

glscopeclient Operator Manual

Andrew D. Zonenberg

March 14, 2023

Copyright ©2012-2023 Andrew D. Zonenberg and contributors.. All rights reserved.

This document may be freely distributed and modified under the terms of the Creative Commons Attribution-ShareAlike 3.0 Unported license (CC BY-SA 3.0).

Contents

1	Introduction	3
1.1	Introduction	3
1.2	Revision History	3
2	Legal Notices	5
2.1	Introduction	5
2.2	License Agreement	5
2.3	Trademarks	6
2.4	Third Party Licenses	6
2.4.1	avx_mathfun.h (zlib license)	6
3	Getting Started	7
3.1	Documentation Conventions	7
3.2	Host System Requirements	7
3.3	Instrument Support	8
3.4	Compilation	8
3.4.1	Linux	8
3.4.2	macOS	9
3.4.3	Windows	10
	Installing from the package manager	10
	Building from source	10
3.5	Running glscopeclient	12
3.5.1	Configuration arguments	12
3.5.2	Console verbosity arguments	13
3.5.3	File arguments	13
3.5.4	Instrument arguments	15
3.6	Design Philosophy	15
3.7	Troubleshooting	15
3.7.1	Corrupted or no graphical output in waveform areas	15
4	Transports	17
4.1	gpib	17
4.2	lan	17
4.3	lxi	17
4.4	null	18
4.5	twinlan	18
4.6	uart	18
4.7	usbtmc	18
4.8	vicp	18
5	Oscilloscope Drivers	21
5.1	Agilent	21
5.1.1	agilent	21

	Typical Performance (MSO6034A, LAN)	21
	Typical Performance (MSOX3104T, LAN)	22
5.2	Antikernel Labs	22
5.2.1	akila	22
5.2.2	aklabs	22
5.3	Demo	23
5.4	Digilent	23
5.4.1	digilent	23
	Typical Performance (ADP3450, USB -> LAN)	24
5.5	DreamSource Lab	24
5.5.1	dslabs	24
	Typical DSCope Performance (DSCope U3P100, USB3, localhost)	25
	Typical DSLogic Performance (DSLogic U3Pro16, USB3, localhost)	25
5.6	Enjoy Digital	25
5.7	Hantek	25
5.8	Keysight	25
5.8.1	agilent	25
5.9	Keysight DCA	25
5.10	Pico Technologies	26
5.10.1	pico	26
	Typical Performance (6824E, LAN)	26
5.11	Rigol	26
5.11.1	rigol	26
	Typical Performance (MSO5000 series, LAN)	26
5.12	Rohde & Schwarz	26
5.13	Saleae	27
5.14	Siglent	27
	Typical Performance (SDS2104X+, LAN)	27
5.15	Teledyne LeCroy / LeCroy	28
5.15.1	lecroy	28
	Typical Performance (HDO9204, VICP)	29
	Typical Performance (WaveRunner 8404M-MS, VICP)	29
5.15.2	lecroy_fwp	29
5.16	Tektronix	30
5.16.1	Note regarding "lan" transport on MSO5/6	30
	Typical Performance (MSO64, LXI, embedded OS)	30
5.17	Xilinx	30
6	Power Supply Drivers	31
6.1	GW Instek	31
6.1.1	gwinstek_gpdx303s	31
6.2	Rohde & Schwarz	31
6.2.1	rs_hmc804x	31
7	Main Window	33
7.1	Menu	33
7.1.1	File	33
7.1.2	Setup	34
7.1.3	View	34
7.1.4	Add	34
7.1.5	Window	35
7.1.6	Help	35

7.2	Toolbar	35
7.2.1	Capture buttons	35
7.2.2	History	35
7.2.3	Refresh Settings	36
7.2.4	Clear Sweeps	36
7.2.5	Fullscreen	36
7.2.6	Opacity slider	36
8	Waveform Groups	39
8.1	Managing Groups	40
9	Timeline	41
10	Triggers	43
10.1	Trigger Properties	43
10.2	Serial Pattern Triggers	43
10.3	Dropout	44
10.3.1	Inputs	44
10.3.2	Parameters	44
10.4	Edge	44
10.4.1	Inputs	44
10.4.2	Parameters	44
10.5	Glitch	44
10.6	Pulse Width	44
10.6.1	Parameters	45
10.7	Runt	45
10.7.1	Parameters	45
10.8	Slew Rate	45
10.8.1	Parameters	45
10.9	UART	45
10.9.1	Inputs	46
10.9.2	Parameters	46
10.10	Window	46
10.10.1	Parameters	46
11	Waveform Views	47
11.1	Navigation	47
11.2	Plot Area	47
11.3	Y Axis Scale	48
11.4	Channel Information Box	48
11.5	Cursors	49
11.5.1	Vertical Cursors	49
11.5.2	Horizontal Cursors	50
11.5.3	Markers	51
11.6	Overlays	51
11.7	Statistics	51
12	History View	53
12.1	Pinning	53
12.2	Labeling	54
12.3	Estimating Waveform Memory Usage	54
13	Protocol Analyzer View	55

13.1	Cursor Interaction	55
13.2	Packet Coloring	56
13.3	Filtering	56
13.3.1	Expressions	57
13.3.2	Operators	57
13.3.3	Examples of filters	57
14	Filter Graph Editor	59
15	Filters	61
15.1	Introduction	61
15.1.1	Key Concepts	61
15.1.2	Conventions	61
15.2	128b/130b	63
15.3	64b/66b	64
15.3.1	Inputs	64
15.3.2	Parameters	64
15.3.3	Output Signal	64
15.4	8B/10B (IBM)	65
15.4.1	Inputs	65
15.4.2	Parameters	65
15.4.3	Output Signal	65
15.5	8B/10B (TMDS)	66
15.5.1	Inputs	66
15.5.2	Parameters	66
15.5.3	Output Signal	66
15.6	AC Couple	67
15.6.1	Inputs	67
15.6.2	Parameters	67
15.6.3	Output Signal	67
15.7	AC RMS	68
15.7.1	Inputs	68
15.7.2	Parameters	68
15.7.3	Output Signal	68
15.8	Area Under Curve	69
15.8.1	Inputs	70
15.8.2	Parameters	70
15.8.3	Output Signal	70
15.9	ADL5205	71
15.9.1	Inputs	71
15.9.2	Parameters	71
15.9.3	Output Signal	71
15.10	Autocorrelation	72
15.10.1	Inputs	72
15.10.2	Parameters	72
15.10.3	Output Signal	72
15.11	Base	73
15.11.1	Inputs	73
15.11.2	Parameters	73
15.11.3	Output Signal	73
15.12	BIN Import	74
15.13	Burst Width	75

15.13.1	Inputs	75
15.13.2	Parameters	75
15.13.3	Output Signal	75
15.14	CAN	76
15.14.1	Inputs	76
15.14.2	Parameters	76
15.14.3	Output Signal	76
15.15	Channel Emulation	77
15.15.1	Inputs	78
15.15.2	Parameters	78
15.15.3	Output Signal	78
15.16	Clip	79
15.16.1	Inputs	79
15.16.2	Parameters	79
15.16.3	Output Signal	79
15.17	Clock Recovery (D-PHY HS Mode)	80
15.18	Clock Recovery (PLL)	81
15.18.1	Inputs	81
15.18.2	Parameters	81
15.18.3	Output Signal	81
15.19	Clock Recovery (UART)	82
15.20	Complex Import	83
15.20.1	Inputs	83
15.20.2	Parameters	83
15.20.3	Output Signal	83
15.21	CSV Export	84
15.22	CSV Import	85
15.23	Current Shunt	86
15.24	DC Offset	87
15.24.1	Inputs	87
15.24.2	Parameters	87
15.24.3	Output Signal	87
15.25	DDJ	88
15.25.1	Inputs	88
15.25.2	Parameters	88
15.25.3	Output Signal	88
15.26	DDR1 Command Bus	89
15.27	DDR3 Command Bus	90
15.28	De-Embed	91
15.28.1	Inputs	91
15.28.2	Parameters	91
15.28.3	Output Signal	91
15.29	Deskew	92
15.29.1	Inputs	92
15.29.2	Parameters	92
15.29.3	Output Signal	92
15.30	Digital to NRZ	93
15.30.1	Inputs	93
15.30.2	Parameters	93
15.30.3	Output Signal	93
15.31	Digital to PAM4	94
15.31.1	Inputs	94

15.31.2	Parameters	94
15.31.3	Output Signal	94
15.32	Divide	95
15.33	Downconvert	96
15.34	Downsample	97
15.35	DRAM Clocks	98
15.36	DRAM Trcd	99
15.37	DRAM Trfc	100
15.38	Duty Cycle	101
15.39	DVI	102
15.40	Emphasis	103
15.41	Emphasis Removal	104
15.42	Enhanced Resolution	105
15.42.1	Inputs	105
15.42.2	Parameters	105
15.43	Envelope	106
15.44	Ethernet - 10baseT	107
15.45	Ethernet - 100baseTX	108
15.46	Ethernet - 1000baseX	109
15.46.1	Parameters	109
15.46.2	Output Signal	109
15.47	Ethernet - GMII	110
15.48	Ethernet - QSGMII	111
15.49	Ethernet - RGMII	112
15.50	Ethernet - RMII	113
15.51	Ethernet - SGMII	114
15.52	Ethernet Autonegotiation	115
15.53	Ethernet Autonegotiation Page	116
15.54	Ethernet Base-X Autonegotiation	117
15.55	Eye Bit Rate	118
15.56	Eye Height	119
15.57	Eye P-P Jitter	120
15.58	Eye Pattern	121
15.59	Eye Period	122
15.60	Eye Width	123
15.61	Fall	124
15.62	FFT	125
15.63	FIR	126
15.64	Frequency	127
15.65	FSK	128
15.66	Group Delay	129
15.66.1	Inputs	129
15.66.2	Parameters	129
15.66.3	Output Signal	129
15.67	Histogram	130
15.67.1	Inputs	130
15.67.2	Parameters	130
15.67.3	Output Signal	130
15.68	Horizontal Bathtub	131
15.69	HDMI	132
15.70	I ² C	133
15.71	I ² C EEPROM	134

15.72	I ² C Register	135
15.73	IBIS Driver	136
15.73.1	Inputs	136
15.73.2	Parameters	136
15.73.3	Output Signal	136
15.74	Invert	137
15.75	Intel eSPI	138
15.76	IPv4	139
15.77	IQ Squelch	140
15.78	Jitter	141
15.78.1	Inputs	141
15.78.2	Parameters	141
15.78.3	Output Signal	141
15.79	Jitter Spectrum	142
15.80	JTAG	143
15.81	Magnitude	144
15.82	MDIO	145
15.83	Memory	146
15.84	MIL-STD-1553	147
15.85	MIPI D-Phy Data	148
15.86	MIPI D-Phy Escape Mode	149
15.87	MIPI D-Phy Symbol	150
15.88	MIPI DSI Frame	151
15.89	MIPI DSI Packet	152
15.90	Moving Average	153
15.91	Multiply	154
15.92	Noise	155
15.93	OFDM Demodulator	156
15.94	Overshoot	157
15.95	PAM4 Demodulator	158
15.96	Parallel Bus	159
15.97	PCIe Data Link	160
15.98	PCIe Gen 1/2 Logical	161
15.99	PCIe Gen 3/4/5 Logical	162
15.100	PCIe Link Training	163
15.101	PCIe Transport	164
15.102	Peak Hold	165
15.103	Peak-to-Peak	166
15.104	Period	167
15.105	Phase	168
15.106	Phase Nonlinearity	169
15.106.1	Inputs	169
15.106.2	Parameters	169
15.106.3	Output Signal	169
15.107	PRBS	170
15.108	Pulse Width	171
15.108.1	Inputs	171
15.108.2	Output Signal	171
15.109	Reference Plane Extension	172
15.110	R _j + BU _j	173
15.111	QSPI	174
15.112	Quadrature	175

15.113	Rise	176
15.114	S-Parameter Cascade	177
15.115	S-Parameter De-Embed	178
15.116	Scale	179
15.117	SD Card Command	180
15.118	Sine	181
15.119	Spectrogram	182
15.120	SPI	183
15.121	SPI Flash	184
15.122	Squelch	185
15.123	Step	186
15.124	Subtract	187
	15.124.1 Inputs	187
	15.124.2 Parameters	187
	15.124.3 Output Signal	187
15.125	SWD	188
	15.125.1 Inputs	188
	15.125.2 Parameters	188
	15.125.3 Output Signal	188
15.126	SWD MEM-AP	190
15.127	Tachometer	191
15.128	Tapped Delay Line	192
15.129	TCP	193
15.130	TDR	194
15.131	TDR Step De-Embed	195
15.132	Time Outside Level	196
	15.132.1 Inputs	196
	15.132.2 Parameters	196
15.133	Thermal Diode	197
15.134	Threshold	198
	15.134.1 Inputs	198
	15.134.2 Parameters	198
	15.134.3 Output Signal	198
15.135	TIE	199
15.136	Top	200
	15.136.1 Inputs	200
	15.136.2 Parameters	200
	15.136.3 Output Signal	200
15.137	Touchstone Export	201
15.138	Touchstone Import	202
15.139	Trend	203
15.140	TRC Import	204
15.141	UART	205
15.142	Unwrapped Phase	206
	15.142.1 Inputs	206
	15.142.2 Parameters	206
	15.142.3 Output Signal	206
15.143	USB 1.0 / 2.x Activity	207
15.144	USB 1.0 / 2.x Packet	208
15.145	USB 1.0 / 2.x PCS	209
15.146	USB 1.0 / 2.x PMA	210
15.147	Undershoot	211

15.148	Upsample	212
15.149	VCD Import	213
15.150	Vector Frequency	214
15.151	Vector Phase	215
15.152	Vertical Bathtub	216
15.153	VICP	217
15.154	Waterfall	218
15.155	WAV Import	219
15.156	WFM Import	220
15.157	Windowed Autocorrelation	221
15.158	Window	222
15.158.1	Inputs	222
15.158.2	Parameters	222
15.158.3	Output Signal	222
16	Export Formats	223
16.1	CSV	223
16.2	Touchstone	223
17	Internals	225
17.1	Introduction	225
17.2	Instruments	225
17.3	SCPI Devices	225
17.4	Transports	226
17.5	Oscilloscopes	226
17.6	Channels	227
17.7	Streams	227
17.8	Triggers	227
17.9	Waveforms	227
17.10	Filters	228
17.11	Plugins	228
17.11.1	Linux	229
17.11.2	Windows	229

Chapter 1

Introduction

1.1 Introduction

This document is the user manual for glscopeclient, a user interface and signal analysis tool for oscilloscopes and logic analyzers. As of this writing, glscopeclient 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 Revision History

- March 14, 2023: [in progress] Initial draft

Chapter 2

Legal Notices

2.1 Introduction

glscopeclient, 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

Copyright (c) 2012-2022 Andrew D. Zonenberg and contributors. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions, and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions, and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the author nor the names of any contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHORS BE HELD LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

2.3 Trademarks

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

2.4 Third Party Licenses

TODO:

- OpenClipArt (dialog-close.png)
- yaml-cpp (shared, MIT license)
- gtkmm (shared, LGPL)
- FFTS (shared, BSD-3)
- liblxi (shared, BSD-3/EPICS)

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

NOTE: This section needs to be rewritten once the Vulkan port is complete!!

The majority of development is performed on Linux operating systems (primarily Debian and Arch), although experimental Windows support is available. `glscopeclient` uses `gtkmm` as the UI toolkit. Current development mostly uses 3.24 but any recent 3.x version should work.

Any 64-bit Intel or AMD processor should be able to run `glscopeclient`. If AVX2 and/or AVX512F support is present `glscopeclient` will use special optimized versions of some signal processing functions, however neither instruction set is required. Compiling in 32-bit mode is 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.

OpenGL 4.2 or later is required, as well as several extensions:

- `GL_ARB_arrays_of_arrays` (or OpenGL 4.3)
- `GL_ARB_compute_shader` (or OpenGL 4.3)
- `GL_ARB_shader_storage_buffer_object` (or OpenGL 4.3)
- `GL_EXT_blend_equation_separate`

The corresponding minimum hardware requirement is an AMD Radeon HD 5000, NVIDIA GeForce GTX 6xx series discrete GPU, or Intel Skylake or newer integrated GPU, plus suitably up-to-date drivers. On Linux with recent Mesa, Intel integrated GPUs as old as Ivy Bridge may be used.

TODO: what AMD integrated/ARM GPUs started supporting GL 4.2?

To check for necessary graphics card support on Linux:

```
glxinfo | grep GL_ARB_arrays_of_arrays
glxinfo | grep GL_ARB_compute_shader
glxinfo | grep GL_ARB_shader_storage_buffer_object
glxinfo | grep GL_EXT_blend_equation_separate
glxinfo | grep "OpenGL version string"
```

The minimum RAM requirement to actually launch glscopeclient is relatively small (the default demo configuration uses under 512 MB) however history mode and deep captures can easily consume many GB of RAM very rapidly. We suggest 8GB as a reasonable minimum, with 32 or more encouraged for deep history.

3.3 Instrument Support

glscopeclient uses the libscopehal library to communicate with oscilloscopes, so any libscopehal-compatible hardware should work with glscopeclient. See the [Oscilloscope Drivers](#) section for more details on which hardware is supported and how to configure specific drivers.

3.4 Compilation

glscopeclient 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.

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 -
      DCMKAKE_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 `glscopeclient` 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 glscopeclient.

Since `glscopeclient` 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 `glscopeclient` 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 glscopeclient

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

- Open a saved session (File | Open)
- 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)
- Generate a synthetic waveform (Add | Generate)

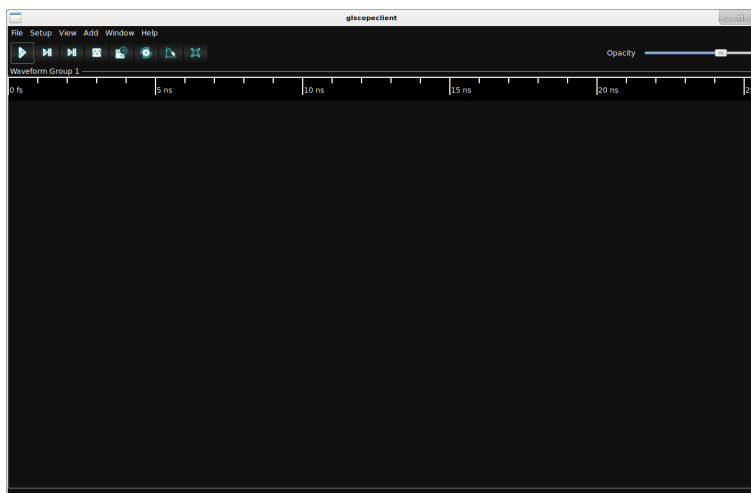


Figure 3.1: Empty `glscopeclient` session

3.5.1 Configuration arguments

Most of these arguments are intended for developers, but they can help troubleshoot unusual bugs.

- `-noavx2`
Do not use AVX2 vector optimizations even if the CPU supports it.
- `-noavx512f`
Do not use AVX512F vector optimizations even if the CPU supports it.
- `-noglint64`
Do not use `GL_ARB_gpu_shader_int64` even if the GPU supports it.
- `-nogpufilter`
Do not use Vulkan (GPU accelerated) implementations of filter blocks, revert to software fallback.

3.5.2 Console verbosity arguments

`glscopeclient` 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.
- `-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).

3.5.3 File arguments

The file extension is used to determine the format. File extensions are case sensitive and must be lowercase to be correctly interpreted.

- `[file.scopesession]` Loads a saved session.
- `[file.bin]`
Imports waveform data from the binary format used by Agilent, Keysight, and Rigol oscilloscopes.

- `[file.complex]`

Imports complex I/Q data from a file. The file must contain interleaved (I, Q) pairs in either 8-bit signed/unsigned integer, 16-bit signed integer, 32-bit normalized floating point, or 64-bit normalized floating point format.

The default format is 8 bit signed integer and may be changed from the filter graph editor or channel properties dialog once the file is loaded. There is currently no way to specify other formats on the command line.

- `[file.csv]`

Imports sample data from a CSV (comma-separated-value) file. More than one CSV file can be loaded at once (displayed as separate points in history) by specifying multiple file names as long as they have identical column schemas.

Lines starting with a '#' character are treated as comments and generally ignored by the parser. (If the comment format matches that used by Digilent's WaveForms utility, timestamps and other metadata are extracted from the comments.)

If the first row of the CSV contains non-numeric characters, it is treated as a header row. Header content in the timestamp column is ignored; headers in other columns are used as channel names in the imported waveform.

The first column of the CSV must contain sample timestamps, in seconds. Scientific notation is supported. Timestamps must be monotonic (each row must have a timestamp strictly greater than that of the previous row).

glscopeclient uses a heuristic to detect uniformly sampled waveforms, which enabled certain optimizations for display and signal processing. If the standard deviation of intervals between samples is less than 1% of the average sample interval, the waveform is assumed to be uniformly sampled and timestamps are rounded to the nearest multiple of the average interval. If the deviation is greater, the waveform is assumed to be sparsely sampled and timestamps are not modified.

- `[file.trc]`

Imports waveform data from a Teledyne LeCroy .trc binary waveform file.

- `[file.vcd]`

Imports digital waveform data from a VCD (value change dump) file, typically created by a logic analyzer or HDL simulator.

- `[file.wav]`

Imports sample data from a WAV file.

- `[file.wfm]`

Imports sample data from a Tektronix .wfm file. This import filter is still experimental and may not support all features of the .wfm file format yet. If you have trouble importing some .wfm files please file a ticket on GitHub.

- `-nodata`

When loading a .scopesession file, load settings only and not saved waveform data.

- `-reconnect`

When loading a .scopesession file, reconnect to the instrument and resume remote control. Current instrument settings are overwritten with the configuration from the saved session.

- `-retrigger`

When loading a .scopesession file, arm the trigger immediately. has no effect unless `-reconnect` is also specified.

3.5.4 Instrument arguments

Example:

```
./glscopeclient --debug \  
    mylecroy:lecroy:vicp:myscope.example.com:1234 \  
    myrigol:rigol:lan:rigol.example.com
```

- [connection string]
Connects to the specified instrument. By default, all channels are enabled and displayed.

Each instrument is described by a “connection string” containing four colon-separated fields.

- Nickname. This can be any text string not containing spaces or colons. If you have only one instrument it’s largely ignored, but when multiple instruments are present channel names in the UI are prefixed with the nickname to avoid ambiguity.
- Driver name. This is a string identifying the command protocol the scope uses. Note that not all scopes from the same vendor will use the same command set or driver!
- Transport. This is a string describing how the driver connects to the scope (e.g. RS232 or Ethernet)
- Arguments for the driver identifying the device to connect to, separated by colons. This varies by driver but is typically a hostname:port combination, TTY device path, or similar.

3.6 Design Philosophy

glscopeclient’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, then hidden as soon as they are not needed.

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).

If you have a multi-button gaming mouse, button 8 stops the trigger and button 9 starts. These bindings are not currently configurable.

Most text fields allow SI prefixes for scaling values (mV, us, GHz, etc).

3.7 Troubleshooting

3.7.1 Corrupted or no graphical output in waveform areas

Some users have reported problems with libglvnd under Linux. Changing the CMP0072 declaration in glscopeclient/CMakeLists.txt from NEW to OLD may fix it.

Chapter 4

Transports

4.1 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
```

4.2 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
```

4.3 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.

Example:

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

4.4 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. Since the argument is ignored, any non-empty string may be used.

Example:

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

4.5 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: host-name/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.

4.6 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
```

4.7 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
```

4.8 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 5

Oscilloscope Drivers

5.1 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	

5.1.1 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

5.2 Antikernel Labs

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

5.2.1 akila

This driver uses a raw binary protocol, not SCPI.

Under-development internal logic 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.

5.2.2 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.

5.3 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 glscopeclient 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	

5.4 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.

5.4.1 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

5.5 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 U3Pro16 has 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

5.5.1 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

5.6 Enjoy Digital

TODO ([scopehal:79](#))

5.7 Hantek

TODO ([scopehal:26](#))

5.8 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:

5.8.1 agilent

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

5.9 Keysight DCA

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

Device Family	Driver	Notes
86100A	keysightdca	

5.10 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 (azonenberg/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	

5.10.1 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

5.11 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	

5.11.1 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

5.12 Rohde & Schwarz

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

TODO (scopehal:59)

5.13 Saleae

TODO (scopehal:16)

5.14 Siglent

A driver for SDS2000X+ is available in the codebase which has been developed according to Siglent official 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)

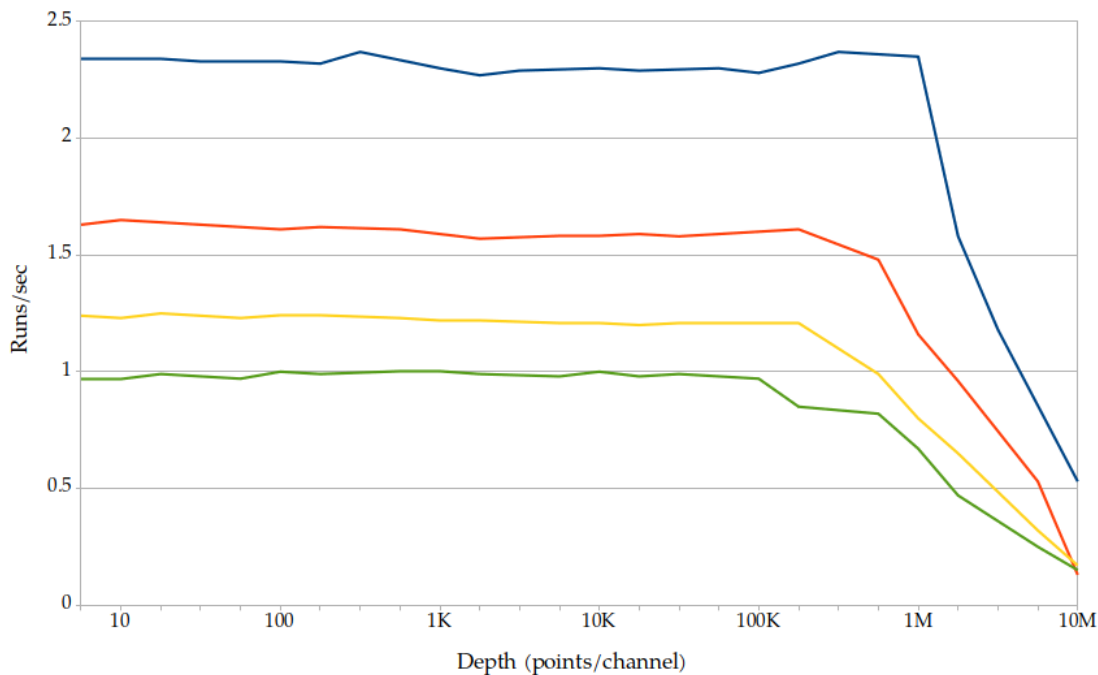


Figure 5.1: Siglent sample speed for various combinations of depth and channels

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

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.

5.15 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 series
T3DSO	???	???	Untested
WaveAce	???	???	Untested
WaveJet	???	???	Untested
WaveMaster	lecroy	vicp	Same hardware as SDA/DDA
WaveRunner	lecroy	vicp	Tested on WaveRunner Xi and 8000 series
WaveSurfer	lecroy	vicp	Tested on WaveSurfer 3000 series

5.15.1 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

5.15.2 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:

- Maximum 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.

5.16 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	

5.16.1 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

5.17 Xilinx

TODO (scopehal:40)

Chapter 6

Power Supply Drivers

NOTE: Power supplies are only supported by ngscopeclient as of this writing (2022-09-24)

6.1 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.

6.1.1 gwinstek_gpdx303s

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

6.2 Rohde & Schwarz

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

6.2.1 rs_hmc804x

Chapter 7

Main Window

The main window of glscopeclient consists of the menu bar and tool bar at top and a status bar at the bottom. All remaining space is occupied by one or more waveform groups.

7.1 Menu

7.1.1 File

This menu contains commands for saving and loading waveform and session files, and managing instrument connections.

- **Connect**
Displays the “connect to instrument” dialog.
- **Recent Instruments**
Contains a list of up to ten recently used instruments.
- **Open Online...**
Loads a .scopesession 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 .scopesession file in offline mode, allowing you to work with saved waveform data without connecting to the instrument(s) the data was captured from.
- **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.
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.

- **Export**

Saves some or all of the data in the active session to a file in another format.

For more detail see the [Export Formats](#) chapter.

- **Close**

Close the current session without exiting glscopeclient.

- **Quit**

Exits the application

7.1.2 Setup

- **Instrument Sync**

Synchronizes two or more instruments under a single glscopeclient instance. TODO: more complete documentation

- **Trigger**

Configures trigger settings

- **Halt Conditions**








Makes glscopeclient pause when a waveform meeting certain conditions is acquired

- **Preferences**

Opens the preferences dialog

7.1.3 View

This menu allows display settings to be configured. As of now, the only option is selection of the color palette for eye patterns.

Name	Colors	Notes
CRT		Similar to a major vendor's color scheme.
Grayscale		Common monochrome palette.
Ironbow		Common "hot metal" palette.
KRain		Similar to a major vendor's color scheme.
Rainbow		Common HSV rainbow palette.
Reverse Rainbow		Common HSV rainbow palette.
Viridis		Perceptually uniform palette from matplotlib.

7.1.4 Add

This menu allows new waveforms to be added to the display

- **Channels**

Lists all channels on the currently connected instrument(s)

- **Import**
Allows waveforms to be loaded from external data files in various interchange formats
- **Generate**
Allows synthetic waveforms to be generated for testing, simulation, and channel design applications

7.1.5 Window

This menu provides access to various utility windows.

- **Filter Graph**
Opens the filter graph editor (see Chapter 14)
- **Analyzer**
Opens protocol analyzer views which have been closed
- **Multimeter**
Displays measurements from digital multimeters attached to the session

7.1.6 Help

About: Displays program version and copyright information

7.2 Toolbar

The toolbar contains buttons and controls for the most frequently used actions.



Figure 7.1: glscopeclient toolbar

7.2.1 Capture buttons

The capture button group (Fig. 7.2) contains three buttons. From left to right these are “arm normal trigger”, “arm one-shot trigger” and “stop trigger”.

Note that the “normal” trigger mode still uses one-shot capture internally so that all waveform data can be downloaded before the next trigger event.

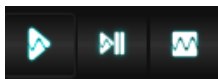


Figure 7.2: Capture control buttons

7.2.2 History

The history button (Fig. 7.3) toggles display of the [waveform history view](#).



Figure 7.3: History button

7.2.3 Refresh Settings

In order to improve performance, glscopeclient caches many instrument settings locally rather than constantly querying the instrument for the current timebase, trigger configuration, etc. If settings are changed via the instrument front panel while glscopeclient is running, glscopeclient may not be aware of these changes.

The Refresh Settings button (Fig. 7.4) clears all cached instrument configuration and updates glscopeclient with the current instrument settings. For most “headless” instruments, such as Pico Technology devices, this button has no effect.



Figure 7.4: Refresh Settings button

7.2.4 Clear Sweeps

The Clear Sweeps button (Fig. 7.5) clears all persistence waveforms, accumulated eye pattern / waterfall data, and statistics. Waveforms saved in history are not deleted.



Figure 7.5: Clear Sweeps button

7.2.5 Fullscreen

The Fullscreen button (Fig. 7.6) switches glscopeclient between normal and full-screen mode.

7.2.6 Opacity slider

The opacity slider (Fig. 7.7) controls the alpha/opacity used to display intensity-graded waveforms. Higher opacity values lead to better display of sparse waveforms (compare the crisp lines of Fig. 7.8 to the barely visible trace in Fig. 7.9) but can lead to a washed-out appearance if too many sample points are shoved into a small area.



Figure 7.7: Trace opacity slider



Figure 7.6: Fullscreen button

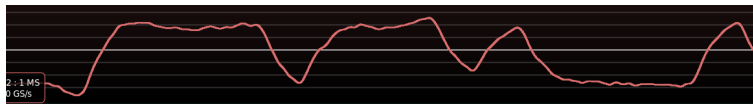


Figure 7.8: Sparse waveform at a high zoom level

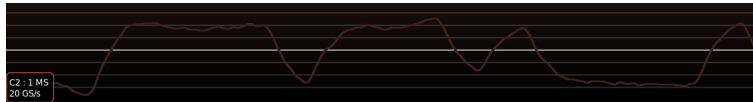


Figure 7.9: Dim waveform showing difficulty of seeing waveform at low opacity

For example, the DVI waveform in Fig. 7.10 looks like a solid white blob with a vaguely visible outline. No fine detail can be observed other than the increased over/undershoot and random-looking edges on the scanlines, compared to the flat appearance of the blanking period between scanlines and at the end of the frame.

When the opacity is reduced in this example, many more nuances of the signal become apparent. The high/low voltage levels of the signal compared to the transitions between them are obvious, and the H/V sync pulses within the blanking period show up as a slightly darker region.

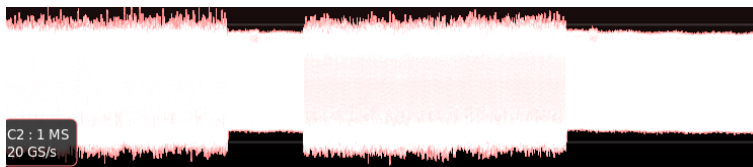


Figure 7.10: Intensity-graded waveform showing washed-out appearance at high opacity

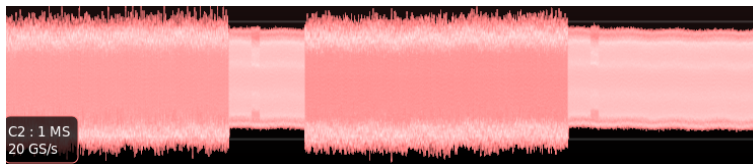


Figure 7.11: Intensity-graded waveform at lower opacity level

As of this writing, the opacity setting is global for the entire application. Should this be changed to per waveform group? If so, how should the group be selected and should there still be an option to make changes globally?

Chapter 8

Waveform Groups

A waveform group is a collection of one or more waveforms stacked vertically under a common timeline. All waveforms within a group share the same timeline and vertical cursor(s).

When glscopeclient starts up, by default all channels on the attached instrument(s) are displayed in a single waveform group (Figure 8.1).

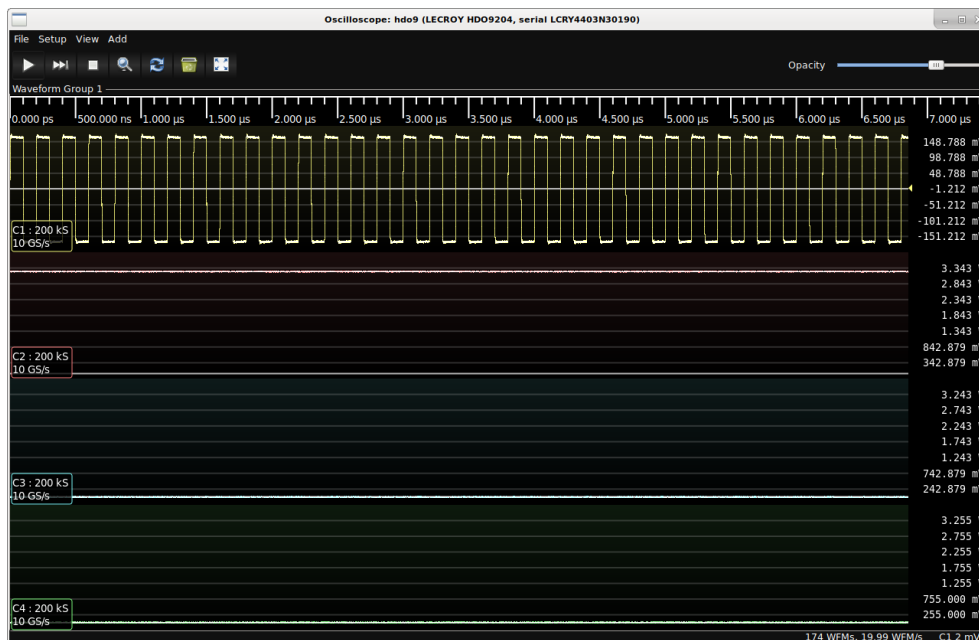


Figure 8.1: Top level glscopeclient window with a single waveform group

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.

8.1 Managing Groups

Additional groups may be created by right clicking a waveform and selecting [Move|Copy] waveform to / Insert new group at [right|bottom] from the context menu. This will split the current group's area in half horizontally or vertically, with the selected waveform moved or copied to the newly added group and all other waveforms in the original group.

Waveforms may also be moved within, or between, groups by clicking the channel information box and dragging it. A yellow insertion bar will appear when dragging, showing the location the waveform will be inserted in.

If a waveform is dragged to the very bottom or right side of a waveform group, the destination group will be split vertically or horizontally and the new waveform will be inserted below or to the right of the destination group. The insertion bar turns orange when dragging near the edge of a group, to indicate that a split will take place.

Dividers between waveform groups may be dragged with the left mouse button. Any group may be subdivided again, to create arbitrarily complex tiles of waveforms. Figure 8.2 shows a two-level hierarchy created by moving channel 2 to a new group at right, then moving channel 4 to a new group below that one.

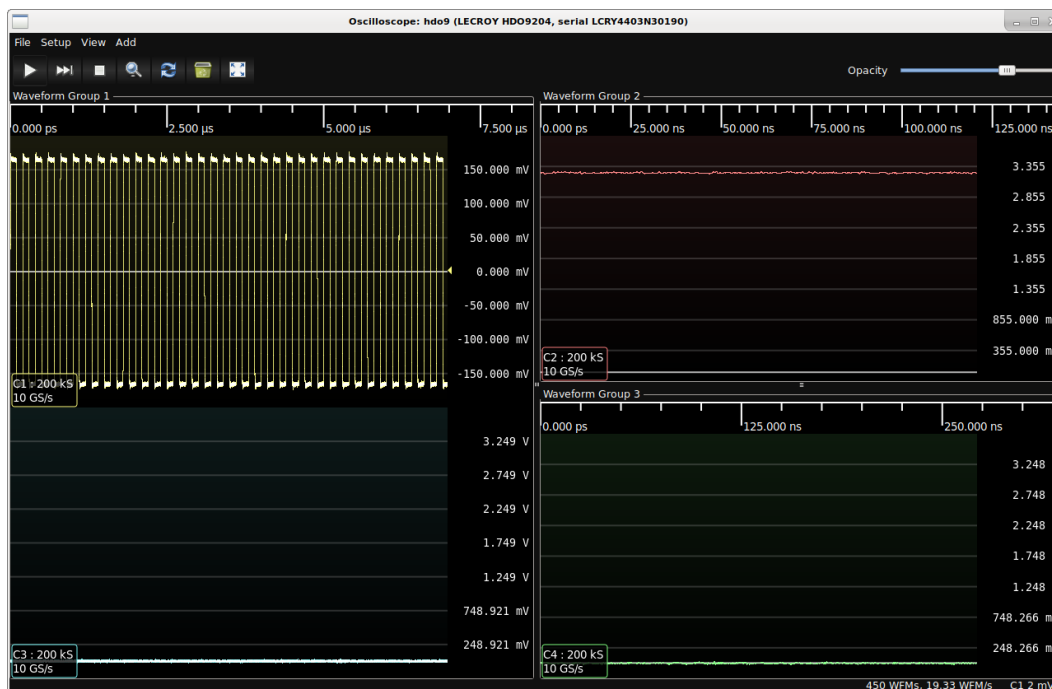


Figure 8.2: Top level glscopeclient window with several waveform groups separated by splitters

Protocol decode overlays may be reordered by dragging the channel information box with the mouse, however they cannot currently be moved to another waveform or group.

New waveform groups are given an automatically generated name when created, for example "Waveform Group 2". This name may be changed by double clicking the group name and entering a new name in the dialog.

Chapter 9

Timeline

The timeline is displayed at the top of each waveform group and shows the X axis scale for the group. The timeline (and all accompanying waveform views in the group) may be zoomed by scrolling with the mouse wheel, or panned by dragging with the left mouse button.

Unlike classical oscilloscope user interfaces, there is *no relationship* between the timeline scale or position and the duration of the acquisition. It is possible to zoom or scroll beyond the end of the acquisition (displaying empty background with no signal) or have a deep capture in which nearly all acquired data is offscreen.

Note that the timeline may occasionally show units other than time. For example, an “eye width” measurement has X axis units of voltage and Y axis units of time, and a spectrum analyzer channel has X axis units of frequency.



Figure 9.1: The timeline

The position of the trigger event is marked by a downward-pointing arrow on the timeline, color coded to match the channel selected as the primary trigger source. The trigger arrow cannot be interacted with currently, but in the future (scopehal-apps:173) it will be draggable to adjust the trigger position.

Double-clicking on the timeline brings up the timebase properties dialog (Fig. 9.2), which allows the sample rate and memory depth to be configured. If multiple instruments are connected, a separate tab appears in the dialog for each instrument.

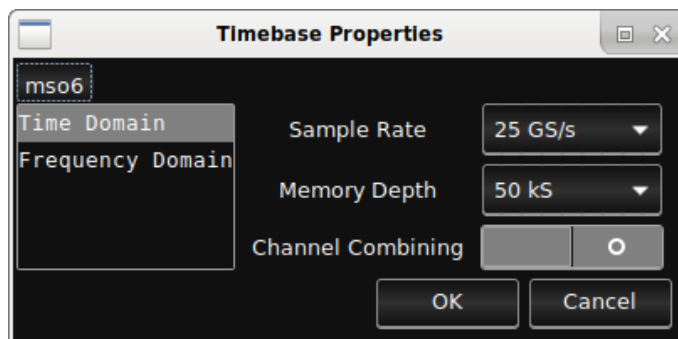


Figure 9.2: Timebase properties (time domain)

If the instrument is a spectrum analyzer, or has frequency-domain analysis capability (such as the Tektronix MSO5 and MSO6 oscilloscopes), the timebase properties dialog will have a second

page (Fig. 9.3) for setting the span and resolution bandwidth for frequency-domain channels. Center frequency is often a per-channel adjustment on multichannel instruments, so it is configured from the channel properties dialog rather than timebase properties.

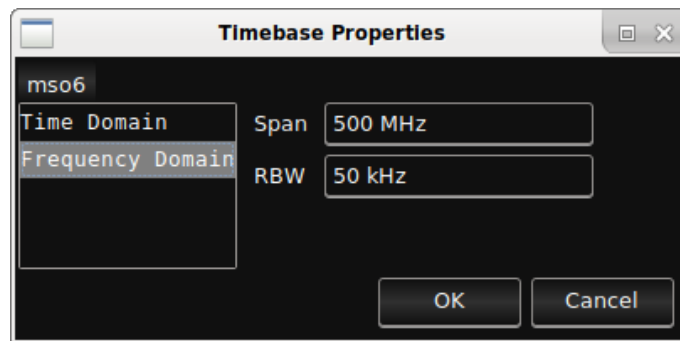


Figure 9.3: Timebase properties (frequency domain)

Chapter 10

Triggers

10.1 Trigger Properties

The Setup / Trigger menu opens the trigger properties dialog (Fig. 10.1).

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

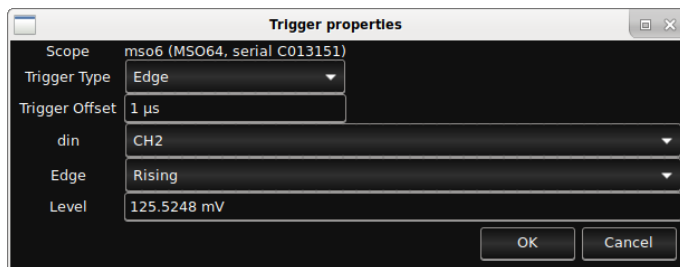


Figure 10.1: Trigger properties dialog

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

10.2 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”.

¹This 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.

10.3 Dropout

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

10.3.1 Inputs

Signal name	Type	Description
din	Analog or digital	Input signal

10.3.2 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

10.4 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.

10.4.1 Inputs

Signal name	Type	Description
din	Analog or digital	Input signal

10.4.2 Parameters

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

10.5 Glitch

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

10.6 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

10.6.1 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

10.7 Runt

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

Signal name	Type	Description
din	Analog	Input signal

10.7.1 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

10.8 Slew Rate

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

Signal name	Type	Description
din	Analog	Input signal

10.8.1 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

10.9 UART

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

10.9.1 Inputs

Signal name	Type	Description
din	Analog or digital	Input signal

10.9.2 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

10.10 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

10.10.1 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 11

Waveform Views

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

11.1 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.

Pressing SHIFT while scrolling moves the view left and right without adjusting zoom. If your mouse has a horizontal scroll feature, this may also be used to pan without zooming.

Pressing the middle mouse button auto-scales the active waveform group so that the entire waveform is visible.

11.2 Plot Area

The plot area shows the waveform being displayed. The background has a subtle gradient from light at top to dark at bottom, in order to visually separate adjacent waveform view within the same group.

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.

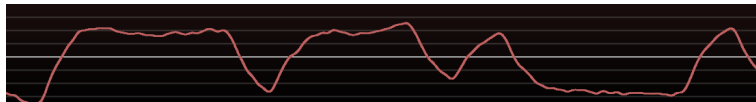


Figure 11.1: Waveform plot area

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”.

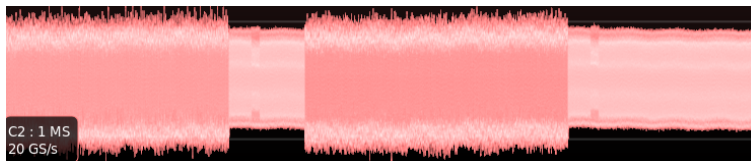


Figure 11.2: Intensity-graded waveform

11.3 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 with the left mouse button. If the view is displaying the output of a filter block, you may use the middle mouse button to auto-scale the vertical axis to the range of the current waveform. The auto-scale feature is not available for physical instrument inputs.

If a left-pointing arrow (as seen in Fig. 11.3) is visible, the current channel 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.

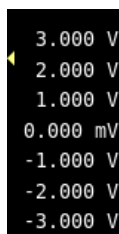


Figure 11.3: Y axis of a waveform view showing trigger arrow

11.4 Channel Information Box

The channel information box is displayed in the lower left corner of each waveform view. It contains summary information about the channel. Currently this is the display name of the channel, the sample rate, and the record length of the acquisition. Other information, such as probe coupling, may be displayed there in the future.

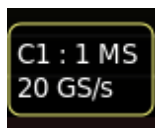


Figure 11.4: Channel information box

The information box may be dragged with the left mouse button to move the entire waveform view to a new location.

Double-clicking the information box opens the channel properties dialog (Fig. 11.5). This dialog allows changing of the channel's nickname or color. The "hardware name" of the channel is also displayed, so that a renamed channel can be easily traced back to a physical instrument input.

Depending on the particular instrument in question, other settings may be displayed here, such as fine deskew, attenuation, polarity inversion, and bandwidth limiting. An input mux selector is

shown if the scope has more than one input per channel (such as the Teledyne LeCroy SDA and WaveMaster series).

The channel properties dialog also allows setting threshold and hysteresis for digital channels, and ADC configuration for scopes that have variable analog resolution (such as the Teledyne LeCroy HDO9000 or Pico Technology 6000E series). These settings are listed under channel properties since some instruments support per-channel adjustment of them, however it is important to note that these settings apply to multiple channels (a bank or even the entire instrument) in some cases. A list box under these settings shows the list of other channels, if any, that share the same configuration.

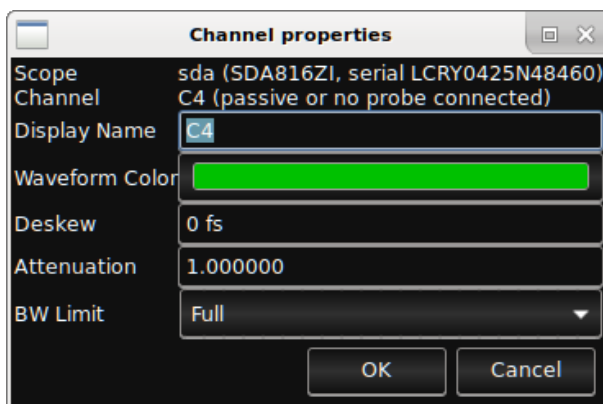


Figure 11.5: Channel properties dialog

11.5 Cursors

11.5.1 Vertical Cursors

To add a vertical cursor (Fig. 11.6), right click on the waveform and select **Cursor | Vertical (single)** or **Cursor | Vertical (dual)** as appropriate.

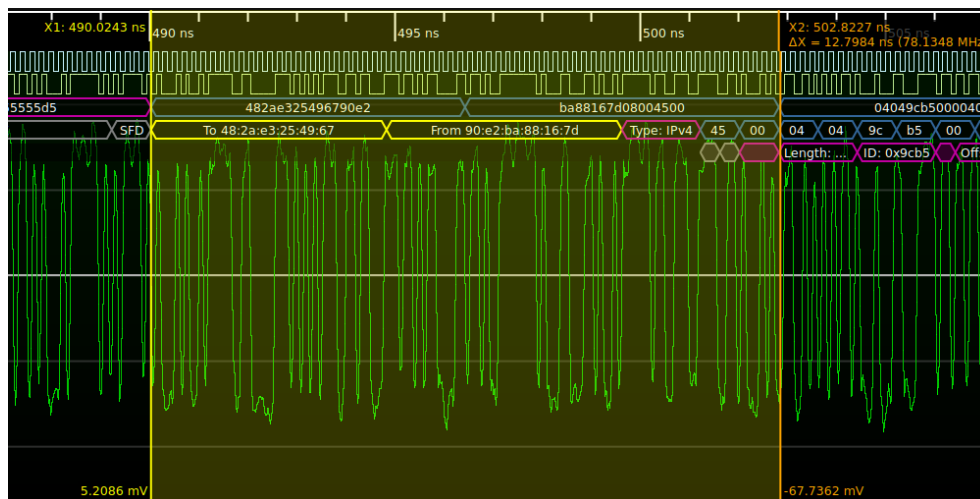


Figure 11.6: Vertical cursor

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.

Cursors will snap to transitions in digital signals or protocol decode overlays if the mouse is within a few pixels of the location. No snapping is applied when the mouse is over an analog waveform.

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.

At the bottom of each waveform area, the Y-axis value of the signal where it crosses the cursor is shown. In FFT / spectrum analyzer plots, the integrated in-band power between both cursors is also shown.

If a protocol analyzer view (Chap. 13) is active, moving a single cursor over a packet will scroll to and highlight that packet.

11.5.2 Horizontal Cursors

To add a horizontal cursor (Fig. 11.7), right click on the waveform and select **Cursor | Horizontal (single)** or **Cursor | Horizontal (dual)** as appropriate.

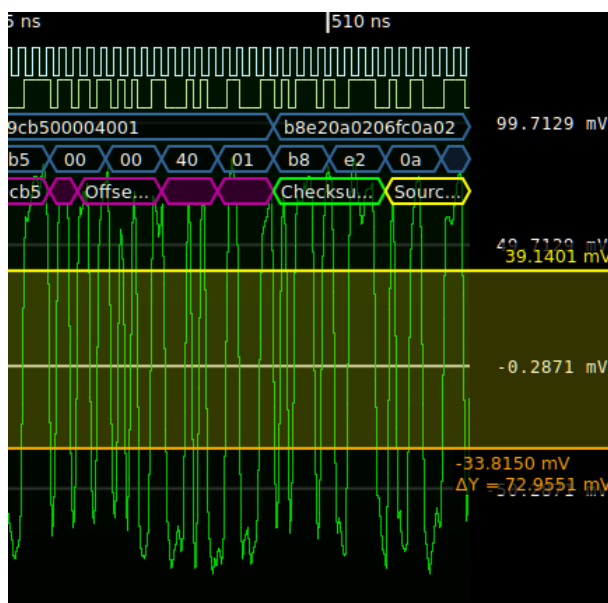


Figure 11.7: Horizontal cursor

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.

At the right side of the plot, each cursor will display its Y-axis location. If both cursors are active, the delta between them is also shown.

All waveform areas in a group share the same Y axis cursor positions.

11.5.3 Markers

11.6 Overlays

Waveforms may have additional information overlaid on top of them, such as protocol decodes. Each overlay has its own information box, which may be double-clicked to open the properties dialog and configure it just like any other channel.

Fig. 11.8 shows an example of an analog waveform with five overlays: a CDR PLL, thresholding, and decodes of the 64/66b line code, the 10Gbase-R Ethernet framing, and IPv4 packet headers.

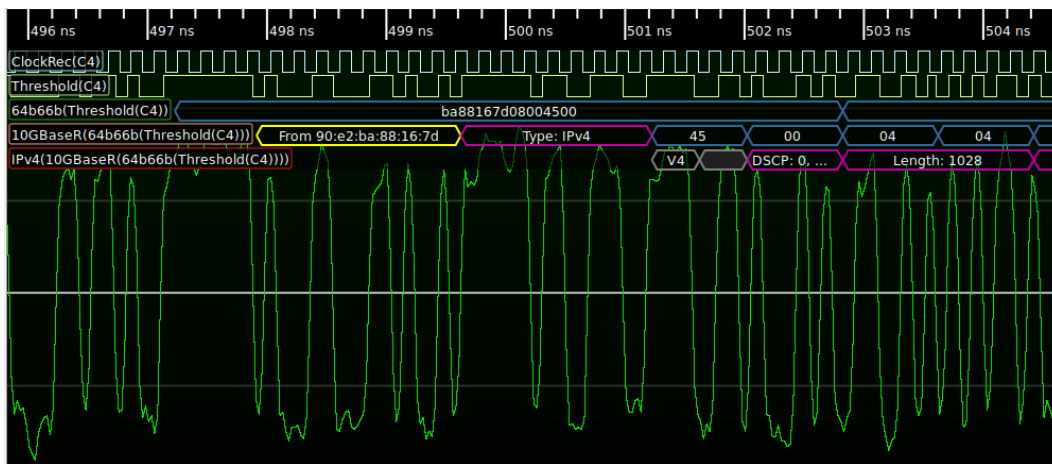


Figure 11.8: Waveform showing two digital overlays and a three decode overlays

Overlays can be deleted by means of the right-click context menu. Dragging the information box with the left mouse button allows overlays to be reordered, however they cannot currently be moved to another waveform view.

11.7 Statistics

Statistics (Fig. 11.9) may be shown for any waveform by checking the “statistics” box in the context menu. The default statistics are minimum, average, and maximum although more may be added in the future.

C4	
Maximum	94.8750 mV
Average	-0.9472 mV
Minimum	-100.0312 mV

Figure 11.9: Statistics for an analog waveform

Chapter 12

History View

glscopeclient has the ability to save every waveform during a session 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 pauses acquisition and loads the historical waveform data for analysis.

By default, the history view (Fig. 12.1) is not displayed and history is limited to ten waveforms. If the history view is closed, history continues to be captured up to the configured maximum history depth.

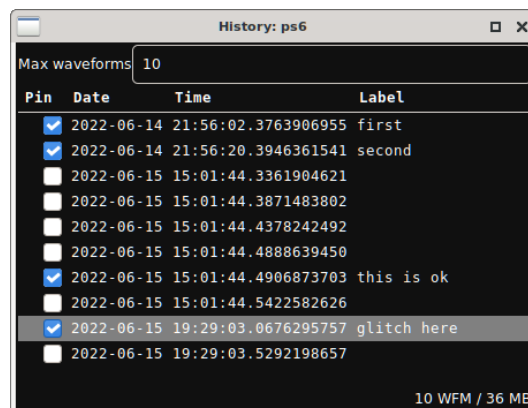


Figure 12.1: Waveform history view

The “max waveforms” box allows the depth of the history to be configured. It defaults to 10 but can be set to any positive integer value. Older waveforms beyond the history limit are deleted as new waveforms are acquired.

The status bar at the bottom of the history view displays the total number of waveforms in the history, as well as an estimate of the amount of RAM used by the history.

12.1 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.

12.2 Labeling

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

12.3 Estimating Waveform Memory Usage

When selecting a maximum depth for the history, it is important to pick a reasonable limit to avoid running out of RAM! `glscopeclient` will happily fill tens or hundreds of gigabytes of memory with deep waveforms if given a chance. Memory usage of waveform data can be roughly estimated as $16 + \text{sizeof}(\text{sample type})$ bytes per point, since each sample contains a 64-bit timestamp and duration plus the sample data.

For example, an analog sample takes 20 bytes of RAM (16 of time plus a 32-bit floating point voltage measurement) per sample. Thus, a 1M point analog waveform takes approximately 20 MB of RAM per channel, or 80 MB per capture on a four-channel oscilloscope with all channels enabled.

On the larger side, a 10M point four channel capture would use 800 MB and a 64M point deep-memory capture would use 5 GB. A deep history setting, such as 100 waveforms, is thus wildly inappropriate for such deep captures! A future software release may support spilling waveform data to a temporary directory on disk, permitting effectively unlimited history depth given sufficient disk space.

Digital waveforms use one byte per sample for the actual measurement, so 17 MB per channel for a 1M point waveform. Most logic analyzer or MSO drivers for `libscopehal` will perform automatic de-duplication when a waveform goes several clock cycles with no toggles, so the actual memory usage is likely to be significantly less than this.

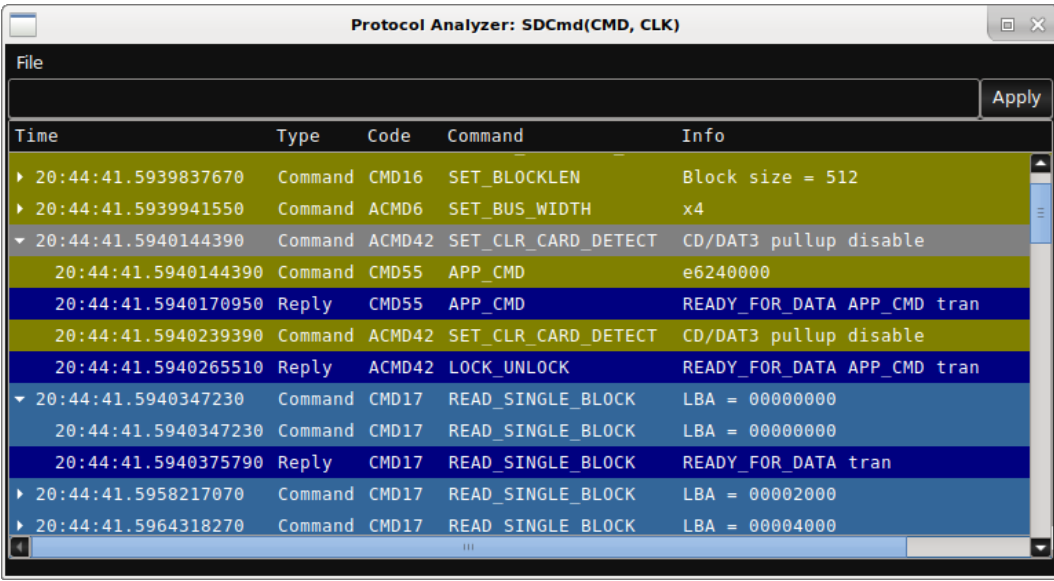
Filter memory usage varies depending on the specific filter in question, however it is typically not a large contributor to the overall `glscopeclient` RAM footprint when using history mode because filters are evaluated dynamically each time a waveform is pulled from history rather than having output cached for every historical waveform. Thus, at most one copy of each filter’s output is present in memory regardless of history depth.

Chapter 13

Protocol Analyzer View

Some filters for decoding packet-oriented data provide an alternate means of visualizing the decoded traffic.

The protocol analyzer view (Fig. 13.1) displays each packet in the history as a row in a list view. The first column is always the timestamp of the packet; remaining columns vary depending on the particular filter in question.



Time	Type	Code	Command	Info
20:44:41.5939837670	Command	CMD16	SET_BLOCKLEN	Block size = 512
20:44:41.5939941550	Command	ACMD6	SET_BUS_WIDTH	x4
20:44:41.5940144390	Command	ACMD42	SET_CLR_CARD_DETECT	CD/DAT3 pullup disable
20:44:41.5940144390	Command	CMD55	APP_CMD	e6240000
20:44:41.5940170950	Reply	CMD55	APP_CMD	READY_FOR_DATA APP_CMD tran
20:44:41.5940239390	Command	ACMD42	SET_CLR_CARD_DETECT	CD/DAT3 pullup disable
20:44:41.5940265510	Reply	ACMD42	LOCK_UNLOCK	READY_FOR_DATA APP_CMD tran
20:44:41.5940347230	Command	CMD17	READ_SINGLE_BLOCK	LBA = 00000000
20:44:41.5940347230	Command	CMD17	READ_SINGLE_BLOCK	LBA = 00000000
20:44:41.5940375790	Reply	CMD17	READ_SINGLE_BLOCK	READY_FOR_DATA tran
20:44:41.5958217070	Command	CMD17	READ_SINGLE_BLOCK	LBA = 00002000
20:44:41.5964318270	Command	CMD17	READ_SINGLE_BLOCK	LBA = 00004000

Figure 13.1: Protocol analyzer view

If closed, the protocol analyzer view may be reopened by selecting the protocol of interest from the Window / Analyzer menu.

Many filters group related packets (request and reply, escape sequences, polling loops, etc) under a single heading to enable easier navigation of large datasets. The tree expansion button at the left of the timestamp column may be used to expand the event into its constituent packets.

13.1 Cursor Interaction

Clicking on a packet pauses acquisition, loads the relevant waveform from history if the packet is not in the current waveform, and scrolls the waveform view containing the protocol decode to show

the packet. If the packet fits entirely within the view at the current zoom setting it is centered in the view; otherwise the beginning of the packet is placed near the left edge of the viewport and the packet continues off the right edge.

If a vertical cursor (Sec. 11.5) is active in the waveform area displaying the protocol decode, clicking on a packet in the analyzer view moves the cursor to the start of the packet. Placing the cursor on a packet highlights the corresponding row in the protocol analyzer.

13.2 Packet Coloring

Protocol packets are color coded according to the high-level function of the packet. The colors are configurable in preferences; defaults are shown in the table below.

Color name	Use case	Default Color
Command	Executing commands	#600050
Control	Changing configuration	#808000
Data read	Reading data	#336699
Data write	Writing data	#339966
Error	Malformed, bad checksum	#ff0000
Status	Status updates, flow control	#000080

13.3 Filtering

To ease analysis of large packet datasets, filters may be applied to the analyzer view by typing a filter expression into the filter bar at the top of the window (Fig. 13.2), then pressing the “Apply” button. A filter can be removed by deleting the contents of the filter bar and pressing “Apply”.

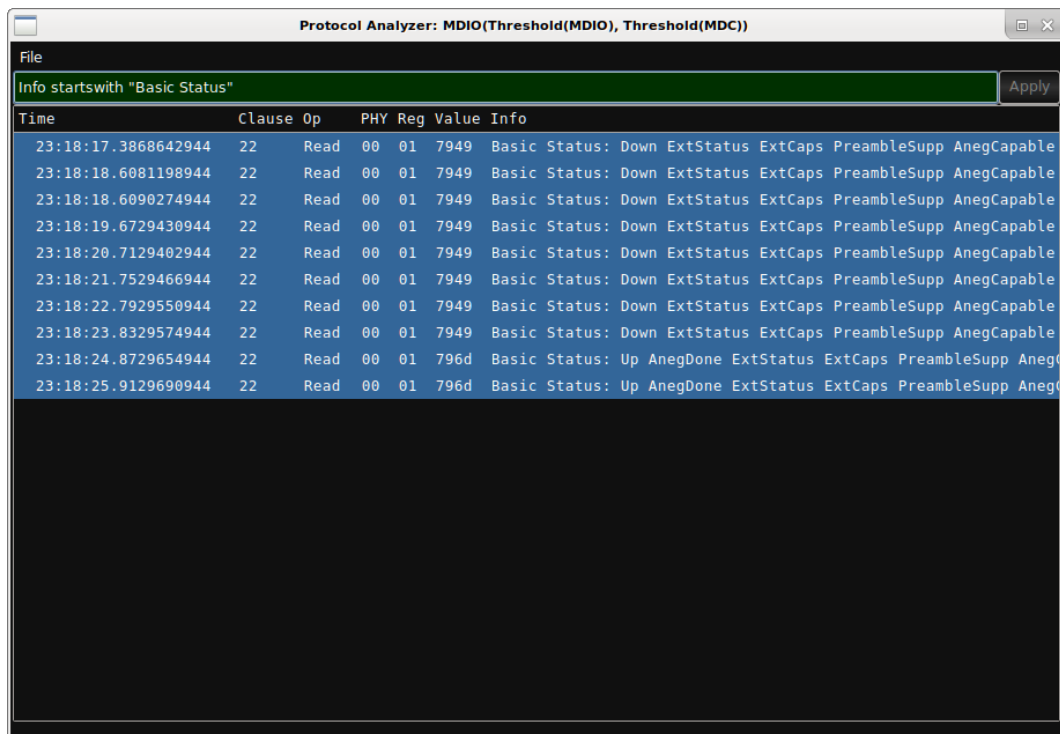


Figure 13.2: Filtering protocol analyzer view

The basic format of a filter is (expression) (operator) (expression).

13.3.1 Expressions

An expression can be:

- A quoted string
- A decimal number
- A field identifier (such as PHY or Reg). Identifiers are case sensitive.
- data[x], where x is an arbitrary numeric expression
- A filter expression in parentheses. This expression must evaluate to boolean true or false. The unary ! operator can be used to negate a parenthetical expression.

13.3.2 Operators

An operator can be:

- ==: returns true if the left and right expression are equal
- !=: returns true if the left and right expression are not equal
- ||: returns true if at least one of the left or right expression is true
- &&: returns true if both the left and right expression is true
- startswith: returns true if the right expression is a string which starts with the left expression
- contains: returns true if the right expression is a string which contains the left expression

13.3.3 Examples of filters

```
1 Op == "Read"
2 Reg == "0f"
3 (Clause == 22) && (Info startswith "Basic Status")
```


Chapter 14

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.

The leftmost column shows all of the input channels which may be used as data sources for the filter graph. Filter nodes are automatically placed in columns such that data flows from left to right, with inputs at the left and outputs at the right of each node.

Nodes may be dragged vertically within their column by using the left mouse button, however the horizontal position of each node is fixed.

To make a connection, click on the source node's output and then on the destination node's input. To cancel an in-progress connection, simply click anywhere outside a node.

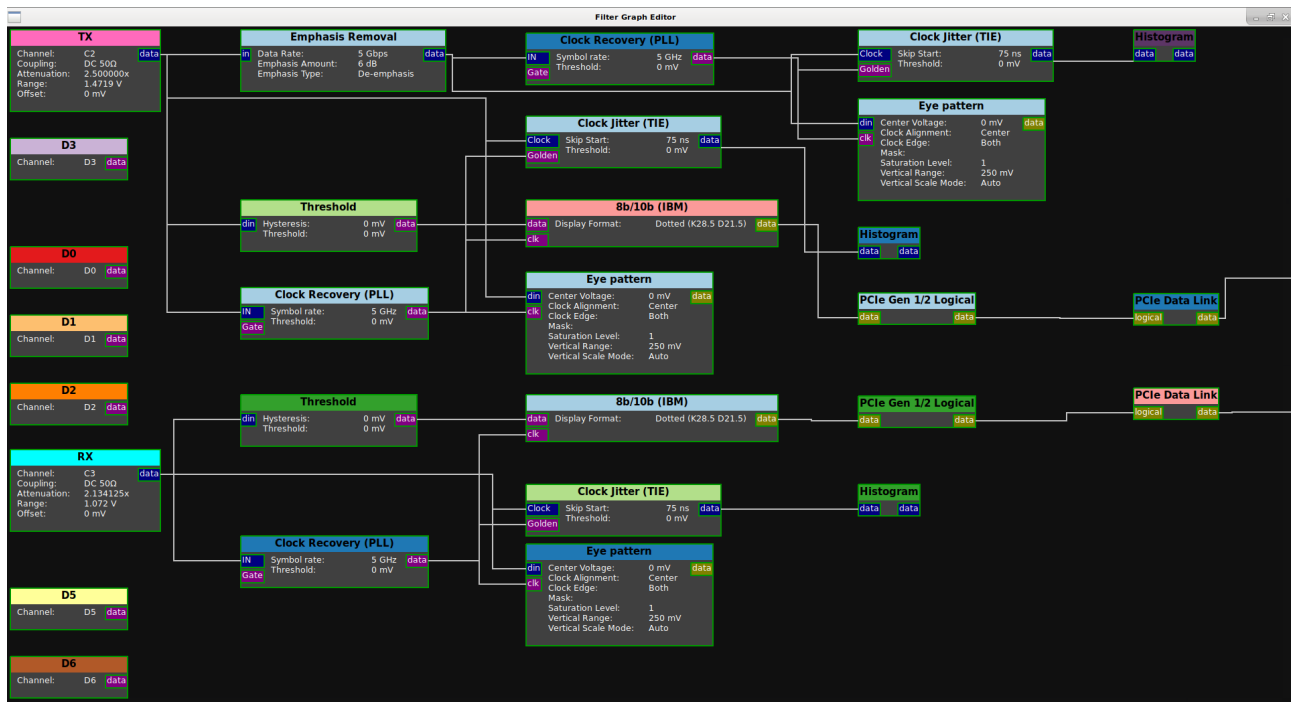


Figure 14.1: Filter graph editor view

The title bar of each node in the graph is color coded to match the channel's color in the

waveform view.

Input and output ports are color coded according to the type of data. The colors are configurable under the **Appearance / Filter Graph** preference category; the default assignment is dark blue for analog, purple for digital, and olive for protocol decodes and other non-primitive types.

Chapter 15

Filters

15.1 Introduction

15.1.1 Key Concepts

glscopeclient 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.

15.1.2 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 may be 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

15.2 128b/130b

Decodes the 128b/130b line code used by PCIe gen 3/4/5. Data fields are descrambled but no further decoding is performed.

15.3 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.

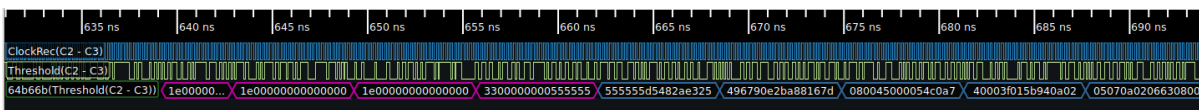


Figure 15.1: Example 64b/66b decode

15.3.1 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.

15.3.2 Parameters

This filter takes no parameters.

15.3.3 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

15.4 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.

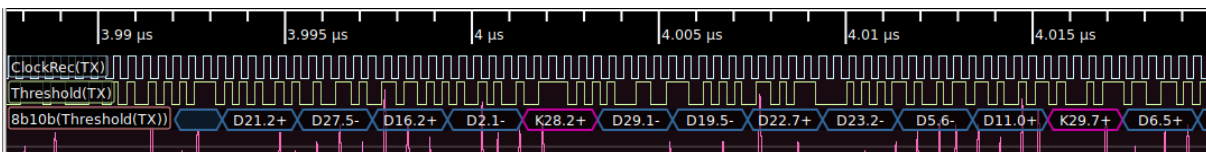


Figure 15.2: Example 8b/10b decode

15.4.1 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.

15.4.2 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.

15.4.3 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+
Data	Upper layer protocol data	Data	D%d.%d+
Error	Malformed data	Error	ERROR

15.5 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.

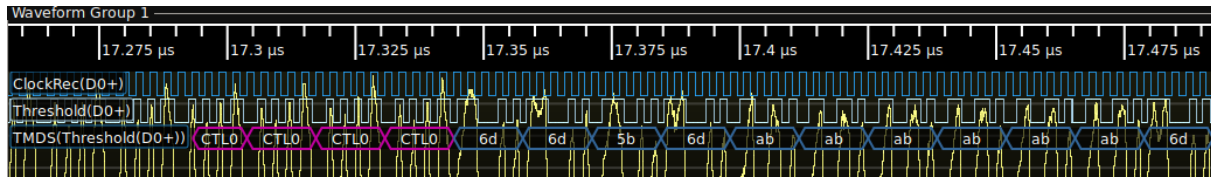


Figure 15.3: Example TMDS decode

15.5.1 Inputs

Signal name	Type	Description
data	1-bit digital	Serial TMDS data line
clk	1-bit digital	DDR <i>bit</i> 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.

15.5.2 Parameters

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

15.5.3 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

15.6 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.

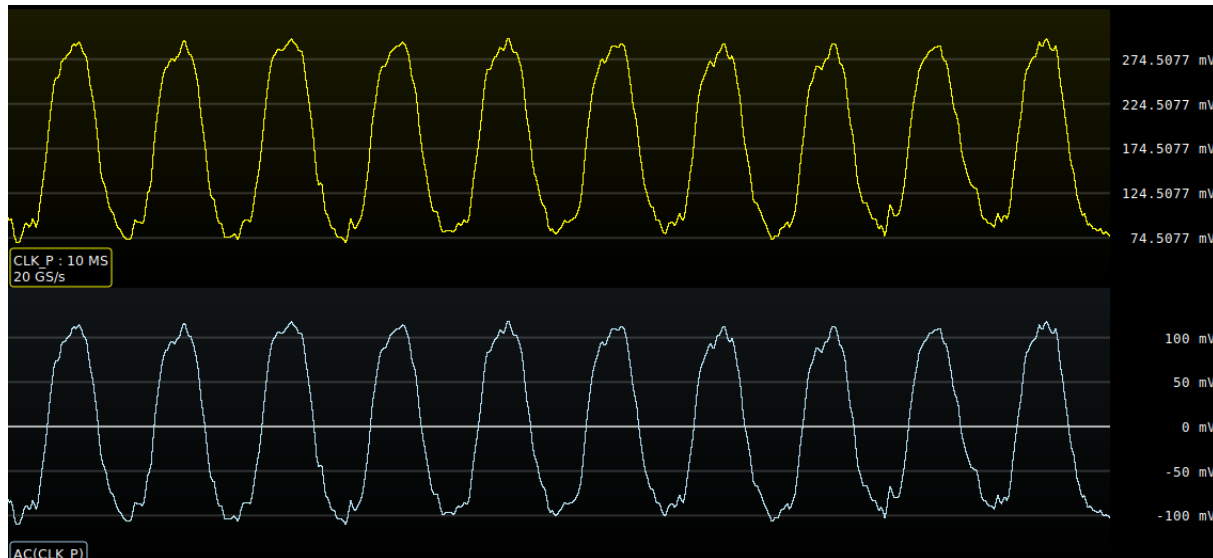


Figure 15.4: Example AC coupling

15.6.1 Inputs

Signal name	Type	Description
din	Analog	Input waveform

15.6.2 Parameters

This filter takes no parameters.

15.6.3 Output Signal

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

15.7 AC RMS

Measures the Root Mean Square value of the waveform after removing any DC offset. This measurement can be made averaged across the entire waveform, or on each cycle in the waveform.

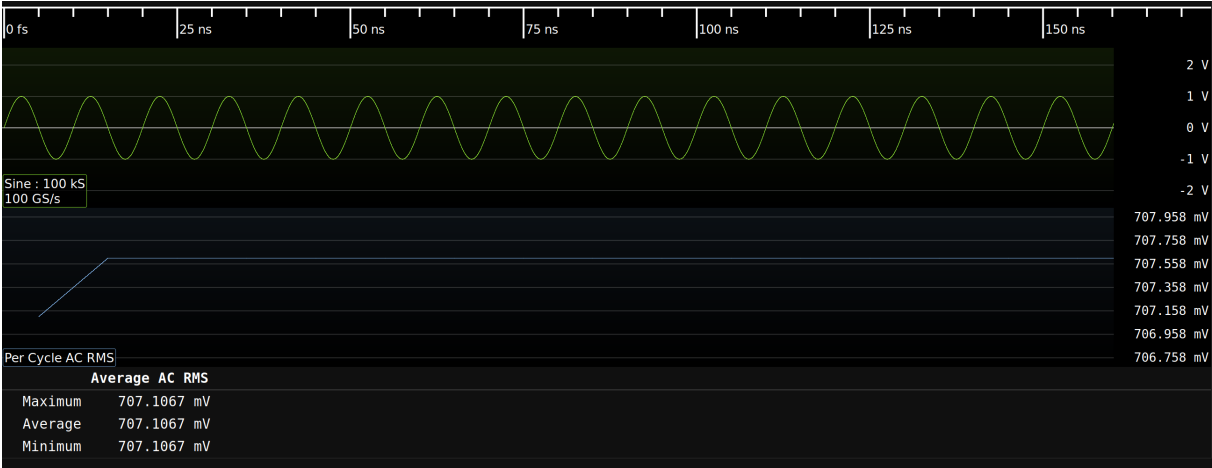


Figure 15.5: Example of an AC RMS Measurement of a Sinewave with 1V peak voltage

15.7.1 Inputs

Signal name	Type	Description
din	Analog	Input waveform

15.7.2 Parameters

Parameter name	Type	Description
Measurement Type	Enum	Average: Measure the average AC RMS value Per Cycle: Measure the per cycle AC RMS value

15.7.3 Output Signal

This filter produces an output waveform only for the per cycle measurement type. This displays the AC RMS value of every cycle.

15.8 Area Under Curve

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.

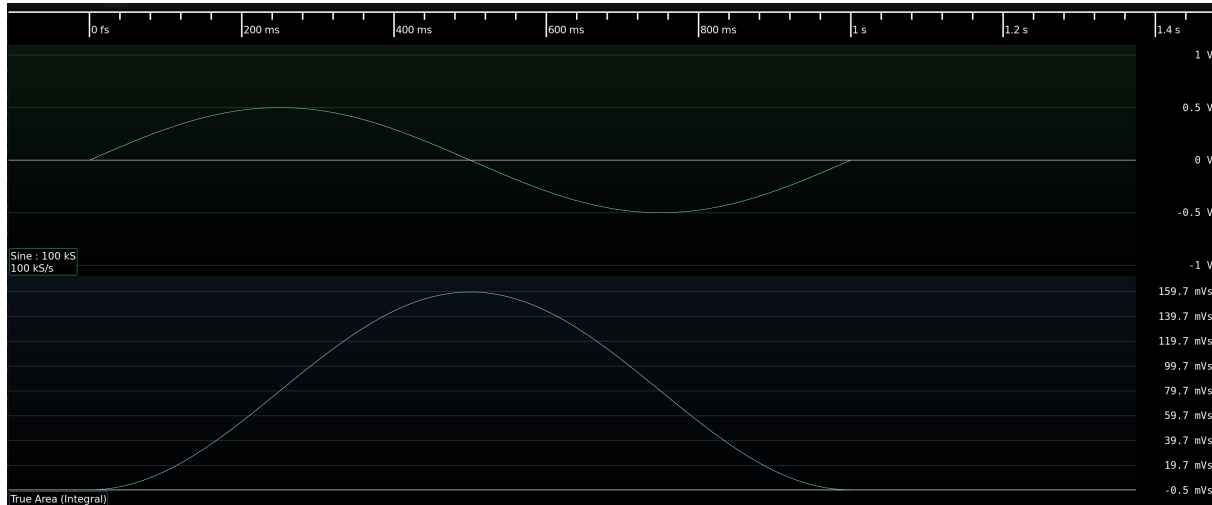


Figure 15.6: Example of true area under the curve measurement (Integral)

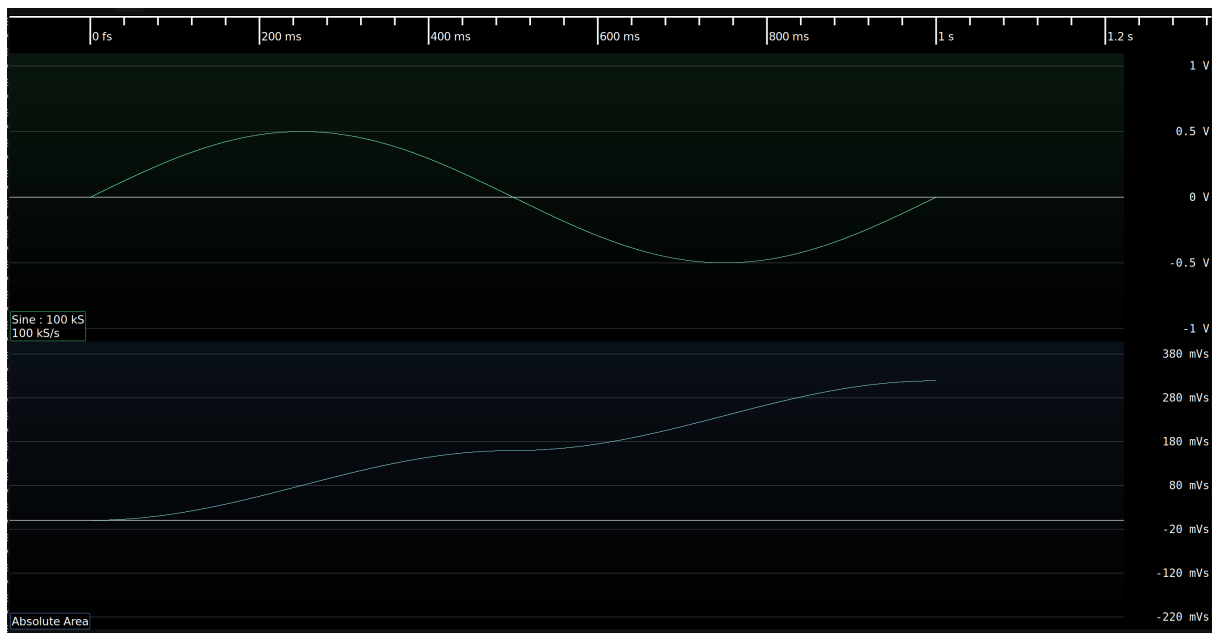


Figure 15.7: Example of absolute area under the curve measurement

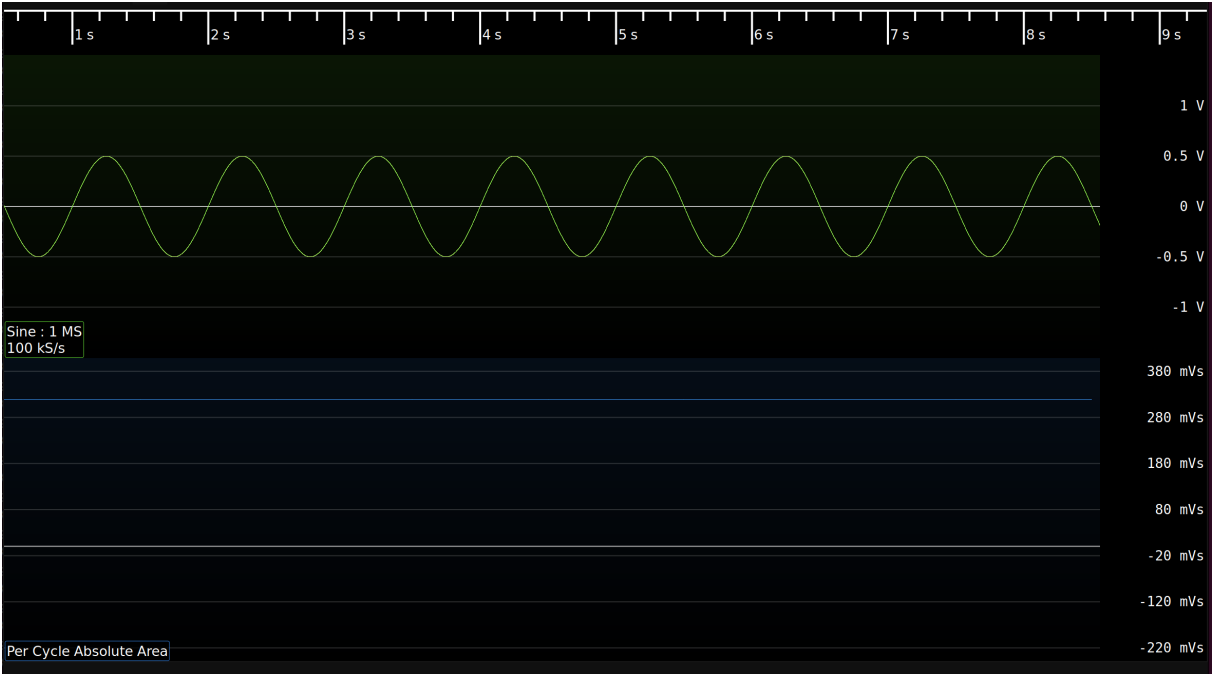


Figure 15.8: Example of per-cycle absolute area under the curve measurement

15.8.1 Inputs

Signal name	Type	Description
din	Analog	Input waveform

15.8.2 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

15.8.3 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.

15.9 ADL5205

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

TODO: Screenshot

15.9.1 Inputs

Signal name	Type	Description
spi	SPI bus	The SPI data bus

15.9.2 Parameters

This filter takes no parameters.

15.9.3 Output Signal

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

15.10 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. 15.9, 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.

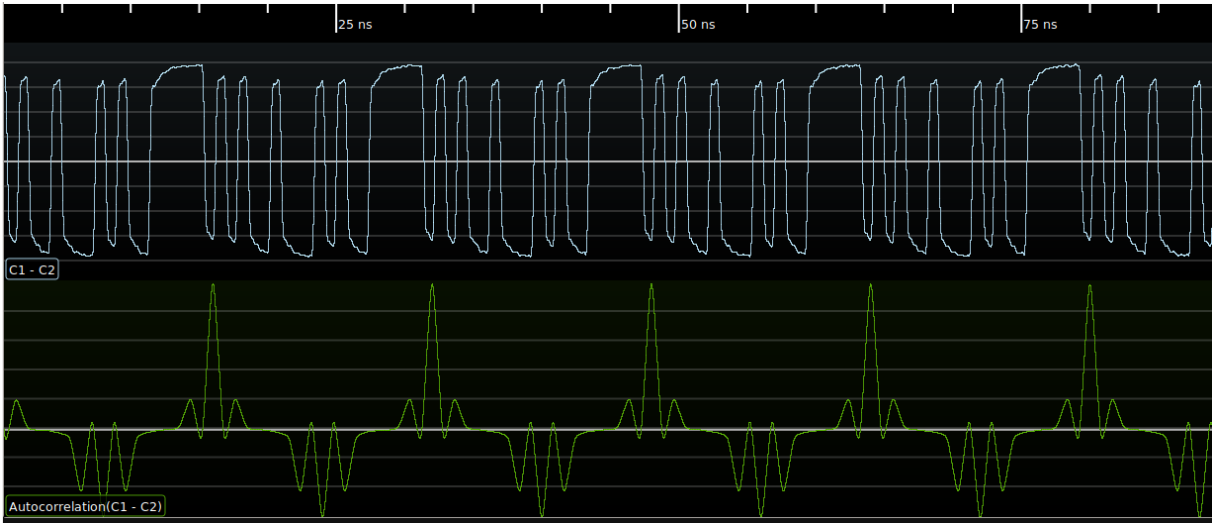


Figure 15.9: Example of autocorrelation on a serial data stream

15.10.1 Inputs

Signal name	Type	Description
din	Analog	Input waveform

15.10.2 Parameters

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

15.10.3 Output Signal

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

15.11 Base

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. 15.10, the vertical eye closure caused by channel ISI is readily apparent.

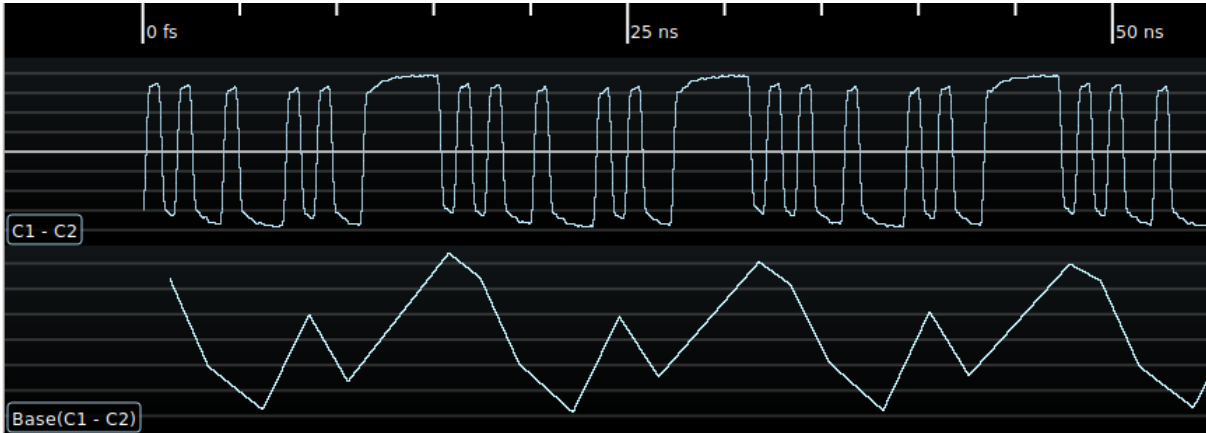


Figure 15.10: Example of base measurement on a serial data stream

15.11.1 Inputs

Signal name	Type	Description
din	Analog	Input waveform

15.11.2 Parameters

This filter takes no parameters.

15.11.3 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.

15.12 BIN Import

Loads an Agilent / Keysight / Rigol binary waveform file.

15.13 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.

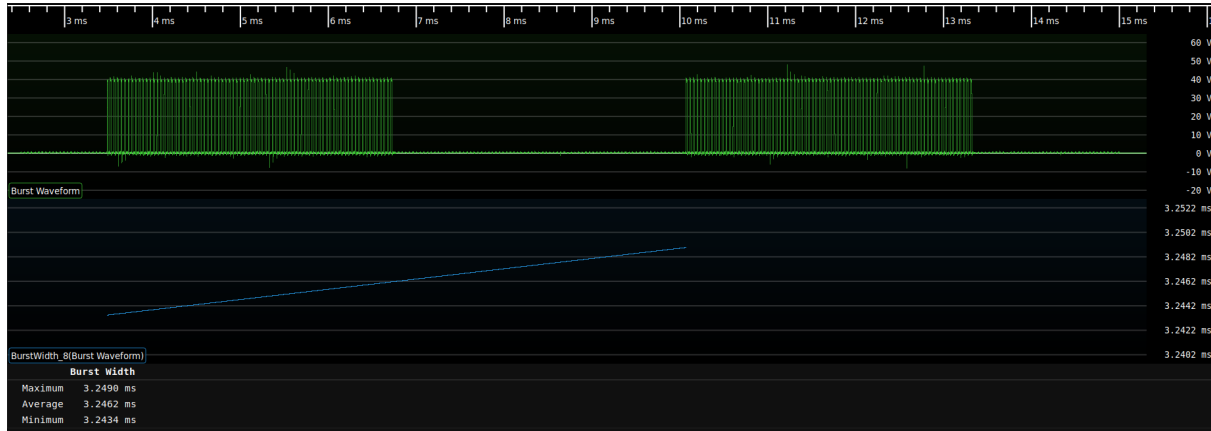


Figure 15.11: Example of burst width measurement

15.13.1 Inputs

Signal name	Type	Description
din	Analog	Input waveform

15.13.2 Parameters

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

15.13.3 Output Signal

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

15.14 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](#)).

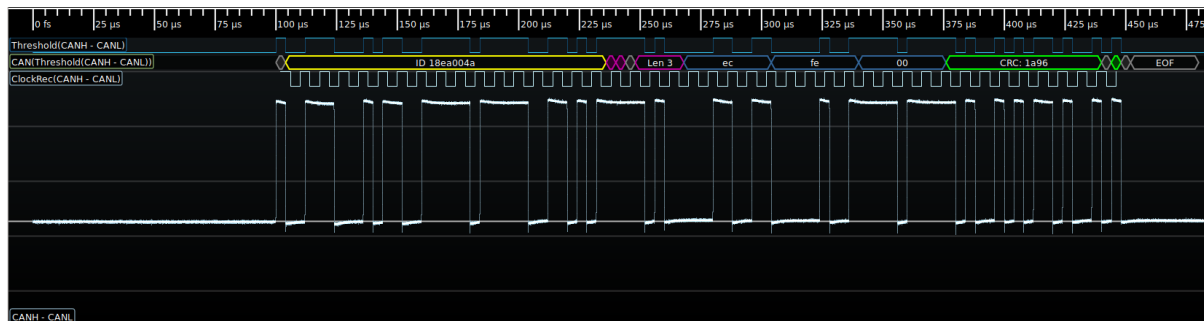


Figure 15.12: Example of CAN bus protocol decode

15.14.1 Inputs

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

15.14.2 Parameters

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

15.14.3 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

15.15 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. 15.13 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.

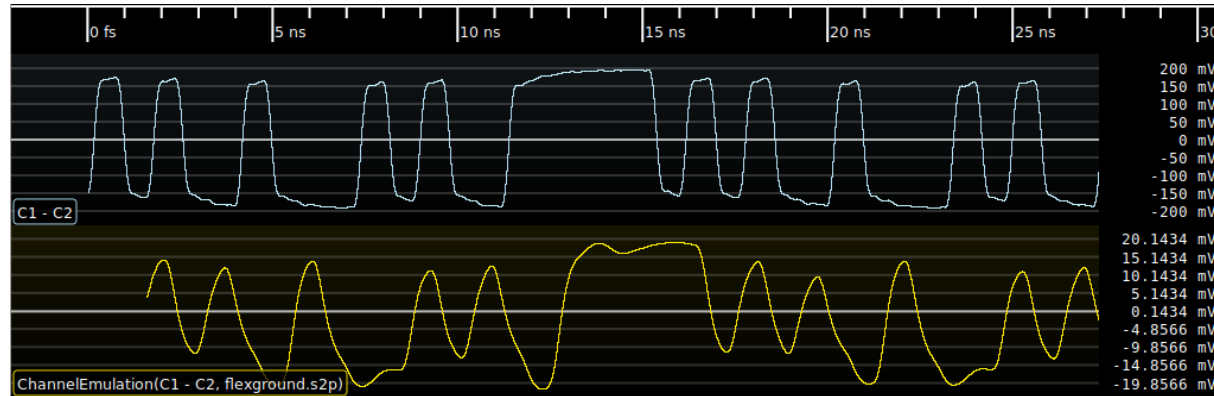


Figure 15.13: Example of channel emulation on a serial data stream

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 to calculate combined S-parameters of the entire circuit and then perform the channel emulation once.

15.15.1 Inputs

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

15.15.2 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

15.15.3 Output Signal

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

15.16 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.

15.16.1 Inputs

Signal name	Type	Description
din	Analog	Input waveform

15.16.2 Parameters

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

15.16.3 Output Signal

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

15.17 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.

15.18 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.

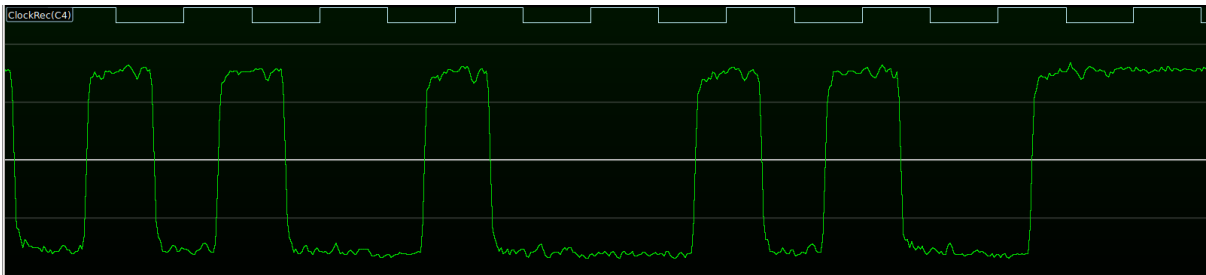


Figure 15.14: Example of CDR PLL on a serial data stream

15.18.1 Inputs

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

15.18.2 Parameters

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

15.18.3 Output Signal

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

15.19 Clock Recovery (UART)

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

15.20 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

15.20.1 Inputs

This filter takes no inputs.

15.20.2 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

15.20.3 Output Signal

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

15.21 CSV Export

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

15.22 CSV Import

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

15.23 Current Shunt

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

15.24 DC Offset

Adds a constant value to each sample in an analog waveform.

15.24.1 Inputs

Signal name	Type	Description
din	Analog	Input waveform

15.24.2 Parameters

Parameter name	Type	Description
Offset	Float	The offset to apply

15.24.3 Output Signal

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

15.25 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.

15.25.1 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

15.25.2 Parameters

This filter takes no parameters.

15.25.3 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.

15.26 DDR1 Command Bus

Decodes the command bus for first-generation DDR SDRAM.

15.27 DDR3 Command Bus

Decodes the command bus for third-generation DDR SDRAM.

15.28 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.

15.28.1 Inputs

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

15.28.2 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

15.28.3 Output Signal

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

15.29 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.

15.29.1 Inputs

Signal name	Type	Description
din	Analog	Input waveform

15.29.2 Parameters

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

15.29.3 Output Signal

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

15.30 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.

15.30.1 Inputs

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

15.30.2 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

15.30.3 Output Signal

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

15.31 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.

15.31.1 Inputs

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

15.31.2 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

15.31.3 Output Signal

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

15.32 Divide

Divides one waveform by another.

15.33 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.

15.34 Downsample

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

15.35 DRAM Clocks

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

15.36 DRAM Trcd

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

15.37 DRAM Trfc

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

15.38 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.

15.39 DVI

Decodes Digital Visual Interface (DVI) video signals.

15.40 Emphasis

Adds pre/de emphasis to a signal.

15.41 Emphasis Removal

Removes pre/de emphasis from a signal.

15.42 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 bandwidth to Fnyquist / 8. The filter properties dialog displays the calculated -3 dB bandwidth based on the current input sample rate.

15.42.1 Inputs

Signal name	Type	Description
in	Analog	Input signal

15.42.2 Parameters

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

15.43 Envelope

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

15.44 Ethernet - 10baseT

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

15.45 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.

15.46 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

15.46.1 Parameters

This filter takes no parameters.

15.46.2 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

15.47 Ethernet - GMII

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

15.48 Ethernet - QSGMII

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

15.49 Ethernet - RGMII

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

15.50 Ethernet - RMII

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

15.51 Ethernet - SGMII

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

15.52 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.

15.53 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).]

15.54 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.

15.55 Eye Bit Rate

Measures the bit rate of an eye pattern.

15.56 Eye Height

Measures the vertical opening of an eye pattern.

15.57 Eye P-P Jitter

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

15.58 Eye Pattern

Calculates an eye pattern.

15.59 Eye Period

Measures the UI width of an eye pattern.

15.60 Eye Width

Measures the horizontal opening of an eye pattern.

15.61 Fall

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

15.62 FFT

Calculates a Fast Fourier Transform and displays the magnitude response.

15.63 FIR

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

15.64 Frequency

Measures the frequency of each cycle in a waveform.

15.65 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?

15.66 Group Delay

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

15.66.1 Inputs

Signal name	Type	Description
Phase	Analog	Phase angle vs frequency

15.66.2 Parameters

This filter takes no parameters.

15.66.3 Output Signal

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

15.67 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.

15.67.1 Inputs

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

15.67.2 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

15.67.3 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.)

15.68 Horizontal Bathtub

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

15.69 HDMI

Decodes HDMI

15.70 I^2C

Decodes the Phillips I^2C bus protocol.

15.71 I^2C EEPROM

Decodes common I^2C EEPROM memory devices

15.72 *I*²*C* Register

Decodes low level *I*²*C* 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.

15.73 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.

15.73.1 Inputs

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

15.73.2 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

15.73.3 Output Signal

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

15.74 Invert

Inverts an analog waveform by negating each sample.

15.75 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).

15.76 IPv4

Internet Protocol version 4

15.77 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.

15.78 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.

15.78.1 Inputs

Signal name	Type	Description
din	Digital	Input waveform

15.78.2 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

15.78.3 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.

15.79 Jitter Spectrum

Calculates an FFT of a TIE waveform.

15.80 JTAG

Joint Test Action Group

15.81 Magnitude

Calculates the magnitude of a complex valued signal

15.82 MDIO

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

15.83 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.

15.84 MIL-STD-1553

Decodes the MIL-STD-1553 avionics data bus.

15.85 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.

15.86 MIPI D-Phy Escape Mode

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

15.87 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.

15.88 MIPI DSI Frame

Converts a MIPI DSI Packet stream into video scanlines.

15.89 MIPI DSI Packet

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

15.90 Moving Average

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

15.91 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.

15.92 Noise

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

15.93 OFDM Demodulator

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

15.94 **Overshoot**

15.95 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.

15.96 Parallel Bus

15.97 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.

15.98 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.

15.99 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.

15.100 PCIe Link Training

Decodes the initial PCIe gen1/2 link training sequence

15.101 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).

15.102 Peak Hold

15.103 Peak-to-Peak

15.104 Period

15.105 Phase

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

15.106 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 average 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 15.15: Example of nonlinear phase of a filter in the stopband

15.106.1 Inputs

Signal name	Type	Description
Phase	Analog	Input waveform

15.106.2 Parameters

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

15.106.3 Output Signal

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

15.107 PRBS

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

15.108 Pulse Width

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

15.108.1 Inputs

Signal name	Type	Description
din	Analog	Input waveform

15.108.2 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.

15.109 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.

15.110 $Rj + BUj$

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

15.111 QSPI

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

15.112 Quadrature

Quadrature pulses from a rotary encoder

15.113 Rise

Calculates the rise time for each cycle of a waveform

15.114 S-Parameter Cascade

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

15.115 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.

15.116 Scale

Multiplies a waveform by a scalar.

15.117 SD Card Command

Decodes the Secure Digital card command bus protocol

15.118 Sine

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

15.119 Spectrogram

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

15.120 SPI

Serial Peripheral Interface.

15.121 SPI Flash

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

15.122 Squelch

Detects periods with no signal.

15.123 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.

15.124 Subtract

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

15.124.1 Inputs

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

15.124.2 Parameters

This filter takes no parameters.

15.124.3 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.

15.125 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.

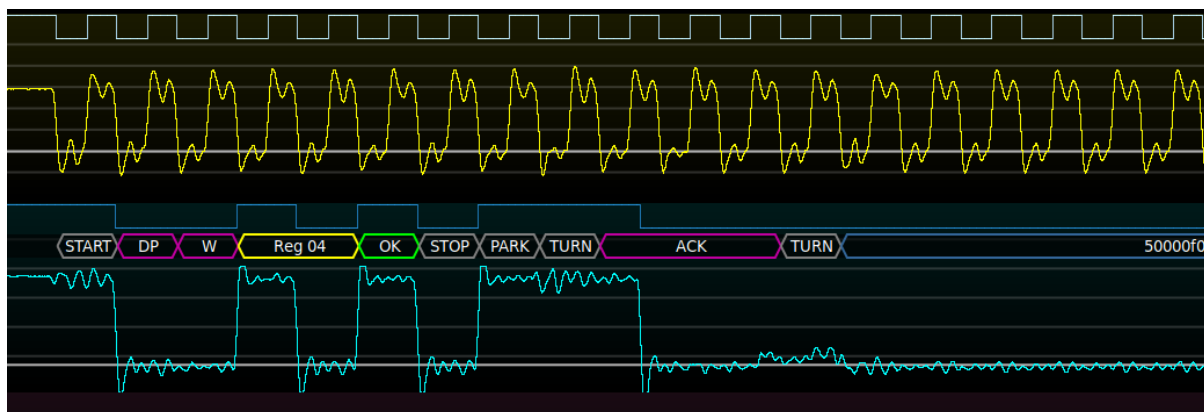


Figure 15.16: Example of SWD protocol decode

15.125.1 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)

15.125.2 Parameters

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

15.125.3 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

15.126 SWD MEM-AP

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

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

15.127 Tachometer

Converts pulses from a tachometer to shaft speed

15.128 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.

15.129 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.

15.130 TDR

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

15.131 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.

15.132 Time Outside Level

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

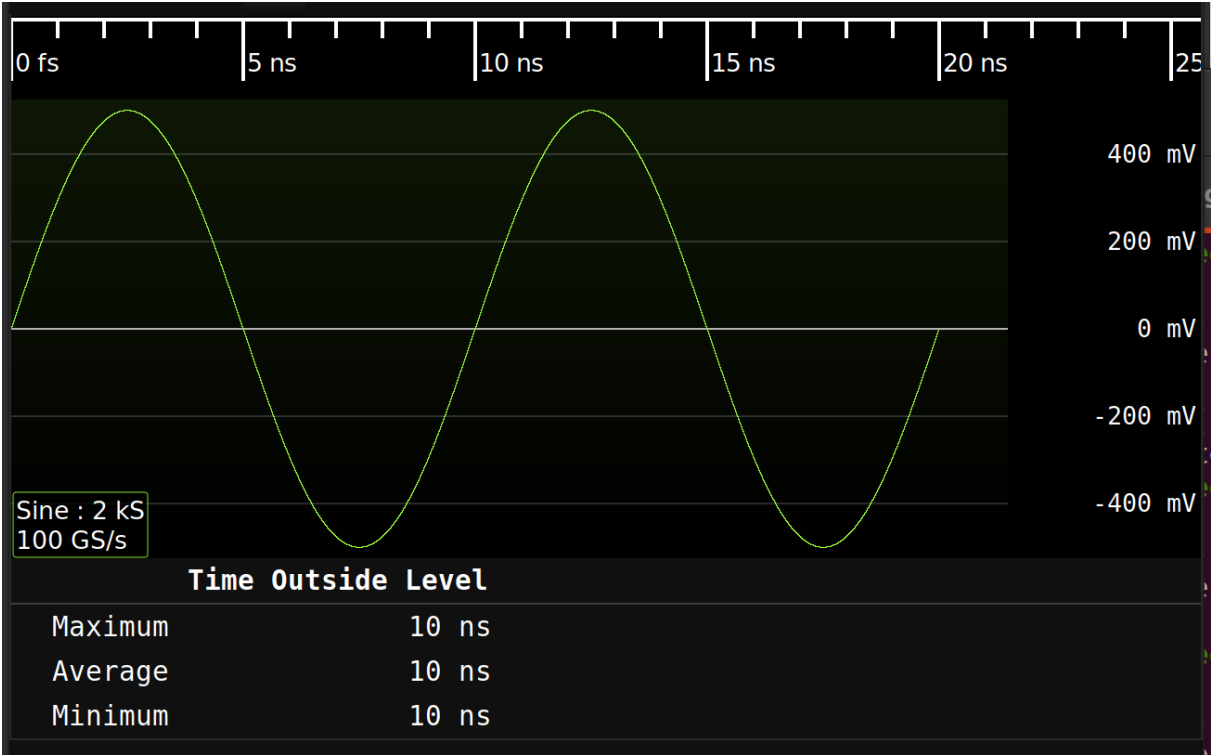


Figure 15.17: Example of time outside high level measurement with a high level threshold of 0mV

15.132.1 Inputs

Signal name	Type	Description
din	Analog	Input waveform

15.132.2 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

15.133 Thermal Diode

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

15.134 Threshold

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

15.134.1 Inputs

Signal name	Type	Description
din	Analog	Input waveform

15.134.2 Parameters

Parameter name	Type	Description
Threshold	Float	Decision threshold

15.134.3 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.

15.135 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).

15.136 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.

15.136.1 Inputs

Signal name	Type	Description
din	Analog	Input waveform

15.136.2 Parameters

This filter takes no parameters.

15.136.3 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.

15.137 Touchstone Export

Saves S-parameter data to a Touchstone file.

15.138 Touchstone Import

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

15.139 Trend

Plots a trend of a scalar value over time

15.140 TRC Import

Loads waveform data from a Teledyne LeCroy TRC waveform file.

15.141 UART

15.142 Unwrapped Phase

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

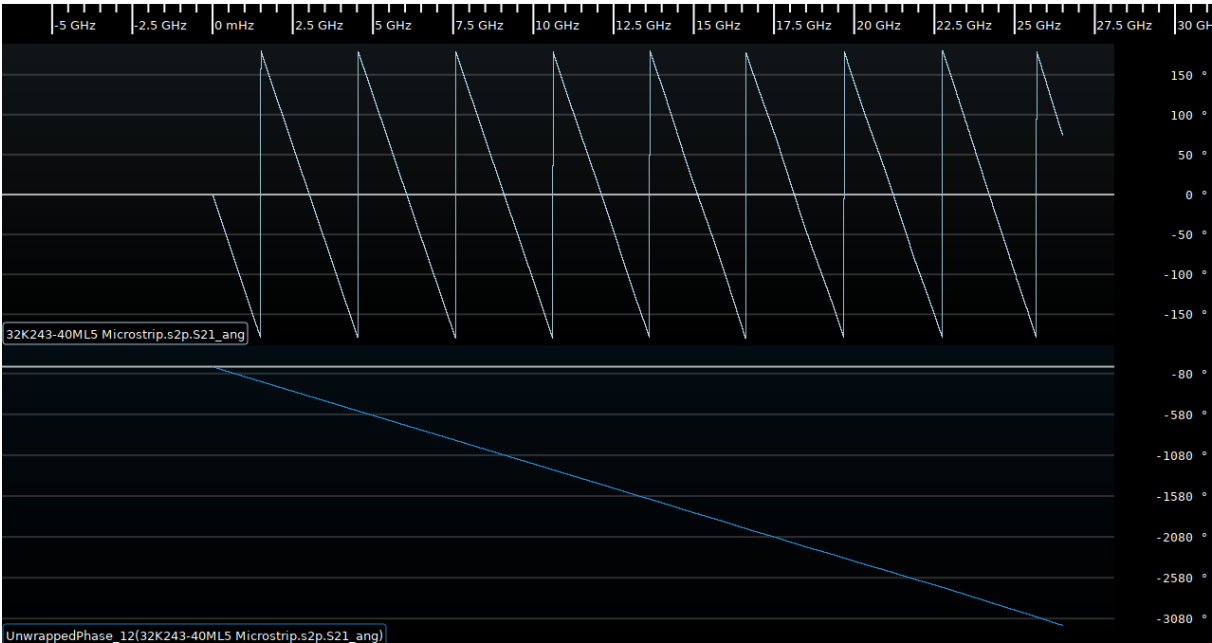


Figure 15.18: Example of wrapped and unwrapped phase of a transmission line

15.142.1 Inputs

Signal name	Type	Description
Phase	Analog	Input waveform

15.142.2 Parameters

This filter takes no parameters.

15.142.3 Output Signal

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

15.143 USB 1.0 / 2.x Activity

15.144 USB 1.0 / 2.x Packet

15.145 USB 1.0 / 2.x PCS

15.146 USB 1.0 / 2.x PMA

15.147 Undershoot

15.148 Upsample

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

15.149 VCD Import

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

15.150 Vector Frequency

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

15.151 Vector Phase

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

15.152 Vertical Bathtub

15.153 VICP

Decodes the Teledyne LeCroy Virtual Instrument Control Protocol (VICP)

15.154 Waterfall

15.155 WAV Import

Loads waveform data from a Microsoft WAV audio file.

15.156 WFM Import

Loads waveform data from a Tektronix .wfm file.

15.157 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.

15.158 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.

15.158.1 Inputs

Signal name	Type	Description
din	Analog or Digital	Input waveform

15.158.2 Parameters

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

15.158.3 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.

Chapter 16

Export Formats

16.1 CSV

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

The first two columns of the generated file contain the X and Y values of the timebase reference channel (the channel selected on the first page of the export wizard). Each sample in this channel maps directly to a row in the generated CSV. Any waveform-type channel can be selected for export, however eye patterns, spectrograms, and other 2D density plots cannot be used.

The second page of the wizard allows additional channels to be added to the export. Each channel maps to an additional column in the generated CSV. The wizard will only allow channels with compatible X axis units; for example if the first channel has X axis units of time then it will not be possible to add a channel with X axis units of frequency to the same CSV.

Since the first channel serves as a timebase reference, all subsequent channels in the exported data share the same timestamps as the first channel. If the channels are not sampled at identical times, the following resampling algorithms are used for additional channels:

- Analog data: linear interpolation
- Digital data: nearest neighbor
- Protocol data: No interpolation. Protocol events are displayed as close as possible to their start timestamp; remaining cells are blank.

16.2 Touchstone

Exports S-parameter data to a Touchstone 1.1 file with an arbitrary number of ports.

The exporter expects waveform data in mag/angle format. Magnitude channels must have X axis units of Hz and Y axis units of dB; angle channels must have X axis units of Hz and Y axis units of degrees. While input data need not be *uniformly* sampled, all input channels must be sampled at the same set of frequencies.

Frequency units for the generated file can be selected as Hz, kHz, MHz, or GHz.

The export wizard allows mag/angle, dB mag/angle, or real/imaginary format to be selected. However, as of this writing only mag/angle is actually implemented. Selecting any other format will result in a warning on the console; the file will be generated in mag/angle format regardless of the user's choice.

Chapter 17

Internals

17.1 Introduction

This chapter provides a high level overview of libscopehal and glscopeclient internals. It is intended for developers to gain an understanding of the overall project architecture and how key pieces fit together, but is not a substitute for the low level API documentation (Doxygen).

Many of the entities described below use a dynamic discovery / registration system. This allows all such classes to be enumerated (and associated with human-readable names), and allows for objects of any registered type - including those provided by plugins - to be created at run time by a factory method given the human-readable class name.

17.2 Instruments

An instrument is an instance of a class derived from `Instrument`, which represents an arbitrary piece of laboratory equipment. As of this writing, an instrument may be an oscilloscope, multimeter, power supply, baseband signal generator, or RF signal generator - or an arbitrary combination of these (for example an oscilloscope with integrated function generator is both an oscilloscope and baseband signal generator).

The type of an instrument is defined by a bit field and may be queried by calling `GetInstrumentTypes()`. Do *not* rely on C++ RTTI to determine the type of an instrument, for example it is incorrect to `dynamic_cast` a `Instrument*` pointer to `Oscilloscope*` to check if the instrument is an oscilloscope. This is because the C++ type of an object is fixed when the driver class is compiled, and the driver may be used with many different instruments with various sets of software and hardware options. In other words, the fact that a given driver supports *some* device that contains multimeter functionality does not in any way imply that the *particular* device you are talking to is a multimeter.

The `Instrument` class provides no functionality other than describing the device (querying make/-model/serial number, assigning display nicknames, and querying feature set). To do any useful work, the object is normally casted to a derived type to gain access to that device class's API.

17.3 SCPI Devices

A SCPI device is an instance of a class derived from `SCPIDevice`, which represents a device which speaks some variant of SCPI. The vast majority of instrument driver classes derive from both `SCPIDevice` and one or more `Instrument` derived classes.

A SCPI device object uses a [transport](#) to communicate with the associated instrument, which avoids the need for the driver class to concern itself with the specifics of how the SCPI commands are transferred to the device.

17.4 Transports

A transport is an instance of a class derived from `SCPITransport`, which provides a means of sending SCPI commands and/or raw byte string data to or from a physical instrument.

Most transports use a single stream in the underlying protocol layer (such as a single TCP socket) to transport both control plane content (SCPI commands) and data plane content (waveform data), however some specialized protocols have multiple physical streams (for example the `SCPItwinLanTransport` transport). For these instruments, the command/reply APIs and raw data APIs may not go to the same place.

All transports must be registered in order to be used by `glscopeclient`. To register a transport class, add the macro `TRANSPORT_INITPROC(FooTransport)` to your class declaration and call `AddTransportClass(FooTransport)` in either the `TransportStaticInit` function within `libscopehal` or the `PluginInit` function of a plugin, as appropriate.

The special class `SCPINullTransport` serves as a `/dev/null` equivalent: it discards anything written to it, and never returns read data. It is primarily intended to be used by the “demo” driver, which does not connect to a real instrument.

While it is in principle possible to create a driver class that talks directly to a device via e.g. a USB API and bypasses the transport model, this is strongly discouraged for user experience and flexibility reasons. Most drivers for such devices (for example the Digilent and Pico drivers) instead consist of two components: a bridge server that converts the instrument API to SCPI commands on one socket and a raw sample data on a second socket, and a `libscopehal`-side driver that converts this to the relevant instrument API.

17.5 Oscilloscopes

An Oscilloscope is an instance of a class derived from `Oscilloscope`, which represents a device for acquiring sampled digital data. All actual oscilloscopes use this API, as do some other instruments such as spectrum analyzers. Most oscilloscope driver classes derive from `SCPIOscilloscope` rather than directly from `Oscilloscope`, as they use SCPI to communicate with the hardware.

An oscilloscope may have zero or more [channels](#).¹

At any given time, an oscilloscope has exactly one [trigger](#) associated with it. A trigger has inputs and properties just like a [filter](#), since both are derived from `FlowGraphNode`. Most triggers take at least one input, however zero-input triggers are possible (for example, triggering on AC mains zero crossings).

Every oscilloscope driver class must contain a public static method `GetDriverNameInternal()`, which returns a `std::string` containing a short, human readable name for the driver (for example “agilent” or “pico”). By convention, the driver name should consist of lowercase letters and numbers only - no spaces, punctuation, or capital letters.

Just like transports, every oscilloscope driver class must be registered in the dynamic cre-

¹All currently extant implementations have at least one channel, however it is plausible that a zero-channel instrument might exist in the future (for example, some sort of external trigger controller that exposes the same trigger API as a conventional oscilloscope) so the API allows for this.

ation table by invoking `OSCILLOSCOPE_INITPROC(MyOscilloscope)` in the class declaration and `AddDriverClass(MyOscilloscope)` in `DriverStaticInit` or `PluginInit`.

17.6 Channels

A channel is an instance of a class derived from `OscilloscopeChannel`, which represents a single source of data and associated controls. A channel may be associated with an [oscilloscope](#), or it may be a [filter](#) which is not associated with any particular physical instrument.

Channels of an oscilloscope generally map 1:1 to analog front ends. Most commonly they are also 1:1 with instrument front panel connectors, however there are some notable exceptions. Some high end oscilloscopes (such as the Teledyne LeCroy WaveMaster family) have multiple inputs with a multiplexer feeding a single front end; this ensemble is considered to be a single channel by `libscopehal`. Network analyzers have separate channels for receive and reflected power, for example S_{11} and S_{12} of a VNA are measured at the same physical port on the instrument but separate channels in `libscopehal`.

A channel normally has one or more output [streams](#), however in some less common situations (such as dedicated trigger inputs) there may be zero streams.

Channels are reference counted: when at least one filter or waveform view is consuming the output of a channel it will be automatically enabled. When the last user of a channel is removed, the channel will be disabled and, if a filter, deleted.

Various properties can be configured on channels, such as gain/offset and bandwidth limiters. Depending on whether the channel is a filter or not, or what kind of oscilloscope it is connected to, not all of these settings may be available.

17.7 Streams

A stream is an output from a [channel](#). Most channels of physical oscilloscopes have only a single stream, however some have multiple (for example I and Q from a realtime spectrum analyzer, or magnitude and angle from a VNA). Many filters have multiple output streams, for example each channel of an imported WAV file is a separate stream of the import filter.

All streams of a channel must have the same X axis unit, however they may have independent Y axis units.

The set of streams provided by a filter may change at run time, most commonly if an import filter is pointed to a new file. When a filter changes its set of output streams, it must emit the `m_outputsChangedSignal` signal so that other code can handle the change appropriately.

17.8 Triggers

TODO: write this section

17.9 Waveforms

A waveform is a class derived from `WaveformBase` which stores a vector of sampled data. The `AnalogWaveform` and `DigitalWaveform` classes store 32-bit floating point and Boolean data respec-

tively. Additional waveform classes are defined by many protocol decodes to store data of arbitrary class type.

The units for X and Y axis are not specified in the waveform, but are properties of the channel / stream that the waveform came from. Most commonly, for analog oscilloscope waveforms, the X axis unit is femtoseconds and the Y axis unit is volts - but other units may be encountered, for example the output of a FFT has X axis units in Hz and Y axis in dBm.²

Waveforms store timestamp / header metadata as well as three vectors of data:

- `m_offsets`: start time of each sample
- `m_durations`: length of each sample
- `m_samples`: actual sample data

All three vectors must always be the same length. (The struct-of-arrays memory format allows for better cache locality and is more SIMD-friendly than an array-of-structs format.)

Sample offsets and durations are measured in time base units (defined by `m_timescale`). This is commonly the sample rate of the ADC or logic analyzer that acquired the data, however for upsampled or interpolated data smaller time scale values - as low as 1 - may be used. A static offset, the “trigger phase” (`m_triggerPhase`), measured in raw X axis units and not scaled by `m_timescale`, is added to the timestamp of every signal after scaling by the time base unit. This is commonly used to apply a sub-sample offset to a waveform for trigger interpolation or de-skewing.

The final timestamp of sample i , in X axis units, is thus `m_offsets[i]*m_timescale + m_triggerPhase`.

Note that the offset/duration allows samples to have arbitrary length and spacing; i.e. waveforms are inherently sparse. This is necessary to support protocol events, irregularly sampled data, etc. Sample timestamps must increase monotonically: sample $i+1$ must start at or after the end of sample i .

The majority of waveforms (such as those coming directly off an oscilloscope) will be uniformly sampled, which renders the sparse storage format inefficient. A waveform of N samples which has a duration of 1 for every sample, and offsets ranging from 0 to $N-1$, is considered to be “dense packed” and should have the `m_densePacked` flag set to enable various processing optimizations. The dense pack flag must NOT be set on a waveform which does not meet these criteria as this can lead to incorrect output.

Filters presented with input marked as dense packed are free to ignore the timestamp and duration flags at their input. Filters generating densely packed output should set the dense pack flag, however they must still fill the timestamp and duration vectors for use by filters which do not have an optimized special case for dense packed inputs.

17.10 Filters

TODO: write this section

17.11 Plugins

A plugin is a shared library which may contain transports, drivers, filters, and export wizards. All of these must be registered in a function called `PluginInit` exported with extern "C" linkage.

²Some variables and methods throughout the project (especially in older code) use “time” or “voltage” terminology to refer to the current X or Y axis units. This will likely be changed through refactoring over the long term.

Plugins are automatically loaded at startup by `glscopeclient`, however standalone applications using `libscopehal` must explicitly call `InitializePlugins()` to load them.

17.11.1 Linux

On Linux, plugins are loaded from the following directories:

- `/usr/lib/scopehal/plugins`
- `/usr/local/lib/scopehal/plugins`
- `/.scopehal/plugins`
- Executable directory, if not under `/usr`

17.11.2 Windows

On Windows, plugins are loaded from the following directories:

- (Executable directory) `\plugins`