# ngscopeclient Operator Manual

ngscopeclient Project

September 26, 2026
v0.3-1-g5e7897a7

Copyright ©2012-2026 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 Documentation Conventions . . . 3
- 1.3 Key Concepts . . . 4
  - 1.3.1 User Interface . . . 4
  - 1.3.2 Design Philosophy . . . 4
  - 1.3.3 Terminology . . . 5
- 1.4 Stability Notes . . . 6
- 1.5 Revision History . . . 6

**2 Legal Notices** **7**
- 2.1 Introduction . . . 7
- 2.2 License Agreement . . . 7
- 2.3 Trademarks . . . 8
- 2.4 Third Party Licenses . . . 8
  - 2.4.1 Optional, only used if scopehal-plugins-gpl plugin is installed . . . 9
  - 2.4.2 Only in unit tests . . . 9

**3 Getting Started** **11**
- 3.1 Host System Requirements . . . 11
  - 3.1.1 Operating Systems . . . 11
  - 3.1.2 Processors . . . 11
  - 3.1.3 Mouse . . . 11
  - 3.1.4 GPU . . . 11
  - 3.1.5 RAM . . . 12
- 3.2 Instrument Support . . . 12
- 3.3 Installation . . . 12
  - 3.3.1 Official Releases . . . 12
  - 3.3.2 Development Builds . . . 13
- 3.4 Compilation . . . 13
  - 3.4.1 Linux . . . 13
  - 3.4.2 macOS . . . 15
  - 3.4.3 Windows . . . 16
- 3.5 Running ngscopeclient . . . 17
  - 3.5.1 General arguments . . . 17
  - 3.5.2 Console verbosity arguments . . . 18

**4 Tutorials** **19**
- 4.1 The Basics . . . 19
  - 4.1.1 Connecting to an Oscilloscope . . . 19
  - 4.1.2 Acquiring Waveforms . . . 20
  - 4.1.3 Navigating the Y Axis . . . 21
  - 4.1.4 Navigating the X Axis . . . 21

**5 Main Window** **23**
5.1 Menu . . . 23
5.1.1 File . . . 23
5.1.2 View . . . 24
5.1.3 Add . . . 24
5.1.4 Setup . . . 25
5.1.5 Window . . . 25
5.1.6 Debug . . . 26
5.1.7 Help . . . 26
5.2 Toolbar . . . 27
5.2.1 Capture buttons . . . 27
5.2.2 History . . . 27
5.2.3 Refresh Settings . . . 27
5.2.4 Clear Sweeps . . . 28
5.2.5 Fullscreen . . . 28
5.2.6 Intensity slider . . . 28
5.2.7 Persistence slider . . . 29

**6 Dialogs** **31**
6.1 Lab Notes . . . 31
6.2 Log Viewer . . . 32
6.3 Measurements . . . 33
6.4 Performance Metrics . . . 34
6.4.1 Rendering . . . 34
6.4.2 Filter graph . . . 34
6.4.3 Acquisition . . . 35
6.4.4 Memory . . . 35
6.5 Preferences . . . 35
6.5.1 Appearance . . . 35
6.5.2 Drivers . . . 38
6.5.3 Files . . . 38
6.5.4 Miscellaneous . . . 39
6.5.5 Power . . . 39
6.6 Speed Bump . . . 39

**7 Workspaces and Window Management** **41**
7.1 Window Management and Docking . . . 41
7.2 Workspaces . . . 42

**8 Waveform Groups** **45**
8.1 Managing Groups . . . 46

**9 Waveform Views** **49**
9.1 Navigation . . . 49
9.2 Plot Area . . . 49
9.3 Y Axis Scale . . . 50
9.4 Channel Label . . . 50
9.5 Cursors and Markers . . . 51
9.5.1 Horizontal Cursors . . . 52
9.5.2 Vertical Cursors . . . 52
9.5.3 Markers . . . 54

**10 History** **57**

10.1 Pinning . . . 58
10.2 Labeling . . . 58

**11 Stream Browser** **59**
11.1 Introduction . . . 59
11.2 Filters . . . 60
11.3 Instruments . . . 60
11.4 Adding streams . . . 60

**12 Filter Graph Editor** **61**
12.1 Introduction . . . 61
12.2 Interaction . . . 62
12.3 Grouping . . . 64
12.4 Performance Analysis . . . 66

**13 Transports** **67**
13.1 gpib . . . 67
13.2 lan . . . 67
13.3 lxi . . . 68
13.4 null . . . 68
13.5 socketcan . . . 68
13.6 twinlan . . . 68
13.7 uart . . . 69
13.8 usbtmc . . . 69
13.9 vicp . . . 69
13.10 hid . . . 69

**14 BERT Drivers** **71**
14.1 Antikernel Labs . . . 71
14.1.1 akl.crossbar . . . 71
14.2 MultiLANE . . . 71
14.2.1 mlbert . . . 71

**15 Function Generator Drivers** **73**
15.1 Owon . . . 73
15.1.1 owon_xdg . . . 73
15.2 Rigol . . . 73
15.2.1 rigol_awg . . . 73
15.3 Siglent . . . 74
15.3.1 siglent_sdg . . . 74

**16 Electronic Load Drivers** **75**
16.1 Siglent . . . 75
16.1.1 siglent_load . . . 75

**17 Multimeter Drivers** **77**
17.1 Owon . . . 77
17.1.1 owon_xdm . . . 77
17.2 Rohde & Schwarz . . . 77
17.2.1 rs_hmc8012 . . . 77

**18 Miscellaneous Drivers** **79**
18.1 Antikernel Labs . . . 79

18.1.1 akl.gpio . . . 79
18.1.2 akl.vio . . . 79
18.2 Generic . . . 79
18.2.1 csvstream . . . 79

**19 Oscilloscope Drivers** **81**
19.1 Agilent . . . 81
19.1.1 agilent . . . 81
19.2 Antikernel Labs . . . 82
19.2.1 akl.ila . . . 82
19.3 Demo . . . 83
19.4 Batronix . . . 83
19.5 Digilent . . . 84
19.5.1 digilent . . . 85
19.6 DrAndyHaas . . . 85
19.6.1 haasoscope pro . . . 85
19.7 DreamSource Lab . . . 85
19.7.1 dslabs . . . 86
19.8 EEVengers . . . 86
19.8.1 thunderscope . . . 86
19.8.2 Breaking Changes . . . 87
19.9 Enjoy Digital . . . 87
19.10 Generic . . . 87
19.10.1 socketcan . . . 87
19.11 Hantek . . . 87
19.12 Keysight . . . 87
19.12.1 agilent . . . 87
19.12.2 keysightdca . . . 88
19.13 Pico Technology . . . 88
19.13.1 pico . . . 88
19.14 Rigol . . . 89
19.14.1 rigol . . . 89
19.15 Rohde & Schwarz . . . 90
19.15.1 rs . . . 90
19.15.2 rs_rto6 . . . 90
19.15.3 rs_rtb2k . . . 90
19.16 Saleae . . . 91
19.17 Siglent . . . 91
19.18 Teledyne LeCroy / LeCroy . . . 93
19.18.1 lecroy . . . 94
19.18.2 lecroy_fwp . . . 95
19.19 Tektronix . . . 95
19.19.1 Note regarding “lan" transport on MSO5/6 . . . 96
19.20 tinySA . . . 96
19.21 Xilinx . . . 96

**20 SDR Drivers** **97**
20.1 Ettus Research . . . 97
20.1.1 uhd . . . 97
20.2 Microphase . . . 97

**21 Spectrometer Drivers** **99**

21.1 ASEQ Instruments . . . 99
21.1.1 aseq . . . 99

**22 Power Supply Drivers** **101**
22.1 Alientek . . . 101
22.1.1 alientek_dp . . . 101
22.2 GW Instek . . . 101
22.2.1 gwinstek_gpdx303s . . . 101
22.3 Kuaiqu . . . 101
22.3.1 kuaiqu_psu . . . 102
22.4 Riden . . . 102
22.4.1 kuaiqu_psu . . . 102
22.5 Rigol . . . 102
22.5.1 rigol_dp8xx . . . 102
22.6 Rohde & Schwarz . . . 102
22.6.1 rs_hmc804x . . . 102
22.7 Siglent . . . 102
22.7.1 siglent_spd . . . 103
22.8 Sinilink . . . 103
22.8.1 siniLink . . . 103

**23 RF Generator Drivers** **105**
23.1 Siglent . . . 105
23.1.1 siglent_ssg . . . 105

**24 VNA Drivers** **107**
24.1 Copper Mountain . . . 107
24.1.1 coppermt . . . 107
24.2 NanoVNA . . . 108
24.2.1 nanovna . . . 108
24.3 Pico Technology . . . 108
24.3.1 picovna . . . 108

**25 Triggers** **109**
25.1 Trigger Properties . . . 109
25.2 Serial Pattern Triggers . . . 109
25.3 Dropout . . . 111
25.3.1 Inputs . . . 111
25.3.2 Parameters . . . 111
25.4 Edge . . . 111
25.4.1 Inputs . . . 111
25.4.2 Parameters . . . 111
25.5 Glitch . . . 111
25.6 Pulse Width . . . 112
25.6.1 Parameters . . . 112
25.7 Runt . . . 112
25.7.1 Parameters . . . 112
25.8 Slew Rate . . . 112
25.8.1 Parameters . . . 113
25.9 UART . . . 113
25.9.1 Inputs . . . 113
25.9.2 Parameters . . . 113
25.10 Window . . . 113

25.10.1 Parameters . . . 114

**26 Filters** **115**
26.1 Introduction . . . 115
26.1.1 Key Concepts . . . 115
26.1.2 Conventions . . . 115
26.2 1-Wire . . . 117
26.2.1 Inputs . . . 117
26.2.2 Parameters . . . 117
26.2.3 Output Signal . . . 117
26.3 128b/130b . . . 118
26.3.1 Inputs . . . 118
26.3.2 Parameters . . . 118
26.3.3 Output Signal . . . 119
26.3.4 Breaking Changes . . . 119
26.4 2-Port Shunt Through . . . 120
26.5 64b/66b . . . 121
26.5.1 Inputs . . . 121
26.5.2 Parameters . . . 121
26.5.3 Output Signal . . . 122
26.5.4 Breaking Changes . . . 122
26.6 8B/10B (IBM) . . . 123
26.6.1 Inputs . . . 123
26.6.2 Parameters . . . 124
26.6.3 Output Signal . . . 124
26.6.4 Breaking Changes . . . 124
26.7 8B/10B (TMDS) . . . 125
26.7.1 Inputs . . . 125
26.7.2 Parameters . . . 125
26.7.3 Output Signal . . . 126
26.7.4 Breaking Changes . . . 126
26.8 AC Couple . . . 127
26.8.1 Inputs . . . 128
26.8.2 Parameters . . . 128
26.8.3 Output Signal . . . 128
26.9 AC RMS . . . 129
26.9.1 Inputs . . . 129
26.9.2 Parameters . . . 129
26.9.3 Output Signal . . . 130
26.10 Add . . . 131
26.10.1 Inputs . . . 131
26.10.2 Parameters . . . 132
26.10.3 Output Signal . . . 132
26.11 ADL5205 . . . 133
26.11.1 Inputs . . . 133
26.11.2 Parameters . . . 133
26.11.3 Output Signal . . . 133
26.12 APB . . . 134
26.12.1 Inputs . . . 135
26.12.2 Parameters . . . 135
26.12.3 Output Signal . . . 135
26.13 Area Under Curve . . . 136

26.13.1 Inputs . . . 136
26.13.2 Parameters . . . 137
26.13.3 Output Signal . . . 137
26.14 Autocorrelation . . . 139
26.14.1 Inputs . . . 140
26.14.2 Parameters . . . 140
26.14.3 Output Signal . . . 140
26.15 Average . . . 141
26.15.1 Inputs . . . 141
26.15.2 Parameters . . . 141
26.15.3 Output Signal . . . 142
26.16 Bandwidth . . . 143
26.16.1 Inputs . . . 143
26.16.2 Parameters . . . 143
26.16.3 Output Signal . . . 144
26.17 Base . . . 145
26.17.1 Inputs . . . 145
26.17.2 Parameters . . . 145
26.17.3 Output Signal . . . 146
26.18 BIN Import . . . 147
26.18.1 Inputs . . . 147
26.18.2 Parameters . . . 147
26.18.3 Output Signal . . . 147
26.19 Burst Width . . . 148
26.19.1 Inputs . . . 148
26.19.2 Parameters . . . 148
26.19.3 Output Signal . . . 149
26.20 Bus Heatmap . . . 150
26.20.1 Parameters . . . 150
26.20.2 Output Signal . . . 151
26.21 CAN . . . 152
26.21.1 Inputs . . . 152
26.21.2 Parameters . . . 153
26.21.3 Output Signal . . . 153
26.21.4 Protocol Analyzer . . . 153
26.22 CAN Analyzer . . . 154
26.23 CAN Bitmask . . . 155
26.23.1 Inputs . . . 155
26.23.2 Parameters . . . 155
26.23.3 Output Signal . . . 156
26.24 Can-Utils Import . . . 157
26.25 Channel Emulation . . . 158
26.25.1 Inputs . . . 159
26.25.2 Parameters . . . 159
26.25.3 Output Signal . . . 159
26.26 Clip . . . 160
26.26.1 Inputs . . . 160
26.26.2 Parameters . . . 160
26.26.3 Output Signal . . . 161
26.27 Clock Jitter (TIE) . . . 162
26.27.1 Inputs . . . 162
26.27.2 Parameters . . . 163

26.27.3 Output Signal . . . 163
26.28 Clock Recovery (D-PHY HS Mode) . . . 164
26.29 Clock Recovery (PLL) . . . 165
26.29.1 Inputs . . . 166
26.29.2 Parameters . . . 166
26.29.3 Output Signal . . . 166
26.30 Clock Recovery (UART) . . . 167
26.30.1 Inputs . . . 167
26.30.2 Parameters . . . 167
26.30.3 Output Signal . . . 167
26.31 Complex Import . . . 168
26.31.1 Inputs . . . 169
26.31.2 Parameters . . . 169
26.31.3 Output Signal . . . 169
26.32 Complex Spectrogram . . . 170
26.32.1 Inputs . . . 171
26.32.2 Parameters . . . 171
26.32.3 Output Signal . . . 171
26.33 Constant . . . 172
26.33.1 Inputs . . . 172
26.33.2 Parameters . . . 172
26.33.3 Output Signal . . . 172
26.34 Constellation . . . 173
26.34.1 Inputs . . . 173
26.34.2 Parameters . . . 174
26.34.3 Output Signal . . . 174
26.34.4 Breaking Changes . . . 174
26.35 Coupler De-Embed . . . 175
26.35.1 Inputs . . . 176
26.35.2 Output Signal . . . 176
26.36 CSV Export . . . 177
26.36.1 Inputs . . . 177
26.36.2 Parameters . . . 177
26.36.3 Output Signal . . . 177
26.37 CSV Import . . . 179
26.37.1 Inputs . . . 179
26.37.2 Parameters . . . 179
26.37.3 Output Signal . . . 179
26.38 CTLE . . . 180
26.38.1 Inputs . . . 180
26.38.2 Parameters . . . 181
26.38.3 Output Signal . . . 181
26.38.4 Breaking Changes . . . 181
26.39 Current Shunt . . . 182
26.40 DCD . . . 183
26.40.1 Inputs . . . 183
26.40.2 Parameters . . . 184
26.40.3 Output Signal . . . 184
26.41 DDJ . . . 185
26.41.1 Inputs . . . 186
26.41.2 Parameters . . . 186
26.41.3 Output Signal . . . 186

26.41.4 Breaking Changes . . . 186
26.42 DDR1 Command Bus . . . 187
26.42.1 Inputs . . . 188
26.42.2 Parameters . . . 188
26.42.3 Output Signal . . . 188
26.43 DDR3 Command Bus . . . 189
26.44 De-Embed . . . 190
26.44.1 Inputs . . . 191
26.44.2 Parameters . . . 191
26.44.3 Output Signal . . . 191
26.45 Deskew . . . 192
26.45.1 Inputs . . . 192
26.45.2 Parameters . . . 193
26.45.3 Output Signal . . . 193
26.46 Digital Constant . . . 194
26.46.1 Inputs . . . 194
26.46.2 Parameters . . . 194
26.46.3 Output Signal . . . 194
26.47 Digital to Analog . . . 195
26.47.1 Inputs . . . 195
26.47.2 Parameters . . . 195
26.47.3 Output Signal . . . 195
26.48 Digital to NRZ . . . 196
26.48.1 Inputs . . . 196
26.48.2 Parameters . . . 197
26.48.3 Output Signal . . . 197
26.49 Digital to PAM4 . . . 198
26.49.1 Inputs . . . 199
26.49.2 Parameters . . . 199
26.49.3 Output Signal . . . 199
26.50 DisplayPort - Aux Channel . . . 200
26.50.1 Inputs . . . 200
26.50.2 Parameters . . . 201
26.50.3 Output Signal . . . 201
26.51 DisplayPort - Main Link . . . 202
26.51.1 Inputs . . . 203
26.51.2 Parameters . . . 203
26.51.3 Output Signal . . . 203
26.52 Divide . . . 204
26.52.1 Inputs . . . 204
26.52.2 Parameters . . . 204
26.52.3 Output Signal . . . 205
26.53 Downconvert . . . 206
26.53.1 Inputs . . . 207
26.53.2 Parameters . . . 207
26.53.3 Output Signal . . . 207
26.54 Downsample . . . 208
26.54.1 Inputs . . . 208
26.54.2 Parameters . . . 209
26.54.3 Output Signal . . . 209
26.55 DRAM Clocks . . . 210
26.56 DRAM Trcd . . . 211

26.57 DRAM Trfc . . . 212
26.58 Duty Cycle . . . 213
26.58.1 Inputs . . . 213
26.58.2 Parameters . . . 213
26.58.3 Output Signal . . . 213
26.59 DVI . . . 214
26.59.1 Inputs . . . 215
26.59.2 Parameters . . . 215
26.59.3 Output Signal . . . 215
26.60 Emphasis . . . 216
26.60.1 Inputs . . . 216
26.60.2 Parameters . . . 217
26.60.3 Output Signal . . . 217
26.61 Emphasis Removal . . . 218
26.61.1 Inputs . . . 218
26.61.2 Parameters . . . 219
26.61.3 Output Signal . . . 219
26.62 Enhanced Resolution . . . 220
26.62.1 Inputs . . . 221
26.62.2 Parameters . . . 221
26.62.3 Output Signal . . . 221
26.63 Envelope . . . 222
26.63.1 Inputs . . . 222
26.63.2 Parameters . . . 223
26.63.3 Output Signal . . . 223
26.64 Ethernet - 10baseT . . . 224
26.64.1 Inputs . . . 224
26.64.2 Parameters . . . 224
26.64.3 Output Signal . . . 225
26.65 Ethernet - 100baseT1 . . . 226
26.65.1 Inputs . . . 227
26.65.2 Parameters . . . 227
26.65.3 Output Signal . . . 227
26.65.4 Breaking Changes . . . 227
26.66 Ethernet - 100baseT1 Link Training . . . 228
26.66.1 Inputs . . . 228
26.66.2 Parameters . . . 229
26.66.3 Output Signal . . . 229
26.66.4 Breaking Changes . . . 229
26.67 Ethernet - 100baseTX . . . 230
26.67.1 Inputs . . . 230
26.67.2 Parameters . . . 230
26.67.3 Output Signal . . . 231
26.67.4 Breaking Changes . . . 231
26.68 Ethernet - 1000baseX . . . 232
26.68.1 Parameters . . . 232
26.68.2 Output Signal . . . 232
26.69 Ethernet - 10Gbase-R . . . 234
26.69.1 Parameters . . . 234
26.69.2 Output Signal . . . 234
26.70 Ethernet - QSGMII . . . 235
26.70.1 Inputs . . . 235

26.70.2 Parameters . . . 236
26.70.3 Output Signal . . . 236
26.71 Ethernet - RGMII . . . 237
26.71.1 Inputs . . . 238
26.71.2 Parameters . . . 238
26.71.3 Output Signal . . . 238
26.72 Ethernet - RMII . . . 239
26.72.1 Inputs . . . 240
26.72.2 Parameters . . . 240
26.72.3 Output Signal . . . 240
26.73 Ethernet - SGMII . . . 241
26.73.1 Inputs . . . 241
26.73.2 Parameters . . . 242
26.73.3 Output Signal . . . 242
26.74 Ethernet Autonegotiation . . . 243
26.74.1 Inputs . . . 244
26.74.2 Parameters . . . 244
26.74.3 Output Signal . . . 244
26.75 Ethernet Autonegotiation Page . . . 245
26.75.1 Inputs . . . 246
26.75.2 Parameters . . . 246
26.75.3 Output Signal . . . 246
26.76 Ethernet Base-X Autonegotiation . . . 247
26.77 Ethernet Clause 73 Autonegotiation . . . 248
26.78 Exponential Moving Average . . . 249
26.78.1 Inputs . . . 249
26.78.2 Parameters . . . 250
26.78.3 Output Signal . . . 250
26.79 Eye Bit Rate . . . 251
26.79.1 Inputs . . . 251
26.79.2 Parameters . . . 251
26.79.3 Output Signal . . . 252
26.80 Eye Height . . . 253
26.80.1 Inputs . . . 253
26.80.2 Parameters . . . 254
26.80.3 Output Signal . . . 254
26.81 Eye P-P Jitter . . . 255
26.81.1 Inputs . . . 255
26.81.2 Parameters . . . 255
26.81.3 Output Signal . . . 256
26.82 Eye Pattern . . . 257
26.82.1 Inputs . . . 257
26.82.2 Parameters . . . 258
26.82.3 Output Signal . . . 258
26.83 Eye Period . . . 259
26.83.1 Inputs . . . 259
26.83.2 Parameters . . . 259
26.83.3 Output Signal . . . 260
26.84 Eye Width . . . 261
26.84.1 Inputs . . . 261
26.84.2 Parameters . . . 262
26.84.3 Output Signal . . . 262

26.85 Fall . . . 263
26.85.1 Inputs . . . 263
26.85.2 Parameters . . . 264
26.85.3 Output Signal . . . 264
26.86 FFT . . . 265
26.86.1 Inputs . . . 266
26.86.2 Parameters . . . 266
26.86.3 Output Signal . . . 266
26.87 FIR . . . 267
26.88 Frequency . . . 268
26.89 Full Width at Half Maximum . . . 269
26.89.1 Inputs . . . 269
26.89.2 Parameters . . . 269
26.89.3 Output Signal . . . 269
26.90 Gate . . . 270
26.91 Glitch Removal . . . 271
26.91.1 Inputs . . . 271
26.91.2 Parameters . . . 271
26.91.3 Output Signal . . . 271
26.92 Group Delay . . . 272
26.92.1 Inputs . . . 272
26.92.2 Parameters . . . 272
26.92.3 Output Signal . . . 272
26.93 Histogram . . . 273
26.93.1 Inputs . . . 273
26.93.2 Parameters . . . 273
26.93.3 Output Signal . . . 273
26.94 Horizontal Bathtub . . . 274
26.95 HDMI . . . 275
26.96 $I^2C$ . . . 276
26.97 $I^2C$ EEPROM . . . 277
26.98 $I^2C$ Register . . . 278
26.99 IBIS Driver . . . 279
26.99.1 Inputs . . . 279
26.99.2 Parameters . . . 279
26.99.3 Output Signal . . . 279
26.100 Invert . . . 280
26.101 Intel eSPI . . . 281
26.102 IPv4 . . . 282
26.103 IQ Demux . . . 283
26.104 IQ Squelch . . . 284
26.105 J1939 Analog . . . 285
26.106 J1939 Bitmask . . . 286
26.107 J1939 PDU . . . 287
26.108 J1939 Source Match . . . 288
26.109 J1939 Transport . . . 289
26.110 Jitter . . . 290
26.110.1 Inputs . . . 290
26.110.2 Parameters . . . 290
26.110.3 Output Signal . . . 290
26.111 Jitter Spectrum . . . 291
26.112 JTAG . . . 292

26.113 Magnitude . . . 293
26.114 Maximum . . . 294
26.114.1 Inputs . . . 294
26.114.2 Parameters . . . 294
26.114.3 Output Signal . . . 294
26.115 MDIO . . . 295
26.116 Memory . . . 296
26.117 MIL-STD-1553 . . . 297
26.118 Minimum . . . 298
26.118.1 Inputs . . . 298
26.118.2 Parameters . . . 298
26.118.3 Output Signal . . . 298
26.119 MIPI D-Phy Data . . . 299
26.120 MIPI D-Phy Escape Mode . . . 300
26.121 MIPI D-Phy Symbol . . . 301
26.122 MIPI DSI Frame . . . 302
26.123 MIPI DSI Packet . . . 303
26.124 Moving Average . . . 304
26.125 Multiply . . . 305
26.126 NCO . . . 306
26.127 Noise . . . 307
26.128 Overshoot . . . 308
26.129 PAM4 Demodulator . . . 309
26.130 PAM Edge Detector . . . 310
26.131 PcapNG Export . . . 311
26.132 PcapNG Import . . . 312
26.133 PCIe Data Link . . . 313
26.134 PCIe Gen 1/2 Logical . . . 314
26.135 PCIe Gen 3/4/5 Logical . . . 315
26.136 PCIe Link Training . . . 316
26.137 PCIe Transport . . . 317
26.138 Peak Hold . . . 318
26.139 Peak-to-Peak . . . 319
26.140 Peaks . . . 320
26.141 Period . . . 321
26.142 Phase . . . 322
26.143 Phase Nonlinearity . . . 323
26.143.1 Inputs . . . 324
26.143.2 Parameters . . . 324
26.143.3 Output Signal . . . 324
26.144 Point Sample . . . 325
26.145 PRBS . . . 326
26.146 Pulse Width . . . 327
26.146.1 Inputs . . . 327
26.146.2 Output Signal . . . 327
26.147 QSPI . . . 328
26.148 Quadrature . . . 329
26.149 Reference Plane Extension . . . 330
26.150 RGB LED . . . 331
26.151 RIS . . . 332
26.152 Rise . . . 333
26.153 Rj + BUj . . . 334

26.154 RMS . . . 335
26.154.1 Inputs . . . 335
26.154.2 Parameters . . . 335
26.154.3 Output Signal . . . 335
26.155 S-Parameter Cascade . . . 336
26.156 Sawtooth . . . 337
26.157 S-Parameter De-Embed . . . 338
26.158 Scalar Pulse Delay . . . 339
26.159 Scalar Stairstep . . . 340
26.160 SD Card Command . . . 341
26.161 Setup / Hold . . . 342
26.161.1 Inputs . . . 342
26.161.2 Parameters . . . 342
26.161.3 Output Signal . . . 342
26.162 Sine . . . 343
26.163 SNR . . . 344
26.163.1 Inputs . . . 344
26.163.2 Parameters . . . 344
26.163.3 Output Signal . . . 344
26.164 Spectrogram . . . 345
26.165 SPI . . . 346
26.166 SPI Flash . . . 347
26.167 Squelch . . . 348
26.168 Step . . . 349
26.169 Subtract . . . 350
26.169.1 Inputs . . . 350
26.169.2 Parameters . . . 350
26.169.3 Output Signal . . . 350
26.170 SWD . . . 351
26.170.1 Inputs . . . 351
26.170.2 Parameters . . . 351
26.170.3 Output Signal . . . 351
26.171 SWD MEM-AP . . . 353
26.172 Tachometer . . . 354
26.173 Tapped Delay Line . . . 355
26.174 TCP . . . 356
26.175 TDR . . . 357
26.176 Time Outside Level . . . 358
26.176.1 Inputs . . . 358
26.176.2 Parameters . . . 358
26.177 Thermal Diode . . . 359
26.178 Threshold . . . 360
26.178.1 Inputs . . . 360
26.178.2 Parameters . . . 360
26.178.3 Output Signal . . . 360
26.179 Top . . . 361
26.179.1 Inputs . . . 361
26.179.2 Parameters . . . 361
26.179.3 Output Signal . . . 361
26.180 Touchstone Export . . . 362
26.181 Touchstone Import . . . 363
26.182 Trend . . . 364

26.183 TRC Import . . . 365
26.184 UART . . . 366
26.185 Unwrapped Phase . . . 367
26.185.1 Inputs . . . 367
26.185.2 Parameters . . . 367
26.185.3 Output Signal . . . 367
26.186 USB 1.0 / 2.x Activity . . . 368
26.187 USB 1.0 / 2.x Packet . . . 369
26.188 USB 1.0 / 2.x PCS . . . 370
26.189 USB 1.0 / 2.x PMA . . . 371
26.190 Undershoot . . . 372
26.191 Upsample . . . 373
26.192 VCD Import . . . 374
26.193 Vector Frequency . . . 375
26.194 Vector Phase . . . 376
26.195 Vertical Bathtub . . . 377
26.196 VICP . . . 378
26.197 Waterfall . . . 379
26.198 WAV Import . . . 380
26.199 WFM Import . . . 381
26.200 Windowed Autocorrelation . . . 382
26.201 Window . . . 383
26.201.1 Inputs . . . 383
26.201.2 Parameters . . . 383
26.201.3 Output Signal . . . 383
26.202 X-Y Sweep . . . 384

# Chapter 1

# Introduction

## 1.1 Introduction

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

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

## 1.2 Documentation Conventions

Numbers are decimal unless explicitly specified otherwise. Binary or hexadecimal values use SystemVerilog notation, for example 'b10 means the binary value 10 (2) with no length specified, and 8'h41 means the 8-bit hexadecimal value 41 (decimal 65)

When referring to colors, HTML-style #RRGGBB or #RRGGBBAA notation is used. For example #ff0000 means pure red with unspecified alpha (assumed fully opaque) and #ff000080 means pure red with 50% opacity.

Printf-style format codes are used when describing output of protocol decodes. For example, "%02x" means data is formatted as hexadecimal bytes with leading zeroes.

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.

## 1.3 Key Concepts

### 1.3.1 User Interface

Most UI elements can be interacted with by left clicking (select), left dragging (move), using the scroll wheel (zoom), double clicking (open properties dialog), or right clicking (context menu).

Hovering the mouse over a main window UI element displays a series of icons in the status bar explaining how to interact with the element in question.

Hovering some UI elements, or a (?) help marker in a dialog box, displays a tooltip with additional information such as the full name of an object, metadata about a waveform, etc.

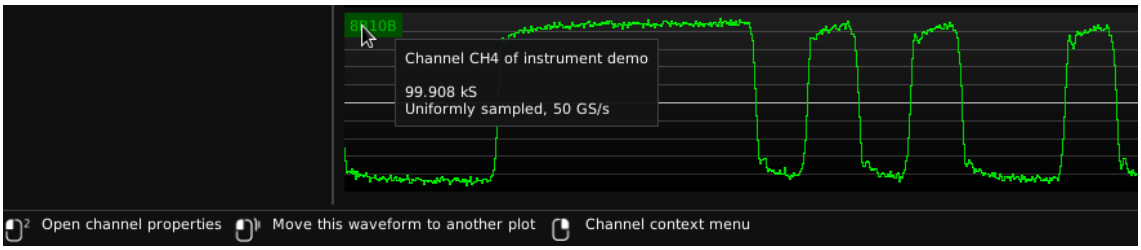



Figure 1.1: Tooltip and help message in status bar

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

### 1.3.2 Design Philosophy

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

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

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

Similarly, vertical scale for waveforms is defined in terms of full-scale range, a far more intuitive and useful metric than arbitrary "divisions". While horizontal grid lines are still displayed in waveform views for convenience, their number, spacing, and locations may change. Tall plots will have more scale divisions than short ones, and the divisions are always located at round numbers even if this requires the grid to not be centered in the plot (Fig. 1.2)

Rather than optimizing for a touch screen (as is common for benchtop oscilloscopes), ngscopeclient's UI is heavily mouse driven and context based. Space used by always-visible buttons, sliders, etc is kept to a minimum in order to keep as much screen real estate as possible usable for waveform

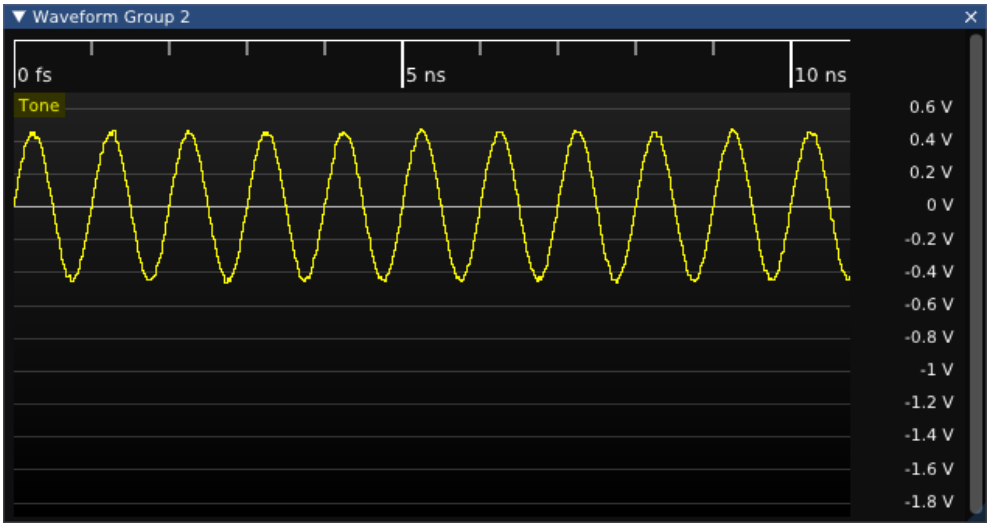



Figure 1.2: Example waveform showing off-center grid and round-numbered grid lines

display. Additional controls are displayed in menus or pop-up dialogs which can be closed, moved out of view, or docked as needed.

### 1.3.3 Terminology

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

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

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

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

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

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

[1] Or a simulated mock-up of one, such as the "demo" oscilloscope driver used for testing
[2] Zero channels is legal in the API, however such an instrument would be of little practical use!

providing function generator output capability.

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

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

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

## 1.4 Stability Notes

The v0.x series of ngscopeclient and libscopehal is under active development. There is no expectation of API or ABI stability for the backend libraries; method signatures and memory layouts may change at any time.

The .scopesession file format is intended to be strictly upward compatibile; any potential crashes or parsing errors when opening older datasets in a newer version are due to bugs and should be reported as such.

There is no expectation of downward compatibility (sessions from a newer ngscopeclient version are very likely to not work in older versions and will display a warning, although it will still attempt to load on a best-effort basis).

Note that filters may occasionally change port or parameter configurations during refactoring, which may require some manual reconfiguration of older sessions in order to work properly.

## 1.5 Revision History

- 2026-09-26: Version 0.3 release
- 2026-08-18: Version 0.2.2 release
- 2026-08-15: Version 0.2.1 release
- 2026-08-13: Version 0.2 release
- 2025-12-31: Version 0.1.1 release
- 2025-09-11: Version 0.1 release

# Chapter 2

# Legal Notices

## 2.1 Introduction

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

## 2.2 License Agreement

Copyright (c) 2012-2026 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 ngscopeclient's compatibility with said products. The reader should assume that these are all trademarks of their respective owners.

## 2.4 Third Party Licenses

- vkFFT (static, MIT license) - https://github.com/ngscopeclient/VkFFT/blob/master/LICENSE
- half (static, MIT license) - https://github.com/ngscopeclient/VkFFT/blob/master/half_lib/half.hpp
- canvas_ity (static, ISC license) - https://github.com/a-e-k/canvas_ity/blob/main/LICENSE.txt
- Dear ImGui (static, MIT license) - https://github.com/ngscopeclient/imgui/blob/docking/LICENSE.txt
- imgui_markdown (static, zlib license) - https://github.com/enkisoftware/imgui_markdown/blob/main/License.txt
- imgui-node-editor (static, MIT license) - https://github.com/ngscopeclient/imgui-node-editor/blob/master/LICENSE
- stb-image (static, MIT license) - https://github.com/ngscopeclient/imgui-node-editor/blob/master/external/stb_image/stb_image.h
- ImGuiFileDialog (static, MIT license) - https://github.com/aiekick/ImGuiFileDialog/blob/master/LICENSE
- Native File Dialog Extended (static, zlib license) - https://github.com/btzy/nativefiledialog-extended/blob/master/LICENSE
- dbus (shared, LGPLv2.1 license) - https://gitlab.gnome.org/GNOME/glib/-/blob/main/COPYING
- Vulkan-Hpp (static, Apache-2.0 license) - https://github.com/KhronosGroup/Vulkan-Hpp/blob/main/LICENSE.txt
- vulkan-headers (static, MIT license) - https://github.com/KhronosGroup/Vulkan-Headers/blob/main/LICENSE.md
- yaml-cpp (shared, MIT license) - https://github.com/jbeder/yaml-cpp/blob/master/LICENSE
- vulkan-loader (shared, Apache-2.0 license) - https://github.com/KhronosGroup/Vulkan-Loader?tab=License-1-ov-file#readme
- zlib (shared, zlib license) - https://zlib.net/zlib_license.html
- glfw (shared, zlib license) - https://www.glfw.org/license
- glslang (shared, BSD-3, BSD-2, MIT, and Apache-2.0 licenses) - https://github.com/KhronosGroup/glslang/blob/main/LICENSE.txt
- libsigc++ (shared, LGPLv3 license) - https://github.com/libsigcplusplus/libsigcplusplus/blob/master/COPYING

- libpng (shared, libpng license) - http://www.libpng.org/pub/png/src/libpng-LICENSE.txt
- liblxi (shared, BSD-3/EPICS license) - https://github.com/lxi-tools/liblxi/blob/master/LICENSE
- GTK 3.0 (shared, LGPLv2 license) - https://gitlab.gnome.org/GNOME/gtk/-/blob/main/COPYING
- hidapi (shared, BSD-3 license) - https://github.com/libusb/hidapi/blob/master/LICENSE.txt
- libtirpc (shared, BSD-3 license) - https://github.com/alisw/libtirpc/blob/master/COPYING
- OpenMP (shared, GPLv3 license with GCC Runtime Library Exception) - https://gcc.gnu.org/git/?p=gcc.git;a=blob;f=libgomp/libgomp.h

### 2.4.1 Optional, only used if scopehal-plugins-gpl plugin is installed

- linux-gpib (shared, GPLv2 license) - https://sourceforge.net/p/linux-gpib/git/ci/master/tree/linux-gpib-user/COPYING

### 2.4.2 Only in unit tests

- FFTW (shared, GPLv2-or-later) - https://www.fftw.org/doc/License-and-Copyright.html
- catch2 (static, Boost license) - https://github.com/catchorg/Catch2/blob/devel/LICENSE.txt

# Chapter 3

# Getting Started

## 3.1 Host System Requirements

### 3.1.1 Operating Systems

We test ngscopeclient on the following platforms:

- Linux: Arch, Debian 12 / 13, Fedora 43, Ubuntu 22.04 / 24.04
- MacOS: Sequoia
- Windows: 11

The majority of development is performed on Linux operating systems (primarily Debian) so this is the most well tested platform.

We do not test on Windows 10, however there are no known incompatibilities.

### 3.1.2 Processors

Any 64-bit Intel, AMD, or ARM processor (including Apple Silicon) should be able to run ngscopeclient.

If AVX2 and/or AVX512F support is present ngscopeclient will use special optimized versions of some signal processing functions, however neither instruction set is required.

We don't actively test on 32-bit platforms due to the significant RAM requirement of typical workloads, but we won't stop you from trying and would love to hear if you get it working.

### 3.1.3 Mouse

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.

### 3.1.4 GPU

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

- NVIDIA: Maxwell architecture (GeForce GTX 700 series and newer, February 2014)

- AMD: GCN based (Radeon HD 7000 and newer, January 2012)
- Intel: Iris Plus 540 or HD Graphics 520 (Skylake, August 2015)
- Apple: all Apple Silicon devices (M1 and newer). Newer Intel devices with Metal support should work but have not been tested.

On Linux, if you do not have a supported GPU you can still use ngscopeclient with software rendering via llvmpipe, however performance is likely to be poor.

Note that many virtual machine graphics stacks (e.g. VMWare) do not provide Vulkan unless a PCIe passthrough GPU is being used.

### 3.1.5 RAM

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

Typical RAM consumption examples:

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

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

The maximum supported waveform size depends on your Vulkan implementation but is typically $2^{32}$ bytes (4 GB) assuming sufficient video memory is available. This translates to one gigapoint analog or four gigapoints digital.

## 3.2 Instrument Support

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

## 3.3 Installation

### 3.3.1 Official Releases

Prebuilt binary packages are available for some of our supported platforms.

The latest released binaries can be downloaded from GitHub at https://github.com/ngscopeclient/scopehal-apps/releases.

### 3.3.2 Development Builds

If you are feeling adventurous and want to try bleeding-edge code, or are testing a fix at a developer's request, packages for the officially supported platforms are automatically built each commit by our CI pipeline. Unlike release packages, CI builds are not signed so you may encounter security warnings on some platforms when installing them.

Development binaries are available on our CI artifact server at https://dl1.ngscopeclient.org/ngscopeclient-ci/. Artifacts are organized in folders by year and month, then by date/time and commit hash of the specific build. Old builds are periodically deleted to recover disk space; the artifact server usually has a month or so worth of historical binaries depending on the pace of development.

## 3.4 Compilation

ngscopeclient can be compiled on Linux, macOS, and Windows. While the compilation process is generally similar, various steps differ among platform and distro.

### 3.4.1 Linux

1. Install dependencies.

#### Debian

Basic requirements:

```
sudo apt-get install build-essential git cmake pkgconf libgtk-3-dev libsigc++-2.0-\
dev libyaml-cpp-dev catch2 libglfw3-dev curl xz-utils libhidapi-dev lsb-release \
dpkg-dev file
```

On Debian bookworm and later, you can use system-provided Vulkan packages. Skip this if you choose to use the upstream Vulkan SDK instead:

```
sudo apt-get install libvulkan-dev glslang-dev glslang-tools spirv-tools glslc
```

To build the LXI component (needed if you have LXI- or VXI-11-based instruments):

```
sudo apt install liblxi-dev libtirpc-dev
```

To build the documentation, you will also need LaTeX packages:

```
sudo apt install texlive texlive-fonts-extra texlive-extra-utils
```

#### Ubuntu

Basic requirements:

```
sudo apt install build-essential git cmake pkgconf libgtk-3-dev libsigc++-2.0-dev \
libyaml-cpp-dev catch2 libglfw3-dev curl xz-utils libhidapi-dev lsb-release
```

On Ubuntu 22.10 and earlier (including 20.04 and 22.04), you will need to use the Vulkan SDK. Instructions for installing this are in a later step. On Ubuntu 23.04 and later, you can instead use system-provided Vulkan packages:

```
sudo apt-get install libvulkan-dev glslang-dev glslang-tools spirv-tools glslc
```

To build the LXI component (needed if you have LXI- or VXI-11-based instruments):

```
sudo apt install liblxi-dev libtirpc-dev
```

To build the documentation, you will also need LaTeX packages:

```
sudo apt install texlive texlive-fonts-extra texlive-extra-utils
```

#### Fedora

Basic requirements:

```
sudo dnf install git gcc g++ cmake make pkgconf gtk3-devel libsigc++30-devel yaml-\
cpp-devel catch-devel glfw-devel hidapi-devel lsb-release
```

System-provided Vulkan packages. Skip these if you choose to use the Vulkan SDK instead:

```
sudo dnf install vulkan-headers vulkan-loader-devel glslang-devel glslc libshaderc-\
devel spirv-tools-devel
```

To build the LXI component (needed if you have LXI- or VXI-11-based instruments):

```
sudo dnf install liblxi-devel libtirpc-devel
```

To build the documentation, you will also need LaTeX packages:

```
sudo dnf install texlive texlive-tex4ht texlive-dvipng texlive-make4ht texlive-\
luaxml texlive-makecell texlive-tocloft texlive-inconsolata texlive-gensymb texlive\
-newtx texlive-upquote
```

#### Alpine Linux

As Alpine Linux uses musl libc, you will need to use system-provided Vulkan packages, and not the Vulkan SDK.

```
apk add git gcc g++ cmake make pkgconf gtk+3.0-dev libsigc++-dev yaml-cpp-dev \
catch2-3 vulkan-loader-dev glslang-dev glslang-static glfw-dev shaderc-dev spirv-\
tools-dev libhidapi-dev lsb-release-minimal
```

If you are using an older stable release (such as CentOS 7), you may need to install some dependencies from source.

2. Install Vulkan SDK:

   In many cases, you can install the SDK components from distro-provided repositories, which is covered above. When possible, this is preferred over installing the Vulkan SDK. If you choose not to, or are running a Linux distro that does not provide these packages (for instance, Debian Bullseye, Ubuntu versions prior to 23.04, or other stable distros), the following instructions cover installing and loading the Vulkan SDK.

   The latest tested SDK at the time of documentation update is version 1.3.275.0. Newer SDKs are supported, but breaking changes sometimes take place. If you are using a newer SDK and run into problems, please file a bug report.

   Alternatively, to use the tarball packaged SDK, download and unpack the tarball. You can manually download the SDK, or do the following:

```
cd ~
mkdir VulkanSDK
cd VulkanSDK
curl -LO 'https://vulkan.lunarg.com/sdk/download/1.3.275.0/linux/vulkansdk-linux-\
x86_64-1.3.275.0.tar.xz'
tar xfv vulkansdk-linux-x86_64-1.3.275.0.tar.xz
```

And then source the ‘setup-env.sh‘ file:

```
source "$HOME/VulkanSDK/1.3.275.0/setup-env.sh"
```

When using the tarball-packaged SDK, you will need to source the ‘setup-env.sh‘ file any time you want to compile or run ngscopeclient. For convenience, you can add this to your ‘.bash_profile‘ or equivalent:

```
echo "source \"$HOME/VulkanSDK/1.3.275.0/setup-env.sh\"" >> ~/.bash_profile
```

3. Build scopehal and scopehal-apps:

```
cd ~
git clone --recursive https://github.com/ngscopeclient/scopehal-apps.git
cd scopehal-apps
mkdir build
cd build
cmake .. -DCMAKE_BUILD_TYPE=Release -DBUILD_TESTING=OFF
make -j4
```

### 3.4.2 macOS

1. Install dependencies.

   You will need Xcode (either from the App Store or the Apple developer site); after installing, run it once for it to install system components. This provides gcc, g++, make, and similar required packages.

   With Homebrew (brew.sh):

2. Basic requirements:

```
brew install pkg-config libsigc++ glfw cmake yaml-cpp catch2 libomp hidapi libpng
```

3. Vulkan SDK components (skip if using the Vulkan SDK):

```
brew install vulkan-headers vulkan-loader glslang shaderc spirv-tools molten-vk
```

4. Alternatively, install the Vulkan SDK:

   Download and install the Vulkan SDK.. The latest tested SDK at the time of documentation update is version 1.3.275.0. Newer SDKs are supported, but breaking changes sometimes take place. If you are using a newer SDK and run into problems, please file a bug report.

   And then source the ‘setup-env.sh‘ file:

```
source "$HOME/VulkanSDK/1.3.275.0/setup-env.sh"
```

When using the SDK, you will need to source the ‘setup-env.sh‘ file any time you want to compile or run ngscopeclient. For convenience, you can add this to your ‘.zprofile‘ or equivalent:

```
echo "source \"$HOME/VulkanSDK/1.3.275.0/setup-env.sh\"" >> ~/.zprofile
```

5. Build scopehal and scopehal-apps:

```
cd ~
git clone --recursive https://github.com/ngscopeclient/scopehal-apps.git
cd scopehal-apps
mkdir build
cd build
cmake .. -DCMAKE_BUILD_TYPE=Release -DCMAKE_PREFIX_PATH="$(brew --prefix);$(brew --\
prefix)/opt/libomp"
make -j4
```

### 3.4.3 Windows

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

#### Building from source

1. Download and install MSYS2. You can download it from msys2.org or github.com/msys2/msys2-installer/releases

   The following steps can be done in any MSYS-provided shell.

2. Install git and the toolchain:

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

3. Install general dependencies:

```
pacman -S mingw-w64-ucrt-x86_64-libsigc++ mingw-w64-ucrt-x86_64-yaml-cpp mingw-w64-\
ucrt-x86_64-glfw mingw-w64-ucrt-x86_64-catch mingw-w64-ucrt-x86_64-hidapi mingw-w64\
-ucrt-x86_64-libpng
```

4. Install Vulkan dependencies:

```
pacman -S mingw-w64-ucrt-x86_64-vulkan-headers mingw-w64-ucrt-x86_64-vulkan-loader \
mingw-w64-ucrt-x86_64-shaderc mingw-w64-ucrt-x86_64-glslang mingw-w64-ucrt-x86_64-\
spirv-tools
```

5. Install FFTW (required for a few unit tests):

```
pacman -S mingw-w64-ucrt-x86_64-fftw
```

6. Check out the code

```
cd ~
git clone --recursive https://github.com/ngscopeclient/scopehal-apps
```

   All following steps are to be done in a UCRT64 shell.

7. Build manually:

```
cd scopehal-apps
mkdir build
cd build
cmake ..
ninja -j4
```

8. Optional, to build MSI installer:

   Download and install WiX Toolset.
   You can download it from https://github.com/wixtoolset/wix3/releases
   If you install it to the path `"C:\Program Files (x86)\WiX Toolset v3.14"` run the following cmake command instead of `cmake ..` mentioned earlier:

```
cmake .. -DWIXPATH="C:\Program Files (x86)\WiX Toolset v3.14\bin"
```

   `ninja` compilation will now generate the installer after binaries.

9. Run scopehal and scopehal-apps:

   Building scopehal and scopehal-apps with MSYS2 will install required dependencies in MSYS2's libpath, so locally compiled builds of ngscopeclient have to be launched from a MSYS2 shell (use MSI installer to generate a standalone package including these libraries).

   The binaries can be found in the build directory, such as ngscopeclient in $HOME/scopehal-apps/build/src/ngscopeclient.

   Use the following commands to run ngscopeclient:

```
cd src/ngscopeclient/
./ngscopeclient.exe
```

   Or with some debug options:

```
./ngscopeclient.exe --debug --trace SCPISocketTransport
```

## 3.5 Running ngscopeclient

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

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

### 3.5.1 General arguments

- `--version`
  Print the application version and exit.
- `--help`, `-h`
  Print a brief description of the available command line arguments and exit.
- `--maximize`, `-m`
  Maximize ngscopeclient window on startup (overrides "Window startup mode" preference).
- `--restore`, `-r`
  Restore previous ngscopeclient window size and position (overrides "Window startup mode" preference).
- `Session files`
  If you wish to resume a prior session, pass the path to a session file saved from the graphical interface as the sole non-option argument.

  The file name must end in .scopesession.
- `Instrument connection strings`
  When starting a new session, you may provide one or more instrument connection strings as arguments, which will be added to the session.

  Connection strings are not accepted when resuming an existing session.

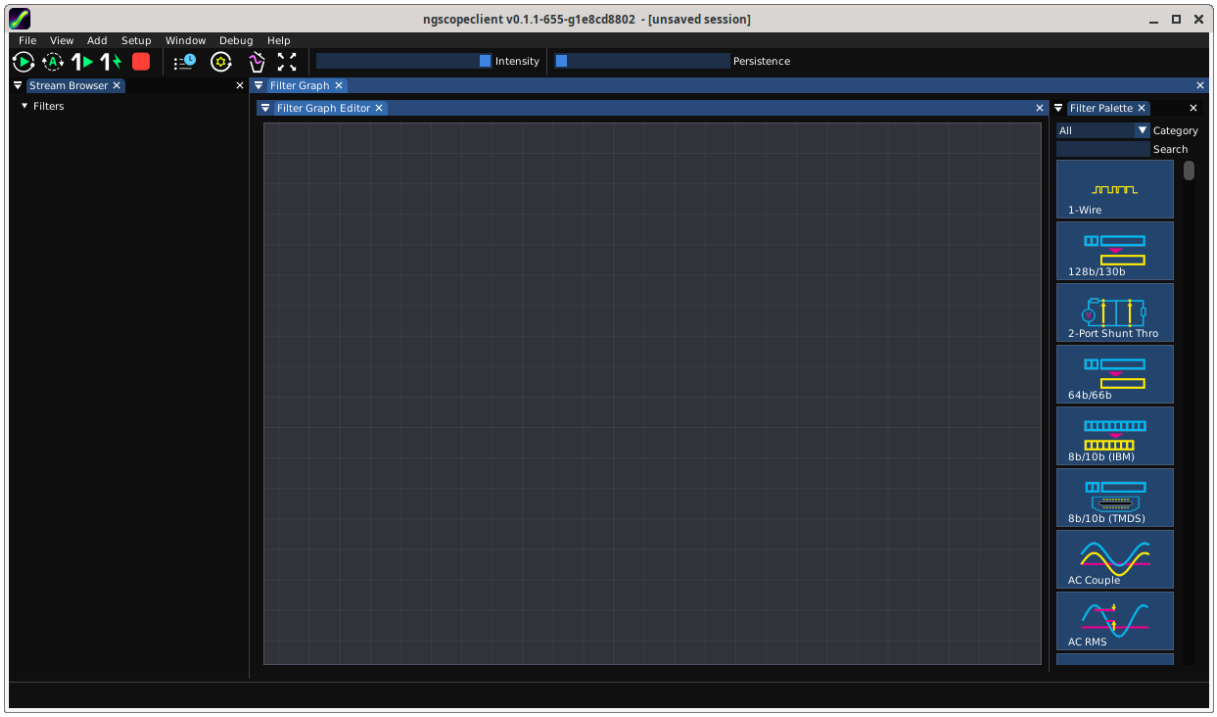



Figure 3.1: Empty ngscopeclient session

### 3.5.2 Console verbosity arguments

ngscopeclient takes standard liblogtools arguments for controlling console debug verbosity.

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

- `--debug`
  Sets the verbosity level to "debug" (5).
- `-l [file]`, `--logfile [file]`
  Writes a copy of all log messages to `file`. This is preferred over simply redirecting output with pipes, as console escape sequences are stripped from the file log output.
- `-L [file]`, `--logfile-lines [file]`
  Same as `--logfile` except line buffering is turned on.
- `-q`, `--quiet`
  Reduces the verbosity level by one. Can be specified more than once to lower verbosity by several steps.
- `--trace [class]`, `--trace [class::function]`
  Enables extra debug output from the class `class` or the function `class::function`. Has no effect unless `--debug` is also specified.
- `--stdout-only`
  Sends all logging output to stdout. By default, error (level 1) and warning (level 2) messages go to stderr.
- `--verbose`
  Sets the verbosity level to "verbose" (4).

# Chapter 4

# Tutorials

This section contains step-by-step examples to help you familiar yourself with the basics of using ngscopeclient.

Many of them can be done offline using the built-in demo oscilloscope or saved example waveform datasets - no lab access required!

## 4.1 The Basics

- **Lab requirements:** None, can be performed offline
- **Learning goals:** Navigating the ngscopeclient UI and performing common operations

### 4.1.1 Connecting to an Oscilloscope

We need to be connected to an instrument to do much of anything useful. Let's use the built-in demo scope for that.

1. Start with a new, empty ngscopeclient session
2. Select `Add / Oscilloscope / Connect...` from the top menu.
3. Select the "demo" driver and "null" transport. Leave the path blank, since the demo scope doesn't need any connection information.
4. Give the demo scope a nickname of your choice. This will be used to disambiguate scope channels and properties dialogs if you have more than one instrument connected, as well as to let you reconnect to the instrument quickly in the future.
5. Click the "add" button. You should be presented with an oscilloscope view showing four empty channels stacked on top of each other.




Figure 4.1: Connection dialog

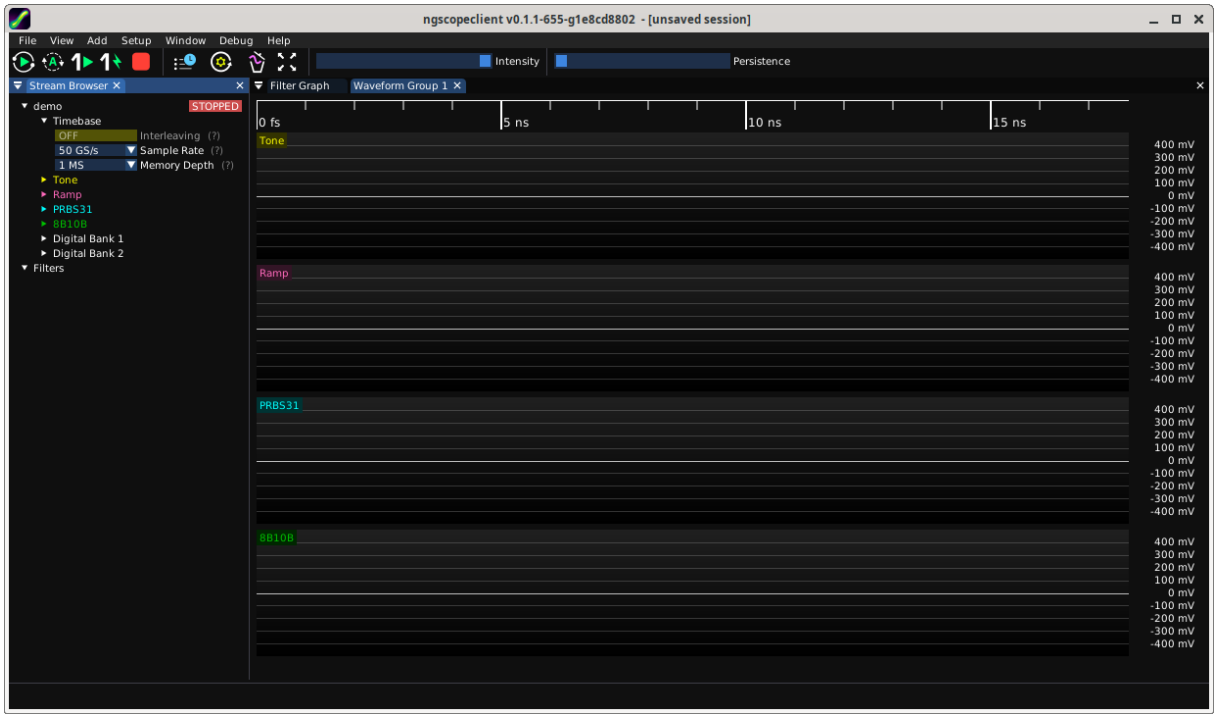



Figure 4.2: Application window after connecting to demo scope

### 4.1.2 Acquiring Waveforms

Looks pretty boring! Let's grab some waveforms so we have something to look at.

Note: The current demo scope is a simplistic instrument that doesn't implement realistic trigger semantics, so most of the usual trigger settings you might expect from real scopes (adjusting trigger level, horizontal position, selecting type of edge or condition) aren't available. It will always trigger immediately when armed and return waveforms at the same horizontal position.

1. Press the "single trigger" button (second from left on the toolbar). You should see waveforms appear in each channel.
2. Press the "normal trigger" button (leftmost on the toolbar). You should see the waveform display begin updating live.
3. Press the "stop" button (fourth from left on the toolbar) to stop acquiring waveforms.




Figure 4.3: Our first waveforms

### 4.1.3 Navigating the Y Axis

All of the waveforms in the demo scope are centered around zero volts and just the right amplitude to fill the view. But in real life we're usually not that lucky. Let's try moving one of the waveforms around.

1. Move the mouse over the Y axis at the right side of one of the plots.
2. Click and drag with the left button to move the waveform vertically (adjusting frontend offset).
3. Scroll with the mouse wheel to scale the waveform vertically (adjusting frontend gain).

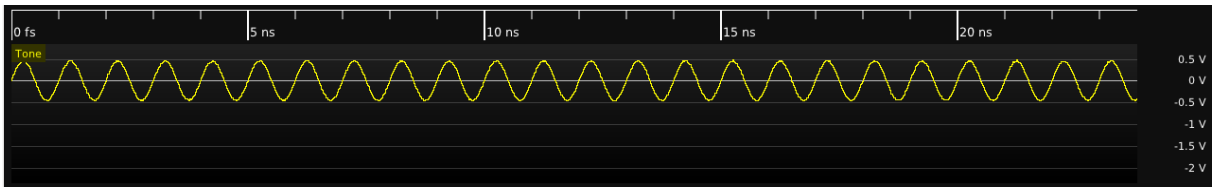



Figure 4.4: Demo channel after making some gain and offset tweaks

### 4.1.4 Navigating the X Axis

The timebase in ngscopeclient is decoupled from the viewport, so you can zoom and pan arbitrarily in the X axis without changing timebase settings.

Although we've only been looking at a 25*ns* wide window of the waveform so far, the default settings for the demo scope are 100K points at 50 Gsps (2$\mu s$ record length). Let's explore the rest of the waveform.

(a) Move the mouse over the main plot area and scroll with the mouse wheel to zoom in or out, centering at the mouse cursor position.

(b) Move the mouse over the timeline at the top of the viewport and drag with the left button to move the waveform side to side without changing zoom.

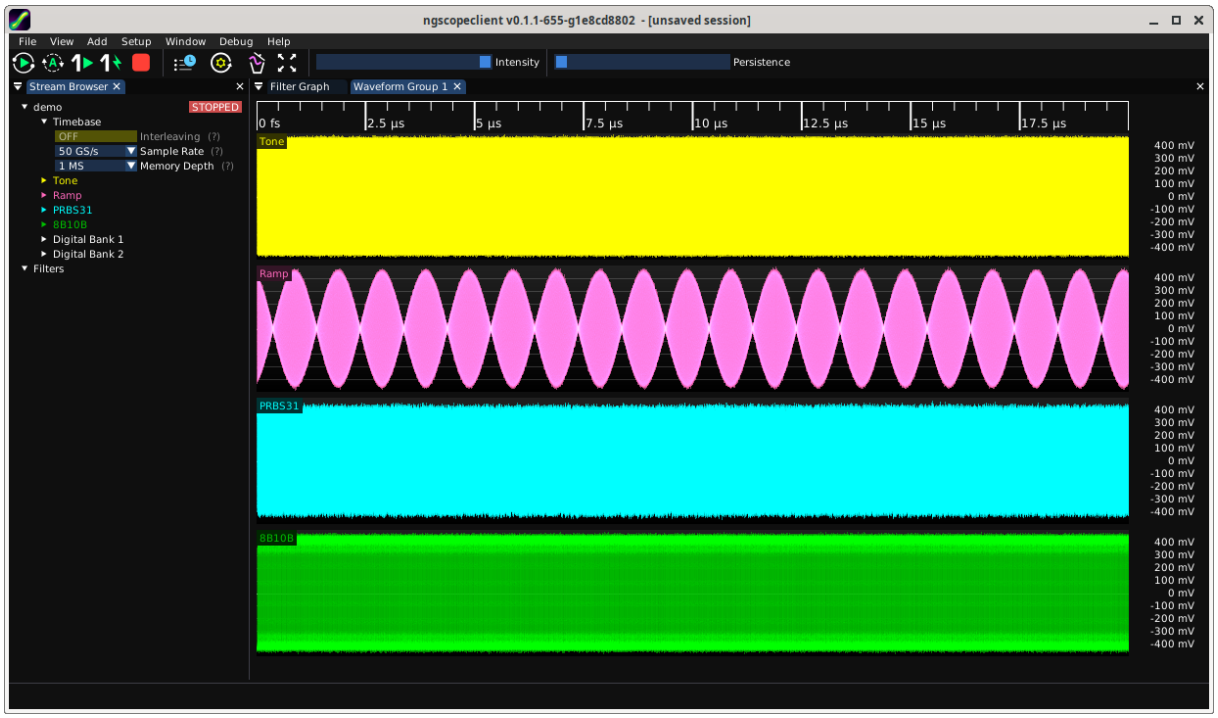



Figure 4.5: Demo session after zooming out to show entire waveform

# Chapter 5

# Main Window

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

## 5.1 Menu

### 5.1.1 File

This menu contains commands for saving and loading session files.

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

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

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

- `Exit`
  Exits the application

### 5.1.2 View

- `Fullscreen`
  Toggles full-screen mode

### 5.1.3 Add

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

- `BERT`
  Connect to a new, or recently used, bit error rate tester
- `Function Generator`
  Connect to a new, or recently used, function generator
- `Load`
  Connect to a new, or recently used, electronic load
- `Misc`
  Connect to a new, or recently used, miscellaneous instrument
- `Multimeter`
  Connect to a new, or recently used, multimeter
- `Oscilloscope`
  Connect to a new, or recently used, oscilloscope
- `Power Supply`
  Connect to a new, or recently used, power supply
- `RF Generator`
  Connect to a new, or recently used, RF signal generator
- `SDR`
  Connect to a new, or recently used, software-defined radio
- `Spectrometer`
  Connect to a new, or recently used, optical spectrometer
- `VNA`
  Connect to a new, or recently used, vector network analyzer
- `Channels`
  Displays a list of filters and instrument channels which can be opened in a new waveform view
- `Generate`
  Allows synthetic waveforms to be generated for testing, simulation, and channel design applications
- `Import`
  Allows waveforms to be loaded from external data files in various interchange formats

### 5.1.4 Setup

- `Manage Instruments...`
  Opens the Manage Instruments dialog, which allows control over synchronization, deskewing, and cross-triggering of multiple instruments.
- `Trigger...`
  Opens the Trigger dialog, allowing configuration of trigger settings.
- `Preferences...`
  Opens the Preferences dialog.

### 5.1.5 Window

This menu provides access to various utility windows.

- `Analyzer`
  Opens protocol analyzer dialogs for active protocol decodes
- `Power Supply`
  Opens the properties dialog for a currently connected power supply
- `Lab Notes`
  Opens the Lab Notes dialog, allowing you to take notes on your experiment.
- `Log Viewer`
  Opens the Log Viewer dialog,, allowing you to see debug log messages generated by the application. This is the same log stream which is normally written to stdout, but this dialog allows it to be accessed even when the application was not launched from a shell session and stdout is thus inaccessible.
- `Measurements`
  Opens the Measurements window, displaying scalar-valued measurements coming from instrument channels or filter blocks.
- `History`
  Opens the History dialog (see Chapter 10), which allows access to a rolling buffer of recently acquired waveforms.
- `Filter Graph`
  Reopens the filter graph editor if it had been closed.
- `Stream Browser`
  Reopens the stream browser if it had been closed.
- `Filter Palette`
  Reopens the filter palette if it had been closed.
- `New Workspace`
  Create a new workspace to organize other windows in.

### 5.1.6 Debug

Provides access to various features intended only for developers.

- `SCPI Console`
  Opens a console window allowing you to send raw SCPI commands to a currently connected instrument.

  This is a low level debug tool primarily intended for use by driver developers. The console is interlocked with background threads polling the instrument, so that replies to commands typed in the console will not be mixed with replies which the instrument driver is expecting to its own commands. However, commands sent in the console will bypass any caching in the driver and can easily lead to the driver and instrument firmware states becoming mutually inconsistent.

- `ImGui Demo`

  Launches a GUI library demo window allowing you to test out various widgets and features

- `Memory Leaker`

  Allows you to allocate a large block of memory (up to 4GB or the Vulkan allocation limit per dialog instance, but multiple dialogs may be spawned) in order to test video or system memory pressure.

- `Hardware Flags`
  Opens the Hardware Flags dialog, which allows you to override automatic detection of CPU and GPU feature flags. This is mostly intended to allow testing of fallback code (by disabling a code path that your hardware supports, pretending your CPU or GPU is less capable than it is in reality).

  NOTE: This is a low level interface with no validation. If you enable a feature that was not automatically enabled, ngscopeclient is very likely to crash with an illegal-instruction exception (GPU-side equivalent) at an unpredictable time in the future when a filter or driver attempts to use the unavailable functionality.

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

- `Memory Analysis`
  Opens the Memory Analysis window, which display a list of all Vulkan memory buffers allocated via the AcceleratorBuffer class (buffers internally allocated by third-party libraries such as Dear ImGui or vkFFT are not shown, but anything allocated within ngscopeclient / libscopehal / libscopeprotocols should be displayed), their data type, allocation size, and actual consumed size. Lines are color coded in yellow or red to highlight particularly inefficient usage of memory (capacity grossly exceeds size).

### 5.1.7 Help

- `Tutorial`
  Launches the tutorial wizard to help you learn the basics of the application

- `About`
  Displays program version and copyright information

## 5.2 Toolbar

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




Figure 5.1: ngscopeclient toolbar

### 5.2.1 Capture buttons

The capture button group (Fig. 5.2) contains five buttons. From left to right these are “arm normal trigger", “arm auto trigger", “arm one-shot trigger", “immediate/forced trigger", and “stop trigger".

Note that the “normal" and “auto" trigger mode still uses one-shot capture internally so that all waveform data can be downloaded before the next trigger event [1], so it is normal to see the scope rapidly switching back and forth between single-shot and stopped mode.



Figure 5.2: Capture control buttons

### 5.2.2 History

The history button (Fig. 5.3) toggles display of the waveform history view.



Figure 5.3: History button

### 5.2.3 Refresh Settings

In order to improve performance, ngscopeclient 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 ngscopeclient is running, ngscopeclient may not be aware of these changes.

The Refresh Settings button (Fig. 5.4) clears all cached instrument configuration and updates ngscopeclient with the current instrument settings. For most “headless" instruments, such as Pico Technology devices or the ThunderScope, this button has no effect.

[1] The repeating-trigger mode on almost all benchtop oscilloscopes is not interlocked with data transfer, meaning that attempting to download waveform data when the trigger is active will likely lead to corruption or desynchronization in which the data from one channel is from a later trigger event than a channel downloaded previously. Halting the trigger during waveform transfer avoids this issue, although refresh rate is reduced somewhat.



Figure 5.4: Refresh Settings button

### 5.2.4 Clear Sweeps

The Clear Sweeps button (Fig. 5.5) clears all persistence waveforms, accumulated eye pattern / waterfall data, and any other data that filter grap blocks persist across multiple waveforms. Waveform data, both history and the currently displayed waveforms, are not deleted.



Figure 5.5: Clear Sweeps button

### 5.2.5 Fullscreen

The Fullscreen button (Fig. 5.6) toggles ngscopeclient between normal and full-screen mode.



Figure 5.6: Fullscreen button

### 5.2.6 Intensity slider

The opacity slider (Fig. 5.7) controls the alpha/opacity used to display intensity-graded waveforms. At the leftmost position, waveforms are invisible. At the rightmost position, waveforms are fully opaque with no intensity grading other than that provided by antialiasing (Fig. 5.9). Experiment to find what setting gives the most useful visualization for your data (Fig. 5.8).




Figure 5.7: Trace opacity slider

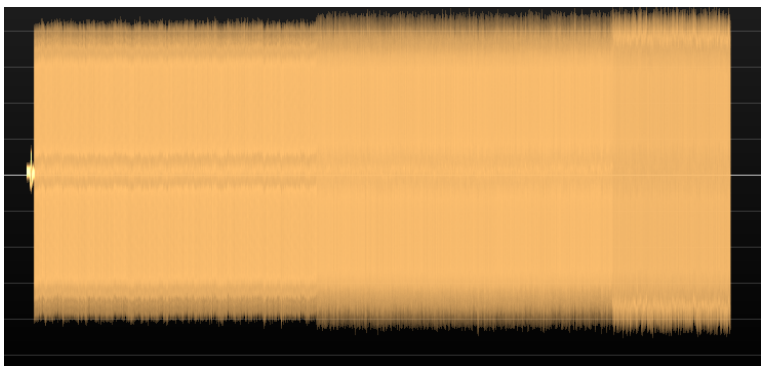


Figure 5.8: Intensity graded waveform

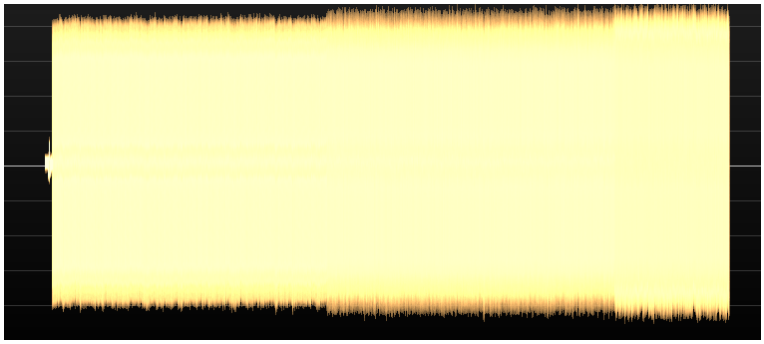


Figure 5.9: Same waveform with opacity turned up to 100%, losing fine details

### 5.2.7 Persistence slider

The persistence slider (Fig. 5.10) controls the decay rate for persistence-mode waveform display. At the far-left position (default) there is no persistence, and only the data from the most recent trigger event is visible. At the far right position, waveforms persist forever (until the clear-sweeps button is clicked or the display is scrolled/zoomed).

Persistence is applied to all analog and digital waveforms by default, but can be turned off for any individual waveform by right clicking the channel name in the plot and unchecking the “persistence" menu item. Digital buses and protocol decodes do not support persistence, and will always only show the most recent waveform state.




Figure 5.10: Waveform persistence slider

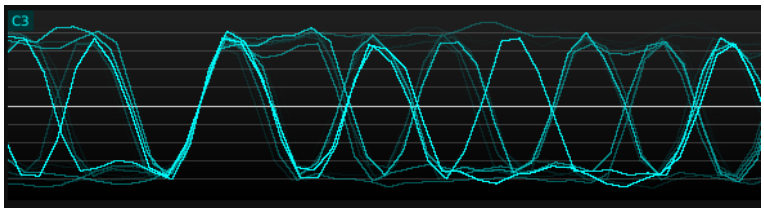



Figure 5.11: Example waveform display with persistence active

# Chapter 6

# Dialogs

## 6.1 Lab Notes

The Lab Notes dialog allows you to take notes on your experimental setup. It contains two tabs: “setup notes" and "general notes".

The contents of the Setup Notes tab are displayed on the Speed Bump dialog when loading a session file. The General Notes are only displayed within the Lab Notes dialog and are intended purely as a place for recording interesting observations made during the experiment.

Minimal Markdown syntax (headings, bullets, and bold font) is currently supported.[1]

Lab notes are saved as Markdown files in the data directory for the session and can be opened in any text editor or Markdown viewer. Note that they are overwritten each time the session is saved, so you should not modify them using an external tool while the session is open in ngscopeclient or your changes may be lost.

[1] Images and links are supported by the Markdown renderer library but the integration to properly use them is not yet finished; tables are not supported but this will likely be added in the future.

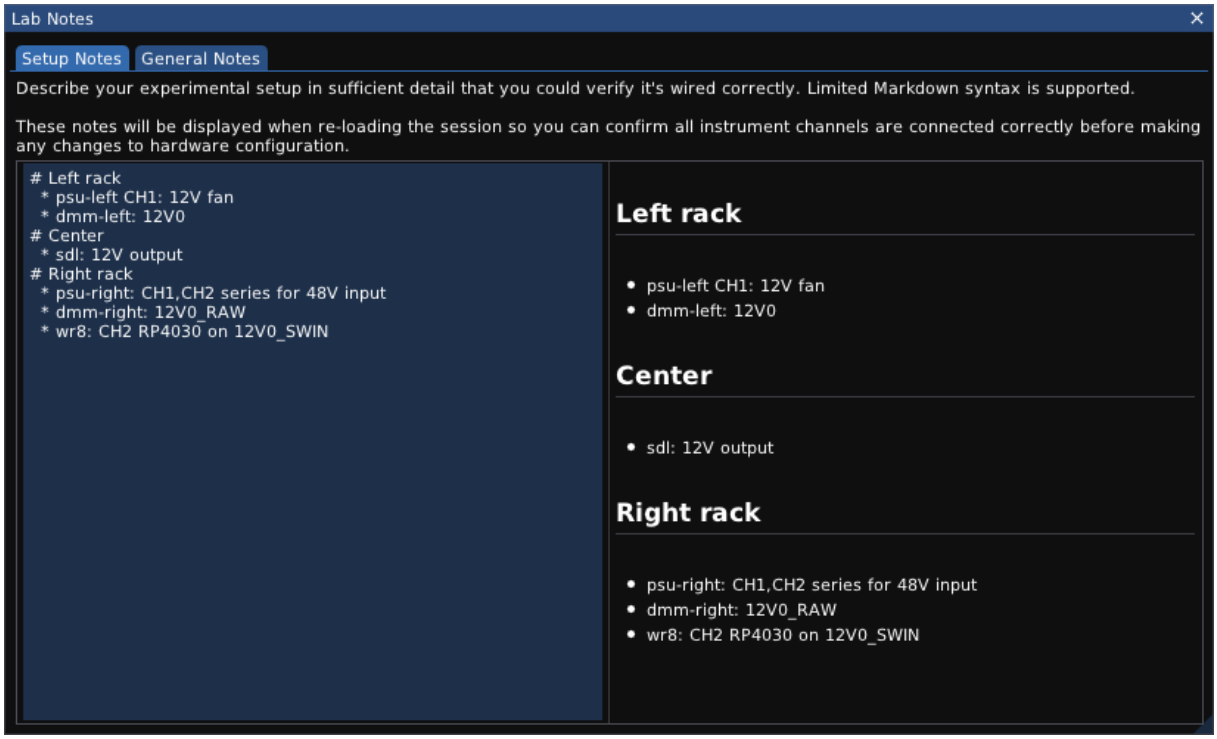



Figure 6.1: Lab notes dialog

## 6.2 Log Viewer

The Log Viewer dialog provides an alternate way to view log messages sent to stdout / stderr, which may be useful for debugging if the application was launched from a desktop icon or similar and there is no access to the console.

It can be opened from the `Window | Log Viewer` menu.

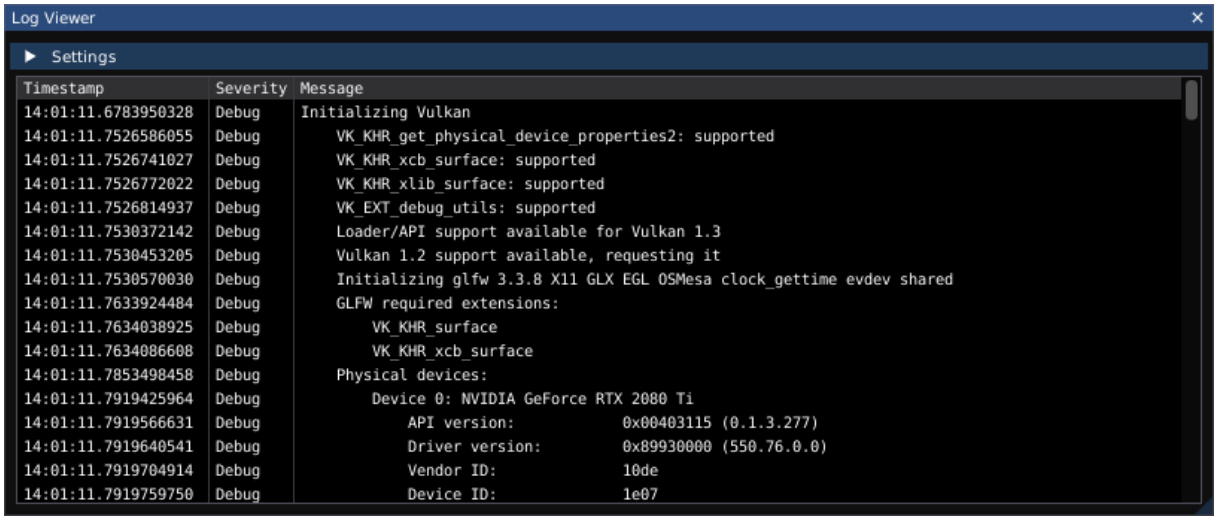



Figure 6.2: Log Viewer dialog

Expanding the “settings" panel allows log messages to be filtered to show more or less detail.

Additionally, if “trace" verbosity is selected, you can type class and/or function names using the

`classname`, `classname::function`, or `::globalfunction` syntax in order to enable high-verbosity messages printed via `LogTrace()`.

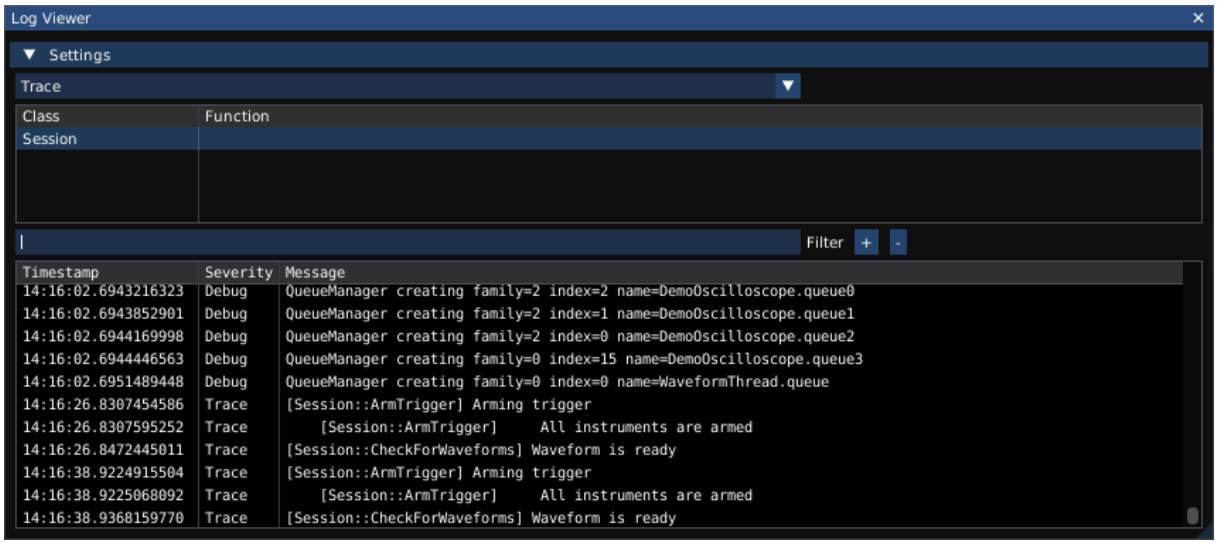



Figure 6.3: Log Viewer dialog with tracing active

## 6.3 Measurements

The Measurements dialog displays scalar values, either produced by filter graph blocks or directly measured by e.g. a multimeter channel.

It can be opened from the `Window | Measurements` menu.

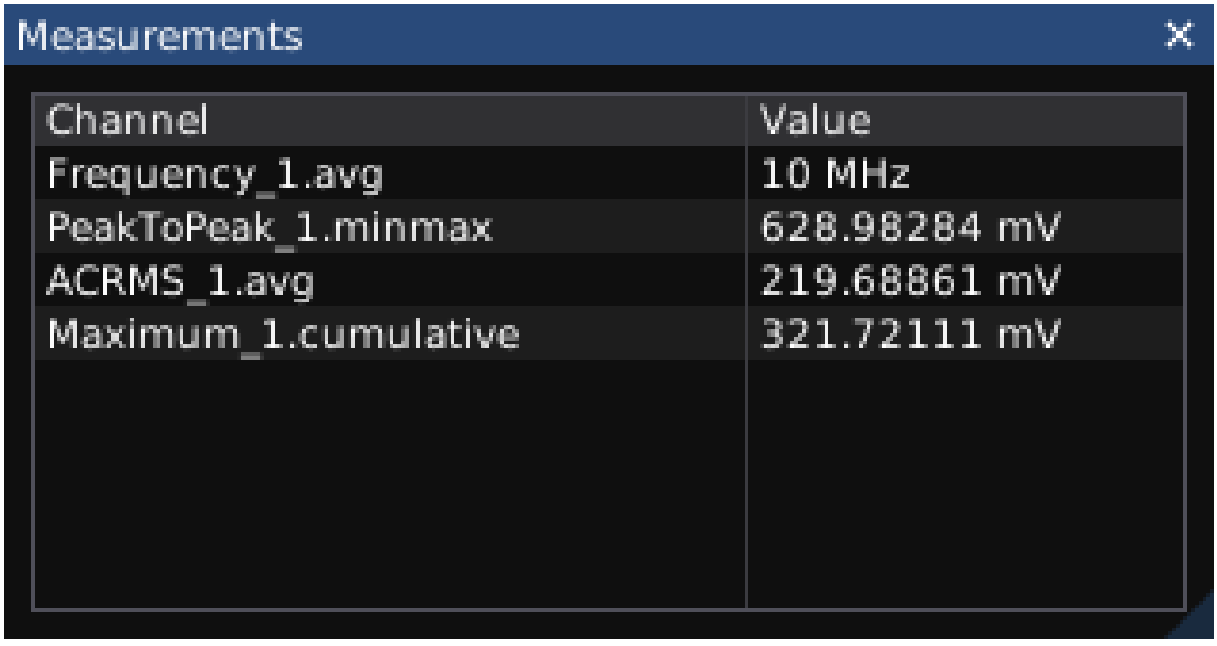



Figure 6.4: Measurements dialog

To remove a measurement, right click the channel name and select “delete" from the context menu.

To add a measurement, drag the stream from the stream browser to a row in the measurements dialog. Measurements can also be reordered by dragging and dropping.

Digital vector values can be displayed as decimal, hex, or binary. To change base, right click the channel name and select the desired format from the context menu.

## 6.4 Performance Metrics

The Performance Metrics dialog displays statistics on performance of rendering, waveform acquisition, and signal processing. The exact set of statistics displayed can vary depending on available Vulkan extensions, discrete vs integrated graphics, etc. This data is primarily intended for developers comparing before/after performance of optimizations and code changes.

It can be opened from the `Debug | Performance Metrics` menu.

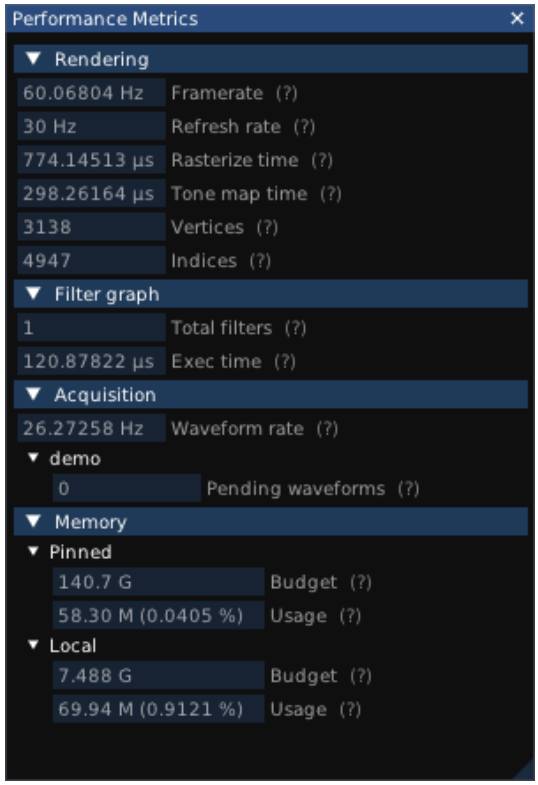



Figure 6.5: Performance Metrics dialog

### 6.4.1 Rendering

Displays render loop framerate, monitor refresh rate, total time spent last frame in the rasterization and tone mapping shaders, and the number of vertices and indices drawn as Vulkan geometry. Note that waveforms are drawn by a compute shader and do not contribute towards the vertex/index totals, other than a single textured rectangle used for displaying the shader output.

### 6.4.2 Filter graph

Number of filter blocks in the current graph, and run time for the most recent evaluation of the filter graph.

### 6.4.3 Acquisition

Displays the acquisition rate, in waveforms per second, for the entire session as well as for each instrument within the session if multiple trigger groups are active.

Additionally, this section displays the number of pending waveforms for each instrument (waveforms which have been acquired but not yet passed to the filter graph). This number should normally be flickering between zero and one if acquisition is active and zero otherwise; larger values indicate that the instrument is supplying data faster than ngscopeclient can process it.

### 6.4.4 Memory

Displays the total amount of available pinned memory (CPU-side memory eligible to be shared with the GPU) and local memory (memory attached to the GPU), as well as the amount of each currently in use.

## 6.5 Preferences

The Preferences dialog allows you to configure various application settings which are not specific to a particular experimental setup. It can be found under the `Setup | Preferences` menu.

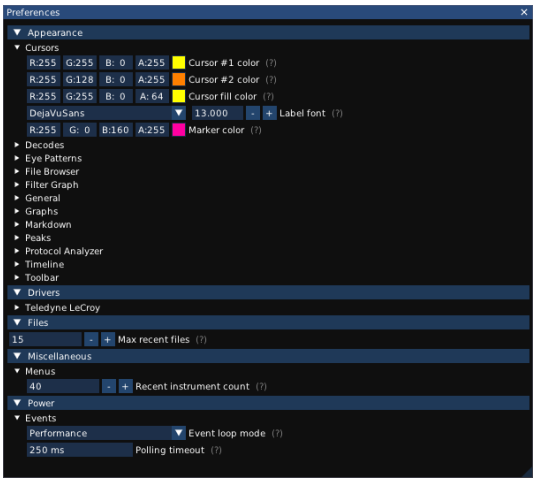



Figure 6.6: Preferences dialog

### 6.5.1 Appearance

This section allows you to configure fonts, colors, and other display settings for the application.

#### Stream Browser

**Numeric value display**

Numeric value display in Stream Browser can take one of the following values:

- *Console font*: use the font defined for console.
- *7 segment*: use 7 segment style display.

- *Default font*: use the application default font.

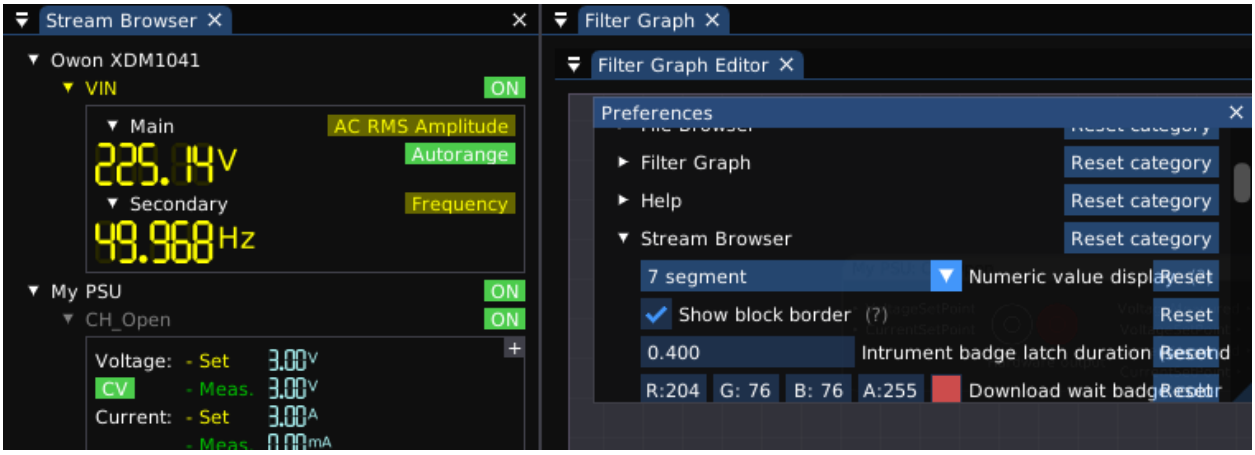



Figure 6.7: 7 Segment

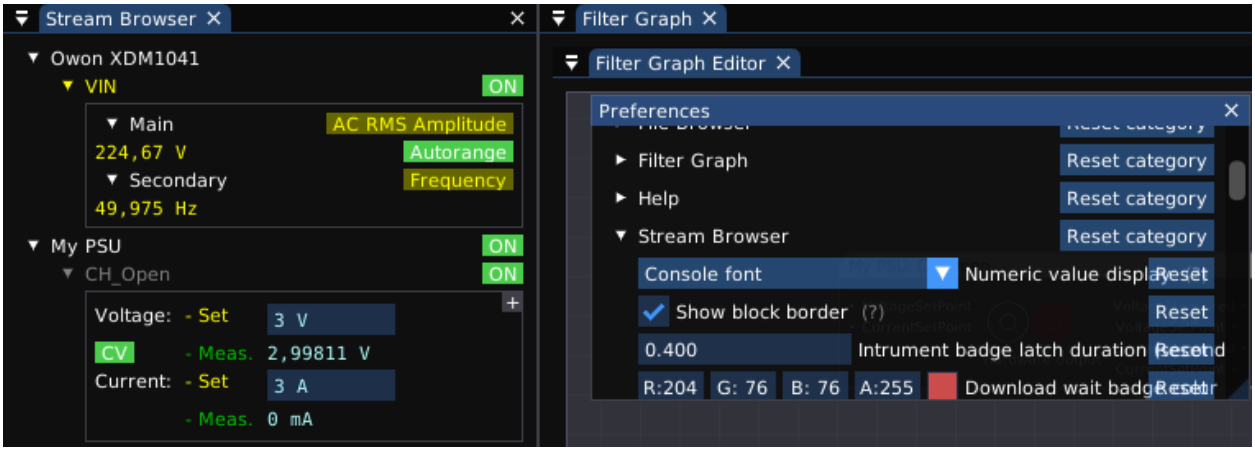



Figure 6.8: Console font

**Show block border**

When ticked, adds a visual border around stream browser blocks (e.g. channel properties)

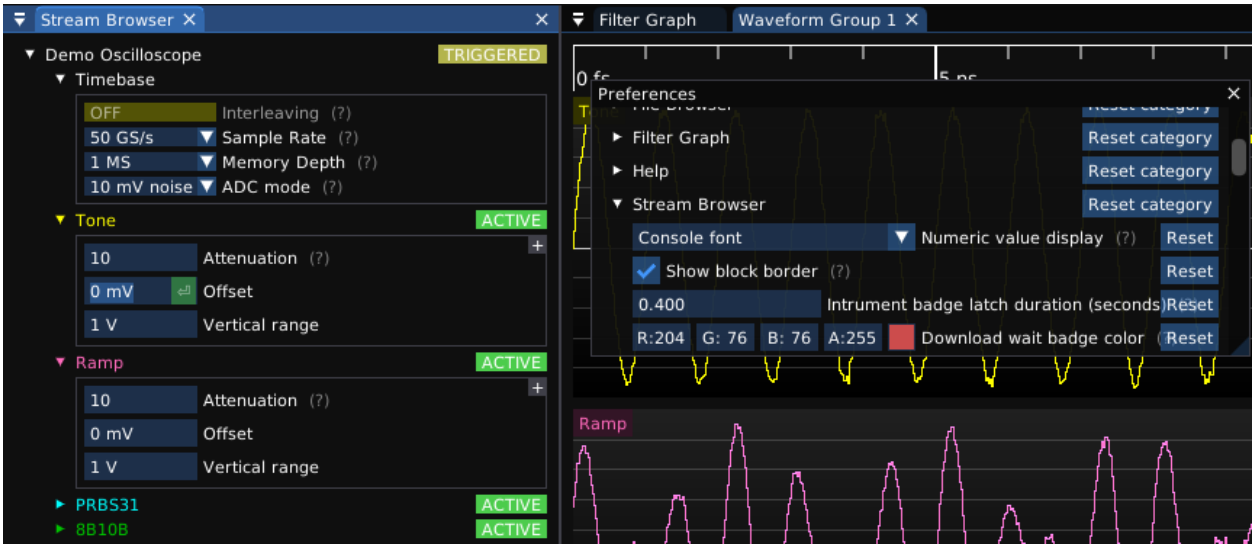



Figure 6.9: With block border

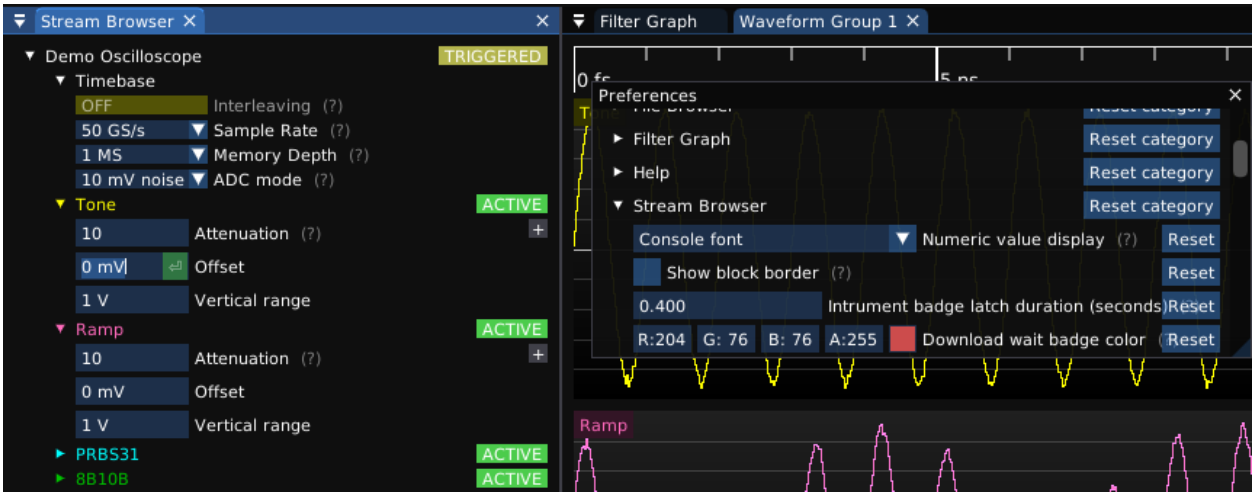



Figure 6.10: Without block border

### Windowing

#### Window startup mode

Window startup mode can be set to one of the following values:

- *Windowed*: the application is started in a fixed 1280x720 window.
- *Maximized*: the window is maximized on the main screen.
- *Last State*: the window is restored at the last position and size.

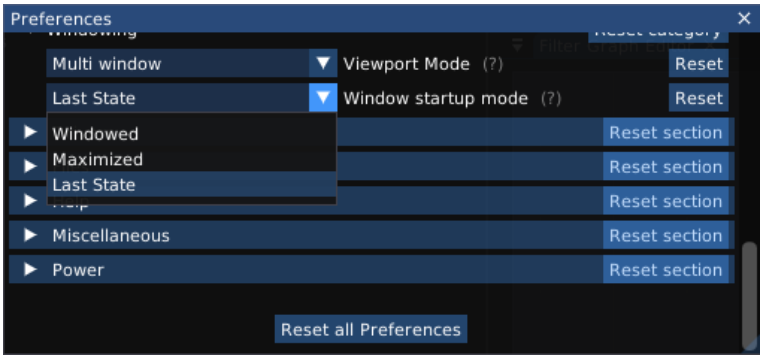



Figure 6.11: Startup configuration

### 6.5.2 Drivers

This section allows you to configure default configurations for various instrument drivers.

#### General

- *Headless scope default state*: When connecting to a headless oscilloscope (one without a front panel display), specify the set of channels which should be shown after connecting.

#### Rigol DHO

- *Data Width* (default auto): Specifies whether to use 8 bit or 16 bit transfer format when downloading samples from the instrument, or automatically decide based on hardware configuration.

#### Siglent SDS HD

- *Data Width* (default auto): Specifies whether to use 8 bit or 16 bit transfer format when downloading samples from the instrument, or automatically decide based on hardware configuration.

#### Teledyne LeCroy

- *Force 16 bit mode* (default on): Always use 16-bit format for downloading data from the instrument, even if it only has an 8-bit ADC. This doubles the amount of network bandwidth required and may reduce waveforms-per-second performance, but provides smoother waveforms.

  Note that Teledyne LeCroy scopes perform DSP flatness corrections in firmware between the ADC and the data presented via the API, meaning that even if the hardware only has an 8-bit ADC (non HD models), downloading data with 16-bit precision will result in improved waveform quality.

### 6.5.3 Files

- *Max recent files*: Specify the number of files to display under the `File | Recent Files` menu.

### 6.5.4 Miscellaneous

**Menus**

- *Recent instrument count*: Specify the number of recently used instruments to remember

### 6.5.5 Power

**Events**

This section provides settings allowing power vs performance tradeoffs. The default settings are appropriate for a desktop or laptop running on AC power; if running on a laptop with battery power you may wish to tune these to extend battery lifespan.

- *Event loop mode*: Controls the operating mode for the main application event loop.
  - In Performance mode, run at the screen refresh rate. This allows for the highest possible waveform processing rate and the smoothest interactivity, but may waste energy if you are spending a lot of time looking at the screen without actively acquiring or processing waveforms.
  - In Power mode, run at a greatly reduced frequency (default 4 Hz but configurable by the Polling Timeout setting) unless a redraw is triggered by mouse movement or keyboard input. This will limit the rate of waveform acquisition and lead to a slightly jerkier user interface, but saves power.
- *Polling timeout*: If the event loop is in Power mode, specifies the timeout before the event loop will run if there is no user input.

## 6.6 Speed Bump

The Speed Bump dialog is displayed when loading a session file online (reconnecting to and reconfiguring hardware), prior to committing changes to the instrument, if:

- The session file contains any user-created notes on the lab setup, or
- Any of the instrument settings in the session file do not match the current configuration of the corresponding instrument, and the direction of the change has potential to cause damage to the instrument or DUT (increasing output voltage, removing input attenuation, etc).

This is intended as a safeguard to prevent damaging hardware by accidentally loading the wrong session file. It also provides an opportunity to confirm that you have re-created the original experimental setup exactly if you are switching a lab bench between multiple projects and using saved sessions to restore instrument state.

Pressing the Abort button cancels loading of the session without applying any of the potentially dangerous changes. The instruments may be partially reconfigured in this state, as some changes (such as sample rate or memory depth configuration) are always safe to make and thus may have been applied prior to the warning being displayed.

Pressing the Proceed button allows ngscopeclient to proceed with loading the session and reconfiguring hardware. You must check the “I have reviewed the instrument configuration" box in order to enable the Proceed button.

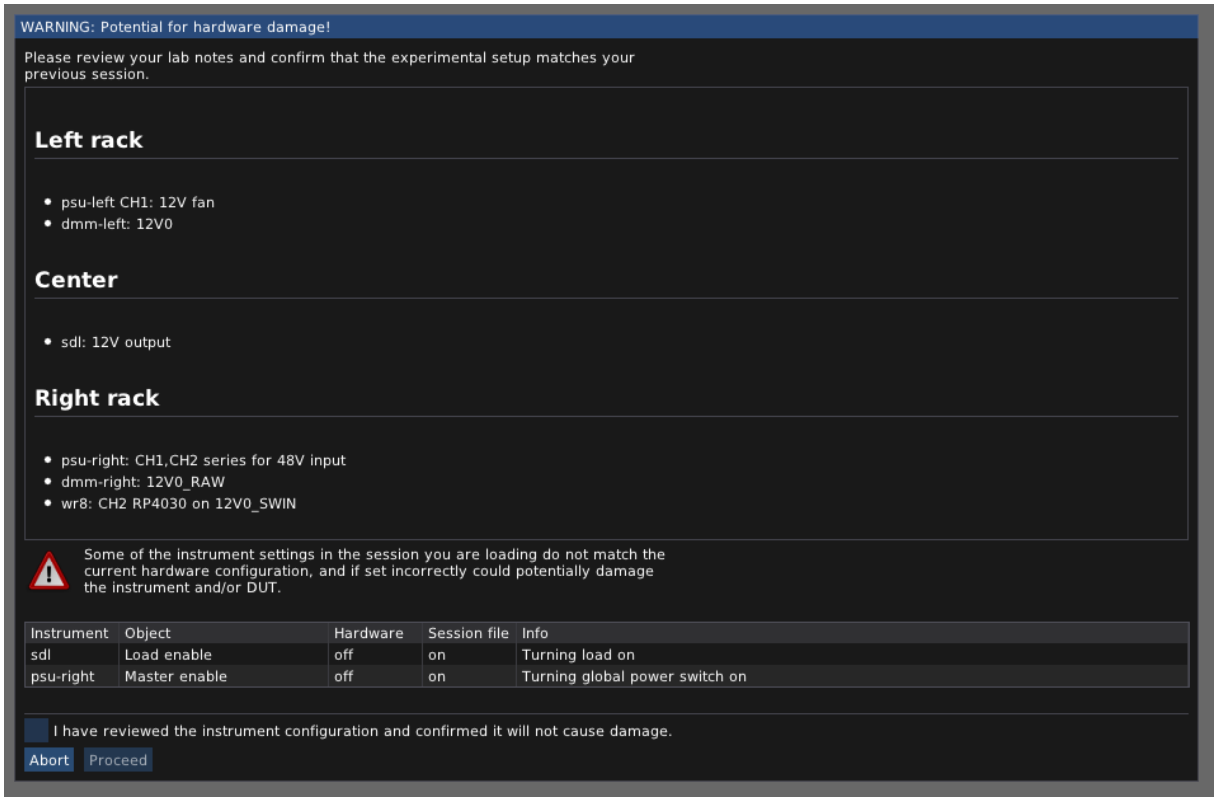



Figure 6.12: Speed Bump dialog

# Chapter 7

# Workspaces and Window Management

## 7.1 Window Management and Docking

All dialog boxes, waveform groups, and other GUI elements in ngscopeclient may be used docked or free-floating as needed.

To dock a window, drag the title bar (if floating) or tab title (if docked) to the desired location (Fig. 7.1).




Figure 7.1: Docking a floating window

On MacOS, Linux X11, and Windows you can drag dialogs or waveform areas out of the main ngscopeclient window to create multiple top-level windows. This can be useful for complex experimental setups or on multi-monitor workstations.

NOTE: Multi-window mode is not currently available on Linux Wayland due to GUI toolkit limitations however we hope to support this in the future. You can still dock and rearrange dialogs within the top-level application window as you see fit.

## 7.2 Workspaces

To create more complex windowing layouts, you may find it helpful to create *workspaces* via the `Window | New Workspace` menu.

A workspace is a window which has no function of its own, and simply serves as a container for docking other windows into. The workspace can itself be docked into another workspace or the main application window, allowing creation of complex multi-window or multi-tab layouts to suit your experimental needs (Fig. 7.2, 7.3, 7.4).

In the default ngscopeclient window layout, for example, the "Filter Graph" tab is a workspace which contains both the filter graph editor and the filter palette.

To rename a workspace, right click on the window title (if floating) or tab title (if docked) and enter the desired name.

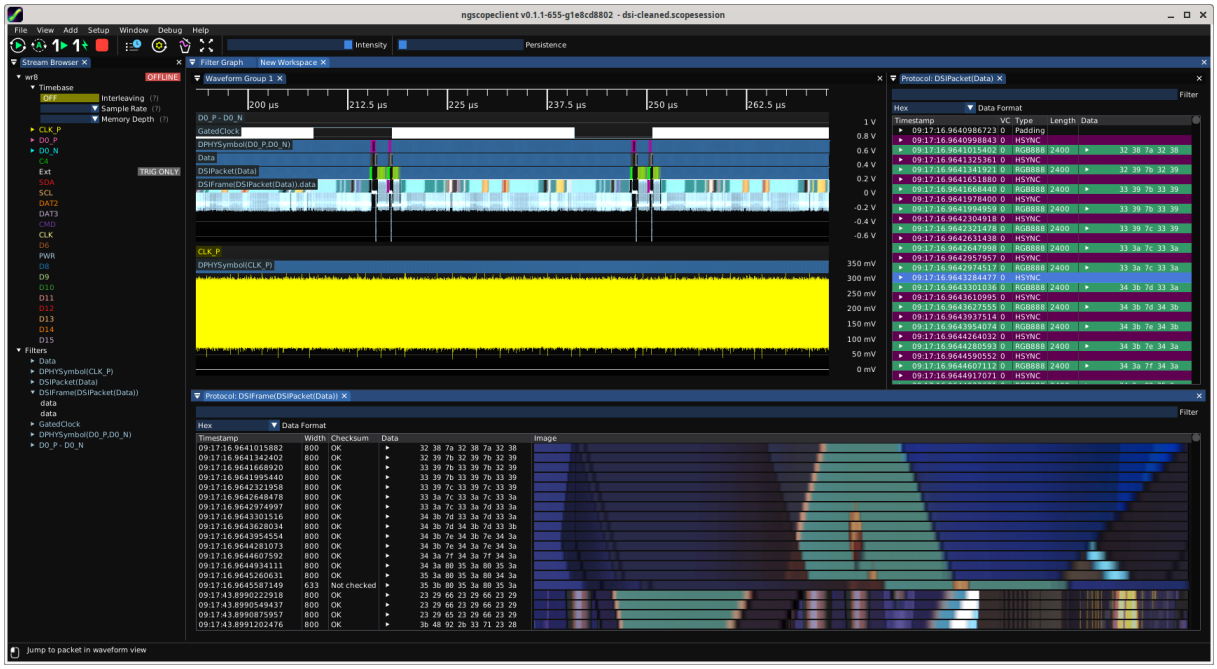



Figure 7.2: A docked workspace containing a waveform group and two protocol analyzer tabs

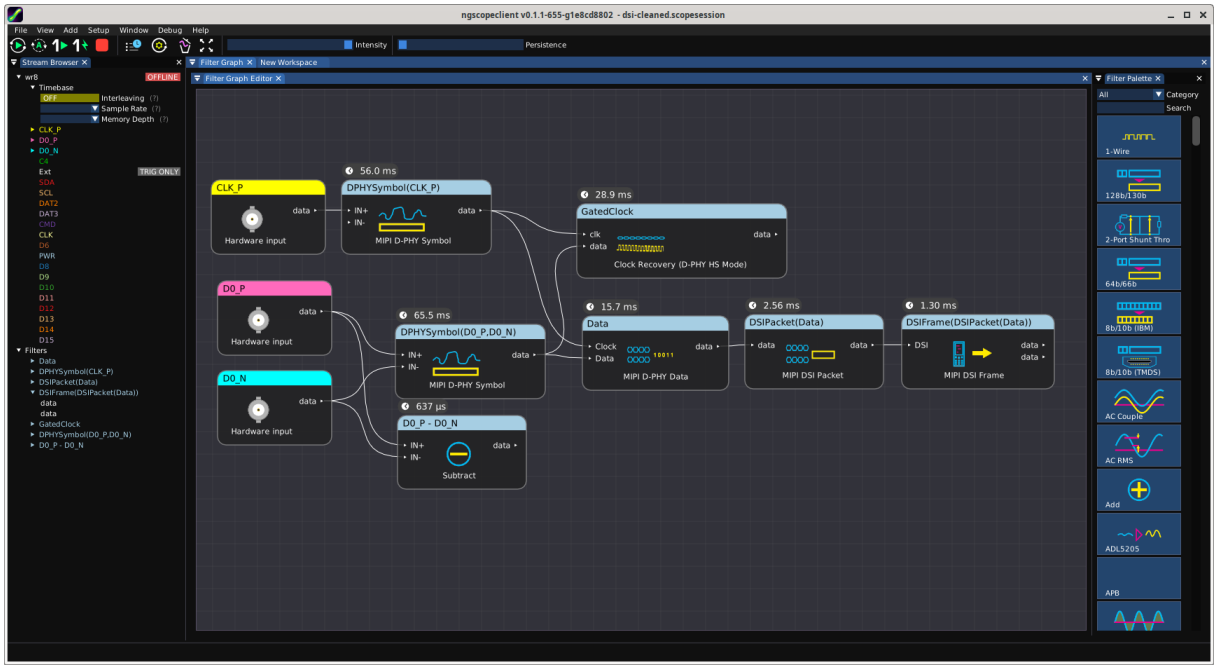



Figure 7.3: Switching from the waveform workspace to the filter graph editor workspace

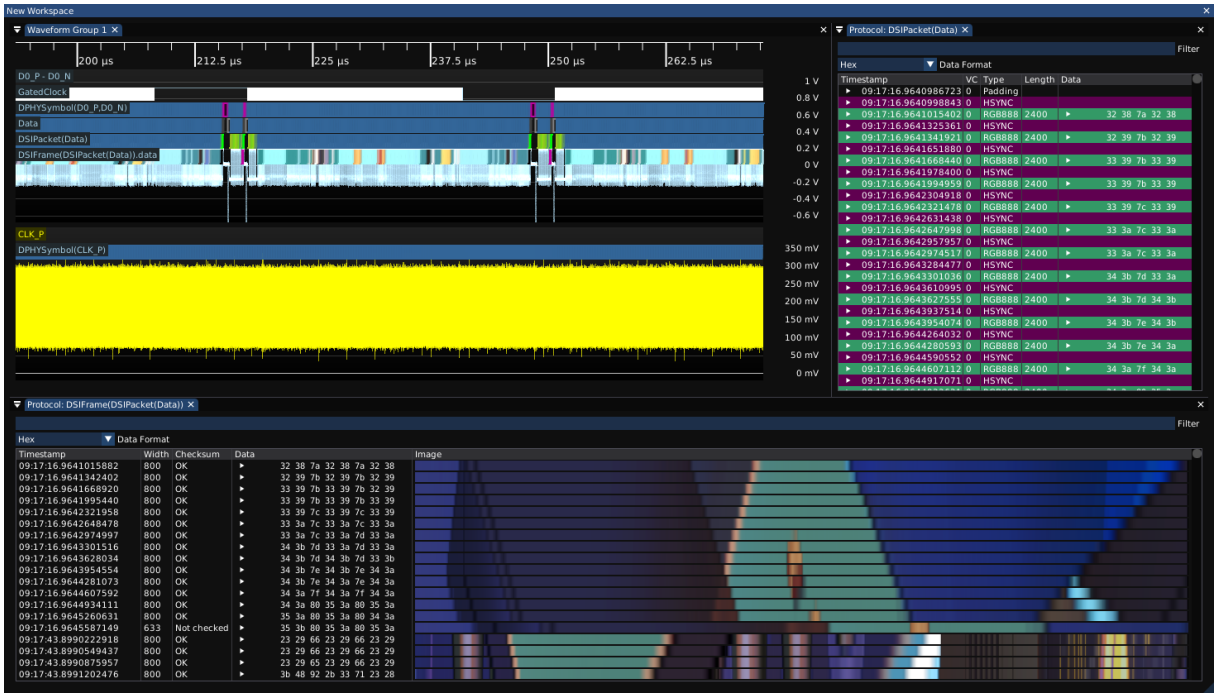



Figure 7.4: Free-floating workspace not docked to the main application window

# Chapter 8

# Waveform Groups

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

When a new oscilloscope is added to an empty ngscopeclient session, all enabled channels on the attached instrument(s) are displayed in a single waveform group (Figure 8.1). If no channels are enabled at connection time, the first channel will be enabled and displayed.

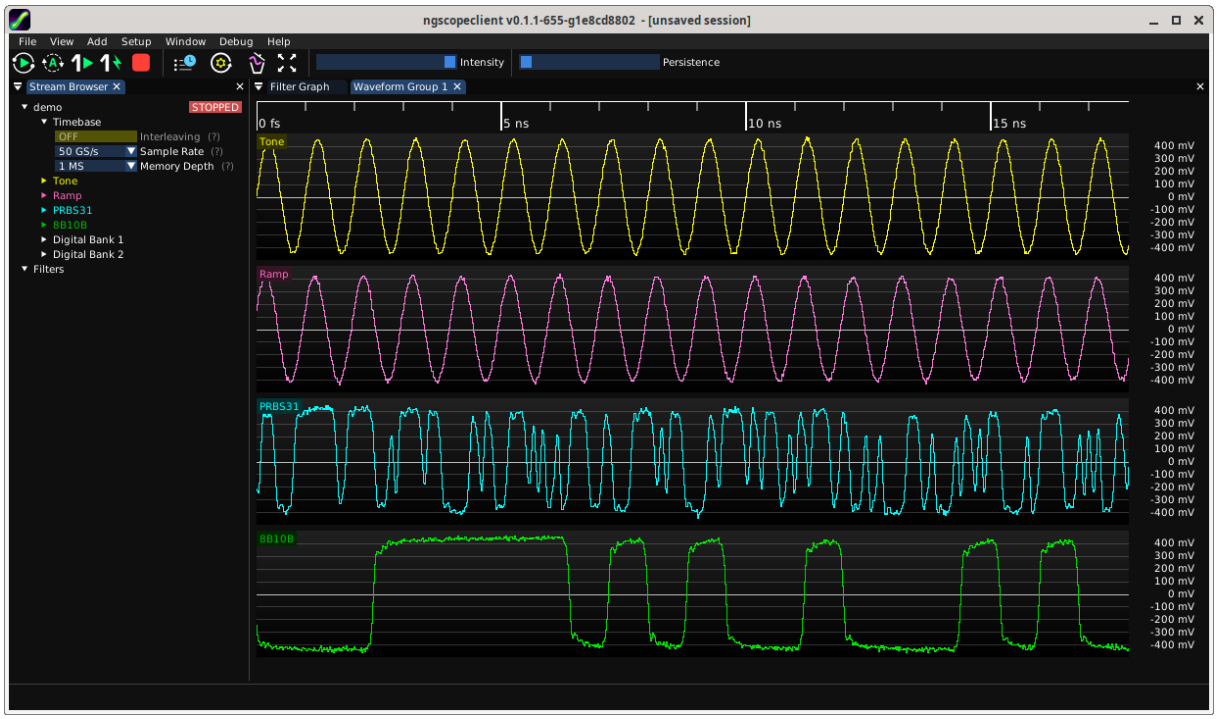



Figure 8.1: Top level ngscopeclient window with a single waveform group

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

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

## 8.1 Managing Groups

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

A new group may also be created at any time by clicking on a channel name and dragging it to the top, bottom, left, or right edge of an existing group. An overlay (Fig. 8.2) will be displayed showing the resulting split. For example, dropping the channel on the right side of the window produces the layout shown in Fig. 8.3.




Figure 8.2: Overlays showing drag-and-drop locations for splitting waveform groups

Waveform groups may be resized arbitrarily by dragging the separator between them. The title bar of a group may also be dragged, allowing the entire group to be undocked as a floating window. Floating windows can be re-docked by dragging the title bar back into the main ngscopeclient window (or another floating window), creating new tabs or splitting existing groups as desired (Fig. 8.4).




Figure 8.3: Result of dropping a waveform to the right side split area




Figure 8.4: Example of complex window layout with multiple tabs, splitters between docked waveform groups, and a group in a floating window

# Chapter 9

# Waveform Views

A waveform view is a single 2D plot area within a waveform group.

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

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

## 9.1 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.

## 9.2 Plot Area

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

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

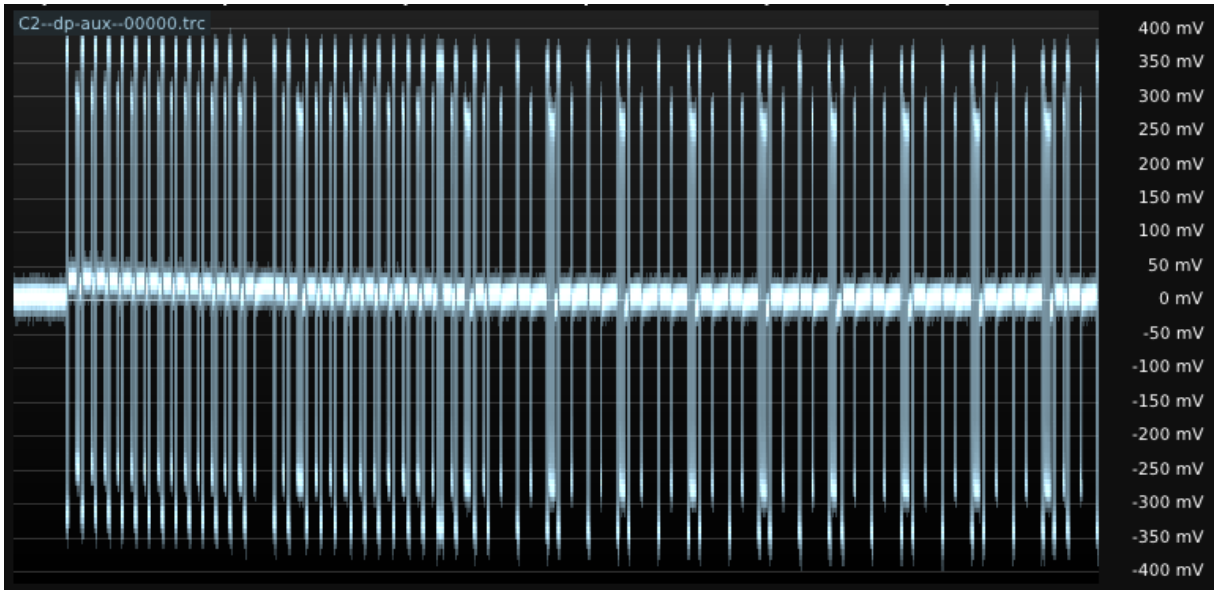



Figure 9.1: Intensity-graded waveform

## 9.3 Y Axis Scale

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

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

If a left-pointing arrow (as seen in Fig. 9.2) is visible, one of the channels in the view is selected as a trigger source. Click on the arrow and drag up or down to select the trigger level. Some trigger types, such as window triggers, have two arrows for upper and lower levels.

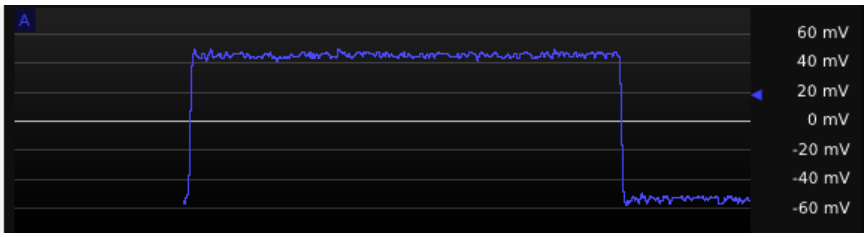



Figure 9.2: Waveform view showing trigger arrow on Y axis

## 9.4 Channel Label

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

Mousing over the channel name displays a tooltip (Fig. 9.3) with some helpful information about the waveform. The exact information displayed in the tooltip depends on the type of data being displayed, for example analog waveforms display sample rate and record length while eye patterns display the number of integrated UIs.

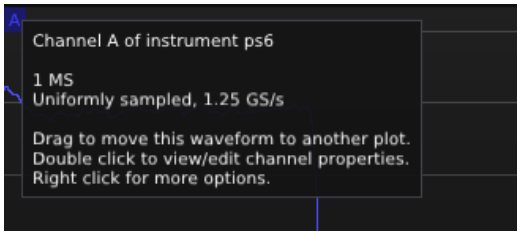



Figure 9.3: Example tooltip on channel label

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

Double-clicking the label opens the channel properties dialog (Fig. 9.4). As with all dialogs in ngscopeclient, the properties dialog may be left in the default floating state or docked.

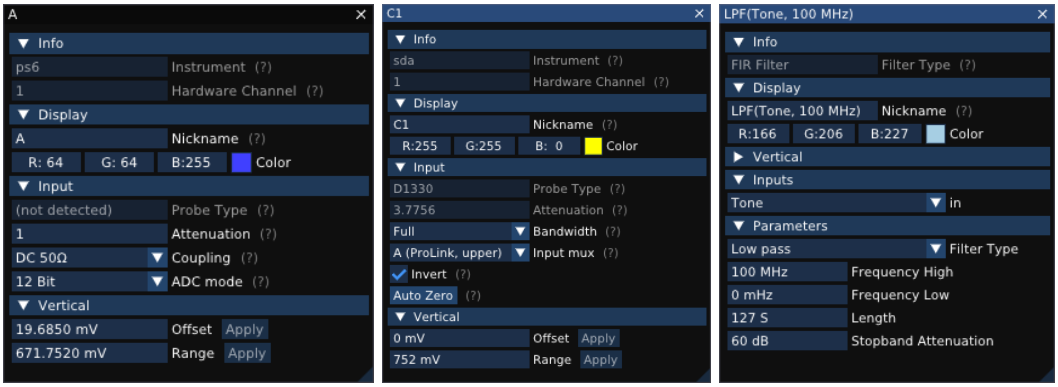



Figure 9.4: Example of properties dialogs for three different channels

The properties dialog will always contain an editable nickname for the channel, a color chooser, and some basic information about the instrument channel or filter block sourcing the data. Additional settings may be available but will vary depending on the type of instrument or filter. In Fig. 9.4, the left dialog shows a direct coaxial input to a Pico PicoScope 6824E, which has variable ADC resolution. The center dialog shows an active differential probe with auto-zero capability, connected to a Teledyne LeCroy SDA816Zi-A which has a mux for selecting between two input connectors for each channel. The right dialog shows a FIR filter with several configurable settings.

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

## 9.5 Cursors and Markers

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

### 9.5.1 Horizontal Cursors

A horizontal cursor describes a position (usually voltage) on the Y axis. As of this writing, there is no readout functionality; horizontal cursors are purely for visual/display purposes.

Horizontal cursors are attached to a waveform view; any number of views may have independent horizontal cursors at a given time.




Figure 9.5: Single and double horizontal cursors

### 9.5.2 Vertical Cursors

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

To add a vertical cursor (Fig. 9.6), right click in the view and select a single or double cursor from the `Cursors | X Axis` menu.

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




Figure 9.6: Single vertical cursor




Figure 9.7: Double vertical cursor

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

In the timeline each cursor will display its X-axis position. If both cursors are active, the delta

between them is shown. If the X axis uses time units, the frequency with period equal to the cursor spacing is also shown.

When a cursor is active, a dockable pop-up dialog appears displaying the value of each waveform (including analog, digital, and protocol channels) in the group at the cursor location. If two cursors are active, both values as well as the difference between them (if analog) is shown (Fig. 9.7)

### 9.5.3 Markers

A marker is a named location in *absolute* time intended for marking specific events (such as protocol packets or glitches) which may need to be re-examined in the future. When new waveforms are acquired, the marker remains attached to the same point in the old waveform and will disappear from view until the old waveform is re-loaded from the history window. In Fig. 9.8, three of the four markers are visible while the fourth is in a different waveform.

Unlike vertical cursors, which are local to a single waveform group, cursors are global and will appear at the same timestamp in all waveform groups. This allows an event of interest to be examined in detail in one view, while a different view provides a global overview of the entire acquisition or examines another event (Fig. 9.9).

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

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

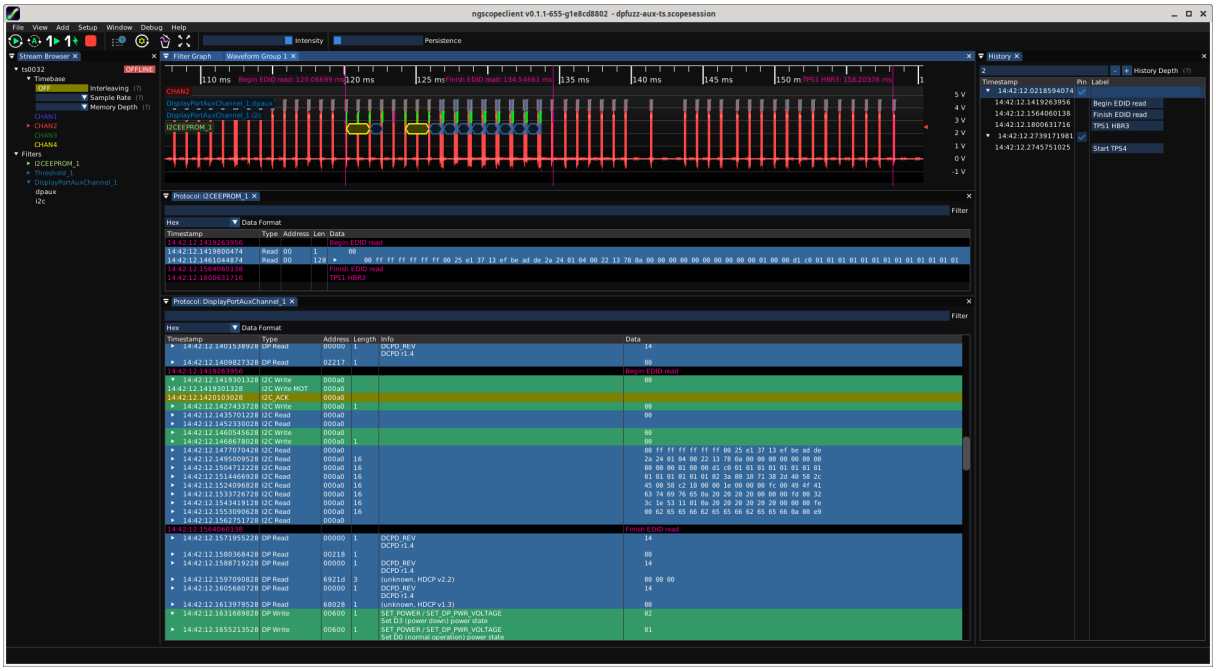


Figure 9.8: Session with four markers, three on the currently displayed waveform and one on a future waveform

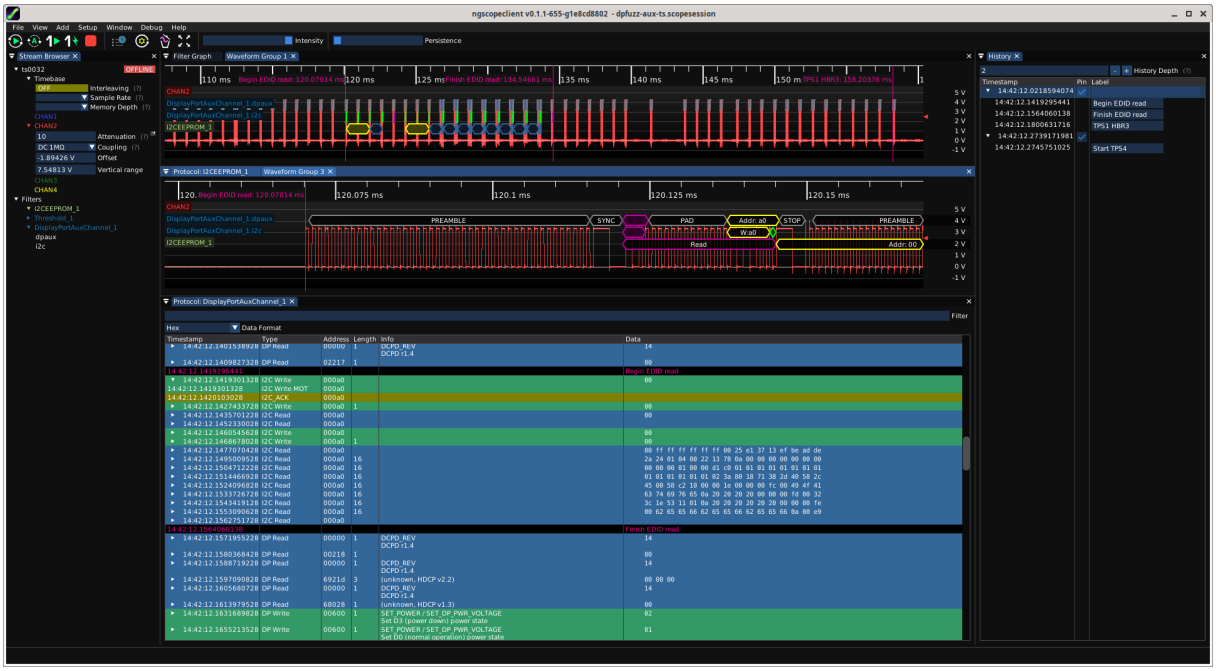



Figure 9.9: A single marker seen at multiple time scales in different views

# Chapter 10

# History

ngscopeclient saves a rolling buffer of previous waveforms in memory, allowing you to go back in time and see previous state of the system being debugged. This buffer is included in saved sessions, allowing a full snapshot of system behavior to be loaded for future analysis. History is always captured up to the configured depth, regardless of whether the history view window is displayed or not.

Clicking on a timestamp in the history view (Fig. 10.1) pauses acquisition and loads the historical waveform data for analysis.

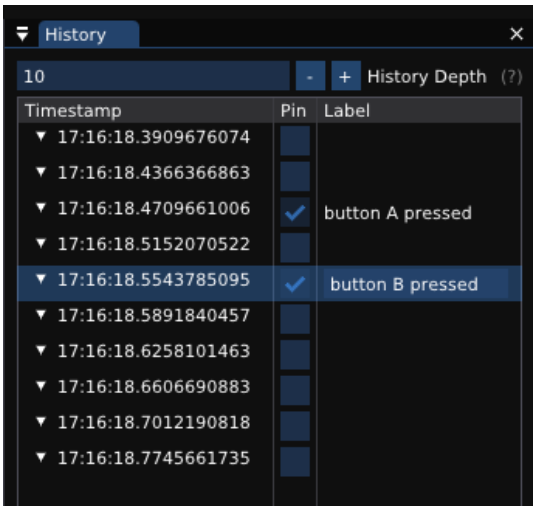



Figure 10.1: Waveform history view

Hovering the mouse over a row in the history view (Fig. 10.2) displays a tooltip with the full date and time of the acquisition, as well as some information about which instruments and channels had data in the capture (if some channels were enabled or disabled during the lab session, the set of active channels may have changed throughout history) and how much memory the capture is consuming.

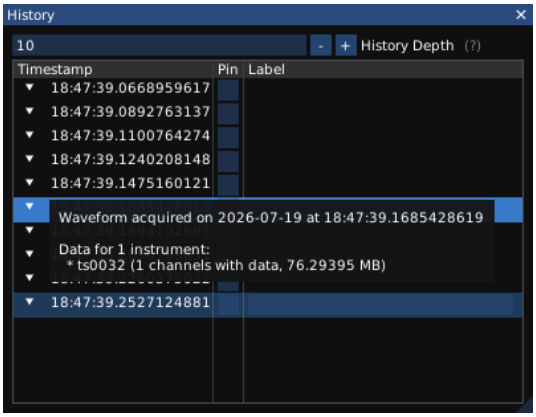



Figure 10.2: Tooltip on waveform history entry

The history depth defaults to 10 waveforms, but can be set arbitrarily within the limits of available RAM. All history must fit in system RAM in the current software version; spilling to disk is planned for the future (scopehal-apps:311). Older waveforms beyond the history limit are deleted automatically as new waveforms are acquired. Any single waveform in history may also be deleted by right clicking on the line and selecting “delete" from the menu.

## 10.1 Pinning

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

If a waveform contains markers (9.5.3), it is automatically pinned and cannot be unpinned unless the marker (or entire waveform) is manually deleted. This prevents accidental loss of an important waveform: if the event was important enough to mark and name, it is probably worth keeping around.

## 10.2 Labeling

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

# Chapter 11

# Stream Browser

## 11.1 Introduction

The stream browser is the primary navigation pane for quick access to instrument settings and channels. By default in a new ngscopeclient session it is docked to the left side of the window but it can be repositioned as needed.

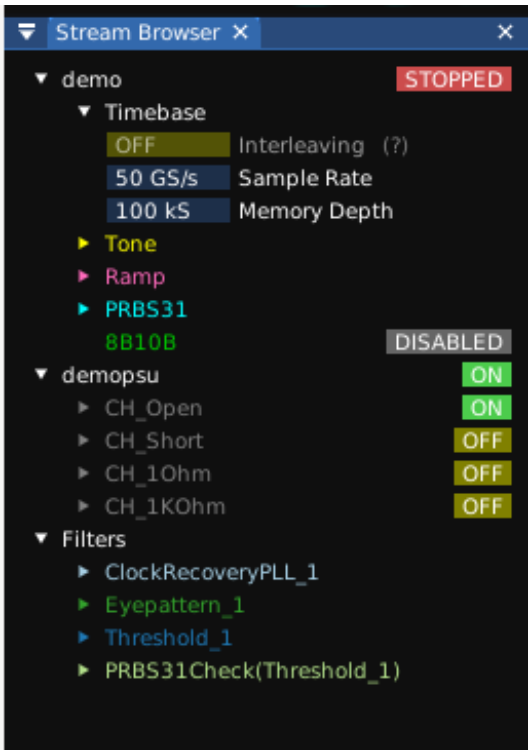



Figure 11.1: Example configuration of the stream browser

NOTE: The stream browser is one of the more recent additions to ngscopeclient and is undergoing rapid development. Some changes to appearance and interaction metaphors should be expected in upcoming versions as we unify the user interface to be more consistent for all types of instrument.

## 11.2 Filters

The "filters" node contains a list of all filter graph blocks active in the current session. Expanding the node for a given filter will show a list of output streams.

## 11.3 Instruments

The node for each instrument contains a list of all channels on the instrument, whether enabled or not. Depending on the type of instrument, additional nodes with commonly used settings such as timebase configuration may be available.

Color coded “badges" are displayed in the right margin next to channels or instruments in some cases, displaying context-dependent information such as trigger state, power supply or signal generator on/off status, etc. These badges are clickable to change the relevant setting.

## 11.4 Adding streams

Instrument channels, filters, and streams can be dragged from the browser to other parts of ngscopeclient in order to visualize or interact with data from them. For example:

- Drag a scalar stream to the measurements window to display its real-time value
- Drag a currently-inactive channel to the filter graph view to add a node for the channel so it can be used
- Drag a stream to a waveform area to display it in that plot

# Chapter 12

# Filter Graph Editor

## 12.1 Introduction

The filter graph editor allows complex signal processing pipelines to be developed in a graphical fashion and is the first window created automatically in a new session. It may be reopened from the `Window | Filter Graph` menu item if closed.

The graph editor view (Fig. 12.1) shows nodes for every instrument channel, trigger, and filter block used in the current session. Additional nodes for inactive channels may be created by dragging them from the stream browser into the graph editor canvas.

Nodes cannot overlap and will automatically move out of the way if another node is dragged on top of them.




Figure 12.1: Filter graph editor showing instrument channels and several processing blocks

## 12.2 Interaction

The view may be zoomed with the mouse wheel, or panned by dragging with the right mouse button, to navigate large filter graphs which do not fit on a single screen at a reasonable zoom level. Right clicking on a node opens a pop-up properties view (Fig. 12.2).




Figure 12.2: Filter graph editor showing properties popup

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

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

Hovering the mouse over an input port displays a tooltip showing what inputs are allowed to be connected to the port (Fig. 12.3). Typical restrictions include accepting only analog or only digital input, only sparse or only uniform sampling, specific X or Y axis units, or a particular protocol data type.

Hovering the mouse over an output port displays a tooltip describing the data provided by that stream (Fig. 12.4). Exact data available depends on the type of waveform (analog, digital, protocol, density plot, etc) but typically includes units, sample rate, number of points, and memory used.

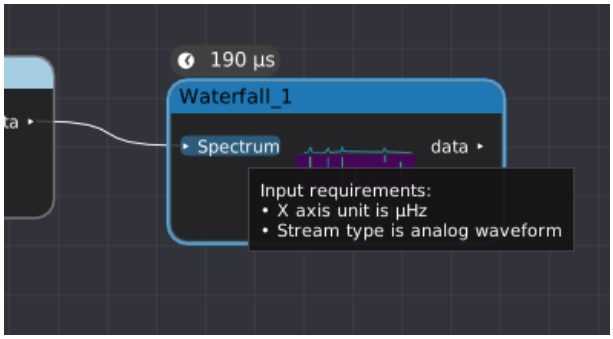



Figure 12.3: Tooltip on input port showing the requirements for source data

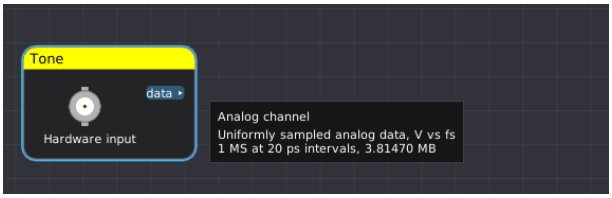



Figure 12.4: Tooltip on output port showing the available data

To create a new node, click on an input or output port and drag to an empty area of the canvas (Fig. 12.5, Fig. 12.6). A context menu will appear, presenting a list of filters which can accept (if dragging from an output) or produce (if dragging from an input) the desired data type. If dragging from an input, the context menu will also include any currently unused instrument channels.

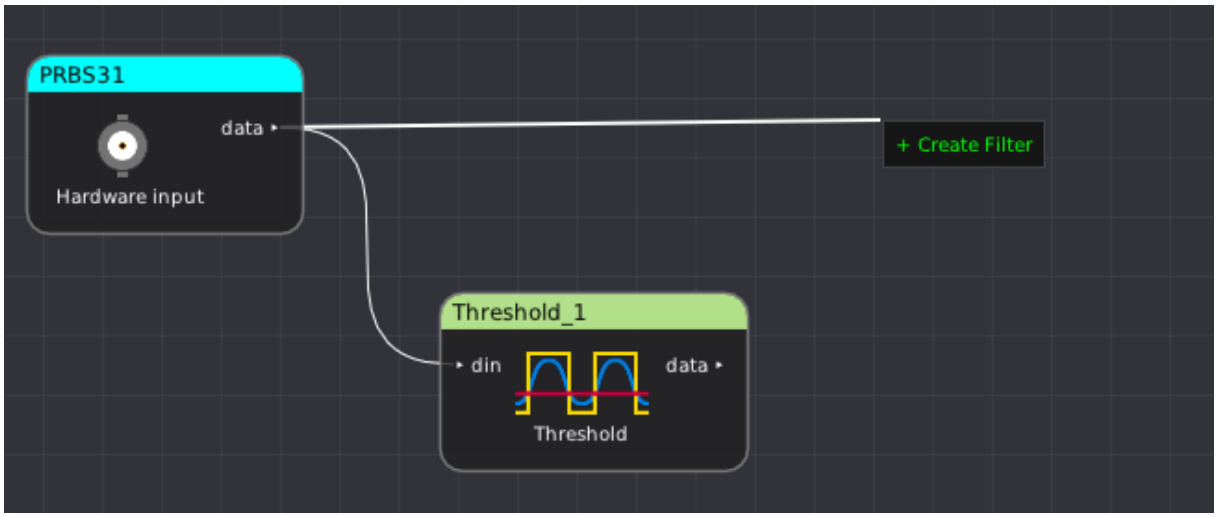



Figure 12.5: Filter graph editor dragging from an output to an empty area of the canvas

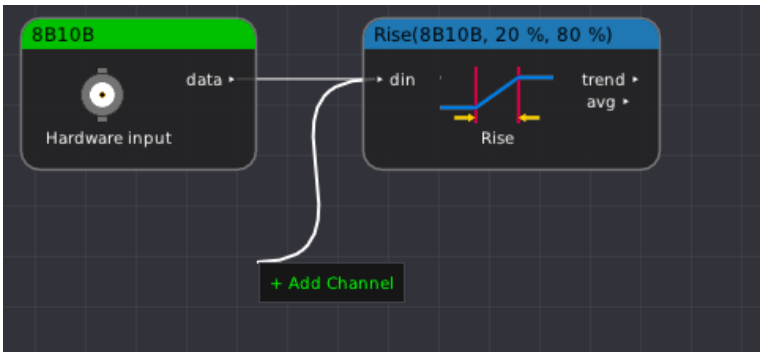



Figure 12.6: Filter graph editor dragging from an input to an empty area of the canvas

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

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

Each node also includes a caption stating the type of node (“hardware input", “hardware output",

or the name of the filter block) and, in most cases, an icon depicting the functionality of the block.[1]

If a filter is missing required inputs, has a nonsensical configuration such as dividing by zero, or is otherwise unable to generate a valid output the node will be outlined in red (Fig. 12.7) and a brief one-line description of the error will be displayed above the node. More details can be found in the errors list window, which docks to the bottom of the graph editor by default but can be dragged and moved like any other window. (The error list is hidden automatically when there are no errors, to save screen space)




Figure 12.7: Example graph node displaying an error

## 12.3 Grouping

In order to better organize complex experimental setups, nodes may be organized in groups. Groups cannot be nested.

To create a group, right click an unused area of the graph editor canvas and select “New Group" from the context menu. This will spawn a new, empty group near the mouse cursor position.

The group will have an automatically generated name (Fig. 12.8) by default. This name may be changed by right clicking on the group’s title bar and typing a new name in the pop-up.

[1]Not all filters currently have icons. We are working with multiple artists to create more filter icons and welcome additional contributions.

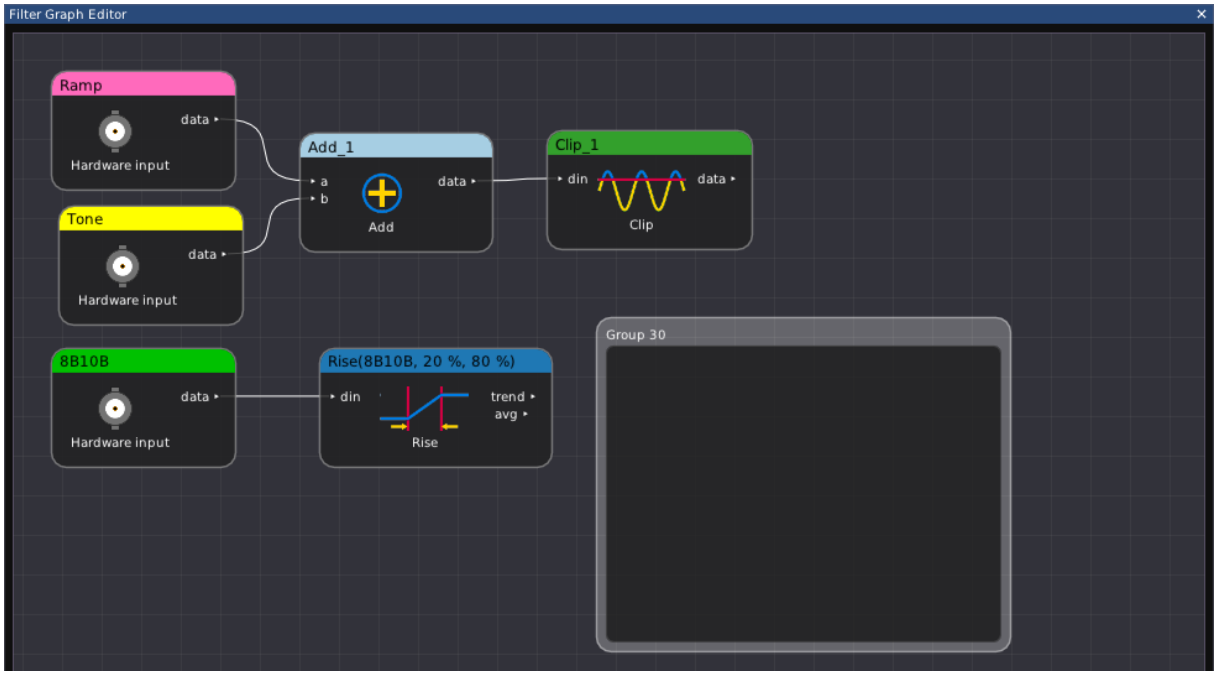



Figure 12.8: Newly created node group

To add a node to a group, simply drag the node by its title bar and move it into the group (Fig. 12.9). All paths from the node to the remainder of the filter graph will be routed through “hierarchical ports" at the left and right edges of the group, reducing clutter. Nodes may be freely moved around within the group to organize them, or dragged out of the group to remove them from the group.

A group (together with its contents) may be moved by dragging the group’s title bar with the left mouse button, or resized by dragging any of its corners. When a group is moved, it will push other nodes or groups out of the way to prevent overlapping.

If not needed, a group can be deleted by selecting it with the left mouse button and pressing the “delete" key. Deleting a group does not remove any nodes contained within it.

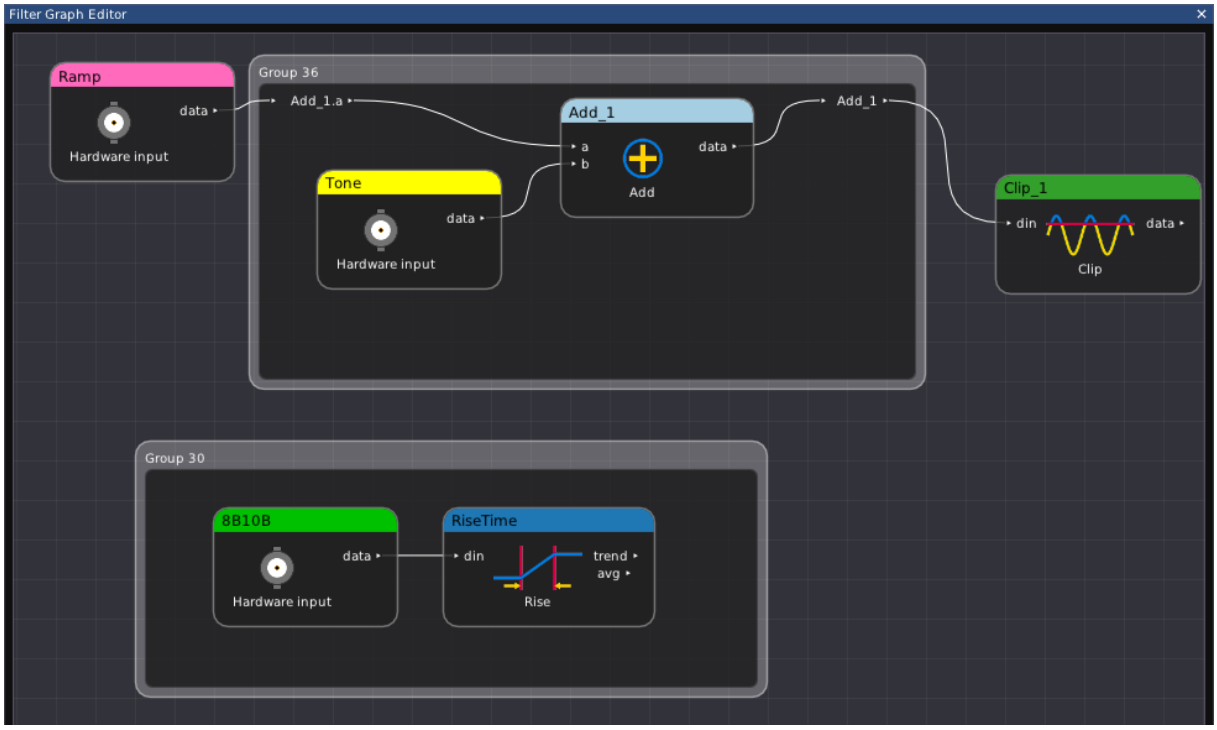



Figure 12.9: Groups containing several nodes with hierarchical ports

## 12.4 Performance Analysis

A clock icon displayed above filter nodes shows the execution time of the block, which can be used to help identify the reason for slowdowns in complex filter graphs.

The workload scheduler tries to batch multiple nodes when possible and send them to the GPU as a single unit to reduce CPU-GPU round trip overhead. When this occurs, all nodes in the batch will show the same execution time and it is not possible to use the graph view to determine which one is the bottleneck.

If you are a developer trying to optimize a specific filter block, finer grained performance data can be obtained using the appropriate profiling tool for your GPU (NVIDIA NSight Systems, AMD Radeon GPU Profiler, etc). When running under a debugger or profiler, ngscopeclient automatically adds tracking events using NVIDIA NVTX and VK_EXT_debug_utils that allow specific shaders to be easily identified in traces. automatically adds debug events

# Chapter 13

# Transports

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

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

## 13.1 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. We currently support using a single GPIB device per GPIB board (interface) in a ngscopeclient session. You cannot currently access multiple devices in the same or across instances. Better support for multiple instrument on a single board is planned.

NOTE: The current implementation of this driver only works on Linux, using the linux-gpib library. Since linux-gpib is licensed under GNU GPL, this driver is packaged separately and must be installed from https://github.com/ngscopeclient/scopehal-plugins-gpl. A future version which interfaces with the Linux kernel GPIB subsystem via the ioctl interface, without using linux-gpib, is planned for the future which will eliminate the need for a plugin.

Example:

```
ngscopeclient myscope:keysightdca:gpib:0:7
```

## 13.2 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:

```
ngscopeclient myscope:rigol:lan:192.0.2.9:5555
```

## 13.3 lxi

SCPI over LXI VXI-11.

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

Example:

```
ngscopeclient myscope:tektronix:lxi:192.0.2.9
```

## 13.4 null

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

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

Example:

```
ngscopeclient sim:demo:null:blah
```

## 13.5 socketcan

This transport provides a bridge for accessing the native Linux SocketCAN API from the scopehal driver layer. When paired with the "socketcan" oscilloscope driver and a suitable CAN peripheral, it allows ngscopeclient to be used as a CAN bus protocol analyzer. Since SocketCAN is a Linux-only API, this transport is not available on other platforms.

This transport takes one argument: the device name (e.g. "can0")

```
ngscopeclient dongle:socketcan:socketcan:can0
```

## 13.6 twinlan

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

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

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

```
ngscopeclient ts1:thunderscope:twinlan:localhost:5025:5026
```

## 13.7 uart

SCPI over RS-232 or USB-UART.

This transport takes 3 arguments: device path (required), baud rate (optional), and flags (optional). If baud rate is not specified, it defaults to 115200.

Example:

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

Some devices, like the NanoVNA, need to activate the DTR line to communicate. This can be achieved by adding the DTR flag to the connection string.

Example:

```
ngscopeclient NanoVNA-F:nanovna:uart:COM6:115200:DTR
```

## 13.8 usbtmc

SCPI over USB Test & Measurement Class protocol.

This transport takes two arguments: the path to the usbtmc kernel device object and the TMC transfer size (optional). As Workaround for Siglent SDS1x04x-E set size to 48.

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

Example:

```
ngscopeclient myscope:siglent:usbtmc:/dev/usbtmc0:48
```

## 13.9 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:

```
ngscopeclient myscope:lecroy:vicp:192.0.2.9
```

## 13.10 hid

This transport layer is usually used by binary drivers to communicate with Test & Measurement equiment over USB.

This transport takes two arguments: vendor ID (hex) and product ID (hex).

Optionally a third argument can be added with the instrument's serial number. If not provided, the instrument's serial number is automatically pulled at first connection and stored in the connection string.

Example:

```
ngscopeclient AlientekDP100:alientek_dp:hid:2e3c:af01
```

# Chapter 14

# BERT Drivers

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

## 14.1 Antikernel Labs

| Device Family | Driver | Transport | Notes |
|---|---|---|---|
| AKL-TXB1 | akl.crossbar | lan | |

### 14.1.1 akl.crossbar

This is the driver for the AKL-TXB1 trigger crossbar and CDR trigger system. The front panel transceiver ports can also be used as a BERT.

## 14.2 MultiLANE

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

### 14.2.1 mlbert

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

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

# Chapter 15

# Function Generator Drivers

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

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

## 15.1 Owon

| Device Family | Driver | Transport | Notes |
|---|---|---|---|
| XDG 2000/3000 series | owon_xdg | lan/usbtmc | Only tested via lan transport, but USBTMC is available too. |

### 15.1.1 owon_xdg

This driver supports all XDG 2000/3000 series function / arbitrary waveform generators.

It has been tested on an Owon XDG 2035.

The default communication port for lan is 3000.

## 15.2 Rigol

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

### 15.2.1 rigol_awg

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

## 15.3 Siglent

| Device Family | Driver | Transport | Notes |
|---|---|---|---|
| SDG2000X series | siglent_awg | lan | Only tested via lan transport, but USBTMC is available too |

### 15.3.1 siglent_sdg

This driver supports the SDG2000X series and possibly higher end models as well.

# Chapter 16

# Electronic Load Drivers

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

## 16.1 Siglent

| Device Family | Driver | Transport | Notes |
|---|---|---|---|
| SDL1000X/X-E series | siglent_load | lan | Only tested via lan transport, but USBTMC and serial are available too |

### 16.1.1 siglent_load

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

# Chapter 17

# Multimeter Drivers

This chapter describes all of the available drivers for multimeters.

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

## 17.1 Owon

| Device Family | Driver | Transport | Notes |
|---|---|---|---|
| Owon XDM1041/1241 | owon_xdm | uart | Driver developed and tested on this meter |
| Owon XDM2041 | owon_xdm | uart | Not tested but should work |

### 17.1.1 owon_xdm

This driver supports Owon XDM 1000 and 2000 series multimeters.

## 17.2 Rohde & Schwarz

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

### 17.2.1 rs_hmc8012

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

# Chapter 18

# Miscellaneous Drivers

This chapter describes all of the available drivers for miscellaneous instruments which do not fit in any other category.

## 18.1 Antikernel Labs

| Device Family | Driver | Transport | Notes |
|---|---|---|---|
| APB GPIO | akl.gpio | lan | |
| APB VIO | akl.vio | lan | |

### 18.1.1 akl.gpio

This driver provides a 32-bit read/write GPIO interface to the `APB_GPIO` FPGA IP from antikernel-ipcores, via via scopehal-akldebug-bridge.

### 18.1.2 akl.vio

This driver provides a multi-bit digital I/O interface with up to eight input and eight output ports to the `APB_VIO` FPGA IP from antikernel-ipcores, via scopehal-akldebug-bridge.

## 18.2 Generic

| Device Family | Driver | Transport | Notes |
|---|---|---|---|
| N/A | csvstream | Any | |

### 18.2.1 csvstream

This driver exposes the most recent line from a stream of comma-separated value (CSV) data as a series of analog scalar channels.

It is primarily intended for extracting low rate I2C sensor readings and ADC values from an embedded DUT, so that that these values may be plotted alongside multimeter/power supply readings or other data coming from more conventional instrumentation.

The data may come from any supported transport, however it is expected that the most likely scenario is either direct connection to a local serial port (“uart" transport), or a TCP socket connected to either a remote UART using socat or an embedded TCP server (“lan" transport).

Data must be generally line oriented and UTF-8 or 7-bit ASCII encoded.

In order to enable csvstream data to share a UART also used by other traffic such as a debug console or syslog, all lines must contain one of three magic prefixes as shown below. Any content in the line before the prefix (such as a timestamp) is ignored.

Upon initial connection, the driver will have a single channel called “CH1". At any time, if the number of fields in a received CSV line exceeds the current channel count, a new channel will be created. If a partial line is received, the values in the missing columns are unchanged but the channel will not be deleted.

- **CSV-NAME**: Contains channel name data. Example:

  ```
  CSV-NAME,Temperature,3V3,RxLevel
  ```

- **CSV-UNIT**: Contains channel unit data (using the text encodings used by the libscopehal Unit class). Example:

  ```
  CSV-UNIT,°C,V,dBm
  ```

- **CSV-DATA**: Contains channel value data. Example:

  ```
  CSV-DATA,31.41,3.291,-59.1
  ```

# Chapter 19

# Oscilloscope Drivers

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

## 19.1 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. |
| EDUX1000 series | agilent | lan | Untested but should be identical to DSOX1200 but with lower sample memory. |
| DSOX1200 series | agilent | lan | Working. No support for wavegen yet. |
| MSOX-2000 series | agilent | lan | |
| MSOX-3000 series | agilent | lan | |

### 19.1.1 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 |

## 19.2 Antikernel Labs

| Device Family | Driver | Notes |
|---|---|---|
| Internal Logic Analyzer IP | akl.ila | |

### 19.2.1 akl.ila

This driver interfaces to the Antikernel Labs `APB_ILA` or any other RTLDebug compatible internal logic analyzer core, via scopehal-akldebug-bridge

## 19.3 Demo

The “demo" driver is a simulation-only driver for development and training purposes, and does not connect to real hardware. It does not currently model realistic trigger behavior (waveforms are aligned to a fixed phase regardless of trigger setting), however we hope to simulate realistic triggering in the future.

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

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

It supports memory depths of 10K, 100K, 1M, and 10M points per waveform at rates of 1, 5, 10, 25, 50, and 100 Gsps. Four analog test signals are provided, as well as a series of digital signals.

The analog signals can be configured in three different modes using the “ADC mode" selector:

- Ideal waveform with no added impairments
- Ideal waveform with 10 mV of AWGN
- Ideal waveform with 10 mV of AWGN through an S-parameter ISI model of a 300mm FR-4 microstrip

Analog test signals:

- Tone: 1.000 GHz tone
- Ramp: 1.000 GHz steady tone plus a second tone which sweeps from 1.100 to 1.500 GHz
- PRBS31: 10.3125 Gbps PRBS-31
- 8B10B: 1.25 Gbps repeating two 8B/10B symbols (K28.5 D16.2)

Digital test signals:

- Unidirectional SPI bus transmitting an ASCII hello-world message
- UART transmitting an ASCII hello-world message
- 8-bit parallel bus plus clock counting from 00 to FF

| Device Family | Driver | Transport | Notes |
|---|---|---|---|
| Simulator | demo | null | |

## 19.4 Batronix

A driver for Batronix Magnova oscilloscope is available in the codebase. It has been developped, with the help of Batronix team, according to official SCPI documentation for Magnova oscilloscope (Magnova® SCPI Manual - Version 1.5). This driver has been developed and tested on a Magnova BMO350 running firmware version 1.7.1.

Here are the known limitations of this driver (that should be fixed with further Magnova firmware updates):

- For now, Magnova's SCPI implementation does not support digital signal acquisition.
- Selecting specific Arbitrary waveform from function generator is not possible with current Magnova SCPI implementation.
- Time Scale / Memory Depth / Sample Rate synchronization only work if the scope is in "Extended Capture" mode (Acquire menu).

| Device Family | Driver | Transport | Notes |
|---|---|---|---|
| Magnova BMO | Batronix | lan | Connection path: <IP>:5025 |

**Typical Performance (Magnova BMO350, LAN)**

| Channels | Memory depth | WFM/s |
|---|---|---|
| 1 | 20K | 9.9 |
| 2 | 10K | 6.6 |
| 4 | 12.5K | 3.99 |
| 1 | 100K | 9.7 |
| 4 | 125K | 3.97 |
| 1 | 5M | 4.5 |
| 4 | 5M | 1.26 |

These figures were obtained from a Magnova BMO350 running firmware version 1.7.1.

## 19.5 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.

### 19.5.1 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<br>No analog output support |
| Analog Discovery Pro | digilent | twinlan | No digital channel support<br>No analog output support |
| Digital Discovery | digilent | twinlan | No digital channel support,<br>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 |

## 19.6 DrAndyHaas

### 19.6.1 haasoscope pro

HaasoscopePro is an affordable 2 GHz bandwidth 3.2 GS/s (expandable to 6.4 GS/s) 12-bit open-source open-hardware USB oscilloscope. For more info see the HaasoscopePro github. This driver connects to a Haasoscope Pro GUI, which must first be running, over a LAN socket. It supports 50 Hz of update rate for moderate memory depth.

| Device Family | Driver | Notes |
|---|---|---|
| DrAndyHaas | haasoscope pro | lan on port 32001 |

## 19.7 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. EDIT 2026-07-23: some user are reporting problems due to DSLabs software changes.

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

### 19.7.1 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 |

## 19.8 EEVengers

### 19.8.1 thunderscope

This driver connects to the TS.NET application to control a ThunderScope.

It supports full-rate 1 Gsps streaming given suitably fast hardware.

| Device Family | Driver | Notes |
|---|---|---|
| ThunderScope | thunderscope | Use twinlan transport to TS.NET |

### 19.8.2 Breaking Changes

* ngscopeclient version 0.3 requires TS.NET v0.1.0 or later. Older TS.NET versions are not supported.

## 19.9 Enjoy Digital

TODO (scopehal:79)

## 19.10 Generic

Drivers in this section are not specific to a particular manufacturer's products and support a wide variety of similar devices.

### 19.10.1 socketcan

This driver exposes the Linux SocketCAN API as a stream of CAN messages which can be displayed as-is or used as input to other filter graph blocks. When paired with the "socketcan" transport and a suitable CAN peripheral, it allows ngscopeclient to be used as a CAN bus protocol analyzer. Since SocketCAN is a Linux-only API, this driver is not available on other platforms.

## 19.11 Hantek

TODO (scopehal:26)

## 19.12 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 (which is also aliased as "keysight" and can be used under either name)

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

### 19.12.1 agilent

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

### 19.12.2 keysightdca

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

| Device Family | Driver | Notes |
|---|---|---|
| 86100A | keysightdca | |

## 19.13 Pico Technology

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

| Device Family | Driver | Notes |
|---|---|---|
| 2000 series | pico | 2205 MSO, 2206, 2207, 2208 only |
| 2000A series | pico | 2204A and 2205A not supported yet |
| 2000B series | pico | |
| 3000 series | pico | 3204/5/6 MSO only |
| 3000A series | pico | |
| 3000B series | pico | |
| 3000C series | pico | |
| 3000D series | pico | |
| 4000 series | pico | 4444 and 4824 only |
| 4000A series | pico | |
| 5000A series | pico | |
| 5000B series | pico | |
| 5000D series | pico | |
| 6000E series | pico | |

### 19.13.1 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 |

## 19.14 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.

NOTE: DS1054Z firmware 00.04.02.SP4 is known to have problems with SCPI remote control (scopehal-apps:790); it is unclear what other models and firmware versions may be affected by this bug. If you encounter problems, please ensure your scope is running the latest firmware release from Rigol before opening a support ticket.

| Device Family | Driver | Notes |
|---|---|---|
| DS1100D/E | rigol | |
| DS1000Z | rigol | |
| MSO5000 | rigol | |
| DHO800 | rigol | |
| DHO900 | rigol | (no digital channels) |
| DHO1000 | rigol | (untested) |
| DHO4000 | rigol | (untested) |
| MHO900 / MHO98 | rigol | (no digital channels) |

### 19.14.1 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 |

**Typical Performance (DHO800/900 series, LAN)**

| Channels | Memory depth | WFM/s |
|---|---|---|
| 1 | 1K | 11.9 (live mode available for 1Kpt/single channel) |
| 2 | 1K | 3.4 |
| 4 | 1K | 1.66 |
| 1 | 10K | 3.31 |
| 2 | 10K | 2.90 |
| 4 | 10K | 1.65 |
| 1 | 100K | 3.30 |
| 4 | 100K | 1.64 |
| 1 | 1M | 1.63 |
| 4 | 1M | 0.57 |
| 1 | 10M | 0.30 |
| 4 | 10M | 0.07 |

## 19.15 Rohde & Schwarz

### 19.15.1 rs

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

TODO (scopehal:59)

### 19.15.2 rs_rto6

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

### 19.15.3 rs_rtb2k

A driver for RTB2000 and RTB2 oscilloscope. Analog and digital channels are supported. An external trigger and a line trigger are available. The function generator can be used with some restrictions.

This driver is in early stage and has significant limitations and bugs. We are exploring whether to merge it with the original legacy `rs` driver or continue updating it, but no progress can be made without a volunteer maintainer who has access to a scope for testing. If you are interested in taking over maintainership of this driver please reach out!

- The bit depth can be changed between 8 bits and 16 bits in the driver settings.
- The selection lists with the sample rate and memory depth only work correctly with a fixed memory depth.
- Entries in the selection list for the memory depth are rounded up, resulting in several identical entries.

- When the oscilloscope is in Run mode and connected to ngscopeclient, data acquisition starts automatically.
- When connecting the oscilloscope with line trigger to ngscopeclient, a trigger channel must be selected before changing the trigger. Otherwise, the program will crash.
- The progress bar for downloading digital channels is only displayed for one channel.
- If only the first logic probe is connected, only these 8 channels are displayed. The second logic probe is not recognized during operation.
- The triggers for pattern and bus are missing. Many other triggers are partially implemented in the code but disabled due to bugs and will need refactoring to be usable.

| Device Family | Driver | Transport | Notes |
|---|---|---|---|
| RTB | rs.rtb2k | lan | Connection path: <IP>:5025 |

**Typical Performance (RTB2000, LAN)**

| Channels | Memory depth | WFM/s |
|---|---|---|
| 1 | 10K | 3.4 |
| 2 | 10K | 2.6 |
| 4 | 10K | 1.8 |
| 1 | 100K | 2.6 |
| 2 | 100K | 2.0 |
| 4 | 100K | 1.4 |
| 4 | 1M | 0.3 |

These figures were obtained from a RTB2000 running firmware version 3.000 on an Raspberry Pi 500.

## 19.16 Saleae

TODO (scopehal:16)

## 19.17 Siglent

A driver for SDS2000X+/HD is available in the codebase which has been developed according to Siglent offical documentation (Programming Guide PG01-E11A). This driver should be functional across the 'next generation' SDS800X HD, SDS1000X HD, SDS2000X+, SDS2000X HD, SDS5000X, SDS6000A/L/Pro and SDS7000A scopes. It has been primarily developed using the SDS2000X+ and SDS2000X HD. 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 |
| SDS800X HD series | siglent | lan | Basic functionality complete/tested. |
| SDS1000X HD series | siglent | lan | Basic functionality complete, needs testing. |
| SDS2000X+ series | siglent | lan | Basic functionality complete. |
| SDS2000X HD series | siglent | lan | Tested and works well on SDS2354x HD. |
| SDS3000X HD series | siglent | lan | Basic functionality complete, needs testing. |
| SDS5000X series | siglent | lan | Initialises, triggers and downloads waveforms. More testing needed |
| SDS6000A/L/Pro series | siglent | lan | Tested and works well on SDS6204A. 10/12 bit models NOT supported, but unavailable for dev (not sold in western markets). |
| SDS7000A series | siglent | lan | Basic functionality complete, needs testing. |

### Typical Performance (SDS2104X+, LAN)

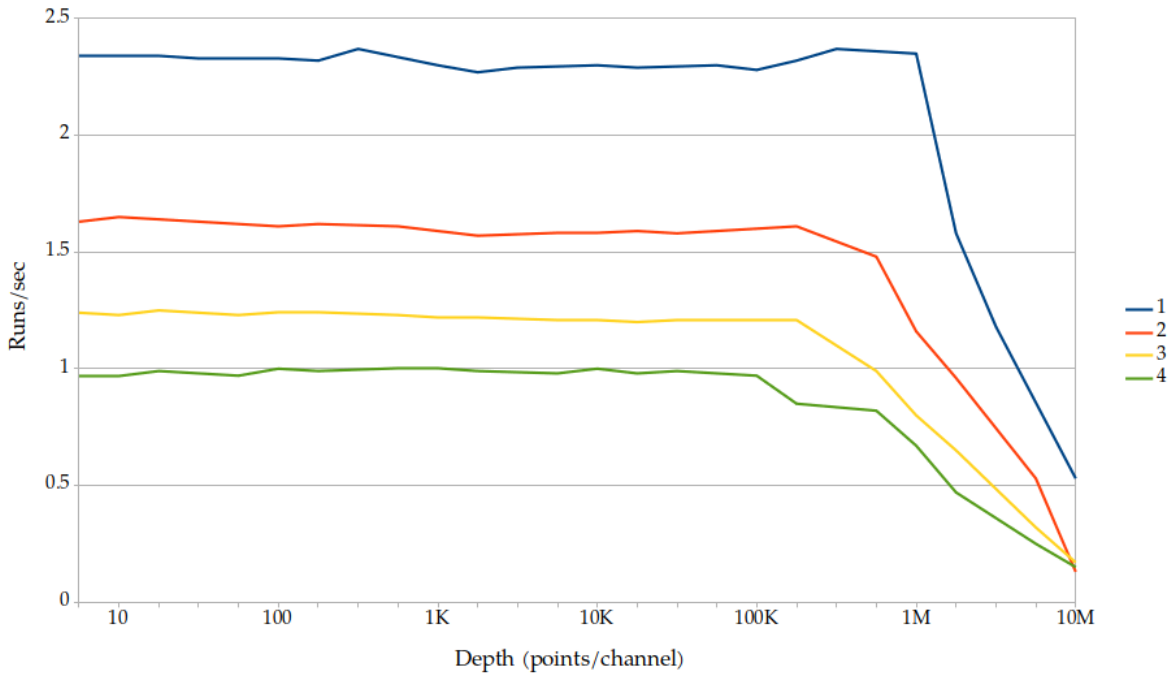



Figure 19.1: Siglent sample speed for various combinations of depth and channels

| 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.

**Typical Performance (SDS2104X HD, LAN)**

| Channels | Memory depth | WFM/s |
|---|---|---|
| 1 | 10K | 8.2 |
| 2 | 10K | 7.7 |
| 4 | 10K | 5.4 |
| 1 | 100K | 7.1 |
| 4 | 100K | 4.2 |
| 1 | 5M | 0.72 |
| 4 | 5M | 0.09 |

These figures were obtained from a SDS2104X HD running firmware version 1.2.2.9.

## 19.18 Teledyne LeCroy / LeCroy

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

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

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

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

### 19.18.1 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 |

### 19.18.2 lecroy_fwp

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

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

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

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

- Maxmimum memory depth is limited to no more than 40M samples per channel, regardless of installed instrument memory. This is an architectural limitation of the FastWavePort API; the next generation FastMultiWavePort API eliminates this restriction however scopehal-fwp-bridge does not yet support it due to poor documentation.
- MSO channels are not supported, because neither FastWavePort nor FastMultiWavePort provide shared memory access to digital channel data. There is no known workaround for this given current instrument APIs.
- A maximum of four analog channels are supported even if the instrument actually has eight. There are no major technical blockers to fixing this under FastWavePort however no 8-channel instruments are available to the developers as of this writing, so there is no way to test potential fixes. FastMultiWavePort has a limit of four channels per instance, but it may be possible to instantiate multiple copies of the FastMultiWavePort block to work around this.
- Math functions F9-F12 are used by the FastWavePort blocks and cannot be used for other math functions.

## 19.19 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 | |

### 19.19.1 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 |

## 19.20 tinySA

This driver is meant to be used with tinySA and tinySA ULTRA spectrum analyzers.

It has been developed and tested on a tinySA ULTRA.

The communication with the device is made with UART transport layer.

| Device Family | Driver | Transport | Notes |
|---|---|---|---|
| tinySA | tiny_sa | uart | Not tested but should work |
| tinySA ULTRA | tiny_sa | uart | Driver tested on this device |

## 19.21 Xilinx

TODO (scopehal:40)

# Chapter 20

# SDR Drivers

This chapter describes all of the available drivers for software-defined radios.

## 20.1 Ettus Research

| Device Family | Driver | Transport | Notes |
|---|---|---|---|
| USRP | uhd | twinlan | |

### 20.1.1 uhd

This driver connects via a TCP socket to a socket server scopehal-uhd-bridge which connects to the appropriate instrument using the UHD API.

This provides network transparency for USB-attached instruments, as well as a license boundary between the BSD-licensed libscopehal core and the GPL-licensed UHD API.

## 20.2 Microphase

The AntSDR running antsdr_uhd firmware is supported by the “uhd" driver for Ettus Research SDRs. There is currently no support for the IIO firmware.

# Chapter 21

# Spectrometer Drivers

This chapter describes all of the available drivers for optical (UV/VIS/IR) spectrometers.

## 21.1 ASEQ Instruments

| Device Family | Driver | Transport | Notes |
|---|---|---|---|
| LR1 | aseq | twinlan | Use scopehal-aseq-bridge |

### 21.1.1 aseq

This driver supports the ASEQ Instruments LR1 spectrometer (and possibly others, not tested) via an external bridge server and the ASEQ USB control API.

# Chapter 22

# Power Supply Drivers

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

## 22.1 Alientek

| Device Family | Driver | Transport | Notes |
|---|---|---|---|
| DP100 | alientek_dp | hid | |

### 22.1.1 alientek_dp

This driver works on Alientek DP100 mini Digital Power Supply.

Path for connection (vendorId:productId) is: 2e3c:af01.

## 22.2 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. |

### 22.2.1 gwinstek_gpdx303s

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

## 22.3 Kuaiqu

| Device Family | Driver | Transport | Notes |
|---|---|---|---|
| Kuaiqu SSPS-S series | kuaiqu_psu | uart | |
| Kuaiqu SPPS*D series | kuaiqu_psu | uart | |
| Kuaiqu SPPS-D series | kuaiqu_psu | uart (9600 baud) | Tested on a Kuaiqu SPPS-D3010-232 |
| Kuaiqu R-SPPS series | kuaiqu_psu | uart | |

### 22.3.1 kuaiqu_psu

This driver supports all Kuaiqu programmable PSUs, including SSPS-S, SPPS*D, SPPS-D and R-SPPS series.

It has been tested on a Kuaiqu SPPS-D3010-232.

## 22.4 Riden

| Device Family | Driver | Transport | Notes |
|---|---|---|---|
| Riden RD series | riden_rd | uart | Tested on a Riden RD6006 |

### 22.4.1 kuaiqu_psu

This driver supports all Riden RD series DC Power Supplies.

It has been tested on a Riden RD6006.

## 22.5 Rigol

| Device Family | Driver | Transport | Notes |
|---|---|---|---|
| DP832, DP832A | rigol_dp8xx | uart, usbtmc, lan | No support for tracking modes yet. |

### 22.5.1 rigol_dp8xx

This driver supports the DP832 and DP832A.

## 22.6 Rohde & Schwarz

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

### 22.6.1 rs_hmc804x

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

## 22.7 Siglent

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

### 22.7.1 siglent_spd

Supported models should include SPD3303X, SPD3303X-E.

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

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

## 22.8 Sinilink

| Device Family | Driver | Transport | Notes |
|---|---|---|---|
| XY series | siniLink | uart | Tested on a XYS3580 |

### 22.8.1 siniLink

This driver should be compatible with most Sinilink XY series powersupplies.

# Chapter 23

# RF Generator Drivers

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

## 23.1 Siglent

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

### 23.1.1 siglent_ssg

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

# Chapter 24

# VNA Drivers

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

## 24.1 Copper Mountain

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

### 24.1.1 coppermt

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

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

## 24.2 NanoVNA

| Device Family | Driver | Transport | Notes |
|---|---|---|---|
| NanoVNA | nanovna | uart | Not tested but should work |
| NanoVNA-D | nanovna | uart | Not tested but should work |
| NanoVNA-F | nanovna | uart | Not tested but should work |
| DeepVNA 101 | nanovna | uart | Development and tests made on this device (a.k.a. NanoVNA-F Deepelec) |
| NanoVNA-F_V2 | nanovna | uart | Not tested but should work |
| NanoVNA-H | nanovna | uart | Not tested but should work |
| NanoVNA-H4 | nanovna | uart | Not tested but should work |

### 24.2.1 nanovna

This driver supports the NanoVNA with different variants of the original design and firmware (see above).

Communication with the device uses UART transport layer with a connection string looking like this (DTR flag is required):

```
COM6:115200:DTR
```

Paginated sweep has been implemented to achieve memory depths greater then the device's internal limit.

Pagination is also used at low RBW to prevent the connection from timing out during sweep.

NanoVNA V2 (with binary protocol) is NOT supported.

## 24.3 Pico Technology

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

### 24.3.1 picovna

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

# Chapter 25

# Triggers

## 25.1 Trigger Properties

The `Setup / Trigger` menu opens the trigger properties dialog (Fig. 25.1).

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

The Position 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 25.1: Trigger properties dialog

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

## 25.2 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

[1] This is a different convention than most oscilloscopes, which typically measure the trigger position from the *midpoint* of the waveform. Since ngscopeclient 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.

specified in hex/binary patterns as “X", for example “3fx8" or “1100010xxx1".

## 25.3 Dropout

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

### 25.3.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| din | Analog or digital | Input signal |

### 25.3.2 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 |

## 25.4 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.

### 25.4.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| din | Analog or digital | Input signal |

### 25.4.2 Parameters

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

## 25.5 Glitch

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

## 25.6 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 |

### 25.6.1 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 |

## 25.7 Runt

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

| Signal name | Type | Description |
|---|---|---|
| din | Analog | Input signal |

### 25.7.1 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 |

## 25.8 Slew Rate

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

| Signal name | Type | Description |
|---|---|---|
| din | Analog | Input signal |

### 25.8.1 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 |

## 25.9 UART

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

### 25.9.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| din | Analog or digital | Input signal |

### 25.9.2 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 |

## 25.10 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 |

### 25.10.1 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 26

# Filters

## 26.1 Introduction

### 26.1.1 Key Concepts

ngscopeclient and libscopehal are based on a “filter graph" architecture internally. The filter graph is composed of nodes (instrument channels, DSP blocks, protocol decodes, file outputs, etc.) connected by edges, which represent streams of data flowing from source to sink[1]. Dataflow is strictly unidirectional and loops or cycles are not allowed (there cannot be a path, directly or through other nodes, from any node's output to one of its inputs)[2]. Data streams may be of either waveform (many samples plotted against another quantity) or scalar (a single measurement with associated unit) type.

A filter is simply a node which does not correspond to a physical instrument input or output. It may generate data in some way (loading samples from a file or synthesizing a waveform parametrically), write data to a file or pass it to an outside helper program (e.g. streaming decoded Ethernet frames to Wireshark for higher level analysis), or transform one or more input streams in some way to create new data streams. There is no architectural distinction between math functions, measurements, and protocol decodes, and it is possible to chain them arbitrarily - as long as the input and output data types are compatible, any processing operation can be applied to anything.

For example, a simple BFSK decoder can be made by applying the Frequency filter to an IF or directly sampled RF waveform, after any necessary preselector filtering to exclude interference from other channels. This produces an analog waveform plotting the frequency of the signal over time. The Threshold filter can then be used to recover the original digital data stream, which can then be fed to an arbitrary serial digital protocol decode block. But it's equally possible to, for example, lock the Clock Recovery (PLL) filter to the frequency waveform and display an eye pattern with Y axis units of frequency.

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

### 26.1.2 Conventions

A filter can take arbitrarily many inputs (vector or scalar values from the filter graph), arbitrarily many parameters (static scalar configuration settings), and outputs arbitrarily many vector or scalar outputs.

[1] Waveform views and the measurements window are internally handled as nodes in the graph as well, but this is typically not visible to users since they are not drawn in the graph view to avoid cluttering it.

[2] In computer science terms, this is a directed acyclic graph.

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.

All filters with protocol 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, but default colors are shown here.

| 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 |

## 26.2 1-Wire

Decodes the Dallas Semiconductor 1-wire bus, commonly used for EEPROMs and sensors in cost / area optimized, low speed applications.

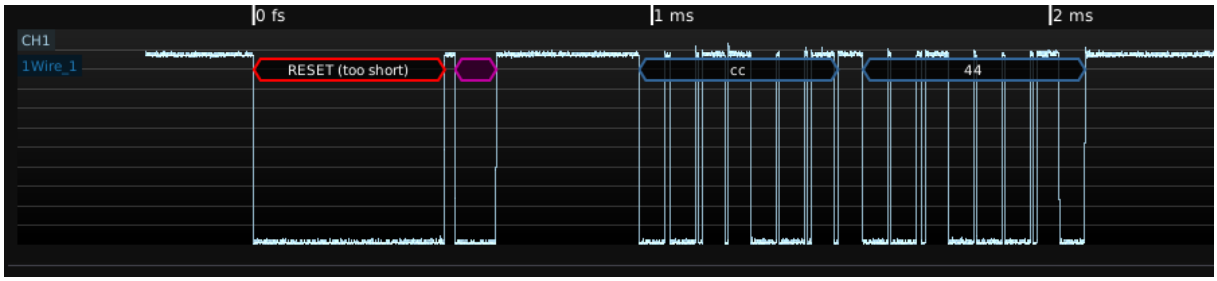



Figure 26.1: Example 1-wire decode

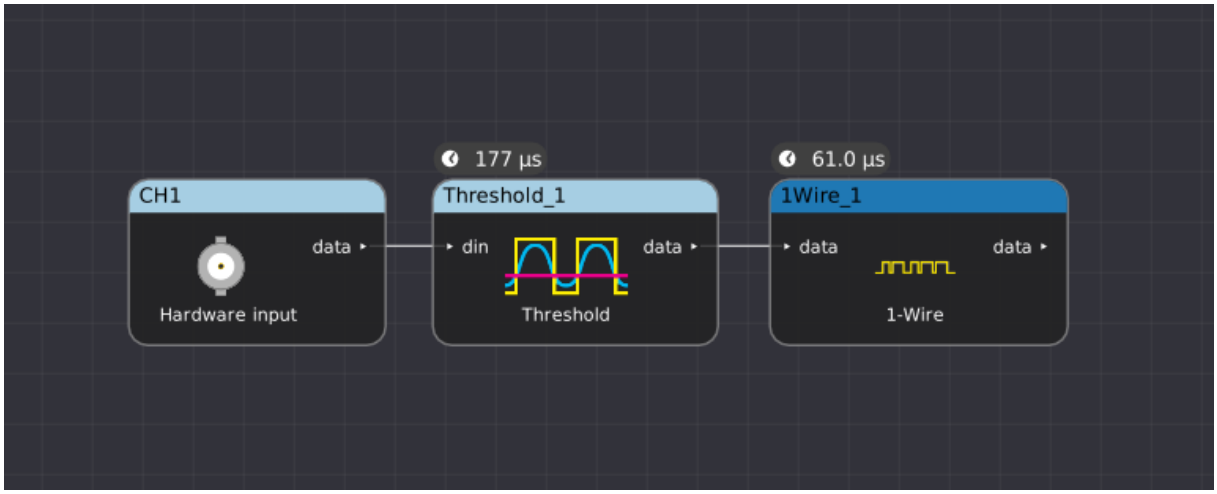



Figure 26.2: Example filter graph using decoding a 1-wire signal

### 26.2.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| data | Digital | Serial data input |

### 26.2.2 Parameters

This filter takes no parameters.

### 26.2.3 Output Signal

This filter outputs a single decoded data stream consisting of 1-wire sample objects.

| Stream name | Type | Description |
|---|---|---|
| data | Sparse protocol | Output decode |

## 26.3 128b/130b

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

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

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

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




Figure 26.3: Example 128b/130b decode

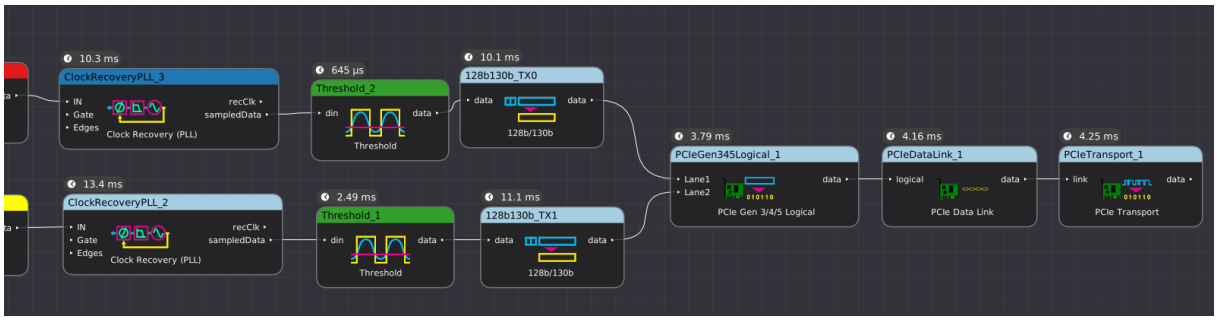



Figure 26.4: Example filter graph using 128b/130b to decode a 2-lane PCIe gen3 link

### 26.3.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| data | Sparse digital | Sampled and thresholded 128b/130b data |

### 26.3.2 Parameters

This filter takes no parameters.

### 26.3.3 Output Signal

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

| Stream name | Type | Description |
|---|---|---|
| data | Sparse protocol | Output decode |

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

### 26.3.4 Breaking Changes

This section lists major interface changes that may require projects to be updated.

- **v0.2:** This filter no longer needs the clock input that was required in v0.1. Instead, threshold the sampledData output of the Clock Recovery (PLL) and pass this to the 128b/130b decode, as shown in Fig. 26.4.

## 26.4 2-Port Shunt Through

Measures the impedance of a DUT connected to a VNA in a 2-port shunt-through topology (VNA ports 1 and 2 connected, with DUT attached between the connection point and ground). This is commonly used for measuring very low impedance networks, such as power distribution networks.

(This filter description is a stub and will be expanded in the future)

## 26.5 64b/66b

Decodes the 64/66b line code used by 10Gbase-R, Aurora, JESD204C, 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 and some other protocols use these fields for different purposes.

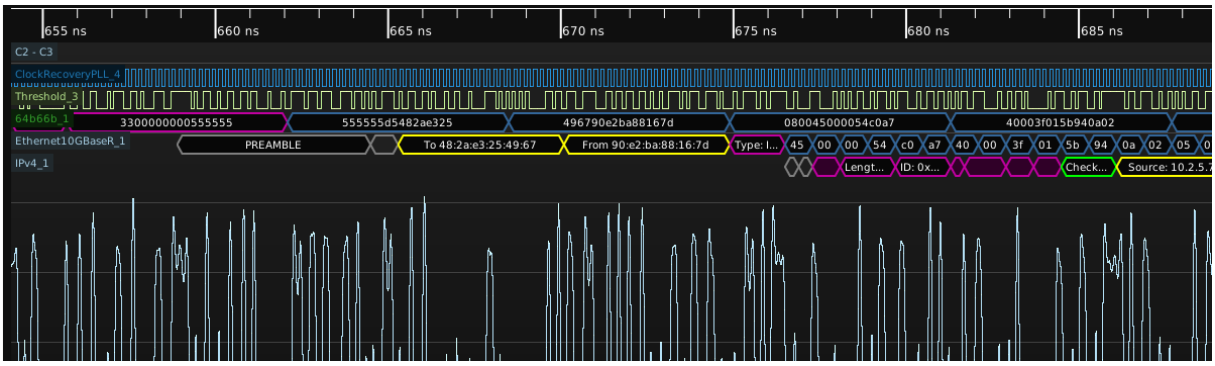



Figure 26.5: Example 64b/66b decode

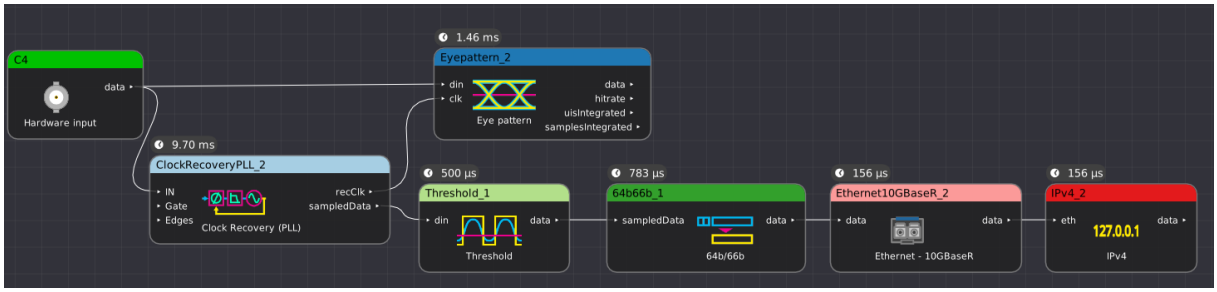



Figure 26.6: Example filter graph using 64b/66b to decode a 10Gbase-R signal

### 26.5.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| data | Sparse digital | Serial 64b/66b data line |

### 26.5.2 Parameters

This filter takes no parameters.

### 26.5.3 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.

| Stream name | Type | Description |
|---|---|---|
| data | Sparse protocol | Output decode |

| 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 |

### 26.5.4 Breaking Changes

This section lists major interface changes that may require projects to be updated.

- **v0.2:** This filter no longer needs the clock input that was required in v0.1. Instead, threshold the sampledData output of the Clock Recovery (PLL) and pass this to the 64/66b decode, as shown in Fig. 26.6.

## 26.6 8B/10B (IBM)

Decodes the standard 8b/10b line code, originally invented at IBM and now used by SGMII, 1000base-X, DisplayPort, JESD204A/B, 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 8-bit input 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.

**NOTE** If an 8B/10B signal has its polarity inverted, e.g. by putting a differential probe on backwards or swapping the pair for routability and correcting in the SERDES, *all Kx.x control characters will be unchanged* and simply flip disparity, while Dx.x data characters will be completely scrambled. If you are looking at a protocol decode that appears to be nonsense but has valid framing, try adding an Invert block before the 8B/10B decode.

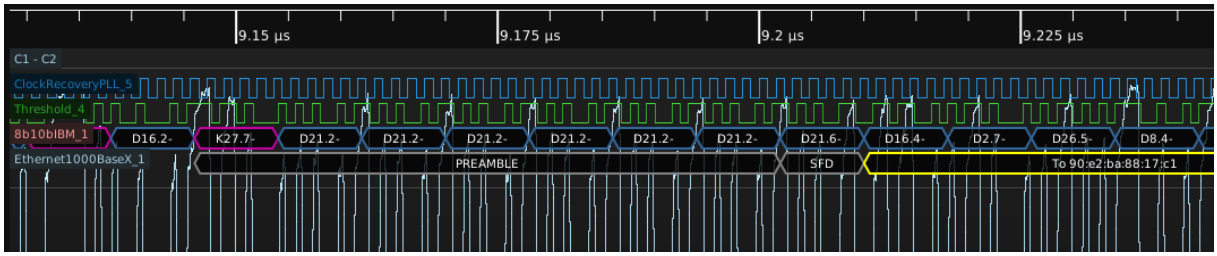



Figure 26.7: Example 8b/10b decode

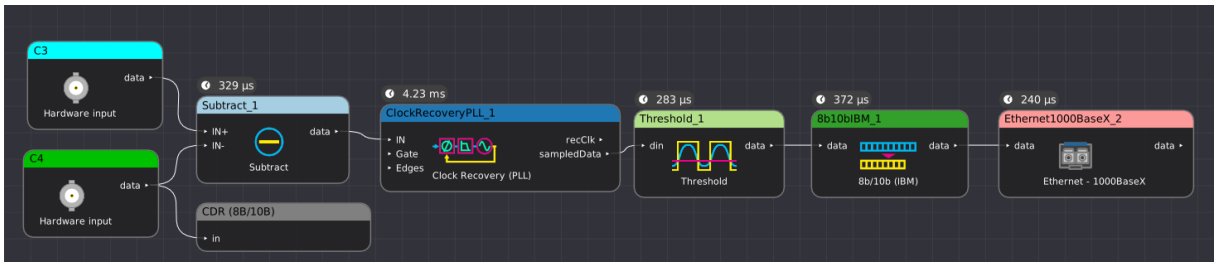



Figure 26.8: Example filter graph using 8b/10b to decode a differential 1000base-X link

### 26.6.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| data | Digital | Serial 8b/10b data line |

### 26.6.2 Parameters

| Parameter name | Type | Description |
|---|---|---|
| Comma Search Window | Integer | Number of unit intervals to search when performing comma alignment. A larger window increases the probability of a correct lock if commas are infrequent, but significantly slows down the decode. The default is 20000 UI. |
| Display Format | Enum | **Dotted (K28.5 D21.5)**: displays the 3b4b and 5b6b code blocks separately, with K or D prefix, as well as the current running disparity.<br>**Hex (K.bc b5)**: displays data as hex byte values and control codes with a K prefix. |

### 26.6.3 Output Signal

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

| Stream name | Type | Description |
|---|---|---|
| data | Sparse protocol | Output decode |

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

### 26.6.4 Breaking Changes

This section lists major interface changes that may require projects to be updated.

- **v0.2:** This filter no longer needs the clock input that was required in v0.1. Instead, threshold the sampledData output of the Clock Recovery (PLL) and pass this to the 8b/10b decode, as shown in Fig. 26.8.

## 26.7 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.

However, libscopeprotocols does not currently provide a frequency synthesis PLL which can recover a pixel clock from the 1/10 rate clock (scopehal:221). As a result, it is currently necessary to perform per-lane CDR on the TMDS data in order to decode it. This is sufficient for protocol decoding, but is not suitable for high-fidelity eye pattern measurements since skew/jitter between the TMDS lane and the clock lane will not be accounted for.




Figure 26.9: Example TMDS decode

### 26.7.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| data | Digital | Serial TMDS data line |

### 26.7.2 Parameters

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

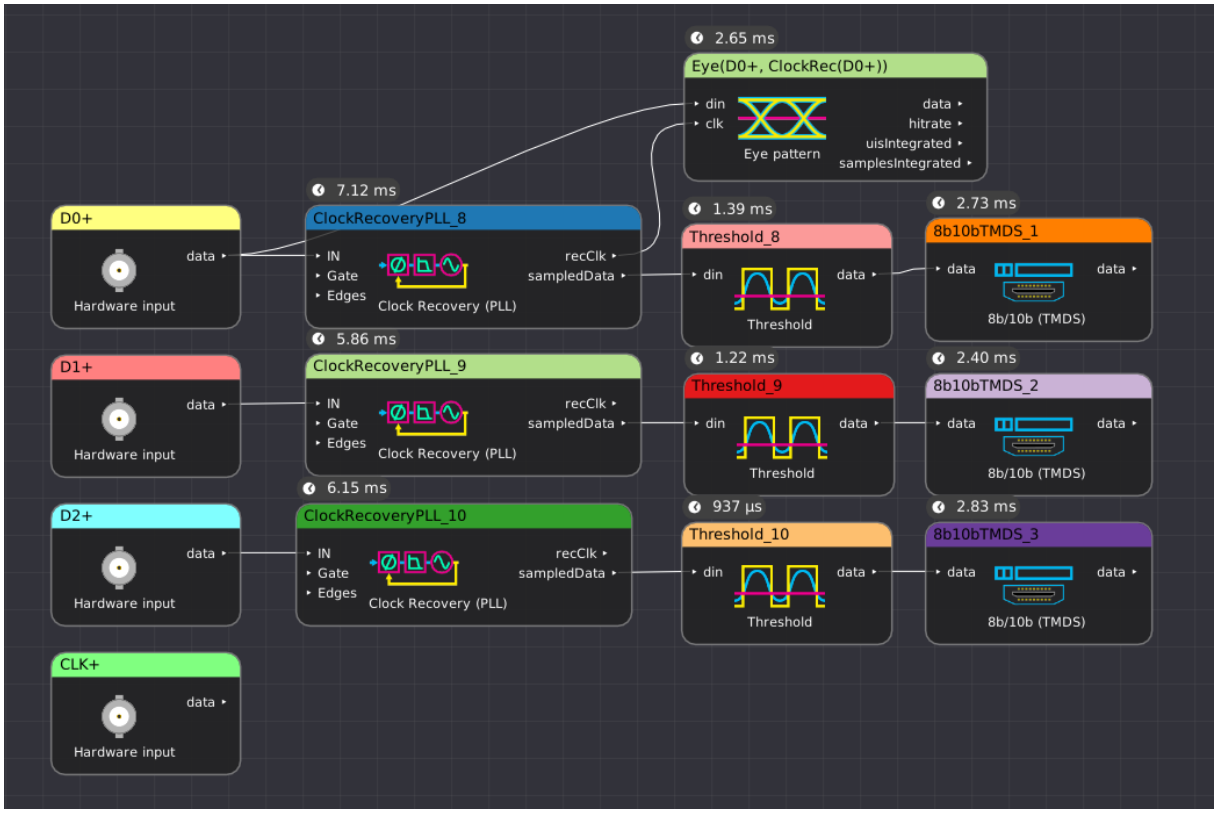



Figure 26.10: Example filter graph decoding TMDS from a single-ended input. Note that this example recovers the clock from the input signal rather than multiplying up the reference clock.

### 26.7.3 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.

| Stream name | Type | Description |
|---|---|---|
| data | Sparse protocol | Output decode |

| 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 |

### 26.7.4 Breaking Changes

This section lists major interface changes that may require projects to be updated.

- **v0.2:** This filter no longer needs the clock input that was required in v0.1. Instead, threshold the sampledData output of the Clock Recovery (PLL) and pass this to the TMDS decode, as shown in Fig. 26.10.

## 26.8 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.

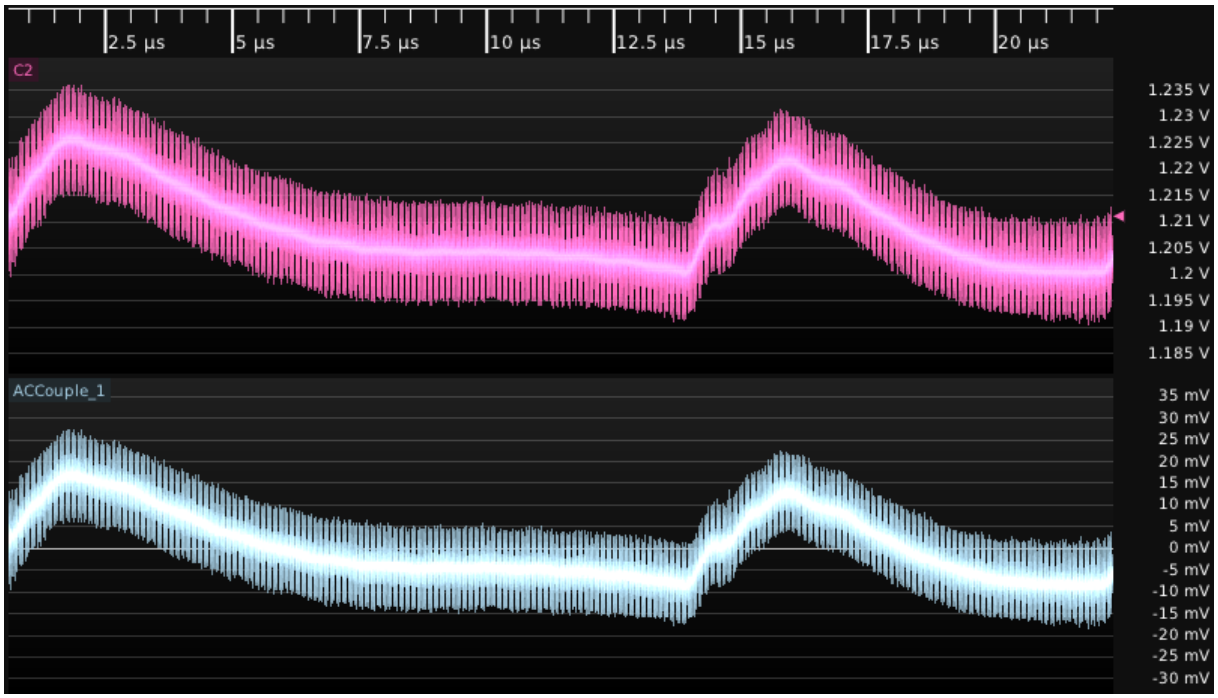



Figure 26.11: Example input and output of the AC Couple filter

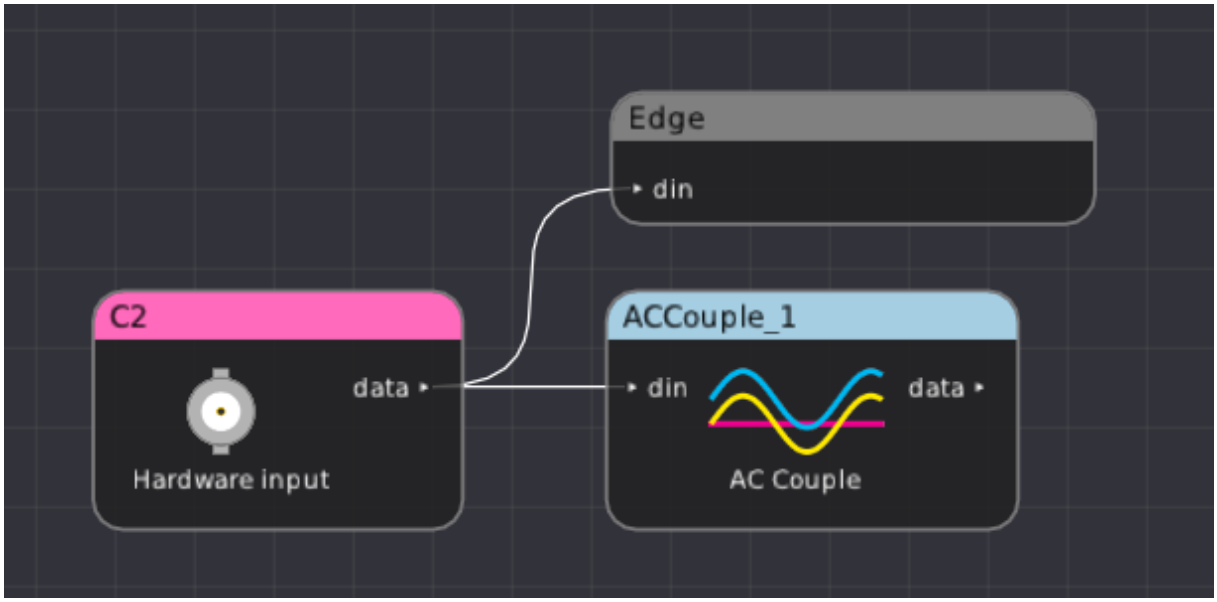



Figure 26.12: Example filter graph AC coupling an input waveform

### 26.8.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| din | Analog | Input waveform |

### 26.8.2 Parameters

This filter takes no parameters.

### 26.8.3 Output Signal

This filter outputs an analog waveform with identical configuration (sparse or uniform) and sample rate to the input, vertically shifted to center the signal at zero volts.

| Stream name | Type | Description |
|---|---|---|
| data | Analog | Output decode |

## 26.9 AC RMS

Measures the Root Mean Square amplitude of the waveform after removing any DC offset. The DC offset is calculated by averaging all samples in the waveform.

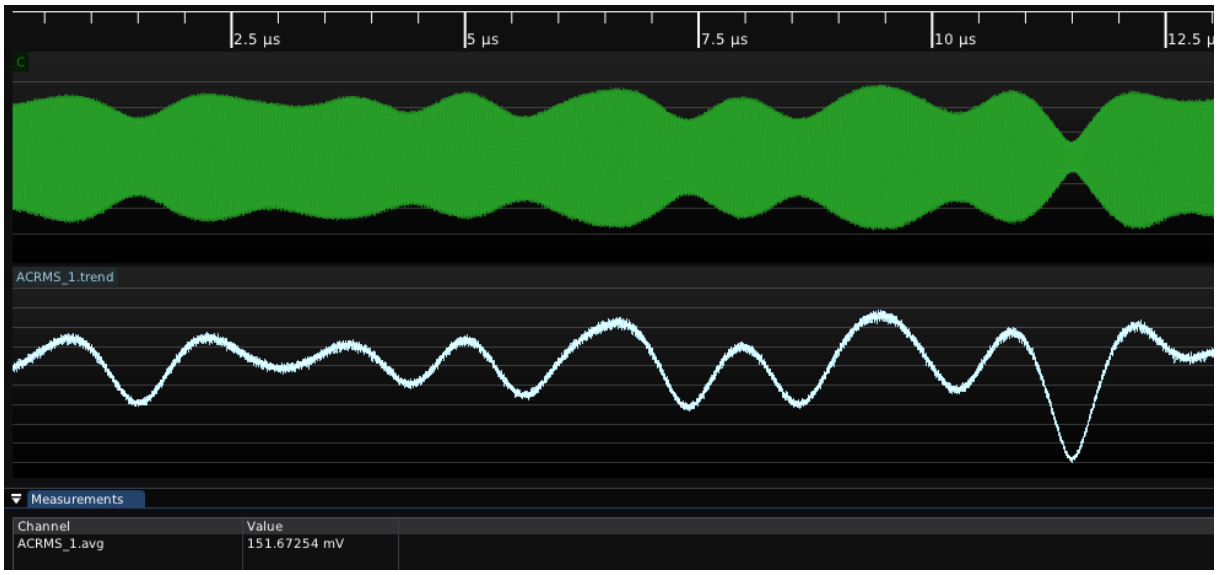



Figure 26.13: Example usage of the AC RMS filter on a QAM modulated signal

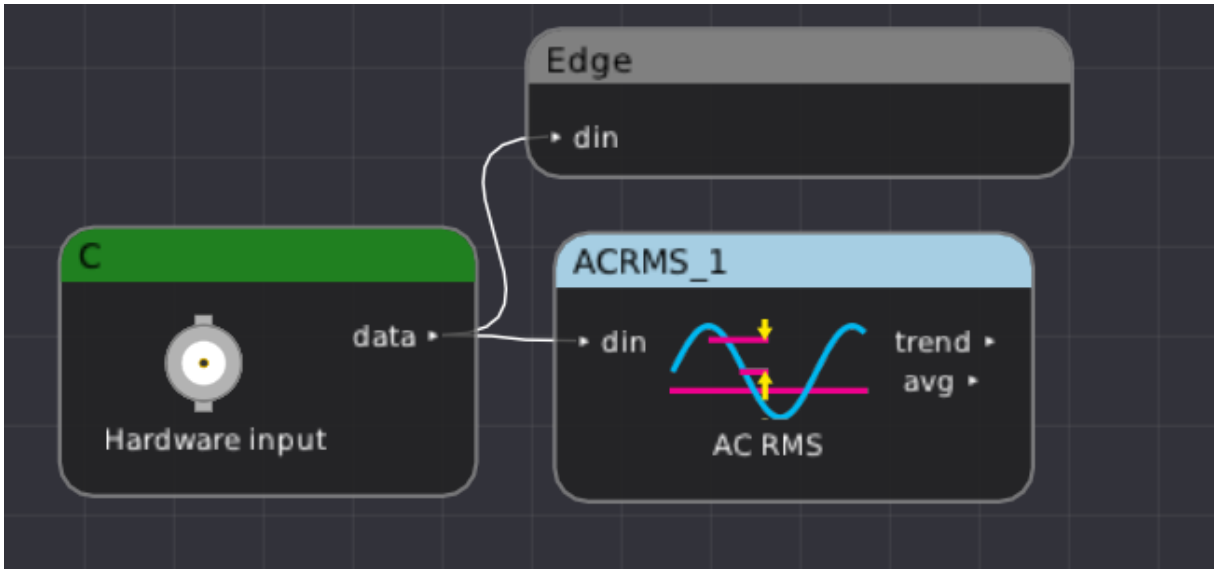



Figure 26.14: Example filter graph measuring RMS value of a waveform

### 26.9.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| din | Analog | Input waveform |

### 26.9.2 Parameters

This filter takes no parameters.

### 26.9.3 Output Signal

This filter has two output streams.

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

## 26.10 Add

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

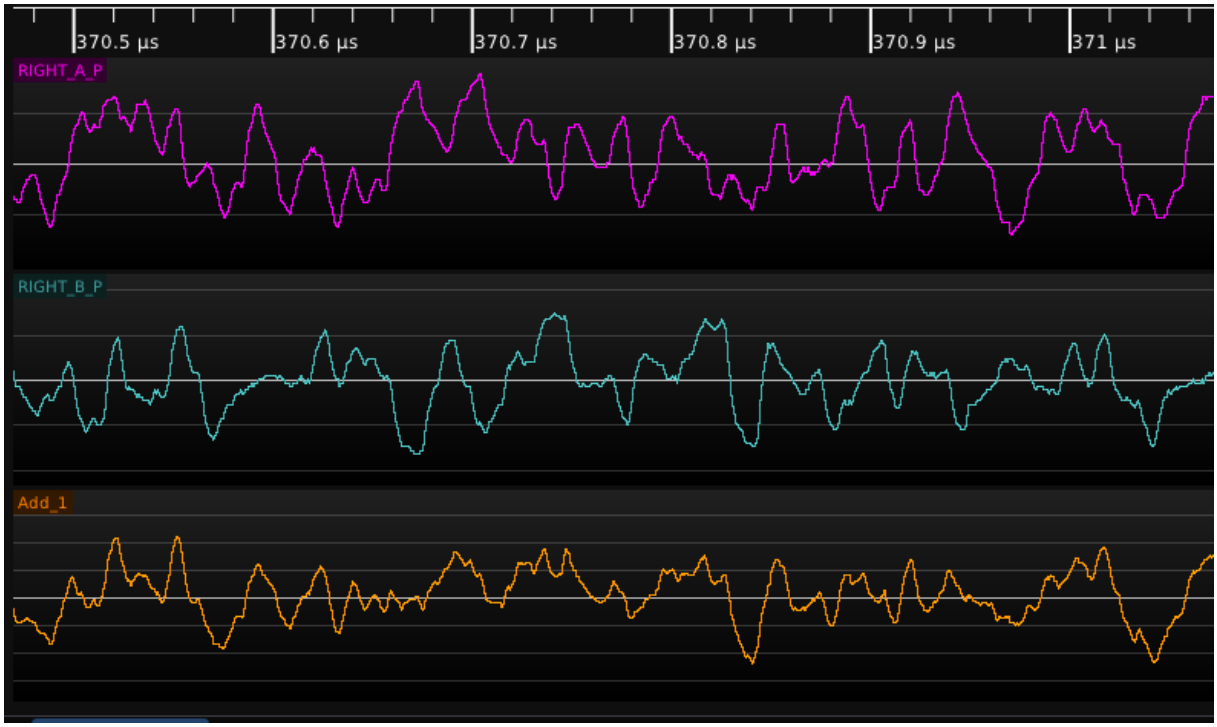



Figure 26.15: Example usage of adding two analog waveforms

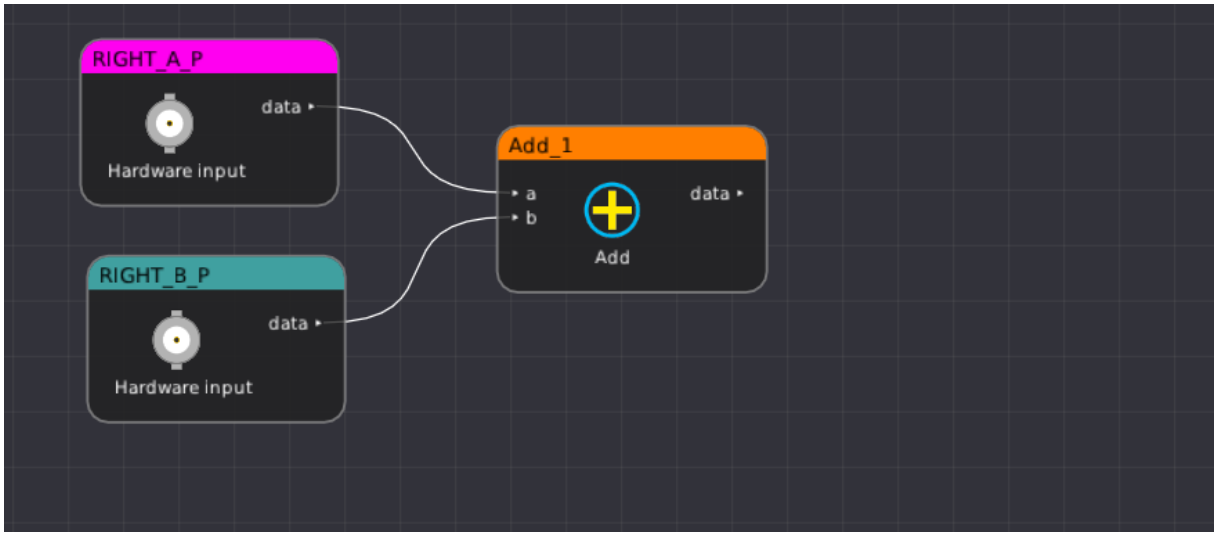



Figure 26.16: Example filter graph adding two analog waveforms

### 26.10.1 Inputs

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

### 26.10.2 Parameters

This filter takes no parameters.

### 26.10.3 Output Signal

If both inputs are vectors, this filter outputs a waveform containing the pairwise sum; i.e. sample $i$ of the output is $a[i] + b[i]$. No resampling is performed on the inputs so incorrect or unexpected results may occur if they do not share the same timebase. Both inputs must be the same type (both sparse or both uniform), mixing sparse and uniform (even if the sample timestamps are the same) is not allowed.

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

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

| Stream name | Type | Description |
|---|---|---|
| data | Analog | One sample per cycle of the input waveform containing the sum of the a and b inputs at that time |

## 26.11 ADL5205

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

TODO: Screenshot

### 26.11.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| spi | SPI bus | The SPI data bus |

### 26.11.2 Parameters

This filter takes no parameters.

### 26.11.3 Output Signal

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

## 26.12 APB

Decodes the AMBA APB bus, commonly used for connecting low-speed peripherals such as timers and serial ports to CPU cores in microcontrollers and SoCs. APB is a parallel bus typically only used in on-chip applications, so this filter is normally used with an internal logic analyzer core recording on-die signals to RAM and then reading out via JTAG or other debug protocol, or for analyzing VCD dumps from RTL simulation.

**NOTE**: The following APBv4/5 features are not yet supported by this decode:

- Byte write strobes
- Bus errors
- Address or data buses larger than 32 bits in size
- ID / user tag data
- Wakeup signaling

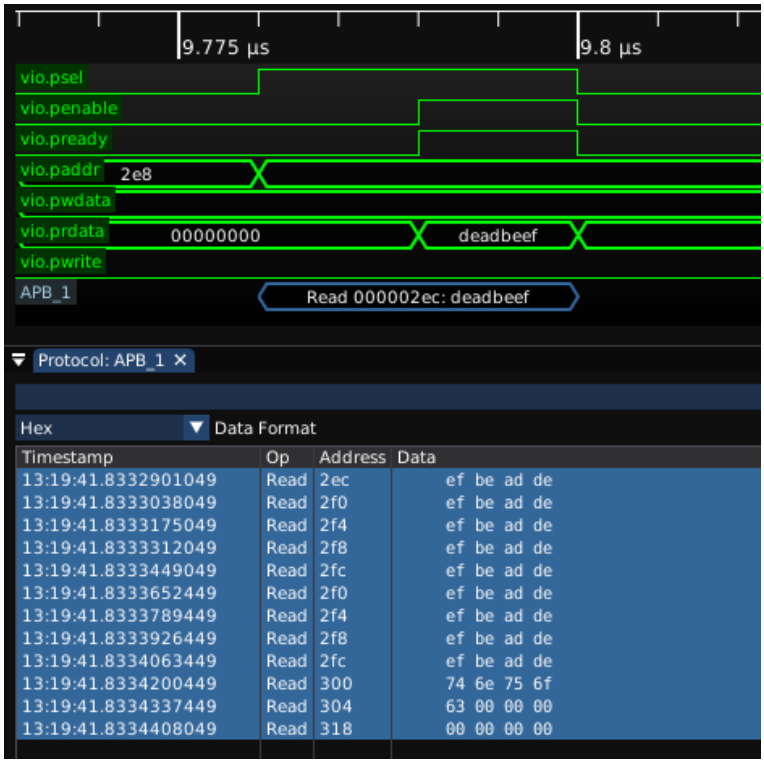



Figure 26.17: Example APB decode

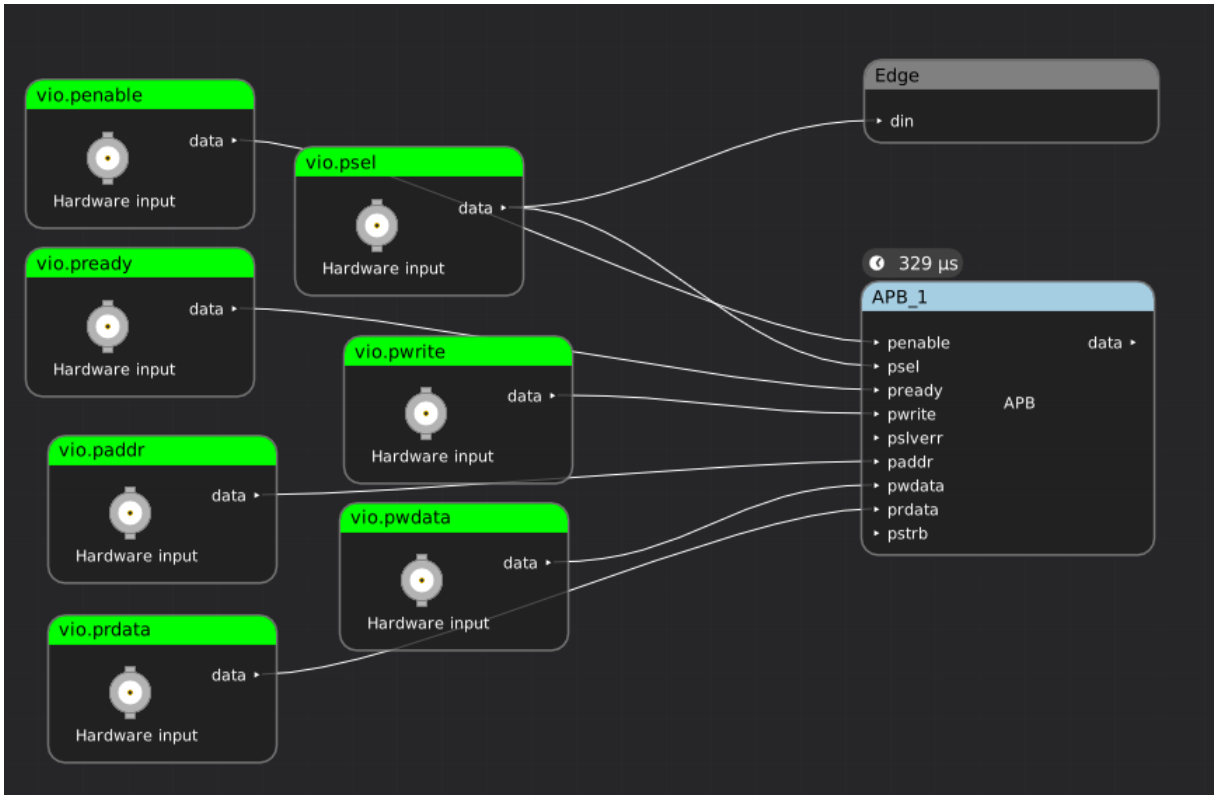



Figure 26.18: Example filter graph showing usage of APB decode

### 26.12.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| penable | 1-bit digital | APB enable |
| psel | 1-bit digital | APB select |
| pready | 1-bit digital | APB ready |
| pwrite | 1-bit digital | APB write enable |
| pslverr | 1-bit digital | APB error indication (optional) |
| paddr | 1-32 bit digital | APB address |
| pwdata | 1-32 bit digital | APB write data |
| prdata | 1-32 bit digital | APB read data |
| pstrb | 1-4 bit digital | APB byte strobe (optional) |

### 26.12.2 Parameters

This filter takes no parameters.

### 26.12.3 Output Signal

This filter outputs a time series of APB sample objects, each consisting of a read/write flag, an address, and a data value. Additional data will be added in the future once decoding of the full APBv5 protocol is implemented.

## 26.13 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 (“true area"). The negative area can also be considered as positive (“absolute area"). The measurement can be performed on the full record or on each cycle.

The following examples demonstrate usage of the filter on a sinusoid.

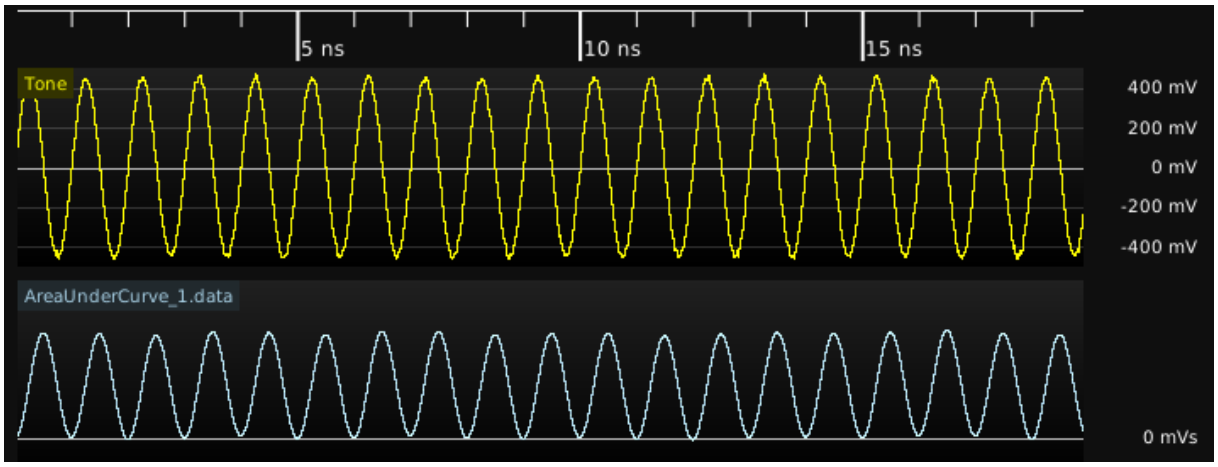



Figure 26.19: True area full record measurement. In this mode, the filter outputs $\int_0^i x$ at sample $i$, so we get $\int sin(x) = cos(x)$.

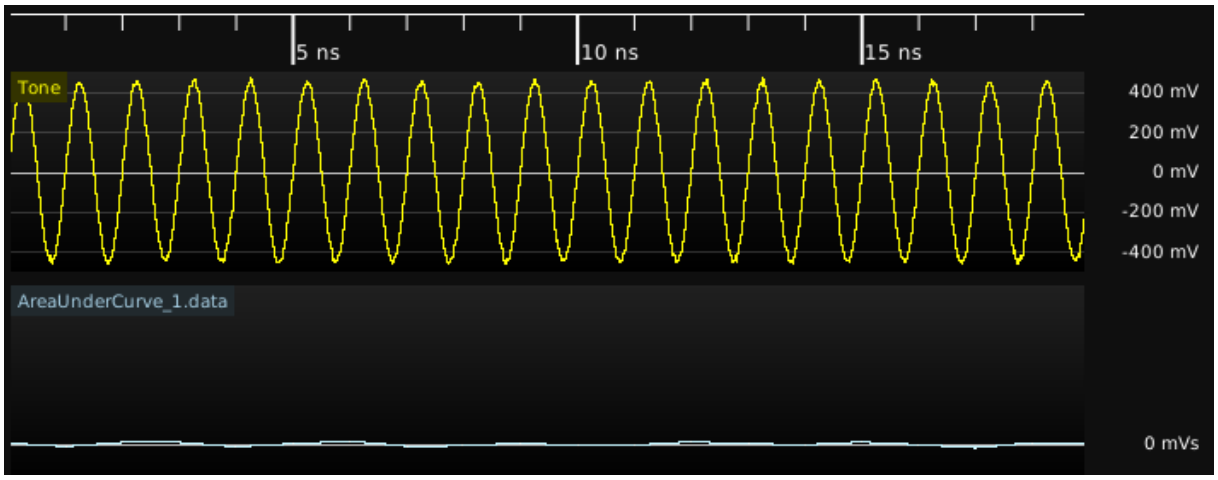



Figure 26.20: True area per cycle measurement. In this mode, the filter outputs $\int_{t1}^{t2} x$ at each zero crossing bounded interval so we get $\int_0^{2\pi} sin(x) = 0$ (plus measurement noise).

### 26.13.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| din | Analog | Input waveform |

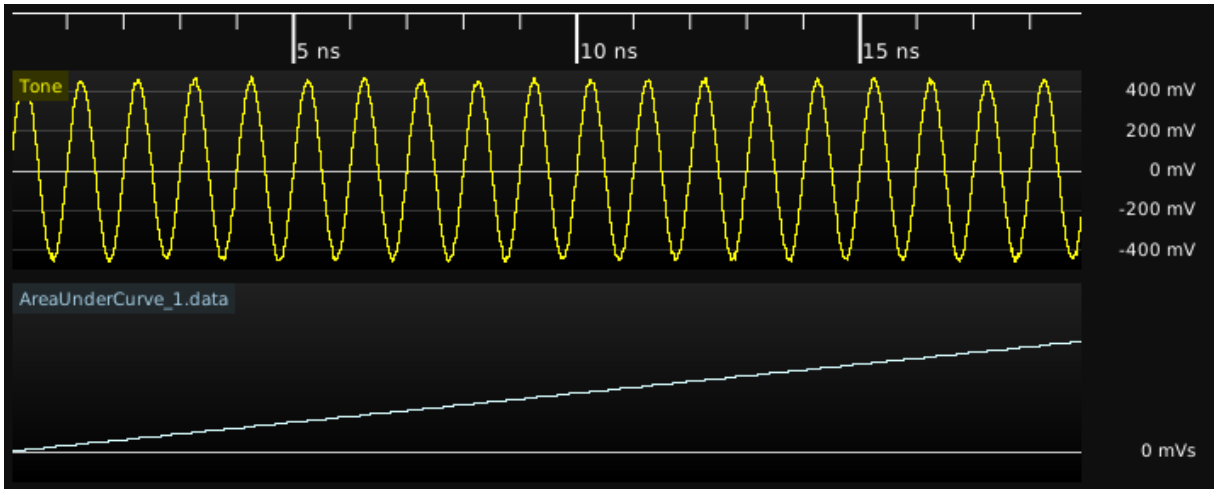



Figure 26.21: Absolute area full record measurement. In this mode, the filter outputs $\int_0^i |x|$ at sample $i$, so we get a continually increasing signal.

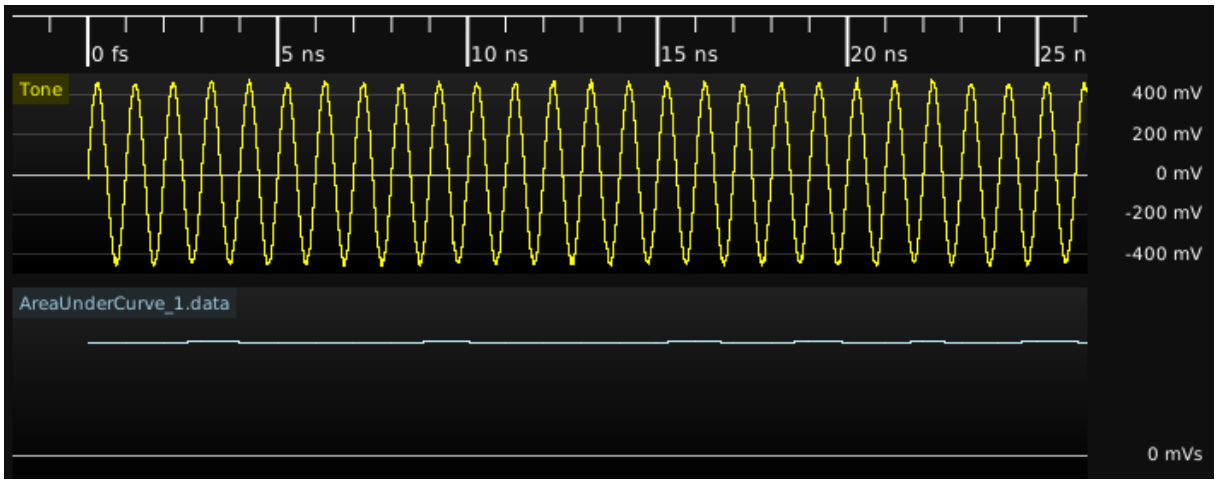



Figure 26.22: Absolute area per cycle measurement. In this mode, the filter outputs $\int_{t1}^{t2} |x|$ at each zero crossing bounded interval, so we get $\int 0^{2\pi} |sin(x)| = 4$ (plus measurement noise).

### 26.13.2 Parameters

| Parameter name | Type | Description |
|---|---|---|
| Measurement Type | Enum | **Full Record**: Measure the area of entire waveform<br>**Per Cycle**: Measure the area of each cycle in the waveform |
| Area Type | Enum | **True Area**: Consider area below ground as negative (integrate X)<br>**Absolute Area**: Consider area below ground as positive (integrate $\|X\|$) |

### 26.13.3 Output Signal

For full record measurement, this filter outputs a waveform indicating total area measured till the time on the waveform (integrating from zero to the current sample).

For per cycle measurement, this filter outputs a waveform representing area of each cycle (inte-

grating between zero crossings).

## 26.14 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. 26.23, 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 is proportional to offset range multiplied by waveform length. A GPU implementation of this filter is planned for the future (scopehal:998) which will significantly increase speed.

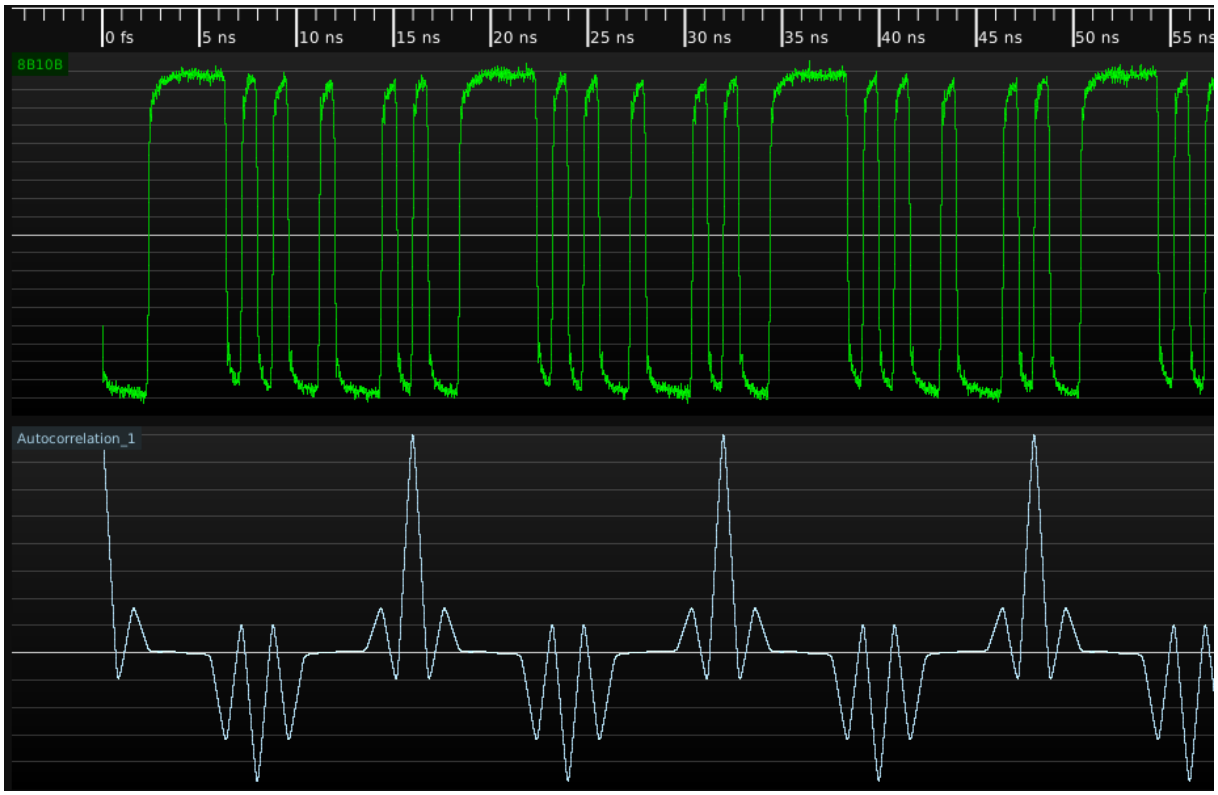



Figure 26.23: Example waveforms showing autocorrelation of an 8b/10b signal

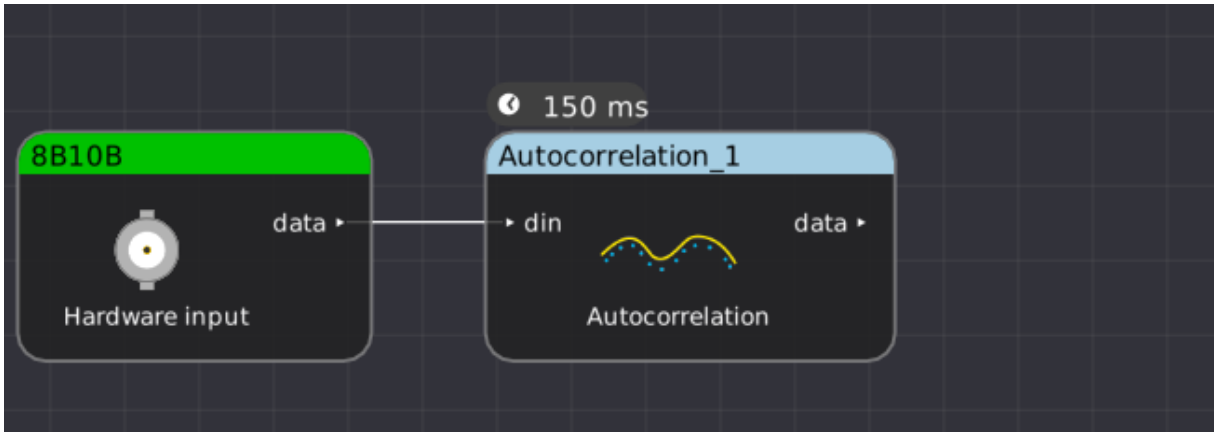



Figure 26.24: Example filter graph showing usage of autocorrelation filter

### 26.14.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| din | Uniform analog | Input waveform |

### 26.14.2 Parameters

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

### 26.14.3 Output Signal

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

| Stream name | Type | Description |
|---|---|---|
| data | Uniform analog | Autocorrelation waveform |

## 26.15 Average

This filter calculates the average of its input.

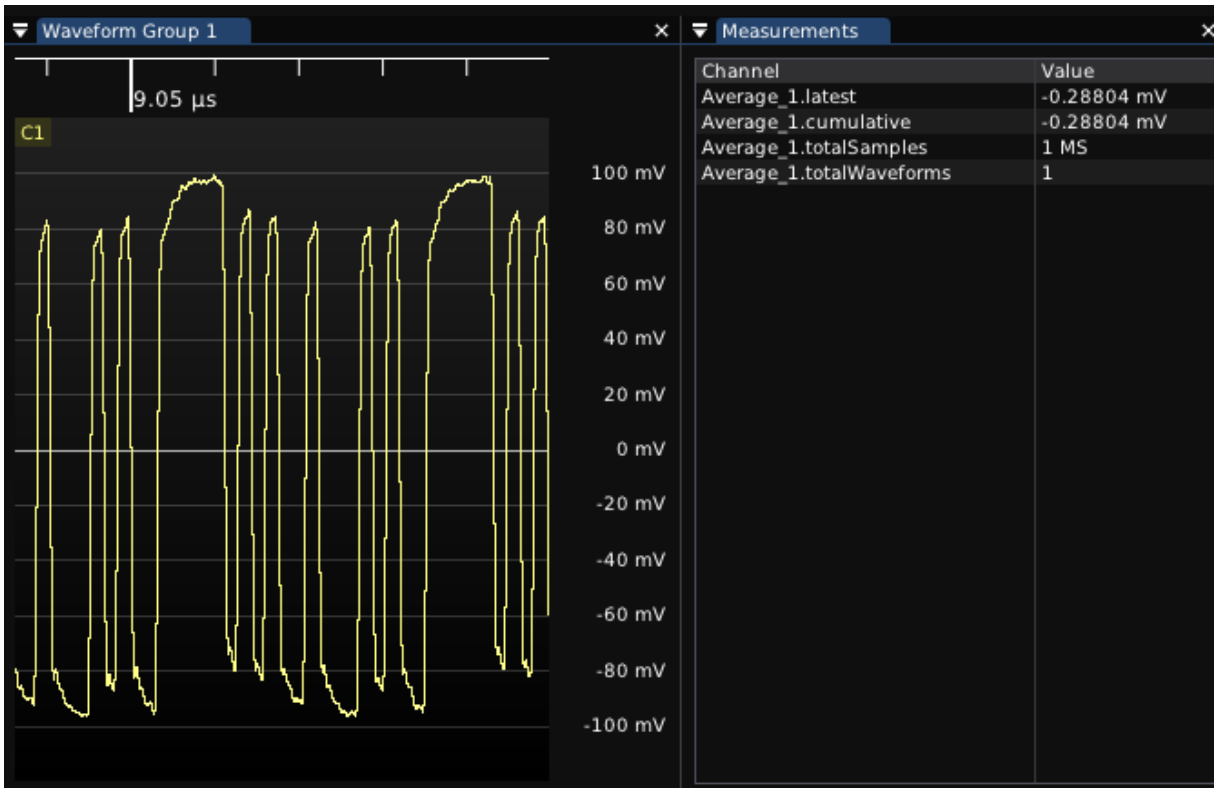



Figure 26.25: Typical usage of average filter

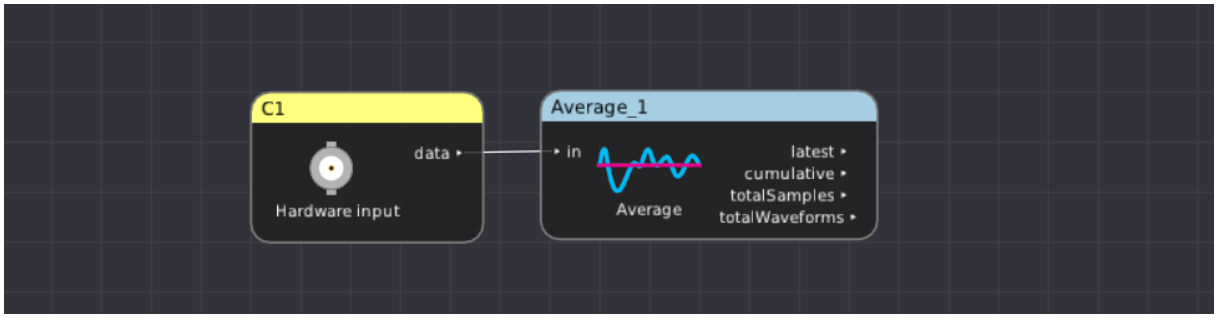



Figure 26.26: Example filter graph showing usage of average filter

### 26.15.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| in | Analog or scalar | Input waveform |

### 26.15.2 Parameters

This filter takes no parameters.

### 26.15.3 Output Signal

| Signal name | Type | Description |
|---|---|---|
| latest | Scalar | Average of the filter's current input |
| cumulative | Scalar | Average of all input since the last clear-sweeps |
| totalSamples | Scalar | Total number of integrated samples |
| totalWaveforms | Scalar | Total number of integrated waveforms |

## 26.16 Bandwidth

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

The bandwidth is measured relative to a user-specified reference level (typically the insertion loss at DC or the low end of the passband); for example the bandwidth of a 20 dB attenuator can be measured by setting the reference level to -20 dB.

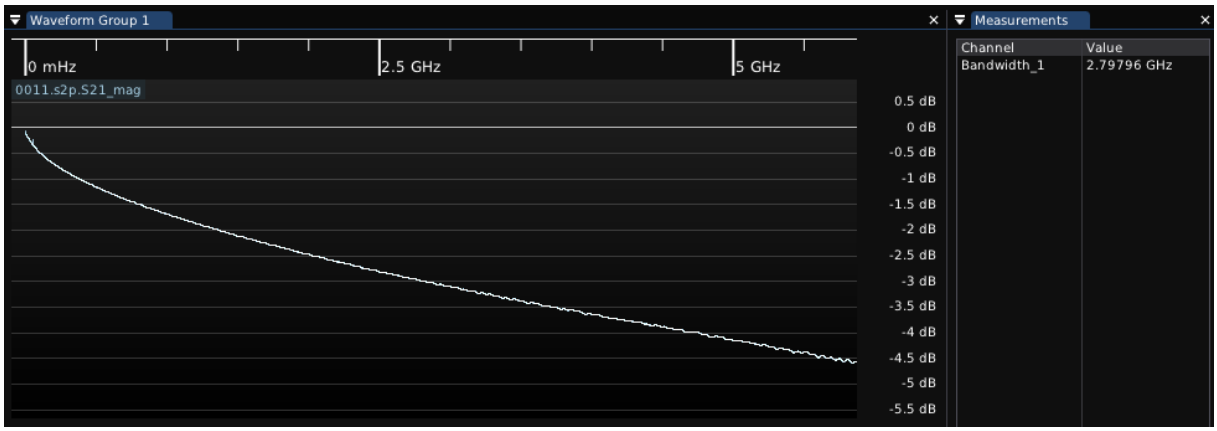



Figure 26.27: Measuring the -3 dB bandwidth of a cable

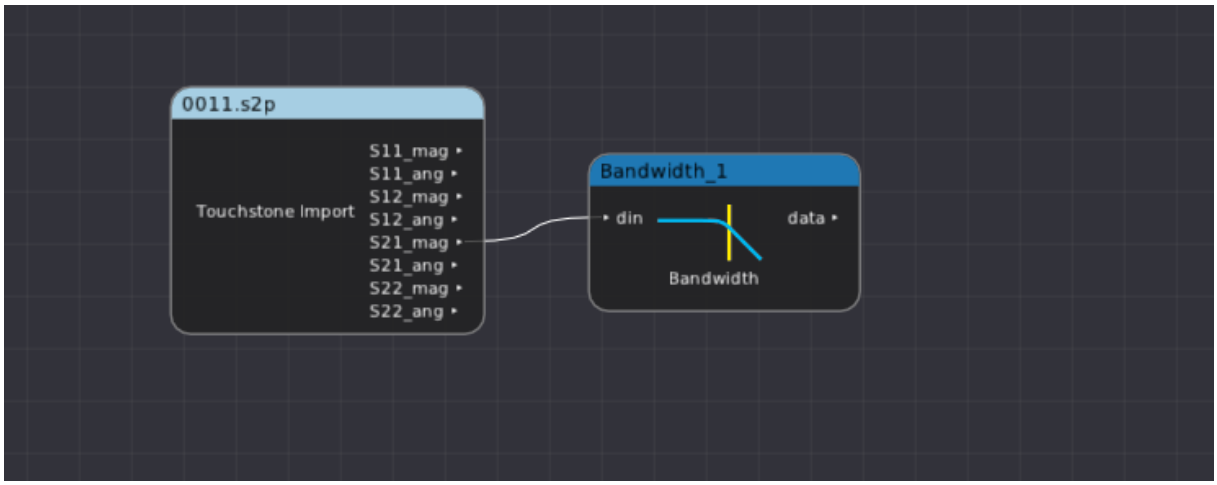



Figure 26.28: Example filter graph showing usage of bandwidth filter on an imported Touchstone file

### 26.16.1 Inputs

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

### 26.16.2 Parameters

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

### 26.16.3 Output Signal

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

| Signal name | Type | Description |
|---|---|---|
| data | Scalar | Calculated bandwidth |

## 26.17 Base

Calculates the base (average logical zero level) of a digital waveform.

It is most commonly used as a scalar to view the base of the entire waveform. At times, however, it may be useful to view the base waveform. For example, in Fig. 26.29, the vertical eye closure caused by channel ISI is readily apparent.

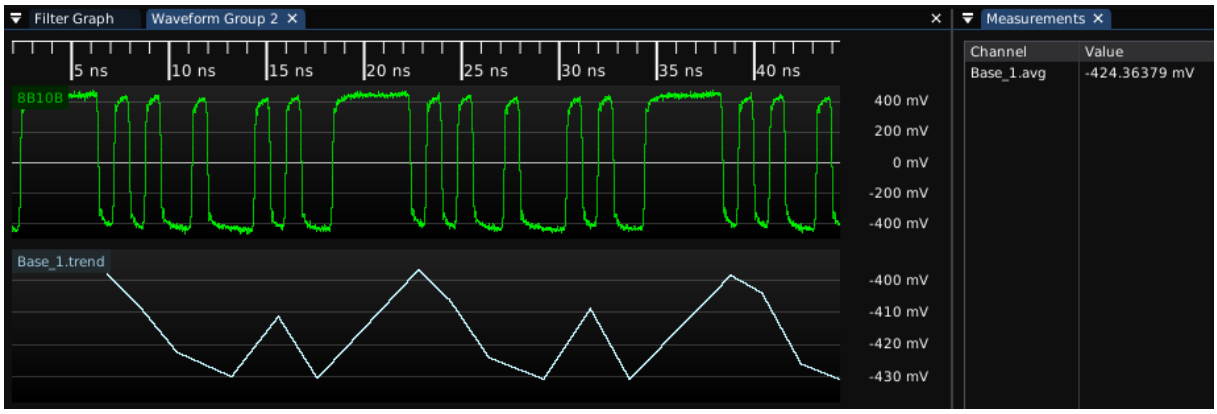



Figure 26.29: Example of base measurement on a serial data stream

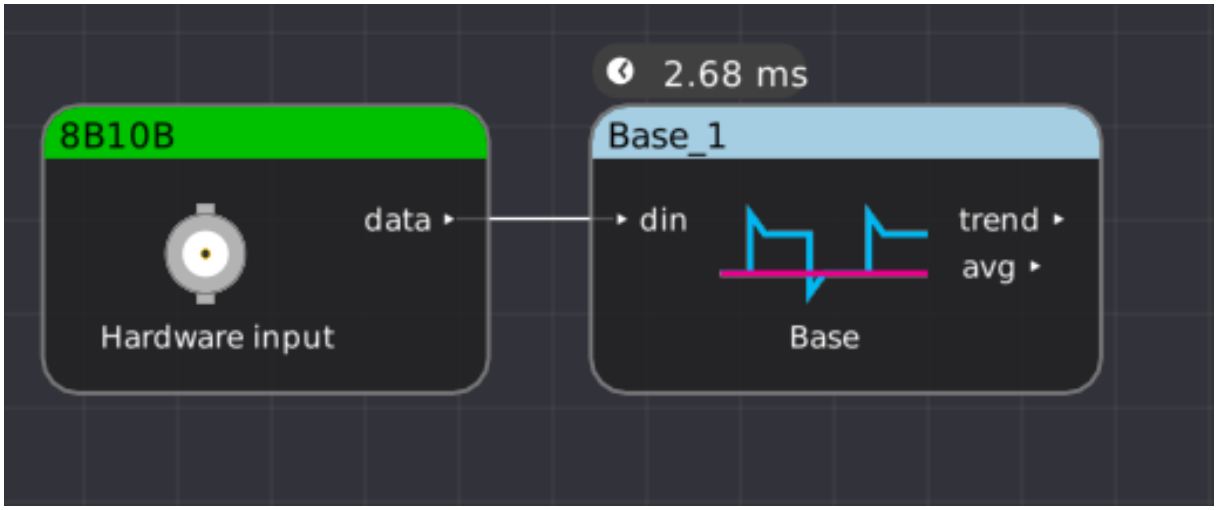



Figure 26.30: Example filter graph showing usage of base filter

### 26.17.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| din | Analog | Input waveform |

### 26.17.2 Parameters

This filter takes no parameters.

### 26.17.3 Output Signal

| Signal name | Type | Description |
|---|---|---|
| trend | Analog | Cycle-by-cycle base |
| avg | Scalar | Average base of the entire waveform |

## 26.18 BIN Import

Loads an Agilent / Keysight / Rigol binary waveform file.

### 26.18.1 Inputs

This filter takes no inputs.

### 26.18.2 Parameters

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

### 26.18.3 Output Signal

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

NOTE: Waveform files generated by the Rigol MSO5000 and possibly other scopes using a similar firmware stack are known to be malformed and will not load properly. We are exploring ways to repair and load these damaged files in the future (scopehal:975).

## 26.19 Burst Width

Measures the burst width of each burst in a waveform. A burst is a sequence of adjacent crossings of the average value of the waveform (for analog waveforms) or toggles (for digital waveforms).

The 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.

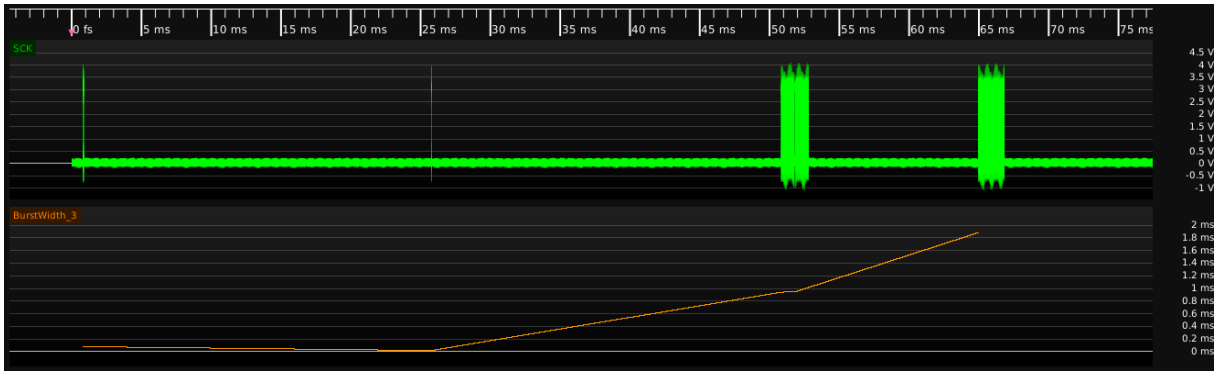



Figure 26.31: Example of burst width measurement

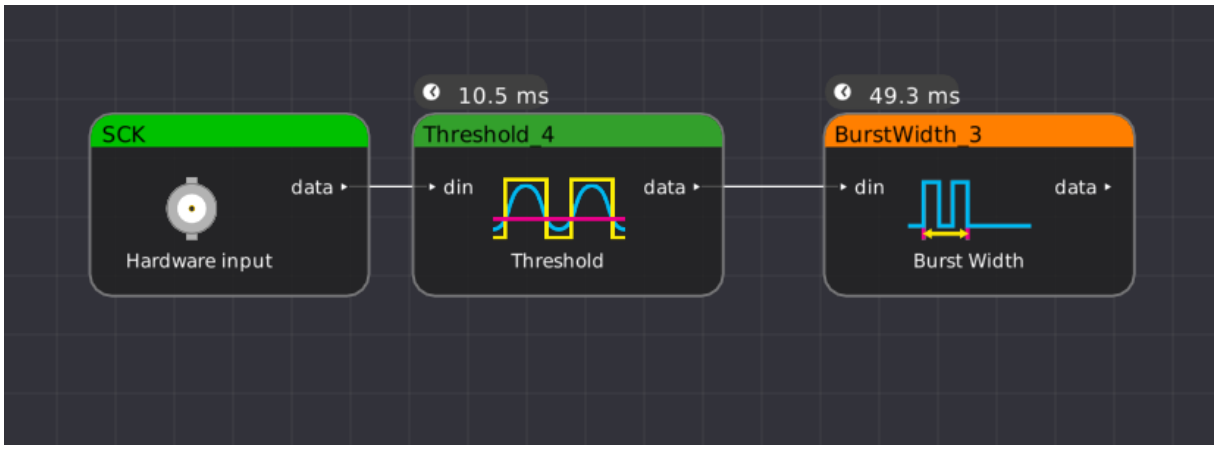



Figure 26.32: Example filter graph showing usage of burst width filter

### 26.19.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| din | Analog | Input waveform |

### 26.19.2 Parameters

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

### 26.19.3 Output Signal

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

## 26.20 Bus Heatmap

Computes a “spectrogram" visualization of bus activity with address on the Y axis and time on the X axis, in order to identify patterns in memory or bus activity.

The current version only supports CAN bus however other common memory interfaces will be added in the future.

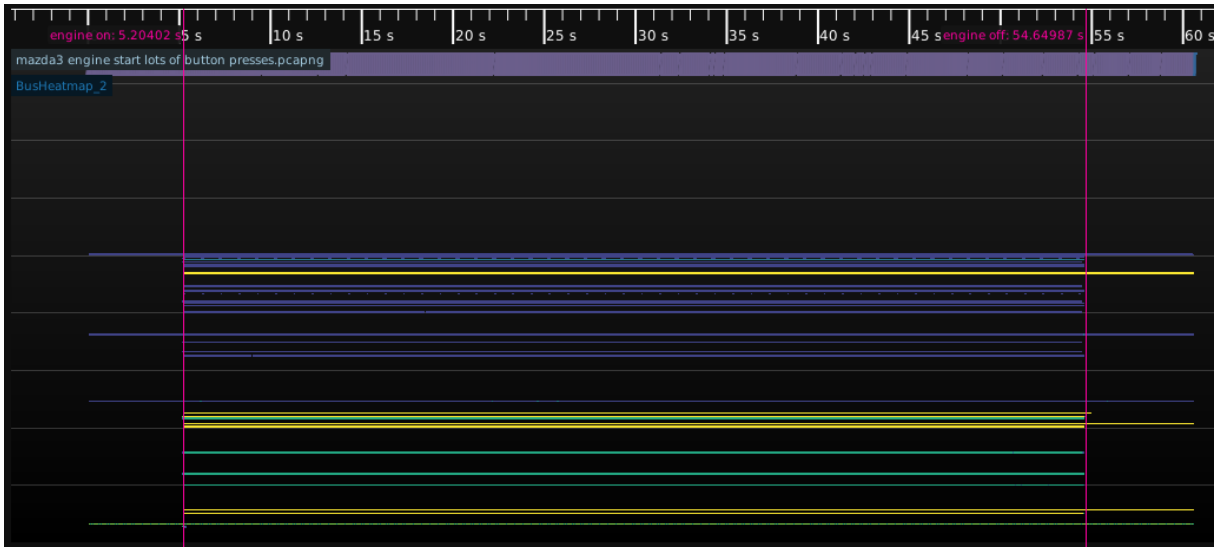



Figure 26.33: CAN bus activity on a car’s OBD port showing the vehicle being started, running for 50 seconds, then shutting down

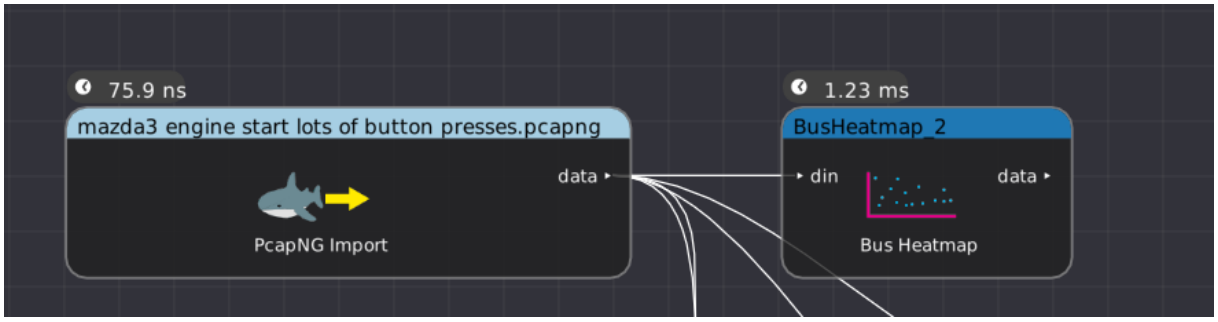



Figure 26.34: Example filter graph showing usage of bus heatmap filter on an imported CAN bus capture

### 26.20.1 Parameters

| Parameter name | Type | Description |
|---|---|---|
| Max Address | Integer | Maximum address to display in the plot |
| X Bin Size | Integer | Width of each pixel in the X axis (timebase units) |
| Y Bin Size | Integer | Number of addresses to merge into each pixel in the Y axis |

### 26.20.2 Output Signal

This filter outputs a 2D density plot that is (max address) / (y bin size) pixels high and (memory depth) / (x bin size) pixels wide, spanning the entire duration of the input and the full address range requested.

All packets within the input waveform have the start time and address rounded to the closest bin in X and Y. The corresponding pixel in the integration buffer is incremented, then the final waveform is normalized to cover the full range of the selected color ramp.

| Signal name | Type | Description |
|---|---|---|
| data | Density map | Calculated heatmap |

## 26.21 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).

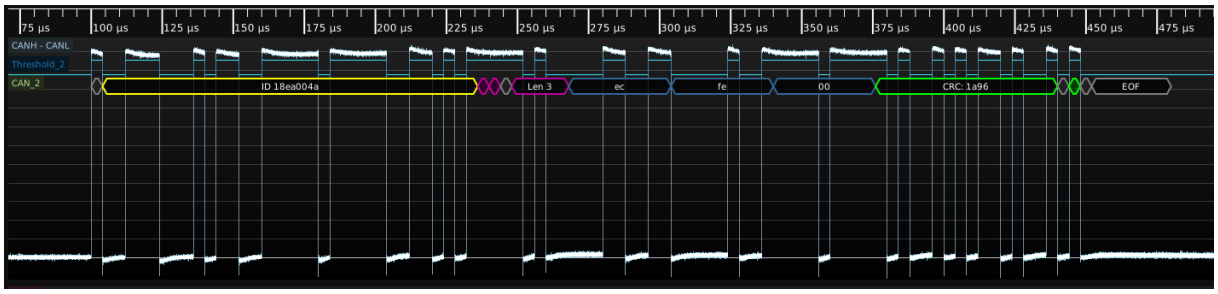



Figure 26.35: Example of decoding a single extended-format frame with 3 bytes of data

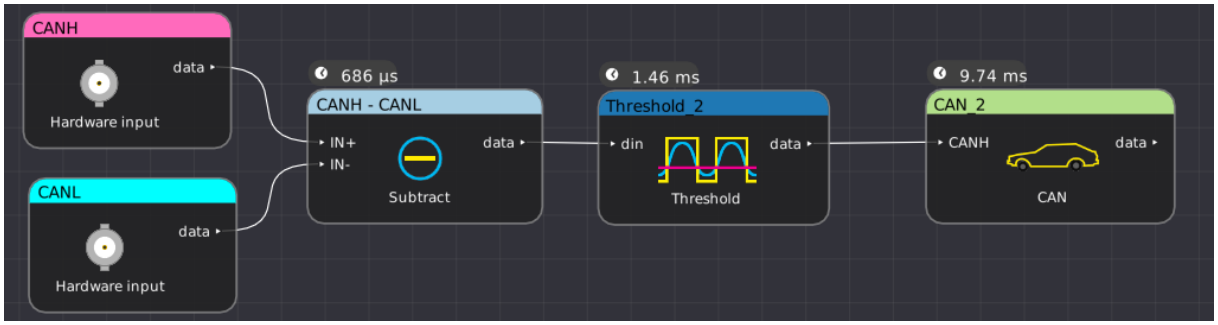



Figure 26.36: Example filter graph showing usage of CAN bus decode

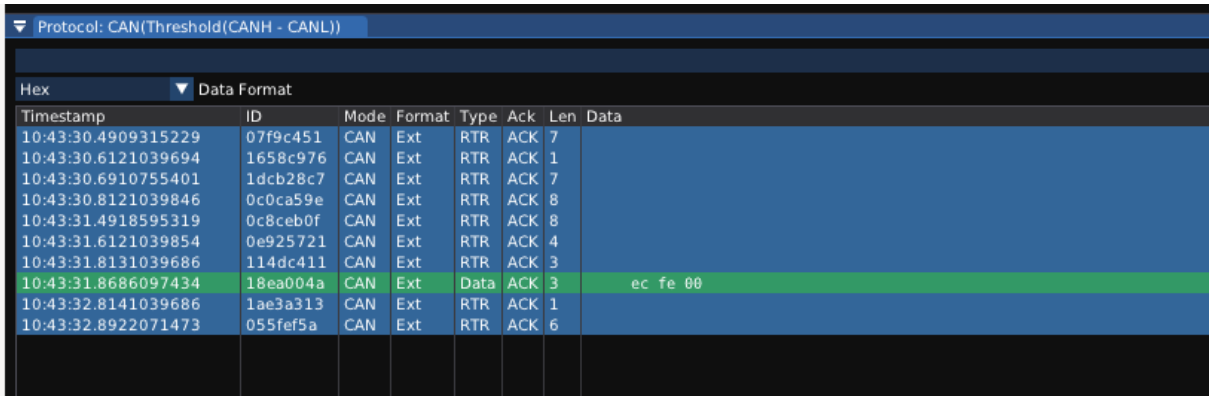



Figure 26.37: Example packet output

### 26.21.1 Inputs

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

### 26.21.2 Parameters

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

### 26.21.3 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.

| Signal name | Type | Description |
|---|---|---|
| data | Protocol | Decoded waveform 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 |

### 26.21.4 Protocol Analyzer

TODO

## 26.22 CAN Analyzer

This filter adds a protocol analyzer table to CAN data sources which do not natively provide one.

The most common use of this block is processing packets acquired from the socketcan driver, since the current libscopehal object model does not allow instruments to output protocol packets directly.

(This filter description is a stub and will be expanded in the future)

## 26.23 CAN Bitmask

Extracts a bit-masked value from a stream of CAN bus packets and outputs a Boolean waveform.

The typical use of this is to display a Boolean control or status signal, such as turn signal indicator state, as a waveform.

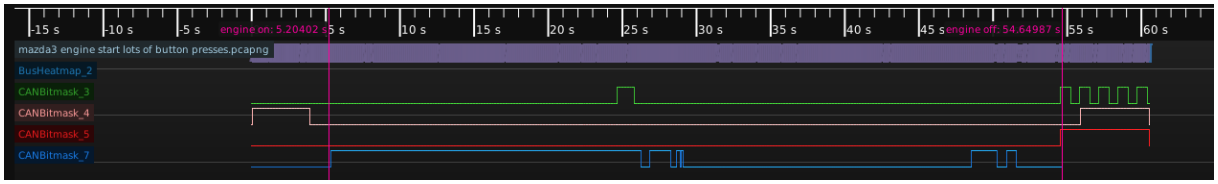



Figure 26.38: Example of decoding several bitfields from a CAN bus waveform

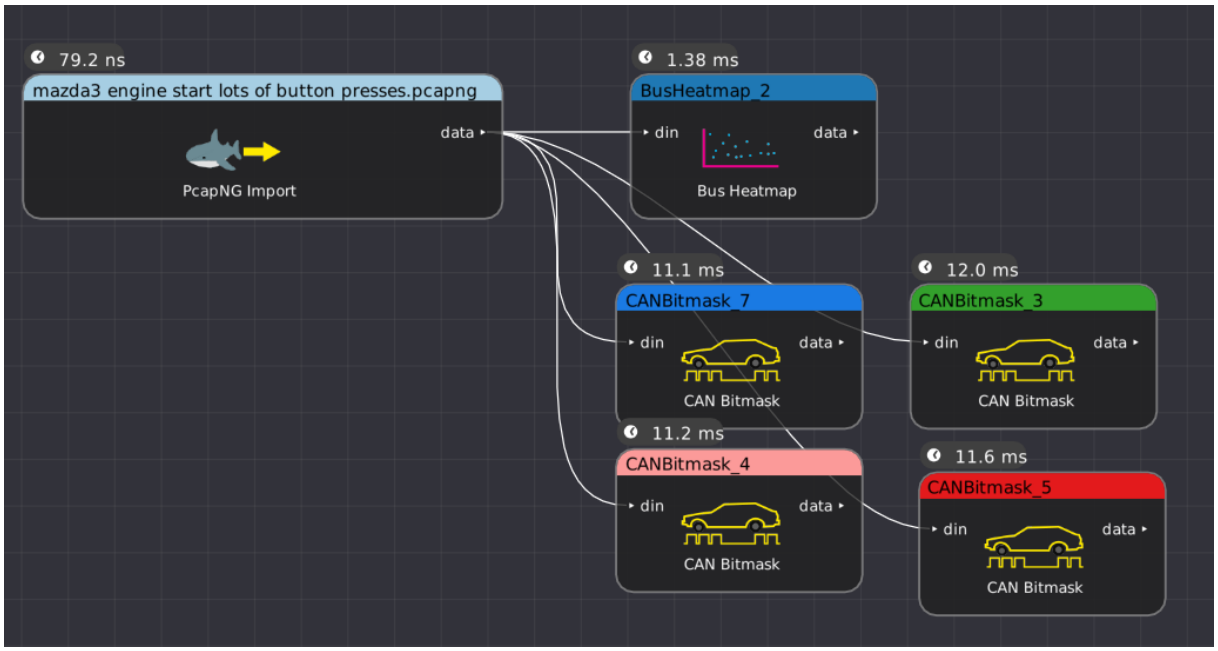



Figure 26.39: Example filter graph showing usage of CAN bitmask decode

### 26.23.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| din | CAN bus | Input waveform |

### 26.23.2 Parameters

| Parameter name | Type | Description |
|---|---|---|
| Bus Address | Hex | CAN ID to match |
| Initial Value | Bool | Starting value to display before the first matching CAN packet is seen |
| Pattern Bitmask | Hex | 64-bit mask for selecting bits within the CAN packet |
| Pattern Target | Hex | 64-bit pattern to look for |

### 26.23.3 Output Signal

This filter outputs a sparse digital waveform with one sample per CAN packet matching the target CAN ID.

If the packet value, interpreted as a 64-bit big endian integer, equals the target when ANDed with the bitmask, the output is 1. Otherwise the output is zero.

## 26.24 Can-Utils Import

Loads a log file generated by the `candump` utility from the Linux `can-utils` software package and displays it as a series of CAN packets.

Example capture command: `candump -l can0`

(This filter description is a stub and will be expanded in the future)

## 26.25 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. 26.40 shows the result of passing a 10.3125 Gbps PRBS through S21 of a 300mm FR-4 channel with insertion loss of approximately 7 dB at the fundamental. The ISI and phase shift introduced by the channel can be seen clearly.

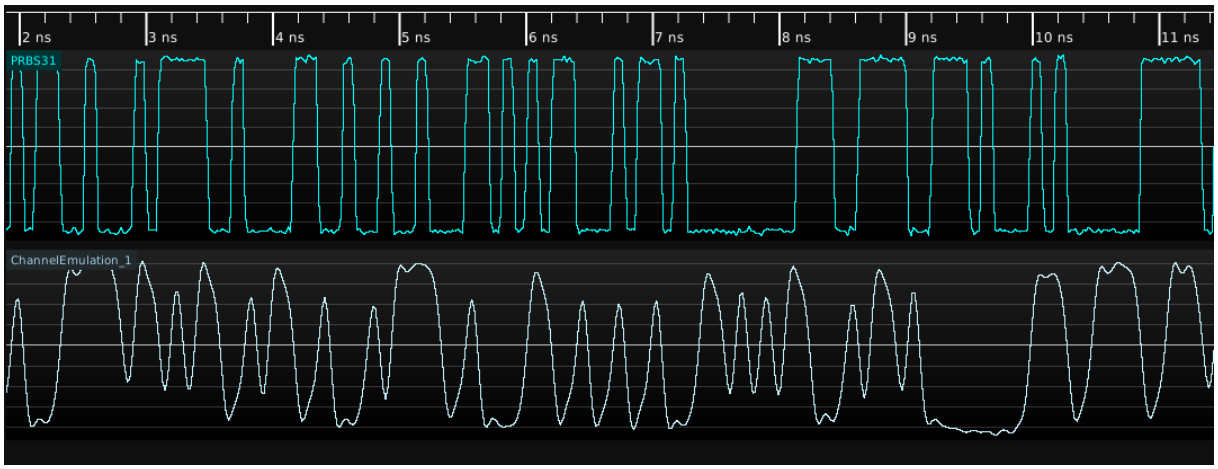



Figure 26.40: Example of channel emulation on a serial data stream

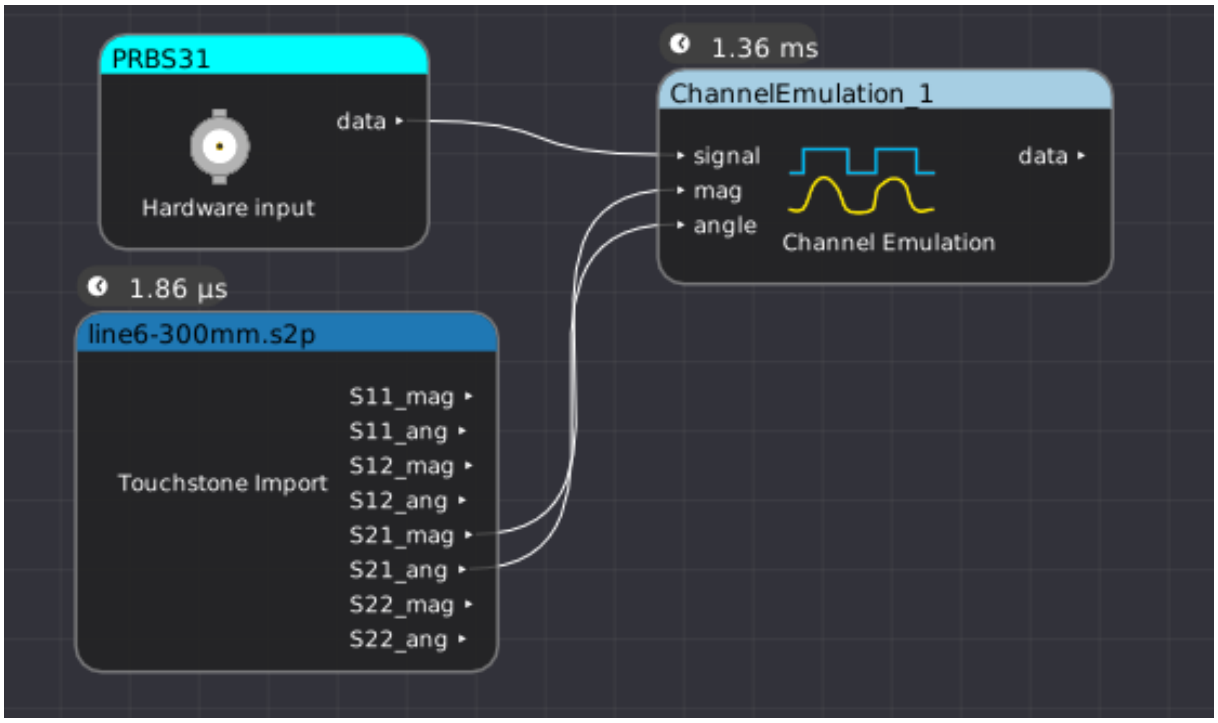



Figure 26.41: Example filter graph showing usage of channel emulation filter

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

The group delay of the channel is then estimated and samples are discarded from the beginning of the waveform to prevent causality violations. For example, when performing channel emulation using a network with a 1ns group delay, the output waveform will begin 1ns after the input (since

the channel output before this depends on input samples before the start of the waveform). Note that the automatic group delay estimation uses points from roughly the center of the S-parameter dataset in the current implementation; channels which do not have a significant passband around this frequency will give incorrect group delay estimates. The “Group Delay Truncation Mode" parameter can be set to manual in this case, selecting the “Group Delay Truncation" parameter instead of the automatically estimated value.

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

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

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

### 26.25.1 Inputs

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

### 26.25.2 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 |

### 26.25.3 Output Signal

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

## 26.26 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.

Clipping both top and bottom is not currently supported, however two instances of the filter can be chained to achieve the same result.

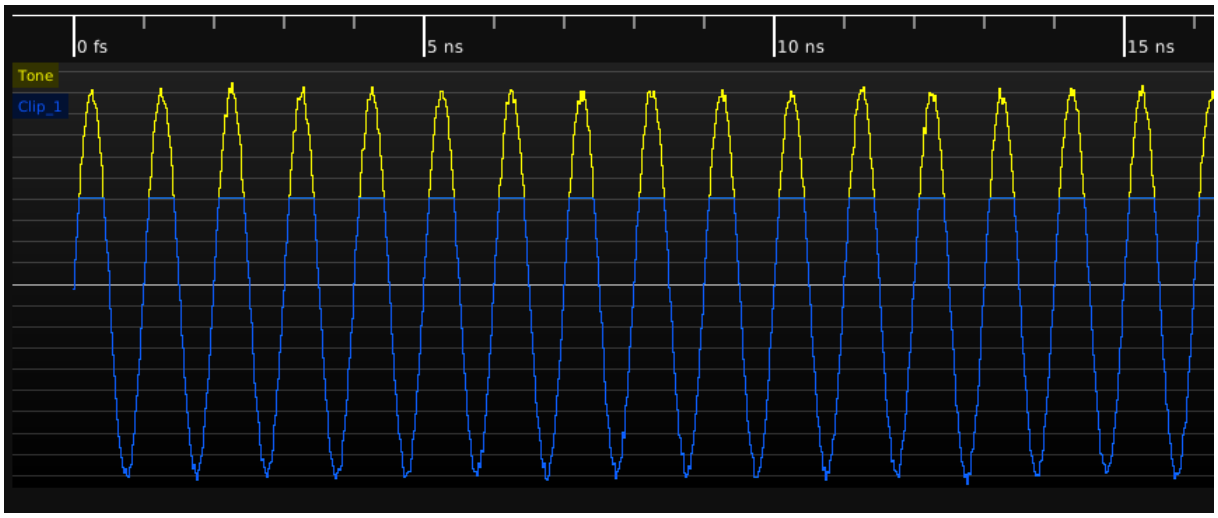



Figure 26.42: Example of clipping a sinewave

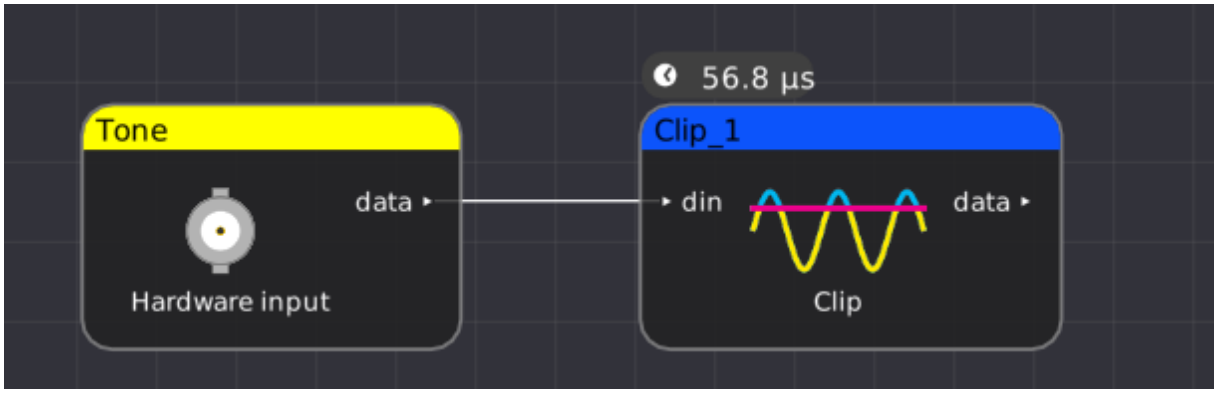



Figure 26.43: Example filter graph showing usage of clip filter

### 26.26.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| din | Analog | Input waveform |

### 26.26.2 Parameters

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

### 26.26.3 Output Signal

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

## 26.27 Clock Jitter (TIE)

Calculates the time interval error of a test data or clock signal with respect to an ideal “golden" clock (typically obtained from the Clock Recovery (PLL) block). The TIE can be either plotted directly to look for time-varying patterns in jitter (e.g. from transmit clock instability) or used as input to other filters to perform more complex jitter analysis.

Each time the test signal crosses the threshold level (if analog) or toggles (if digital) the corresponding edge in the golden clock is located and the difference in time between the two is reported as the time interval error for that sample.

**Notes:**

- If the PLL does not start in a fully locked state (e.g. if the test signal is slightly off the PLL's nominal frequency due to spread spectrum modulation) the TIE filter may report very large jitter values due to beating between the golden and recovered clocks, which can throw off jitter analysis. The “Skip Start" parameter allows truncating the beginning of the dataset to avoid this issue.
- By convention, recovered clocks in ngscopeclient have 90° phasing so the recovered clock edges are centered in the data eye and can be directly used to sample the data. The TIE filter applies an internal -90° phase shift to compensate, which must be accounted for when using a golden clock not generated by the Clock Recovery (PLL) block.

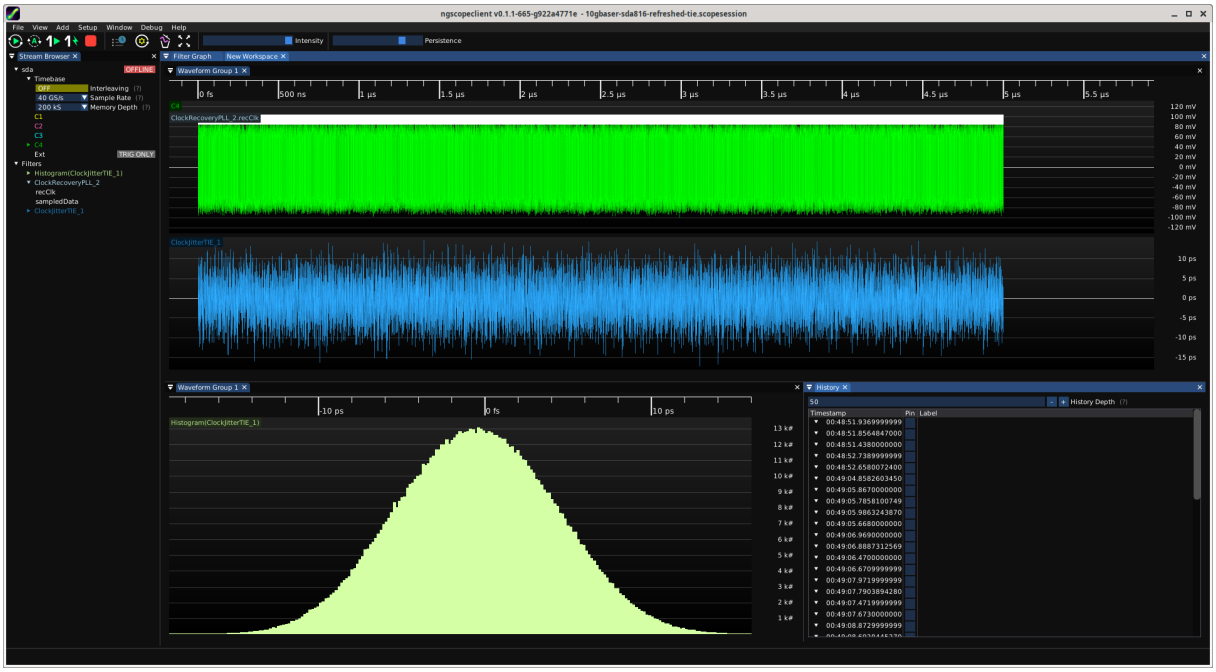


Figure 26.44: Example of measuring TIE and jitter histogram of a 10.3125 Gbps Ethernet signal

### 26.27.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| Clock | Analog or digital | Test signal |
| Golden | Sparse digital | Golden PLL reference output |

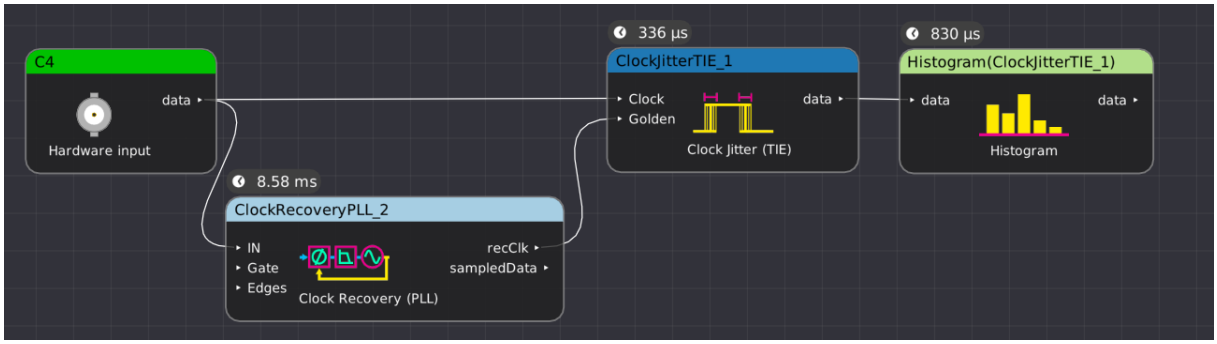



Figure 26.45: Example filter graph showing usage of TIE filter to display a jitter histogram

### 26.27.2 Parameters

| Signal name | Type | Description |
|---|---|---|
| Skip Start | Integer (s) | Ignore this much time after the start of the waveform before generating output |
| Threshold | Float (V) | Level to threshold analog inputs at. Ignored for digital inputs |

### 26.27.3 Output Signal

This filter outputs a sparse analog sample with one sample per level crossing or toggle of the test signal containing the calculated TIE value.

## 26.28 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.

(This filter description is a stub and will be expanded in the future)

## 26.29 Clock Recovery (PLL)

This filter uses a PLL to recover a clock from a serial NRZ 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. Additionally, a sampled data stream is provided containing data values at each recovered clock edge which may be fed directly to various protocol decode blocks.

When the optional clock gating input is low, the output does not toggle and any edges in the input signal are ignored. This is most commonly used as a squelch to disable decoding and eye processing in between bursts of data. As soon as the gate goes high, the PLL will instantly phase shift the internal NCO to align with the next transition in the input signal and then begin running closed-loop continuing from the previous NCO frequency value.

With analog inputs, this filter uses a single threshold suitable for NRZ inputs. Using either the upper or lower eye midpoint as a threshold will *work* on a MLT-3 signal (since the -1 to +1 transition is not allowed by the protocol) however high frequency jitter will increase slightly since the PLL is not able to track crossings of the lower eye in this configuration and will run open-loop a larger fraction of the time than normal.

Using a single threshold for a PAM signal is not recommended and will add data-dependent jitter (since the slew rate for, for example, a -2 to +2 transition is different than that for a -1 to +1 transition), potentially severe enough to completely close the eye. For proper results on PAM signals, use the PAM edge detector filter to find level crossings and then apply the CDR to the detected transitions. Using the PAM edge detector will also give better tracking of MLT-3 although it is not strictly necessary in this case.

The current implementation of this filter uses a simple bang-bang control loop which is reasonably fast and provides generally acceptable jitter transfer performance for most purposes (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.

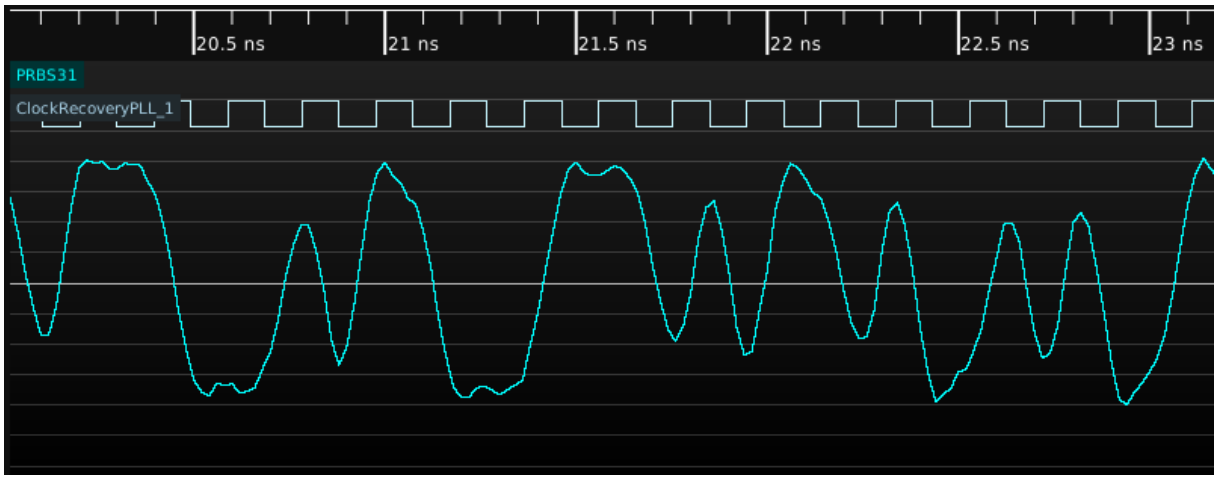



Figure 26.46: Example of CDR PLL on a serial data stream

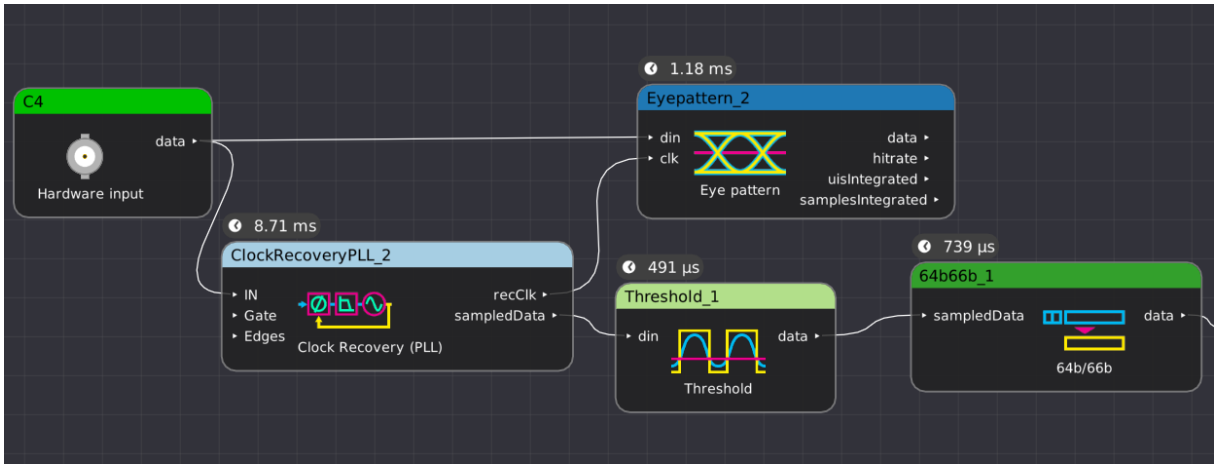



Figure 26.47: Example filter graph showing usage of CDR PLL filter with recovered clock used for eye pattern and sampled data for a protocol decode

### 26.29.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| IN | Analog or digital | Input waveform |
| Gate | Digital | Clock enable signal. Leave unconnected to disable gating. |
| Edges | Digital | Output of external edge detector filter (used for PAM4, not needed for NRZ) |

### 26.29.2 Parameters

| Parameter name | Type | Description |
|---|---|---|
| Symbol rate | Float | Symbol rate, in Hz |
| Threshold | Float | Decision threshold for the edge detector, in volts. Ignored for digital inputs. |

### 26.29.3 Output Signal

This filter outputs an digital waveform with one clock sample per transition of the recovered clock, and a sampled data waveform (analog or digital as appropriate) with one sample per transition of the recovered clock.

## 26.30 Clock Recovery (UART)

This filter uses a DLL to recover a sampling clock from UART or similar protocol at a known baud rate. The single-rate recovered clock idles low and toggles for each bit in each frame and is phased 90°with respect to data, such that each bit can be sampled on the rising edge of the DLL output clock. This filter can be used for generating an eye pattern of the serial signal.

The current implementation limits support to serial protocols with 10 bits/symbols per frame. Consider using the PLL-based clock recovery for unsupported serial formats if applicable.

The current implementation does not synchronize by aligning falling clock edges with symbol edges.

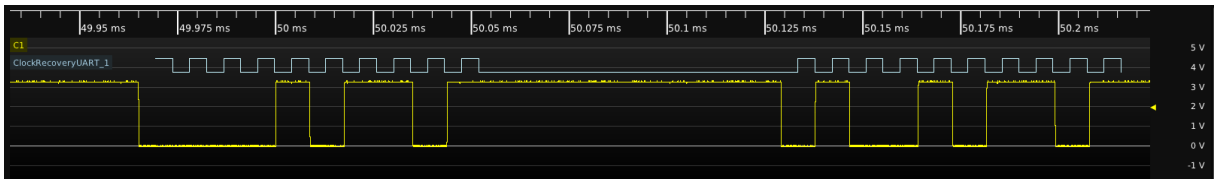



Figure 26.48: Example of UART CDR on two serial data frames separated by a short delay

### 26.30.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| din | Analog | Input waveform |

### 26.30.2 Parameters

| Parameter name | Type | Description |
|---|---|---|
| Baud rate | Float | Symbol rate, in bps |
| Threshold | Float | Decision threshold for the edge detector, in volts |

### 26.30.3 Output Signal

This filter outputs a digital waveform with the sampling clock recovered from the analog stream.

## 26.31 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 interleaved in I-Q-I-Q order.

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

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

Integer sample data is rescaled to a nominal +/- 1V p-p. Floating point data is not rescaled and the raw floating point values are output as volts.

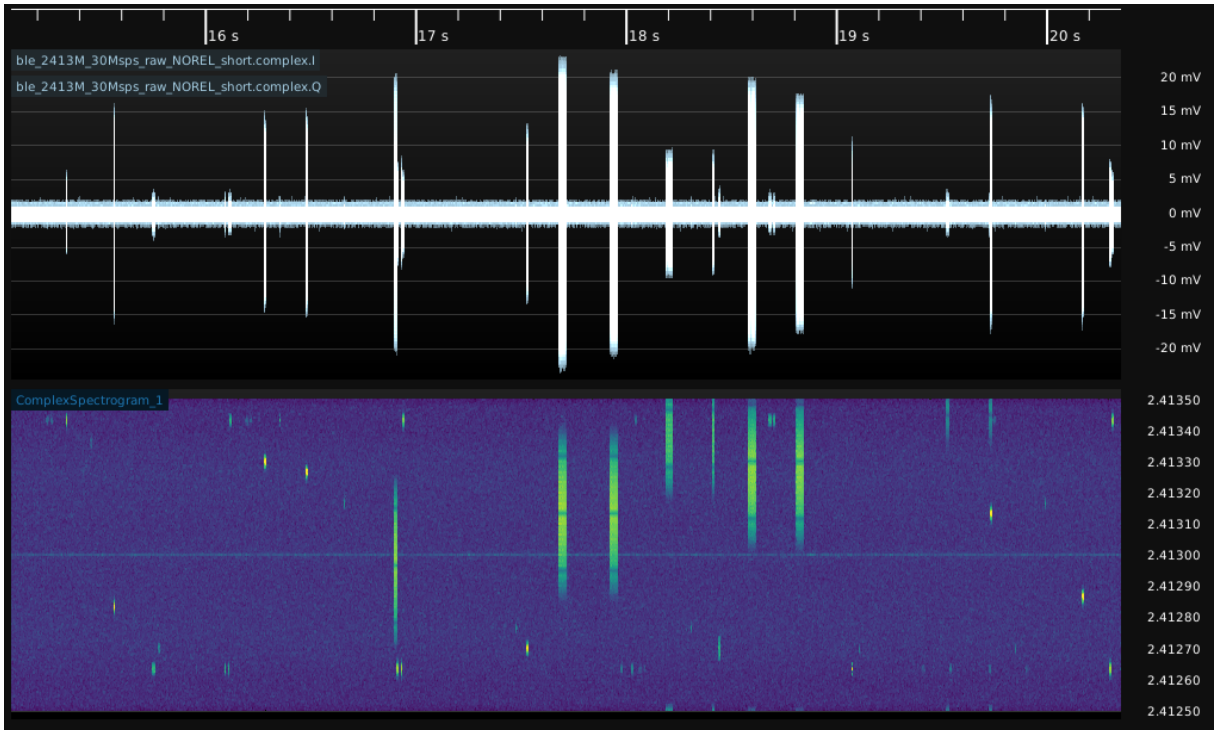



Figure 26.49: Example of importing and processing complex data

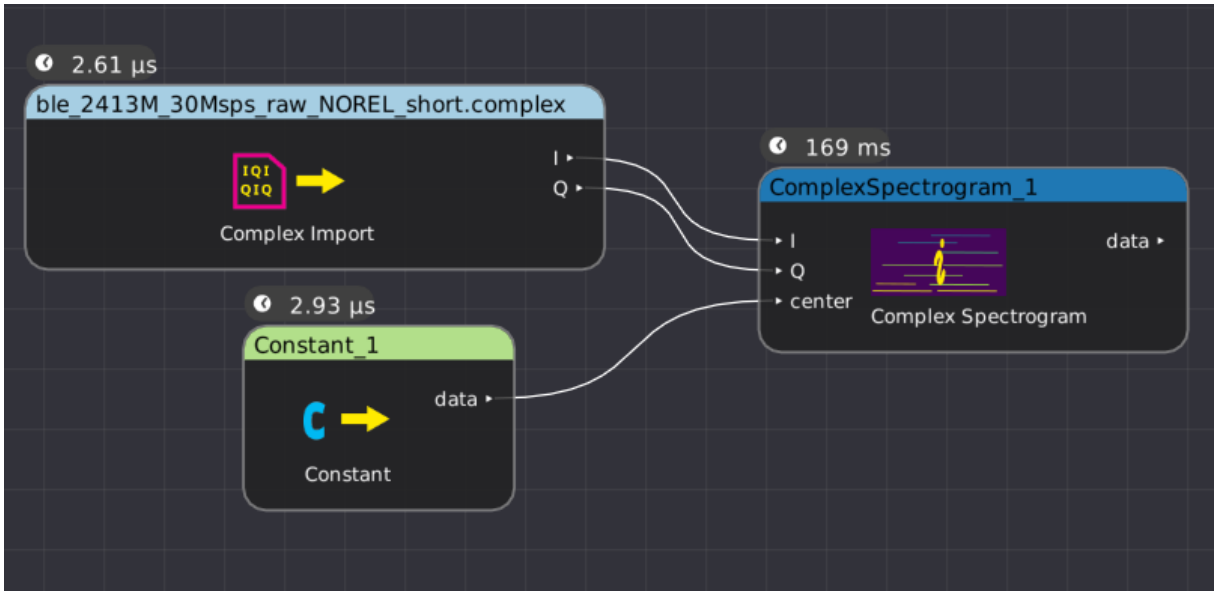



Figure 26.50: Example filter graph showing usage of complex import filter

### 26.31.1 Inputs

This filter takes no inputs.

### 26.31.2 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 |

### 26.31.3 Output Signal

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

## 26.32 Complex Spectrogram

Plots a spectrogram of complex I/Q data.

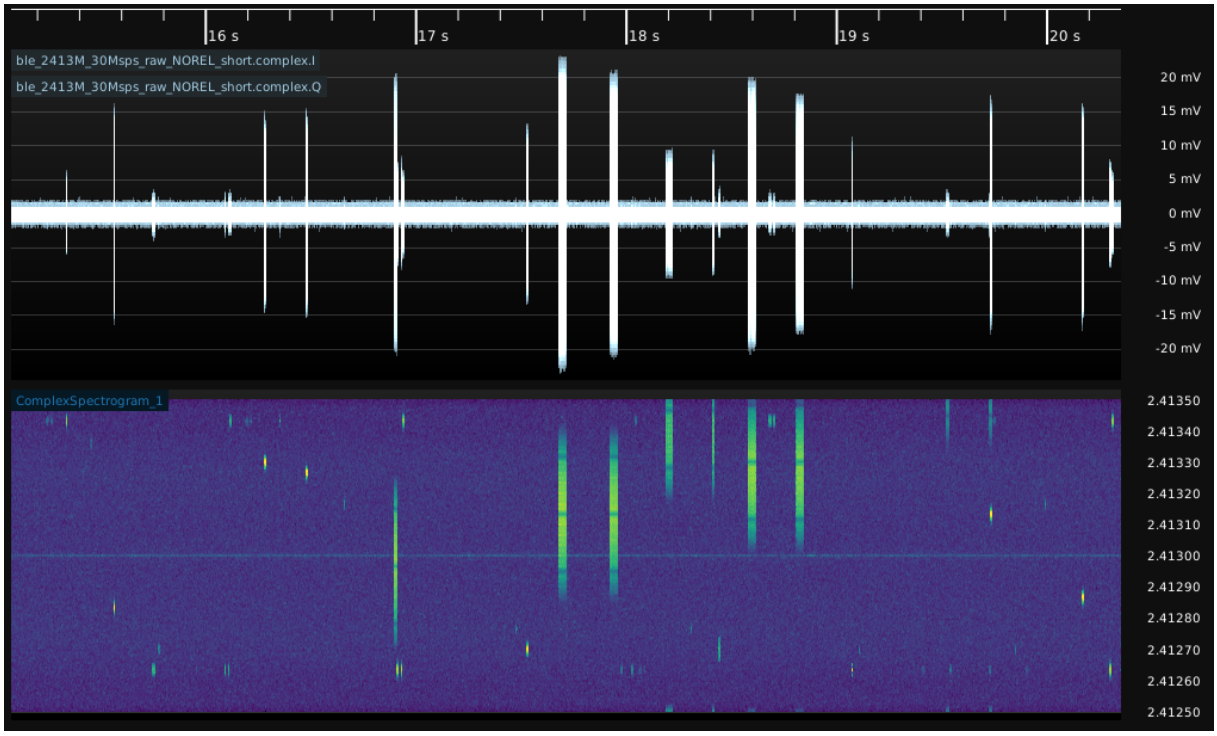



Figure 26.51: Example of a complex spectrogram

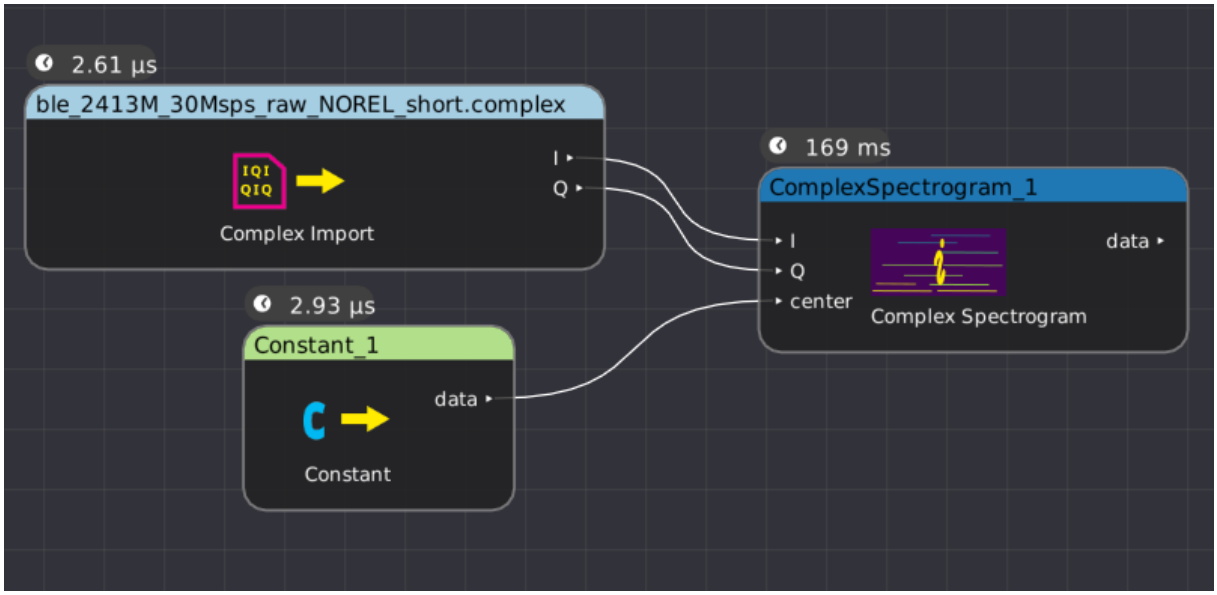



Figure 26.52: Example filter graph showing usage of complex spectrogram filter

### 26.32.1 Inputs

| Parameter name | Type | Description |
|---|---|---|
| I | Analog | Real part of input signal |
| Q | Analog | Imaginary part of input signal |
| center | Analog scalar | IF frequency to display the zero point of the complex waveform at |

### 26.32.2 Parameters

| Parameter name | Type | Description |
|---|---|---|
| FFT length | Int | Number of FFT points per spectrogram |
| Range Max | Float | Amplitude, in dBm, to use as the full-scale intensity value |
| Range Min | Float | Amplitude, in dBm, to use as the minimum-scale intensity value |
| Window | Enum | Window function to use for the FFTs |

### 26.32.3 Output Signal

This filter outputs a 2D density plot showing amplitude vs frequency and time.

## 26.33 Constant

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

### 26.33.1 Inputs

This filter takes no inputs.

### 26.33.2 Parameters

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

### 26.33.3 Output Signal

This filter outputs a single scalar with a constant value.

## 26.34 Constellation

This filter takes a pair of I/Q streams and outputs a constellation diagram.

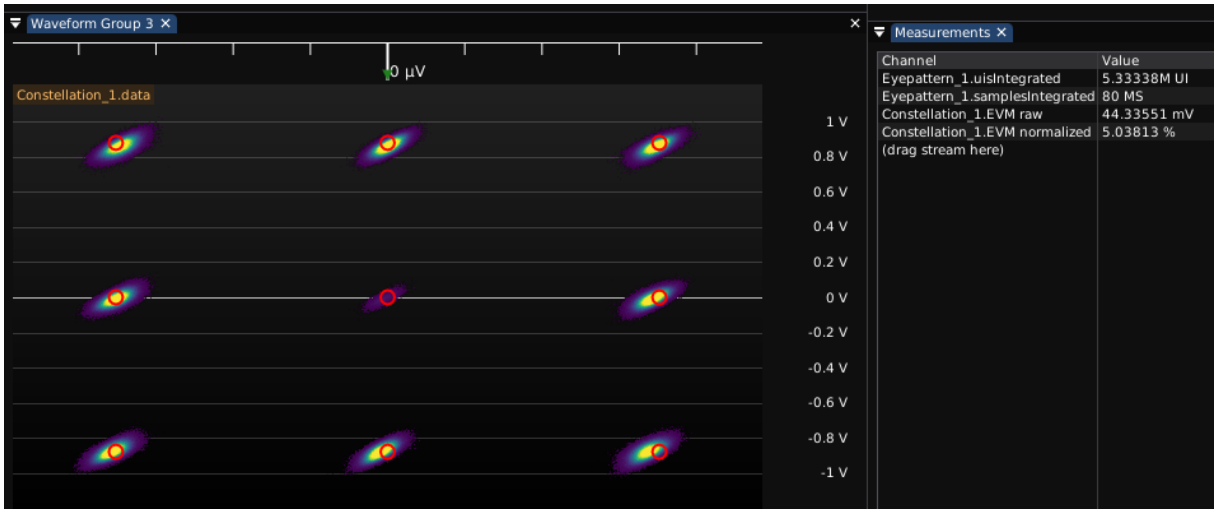



Figure 26.53: Example of a 9-point 2D-PAM3 constellation

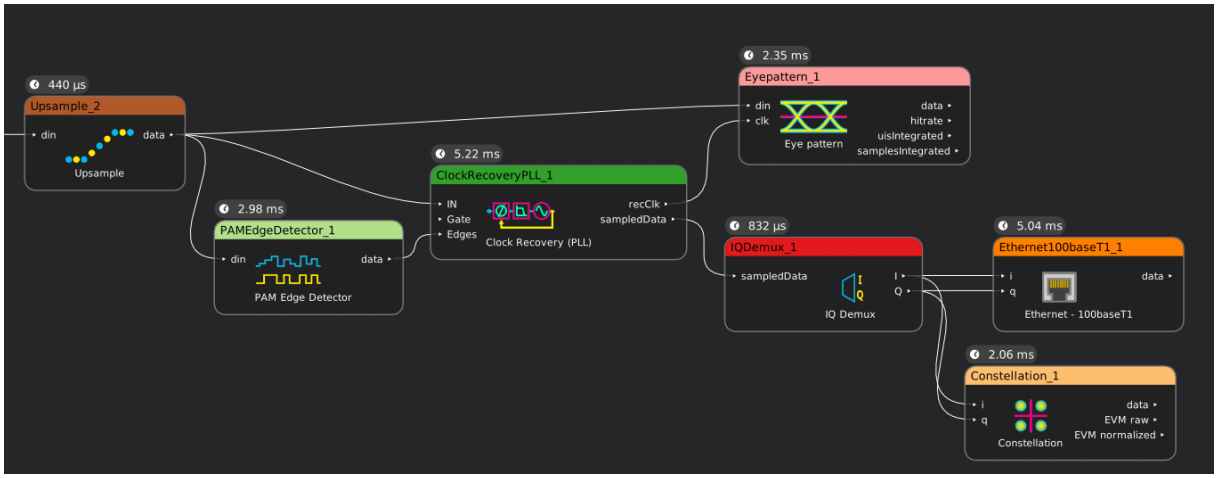



Figure 26.54: Example filter graph showing usage of constellation filter on 2D-PAM3 data

### 26.34.1 Inputs

| Parameter name | Type | Description |
|---|---|---|
| I | Sparse analog | Real part of input signal |
| Q | Sparse analog | Imaginary part of input signal |

### 26.34.2 Parameters

| Parameter name | Type | Description |
|---|---|---|
| Center I | Float | Voltage for center of the constellation in I axis |
| Center Q | Float | Voltage for center of the constellation in Q axis |
| Modulation | Enum | Constellation shape to use for EVM calculations |
| Range | Float | Nominal full-scale range |

### 26.34.3 Output Signal

This filter outputs a 2D density plot showing amplitude vs frequency and time, as well as error vector magnitude (EVM) measurements.

| Parameter name | Type | Description |
|---|---|---|
| data | Density plot | Constellation image |
| EVM raw | Analog scalar | Raw EVM in volts |
| EVM normalized | Analog scalar | Normalized EVM in percent |

### 26.34.4 Breaking Changes

This section lists major interface changes that may require projects to be updated.

- **v0.2:** This filter no longer needs the clock input that was required in v0.1. Instead, threshold the sampledData output of the Clock Recovery (PLL) and pass this to the constellation block, as shown in Fig. 26.54.

## 26.35 Coupler De-Embed

Given waveforms from both coupled ports of a dual directional coupler and the 4-port S-parameters of the coupler, de-embeds the coupler response in order to recover the forward and reverse waveforms.

NOTE: The current implementation of this filter requires the `VK_KHR_push_descriptor` Vulkan extension. A fallback implementation for GPUs without this extension will be added at some point in the future.

Both coupled-port waveforms must be the same sample rate, memory depth, and phase aligned with respect to each other. Use of phase-matched cables or manual deskew in postprocessing is very important especially for higher harmonics: since the de-embed algorithm relies on subtracting a modified version of one waveform from the the other, any time shift will increase leakage between the ports and degrade the quality of the de-embed. If the phase error is large enough the de-embedded waveforms can even have *more* leakage between paths than in the original signal, although with a reversed sign.

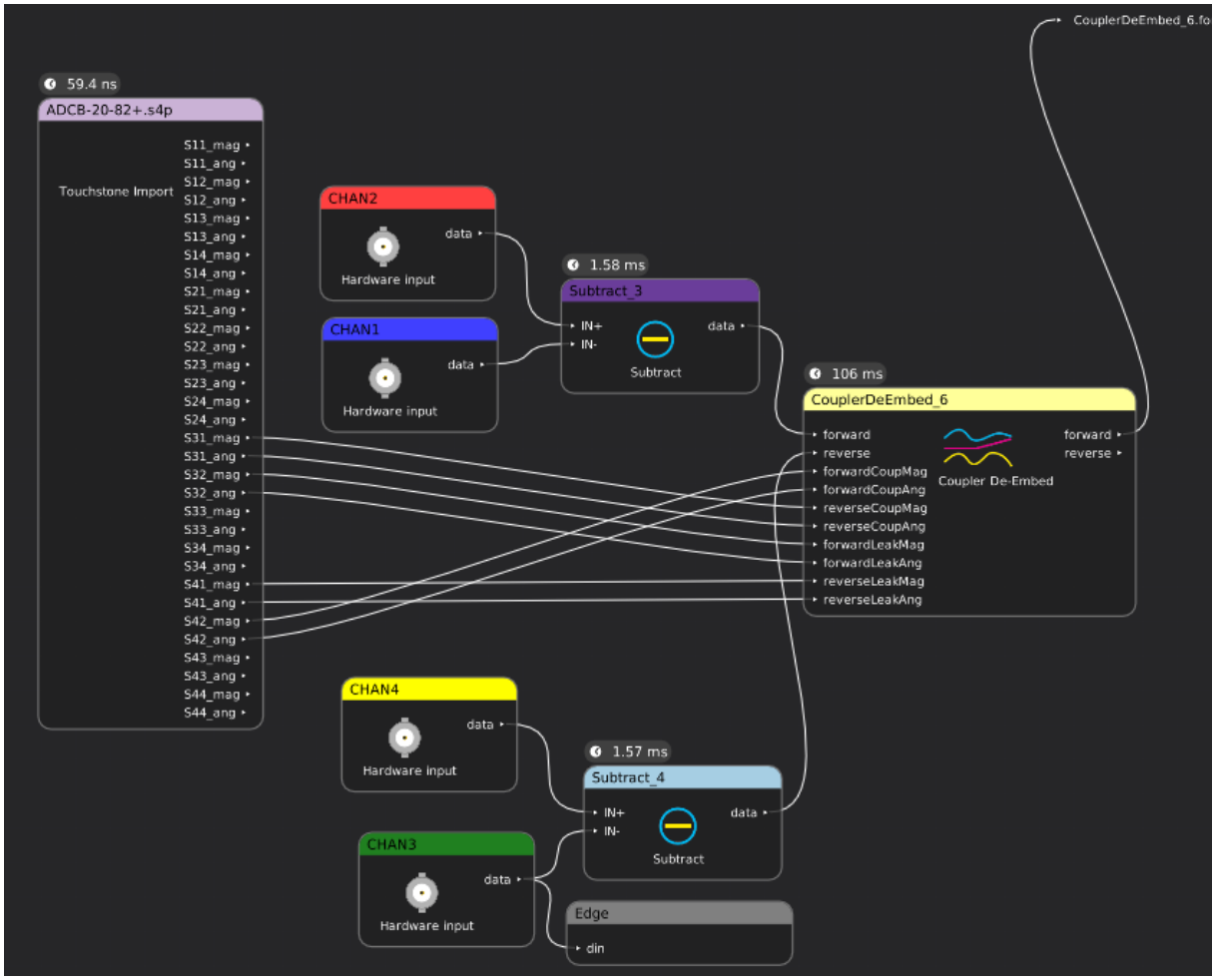



Figure 26.55: Example filter graph showing de-embed of differential signals traveling through Mini-Circuits ADCB-20-82+ couplers

This filter uses a multi-step algorithm to de-embed both the insertion loss of the coupled path and enhance the apparent directivity of the coupler:

1. De-embed the coupled path response from the coupled port waveforms in order to calculate an initial estimate of the input port waveforms. The same FFT-based algorithm as the De-Embed filter is used.

2. Given the initial estimated input port waveforms, calculate the leakage from the forward path to the reverse coupled port, and from the reverse path to the forward coupled port. The same FFT-based algorithm as the Channel Emulation filter is used. This estimate is imperfect since it assumes perfect directivity, so a small amount of the legitimate waveform is incorrectly included in the leakage waveform.

3. Subtract the leakage waveforms from the measured coupled port waveforms. This removes most of the leakage (as well as a small amount of the legitimate waveform).

4. De-embed the coupled path response from the subtracted waveform in order to get a revised estimate of input port waveforms. This is the final output of the filter.

### 26.35.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| forward | Analog | Forward coupled port waveform |
| reverse | Analog | Reverse coupled port waveform |
| forwardCoupMag | Analog | Magnitude response of forward coupled path |
| forwardCoupAng | Analog | Angle response of forward coupled path |
| reverseCoupMag | Analog | Magnitude response of reverse coupled path |
| reverseCoupAng | Analog | Angle response of reverse coupled path |
| forwardLeakMag | Analog | Magnitude response of forward leakage path |
| forwardLeakAng | Analog | Angle response of forward leakage path |
| reverseLeakMag | Analog | Magnitude response of reverse leakage path |
| reverseLeakAng | Analog | Angle response of reverse leakage path |

### 26.35.2 Output Signal

This filter outputs uniform analog waveforms containing waveforms de-embedded to the coupler's forward and reverse input ports.

## 26.36 CSV Export

Saves waveform data to a comma-separated-value file which can be imported into a spreadsheet, external script, or another visualization tool.

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

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

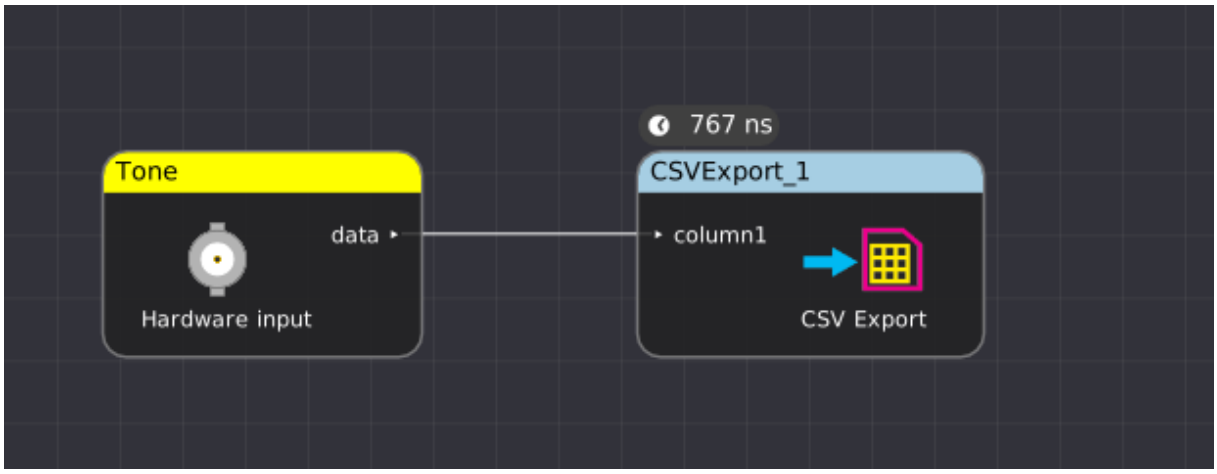



Figure 26.56: Example filter graph showing usage of CSV export block

### 26.36.1 Inputs

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

### 26.36.2 Parameters

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

### 26.36.3 Output Signal

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

The CSV file will consist of a header line with name “Time (s)" for the first column, and the names of the filter graph inputs for subsequent columns. There is no escaping of commas in the filter block names, so column names may be corrupted if a comma is present in a block name.

## 26.37 CSV Import

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

Lines starting with a # are assumed to be comments, and not parsed as sample data. If Digilent WaveForms format header comments are detected at the start of the file, they are parsed to extract timestamps and other metadata. If non-numeric characters are found in the first non-comment row of the file, the fields are interpreted as channel names and propagated to the output streams of the filter.

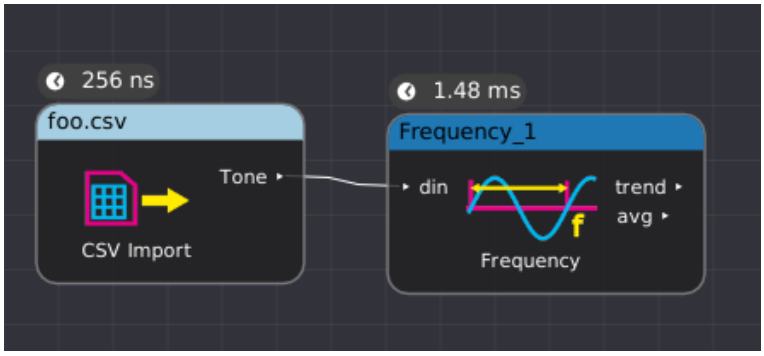



Figure 26.57: Example filter graph showing usage of CSV import block

### 26.37.1 Inputs

This filter takes no inputs.

### 26.37.2 Parameters

| Parameter name | Type | Description |
|---|---|---|
| CSV File | String | Path to the CSV file |
| X Axis Unit | Enum | Unit used for row timestamps |
| Y Axis Unit N | Enum | Unit used for Y axis values in column N |

### 26.37.3 Output Signal

This filter outputs one stream for each column in the CSV, other than the first which is assumed to be a timestamp. Timestamps must be monotonic (each sample's timestamp must be greater than the one before) but need not be evenly spaced.

If all values in the first 10 non-header lines for a given column are "0" or "1", with no decimal point, the column is interpreted as digital and parsed as such. Otherwise, it is assumed to be analog. There is currently no way to override this detection heuristic, but it may be added in the future.

After the import is complete, some statistics are computed to the intervals between timestamps to determine if the data is uniformly sampled. If the standard deviation of the intervals is greater than 2% of the average interval, or if the maximum sample interval is more than 1.5 times the minimum interval, the data is assumed to be sampled irregularly and a sparse waveform is created. If not, the data is assumed to be uniformly sampled (with any variability in sample timestamps due to rounding error) and a uniform waveform is greated. There is currently no way to override this detection heuristic, but it may be added in the future.

## 26.38 CTLE

Calculates S-parameters of a continuous-time linear equalizer with the specified pole/zero combination. As of now only two poles and one zero are supported, but higher order CTLE support may be implemented in the future.

Typically, this filter will be chained with a Channel Emulation block to actually apply the equalizer to a time-domain signal.

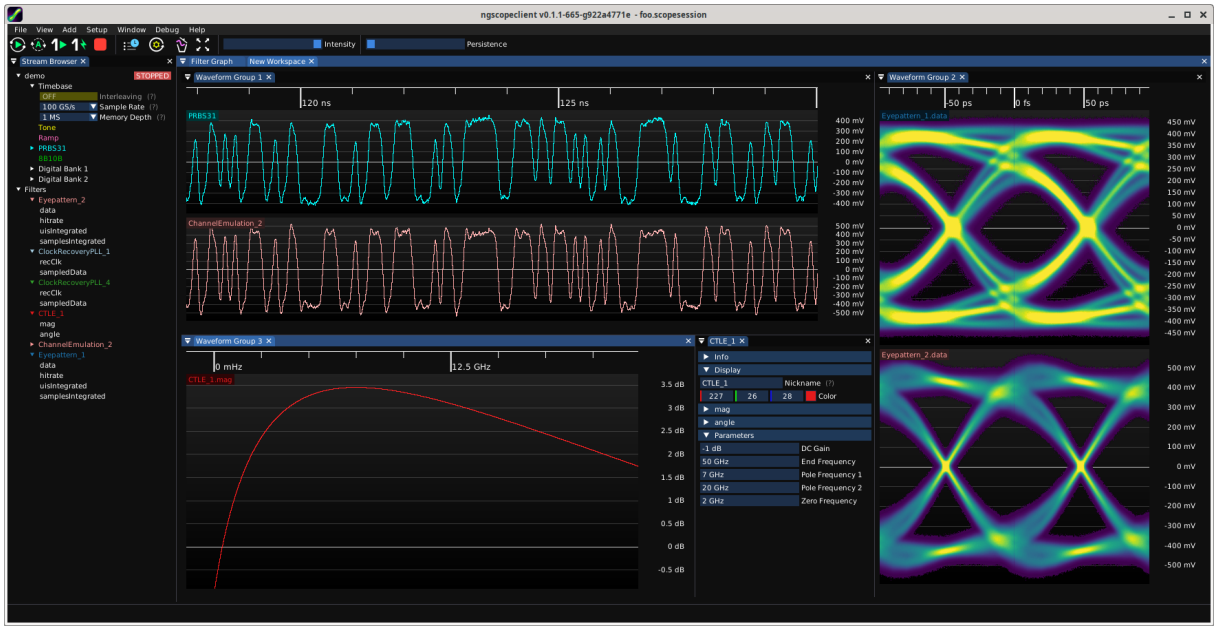


Figure 26.58: Example usage of CTLE block, comparing eyes of a signal before and after equalization

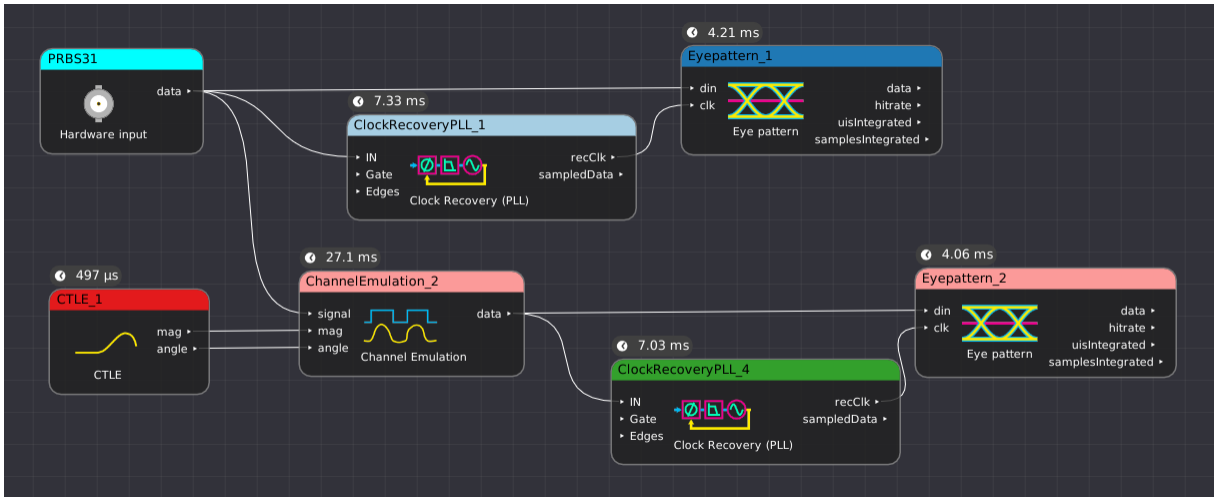



Figure 26.59: Example filter graph showing usage of CTLE block, comparing eyes of a signal before and after equalization

### 26.38.1 Inputs

This filter takes no inputs.

### 26.38.2 Parameters

| Parameter name | Type | Description |
|---|---|---|
| DC Gain | Float | Gain, in dB, of the equalizer at DC |
| End Frequency | Int | Upper end of the S-parameter sweep |
| Pole Frequency 1 | Int | Frequency of the first pole |
| Pole Frequency 2 | Int | Frequency of the second pole |
| Zero Frequency | Int | Frequency of the first zero |

### 26.38.3 Output Signal

This filter outputs log magnitude and phase angle values which can be passed to any other filter block that accepts an S-parameter input.

### 26.38.4 Breaking Changes

This section lists major interface changes that may require projects to be updated.

- **v0.2:** This filter now outputs S-parameters which must be passed to the Channel Emulation filter to apply them to a time-domain signal. The original version of the filter performed the channel emulation internally.

## 26.39 Current Shunt

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

(This filter description is a stub and will be expanded in the future)

## 26.40 DCD

Calculates the duty-cycle distortion for a serial data stream. This is the component of data-dependent jitter which is attributable to asymmetry between rising and falling edges.

The DCD calculation involves examining the DDJ table and splitting the measurements up into two bins, one that ends in a rising edge and one ending in a falling. All of the DDJ measurements in each bin are averaged, giving one value for the average jitter of a rising edge and one for the jitter of a falling. The difference between the two gives the final DCD measurement.

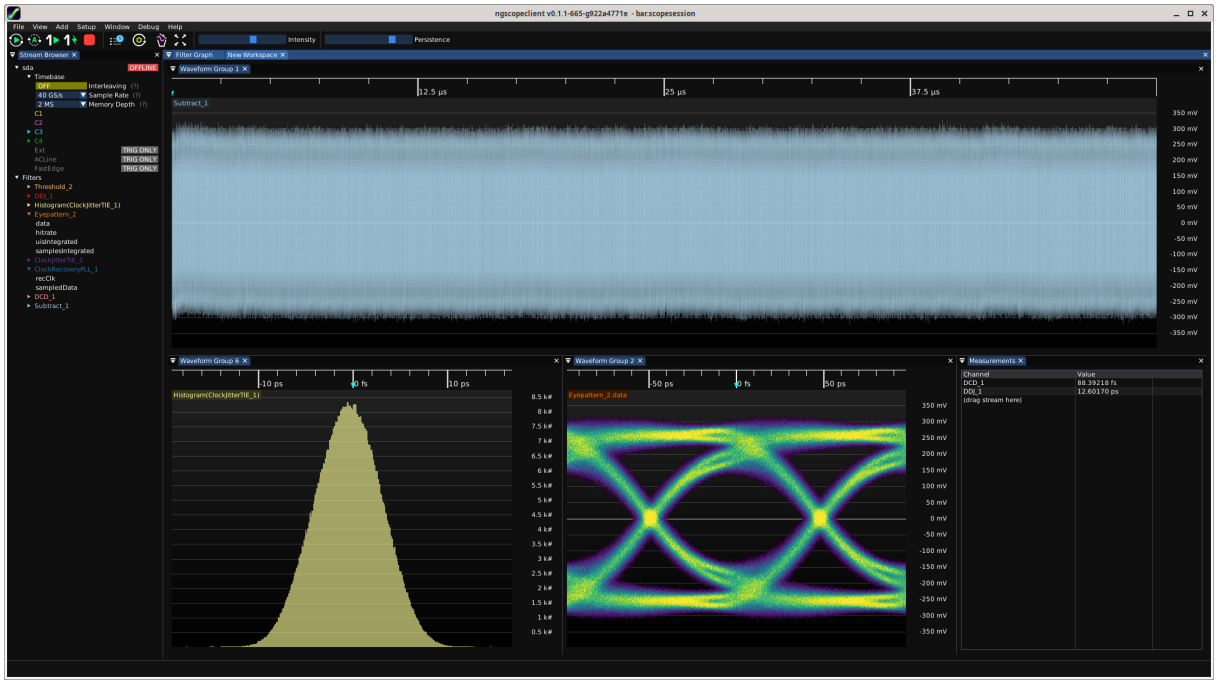


Figure 26.60: Example usage of DCD block

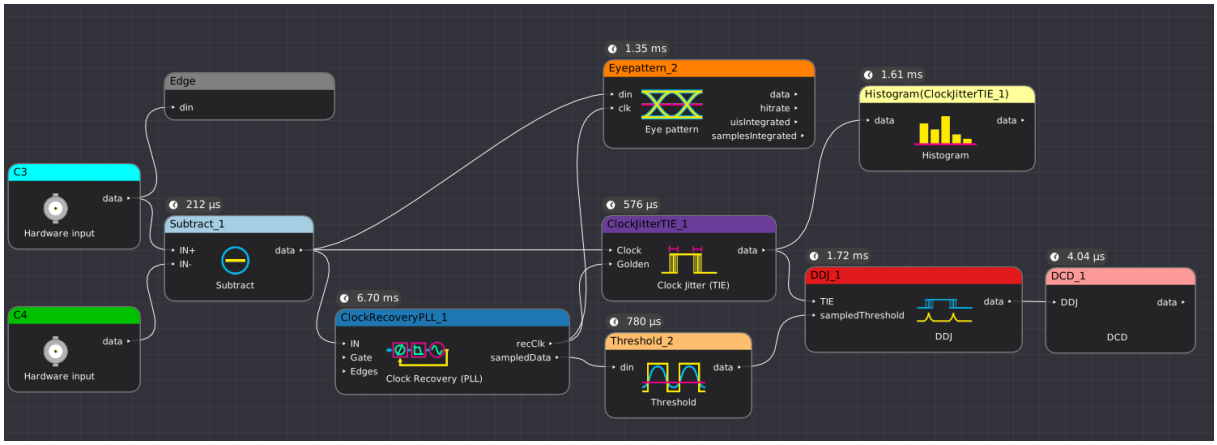



Figure 26.61: Example filter graph showing usage of DCD block

### 26.40.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| DDJ | Protocol | Output of DDJ measurement |

### 26.40.2 Parameters

This filter takes no parameters.

### 26.40.3 Output Signal

This filter outputs an analog scalar with the computed DCD measurement.

## 26.41 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 (random and periodic) 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.

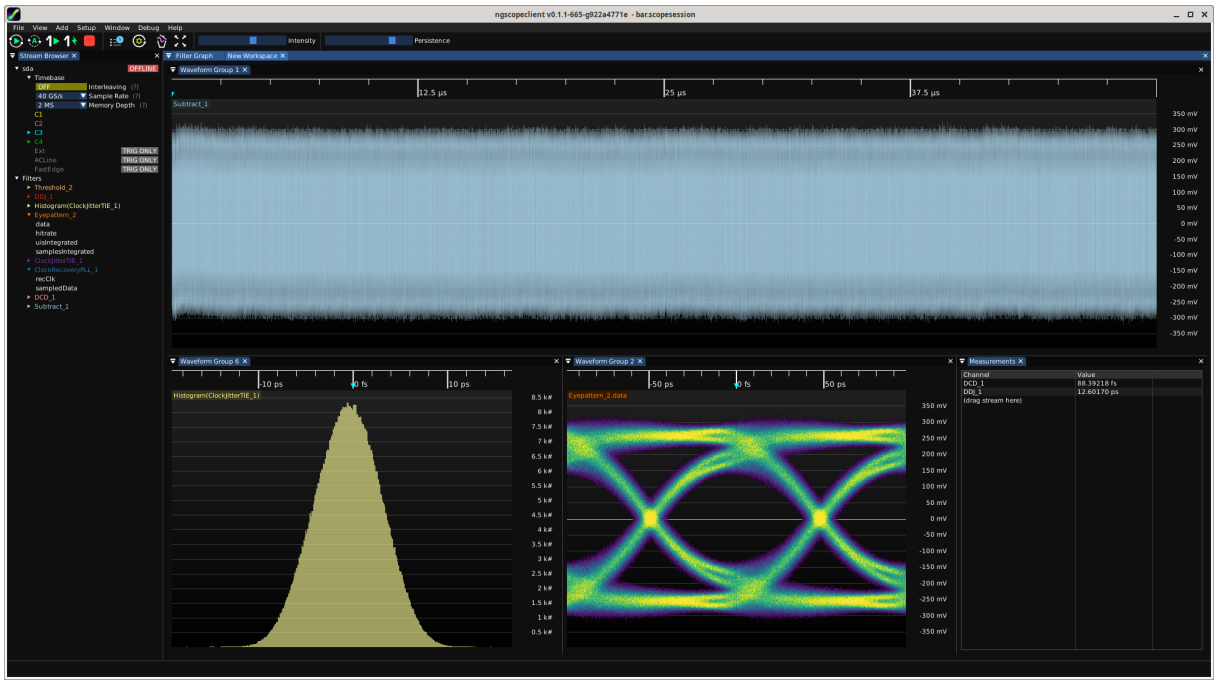


Figure 26.62: Example usage of DDJ block

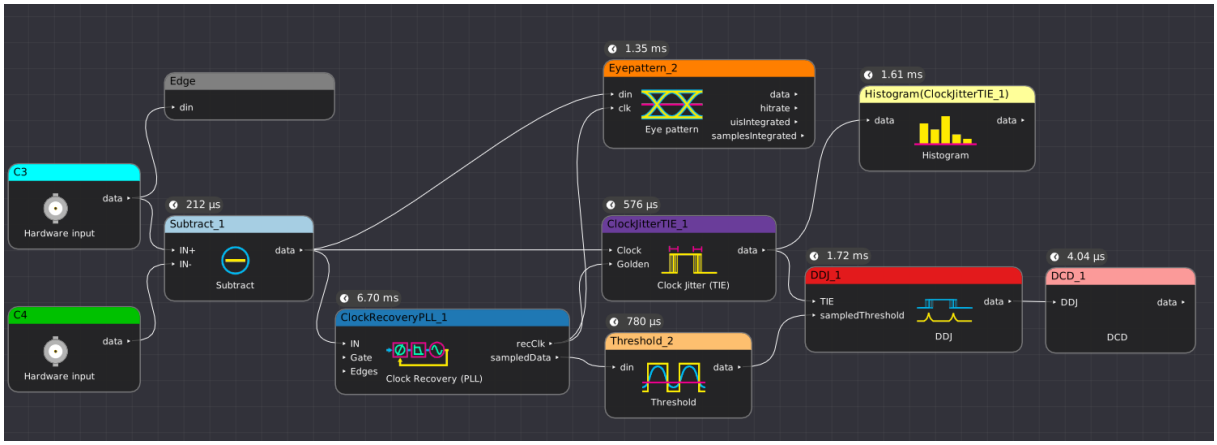



Figure 26.63: Example filter graph showing usage of DDJ block

### 26.41.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| TIE | Analog | TIE waveform computed by the TIE filter |
| sampledThreshold | Digital | Thresholded digital sample values |

### 26.41.2 Parameters

This filter takes no parameters.

### 26.41.3 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.

### 26.41.4 Breaking Changes

This section lists major interface changes that may require projects to be updated.

- **v0.2:** This filter no longer needs the clock input that was required in v0.1. Instead, threshold the sampledData output of the Clock Recovery (PLL) and pass this to the sampledThreshold block, as shown in Fig. 26.65.

## 26.42 DDR1 Command Bus

Decodes the command bus for first-generation DDR SDRAM.

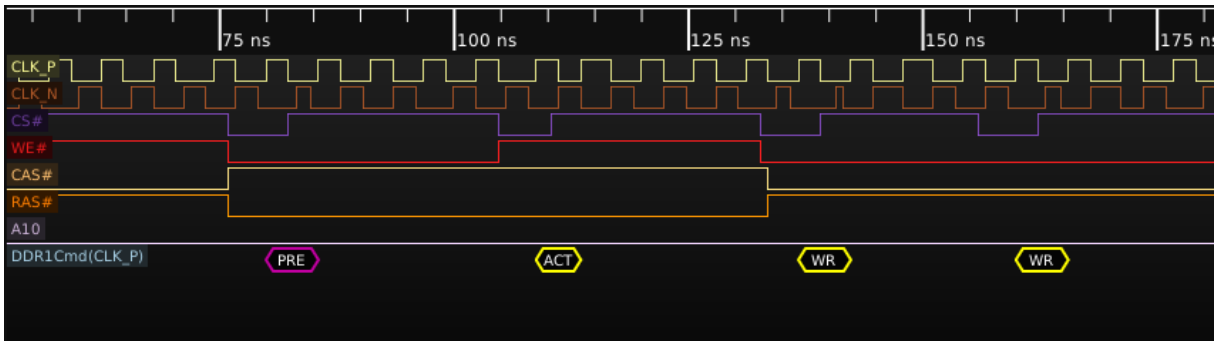



Figure 26.64: Example usage of DDR1 command bus decode

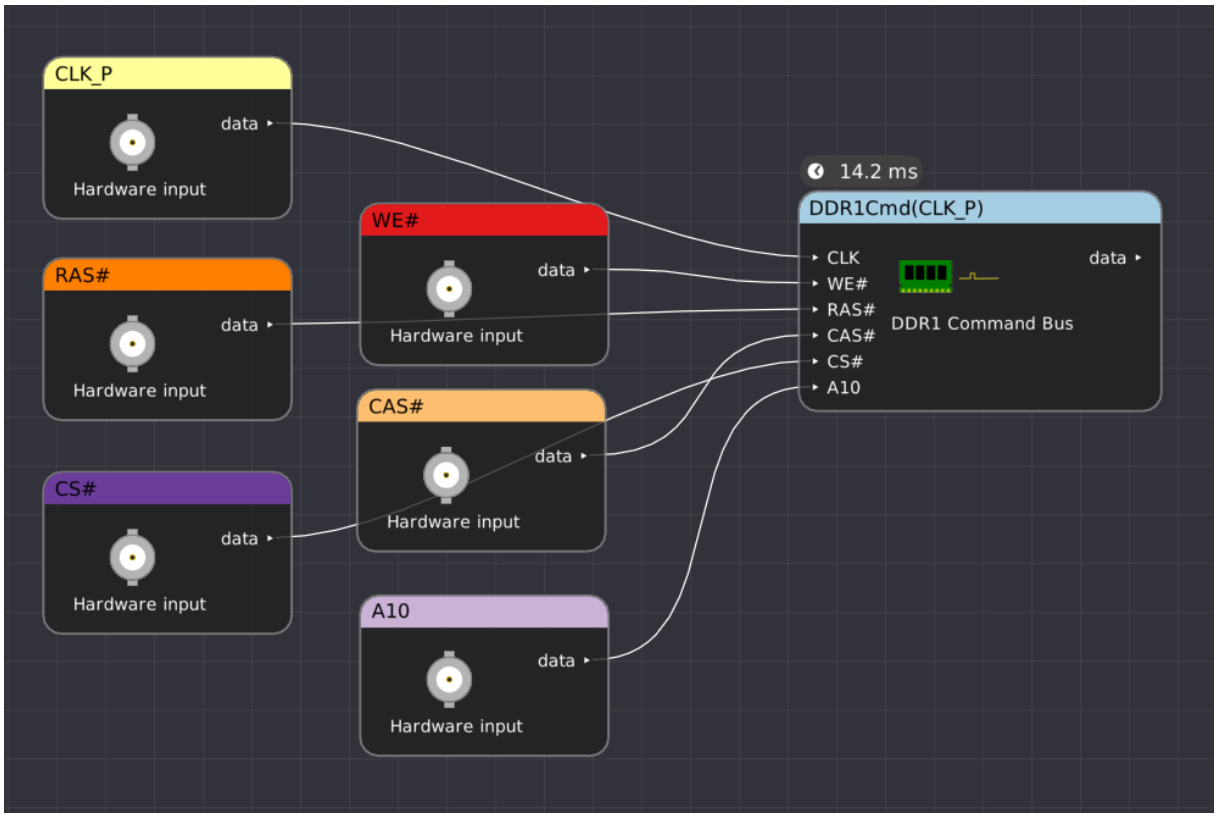



Figure 26.65: Example filter graph showing usage of DDR1 command bus decode

### 26.42.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| CLK | Digital | DDR clock |
| WE# | Digital | DDR write enable |
| RAS# | Digital | DDR row address strobe |
| CAS# | Digital | DDR column address strobe |
| CS# | Digital | DDR chip select |
| A10 | Digital | DDR address bit 10 (used as command input) |

### 26.42.2 Parameters

This filter takes no parameters.

### 26.42.3 Output Signal

This filter outputs a stream of decoded DDR1 command events.

## 26.43 DDR3 Command Bus

Decodes the command bus for third-generation DDR SDRAM.

(This filter description is a stub and will be expanded in the future)

## 26.44 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.

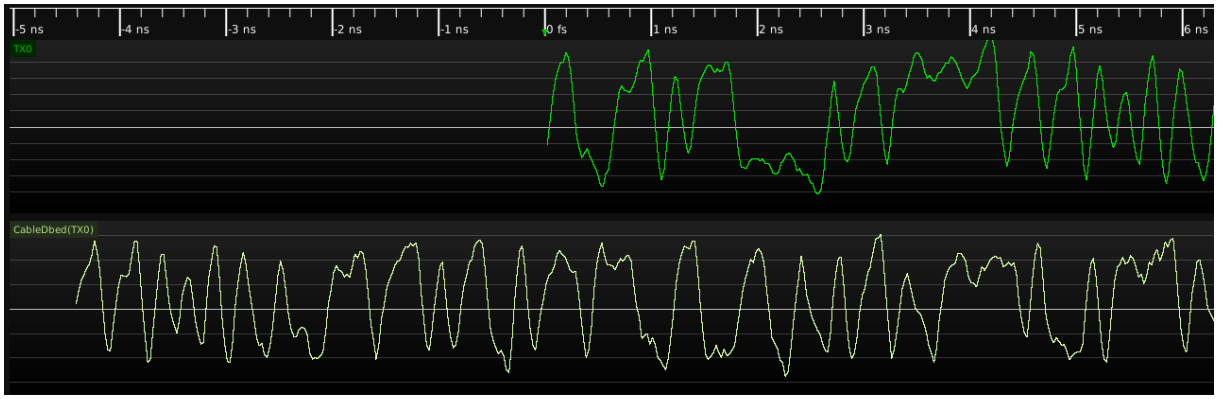



Figure 26.66: Example of de-embedding a cable from a measurement

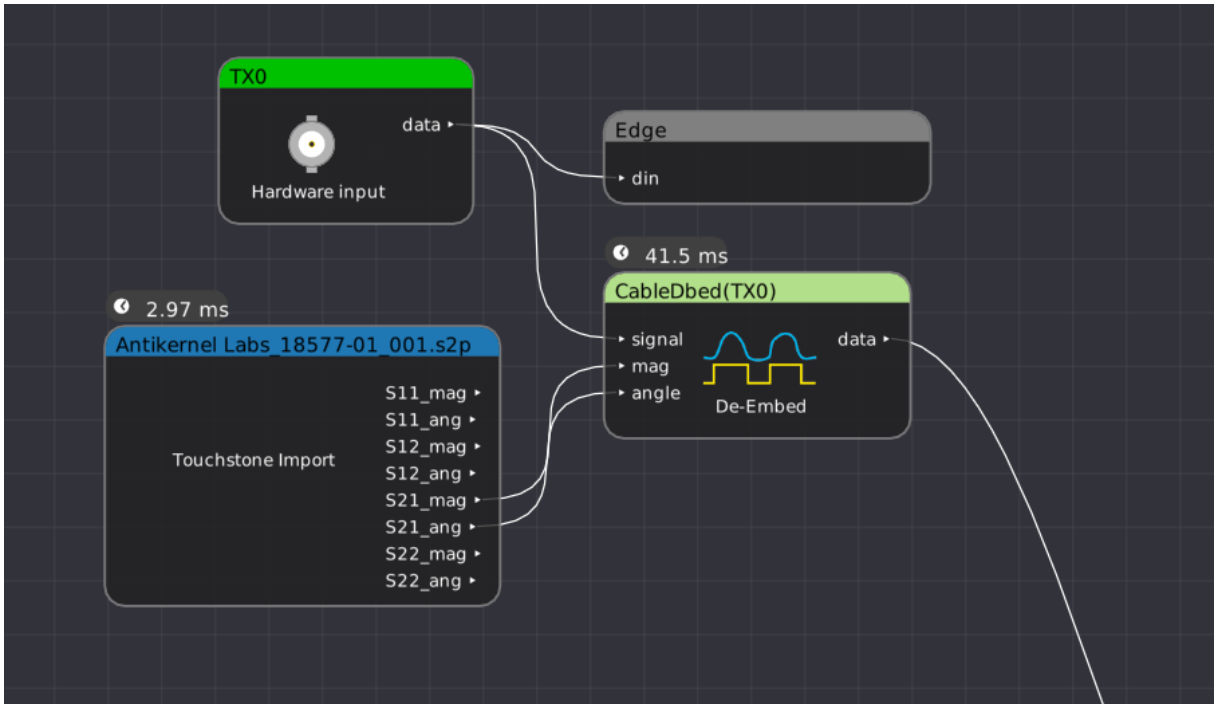



Figure 26.67: Example filter graph showing usage of de-embed filter

### 26.44.1 Inputs

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

### 26.44.2 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 |

### 26.44.3 Output Signal

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

## 26.45 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 if supported by hardware; this filter is provided for correction in postprocessing or for use with instruments that do not have hardware deskew capability.

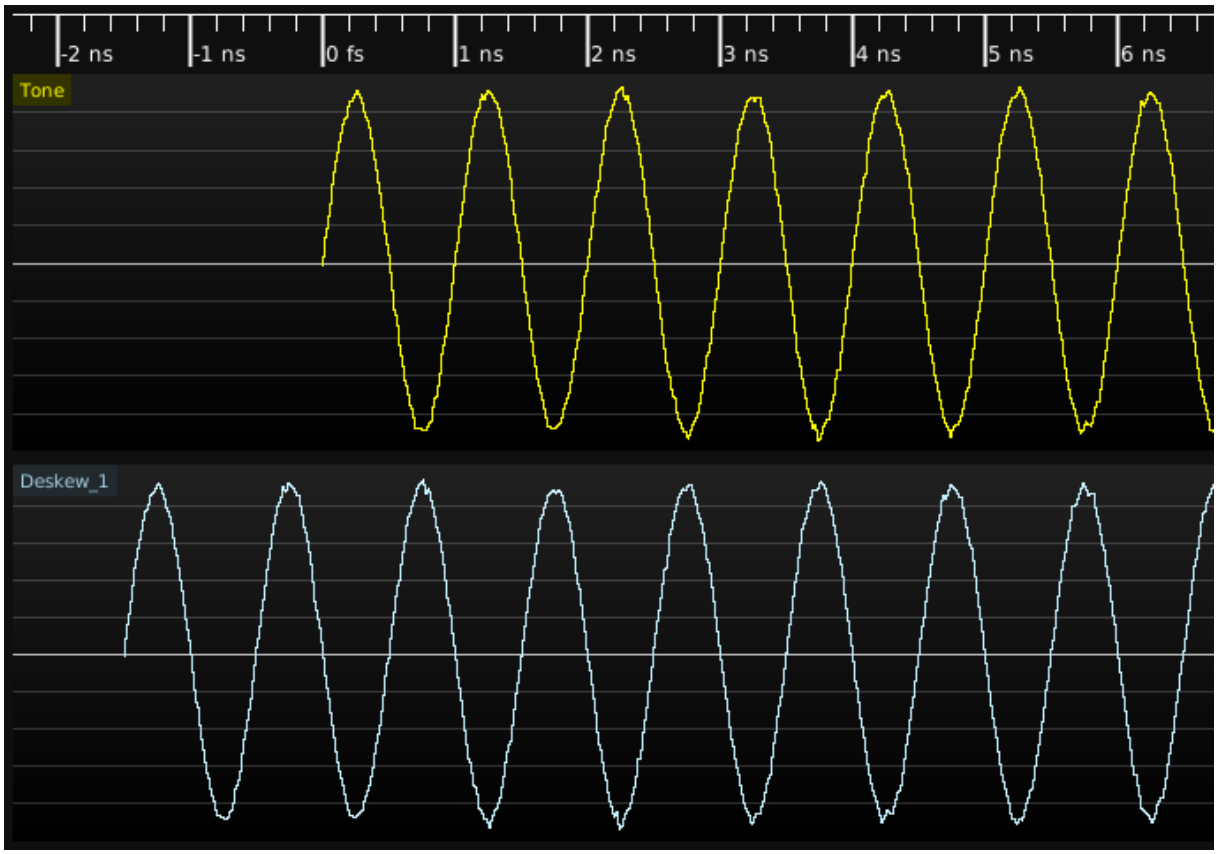



Figure 26.68: Example of deskewing a signal

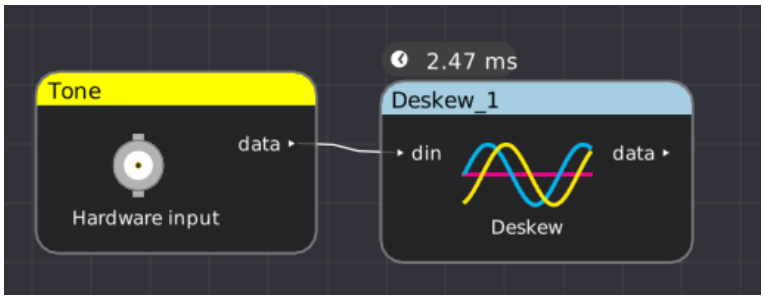



Figure 26.69: Example filter graph showing usage of deskew filter

### 26.45.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| din | Analog | Input waveform |

### 26.45.2 Parameters

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

### 26.45.3 Output Signal

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

## 26.46 Digital Constant

Outputs a constant digital value of configurable width.

This filter is typically used either as a configuration strap to filters that take a digital scalar input (such as enabling or disabling some mode), or connected to a digital GPIO channel (either a physical pin on an I/O board, or an internal net connected to a debug IP block on an FPGA).

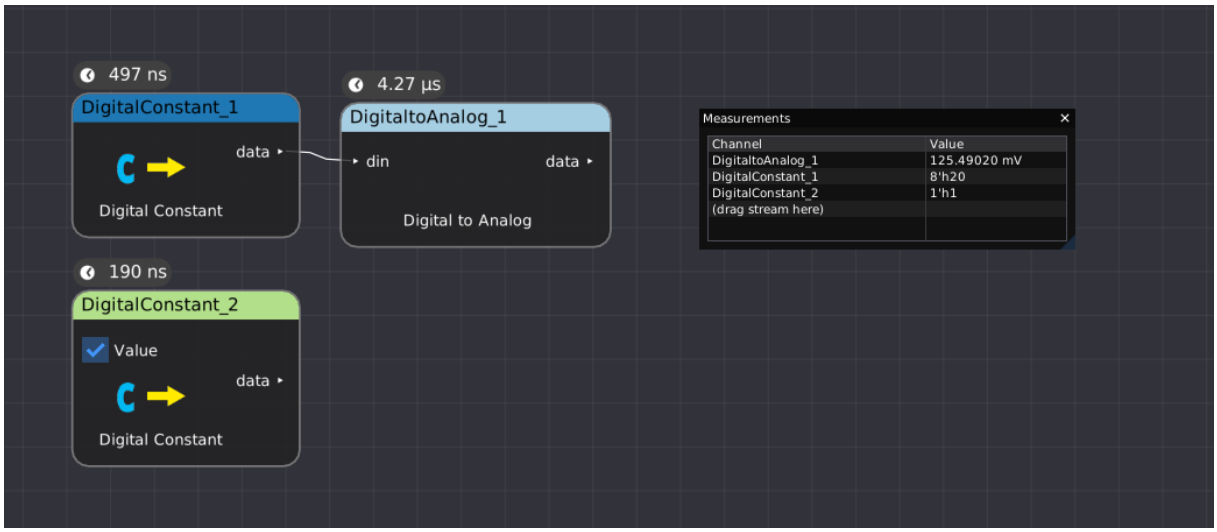



Figure 26.70: Example filter graph showing usage of digital to constant filter

### 26.46.1 Inputs

This filter takes no inputs.

### 26.46.2 Parameters

| Parameter name | Type | Description |
|---|---|---|
| Value | Hex integer | The value to drive on the output |
| Width | Integer | Width of the output signal, in bits |

### 26.46.3 Output Signal

This filter outputs a digital scalar containing the requested value.

## 26.47 Digital to Analog

Converts a digital value to an analog value using configurable units, gain, and offset. This is typically used to visualize data captured from FPGA debug IPs as digital samples.

This filter will be extended to support both waveform and scalar input in the future, but currently only supports scalars.

The output value is computed as (input - offset) * gain.

The input may be interpreted as normalized (0x0 maps to an output of 0.0 before scaling, while a full-scale input regardless of bit width maps to an output of 1.0 before scaling) or non-normalized (an input of 0x20 maps to 32.0 before scaling).

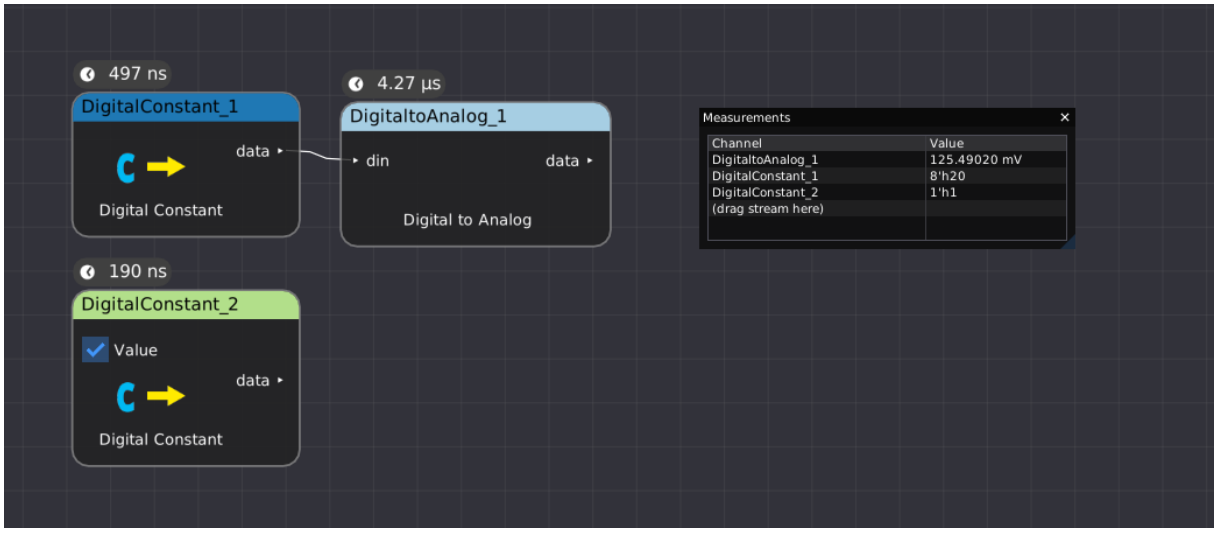



Figure 26.71: Example filter graph showing usage of digital to analog filter

### 26.47.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| din | Digital scalar | Value to convert |

### 26.47.2 Parameters

| Parameter name | Type | Description |
|---|---|---|
| Gain | Float | Gain to multiply the input by |
| Mode | Enum | How to interpret the input data |
| Offset | Float | Constant offset to subtract from the input |
| Unit | Enum | Units to apply to the output data |

### 26.47.3 Output Signal

This filter outputs an analog scalar containing the scaled output value.

## 26.48 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 the new value. For more accurate modeling of edge shape use the IBIS Driver filter with the appropriate IBIS model for your DUT.

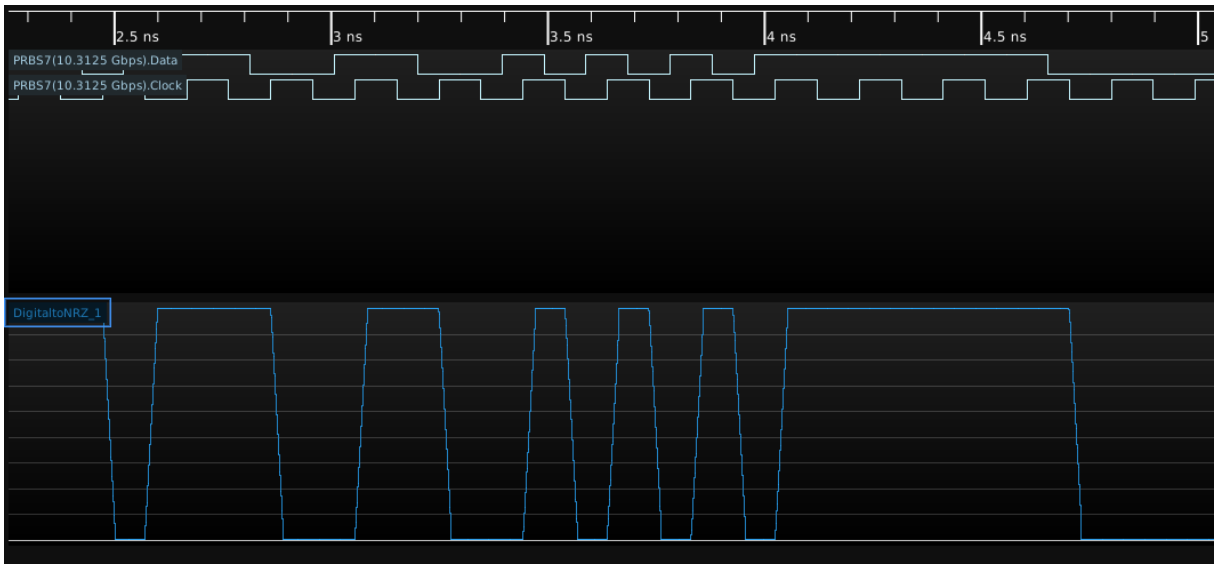



Figure 26.72: Example of converting digital waveform to analog NRZ

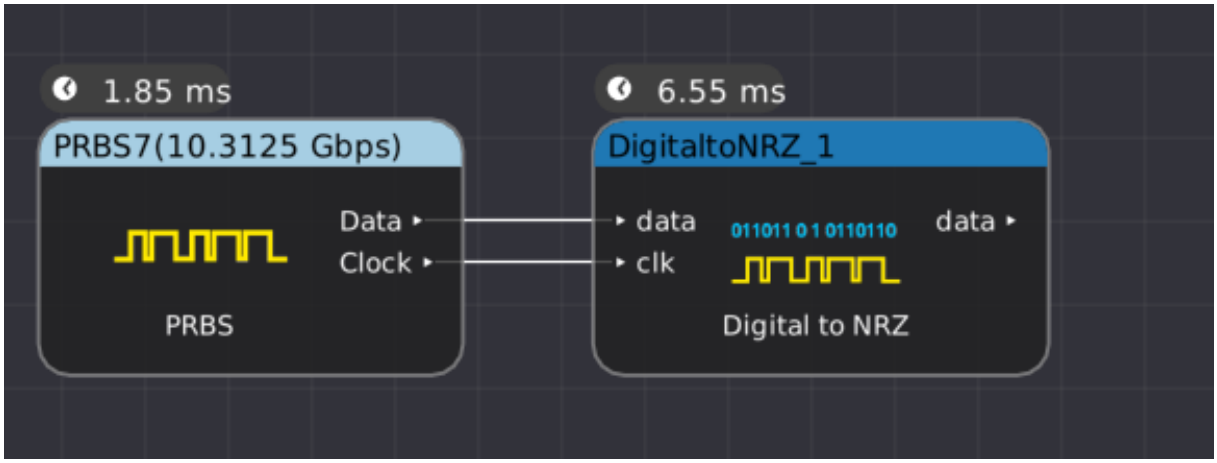



Figure 26.73: Example filter graph showing usage of digital to NRZ filter

### 26.48.1 Inputs

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

### 26.48.2 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 |

### 26.48.3 Output Signal

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

## 26.49 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 the 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.

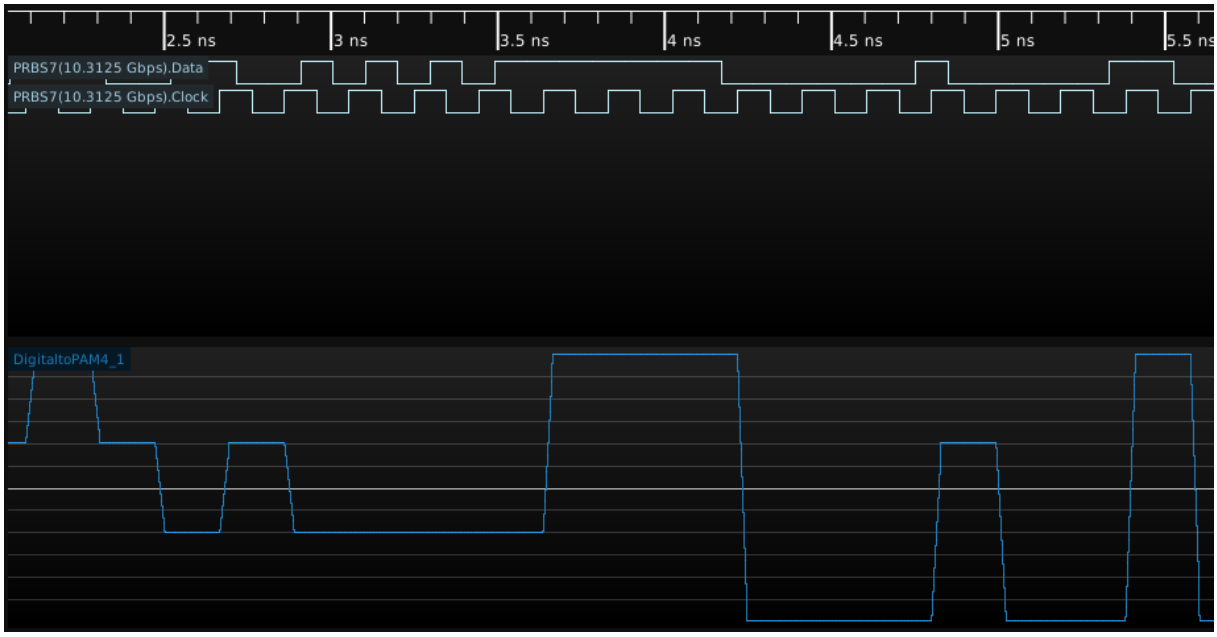



Figure 26.74: Example of converting digital waveform to analog PAM-4

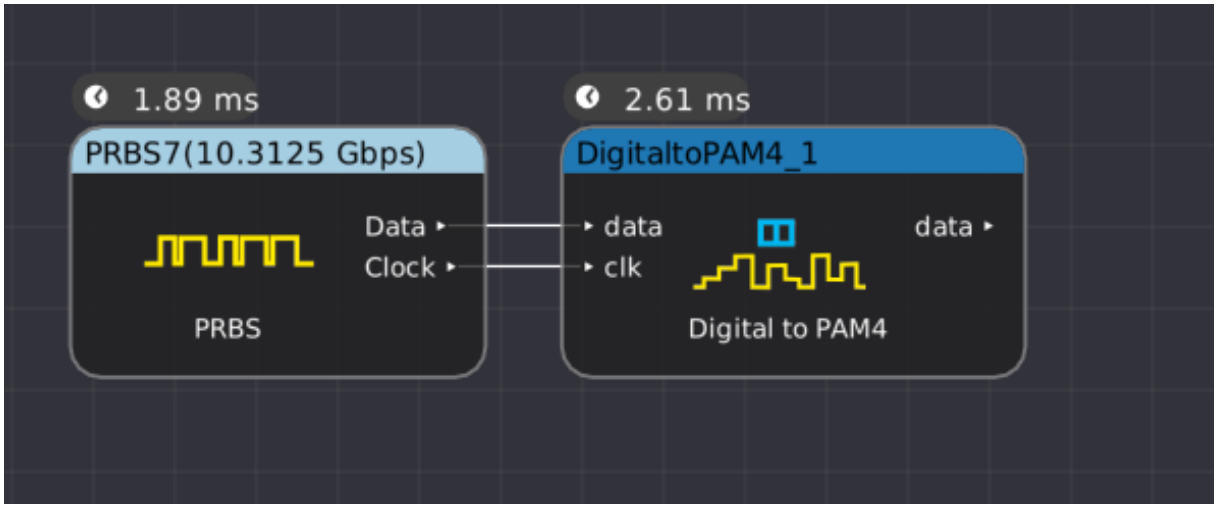



Figure 26.75: Example filter graph showing usage of digital to PAM-4 filter

### 26.49.1 Inputs

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

### 26.49.2 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 |

### 26.49.3 Output Signal

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

## 26.50 DisplayPort - Aux Channel

Decodes the Auxiliary Channel of DisplayPort.

This filter supports decoding both analog captures of the raw differential signal (via a differential probe or two single-ended inputs subtracted), and digital captures after thresholding.

Analog inputs are assumed to be differential and centered at zero, so if you are probing the output of a differential receiver with an analog probe the signal must be thresholded before passing to this block. This configuration is shown in Fig. 26.76 and Fig. 26.77.

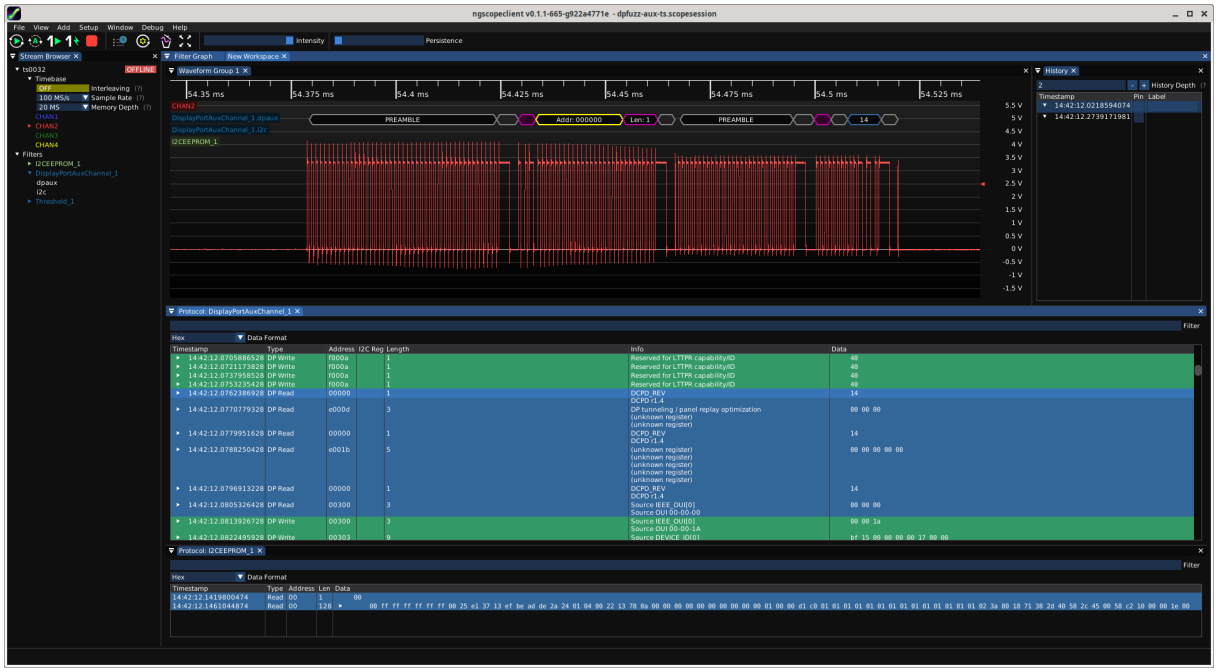


Figure 26.76: Example of decoding DisplayPort aux channel

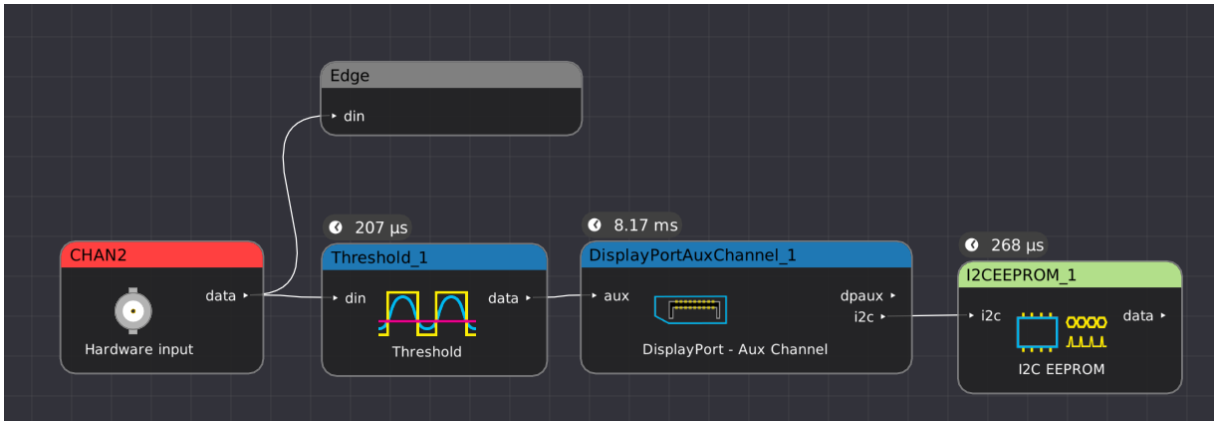



Figure 26.77: Example filter graph showing usage of DisplayPort aux channel decode

### 26.50.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| aux | Analog or digital | Auxiliary channel data |

### 26.50.2 Parameters

This filter takes no parameters.

### 26.50.3 Output Signal

This filter outputs a time series of DisplayPort protocol events containing accesses to both native registers and tunneled I2C traffic.

A second output stream provides access to the tunneled I2C data which can, for example, be passed to the I2C EEPROM protocol decode.

## 26.51 DisplayPort - Main Link

Decodes the Main Link of DisplayPort

This filter is a work in progress and does not support all features of the protocol yet. Notably missing features:

- 2 and 4 lane configurations
- MST mode
- 128b/132b line coding and UHBR bit rates
- FEC
- Display Stream Compression
- Color depths other than RGB24
- Decoding blanking interval data (currently only video scanlines are decoded)



Figure 26.78: Example of decoding DisplayPort main channel

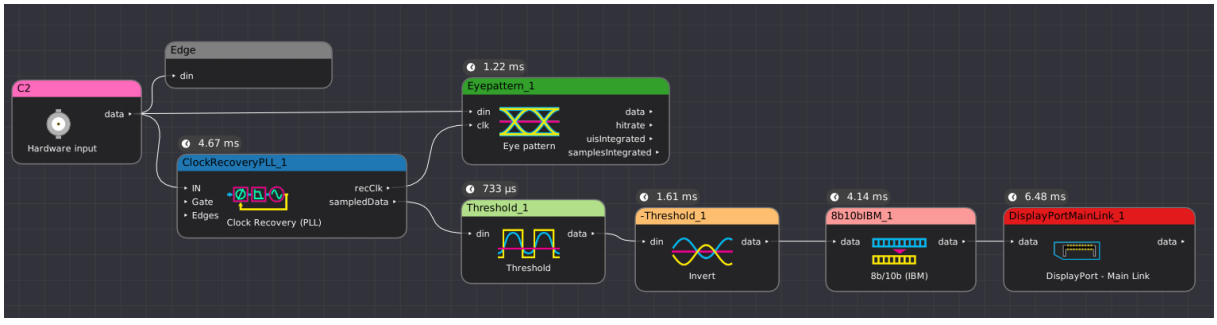



Figure 26.79: Example filter graph showing usage of DisplayPort main channel decode

### 26.51.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| data | 8B/10B | PHY layer symbols to decode |

### 26.51.2 Parameters

This filter takes no parameters.

### 26.51.3 Output Signal

This filter outputs a time series of DisplayPort protocol events containing pixel data.

As with all other video protocol decodes in ngscopeclient, scanlines are decoded to packets as both image data and hex byte values. Pixel data on the timeline is decoded as HTML-style #RRGGBB color codes and drawn in the appropriate color.

## 26.52 Divide

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

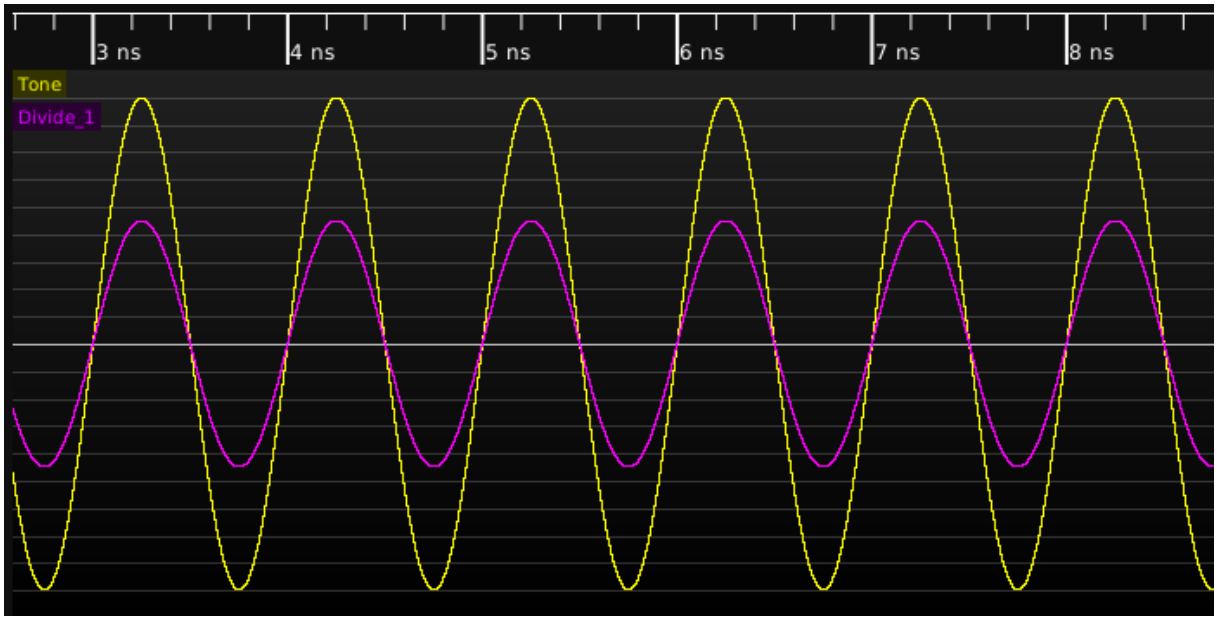



Figure 26.80: Example usage of dividing a waveform by a constant

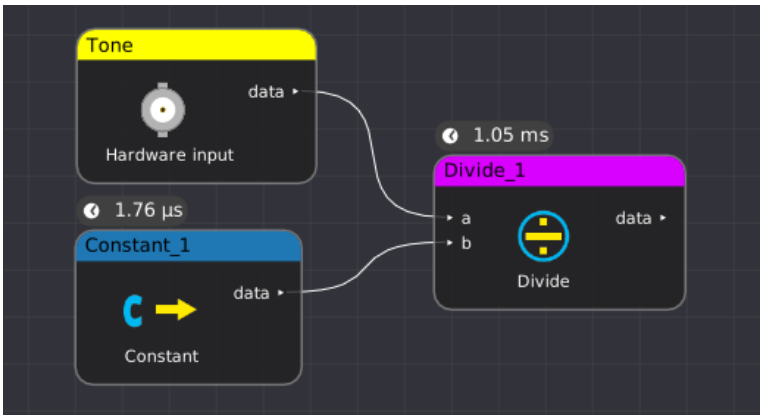



Figure 26.81: Example filter graph dividing a waveform by a constant

### 26.52.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| a | Analog waveform or scalar | Dividend |
| b | Analog waveform or scalar | Divisor |

### 26.52.2 Parameters

This filter takes no parameters.

### 26.52.3 Output Signal

If both inputs are vectors, this filter outputs a waveform containing the pairwise quotient; i.e. sample $i$ of the output is $a[i]/b[i]$. No resampling is performed on the inputs so incorrect or unexpected results may occur if they do not share the same timebase. Both inputs must be the same type (both sparse or both uniform), mixing sparse and uniform (even if the sample timestamps are the same) is not allowed.

If both inputs are scalars, this filter outputs $a/b$.

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

## 26.53 Downconvert

Performs digital downconversion by mixing a directly sampled RF signal with two phases of a local oscillator, then outputs the quadrature mix products. No LO rejection filtering or decimation is performed in the current version of the filter so these must be performed using external blocks.[3]

The optional LOFrequency scalar input can be used to dynamically control the LO from another filter block and, if connected, overrides the LO Frequency parameter.[4]

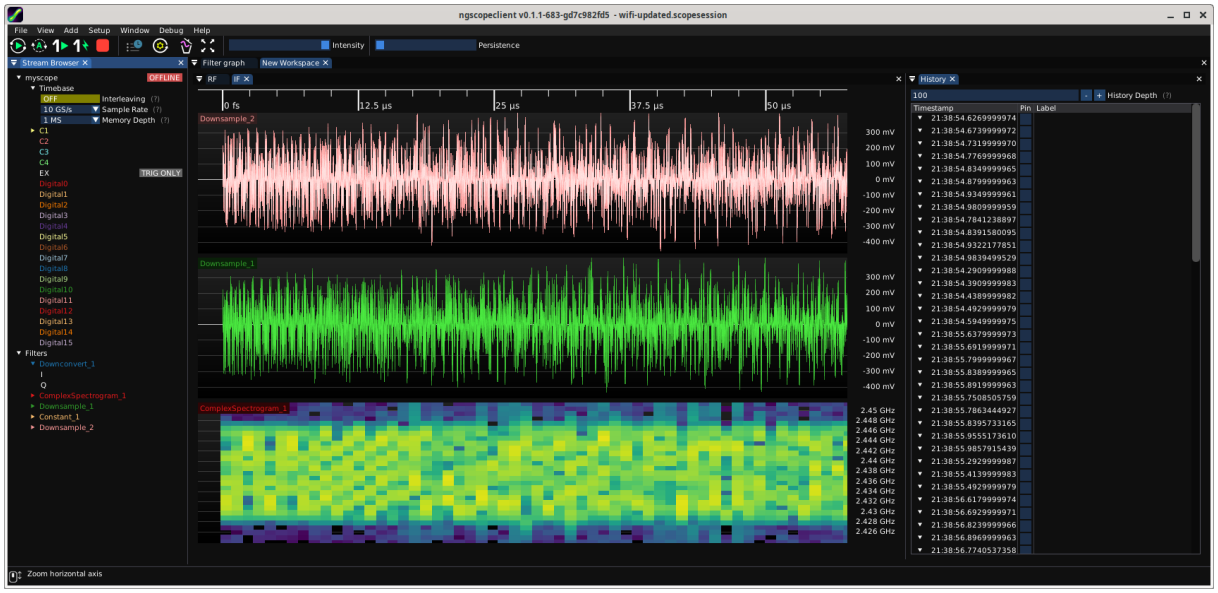



Figure 26.82: Example usage of downconverting a 2.437 GHz Wi-Fi signal to zero IF and displaying a spectrogram

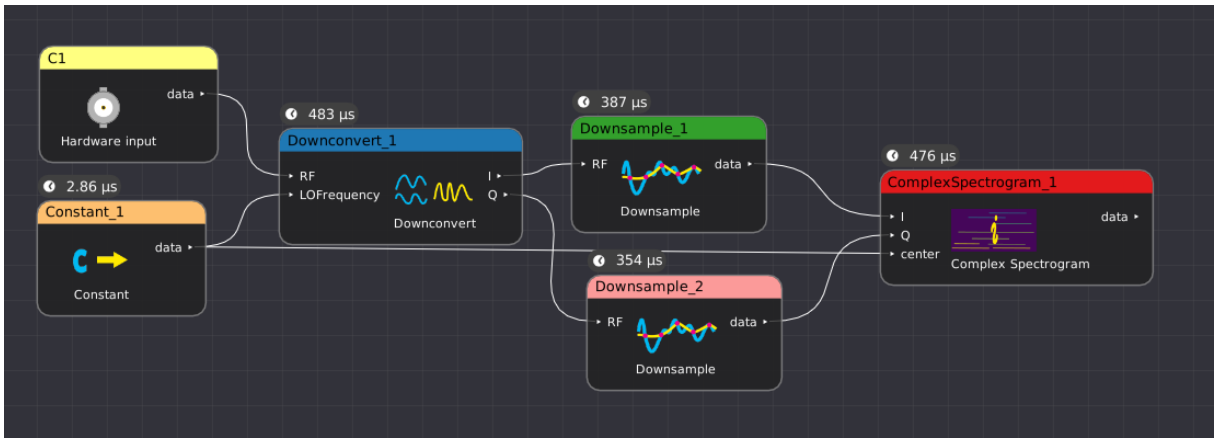



Figure 26.83: Example filter graph downconverting a 2.437 GHz Wi-Fi signal to zero IF, decimating and filtering, and displaying a spectrogram

[3] In the future, as we build out a more complete RF toolbox in ngscopeclient, we may merge some of these functions into the downconvert block to improve performance and ease setup. Expect the interface to this block to change in future versions.

[4] This is also likely not to be the final interface of the block; we are exploring various ways to unify parameters and scalar inputs and intend to merge them in some way in the future.

### 26.53.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| RF | Analog waveform | Directly sampled RF waveform |
| LOFrequency | Analog scalar | Optional LO frequency override |

### 26.53.2 Parameters

| Parameter name | Type | Description |
|---|---|---|
| LO Frequency | Int | Local oscillator frequency |

### 26.53.3 Output Signal

This filter outputs two analog waveforms containing the I/Q mix products.

## 26.54 Downsample

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

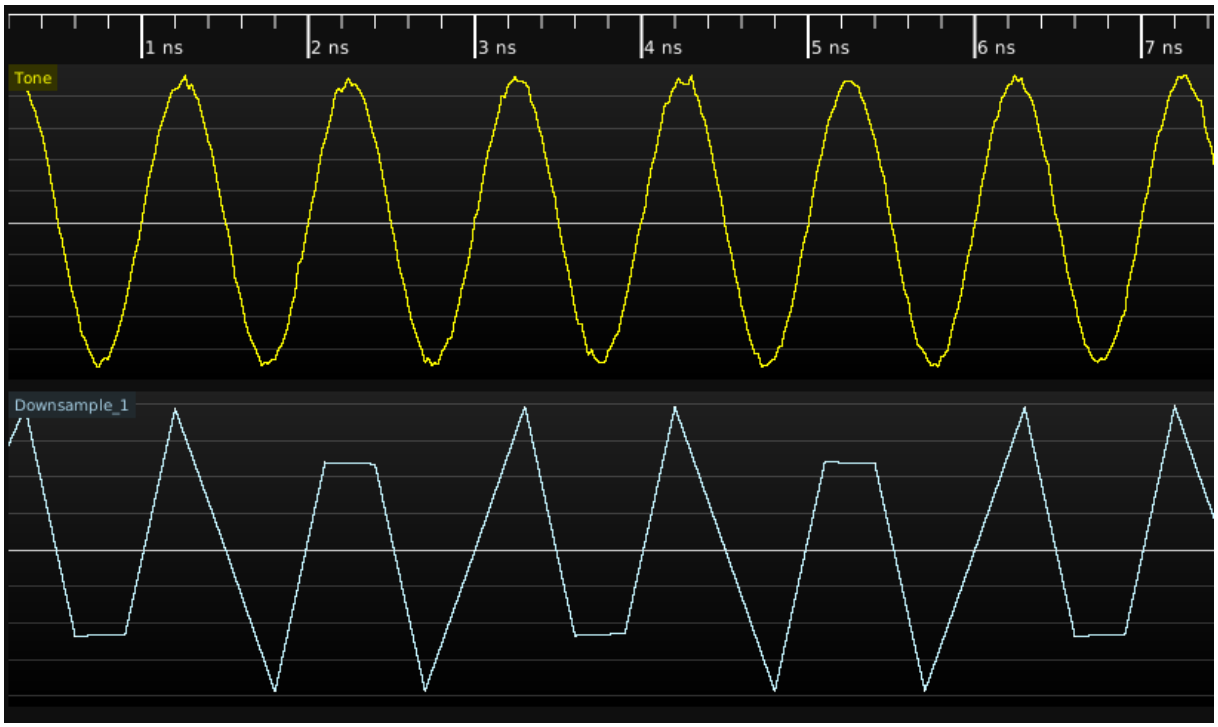



Figure 26.84: Example of downsampling a sinusoid

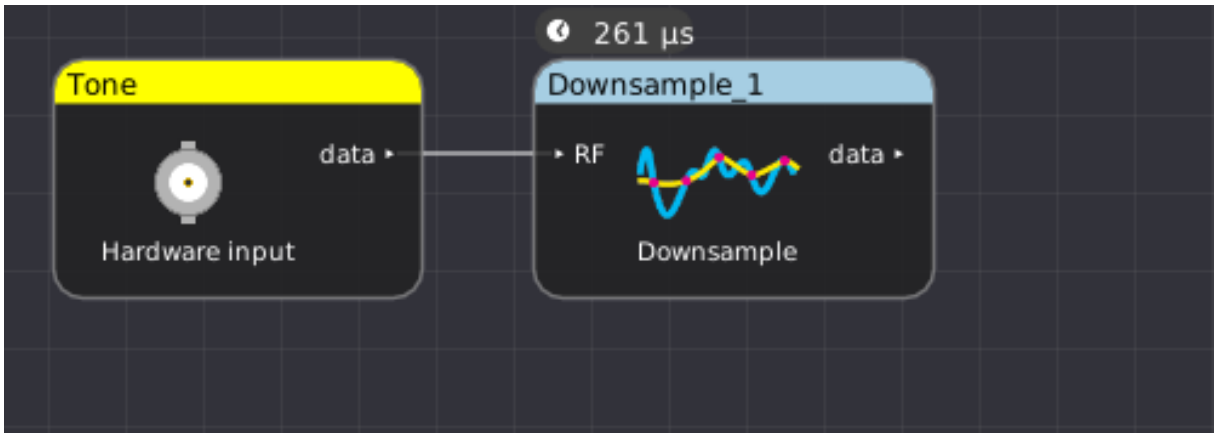



Figure 26.85: Example filter graph showing usage of downsample filter

### 26.54.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| RF | Analog | Input waveform |

### 26.54.2 Parameters

| Parameter name | Type | Description |
|---|---|---|
| Antialiasing Filter | Bool | If true, applies a low-pass filter to the input before decimating |
| Downsample Factor | Int | Divisor for output sample rate |

### 26.54.3 Output Signal

This filter outputs a decimated version of the input.

## 26.55 DRAM Clocks

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

(This filter description is a stub and will be expanded in the future)

## 26.56 DRAM Trcd

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

(This filter description is a stub and will be expanded in the future)

## 26.57 DRAM Trfc

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

(This filter description is a stub and will be expanded in the future)

## 26.58 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.

Analog waveforms are automatically thresholded at the average voltage.

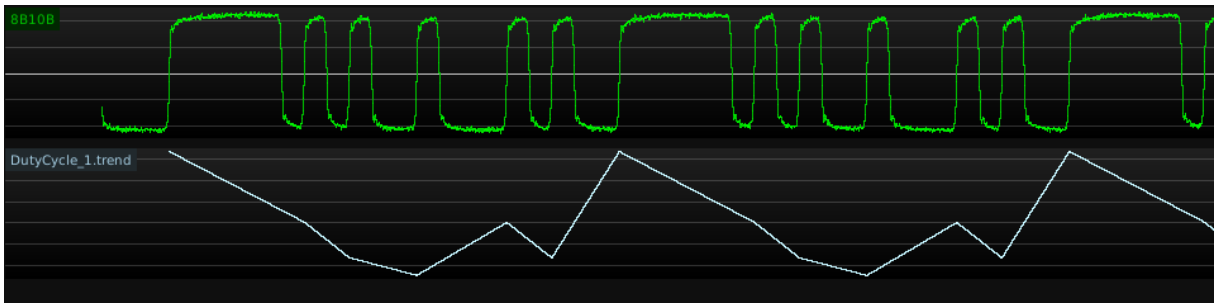



Figure 26.86: Example of measuring duty cycle of a digital waveform

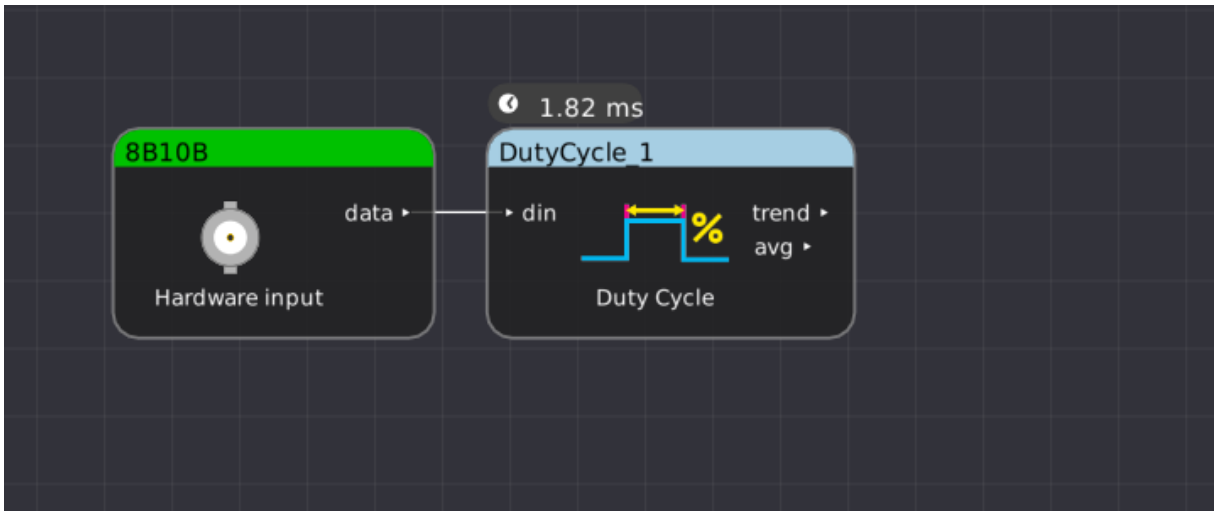



Figure 26.87: Example filter graph showing usage of duty cycle filter

### 26.58.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| din | Analog or digital | Input waveform |

### 26.58.2 Parameters

This filter takes no parameters.

### 26.58.3 Output Signal

| Signal name | Type | Description |
|---|---|---|
| trend | Analog | Instantaneous duty cycle of each cycle of the waveform |
| avg | Analog scalar | Average duty cycle over the entire input signal |

## 26.59 DVI

Decodes Digital Visual Interface (DVI) video signals.




Figure 26.88: Example of decoding a DVI signal

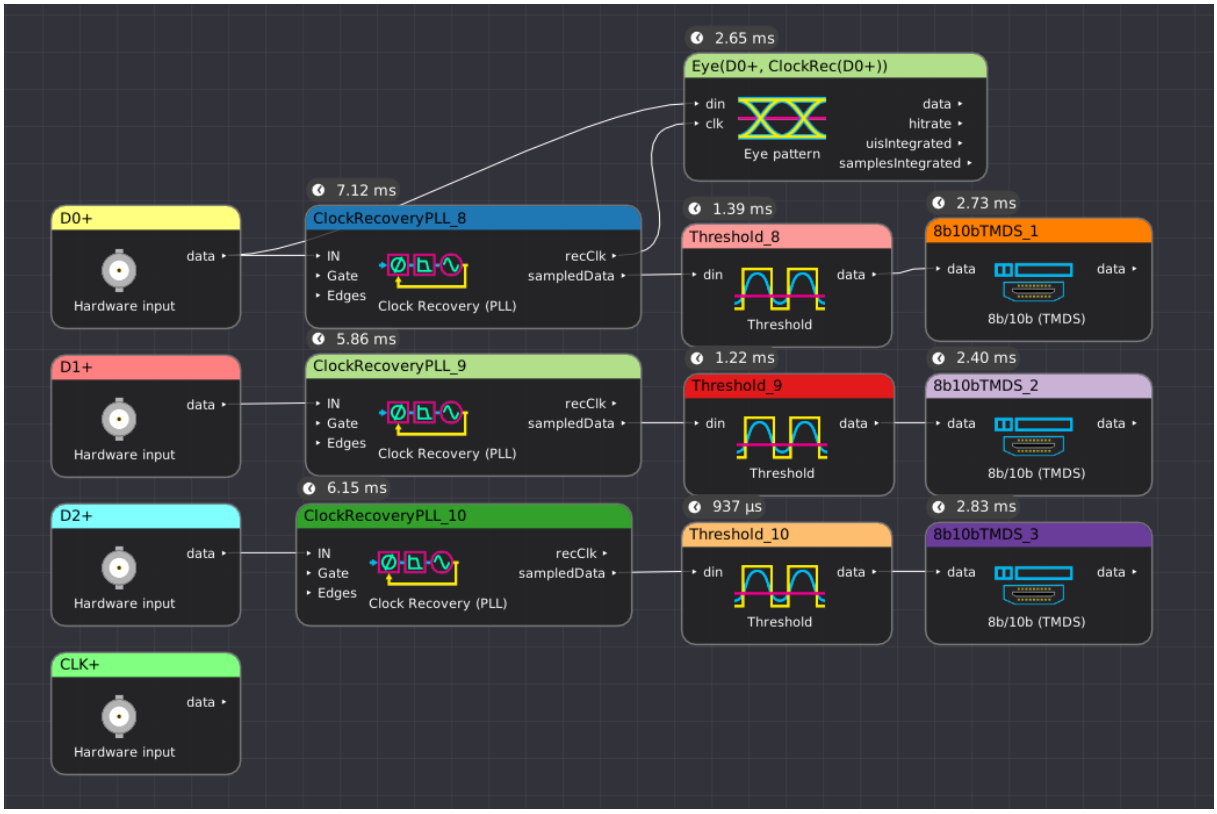



Figure 26.89: Example filter graph showing usage of DVI decode filter

### 26.59.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| D0 (blue) | Protocol | TMDS decode for lane 0 |
| D1 (green) | Protocol | TMDS decode for lane 1 |
| D2 (blue) | Protocol | TMDS decode for lane 2 |

### 26.59.2 Parameters

This filter takes no parameters.

### 26.59.3 Output Signal

The output is a sequence of DVI scan lines including RGB pixel values.

## 26.60 Emphasis

Adds pre/de emphasis to a signal by convolving it with a tapped delay line filter. This introduces a small phase shift.

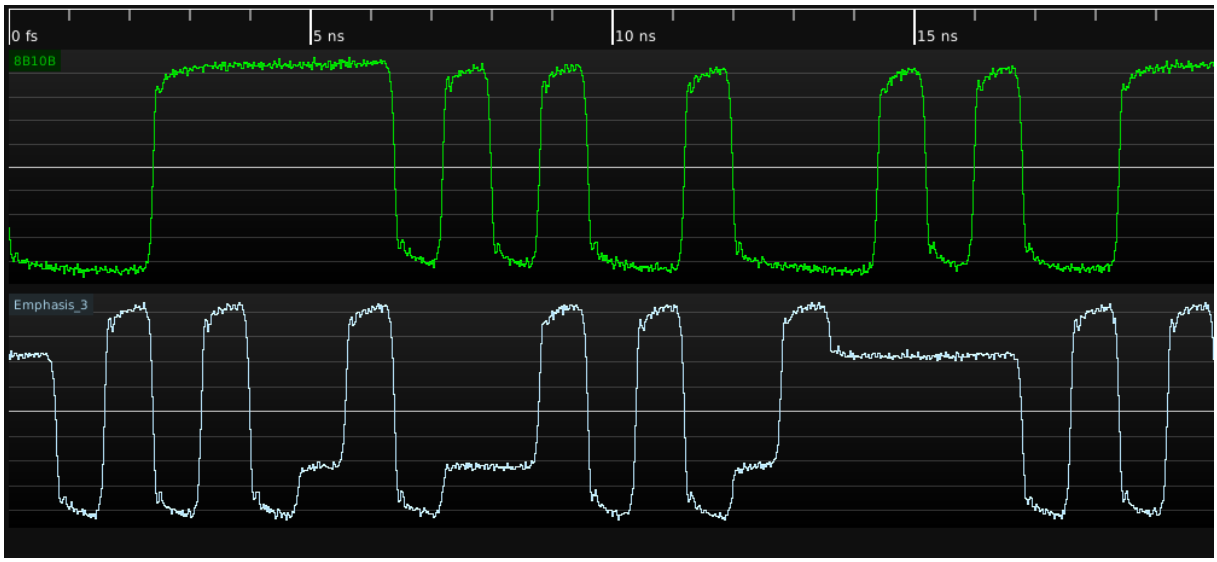



Figure 26.90: Example of applying 6 dB of de-emphasis to a 1.25 Gbps 8B/10B signal

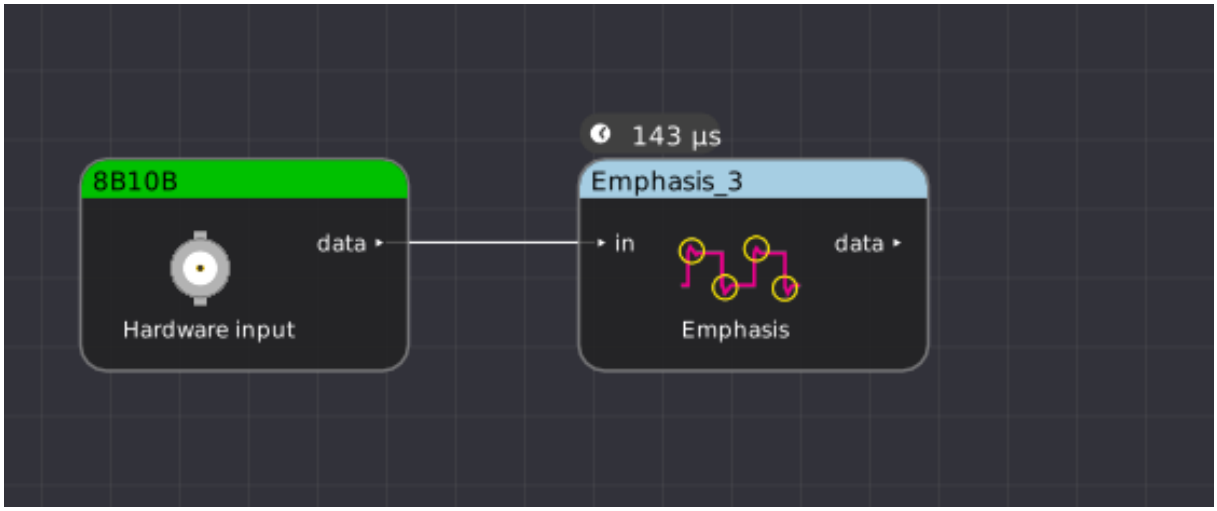



Figure 26.91: Example filter graph showing usage of emphasis filter

### 26.60.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| in | Analog | Input waveform |

### 26.60.2 Parameters

| Parameter name | Type | Description |
|---|---|---|
| Data Rate | Int | Data rate, in symbols per second |
| Emphasis Amount | Float | Amount of pre/de emphasis to add, in dB |
| Emphasis Type | Enum | Type of emphasis to add (pre-emphasis or de-emphasis) |

### 26.60.3 Output Signal

The output is an analog signal with the requested amount of emphasis applied.

## 26.61 Emphasis Removal

Removes pre/de emphasis from a signal by convolving it with a tapped delay line filter. This introduces a small phase shift.

This filter is typically used when probing a signal at the transmitter which has a high level of emphasis. Removing the emphasis can increase the eye opening in the digitized signal and make interpretation easier.

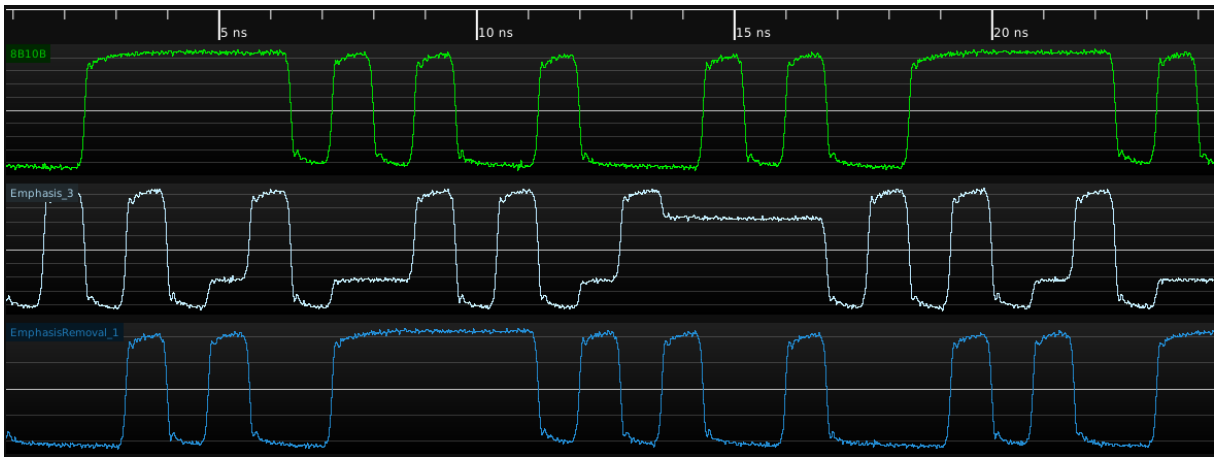



Figure 26.92: Example of applying 6 dB of de-emphasis to a 1.25 Gbps 8B/10B signal then removing it

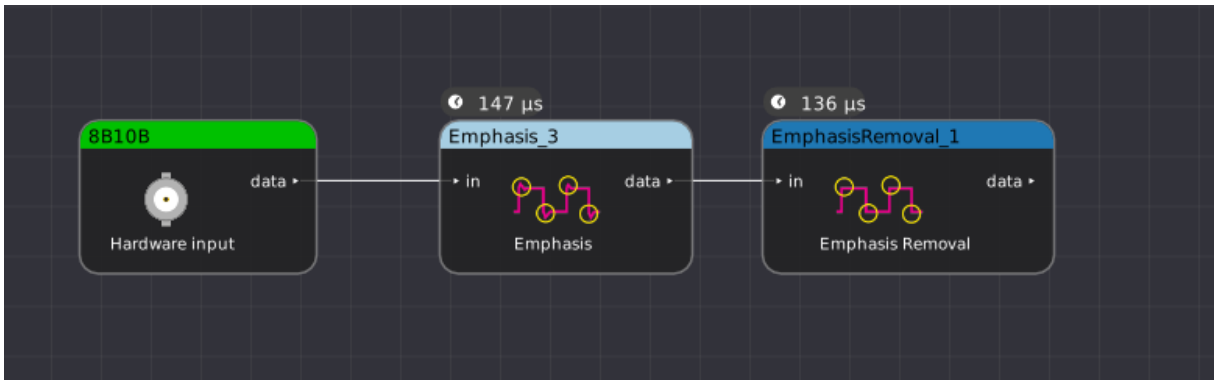



Figure 26.93: Example filter graph showing usage of emphasis removal filter

### 26.61.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| in | Analog | Input waveform |

### 26.61.2 Parameters

| Parameter name | Type | Description |
|---|---|---|
| Data Rate | Int | Data rate, in symbols per second |
| Emphasis Amount | Float | Amount of pre/de emphasis to remove, in dB |
| Emphasis Type | Enum | Type of emphasis to remove (pre-emphasis or de-emphasis) |

### 26.61.3 Output Signal

The output is an analog signal with the requested amount of emphasis removed.

## 26.62 Enhanced Resolution

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

Each half bit of resolution reduces the bandwidth by an additional factor of two beyond the Nyquist limit. For example, a 1.5 bit resolution improvement reduces the bandwith to $F_{nyquist}/8$, or 1/16 the sampling frequency. The filter properties dialog displays the calculated -3 dB bandwidth based on the current input sample rate.

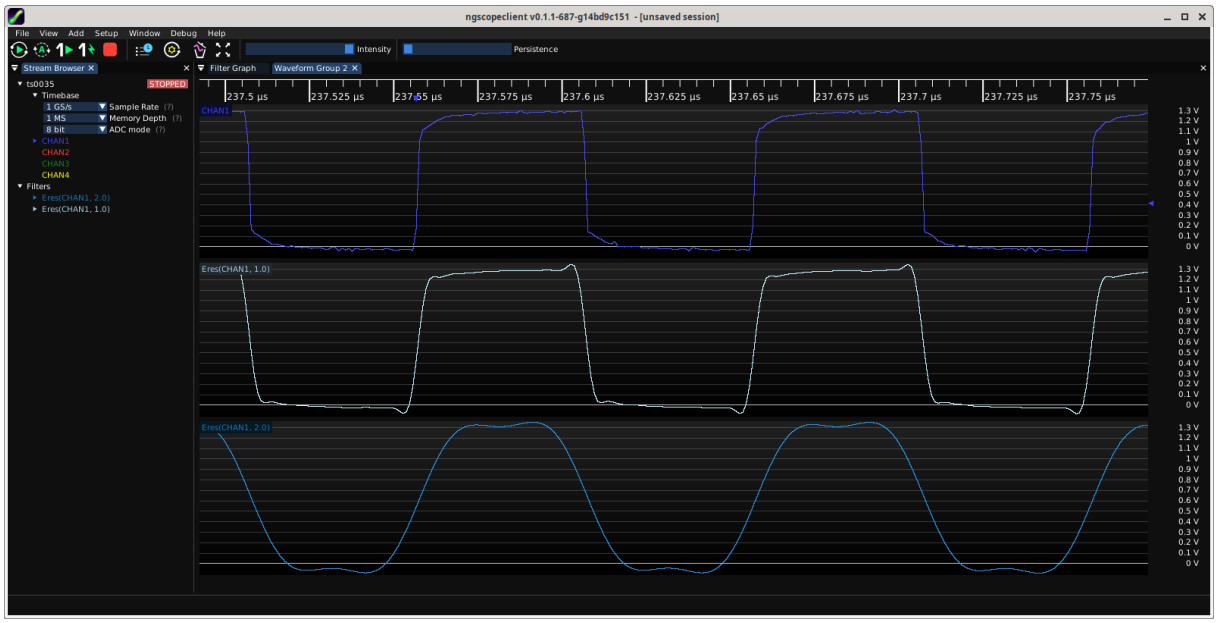



Figure 26.94: Example showing a squarewave with 1 and 2 bits of enhanced resolution

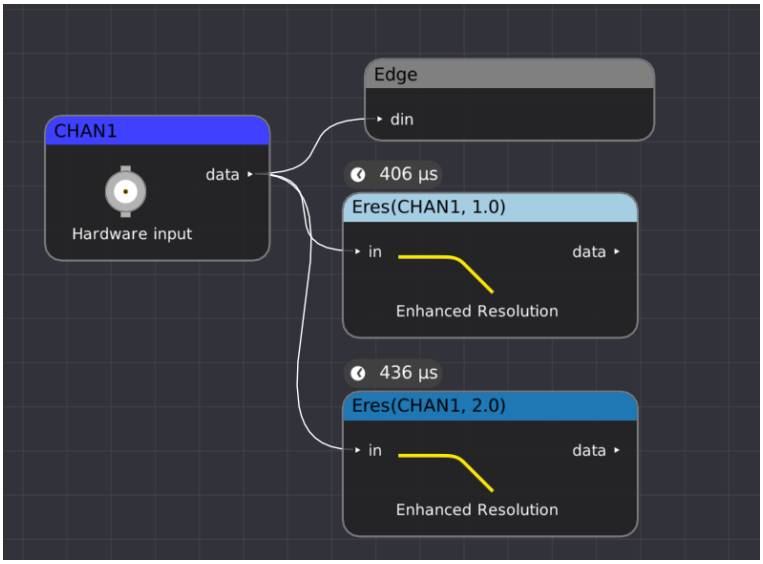



Figure 26.95: Example filter graph showing usage of enhanced resolution filter

### 26.62.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| in | Analog | Input signal |

### 26.62.2 Parameters

| Parameter name | Type | Description |
|---|---|---|
| Bits | Enum | Number of additional bits of resolution to add |
| Cutoff Frequency | Int | Calculated -3 dB cutoff of the filter |

### 26.62.3 Output Signal

The output is an analog signal containing the filtered version of the input.

## 26.63 Envelope

Finds the minimum and maximum of each sample in the input over time, and outputs them as separate streams. In other words, sample N of the output is the min/max of the Nth sample of all waveforms processed by the filter.

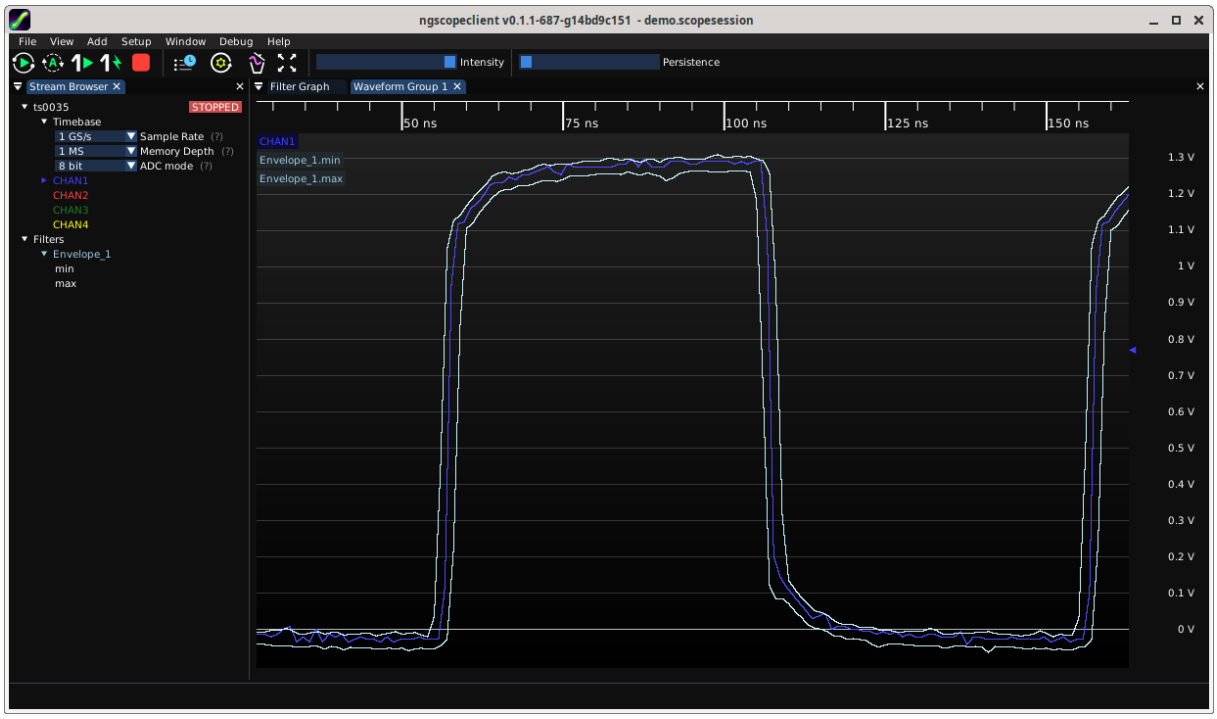



Figure 26.96: Example showing envelope of a signal

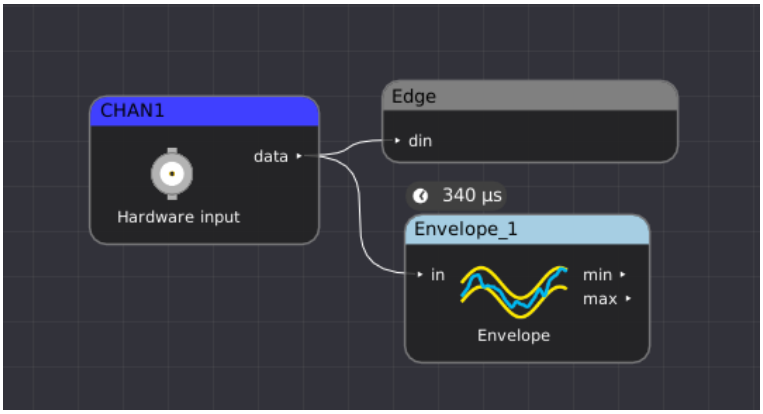



Figure 26.97: Example filter graph showing usage of envelope filter

### 26.63.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| in | Analog | Input signal |

### 26.63.2 Parameters

This filter takes no parameters.

### 26.63.3 Output Signal

The output is two analog signals called “min" and “max", containing the lower and upper envelope boundaries of the signal.

## 26.64 Ethernet - 10baseT

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

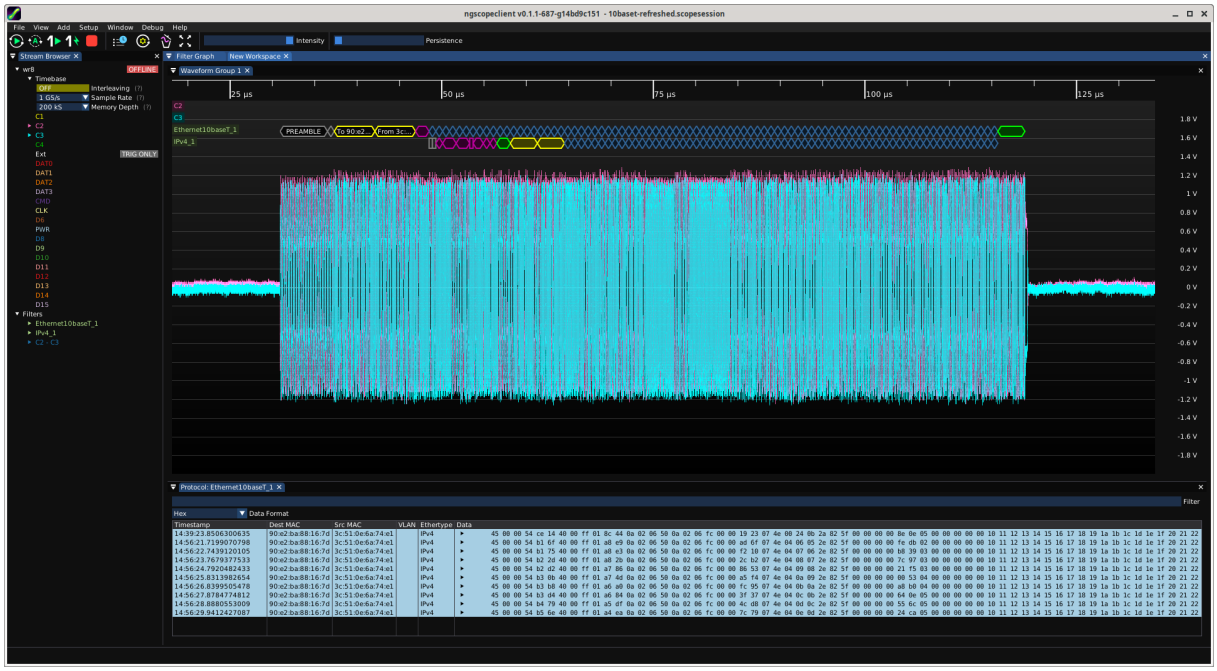


Figure 26.98: Example showing decode of a 10baseT signal

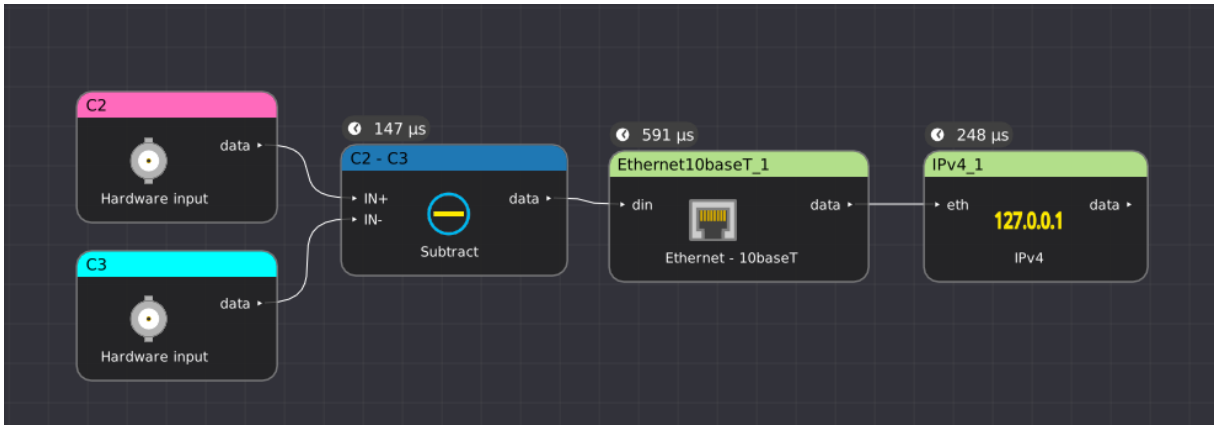



Figure 26.99: Example filter graph showing usage of 10baseT decode

### 26.64.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| din | Analog | Input signal |

### 26.64.2 Parameters

This filter takes no parameters.

### 26.64.3 Output Signal

This filter outputs a stream of Ethernet sample objects which may be passed to upper layer decodes or exported to a PCAPNG file.

## 26.65 Ethernet - 100baseT1

Decodes the 100base-T1 single-pair / automotive Ethernet PMA/PCS, as specified in IEEE 802.3-2018 clause 96.

Due to the bidirectional nature of 100baseT1 signaling, it is *not* possible to decode this protocol by simply placing probes on the differential pair. A dedicated test fixture with two dual directional couplers must be used, and there is typically enough crosstalk between the forward and reverse coupled path that the Coupler De-Embed filter (which requires connections to both the forward and reverse coupled ports of both differential legs) must be used to improve directivity.

Additionally, the 100baseT1 PHY uses 2D-PAM3 signaling which must be demuxed into separate I and Q channels by the IQ Demux block before decoding.

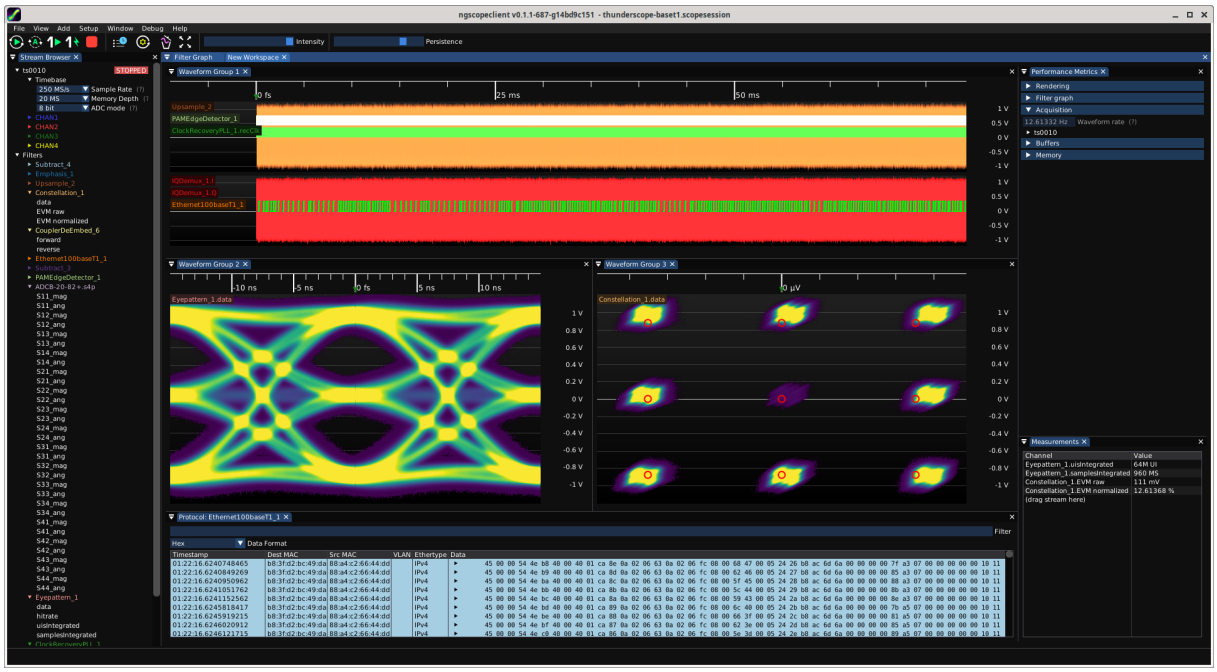


Figure 26.100: Example showing decode of a 100baseT1 signal

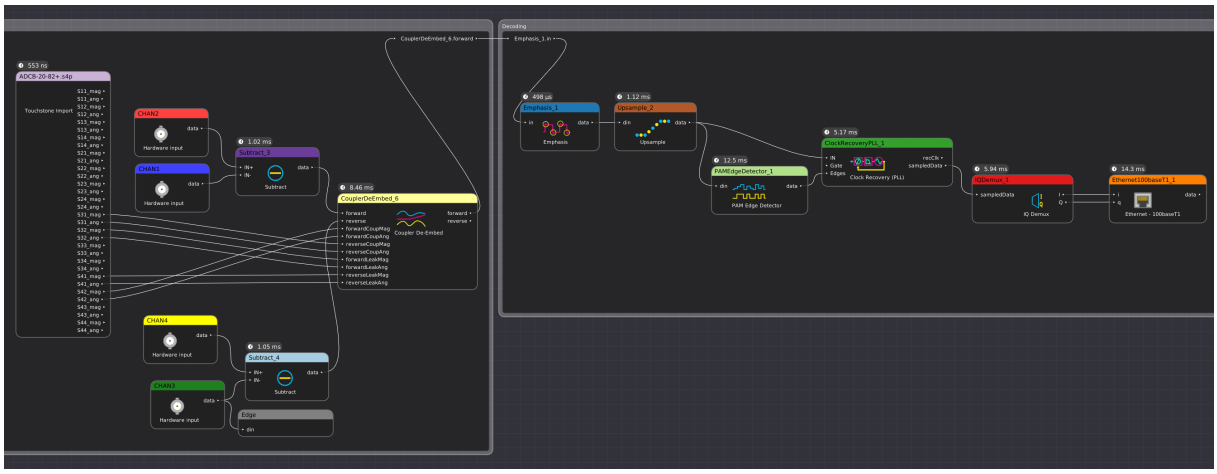


Figure 26.101: Example filter graph showing usage of 100baseT1 decode

### 26.65.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| i | Analog | Input signal, I channel |
| q | Analog | Input signal, Q channel |

### 26.65.2 Parameters

| Parameter name | Type | Description |
|---|---|---|
| Scrambler Polynomial | Enum | Select the correct LFSR polynomial depending on which side of the link is being decoded |
| Threshold I+ | Float | Decision threshold between 0 and +1 states on the I channel |
| Threshold I- | Float | Decision threshold between -1 and 0 states on the I channel |
| Threshold Q+ | Float | Decision threshold between 0 and +1 states on the Q channel |
| Threshold Q- | Float | Decision threshold between -1 and 0 states on the Q channel |

### 26.65.3 Output Signal

This filter outputs a stream of Ethernet sample objects which may be passed to upper layer decodes or exported to a PCAPNG file.

### 26.65.4 Breaking Changes

This section lists major interface changes that may require projects to be updated.

- **v0.2:** This filter no longer needs the clock input that was required in v0.1. Instead, simply connect to the I/Q demux output

## 26.66 Ethernet - 100baseT1 Link Training

Decodes the link training stage of 100base-T1 single-pair / automotive Ethernet, as specified in IEEE 802.3-2018 clause 96. As with the 100baseT1 decode, this filter requires special fixturing to acquire the necessary signals for a decode.

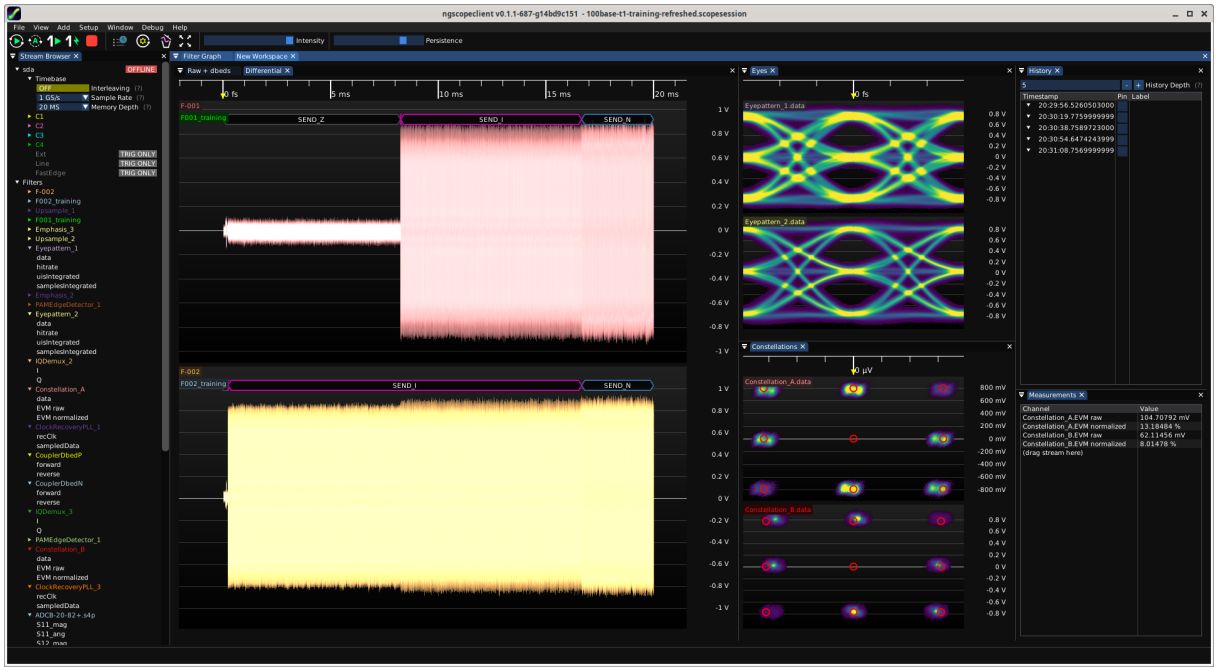


Figure 26.102: Example showing decode of a 100baseT1 training signal

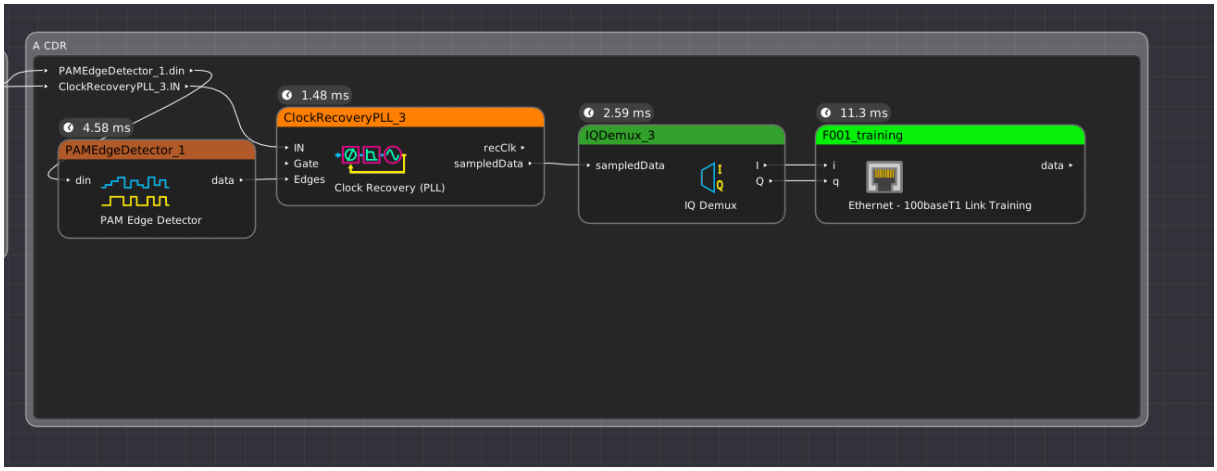



Figure 26.103: Example filter graph showing usage of 100baseT1 training decode

### 26.66.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| i | Analog | Input signal, I channel |
| q | Analog | Input signal, Q channel |

### 26.66.2 Parameters

| Parameter name | Type | Description |
|---|---|---|
| Scrambler Polynomial | Enum | Select the correct LFSR polynomial depending on which side of the link is being decoded |

### 26.66.3 Output Signal

This filter outputs a stream of decode state values indicating whether the link is in SEND_Z, SEND_I, or SEND_N state.

### 26.66.4 Breaking Changes

This section lists major interface changes that may require projects to be updated.

- **v0.2:** This filter no longer needs the clock input that was required in v0.1. Instead, simply connect to the I/Q demux output

## 26.67 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.

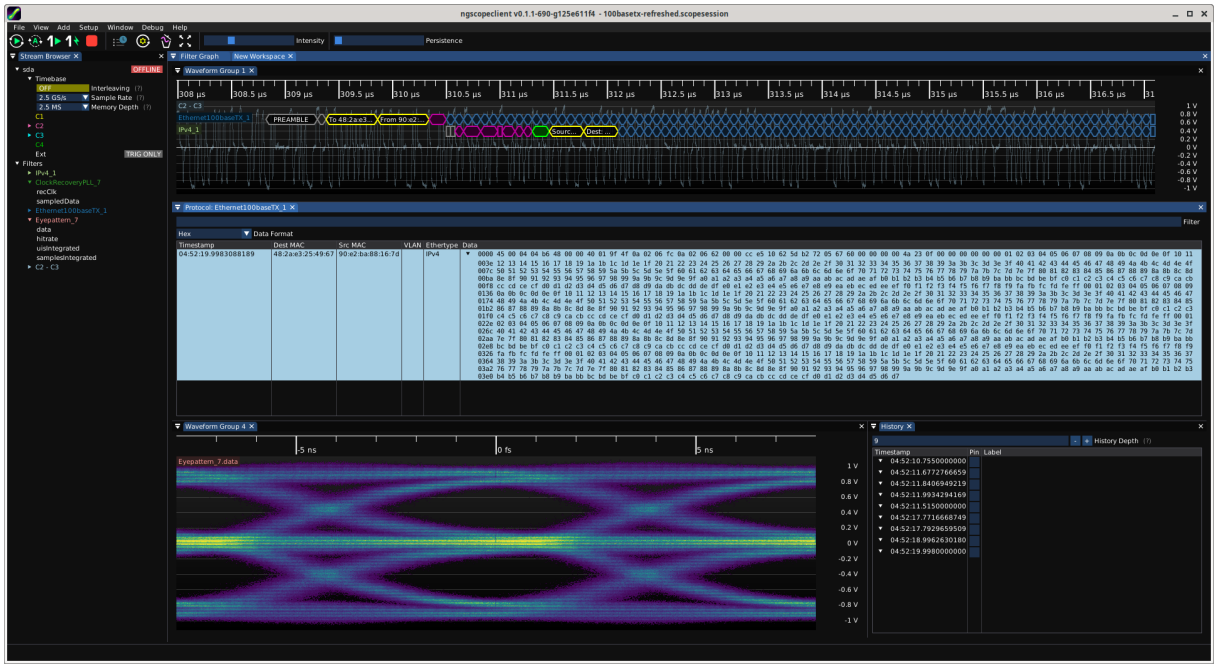


Figure 26.104: Example showing decode of a 100baseTX signal

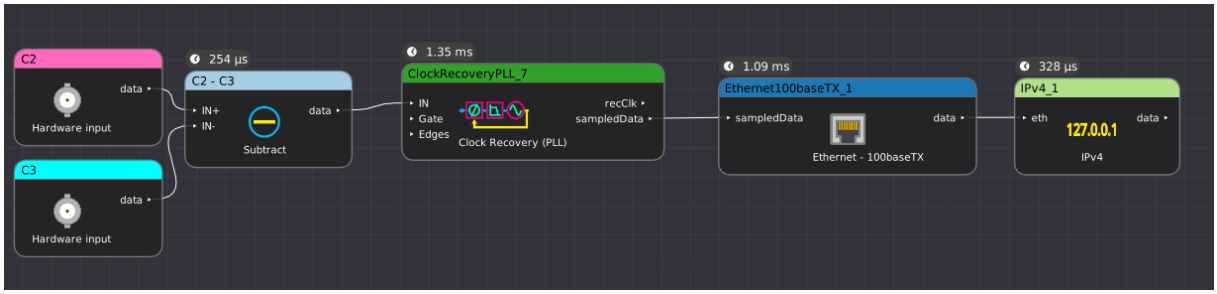



Figure 26.105: Example filter graph showing usage of 100baseTX decode

### 26.67.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| sampledData | Analog | Sampled data output from CDR PLL |

### 26.67.2 Parameters

This filter takes no parameters.

### 26.67.3 Output Signal

This filter outputs a stream of Ethernet sample objects which may be passed to upper layer decodes or exported to a PCAPNG file.

### 26.67.4 Breaking Changes

This section lists major interface changes that may require projects to be updated.

- **v0.2:** This filter no longer needs the clock input that was required in v0.1. Instead, simply pass the sampledData output of the Clock Recovery (PLL) directly to the 100baseTX decode, as shown in Fig. 26.105.

## 26.68 Ethernet - 1000baseX

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

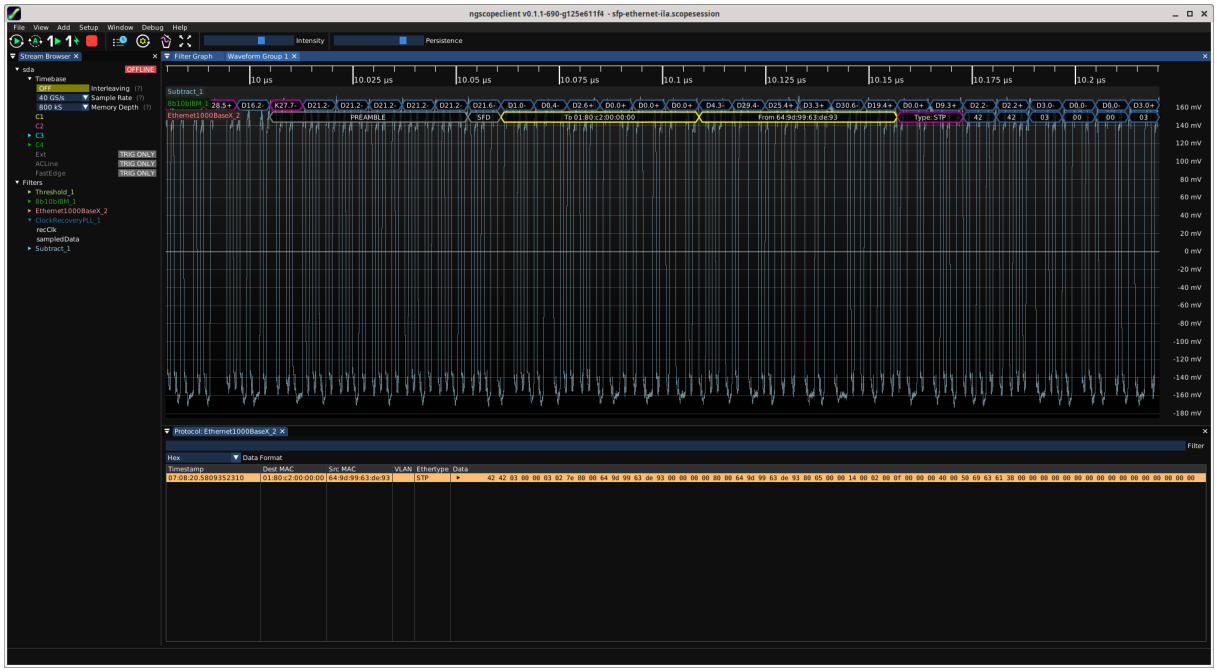


Figure 26.106: Example showing decode of a 1000baseX signal

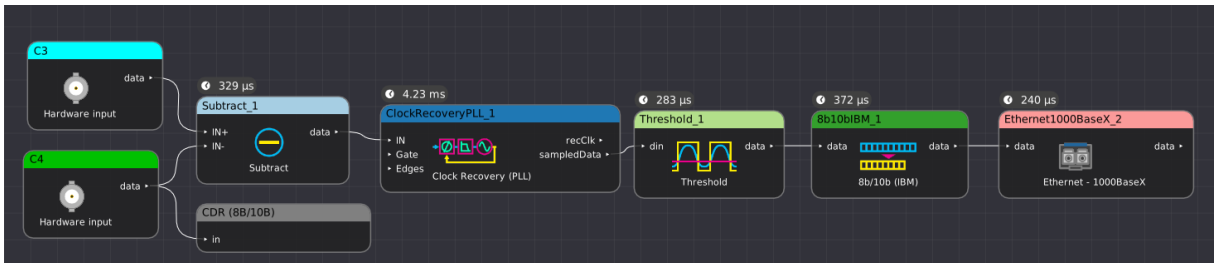



Figure 26.107: Example filter graph showing usage of 1000baseX decode

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

### 26.68.1 Parameters

This filter takes no parameters.

### 26.68.2 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<br>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

## 26.69 Ethernet - 10Gbase-R

Decodes the 10Gbase-R Ethernet PCS as specified in IEEE 802.3-2018 clause 49.

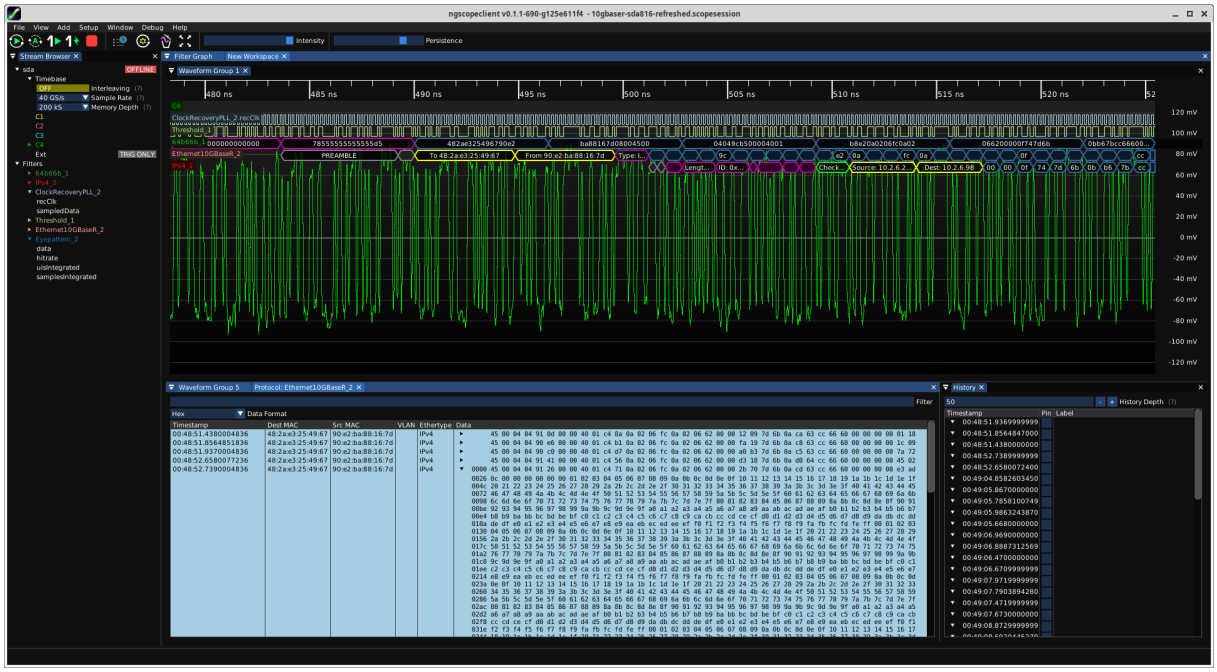


Figure 26.108: Example showing decode of a 10Gbase-R signal

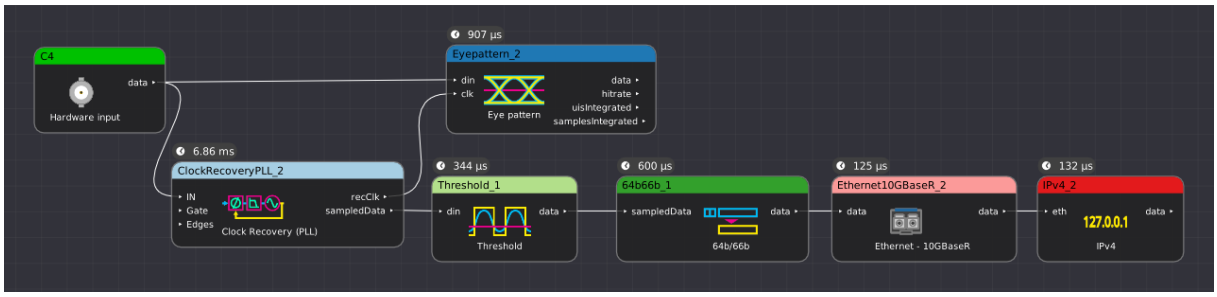



Figure 26.109: Example filter graph showing usage of 10Gbase-R decode

| Signal name | Type | Description |
|---|---|---|
| data | 64b/66b | Output of 64b/66b protocol decode |

### 26.69.1 Parameters

This filter takes no parameters.

### 26.69.2 Output Signal

This filter outputs a stream of Ethernet sample objects which may be passed to upper layer decodes or exported to a PCAPNG file.

## 26.70 Ethernet - QSGMII

Converts a 5 Gbps Quad SGMII data stream for a multiport gigabit Ethernet PHY into four separate 1.25 Gbps SGMII data streams, containing data for each of the PHY ports, which can be independently decoded.

Note that unlike most other Ethernet decodes, this block does *not* decode QSGMII up to Ethernet frames; it only demuxes the QSGMII to 4x SGMII. You can then use the SGMII decode block to decode any or all of the individual PHY lanes.



Figure 26.110: Example of decoding TX and RX lanes of a QSGMII PHY

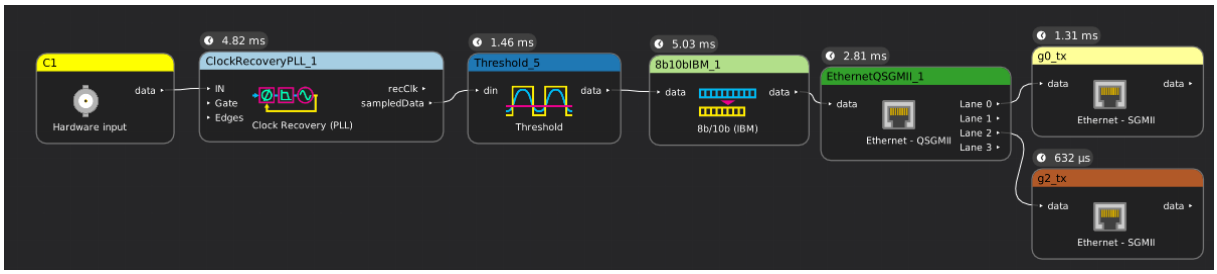



Figure 26.111: Example filter graph showing usage of QSGMII filter

### 26.70.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| data | Protocol | QSGMII Input waveform |

### 26.70.2 Parameters

| Parameter name | Type | Description |
|---|---|---|
| Display Format | Enum | Specifies Dx.x or hexadecimal format for 8B/10B output |

### 26.70.3 Output Signal

The output is four separate 8B/10B data streams suitable for feeding to the SGMII decode filter

| Signal name | Type | Description |
|---|---|---|
| Lane 0 | Protocol | SGMII data stream for PHY 0 |
| Lane 1 | Protocol | SGMII data stream for PHY 1 |
| Lane 2 | Protocol | SGMII data stream for PHY 2 |
| Lane 3 | Protocol | SGMII data stream for PHY 3 |

## 26.71 Ethernet - RGMII

Decodes the Reduced Gigabit Media Independent Interface as specified in the RGMII 2.0 specification. This filter assumes that the clock and data are center-aligned such that the clock can be used directly to sample the data without additional phase shifting. You may need to use instrument-side skew correction or Deskew block to adjust timing on the clock if you are using PHY-internal delays and have an edge-aligned clock.

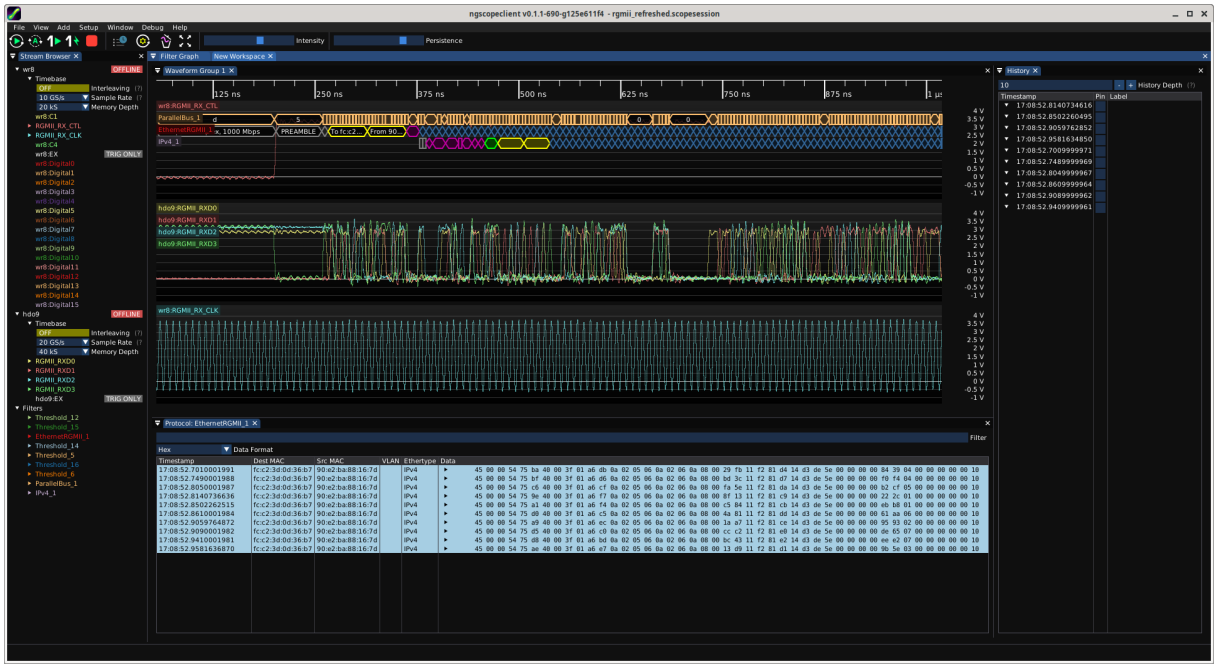


Figure 26.112: Example of decoding RGMII

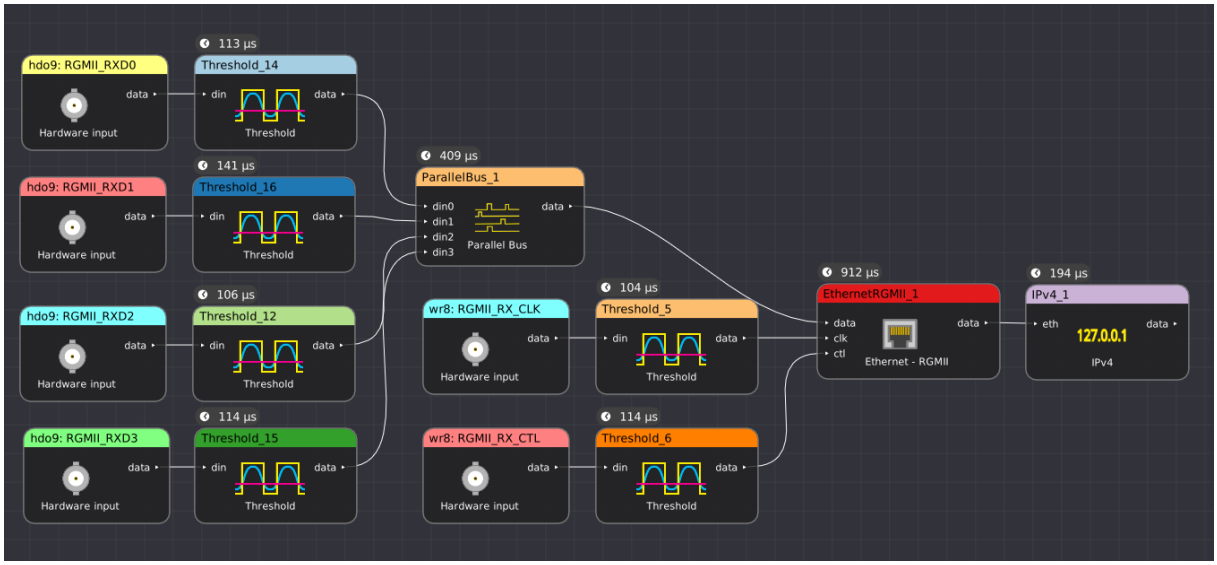



Figure 26.113: Example filter graph showing usage of RGMII filter

### 26.71.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| data | Digital bus | 4-bit data bus |
| clk | Digital | RGMII clock |
| ctl | Digital | RGMII TX_CTL or RX_CTL signal |

### 26.71.2 Parameters

This filter takes no parameters.

### 26.71.3 Output Signal

This filter outputs a stream of Ethernet sample objects which may be passed to upper layer decodes or exported to a PCAPNG file. In-band status is also decoded.

## 26.72 Ethernet - RMII

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

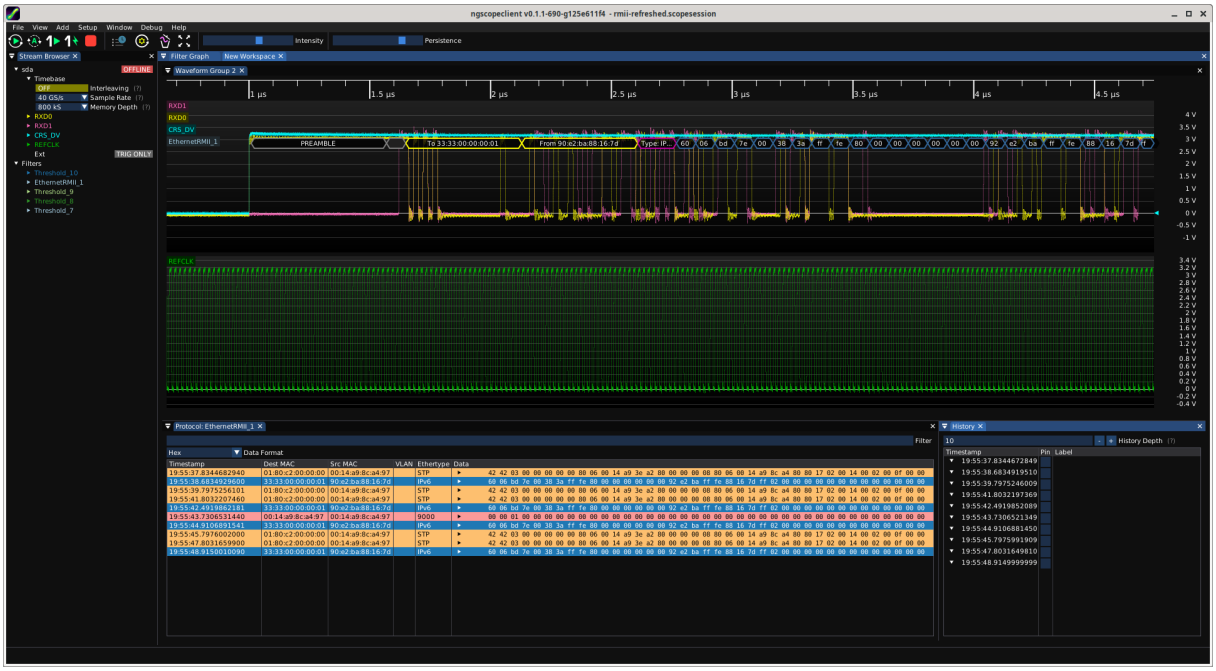


Figure 26.114: Example of decoding RMII

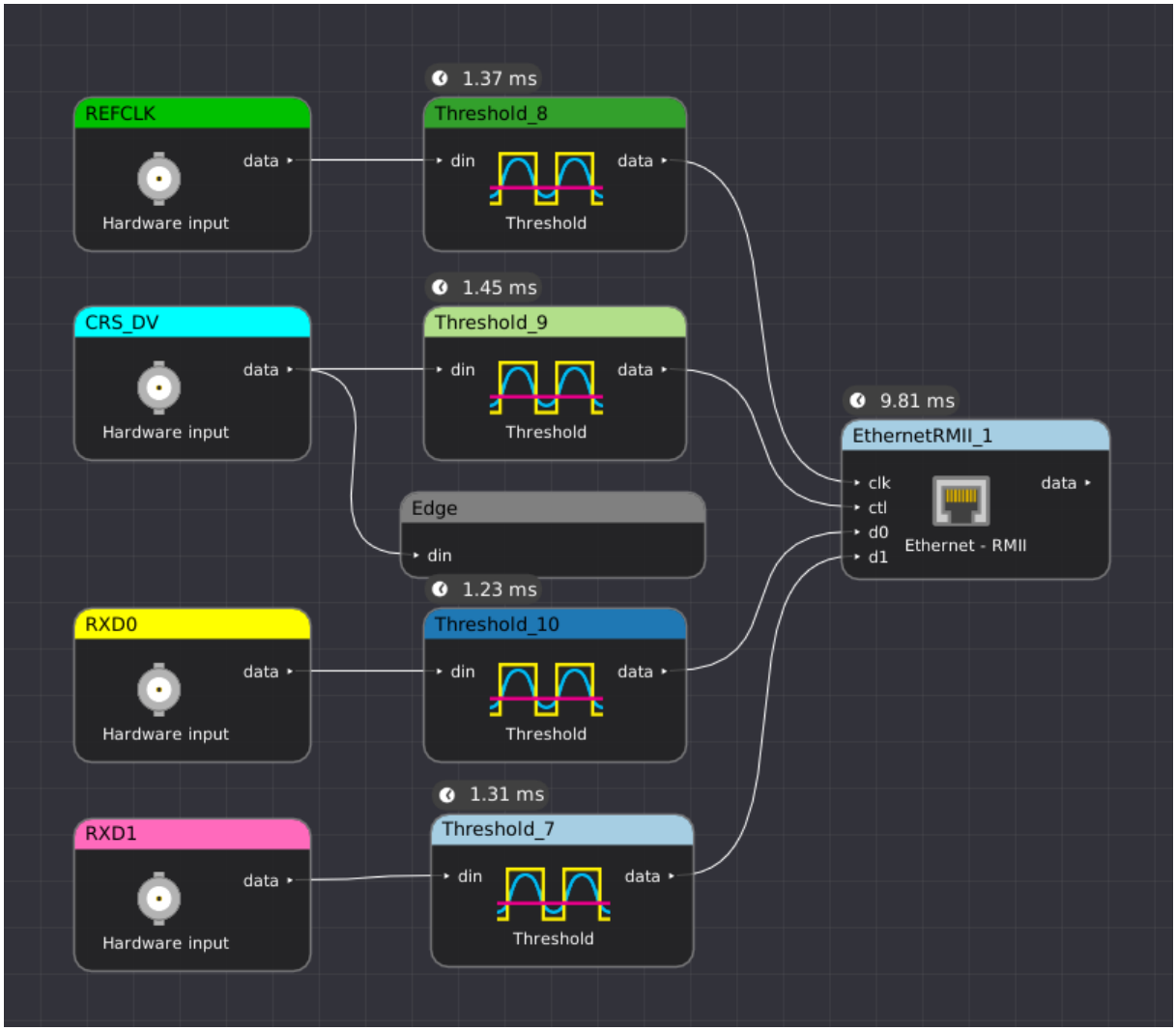



Figure 26.115: Example filter graph showing usage of RMII filter

### 26.72.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| clk | Digital | RMII system clock |
| ctl | Digital | Enable / control line (CRS_DV or TX_EN depending on direction) |
| d0 | Digital | Data bit 0 |
| d1 | Digital | Data bit 1 |

### 26.72.2 Parameters

This filter takes no parameters.

### 26.72.3 Output Signal

This filter outputs a stream of Ethernet sample objects which may be passed to upper layer decodes or exported to a PCAPNG file.

## 26.73 Ethernet - SGMII

Decodes Serial GMII data at 10, 100, or 1000 Mbps rates to Ethernet frames. Manual configuration of the data rate is currently required: SGMII always runs at 1.25 Gbps at the physical layer and simply repeats 8b/10b symbols 10 or 100 times at lower data rates. Automatic detection of data rate may be added in the future by parsing autonegotiation traffic or looking at the number of preamble bytes between the K27.7 start token and the D21.6 start-of-frame marker.

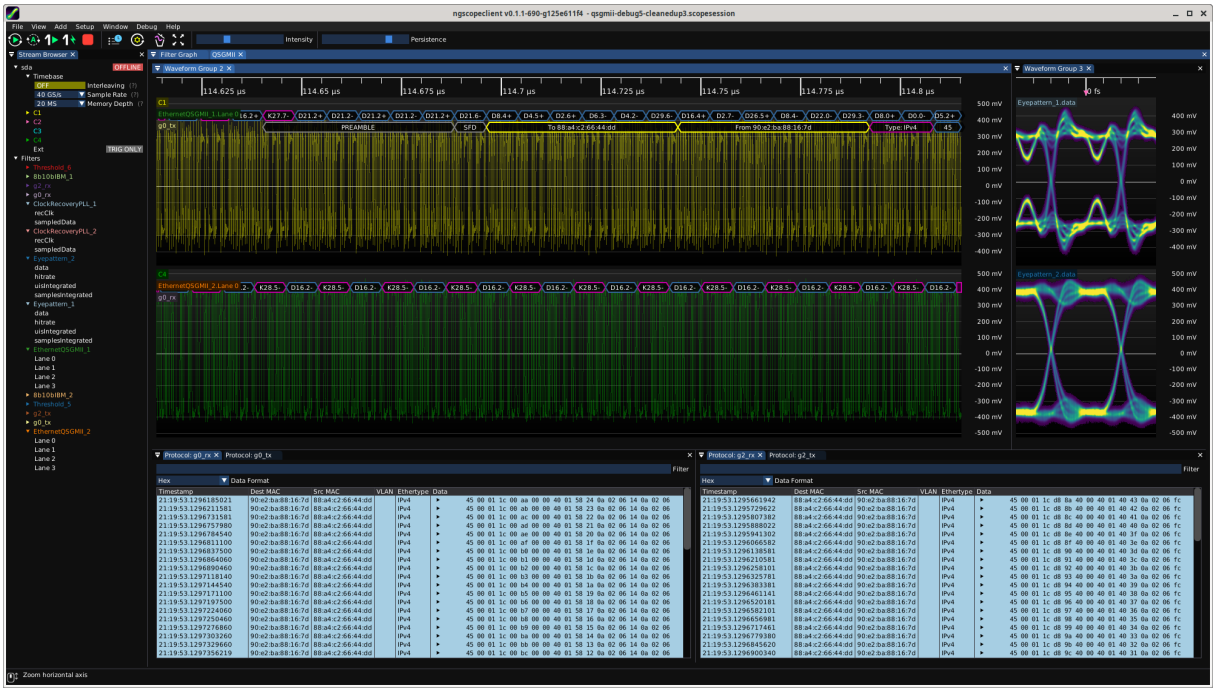


Figure 26.116: Example of decoding SGMII

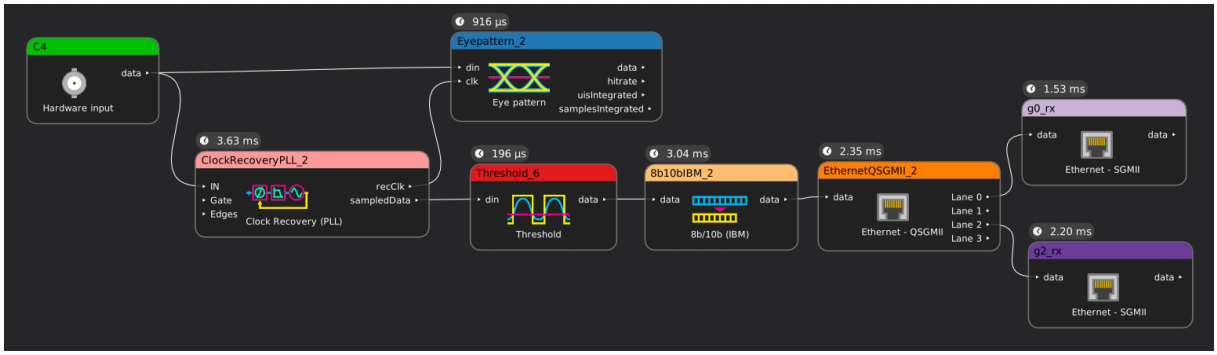



Figure 26.117: Example filter graph showing usage of SGMII filter

### 26.73.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| data | 8B/10B | Coded input symbols |

### 26.73.2 Parameters

| Parameter name | Type | Description |
|---|---|---|
| Speed | Enum | The current operating speed of the link |

### 26.73.3 Output Signal

This filter outputs a stream of Ethernet sample objects which may be passed to upper layer decodes or exported to a PCAPNG file.

## 26.74 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.

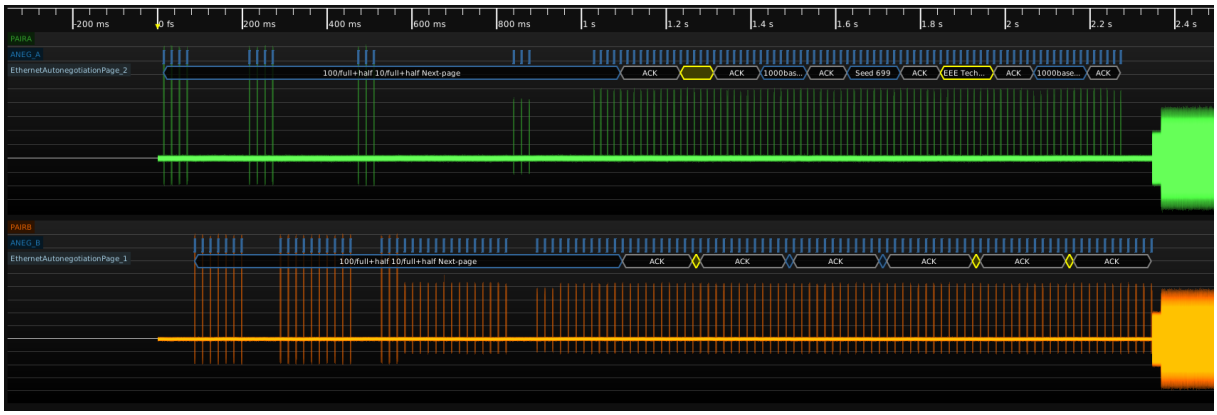



Figure 26.118: Example of decoding Ethernet autonegotiation traffic

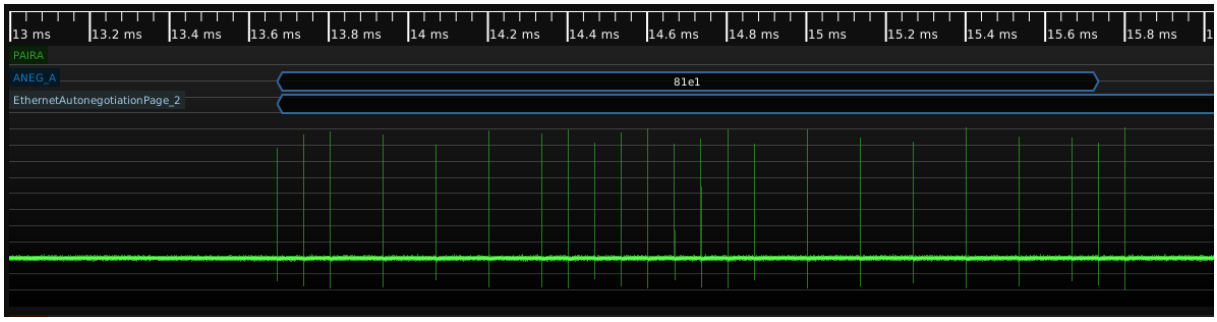



Figure 26.119: Decoding a single 16-bit link code word

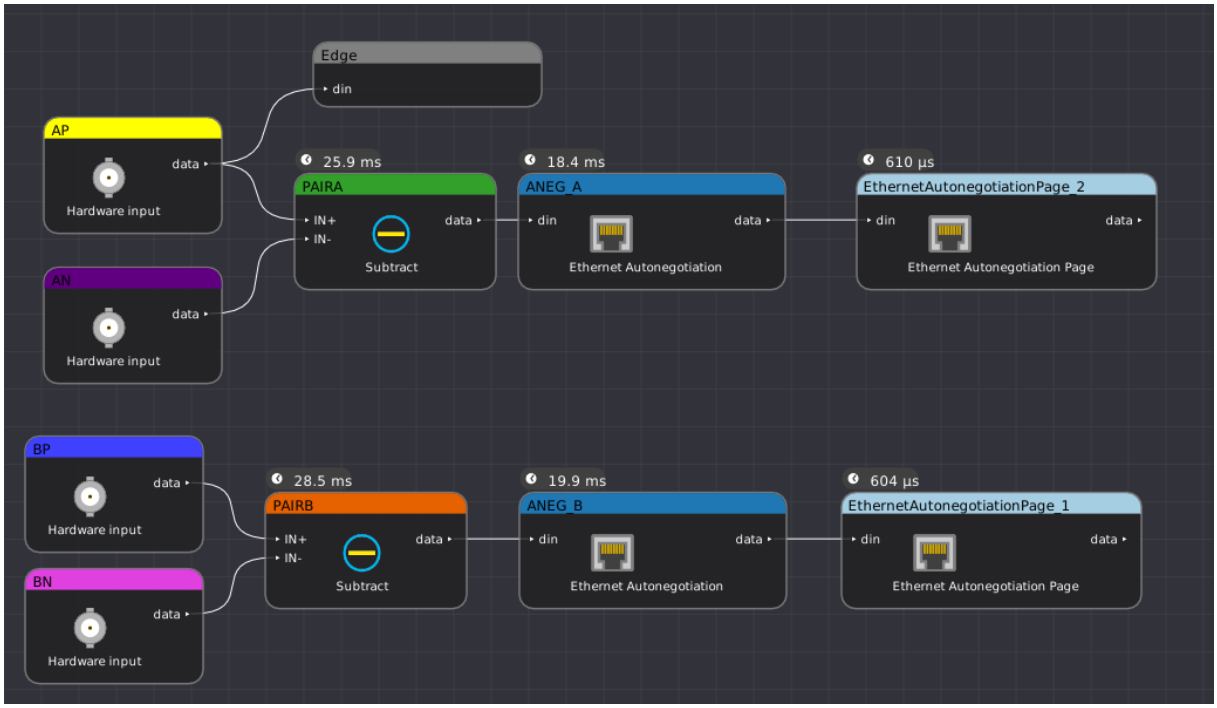



Figure 26.120: Example filter graph showing usage of autonegotiation filter

### 26.74.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| din | Analog | Ethernet differential pair waveform |

### 26.74.2 Parameters

This filter takes no parameters.

### 26.74.3 Output Signal

This filter outputs a stream of 16-bit link codewords.

## 26.75 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).

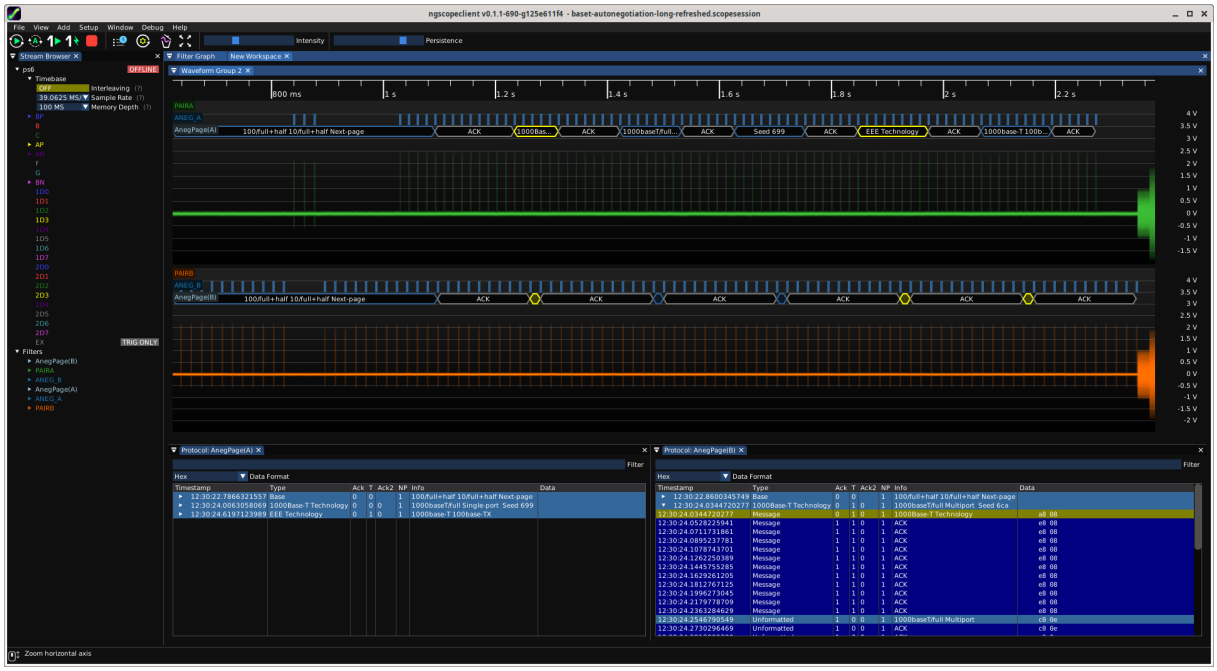



Figure 26.121: Example of decoding Ethernet autonegotiation traffic

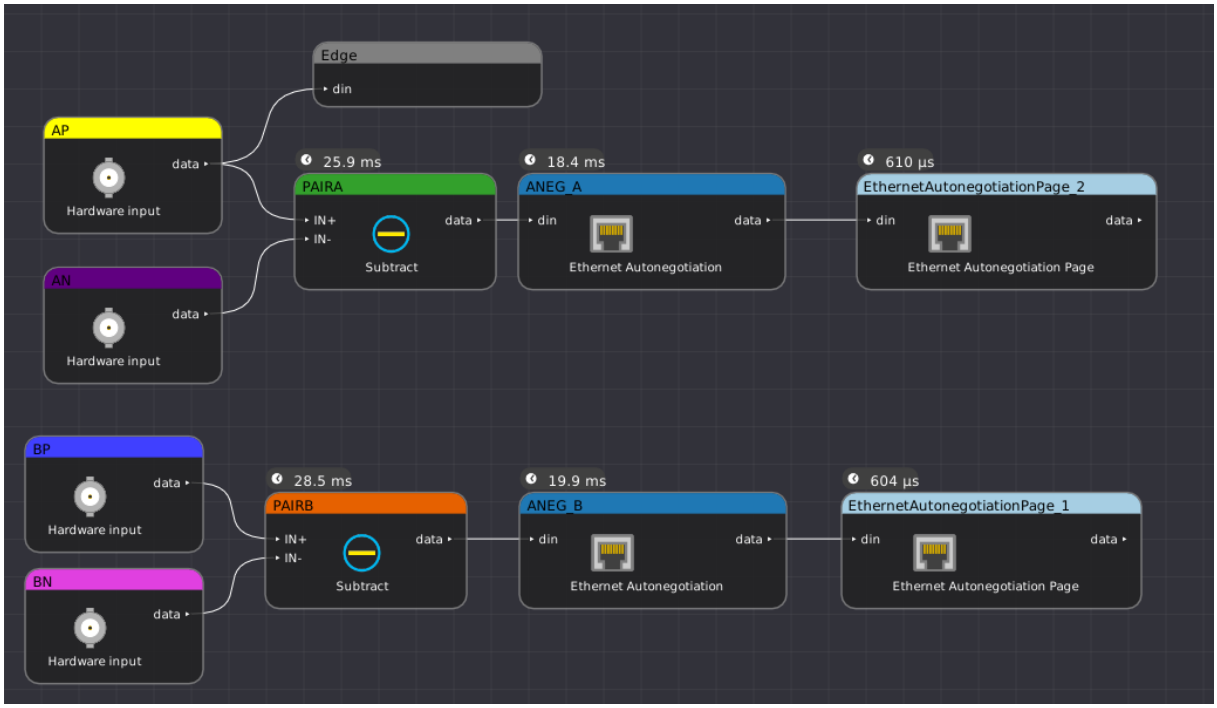



Figure 26.122: Example filter graph showing usage of autonegotiation page filter

### 26.75.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| din | Protocol | Link codewords from the Ethernet autonegotiation filter |

### 26.75.2 Parameters

This filter takes no parameters.

### 26.75.3 Output Signal

This filter outputs a stream of decoded PHY ability values.

## 26.76 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.

(This filter description is a stub and will be expanded in the future)

## 26.77 Ethernet Clause 73 Autonegotiation

Decodes the Base-KX autonegotiation signaling for Ethernet as specified in IEEE 802.3-2018 clause 73.

(This filter description is a stub and will be expanded in the future)

## 26.78 Exponential Moving Average

Calculates an exponential moving average of the input waveform, averaging the data at each sample index with the previous values of the same over multiple consecutive acquisitions.

The average is calculated recursively; for sample value $S$ and half life $T$, the recurrence relation is:

$$Out_i = \left(1 - \frac{1}{2^{\frac{1}{x}}}\right)(Out_{i-1}) + \left(\frac{1}{2^{\frac{1}{x}}}\right)(S)$$

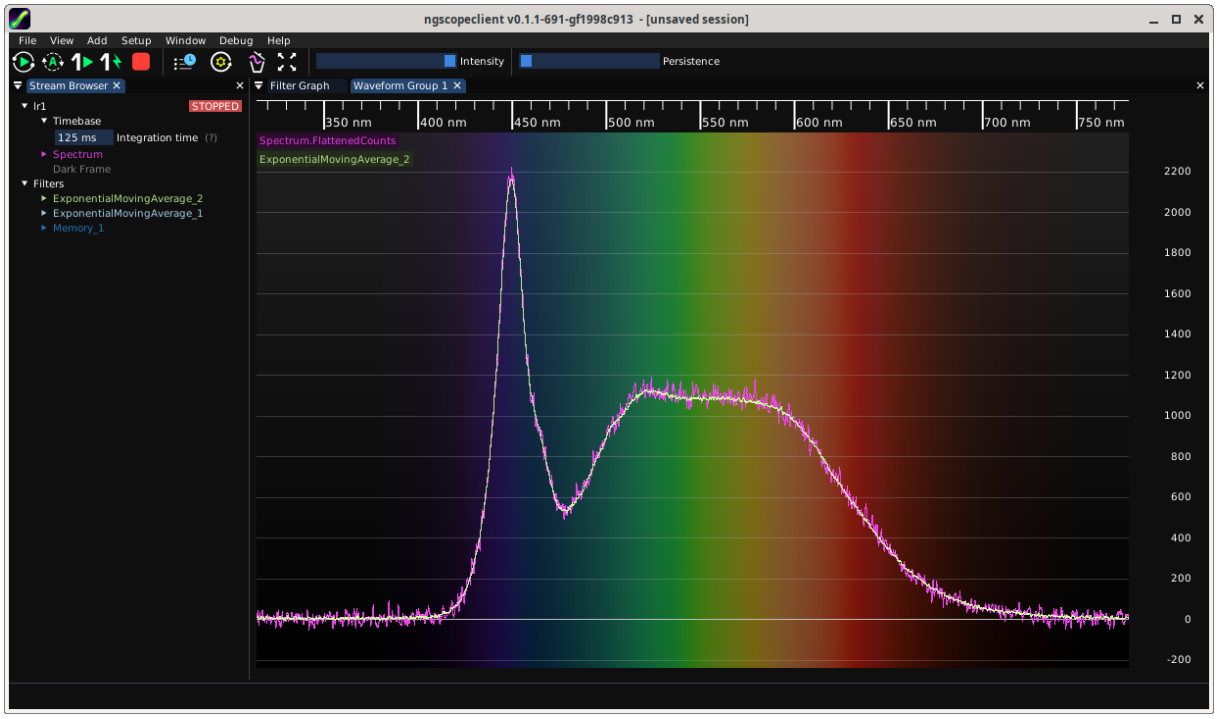



Figure 26.123: Example of using averaging to reduce noise on data from a UV/VIS/NIR spectrometer

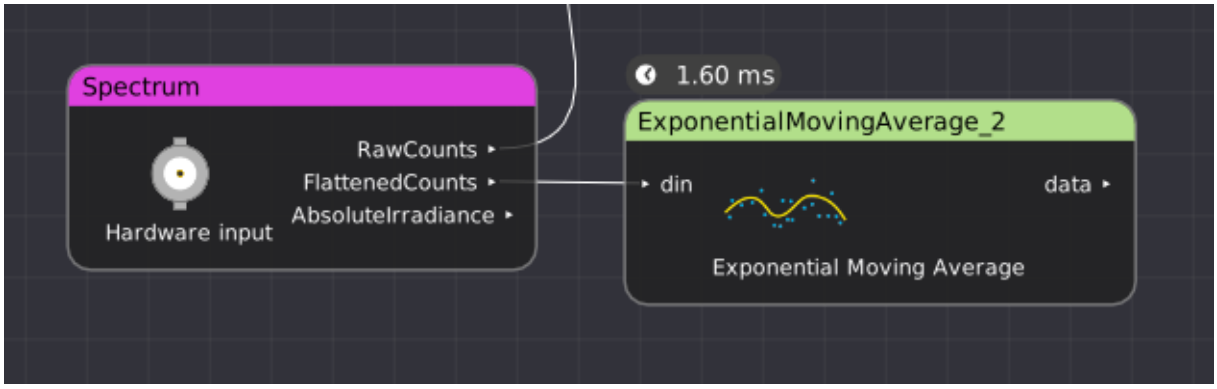



Figure 26.124: Example filter graph showing usage of exponential moving average filter

### 26.78.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| din | Analog | Input signal |

### 26.78.2 Parameters

| Parameter name | Type | Description |
|---|---|---|
| Half-life | Integer | Half life of the average, in waveforms |

### 26.78.3 Output Signal

This filter outputs an analog waveform with one sample per sample of the input, containing the averaged waveform.

## 26.79 Eye Bit Rate

Measures the bit rate of an eye pattern.

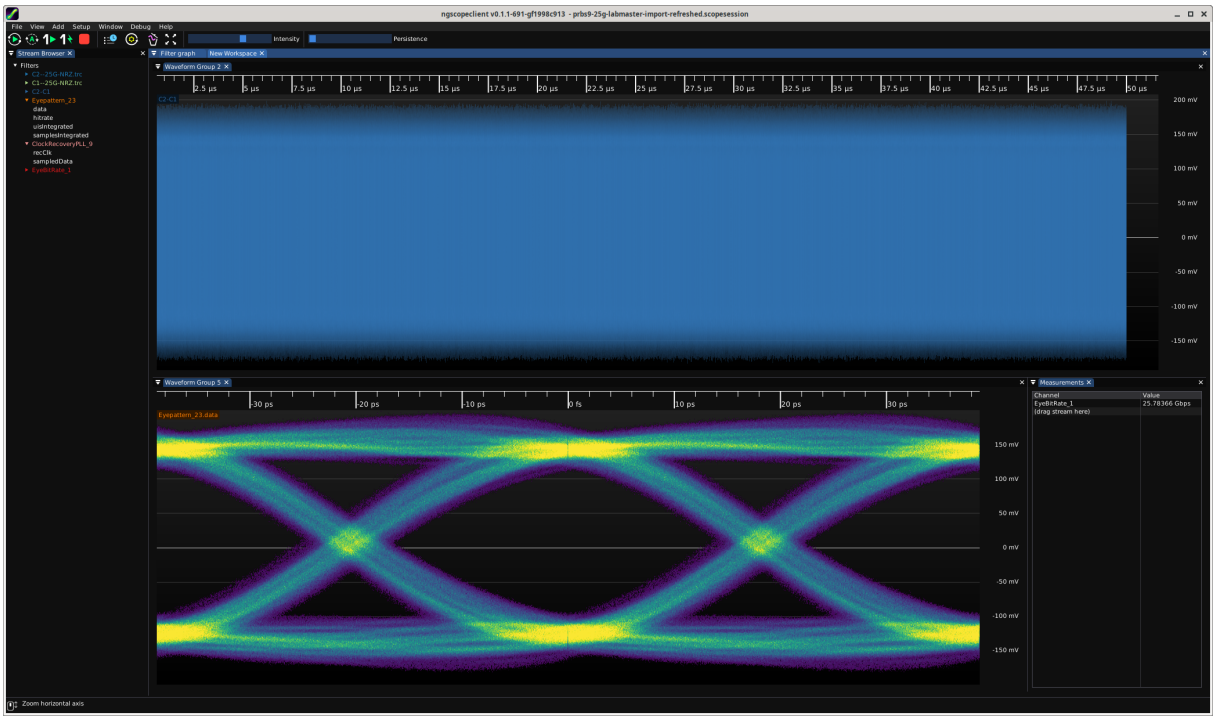


Figure 26.125: Example of measuring bit rate of an eye pattern

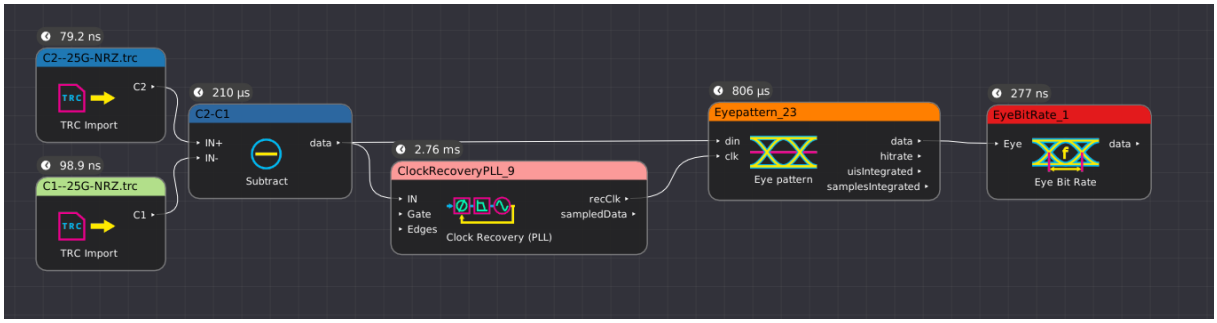



Figure 26.126: Example filter graph measuring bit rate of an eye pattern

### 26.79.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| Eye | Eye pattern | Input signal |

### 26.79.2 Parameters

This filter takes no parameters.

### 26.79.3 Output Signal

This filter outputs a scalar containing the calculated bit rate.

## 26.80 Eye Height

Measures the vertical opening of an eye pattern. The opening can be measured as either at a single point, or across a range of values representing the receiver sampling aperture.

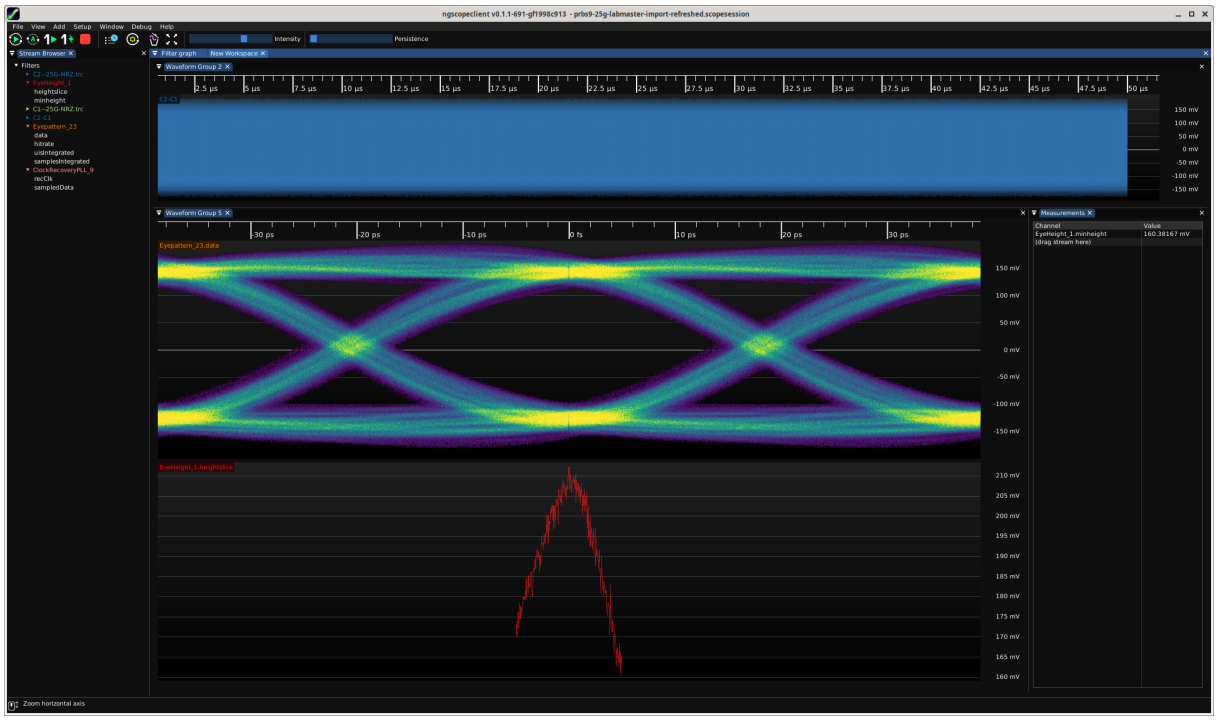


Figure 26.127: Example of measuring height of an eye pattern

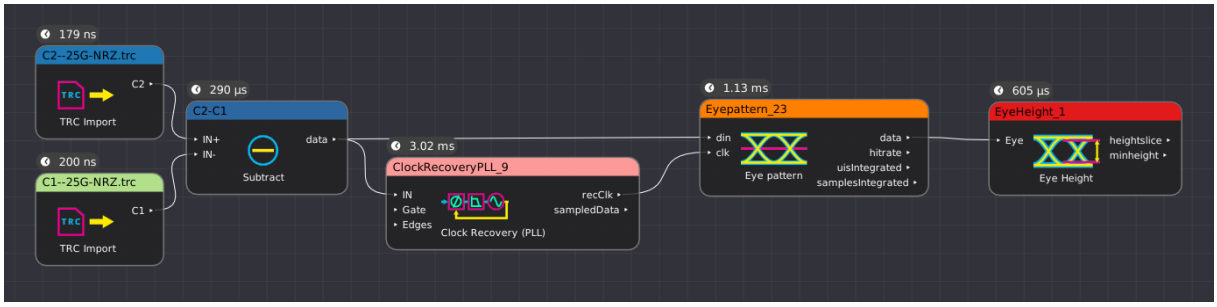



Figure 26.128: Example filter graph measuring height of an eye pattern

### 26.80.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| Eye | Eye pattern | Input signal |

### 26.80.2 Parameters

| Parameter name | Type | Description |
|---|---|---|
| Begin Time | Integer | Start position of measurement (eye center = 0) |
| End Time | Integer | End position of measurement (eye center = 0) |
| Midpoint Voltage | Integer | Vertical center of the eye |

### 26.80.3 Output Signal

| Output name | Type | Description |
|---|---|---|
| heightslice | Analog waveform | Plot of vertical opening vs horizontal position across the measurement window |
| minheight | Float | Smallest opening found within the measurement window |

## 26.81 Eye P-P Jitter

Measures the peak-to-peak jitter of an eye pattern. The jitter can be measured as either at a single point, or across a range of values representing the receiver sampling aperture.

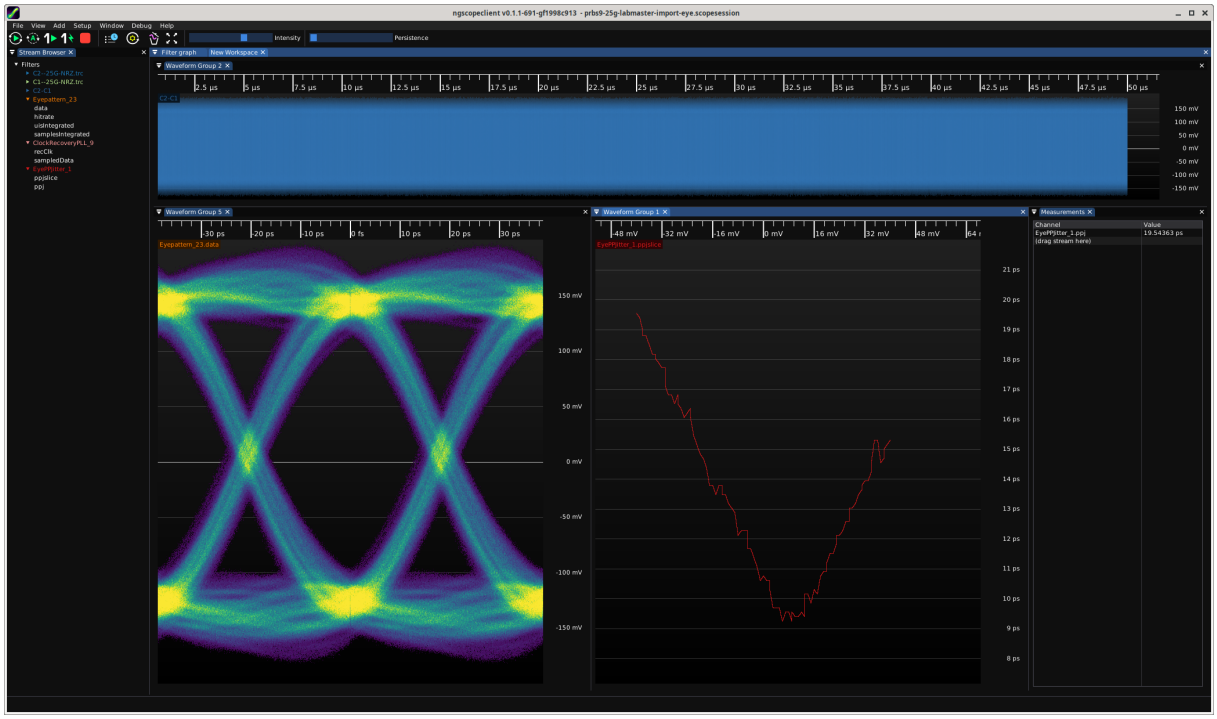


Figure 26.129: Example of measuring p-p jitter of an eye pattern

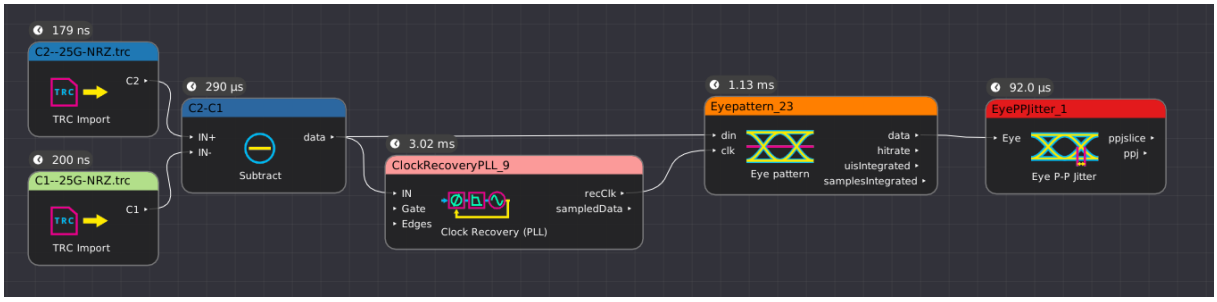



Figure 26.130: Example filter graph measuring p-p jitter of an eye pattern

### 26.81.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| Eye | Eye pattern | Input signal |

### 26.81.2 Parameters

| Parameter name | Type | Description |
|---|---|---|
| Begin Voltage | Float | Start position of measurement |
| End Voltage | Float | End position of measurement |

### 26.81.3 Output Signal

| Output name | Type | Description |
|---|---|---|
| ppjslice | Analog waveform | Plot of jitter vs vertical position across the measurement window |
| ppj | Float | Largest jitter found within the measurement window |

## 26.82 Eye Pattern

Calculates an eye pattern given an analog waveform and a clock (typically generated by the Clock Recovery (PLL) filter).

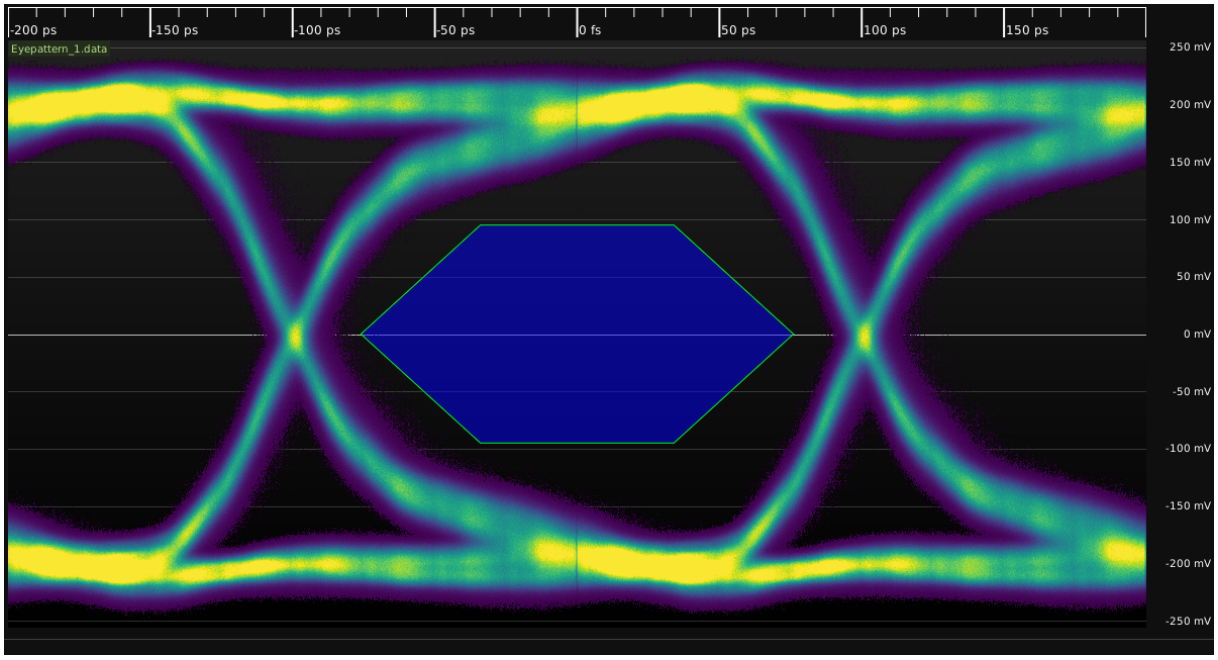



Figure 26.131: Typical usage of the eye pattern filter

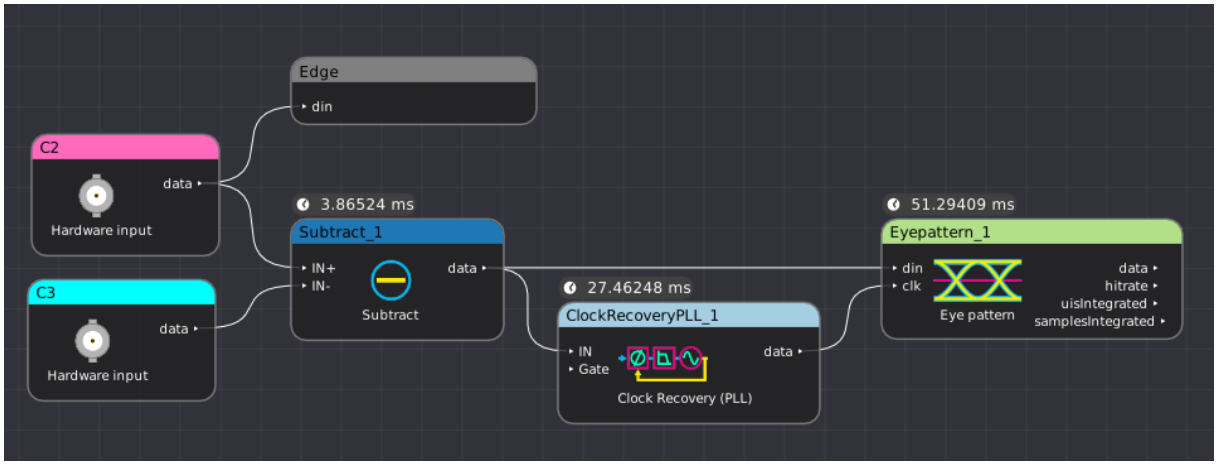



Figure 26.132: Example filter graph for computing an eye pattern from a differential input

### 26.82.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| din | Analog | Input waveform |
| clk | Digital | Symbol clock |

### 26.82.2 Parameters

| Parameter name | Type | Description |
|---|---|---|
| Bit Rate | Int | Symbol rate (if not using automatic bit rate calculation) |
| Bit Rate Mode | Enum | Specifies whether to automatically calculate the symbol rate from the input clock, or use a user-specified symbol rate |
| Center Voltage | Float | Vertical midpoint of the eye (normally 0) |
| Clock Edge | Float | Selects whether the sampling clock is rising edge triggered, falling edge triggered, or DDR (default) |
| Mask | Filename | Path to a YAML file containing pass/fail mask polygons |
| Saturation Level | Float | Saturation level for normalizing eye patterns, in range [0, 1.0]. 1.0 is the default and maps the full input range to the full output range; lower values saturate. For example 0.5 means the bottom half of count values map to the full scale output range and anything larger is clipped to full scale. |
| Vertical Range | Float | Full scale Y-axis range (if not autoscaling) |
| Vertical Scale Mode | Enum | Specifies whether to autoscale the input or use a user-specified full scale range |

### 26.82.3 Output Signal

| Stream name | Type | Description |
|---|---|---|
| data | Eye | Output eye pattern |
| hitrate | Scalar | Fraction of samples touching the mask (mask hits / samples integrated). If no mask is set, this output will always be zero. |
| uisIntegrated | Scalar | Number of unit intervals integrated by the eye |
| samplesIntegrated | Scalar | Number of samples integrated by the eye |

## 26.83 Eye Period

Measures the UI width of an eye pattern.

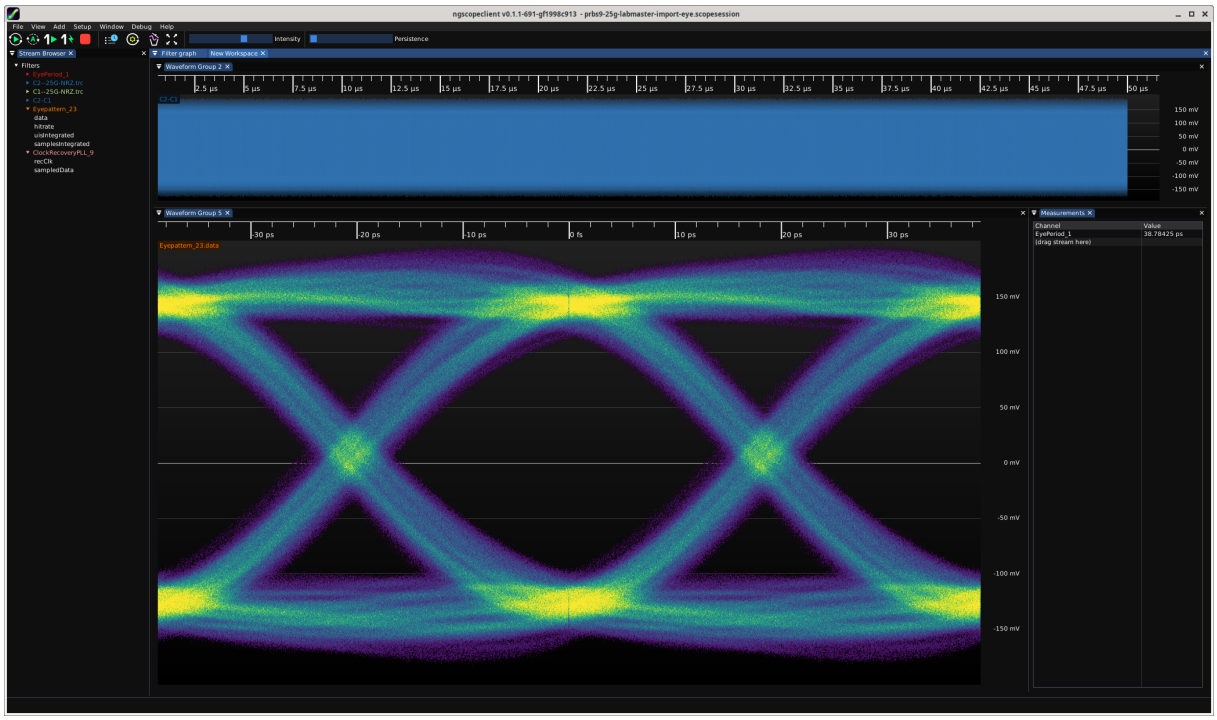


Figure 26.133: Example of measuring period of an eye pattern

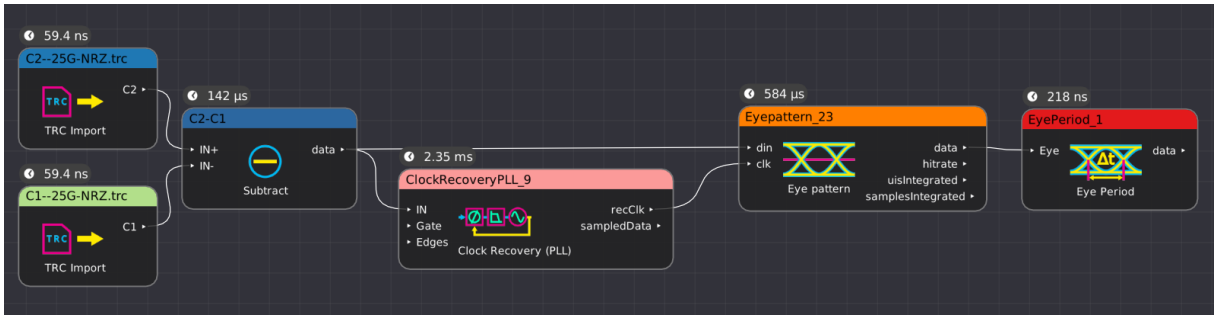



Figure 26.134: Example filter graph measuring period of an eye pattern

### 26.83.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| Eye | Eye pattern | Input signal |

### 26.83.2 Parameters

This filter takes no parameters.

### 26.83.3 Output Signal

This filter outputs a scalar containing the calculated period.

## 26.84 Eye Width

Measures the horizontal opening of an eye pattern. The opening can be measured as either at a single point, or across a range of values representing the receiver sampling aperture.

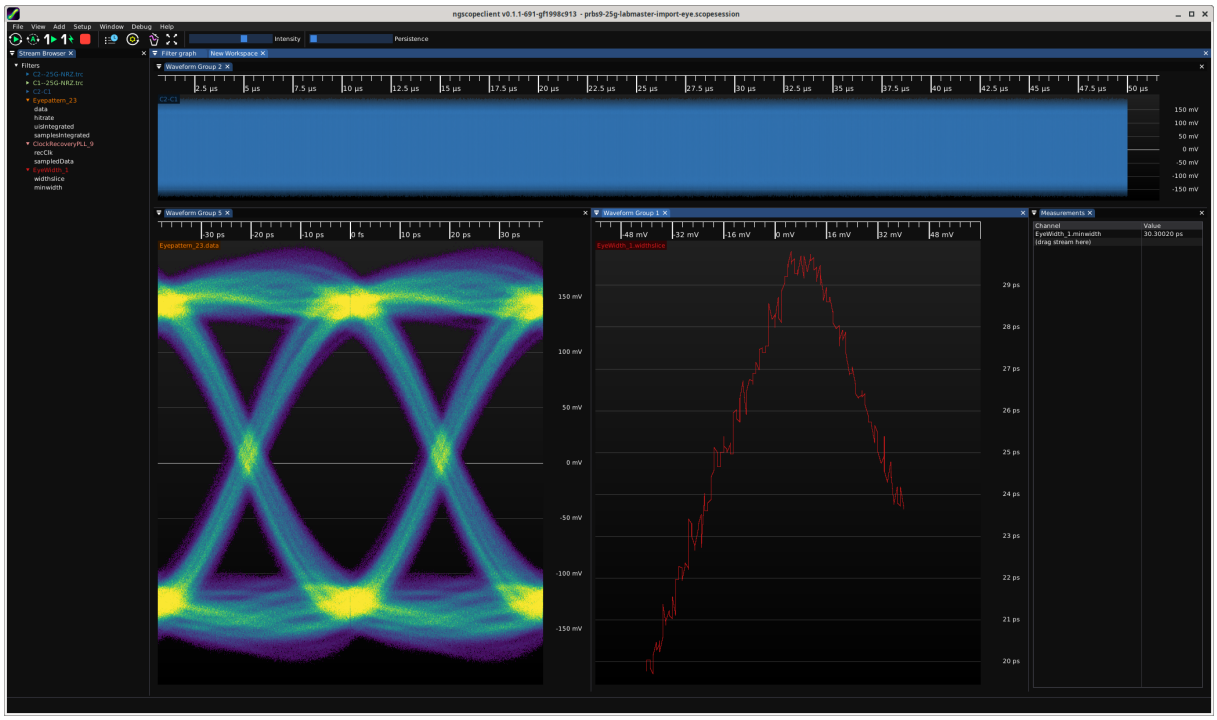


Figure 26.135: Example of measuring width of an eye pattern

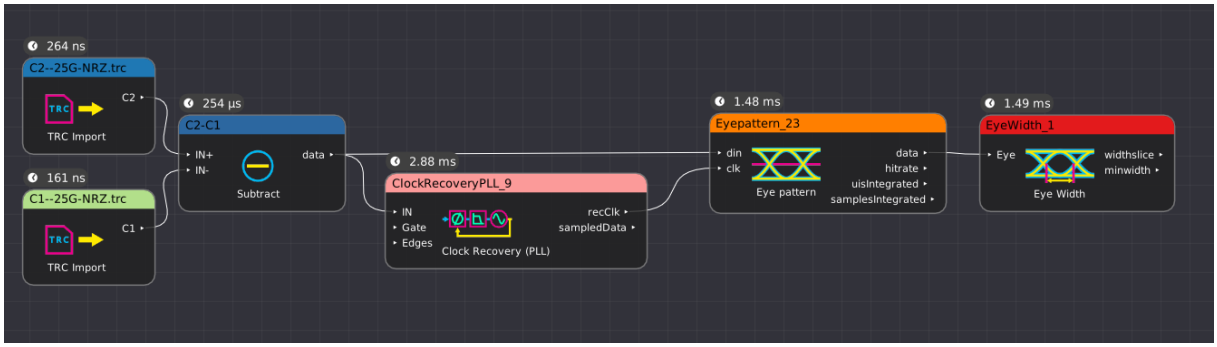



Figure 26.136: Example filter graph measuring width of an eye pattern

### 26.84.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| Eye | Eye pattern | Input signal |

### 26.84.2 Parameters

| Parameter name | Type | Description |
|---|---|---|
| Start Voltage | Float | Start position of measurement |
| End Voltage | Float | End position of measurement |

### 26.84.3 Output Signal

| Output name | Type | Description |
|---|---|---|
| heightslice | Analog waveform | Plot of horizontal opening vs vertical position across the measurement window |
| minheight | Float | Smallest opening found within the measurement window |

## 26.85 Fall

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

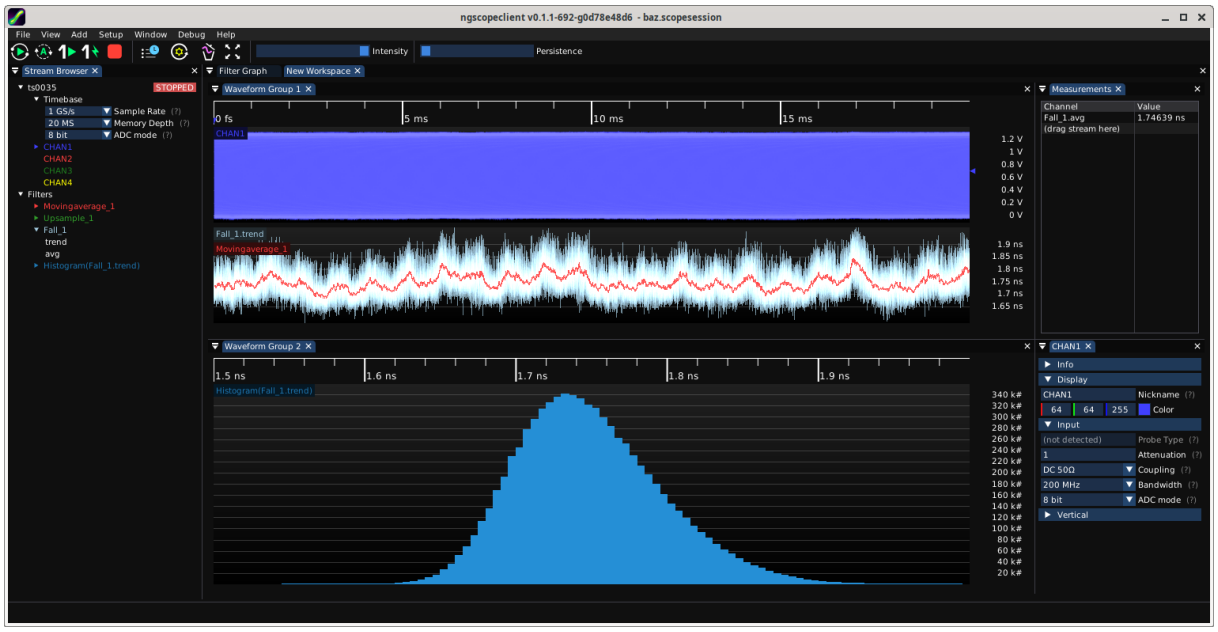



Figure 26.137: Example of measuring fall time of a waveform and performing statistics on the results

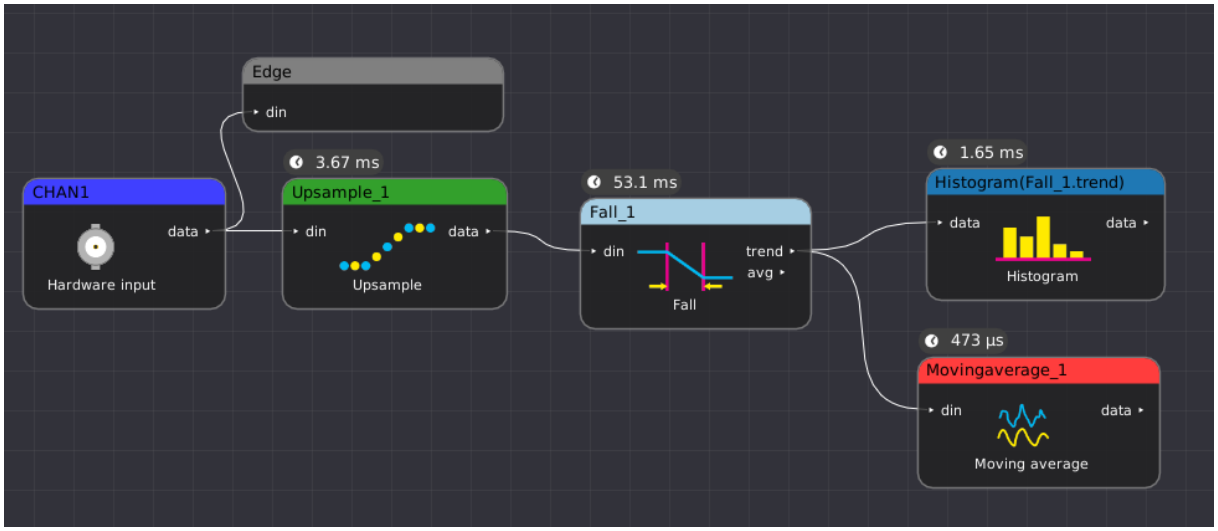



Figure 26.138: Example filter graph measuring fall time and performing statistics

### 26.85.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| din | Analog | Input signal |

### 26.85.2 Parameters

| Parameter name | Type | Description |
|---|---|---|
| Start Fraction | Float | Level to begin the fall time measurement at, in percent |
| End Fraction | Float | Level to end the fall time measurement at, in percent |

### 26.85.3 Output Signal

| Output name | Type | Description |
|---|---|---|
| trend | Analog waveform | Plot of fall time for each edge in the signal |
| avg | Analog scalar | Average fall time over the whole waveform |

## 26.86 FFT

Calculates a Fast Fourier Transform and displays the log magnitude response.

NOTE: As of this writing (v0.2) the peak detection algorithm is CPU based and very slow compared to the FFT itself, especially at deep memory depths. For example, on a Xeon 8362 + AMD Radeon 9700, a 20M point FFT runs in about 2.5 milliseconds while the peak detection takes nearly 450 ms. We intend to optimize and GPU accelerate this in the future. To avoid unexpected slowdowns when adding a FFT to a filter graph, the peak detector defaults to disabled; to enable it simply set the "number of peaks" parameter to a nonzero value.




Figure 26.139: Example showing FFT of a modulated signal and using cursors to measure total in-band power

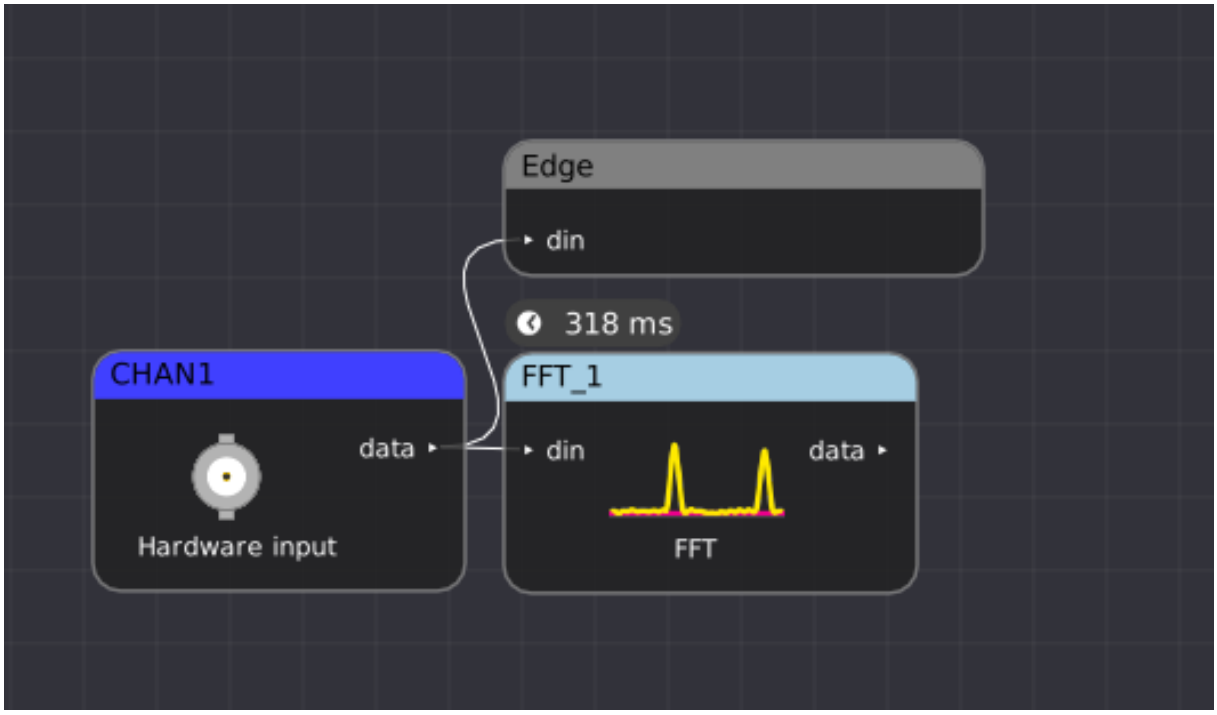



Figure 26.140: Example filter graph showing FFT of a scope channel

### 26.86.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| din | Analog | Input signal |

### 26.86.2 Parameters

| Parameter name | Type | Description |
|---|---|---|
| Number of Peaks | Int | Maximum number of peaks to report |
| Peak Window | Int | Minimum spacing between two peaks for them to be reported separately |
| Window | Enum | FFT window function |

### 26.86.3 Output Signal

## 26.87 FIR

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

(This filter description is a stub and will be expanded in the future)

## 26.88 Frequency

Measures the frequency of each cycle in a waveform.

(This filter description is a stub and will be expanded in the future)

## 26.89 Full Width at Half Maximum

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




Figure 26.141: Example of full width at half maximum of a Sinewave input waveform.

### 26.89.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| din | Analog | Input waveform |

### 26.89.2 Parameters

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

### 26.89.3 Output Signal

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

## 26.90 Gate

This filter outputs a copy of its input with zero delay if the enable signal is high. If the enable signal is low, the output is either unchanged (latched) or no waveform is produced (gated).

(This filter description is a stub and will be expanded in the future)

## 26.91 Glitch Removal

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

### 26.91.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| data | Digital | Input data. |

### 26.91.2 Parameters

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

### 26.91.3 Output Signal

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

## 26.92 Group Delay

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

### 26.92.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| Phase | Analog | Phase angle vs frequency |

### 26.92.2 Parameters

This filter takes no parameters.

### 26.92.3 Output Signal

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

## 26.93 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.

### 26.93.1 Inputs

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

### 26.93.2 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 |

### 26.93.3 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.)

## 26.94 Horizontal Bathtub

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

(This filter description is a stub and will be expanded in the future)

## 26.95 HDMI

Decodes HDMI

(This filter description is a stub and will be expanded in the future)

## 26.96 $I^2C$

Decodes the Phillips $I^2C$ bus protocol.

(This filter description is a stub and will be expanded in the future)

## 26.97 $I^2C$ EEPROM

Decodes common $I^2C$ EEPROM memory devices

(This filter description is a stub and will be expanded in the future)

## 26.98 $I^2C$ Register

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

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

(This filter description is a stub and will be expanded in the future)

## 26.99 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.

### 26.99.1 Inputs

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

### 26.99.2 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 |

### 26.99.3 Output Signal

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

## 26.100 Invert

Inverts an analog waveform by negating each sample.

(This filter description is a stub and will be expanded in the future)

## 26.101 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).

(This filter description is a stub and will be expanded in the future)

## 26.102 IPv4

Internet Protocol version 4

(This filter description is a stub and will be expanded in the future)

## 26.103 IQ Demux

Given a sampled data waveform containing consecutively sampled I and Q values, output separate sampled I and Q waveforms and a half-rate clock.

I is always sampled before Q.

Two alignment methods are supported: None (first clock edge in the input is arbitrarily declared to be I) and 100Base-T1 (the alignment with the least (0,0) symbols is preferred)

(This filter description is a stub and will be expanded in the future)

## 26.104 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.

(This filter description is a stub and will be expanded in the future)

## 26.105 J1939 Analog

Outputs an analog signal extracted from a specific J1939 PGN

(This filter description is a stub and will be expanded in the future)

## 26.106 J1939 Bitmask

Outputs a Boolean signal indicating whether a specific J1939 PGN matches a bitmask

(This filter description is a stub and will be expanded in the future)

## 26.107 J1939 PDU

Decodes CAN bus IDs to decode the J1939 PDU1 and PDU2 format.

(This filter description is a stub and will be expanded in the future)

## 26.108 J1939 Source Match

Filters a sequence of J1939 PDUs to output only those which are sent from a specified source address

(This filter description is a stub and will be expanded in the future)

## 26.109 J1939 Transport

Decodes the J1939 transport layer (PGNs 60416 and 60160) and reassembles multi-part packets into single logical packets. All other PGNs pass through this filter unchanged.

As of now, only broadcast mode (BAM) packets are decoded, due to lack of CM test datasets.

(This filter description is a stub and will be expanded in the future)

## 26.110 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.

### 26.110.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| din | Digital | Input waveform |

### 26.110.2 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 |

### 26.110.3 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.

## 26.111 Jitter Spectrum

Calculates an FFT of a TIE waveform.

(This filter description is a stub and will be expanded in the future)

## 26.112 JTAG

Joint Test Action Group

(This filter description is a stub and will be expanded in the future)

## 26.113 Magnitude

Calculates the magnitude of a complex valued signal

(This filter description is a stub and will be expanded in the future)

## 26.114 Maximum

This filter calculates the maximum of its input.

### 26.114.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| in | Analog | Input waveform |

### 26.114.2 Parameters

This filter takes no parameters.

### 26.114.3 Output Signal

| Signal name | Type | Description |
|---|---|---|
| latest | Scalar | Maximum of the filter's current input |
| cumulative | Scalar | Maximum of all input since the last clear-sweeps |
| totalSamples | Scalar | Total number of integrated samples |
| totalWaveforms | Scalar | Total number of integrated waveforms |

## 26.115 MDIO

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

(This filter description is a stub and will be expanded in the future)

## 26.116 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.

(This filter description is a stub and will be expanded in the future)

## 26.117 MIL-STD-1553

Decodes the MIL-STD-1553 avionics data bus.

(This filter description is a stub and will be expanded in the future)

## 26.118 Minimum

This filter calculates the minimum of its input.

### 26.118.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| in | Analog | Input waveform |

### 26.118.2 Parameters

This filter takes no parameters.

### 26.118.3 Output Signal

| Signal name | Type | Description |
|---|---|---|
| latest | Scalar | Minimum of the filter's current input |
| cumulative | Scalar | Minimum of all input since the last clear-sweeps |
| totalSamples | Scalar | Total number of integrated samples |
| totalWaveforms | Scalar | Total number of integrated waveforms |

## 26.119 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.

(This filter description is a stub and will be expanded in the future)

## 26.120 MIPI D-Phy Escape Mode

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

(This filter description is a stub and will be expanded in the future)

## 26.121 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.

(This filter description is a stub and will be expanded in the future)

## 26.122 MIPI DSI Frame

Converts a MIPI DSI Packet stream into video scanlines.

(This filter description is a stub and will be expanded in the future)

## 26.123 MIPI DSI Packet

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

(This filter description is a stub and will be expanded in the future)

## 26.124 Moving Average

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

(This filter description is a stub and will be expanded in the future)

## 26.125 Multiply

Multiplies one waveform or scalar by another. No resampling is performed; if both inputs are waveforms they must have identical sample rates and be aligned in time.

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

(This filter description is a stub and will be expanded in the future)

## 26.126 NCO

Numerically controlled oscillator: creates a sinusoidal waveform whose frequency tracks a target value

(This filter description is a stub and will be expanded in the future)

## 26.127 Noise

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

(This filter description is a stub and will be expanded in the future)

## 26.128 Overshoot

Measures overshoot of a signal's rising edges past the top value

(This filter description is a stub and will be expanded in the future)

## 26.129 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.

(This filter description is a stub and will be expanded in the future)

## 26.130 PAM Edge Detector

Finds level crossings in a PAM signal (of arbitrary order) and outputs a digital waveform which toggles each time the PAM signal transitions to a new level. This may be used as the input to a CDR PLL block which is designed to work on NRZ input.

(This filter description is a stub and will be expanded in the future)

## 26.131 PcapNG Export

Exports a series of packets to a PCAPNG file (or pipe). As of this writing, Ethernet is the only implemented link layer.

(This filter description is a stub and will be expanded in the future)

## 26.132 PcapNG Import

Imports a PcapNG file as a list of packets. As of this writing, CAN is the only implemented link layer.

(This filter description is a stub and will be expanded in the future)

## 26.133 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.

(This filter description is a stub and will be expanded in the future)

## 26.134 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.

(This filter description is a stub and will be expanded in the future)

## 26.135 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.

(This filter description is a stub and will be expanded in the future)

## 26.136 PCIe Link Training

Decodes the initial PCIe gen1/2 link training sequence. Training to gen3 and higher speeds is not yet implemented.

(This filter description is a stub and will be expanded in the future)

## 26.137 PCIe Transport

Decodes the Transaction 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).

(This filter description is a stub and will be expanded in the future)

## 26.138 Peak Hold

(This filter description is a stub and will be expanded in the future)

## 26.139 Peak-to-Peak

Measures the peak-to-peak amplitude of a signal

(This filter description is a stub and will be expanded in the future)

## 26.140 Peaks

Finds peaks in a waveform (typically a spectrum of some sort)

(This filter description is a stub and will be expanded in the future)

## 26.141 Period

Measures the period of a waveform

(This filter description is a stub and will be expanded in the future)

## 26.142 Phase

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

(This filter description is a stub and will be expanded in the future)

## 26.143 Phase Nonlinearity

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

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

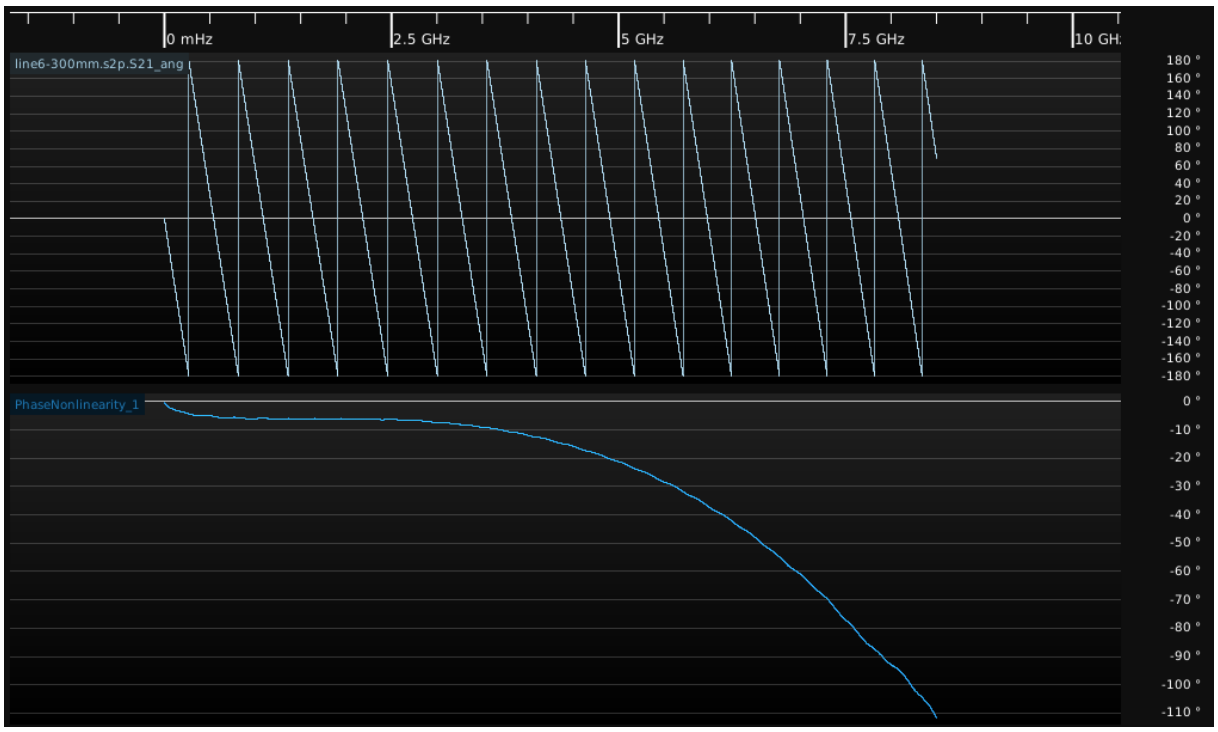



Figure 26.142: Example of nonlinear phase of a dispersive transmission line

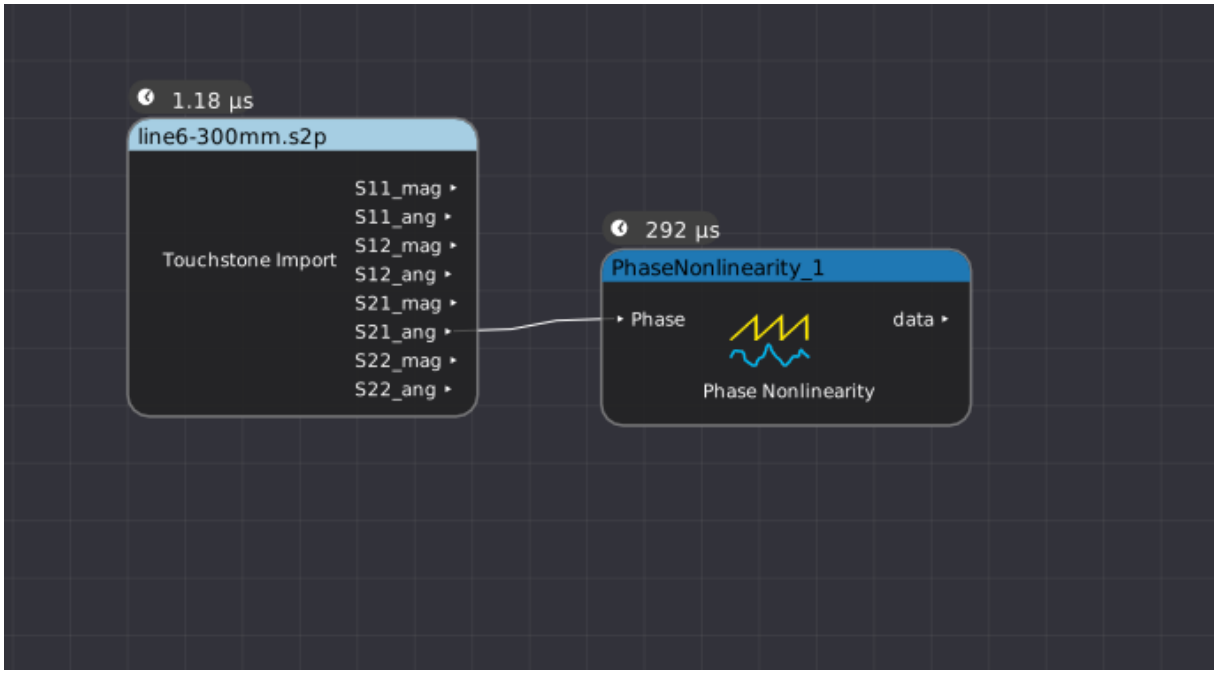



Figure 26.143: Example filter graph for measuring nonlinearity

### 26.143.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| Phase | Analog | Input waveform |

### 26.143.2 Parameters

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

### 26.143.3 Output Signal

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

## 26.144 Point Sample

This filter samples the value of a waveform at a given X coordinate and outputs the sampled value as a scalar.

(This filter description is a stub and will be expanded in the future)

## 26.145 PRBS

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

(This filter description is a stub and will be expanded in the future)

## 26.146 Pulse Width

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

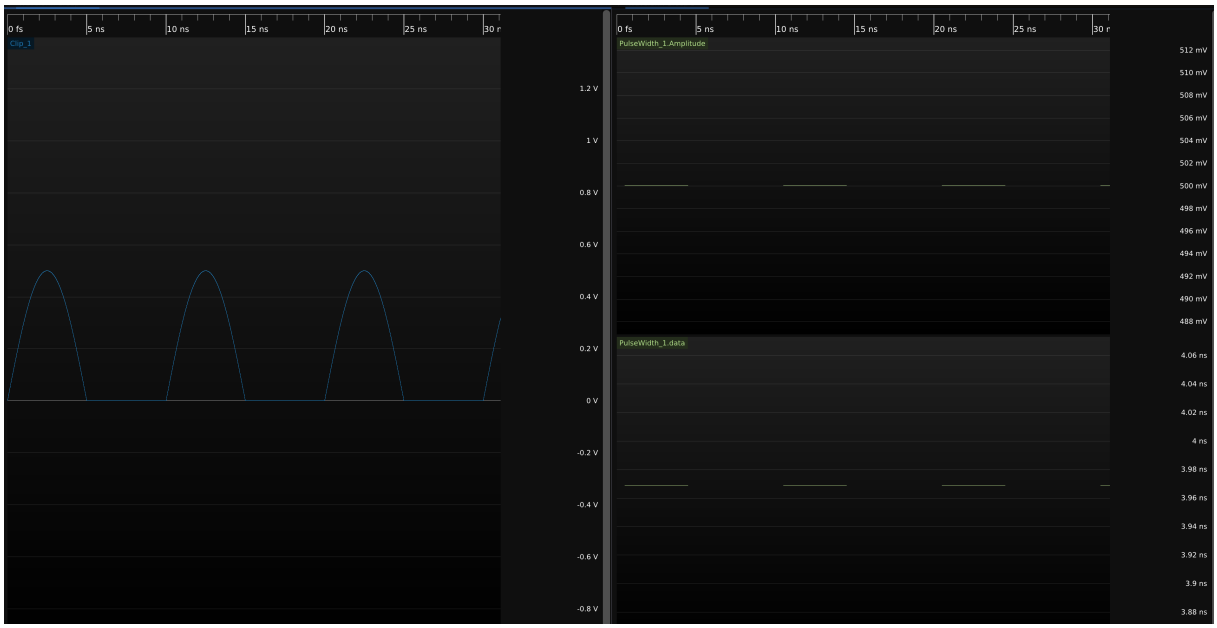



Figure 26.144: Example of pulse width measurement of a clipped sinewave input waveform.

### 26.146.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| din | Analog | Input waveform |

### 26.146.2 Output Signal

This filter outputs two output waveforms. One is a 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. Other is a similar sparse analog waveform, but its values are equal to the amplitude of the pulses. In case, the input is uniform or sparse digital, this second output waveform is uniform or sparse digital respectively instead of analog.

## 26.147 QSPI

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

(This filter description is a stub and will be expanded in the future)

## 26.148 Quadrature

Quadrature pulses from a rotary encoder

(This filter description is a stub and will be expanded in the future)

## 26.149 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.

(This filter description is a stub and will be expanded in the future)

## 26.150 RGB LED

Decodes the single-wire serial protocol used by common addressable RGB LEDs such as the WS2812

(This filter description is a stub and will be expanded in the future)

## 26.151 RIS

Performs random interleaved sampling (RIS) on a repetitive, uniformly sampled analog waveform in order to create a waveform with a higher effective sample rate.

(This filter description is a stub and will be expanded in the future)

## 26.152 Rise

Calculates the rise time for each cycle of a waveform

(This filter description is a stub and will be expanded in the future)

## 26.153 Rj + BUj

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

(This filter description is a stub and will be expanded in the future)

## 26.154 RMS

Measures the Root Mean Square value of the waveform, including any DC component

### 26.154.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| din | Analog | Input waveform |

### 26.154.2 Parameters

This filter takes no parameters.

### 26.154.3 Output Signal

This filter has two output streams.

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

## 26.155 S-Parameter Cascade

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

(This filter description is a stub and will be expanded in the future)

## 26.156 Sawtooth

Generates a sawtooth waveform.

(This filter description is a stub and will be expanded in the future)

## 26.157 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.

(This filter description is a stub and will be expanded in the future)

## 26.158 Scalar Pulse Delay

Delays a scalar pulse by approximately the specified real time.

This filter is intended for use in control or test applications to trigger a measurement after an experimental setup has had time to stabilize.

(This filter description is a stub and will be expanded in the future)

## 26.159 Scalar Stairstep

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

(This filter description is a stub and will be expanded in the future)

## 26.160 SD Card Command

(This filter description is a stub and will be expanded in the future) Decodes the Secure Digital card command bus protocol

## 26.161 Setup / Hold

Calculates the setup and hold times of a data signal relative to a clock.

Setup time is measured from data valid (leaving the undefined region between Vil and Vih) to clock invalid (entering the undefined region). Hold time is measured from clock valid to data invalid.

The overall setup and hold times reported by the filter are the minimums observed across the entire waveform.

If the undefined regions of data and clock signals ever overlap, both setup and hold time are reported as zero.

### 26.161.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| data | Analog | Input data |
| clock | Analog | Input clock |

### 26.161.2 Parameters

| Parameter name | Type | Description |
|---|---|---|
| Vih | Float | High level threshold for the input buffer |
| Vil | Float | Low level threshold for the input buffer |
| Clock Edge | Enum | **Rising**: Sample data on the rising clock edge<br>**Falling**: Sample data on the falling clock edge<br>**Both**: Sample data on both clock edges |

### 26.161.3 Output Signal

| Signal name | Type | Description |
|---|---|---|
| tsetup | Analog scalar | Lowest setup time observed in the waveform |
| thold | Analog scalar | Lowest hold time observed in the waveform |

## 26.162 Sine

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

(This filter description is a stub and will be expanded in the future)

## 26.163 SNR

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

(This filter description is a stub and will be expanded in the future)

### 26.163.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| in | Analog | Input Waveform |

### 26.163.2 Parameters

This filter takes no parameters.

### 26.163.3 Output Signal

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

## 26.164 Spectrogram

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

(This filter description is a stub and will be expanded in the future)

## 26.165 SPI

Serial Peripheral Interface.

(This filter description is a stub and will be expanded in the future)

## 26.166 SPI Flash

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

(This filter description is a stub and will be expanded in the future)

## 26.167 Squelch

Detects periods with no signal.

(This filter description is a stub and will be expanded in the future)

## 26.168 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.

(This filter description is a stub and will be expanded in the future)

## 26.169 Subtract

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

(This filter description is a stub and will be expanded in the future)

### 26.169.1 Inputs

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

### 26.169.2 Parameters

This filter takes no parameters.

### 26.169.3 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.

## 26.170 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.

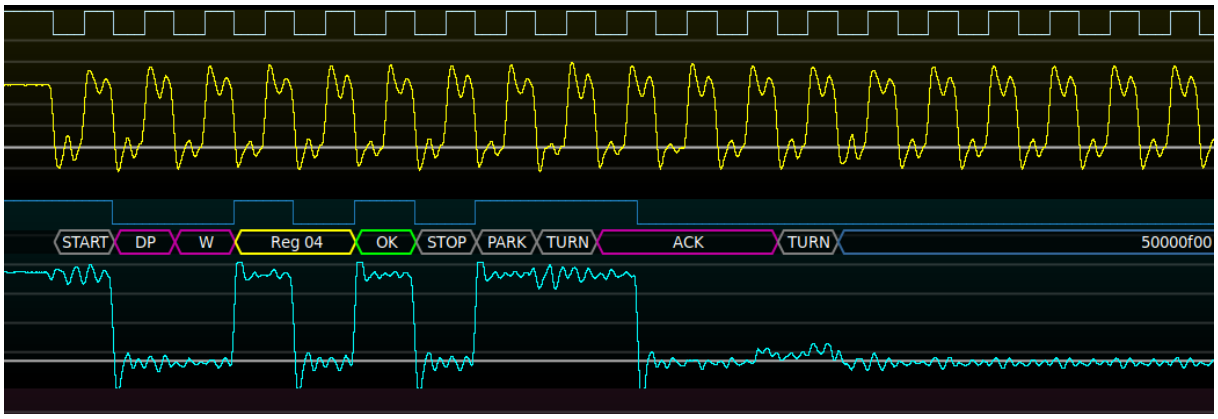



Figure 26.145: Example of SWD protocol decode

### 26.170.1 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) |

### 26.170.2 Parameters

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

### 26.170.3 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 |

## 26.171 SWD MEM-AP

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

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

(This filter description is a stub and will be expanded in the future)

## 26.172 Tachometer

Converts pulses from a tachometer to shaft speed

(This filter description is a stub and will be expanded in the future)

## 26.173 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.

(This filter description is a stub and will be expanded in the future)

## 26.174 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.

(This filter description is a stub and will be expanded in the future)

## 26.175 TDR

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

(This filter description is a stub and will be expanded in the future)

## 26.176 Time Outside Level

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

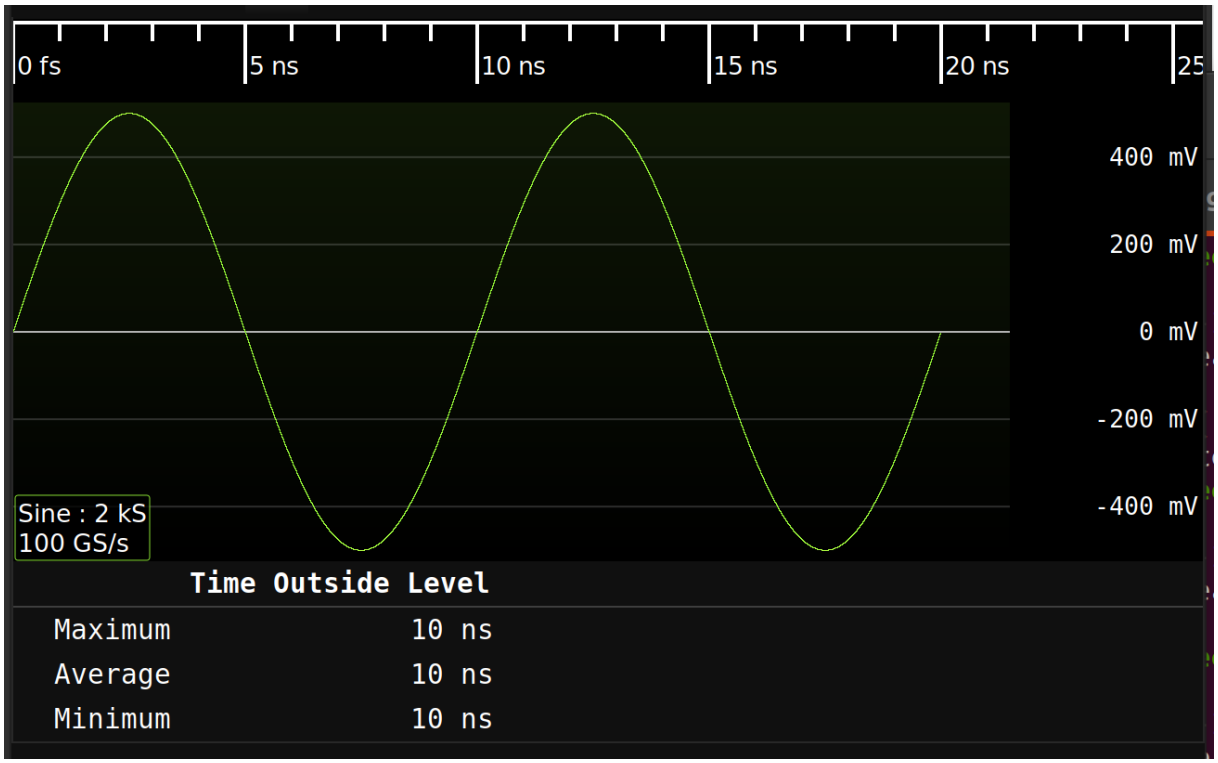



Figure 26.146: Example of time outside high level measurement with a high level threshold of 0mV

### 26.176.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| din | Analog | Input waveform |

### 26.176.2 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<br>**Low Level**: Measure the total time the signal is below low level reference voltage<br>**Both**: Measure the total time the signal is both above and below high level and low level reference voltages respectively |

## 26.177 Thermal Diode

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

(This filter description is a stub and will be expanded in the future)

## 26.178 Threshold

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

(This filter description is a stub and will be expanded in the future)

### 26.178.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| din | Analog | Input waveform |

### 26.178.2 Parameters

| Parameter name | Type | Description |
|---|---|---|
| Threshold | Float | Decision threshold |

### 26.178.3 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.

## 26.179 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.

(This filter description is a stub and will be expanded in the future)

### 26.179.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| din | Analog | Input waveform |

### 26.179.2 Parameters

This filter takes no parameters.

### 26.179.3 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.

## 26.180 Touchstone Export

Saves S-parameter data to a Touchstone file.

(This filter description is a stub and will be expanded in the future)

## 26.181 Touchstone Import

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

(This filter description is a stub and will be expanded in the future)

## 26.182 Trend

Plots a trend of a scalar value over time

(This filter description is a stub and will be expanded in the future)

## 26.183 TRC Import

Loads waveform data from a Teledyne LeCroy TRC waveform file.

(This filter description is a stub and will be expanded in the future)

## 26.184 UART

Decode serial ports

(This filter description is a stub and will be expanded in the future)

## 26.185 Unwrapped Phase

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

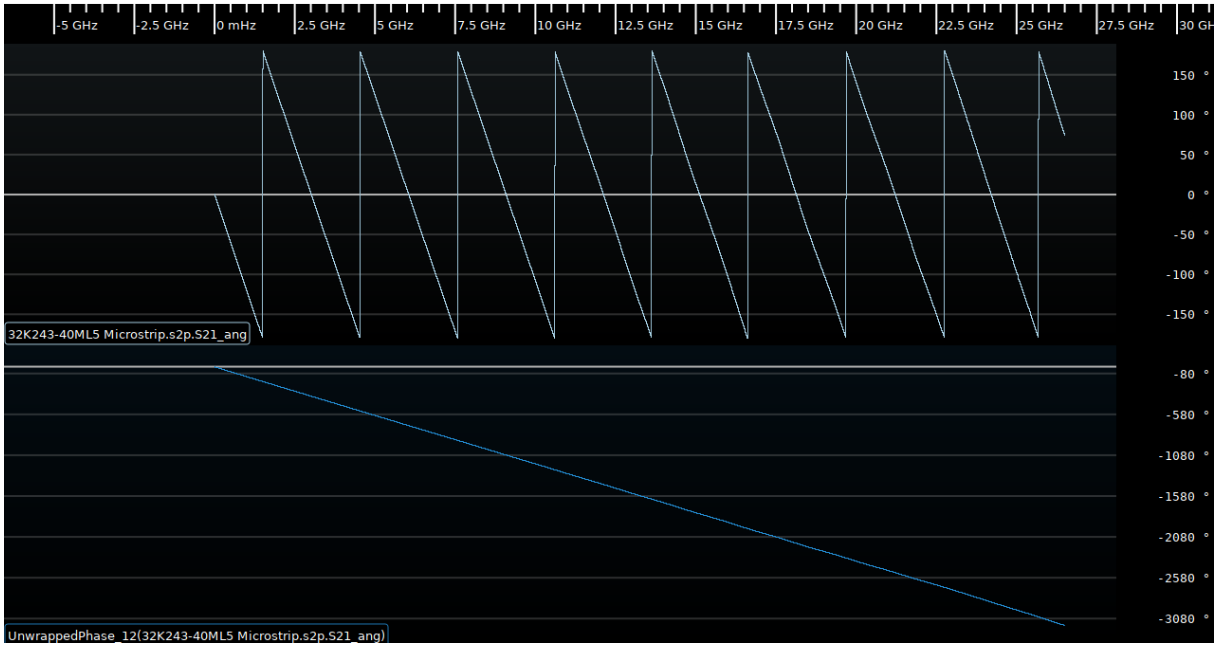



Figure 26.147: Example of wrapped and unwrapped phase of a transmission line

### 26.185.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| Phase | Analog | Input waveform |

### 26.185.2 Parameters

This filter takes no parameters.

### 26.185.3 Output Signal

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

## 26.186 USB 1.0 / 2.x Activity

(This filter description is a stub and will be expanded in the future)

## 26.187 USB 1.0 / 2.x Packet

(This filter description is a stub and will be expanded in the future)

## 26.188 USB 1.0 / 2.x PCS

(This filter description is a stub and will be expanded in the future)

## 26.189 USB 1.0 / 2.x PMA

(This filter description is a stub and will be expanded in the future)

## 26.190 Undershoot

(This filter description is a stub and will be expanded in the future)

## 26.191 Upsample

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

(This filter description is a stub and will be expanded in the future)

## 26.192 VCD Import

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

(This filter description is a stub and will be expanded in the future)

## 26.193 Vector Frequency

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

(This filter description is a stub and will be expanded in the future)

## 26.194 Vector Phase

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

(This filter description is a stub and will be expanded in the future)

## 26.195 Vertical Bathtub

(This filter description is a stub and will be expanded in the future)

## 26.196 VICP

Decodes the Teledyne LeCroy Virtual Instrument Control Protocol (VICP)

(This filter description is a stub and will be expanded in the future)

## 26.197 Waterfall

(This filter description is a stub and will be expanded in the future)

## 26.198 WAV Import

Loads waveform data from a Microsoft WAV audio file.

(This filter description is a stub and will be expanded in the future)

## 26.199 WFM Import

Loads waveform data from a Tektronix .wfm file.

(This filter description is a stub and will be expanded in the future)

## 26.200 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.

## 26.201 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.

(This filter description is a stub and will be expanded in the future)

### 26.201.1 Inputs

| Signal name | Type | Description |
|---|---|---|
| din | Analog or Digital | Input waveform |

### 26.201.2 Parameters

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

### 26.201.3 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.

## 26.202 X-Y Sweep

This filter converts a sweeping X scalar value and a corresponding Y scalar value into a waveform plotting X against Y.

Note that this filter assumes that the X value is sweeping in an upwards ramp, and is not intended for use with arbitrary X-Y data. In particular, the output is a standard sparse waveform type rather than an X-Y density map.

(This filter description is a stub and will be expanded in the future)