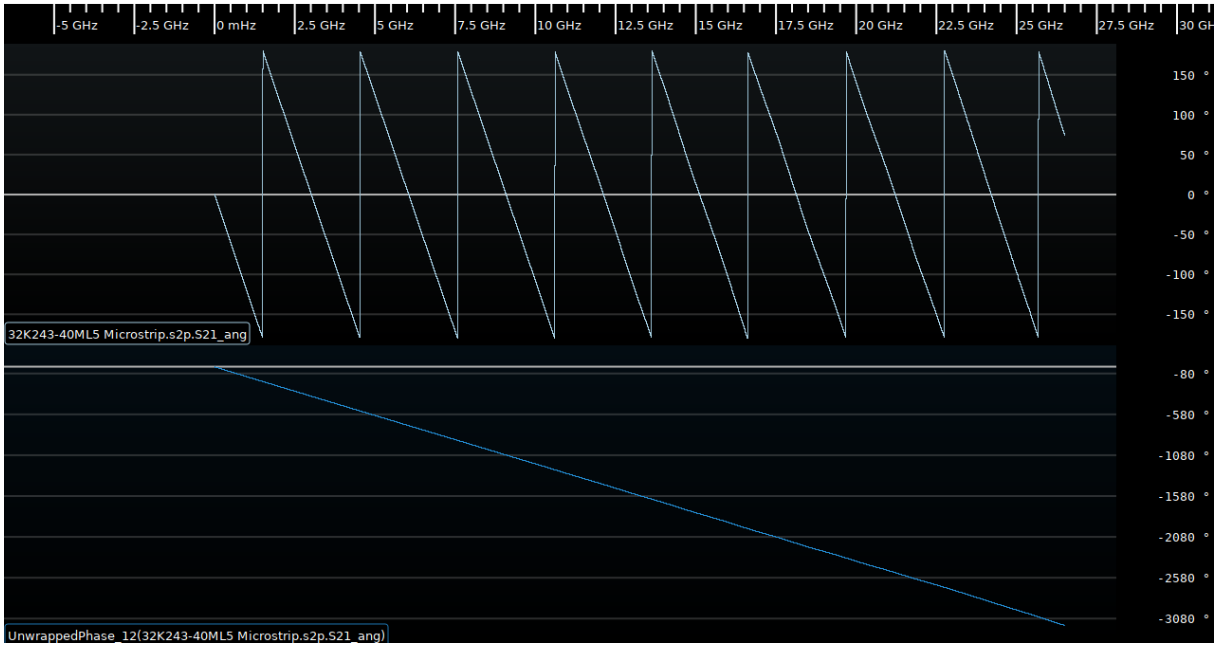

Figure 19.19: Example of wrapped and unwrapped phase of a transmission line

19.149.1 Inputs

Signal name	Type	Description
Phase	Analog	Input waveform

19.149.2 Parameters

This filter takes no parameters.

19.149.3 Output Signal

This filter outputs an analog waveform with one sample for each sample in the input, containing the unwrapped phase angle.

19.150 USB 1.0 / 2.x Activity

19.151 USB 1.0 / 2.x Packet

19.152 USB 1.0 / 2.x PCS

19.153 USB 1.0 / 2.x PMA

19.154 Undershoot

19.155 Upsample

Upsamples a waveform using sin(x)/x interpolation.

19.156 VCD Import

Loads digital waveform data from a Value Change Dump (VCD) file.

19.157 Vector Frequency

Calculates the instantaneous frequency (rotational velocity) of a complex I/Q signal.

19.158 Vector Phase

Calculates the instantaneous phase of a complex I/Q signal.

19.159 Vertical Bathtub

19.160 VICP

Decodes the Teledyne LeCroy Virtual Instrument Control Protocol (VICP)

19.161 Waterfall

19.162 WAV Import

Loads waveform data from a Microsoft WAV audio file.

19.163 WFM Import

Loads waveform data from a Tektronix .wfm file.

19.164 Windowed Autocorrelation

Calculates the cross-correlation between a fixed size block of the input signal and another block of the same size.

This will produce maximal response for a signal which has periodicity with the specified period and block size.

For example, period 4 and block size 2 will match aa**aa**.

This can be used to identify OFDM symbols.

19.165 Window

Selects a temporal subset of an input waveform. Useful for running intensive analyses only on a region of interest. Start and end times are rounded to the sample that starts at or nearest after the given time.

19.165.1 Inputs

Signal name	Type	Description
din	Analog or Digital	Input waveform

19.165.2 Parameters

Parameter name	Type	Description
Start Time	Float	Start of selected window
Duration	Float	Length of selected window

19.165.3 Output Signal

This filter outputs a subset of the input signal. If the input is sparse, so is the output and vice versa. No samples are added.