Advanced Design System 2011.01

Feburary 2011
Measurement Expressions

© Agilent Technologies, Inc. 2000-2011
5301 Stevens Creek Blvd., Santa Clara, CA 95052 USA
No part of this documentation may be reproduced in any form or by any means (including electronic storage and retrieval or translation into a foreign language) without prior agreement and written consent from Agilent Technologies, Inc. as governed by United States and international copyright laws.

Acknowledgments

Mentor Graphics is a trademark of Mentor Graphics Corporation in the U.S. and other countries. Mentor products and processes are registered trademarks of Mentor Graphics Corporation. * Calibre is a trademark of Mentor Graphics Corporation in the US and other countries. "Microsoft®, Windows®, MS Windows®, Windows NT®, Windows 2000® and Windows Internet Explorer® are U.S. registered trademarks of Microsoft Corporation. Pentium® is a U.S. registered trademark of Intel Corporation. PostScript® and Acrobat® are trademarks of Adobe Systems Incorporated. UNIX® is a registered trademark of the Open Group. Oracle and Java and registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners. SystemC® is a registered trademark of Open SystemC Initiative, Inc. in the United States and other countries and is used with permission. MATLAB® is a U.S. registered trademark of The Math Works, Inc.. HiSIM2 source code, and all copyrights, trade secrets or other intellectual property rights in and to the source code in its entirety, is owned by Hiroshima University and STARC. FLEXlm is a trademark of Globetrotter Software, Incorporated. Layout Boolean Engine by Klaas Holwerda, v1.7 http://www.xs4all.nl/~kholwerd/bool.html . FreeType Project, Copyright (c) 1996-1999 by David Turner, Robert Wilhelm, and Werner Lemberg. QuestAgent search engine (c) 2000-2002, JObjects. Motif is a trademark of the Open Software Foundation. Netscape is a trademark of Netscape Communications Corporation. Netscape Portable Runtime (NSPR), Copyright (c) 1998-2003 The Mozilla Organization. A copy of the Mozilla Public License is at http://www.mozilla.org/MPL/ . FFTW, The Fastest Fourier Transform in the West, Copyright (c) 1997-1999 Massachusetts Institute of Technology. All rights reserved.

The following third-party libraries are used by the NlogN Momentum solver:

"This program includes Metis 4.0, Copyright © 1998, Regents of the University of Minnesota", http://www.cs.umn.edu/~metis , METIS was written by George Karypis (karypis@cs.umn.edu).

Intel@ Math Kernel Library, http://www.intel.com/software/products/mkl

SuperLU_MT version 2.0 - Copyright © 2003, The Regents of the University of California, through Lawrence Berkeley National Laboratory (subject to receipt of any required approvals from U.S. Dept. of Energy). All rights reserved. SuperLU Disclaimer: THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "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 COPYRIGHT OWNER OR CONTRIBUTORS BE 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.

7-zip - 7-Zip Copyright: Copyright (C) 1999-2009 Igor Pavlov. Licenses for files are: 7z.dll: GNU LGPL + unRAR restriction, All other files: GNU LGPL. 7-zip License: This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version. This library is distributed in the hope that it will be useful,but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details. You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA. unRAR copyright: The decompression engine for RAR archives was developed using source code of unRAR program.All copyrights to original unRAR code are owned by Alexander Roshal. unRAR License: The unRAR sources cannot be used to re-create the RAR compression algorithm, which is proprietary. Distribution of modified unRAR sources in separate form or as a part of other software is permitted, provided that it is clearly stated in the documentation and source comments that the code may not be used to develop a RAR (WinRAR) compatible archiver. 7-zip Availability: http://www.7-zip.org/

AMD Version 2.2 - AMD Notice: The AMD code was modified. Used by permission. AMD copyright: AMD Version 2.2, Copyright © 2007 by Timothy A. Davis, Patrick R. Amestoy, and Iain S. Duff. All Rights Reserved. AMD License: Your use or distribution of AMD or any modified version of AMD implies that you agree to this License. This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version. This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details. You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA Permission is hereby granted to use or copy this program under the terms of the GNU LGPL, provided that the Copyright, this License, and the Availability of the original version is retained on all copies.User documentation of any code that uses this code or any modified version of this code must cite the Copyright, this License, the Availability note, and "Used by permission." Permission to modify the code and to distribute modified code is granted, provided the Copyright, this License, and the Availability note are retained, and a notice that the code was modified is included. AMD Availability: http://www.cise.ufl.edu/research/sparse/amd

UMFPACK 5.0.2 - UMFPACK Notice: The UMFPACK code was modified. Used by permission. UMFPACK Copyright: UMFPACK Copyright © 1995-2006 by Timothy A. Davis. All Rights Reserved. UMFPACK License: Your use or distribution of UMFPACK or any modified version of UMFPACK implies that you agree to this License. This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version. This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details. You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA Permission is hereby granted to use or copy this

program under the terms of the GNU LGPL, provided that the Copyright, this License, and the Availability of the original version is retained on all copies. User documentation of any code that uses this code or any modified version of this code must cite the Copyright, this License, the Availability note, and "Used by permission." Permission to modify the code and to distribute modified code is granted, provided the Copyright, this License, and the Availability note are retained, and a notice that the code was modified is included. UMFPACK Availability: http://www.cise.ufl.edu/research/sparse/umfpack UMFPACK (including versions 2.2.1 and earlier, in FORTRAN) is available at http://www.cise.ufl.edu/research/sparse . MA38 is available in the Harwell Subroutine Library. This version of UMFPACK includes a modified form of COLAMD Version 2.0, originally released on Jan. 31, 2000, also available at http://www.cise.ufl.edu/research/sparse . COLAMD V2.0 is also incorporated as a built-in function in MATLAB version 6.1, by The MathWorks, Inc. http://www.mathworks.com . COLAMD V1.0 appears as a column-preordering in SuperLU (SuperLU is available at http://www.netlib.org). UMFPACK v4.0 is a built-in routine in MATLAB 6.5. UMFPACK v4.3 is a built-in routine in MATLAB 7.1.

Qt Version 4.6.3 - Qt Notice: The Qt code was modified. Used by permission. Qt copyright: Qt Version 4.6.3, Copyright (c) 2010 by Nokia Corporation. All Rights Reserved. Qt License: Your use or distribution of Qt or any modified version of Qt implies that you agree to this License. This library is free software; you can redistribute it and/or modify it under the
terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version. This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details. You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA Permission is hereby granted to use or copy this program under the terms of the GNU LGPL, provided that the Copyright, this License, and the Availability of the original version is retained on all copies.User
documentation of any code that uses this code or any modified version of this code must cite the Copyright, this License, the Availability note, and "Used by permission." Permission to modify the code and to distribute modified code is granted, provided the Copyright, this License, and the Availability note are retained, and a notice that the code was modified is included. Qt Availability: http://www.qtsoftware.com/downloads Patches Applied to Qt can be found in the installation at:
$HPEESOF_DIR/prod/licenses/thirdparty/qt/patches. You may also contact Brian Buchanan at Agilent Inc. at brian_buchanan@agilent.com for more information.

The HiSIM_HV source code, and all copyrights, trade secrets or other intellectual property rights in and to the source code, is owned by Hiroshima University and/or STARC.

Errata The ADS product may contain references to "HP" or "HPEESOF" such as in file names and directory names. The business entity formerly known as "HP EEsof" is now part of Agilent Technologies and is known as "Agilent EEsof". To avoid broken functionality and to maintain backward compatibility for our customers, we did not change all the names and labels that contain "HP" or "HPEESOF" references.

Warranty The material contained in this document is provided "as is", and is subject to being changed, without notice, in future editions. Further, to the maximum extent permitted by applicable law, Agilent disclaims all warranties, either express or implied,

with regard to this documentation and any information contained herein, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. Agilent shall not be liable for errors or for incidental or consequential damages in connection with the furnishing, use, or performance of this document or of any information contained herein. Should Agilent and the user have a separate written agreement with warranty terms covering the material in this document that conflict with these terms, the warranty terms in the separate agreement shall control.

Technology Licenses The hardware and/or software described in this document are furnished under a license and may be used or copied only in accordance with the terms of such license. Portions of this product include the SystemC software licensed under Open Source terms, which are available for download at http://systemc.org/ . This software is redistributed by Agilent. The Contributors of the SystemC software provide this software "as is" and offer no warranty of any kind, express or implied, including without limitation warranties or conditions or title and non-infringement, and implied warranties or conditions merchantability and fitness for a particular purpose. Contributors shall not be liable for any damages of any kind including without limitation direct, indirect, special, incidental and consequential damages, such as lost profits. Any provisions that differ from this disclaimer are offered by Agilent only.

Restricted Rights Legend U.S. Government Restricted Rights. Software and technical data rights granted to the federal government include only those rights customarily provided to end user customers. Agilent provides this customary commercial license in Software and technical data pursuant to FAR 12.211 (Technical Data) and 12.212 (Computer Software) and, for the Department of Defense, DFARS 252.227-7015 (Technical Data - Commercial Items) and DFARS 227.7202-3 (Rights in Commercial Computer Software or Computer Software Documentation).

Introduction to Measurement Expressions 13
Measurement Expressions Syntax 15
Manipulating Simulation Data with Expressions 18
User-Defined Functions 21
Functions Reference Format 21
Using Measurement Expressions in Advanced Design System 23
MeasEqn(Measurement Equations Component) 23
Circuit Budget Functions 26
Budget Measurement Analysis 26
bud_freq() 27
bud_gain() 29
bud_gain_comp() 30
bud_gamma() 32
bud_ip3_deg() 33
bud_nf() 34
bud_nf_deg() 35
bud_noise_pwr() 37
bud_pwr() 38
bud_pwr_inc() 39
bud_pwr_refl() 40
bud_snr() 42
bud_tn() 43
bud_vswr() 44
Circuit Envelope Functions 46
Working with Envelope Data 46
ACPR_ChPwr_or_EVM_from_1tone_swp() 46
acpr_vi() 48
acpr_vr() 50
channel_power_vi() 51
channel_power_vr() 53
const_evm() 54
cross_hist() 56
delay_path() 57
evm_wlan_dsss_cck_pbcc() 58
evm_wlan_ofdm() 65
fs() 72
Mod_Data_from_1tone_swpUNI() 76
peak_pwr() 79
peak_to_avg_pwr() 80
power_ccdf() 81
power_ccdf_ref() 83
pwr_vs_t() 84
relative_noise_bw() 85
sample_delay_pi4dqpsk() 86
sample_delay_qpsk() 87
spectrum_analyzer() 88
total_pwr() 93
trajectory() 94
Data Access Functions for Measurement Expressions 96
build_subrange() 96
chop() 97
chr() 98
circle() 98

collapse() 99
contour_ex() 100
contour() 101
contour_polar() 102
copy() 103
create() 104
dd_threshold() 105
delete() 106
expand() 106
find() 107
find_index() 108
generate() 109
get_attr() 110
get_indep_values() 110
indep() 111
max_index() 112
min_index() 113
permute() 113
plot_vs() 114
set_attr() 115
size() 116
sort() 117
sweep_dim() 117
sweep_size() 118
type() 119
vs() 119
what() 120
write_var() 121
FrontPanel Eye Diagram Functions 123
Working with the Eye Diagram FrontPanel 123
eye_binning() 123
eye_density() 124
FrontPanel_eye_2d_indepvar_maximum_inner() 125
FrontPanel_eye() 126
FrontPanel_eye_amplitude_histogram() 127
FrontPanel_eye_crossings() 128
FrontPanel_eye_delay() 129
FrontPanel_eye_fall_trace() 130
FrontPanel_eye_horizontal_histogram() 131
FrontPanel_eye_regular() 132
FrontPanel_eye_risefall_marker() 133
FrontPanel_eye_rise_trace() 134
FrontPanel_eye_topbase() 135
Frontpanel_get_histogram_mean_stddev() 136
FrontPanel_pp_rms_jitter() 137
FrontPanel_wave_1st_falling_edge_period() 138
FrontPanel_wave_1st_rising_edge_period() 139
FrontPanel_wave_1st_transition_fall_time() 141
FrontPanel_wave_1st_transition_rise_time() 142
FrontPanel_wave_datarate() 143
FrontPanel_wave_negative_pulse_width() 144
FrontPanel_wave_positive_pulse_width() 145
FrontPanel_wave_topbase() 146

Jitter Analysis Functions 148
- Working with Jitter Analysis Data 148
- Jitter Analysis Process 149
- Viewing Results 151
- bathtub() 156
- jitter_separation() 160

FrontPanel S-Parameter TDR Functions 165
- FrontPanel_TDRExtrapolate() 165
- FrontPanel_TDREye() 166
- FrontPanel_TDRFreqMode() 167
- FrontPanel_TDRFreqScale() 167
- FrontPanel_TDRFreqSweep() 168
- FrontPanel_TDRGate() 168
- FrontPanel_TDRIFW() 169
- FrontPanel_TDRInversePeeling() 169
- FrontPanel_TDRPeeling() 169
- FrontPanel_TDRPortExt() 170
- FrontPanel_TDRPortMap() 171
- FrontPanel_TDRSmooth() 171
- FrontPanel_TDRTimeScale() 172
- FrontPanel_TDRTimeSweep() 172
- FrontPanel_TDRWindow() 173
- tdr_inverse_peeling() 173
- tdr_peeling() 174

Harmonic Balance Functions For Measurement Expressions 175
- Working with Harmonic Balance Data 176
- carr_to_im() 176
- cdrange() 177
- dc_to_rf() 178
- ifc() 178
- ip3_in() 179
- ip3_out() 180
- ipn() 181
- it() 182
- mix() 183
- pae() 184
- pfc() 185
- phase_gain() 186
- pspec() 187
- pt() 187
- remove_noise() 188
- sfdr() 189
- snr() 190
- spur_track() 191
- spur_track_with_if() 192
- thd_func() 193
- ts() 193
- vfc() 196
- vspec() 197
- vt() 198

Math Functions For Measurement Expressions 199
- abs() 200
- acos() 201

acosh() 202
acot() 203
acoth() 203
asin() 204
asinh() 204
atan2() 205
atan() 206
atanh() 206
ceil() 207
cint() 208
cmplx() 208
complex() 209
conj() 210
convBin() 210
convHex() 211
convInt() 212
convOct() 212
cos() 213
cosh() 214
cot() 214
coth() 215
cum_prod() 216
cum_sum() 216
db() 217
dbm() 217
dbmtow() 219
deg() 220
diagonal() 220
diff() 221
erf() 222
erfc() 222
erfcinv() 223
erfinv() 223
exp() 224
fft() 225
fix() 226
float() 226
floor() 227
fmod() 227
hypot() 228
identity() 228
im() 229
imag() 230
int() 230
integrate() 231
interp() 232
interpolate() 233
inverse() 234
jn() 234
ln() 235
log10() 235
log() 236
mag() 237

max2() 237
max() 238
max_outer() 239
min2() 239
min() 240
min_outer() 241
num() 242
ones() 242
phase() 243
phasedeg() 243
phaserad() 244
polar() 245
pow() 245
prod() 246
rad() 246
re() 247
real() 248
rms() 248
round() 249
sgn() 250
sin() 250
sinc() 251
sinh() 252
sqr() 252
sqrt() 253
step() 253
sum() 254
tan() 255
tanh() 255
transpose() 256
wtodbm() 257
xor() 257
zeros() 258
Signal Processing Functions 259
add_rf() 259
ber_pi4dqpsk() 259
ber_qpsk() 261
eye() 262
eye_amplitude() 262
eye_closure() 263
eye_fall_time() 264
eye_height() 265
eye_rise_time() 266
spec_power() 266
S-Parameter Analysis Functions for Measurement Expressions 269
abcdtoh() 270
abcdtos() 271
abcdtoy() 271
abcdtoz() 272
bandwidth_func() 272
center_freq() 273
dev_lin_gain() 274
dev_lin_phase() 275

ga_circle() 275
gain_comp() 277
gl_circle() 277
gp_circle() 279
gs_circle() 280
htoabcd() 281
htos() 282
htoy() 283
htoz() 283
ispec() 284
l_stab_circle() 284
l_stab_circle_center_radius() 285
l_stab_region() 286
map1_circle() 286
map2_circle() 287
max_gain() 288
mu() 289
mu_prime() 289
ns_circle() 290
ns_pwr_int() 292
ns_pwr_ref_bw() 292
phase_comp() 293
pwr_gain() 294
ripple() 295
sm_gamma1() 295
sm_gamma2() 296
sm_y1() 297
sm_y2() 297
sm_z1() 298
sm_z2() 298
s_stab_circle() 299
s_stab_circle_center_radius() 300
s_stab_region() 301
stab_fact() 301
stab_meas() 302
stoabcd() 303
stoh() 304
stos() 304
stot() 305
stoy() 306
stoz() 306
tdr_step_impedance() 307
tdr_sp_gamma() 307
tdr_sp_imped() 308
tdr_step_imped() 309
ttos() 310
unilateral_figure() 311
unwrap() 312
v_dc() 312
volt_gain() 313
volt_gain_max() 314
vswr() 315
write_snp() 316

yin() 317
yopt() 318
ytoabcd() 318
ytoh() 319
ytos() 319
ytoz() 320
zin() 320
zopt() 321
ztoabcd() 322
ztoh() 322
ztos() 323
ztoy() 323
Statistical Analysis Functions 325
cdf() 325
cross_corr() 326
fun_2d_outer() 326
histogram() 327
histogram_multiDim() 328
histogram_sens() 329
histogram_stat() 330
interpolate_swept_data() 331
lognorm_dist_1D() 332
lognorm_dist_inv1D() 333
mean() 333
mean_outer() 334
median() 335
moving_average() 335
norm_dist_1D() 336
norm_dist_inv1D() 337
norms_dist1D() 338
norms_dist_inv1D() 339
pdf() 339
stddev() 340
stddev_outer() 341
uniform_dist1D() 342
uniform_dist_inv1D() 343
yield_sens() 343
Transient Analysis Functions 345
Working with Transient Data 345
constellation() 345
cross() 346
fspot() 347
ifc_tran() 349
ispec_tran() 350
pfc_tran() 351
pspec_tran() 352
pt_tran() 353
vfc_tran() 354
vspec_tran() 355
vt_tran() 356
Utility Functions for Measurement Expressions 358
amodelb_snp() 358
design_name() 358

Duplicated Expression Names 359

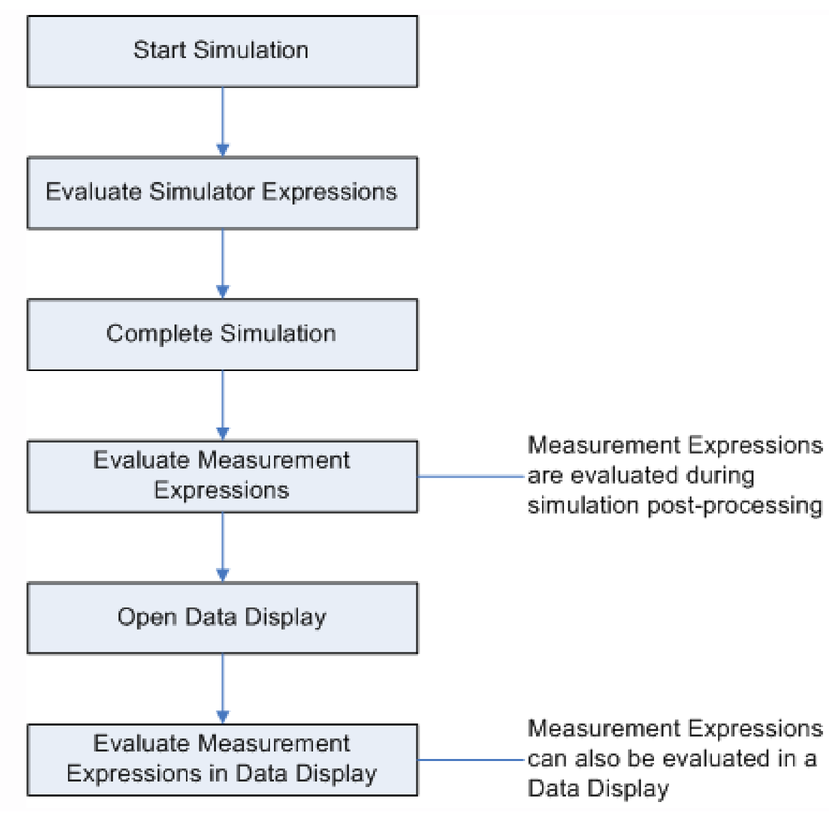
Introduction to Measurement Expressions

This document describes the measurement expressions that are available for use with several Agilent EEsof EDA products. For a complete list of available measurement functions, refer to the *Measurement Expression Functions (by category)* (expmeas).

Measurement expressions are equations that are evaluated during simulation post processing. They can be entered into the program using various methods, depending on which product you are using. Unlike the expressions described in *Simulator Expressions* (expsim), these expressions are evaluated after a simulation has completed, not before the simulation is run. Measurement expressions can also be easily used in a Data Display. For more information on entering equations in a data display, refer to *Data Display* (data).

Although there is some overlap among many of the more commonly used functions, measurement expressions and simulator expressions are derived from separate sources, evaluated at different times, and can have subtle differences in their usages. Thus, these two types of expressions need to be considered separately. For an overview of how measurement expressions are evaluated, refer to How Measurement Expressions are Evaluated.

How Measurement Expressions are Evaluated



Within this document you will find information on:

- Measurement Expressions Syntax

- Manipulating Simulation Data with Expressions

- Information on working with different types of data.
- Information specific to entering simulator expressions in your particular product.

You will also find a complete list of functions that can be used as measurement expressions individually, or combined together as a nested expression. These functions have been separated into libraries and are listed in alphabetical order within each library. The functions available include:

- *Circuit Budget Functions* (expmeas)
- *Circuit Envelope Functions* (expmeas)
- *Data Access Functions* (expmeas)
- *Harmonic Balance Functions* (expmeas)
- *Math Functions* (ael)
- *Signal Processing Functions* (expmeas)
- *S-Parameter Analysis Functions* (expmeas)
- *Statistical Analysis Functions* (expmeas)
- *Transient Analysis Functions* (expmeas)

For a complete list of all functions provided in this document, refer to the *Table of Contents* (expmeas).

Measurement Expressions Syntax

Use the following guidelines when creating measurement expressions:

- Measurement expressions are based on the mathematical syntax in Application Extension Language (AEL).
- Function names, variable names, and constant names are all case sensitive in measurement expressions.
- Use commas to separate arguments.
- White space between arguments is acceptable.

Case Sensitivity

All variable names, functions names, and equation names are case sensitive in measurement expressions.

Variable Names

Variables produced by the simulator can be referenced in equations with various degrees of rigidity. In general a variable is defined as:
DatasetName.AnalysisName.AnalysisType.CircuitPath.VariableName
By default, in the Data Display window, a variable is commonly referenced as:
DatasetName..VariableName where the double dot ".." indicates that the variable is unique in this dataset. If a variable is referenced without a dataset name, then it is assumed to be in the current default dataset.

When the results of several analyses are in a dataset, it becomes necessary to specify the analysis name with the variable name. The double dot can always be used to pad a variable name instead of specifying the complete name.

In most cases a dataset contains results from a single analysis only, and so the variable name alone is sufficient. The most common use of the double dot is when it is desired to tie a variable to a dataset other than the default dataset.

Built-in Constants

Constant	Description	Value
PI (also pi)		3.1415926535898
e	Euler's constant	2.718281822
ln10	natural log of 10	2.302585093
boltzmann	Boltzmann's constant	1.380658e-23 J/K
qelectron	electron charge	1.60217733e-19 C
planck	Planck's constant	6.6260755e-34 J*s
c0	Speed of light in free space	2.99792e+08 m/s
e0	Permittivity of free space	8.85419e-12 F/m
u0	Permeability of free space	12.5664e-07 H/m
i, j	sqrt(-1)	1i

Operator Precedence

Measurement expressions are evaluated from left to right, unless there are parentheses. Operators are listed from higher to lower precedence. Operators on the same line have the same precedence. For example, a+b*c means a+(b*c), because * has a higher precedence than +. Similarly, a+b-c means (a+b)-c, because + and - have the same precedence (and because + is left-associative).
The operators !, &&, and || work with the logical values. The operands are tested for the values TRUE and FALSE, and the result of the operation is either TRUE or FALSE. In AEL a logical test of a value is TRUE for non-zero numbers or strings with non-zero length, and FALSE for 0.0 (real), 0 (integer), NULL or empty strings. Note that the right hand operand of && is only evaluated if the left hand operand tests TRUE, and the right hand operand of

|| is only evaluated if the left hand operand tests FALSE.

Operator Precedence

<table>
<tr><th>Operator</th><th>Name</th><th>Example</th></tr>
<tr><td>( )</td><td>function call, matrix indexer</td><td>foo(expr_list)</td></tr>
<tr><td>[ ]</td><td>sweep indexer, sweep generator</td><td>X[expr_list]</td></tr>
<tr><td>{ }</td><td>matrix generator</td><td>{expr_list}</td></tr>
<tr><td>**</td><td>exponentiation</td><td>expr**expr</td></tr>
<tr><td>!</td><td>not</td><td>!expr</td></tr>
<tr><td>*</td><td>multiply</td><td>expr * expr</td></tr>
<tr><td>+</td><td>add</td><td>expr + expr</td></tr>
<tr><td>::</td><td>sequence operator</td><td>exp::expr::expr</td></tr>
<tr><td><</td><td>less than</td><td>expr < expr</td></tr>
<tr><td>==,EQUALS</td><td>equal</td><td>expr == expr</td></tr>
<tr><td>&& AND</td><td>logical and</td><td>expr && expr</td></tr>
<tr><td>|| OR</td><td>logical or</td><td>expr || expr</td></tr>
</table>

Conditional Expressions

The if-then-else construct provides an easy way to apply a condition on a per-element basis over a complete multidimensional variable. It has the following syntax:

A = if (condition) then true_expression else false_expression

Condition, true_expression, and false_expression are any valid expressions. The dimensionality and number of points in these expressions follow the same matching conditions required for the basic operators.
Multiple nested if-then-else constructs can also be used:

A = if (condition) then true_expression elseif (condition2) then true_expression else false_expression

The type of the result depends on the type of the true and false expressions. The size of the result depends on the size of the condition, the true expression, and the false expression.

Examples

The following information shows several examples of conditional expressions using various operators.
boolV1=1
boolV2=1
eqOp=if (boolV1 == 1) then 1 else 0 eqOp returns 1
eqOp1=if (boolV1 EQUALS 1) then 1 else 0 eqOp1 returns 1
notEqOp=if (boolV1 != 1) then 1 else 0 notEqOp returns 1
notEqOp1=if (boolV1 NOTEQUALS 1) then 1 else 0 notEqOp1 returns 1

andOp=if (boolV1 == 1 AND boolV2 == 1) then 1 else 0 andOp returns 1
andOp1=if (boolV1 == 1 && boolV2 == 1) then 1 else 0 andOp returns 1
orOp=if (boolV1 == 1 OR boolV2 == 1) then 1 else 0 orOp returns 1
orOp1=if (boolV1 == 1 || boolV2 == 1) then 1 else 0 orOp returns 1

Manipulating Simulation Data with Expressions

Expressions defined in this documentation are designed to manipulate data produced by the simulator. Expressions may reference any simulation output, and may be placed in a Data Display window. For details on using and applying simulation data with measurement expressions, refer to *Applying Measurements* in *Preparing a Circuit for Simulation in ADS* (cktsim) of the *Using Circuit Simulators* (cktsim).

Simulation Data

The expressions package has inherent support for two main simulation data features. First, simulation data are normally multidimensional. Each sweep introduces a dimension. All operators and relevant functions are designed to apply themselves automatically over a multidimensional simulation output. Second, the independent (swept) variable is associated with the data (for example, S-parameter data). This independent is propagated through expressions, so that the results of equations are automatically plotted or listed against the relevant swept variable.

Measurements and Expressions

Measurements are evaluated after a simulation is run and the results are stored in the dataset. The tag *meqn_xxx* (where *xxx* is a number) is placed at the beginning of all measurement results, to distinguish those results from data produced directly by the simulator.
Complex measurement equations are available for both circuit and signal processing simulations. Underlying a measurement is the same generic equations handler that is available in the Data Display window. Consequently, simulation results can be referenced directly, and the expression syntax is identical. All operators and almost all functions are available.

The expression used in an optimization goal or a yield specification is a measurement expression. It may reference any other measurement on the schematic.

Generating Data

The simulator produces scalars and matrices. When a sweep is being performed, the sweep can produce scalars and matrices as a function of a set of swept variables. It is also possible to generate data by using expressions. Two operators can be used to do this. The first is the sweep generator "[ ]", and the second is the matrix generator "{ }". These

operators can be combined in various ways to produce swept scalars and matrices. The data can then be used in the normal way in other expressions. The operators can also be used to concatenate existing data, which can be very useful when combined with the indexing operators.

Sweep Generator Examples

Several sweep generator examples are given below:

arr1=[0,1,2,3,4,5] creates an array of six values
arr2=[0::1::5] generates the above data using the sequence operator
arrCat=[arr1,arr2] concatenates the two arrays
sunArr1=[arr1[3::5],arr1[0::2]] re-arranges the existing data in a different order
z=0*[1::50]
vpadded=[arr1,z] creates a zero-padded array

Matrix Generator Examples

Some examples of the matrix builds operator are given below:

v1={1,2,3,4,5} five-element vector
v2={1::5} five-element vector using the sequence operator
v3=1,0}, {0,1 2X2 identity matrix

Simple Sweeps and Using "[ ]"

Parameter sweeps are commonly used in simulations to generate, for example, a frequency response or a set of DC IV characteristics. The simulator always attaches the swept variable to the actual data (the data often being called the *attached independent* in equations).
Often after performing a swept analysis we want to look at a single sweep point or a group of points. The sweep indexer "[ ]" can be used to do this. The sweep indexer is zero offset, meaning that the first sweep point is accessed as index 0. A sweep of n points can be accessed by means of an index that runs from 0 to n-1. Also, the what() function can be useful in indexing sweeps. Use what() to find out how many sweep points there are, and then use an appropriate index. Indexing out of range yields an invalid result.

The sequence operator can also be used to index into a subsection of a sweep. Given a parameter X, a subsection of X may be indexed as

    a=X[start::increment::stop]

Because increment defaults to one,

    a=X[start::stop]

is equivalent to

a=X[start::1::stop]

The "::" operator alone is the wildcard operator, so that X and X[::] are equivalent. Indexing can similarly be applied to multidimensional data. As will be shown later, an index may be applied in each dimension.

S-Parameters and Matrices

As described above, the sweep indexer "[ ]" is used to index into a sweep. However, the simulator can produce a swept matrix, as when an S-parameter analysis is performed over some frequency range. Matrix entries can be referenced as S11 through Snm. While this is sufficient for most simple applications, it is also possible to index matrices by using the matrix indexer "()". For example, S(1,1) is equivalent to S11. The matrix indexer is offset by one meaning the first matrix entry is X(1,1). When it is used with swept data its operation is transparent with respect to the sweep. Both indexers can be combined. For example, it is possible to access S(1,1) at the first sweep point as S(1,1)[0]. As with the sweep indexer "[ ]", the matrix indexer can be used with wild cards and sequences to extract a submatrix from an original matrix.

Matrices

S-parameters above are an example of a matrix produced by the simulator. Matrices are more frequently found in signal processing applications. Mathematical operators implement matrix operations. Element-by-element operations can be performed by using the dot modified operators (.* and ./).

The matrix indexer conveniently operates over the complete sweep, just as the sweep indexer operates on all matrices in a sweep. As with scalars, the mathematical operators allow swept and non-swept quantities to be combined. For example, the first matrix in a sweep may be subtracted from all matrices in that sweep as

Y = X-X[0]

and Indexing

Multidimensional Sweeps and Indexing

In the previous examples we looked at single-dimensional sweeps. Multidimensional sweeps can be generated by the simulator by using multiple parameter sweeps. Expressions are designed to operate on the multidimensional data. Functions and operators behave in a meaningful way when a parameter sweep is added or taken away. A common example is DC IV characteristics.

The sweep indexer accepts a list of indices. Up to N indices are used to index N-dimensional data. If fewer than N lookup indices are used with the sweep indexer, then wild cards are inserted automatically to the left. This is best explained by referring to the above example files.

User-Defined Functions

By writing some Application Extension Language (AEL) code, you can define your own custom functions. The following file is provided specifically for this purpose:

$HPEESOF_DIR/expressions/ael/user_defined_fun.ael

By reviewing the other *_fun.ael* files in this directory, you can see how to write your own code. You can have as many functions as you like in this one file, and they will all be compiled upon program start-up. If you have a large number of functions to define, you may want to organize them into more than one file. In this case, include a line such as:

```
load("more_user_defined_fun.ael");
```

These load statements are added to the *user_defined_fun.ael* in the same directory in order to have your functions all compile. To create your own custom user defined functions:

1. Copy the $HPEESOF_DIR/expressions/ael/user_defined_fun.ael file to one of the following directories.
   $HOME/hpeesof/expressions/ael (User Config)
   $HPEESOF_DIR/custom/expressions/ael (Site Config)
   Create the appropriate subdirectories if they do not already exist. The User Config is setup for a single user. The Site Config can be set up by a CAD Manager or librarian to control a site configuration for a group of users.
2. Edit the new file and add any custom defined functions. If your custom functions reside in another file, you can add a *load* statement to your new *user_defined_fun.ael* file to include your functions in another file. For example:

   ```
   load("my_custom_functions_file.ael");
   ```
3. Save your changes to the new file and restart so your changes take effect. The search path looks in the following locations for user defined functions.
   $HOME/hpeesof/expressions/ael (User Config)
   $HPEESOF_DIR/custom/expressions/ael (Site Config)
   $HPEESOF_DIR/expressions/ael (Default Config)

   Note
   If for some reason your functions are not recognized by the simulator, check to ensure that the *user_defined_fun.atf* (compiled version of user_defined_fun.ael file) was generated after restarting the software.

Functions Reference Format

The information below illustrates how each measurement expression in the functions reference is described.
<function name>
Presents a brief description of what the function does.
Syntax
Presents the general syntax of the function.

Arguments
Presents a table that includes each argument name, description, range, type, default value, and whether or not the argument is optional.

Examples
Presents one or more simple examples that use the function.

Defined in
Indicates whether the measurement function is defined in a script or is built in. All AEL functions are built in.

See also
Presents links to related functions, if there are any.

Notes/Equations
Describes any additional notes and/or equations that may help with understanding the function.

Using Measurement Expressions in Advanced Design System

Measurement Expressions are equations that are used during simulation post processing. These expressions are entered into the program using the *MeasEqn* (Measurement Equation) component, available on the Simulation palettes in an Analog/RF Schematic window (such as Simulation-AC or Simulation-Envelope), or from the Controllers palette in a Signal Processing Schematic window.

Many of the more commonly used measurement items are built in, and are found in the palettes of the appropriate simulator components. Common expressions are included as measurements, which makes it easy for beginning users to utilize the system. To make simulation and analyses convenient, all the measurement items, including the built-in items, can be edited to meet specific requirements. Underlying each measurement is a function; the functions themselves are available for modification. Moreover, it is also possible for you to write entirely new measurements and functions.

The measurement items and their underlying expressions are based on Advanced Design System's Application Extension Language (AEL). Consequently, they can serve a dual purpose:

- They can be used on the schematic page, in conjunction with simulations, to process the results of a simulation (this is useful, for example, in defining and reaching optimization goals). Unlike Simulator Expressions, the MeasEqn items are processed after the simulation engine has finishing its task and just before the dataset is written.
- They can be used in the Data Display window to process the results of a dataset that can be displayed graphically. Here the MeasEqn items are used to post-process the data written after simulation is complete.
  In either of the above cases, the same syntax is used. However, some measurements can be used on the schematic page and not the Data Display window, and vice versa. These distinctions will be noted where they occur.

Note
Not all Measurement Expression Functions have an explicit measurement component. These functions can be used by means of the MeasEqn component.

MeasEqn(Measurement Equations Component)

For a complete list of Measurement Functions, refer to the *Measurement Expressions* (expmeas) in the Advanced Design System.

Symbol

Parameters

Instance Name

Displays name of the MeasEqn component in ADS. You can edit the instance name and place more than one MeasEqn component on the schematic.

Select Parameter

Add	Add an equation to the Select Parameter field.
Cut	Delete an equation from the Select Parameter field.
Paste	Copy an equation that has been cut and place it in the Select Parameter field.

Meas

Enter your equation in this field.

Display parameter on schematic

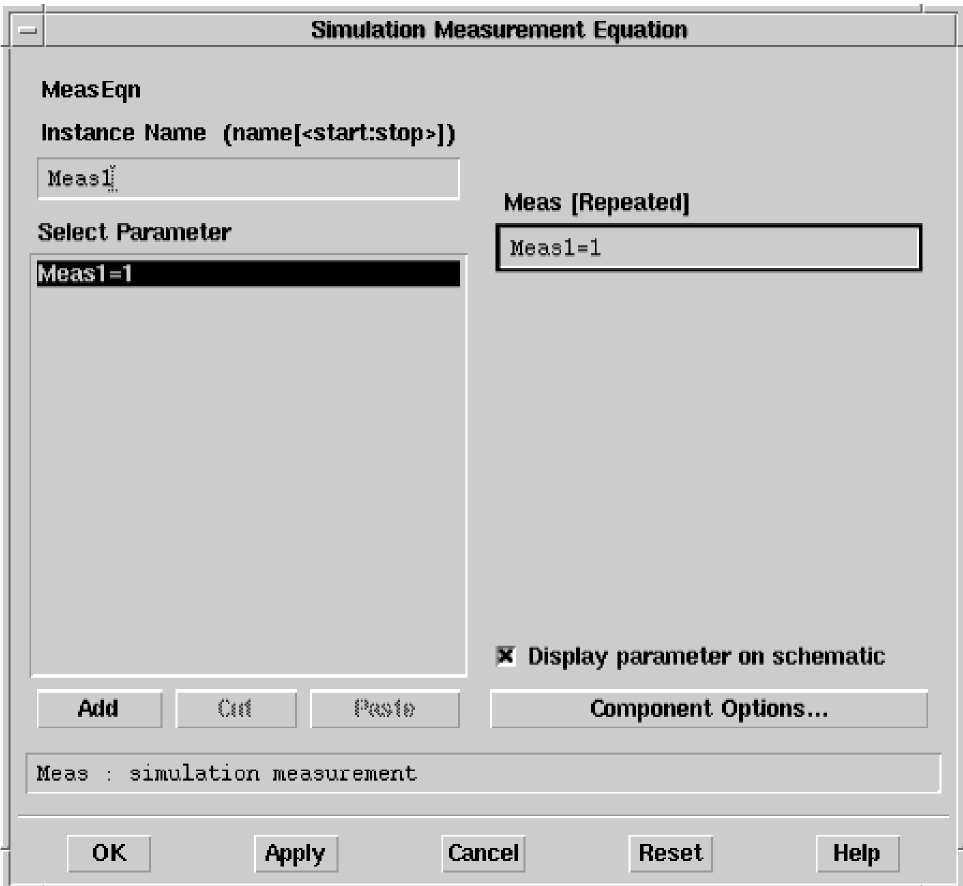
Displays or hides a selected equation on the ADS schematic.

Component Options

For information on this dialog box, refer to " *Editing Component Parameters* " in the ADS " *Schematic Capture and Layout (usrguide)* " documentation.

Notes/Equations

If you are using Advanced Design System, you can place a MeasEqn (Measurement Equation) component in a schematic window. By placing a MeasEqn component on an ADS schematic, you can write an equation that can be evaluated, following a simulation, and displayed in a Data Display window.



Note
The if-then-else construct can be used in a MeasEqn component on a schematic. It has the following syntax: A = if (condition) then true_expression else false_expression

Circuit Budget Functions

This section describes the circuit budget functions in detail. The functions are listed in alphabetical order.

- *bud freq()* (expmeas)
- *bud gain()* (expmeas)
- *bud gain comp()* (expmeas)
- *bud gamma()* (expmeas)
- *bud ip3 deg()* (expmeas)
- *bud nf()* (expmeas)
- *bud nf deg()* (expmeas)
- *bud noise pwr()* (expmeas)
- *bud pwr()* (expmeas)
- *bud pwr inc()* (expmeas)
- *bud pwr refl()* (expmeas)
- *bud snr()* (expmeas)
- *bud tn()* (expmeas)
- *bud vswr()* (expmeas)

Budget Measurement Analysis

Budget analysis determines the signal and noise performance for elements in the top-level design. Therefore, it is a key element of system analysis. Budget measurements show performance at the input and output pins of the top-level system elements. This enables the designer to adjust, for example, the gains at various components, to reduce non-linearities. These measurements can also indicate the degree to which a given component can degrade overall system performance.

Budget measurements are performed upon data generated during a special mode of circuit simulation. AC and HB simulations are used in budget mode depending upon if linear or nonlinear analysis is needed for a system design. The controllers for these simulations have a flag called, *OutputBudgetIV* which must be set to " *yes* " for the generation of budget data. Alternatively, the flag can be set by editing the AC or HB simulation component and selecting the *Perform Budget simulation* button on the Parameters tab.

Budget data contains signal voltages and currents, and noise voltages at every node in the top level design. Budget measurements are functions that operate upon this data to characterize system performance parameters including gain, power, and noise figure. These functions use a constant reference impedance for all nodes for calculations. By default this impedance is 50 Ohms. The available source power at the input network port is assumed to equal the incident power at that port.

Budget measurements are available in the schematic and the data display windows. The budget functions can be evaluated by placing the budget components from Simulation-AC or Simulation-HB palettes on the schematic. The results of the budget measurements at the terminal(s) are sorted in ascending order of the component names. The component names are attached to the budget data as additional dependent variables. To use one of these measurements in the data display window, first reference the appropriate data in

the default dataset, and then use the equation component to write the budget function. For more detailed information about Budget Measurement Analysis, see *Budget Analysis* (cktsim) in *Using Circuit Simulators* (cktsim).

Note
The budget function can refer only to the default dataset, that is, the dataset selected in the data display window.

Frequency Plan

A frequency plan of the network is determined for budget mode AC and HB simulations. This plan tracks the reference carrier frequency at each node in a network. When performing HB budget, there may be more than one frequency plan in a given network. This is the case when double side band mixers are used. Using this plan information, budget measurements are performed upon selected reference frequencies, which can differ at each node. When mixers are used in an AC simulation, be sure to set the *Enable AC frequency conversion* option on the controller, to generate the correct plan.

The budget measurements can be performed on arbitrary networks with multiple signal paths between the input and output ports. As a result, the measurements can be affected by reflection and noise generated by components placed between the terminal of interest and the output port on the same signal path or by components on different signal paths.

Reflection and Backward-Traveling Wave Effects

The effects of reflections and backward-traveling signal and noise waves generated by components along the signal path can be avoided by inserting a forward-traveling wave sampler between the components. A forward-traveling wave sampler is an ideal, frequency-independent directional coupler that allows sampling of forward-traveling voltage and current waves
This sampler can be constructed using the equation-based linear three-port S-parameter component. To do this, set the elements of the scattering matrix as follows: *S12 = S21 = S31 = 1* , and all other *Sij = 0* . The temperature parameter is set to -273.16 deg C to make the component noiseless. A noiseless shunt resistor is attached to port 3 to sample the forward-traveling waves.

bud_freq()

Returns the frequency plan of a network

Syntax

y = bud_freq(freqIn, pinNumber, "simName") for AC analysis or y = bud_freq(planNumber, pinNumber) for HB analysis

Arguments

Name	Description	Default	Range	Type	Required
freqIn	input source frequency	None	(0:∞)	Real	No
planNumber	represents the chosen frequency plan and is required when using the bud_freq() function with HB data.	None	[1:∞)	Integer	Yes
pinNumber	used to choose which pins of each network element are referenced †	None	[1:∞)	Integer	No
simName	simulation instance name, such as "AC1" or "HB1", used to qualify the data when multiple simulations are performed.	None	None	String	No

† If 1 is passed as the pinNumber, the frequency plan displayed references pin 1 of each element; otherwise, the frequency plan is displayed for all pins of each element. (Note that this means it is not possible to select only pin 2 of each element, for example.) By default, the frequency plan is displayed for pin 1 of each element.

Examples

```
x = bud_freq()
Returns frequency plan for AC analysis.
x = bud_freq(1MHz)
Returns frequency plan for frequency swept AC analysis. By passing the value of
1MHz the plan is returned for the subset of the sweep, when the source value is
1MHz
x = bud_freq(2)
For HB, returns a selected frequency plan, 2, with respect to pin 1 of every
network element.
```

Defined in

$HPEESOF_DIR/expressions/ael/budget_fun.ael

Notes/Equations

Used in AC and harmonic balance (HB) simulations.
This function is used in AC and HB simulations with the budget parameter turned on. For AC, the options are to pass no parameters, or the input source frequency (freqIn), for the first parameter if a frequency sweep is performed. freqIn can still be passed if no sweep is performed, table data is just formatted differently. The first argument must be a real number for AC data and the second argument is an integer, used optionally to choose pin references.
When a frequency sweep is performed in conjunction with AC, the frequency plan of a particular sweep point can be chosen.

For HB, this function determines the fundamental frequencies at the terminal(s) of each component, thereby given the entire frequency plan for a network. Sometimes more than one frequency plan exists in a network. For example when double sideband mixers are used. This function gives the user the option of choosing the frequency plan of interest. Note that a negative frequency at a terminal means that a spectral inversion has occurred at the terminal. For example, in frequency-converting AC analysis, where vIn and vOut are the voltages at the input and output ports, respectively, the relation may be either vOut=alpha*vIn if no spectral inversion has occurred, or vOut=alpha*conj(vIn) if there was an inversion. Inversions may or may not occur depending on which mixer sidebands

one is looking at.

Note
Remember that the budget function can refer only to the default dataset, that is, the dataset selected in the data display window.

Budget Path Measurements

Instead of all components in alphabetical order, this function can report its values just for the components selected in a budget path, and following the sequence in that path. To facilitate the budget path measurements the name of the budget path variable, as defined in the Schematic window, needs to be entered as the pinNumber argument.

bud_gain()

Returns budget transducer-power gain

Syntax

y = bud_gain(vIn, iIn, Zs, Plan, pinNumber, "simName") or y = bud_gain("SourceName", SrcIndx, Zs, Plan, budgetPath)

Arguments

Name	Description	Default	Range	Type	Required
vIn	voltage flowing into the input port	None	(-∞:∞)	Complex	Yes
iIn	current flowing into the input port	None	(-∞:∞)	Complex	Yes
SourceName	component name at the input port	None	None	String	Yes
SrcIndx †	frequency index that corresponds to the source frequency to determine which frequency to use from a multitone source as the reference signal	1	[1:∞)	Integer	No
Zs	input source port impedance	50.0	[0:∞)	Real	No
Plan †	number of the selected frequency plan(needed only for HB)	None	None	String	No
pinNumber	Used to choose which pins of each network element are referenced ††	1	[1:∞)	Integer	No
budgetPath	Selects the budget path, specified as an array eg. ["PORT1.t1","Tee1.t3","Term3.t1"]	None	None	String Array	No
simName	simulation instance name, such as "AC1" or "HB1", used to qualify the data when multiple simulations are performed.	None	None	String	No

† Note that for AC simulation, both the SrcIndx and Plan arguments must not be specified; these are for HB only.
†† If 1 is passed as the pinNumber, the results at pin 1 of each element are returned; otherwise, the results for all pins of each element are returned. By default, the pinNumber is set to 1.

Examples

```
x = bud_gain(PORT1.t1.v, PORT1.t1.i)
or
```

```
x = bud_gain("PORT1")
y= bud_gain(PORT1.t1.v, PORT1.t1.i, 75)
or
y= bud_gain("PORT1", , 75., 1)
z = bud_gain(PORT1.t1.v[3], PORT1.t1.i[3], , 1)
or
z= bud_gain("PORT1", 3, , 1)
```

Defined in

$HPEESOF_DIR/expressions/ael/budget_fun.ael

See Also

bud_gain_comp() (expmeas)

Notes/Equations

Used in AC and harmonic balance simulations
This is the power gain (in dB) from the input port to the terminal(s) of each component, looking into that component. Power gain is defined as power delivered to the resistive load minus the power available from the source. Note that the fundamental frequency at different pins can be different. If vIn and iIn are passed directly, one may want to use the index of the frequency sweep explicitly to reference the input source frequency.

Note
Remember that the budget function can refer only to the default dataset, that is, the dataset selected in the data display window.

Budget Path Measurements

Instead of all components in alphabetical order, this function can report its values just for the components selected in a budget path, and following the sequence in that path. To facilitate the budget path measurements, the name of the budget path variable (as defined in the Schematic window), must be entered as the *budgetPath* argument. Use the alternate syntax for budget path measurements as shown below.

```
bud_gain("SourceName", SrcIndx, Zs, Plan, budgetPath)
```

bud_gain_comp()

Returns budget gain compression at fundamental frequencies as a function of power.

Syntax

y = bud_gain_comp(vIn, iIn, Zs, Plan, freqIndex, pinNumber, "simName") or y = bud_gain_comp("SourceName", SrcIndx, Zs, Plan, freqIndex, pinNumber, "simName")

Arguments

Name	Description	Default	Range	Type	Required
vIn	voltage flowing into the input port	None	(-∞:∞)	Complex	Yes
iIn	current flowing into the input port	None	(-∞:∞)	Complex	Yes
SourceName	component name at the input port	None	None	String	Yes
SrcIndx †	frequency index that corresponds to the source frequency to determine which frequency to use from a multitone source as the reference signal	1	[1:∞)	Integer	No
Zs	input source port impedance	50.0	[0:∞)	Real	No
freqIndex †	index of harmonic frequency	None	(-∞:∞)	Integer	No
Plan ††	number of the selected frequency plan(needed only for HB)	None	None	String	No
pinNumber	Used to choose which pins of each network element are referenced † ††	1	[1:∞)	Integer	No
simName	simulation instance name, such as "AC1" or "HB1", used to qualify the data when multiple simulations are performed.	None	None	String	No

† Used if Plan is not selected.
†† Note that for AC simulation, both the SrcIndx and Plan arguments must not be specified; these are for HB only.
† †† If 1 is passed as the pinNumber, the results at pin 1 of each element are returned; otherwise, the results for all pins of each element are returned. By default, the pinNumber is set to 1.

Examples

```
x = bud_gain_comp(PORT1.t1.v[3], PORT1.t1.i[3], , 1)
x = bud_gain_comp("PORT1", 3, , 1)
returns the gain compression at the fundamental frequencies as a function of
power.
y= bud_gain_comp(PORT1.t1.v[3], PORT1.t1.i[3], , , 1)
y= bud_gain_comp("PORT1", 3, , , 1)
returns the gain compression at the second harmonic frequency as a function of
power.
```

Defined in

$HPEESOF_DIR/expressions/ael/budget_fun.ael

See Also

bud_gain() (expmeas)

Notes/Equations

Used in Harmonic balance simulation with sweep
This is the gain compression (in dB) at the given input frequency from the input port to

the terminal(s) of each component, looking into that component. Gain compression is defined as the small signal linear gain minus the large signal gain. Note that the fundamental frequency at each element pin can be different by referencing the frequency plan. A power sweep of the input source must be used in conjunction with HB. The first power sweep point is assumed to be in the linear region of operation.

Note
Remember that the budget function can refer only to the default dataset, that is, the dataset selected in the data display window.

Budget Path Measurements

This function does not support the budget path feature.

bud_gamma()

Returns the budget reflection coefficient.

Syntax

y = bud_gamma(Zref, Plan, pinNumber, "simName")

Arguments

Name	Description	Default	Range	Type	Required
Zref	input source port impedance	50.0	[0:∞)	Real	No
Plan †	number of the selected frequency plan(needed only for HB)	None	None	String	No
pinNumber	Used to choose which pins of each network element are referenced ††	1	[1:∞)	Integer	No
simName	simulation instance name, such as "AC1" or "HB1", used to qualify the data when multiple simulations are performed.	None	None	String	No

† Note that for AC simulation, both the SrcIndx and Plan arguments must not be specified; these are for HB only.
†† If 1 is passed as the pinNumber, the results at pin 1 of each element are returned; otherwise, the results for all pins of each element are returned. By default, the pinNumber is set to 1.

Examples

```
x = bud_gamma()
returns reflection coefficient at all frequencies.
y = bud_gamma(75, 1)
returns reflection coefficient at reference frequencies in plan 1
```

Defined in

$HPEESOF_DIR/expressions/ael/budget_fun.ael

See Also

bud_vswr() (expmeas)

Notes/Equations

Used in AC and harmonic balance simulations
This is the complex reflection coefficient looking into the terminal(s) of each component. Note that the fundamental frequency at different pins can in general be different, and therefore values are given for all frequencies unless a Plan is referenced.

Note
Remember that the budget function can refer only to the default dataset, that is, the dataset selected in the data display window.

Budget Path Measurements

Instead of all components in alphabetical order, this function can report its values just for the components selected in a budget path, and following the sequence in that path. To facilitate the budget path measurements the name of the budget path variable, as defined in the Schematic window, needs to be entered as the pinNumber argument.

bud_ip3_deg()

Returns the budget third-order intercept point degradation

Syntax

y = bud_ip3_deg(vOut, LinearizedElement, fundFreq, imFreq, zRef)

Arguments

Name	Description	Default	Range	Type	Required
vOut	signal voltage at the output	None	(-∞:∞)	Real, Complex	Yes
LinearizedElement	variable containing the names of the linearized components	None	None	String	Yes
fundFreq	harmonic frequency indices for the fundamental frequency	None	(-∞:∞)	Integer	Yes
imFreq	harmonic frequency indices for the intermodulation frequency	None	(-∞:∞)	Integer	Yes
Zref	input source port impedance	50.0	[0:∞)	Real	No

Examples

```
y = bud_ip3_deg(vOut, LinearizedElement, {1, 0}, {2, -1})
returns the budget third-order intercept point degradation
```

Defined in

$HPEESOF_DIR/expressions/ael/budget_fun.ael

See Also

ip3_out() (expmeas), *ipn()* (expmeas)

Notes/Equations

Used in Harmonic balance simulation with the BudLinearization Controller.
This measurement returns the budget third-order intercept point degradation from the input port to any given output port. It does this by setting to linear each component in the top-level design, one at a time.

For the components that are linear to begin with, this measurement will not yield any useful information. For the nonlinear components, however, this measurement will indicate how the nonlinearity of a certain component degrades the overall system IP3. To perform this measurement, the BudLinearization Controller needs to be placed in the schematic window. If no component is specified in this controller, all components on the top level of the design are linearized one at a time, and the budget IP3 degradation is computed.

Budget Path Measurements

This function does not support the budget path feature.

bud_nf()

Returns the budget noise figure

Syntax

y = bud_nf(vIn, iIn, noisevIn) or y=bud_nf("SourceName",Zs,BW,pinNumber,"simName")

Arguments

Name	Description	Default	Range	Type	Required
vIn	voltage flowing into the input port	None	(-∞:∞)	Complex	Yes
iIn	current flowing into the input port	None	(-∞:∞)	Complex	Yes
noisevIn	noise input at the input port	None	(-∞:∞)	Complex	Yes
SourceName	component name at the input port	None	None	String	Yes
Zs	input source port impedance	50.0	[0:∞)	Real	No
BW †	bandwidth	1	[1:∞)	Real	No
pinNumber	Used to choose which pins of each network element are referenced ††	1	[1:∞)	Integer	No
simName	simulation instance name, such as "AC1" or "HB1", used to qualify the data when multiple simulations are performed.	None	None	String	No

† BW must be set as the value of Bandwidth used on the noise page of the AC controller
†† If 1 is passed as the pinNumber, the results at pin 1 of each element are returned; otherwise, the results for all pins of each element are returned. By default, the pinNumber

is set to 1.

Examples

```
x = bud_nf(PORT1.t1.v, PORT1.t1.i, PORT1.t1.v.noise)
x = bud_nf("PORT1")
In an AC analysis, bud_nf() can be used as below:
BudNF1=bud_nf("PORT1")
BudNF2=bud_nf("PORT1",50.0,1 Hz,2,"AC1")
BudNF3=bud_nf("PORT1",,,,,budget_path)
BudNF4=bud_nf(,,,,,budget_path)
where budget_path could be defined as:
budget_path =
["PORT1.t1","b2_AMP1.t2","b3_MIX1.t2","b4_AMP2.t2","b5_BPF2.t2","b6.t1"]
```

Defined in

$HPEESOF_DIR/expressions/ael/budget_fun.ael

See Also

bud_nf_deg() (expmeas), *bud_tn()* (expmeas)

Notes/Equations

Used in AC simulation
This is the noise figure (in dB) from the input port to the terminal(s) of each component, looking into that component. The noise analysis control parameters in the AC Simulation component must be selected: "Calculate Noise" and "Include port noise". For the source, the parameter "Noise" should be set to yes. The noise figure is always calculated per IEEE standard definition with the input termination at 290 K.

Note
Remember that the budget function can refer only to the default dataset, that is, the dataset selected in the data display window.

Budget Path Measurements

Instead of all components in alphabetical order, this function can report its values just for the components selected in a budget path, and following the sequence in that path. To facilitate the budget path measurements the name of the budget path variable, as defined in the Schematic window, needs to be entered as the pinNumber argument.

bud_nf_deg()

Returns budget noise figure degradation

Syntax

y = bud_nf_deg(vIn, iIn, vOut, iOut, vOut.NC.vnc, vOut.NC.name, Zs, BW)
y = bud_nf_deg("PORT1", "Term1", "vOut")

Arguments

Name	Description	Default	Range	Type	Required
vIn	voltage flowing into the input port	None	(-∞:∞)	Complex	Yes
iIn	current flowing into the input port	None	(-∞:∞)	Complex	Yes
vOut	voltage flowing into the output port	None	(-∞:∞)	Complex	Yes
iOut	current flowing into the output port	None	(-∞:∞)	Complex	Yes
vOut.NC.vnc	noise contributions at the output port	None	None	String	Yes
vOut.NC.name	noise contributions component names at the output port	None	None	String	Yes
Zs	input source port impedance	50.0	[0:∞)	Real	No
BW †	bandwidth	1	[1:∞)	Real	No

† BW must be set as the value of Bandwidth used on the noise page of the AC controller

Examples

```
x = bud_nf_deg(PORT1.t1.v, PORT1.t1.i, Term1.t1.v, Term1.t1.i, vOut.NC.vnc,
vOut.NC.name)
x = bud_nf_deg("PORT1", "Term1", "vOut")
```

Defined in

$HPEESOF_DIR/expressions/ael/budget_fun.ael

See Also

bud_nf() (expmeas), *bud_tn()* (expmeas)

Notes/Equations

Used in AC simulation
The improvement of system noise figure is given when each element is made noiseless. This is the noise figure (in dB) from the source port to a specified output port, obtained while setting each component noiseless, one at a time. The noise analysis and noise contribution control parameters in the AC Simulation component must be selected. For noise contribution, the output network node must be labeled and referenced on the noise page in the AC Controller. Noise contributors mode should be set to "Sort by Name." The option "Include port noise "on the AC Controller should be selected. For the source, the parameter "Noise" should be set to yes. For this particular budget measurement the AC controller parameter "OutputBudgetIV" can be set to no. The noise figure is always calculated per IEEE standard definition with the input termination at 290 K.

Note
Remember that the budget function can refer only to the default dataset, that is, the dataset selected in the data display window.

Budget Path Measurements

This function does not support the budget path feature.

bud_noise_pwr()

Returns the budget noise power

Syntax

y = bud_noise_pwr(Zref, Plan, pinNumber, "simName")

Arguments

Name	Description	Default	Range	Type	Required
Zref	input source port impedance	50.0	[0:∞)	Real	No
Plan	number of the selected frequency plan(needed only for HB)	None	None	String	No
pinNumber	Used to choose which pins of each network element are referenced †	1	[1:∞)	Integer	No
simName	simulation instance name, such as "AC1" or "HB1", used to qualify the data when multiple simulations are performed.	None	None	String	No

† If 1 is passed as the pinNumber, the results at pin 1 of each element are returned; otherwise, the results for all pins of each element are returned. By default, the pinNumber is set to 1.

Examples

```
x = bud_noise_pwr() returns the noise power at all frequencies
y = bud_noise_pwr(75, 1) returns the noise power at reference frequencies in
plan 1
```

Defined in

$HPEESOF_DIR/expressions/ael/budget_fun.ael

See Also

bud_pwr() (expmeas)

Notes/Equations

Used in AC and harmonic balance simulations
This is the noise power (in dBm) at the terminal(s) of each component, looking into the component. If Zref is not specified, the impedance that relates the signal voltage and

current is used to calculate the noise power. Note that the fundamental frequency at different pins can be different, and therefore values are given for all frequencies unless a Plan is referenced.

Note
Remember that the budget function can refer only to the default dataset, that is, the dataset selected in the data display window.

This function does not have the bandwidth parameter, so the results rely entirely on the setting of the bandwidth in the Noise tab of the simulation controller.
For AC noise and budget calculations, the bandwidth parameter flatly scales the noise voltages, and consequently noise powers, SNR, etc., regardless of the frequency response. If you make a frequency sweep then the narrow band noise voltages properly follow the frequency characteristic of the circuit and are presented as a function of the swept frequency values. However, changing the bandwidth in the simulator controller would again just flatly rescale all the values, regardless of whether the frequency response remains flat or changes drastically over the (local) bandwidth, or whether the adjacent bands overlap or not.
The only true integration over a bandwidth is done for the phase noise (as one of the options in the NoiseCon controller).

A work-around solution is to do an appropriately wide frequency sweep setting the bandwidth value in the AC controller to that of the frequency step. Then adding all the powers (need to be first converted from dBm to watts) would do the integration.

Budget Path Measurements

Instead of all components in alphabetical order, this function can report its values just for the components selected in a budget path, and following the sequence in that path. To facilitate the budget path measurements the name of the budget path variable, as defined in the Schematic window, needs to be entered as the pinNumber argument.

bud_pwr()

Returns the budget signal power in dBm

Syntax

y = bud_pwr(Plan, pinNumber, "simName")

Arguments

Name	Description	Default	Range	Type	Required
Plan	number of the selected frequency plan(needed only for HB)	None	None	String	No
pinNumber	Used to choose which pins of each network element are referenced †	1	[1:∞)	Integer	No
simName	simulation instance name, such as "AC1" or "HB1", used to qualify the data when multiple simulations are performed.	None	None	String	No

† If 1 is passed as the pinNumber, the results at pin 1 of each element are returned; otherwise, the results for all pins of each element are returned. By default, the pinNumber is set to 1.

Examples

```
x = bud_pwr() returns the signal power at all frequencies when used in AC or HB
simulations
y = bud_pwr(50, 1) returns the signal power at reference frequencies in plan 1
when used for HB simulations
```

Defined in

$HPEESOF_DIR/expressions/ael/budget_fun.ael

See Also

bud_noise_pwr() (expmeas)

Notes/Equations

Used in AC and harmonic balance simulations.
This is the signal power (in dBm) at the terminal(s) of each component, looking into the component. Note that the fundamental frequency at different pins can be different, and therefore values are given for all frequencies unless a Plan is referenced.

Note
Remember that the budget function can refer only to the default dataset, that is, the dataset selected in the data display window.

Budget Path Measurements

Instead of all components in alphabetical order, this function can report its values just for the components selected in a budget path, and following the sequence in that path. To facilitate the budget path measurements the name of the budget path variable, as defined in the Schematic window, needs to be entered as the pinNumber argument.

bud_pwr_inc()

Returns the budget incident power.

Syntax

y = bud_pwr_inc(Zref, Plan, pinNumber, "simName")

Arguments

Name	Description	Default	Range	Type	Required
Zref	input source port impedance	50.0	[0:∞)	Real	No
Plan	number of the selected frequency plan(needed only for HB)	None	None	String	No
pinNumber	Used to choose which pins of each network element are referenced †	1	[1:∞)	Integer	No
simName	simulation instance name, such as "AC1" or "HB1", used to qualify the data when multiple simulations are performed.	None	None	String	No

† If 1 is passed as the pinNumber, the results at pin 1 of each element are returned; otherwise, the results for all pins of each element are returned. By default, the pinNumber is set to 1.

Examples

```
x = bud_pwr_inc() returns incident power at all frequencies
y = bud_pwr_inc(75, 1) returns incident power at reference frequencies in plan
1
```

Defined in

$HPEESOF_DIR/expressions/ael/budget_fun.ael

See Also

bud_pwr_refl() (expmeas)

Notes/Equations

Used in AC and harmonic balance simulations
This is the incident power (in dBm) at the terminal(s) of each component, looking into the component. Note that the fundamental frequency at different pins can be different, and therefore values are given for all frequencies unless a Plan is referenced.

Note
Remember that the budget function can refer only to the default dataset, that is, the dataset selected in the data display window.

Budget Path Measurements

Instead of all components in alphabetical order, this function can report its values just for the components selected in a budget path, and following the sequence in that path. To facilitate the budget path measurements the name of the budget path variable, as defined in the Schematic window, needs to be entered as the pinNumber argument.

bud_pwr_refl()

Returns the budget reflected power

Syntax

y = bud_pwr_refl(Zref, Plan, pinNumber, "simName")

Arguments

Name	Description	Default	Range	Type	Required
Zref	input source port impedance	50.0	[0:∞)	Real	No
Plan	number of the selected frequency plan(needed only for HB)	None	None	String	No
pinNumber	Used to choose which pins of each network element are referenced †	1	[1:∞)	Integer	No
simName	simulation instance name, such as "AC1" or "HB1", used to qualify the data when multiple simulations are performed.	None	None	String	No

† If 1 is passed as the pinNumber, the results at pin 1 of each element are returned; otherwise, the results for all pins of each element are returned. By default, the pinNumber is set to 1.

Examples

```
x = bud_pwr_refl() returns reflected power at all frequencies
y = bud_pwr_refl(75, 1) returns reflected power at reference frequencies in
plan 1
```

Defined in

$HPEESOF_DIR/expressions/ael/budget_fun.ael

See Also

bud_pwr_inc() (expmeas)

Notes/Equations

Used in AC and harmonic balance simulations
This is the reflected power (in dBm) at the terminal(s) of each component, looking into the component. Note that the fundamental frequency at different pins can be different, and therefore values are given for all frequencies unless a Plan is referenced.

Note
Remember that the budget function can refer only to the default dataset, that is, the dataset selected in the data display window.

Budget Path Measurements

Instead of all components in alphabetical order, this function can report its values just for the components selected in a budget path, and following the sequence in that path. To facilitate the budget path measurements the name of the budget path variable, as defined

in the Schematic window, needs to be entered as the pinNumber argument.

bud_snr()

Returns the budget signal-to-noise-power ratio

Syntax

y = bud_snr(Plan, pinNumber, "simName")

Arguments

Name	Description	Default	Range	Type	Required
Plan	number of the selected frequency plan(needed only for HB)	None	None	String	No
pinNumber	Used to choose which pins of each network element are referenced †	1	[1:∞)	Integer	No
simName	simulation instance name, such as "AC1" or "HB1", used to qualify the data when multiple simulations are performed.	None	None	String	No

† If 1 is passed as the pinNumber, the results at pin 1 of each element are returned; otherwise, the results for all pins of each element are returned. By default, the pinNumber is set to 1.

Examples

```
x = bud_snr() returns the SNR at all frequencies
y = bud_snr(1) returns the SNR at reference frequencies in plan 1
```

Defined in

$HPEESOF_DIR/expressions/ael/budget_fun.ael

Notes/Equations

Used in AC and harmonic balance simulations
This is the SNR (in dB) at the terminal(s) of each component, looking into that component. Note that the fundamental frequency at different pins can in general be different, and therefore values are given for all frequencies unless a Plan is referenced. The noise analysis control parameter in the AC and Harmonic Balance Simulation components must be selected. For the AC Simulation component select: "Calculate Noise" and "Include port noise." For the source, the parameter "Noise" should be set to yes. In Harmonic Balance select the "Nonlinear noise" option.

Note
Remember that the budget function can refer only to the default dataset, that is, the dataset selected in the data display window.

Budget Path Measurements

Instead of all components in alphabetical order, this function can report its values just for the components selected in a budget path, and following the sequence in that path. To

facilitate the budget path measurements the name of the budget path variable, as defined in the Schematic window, needs to be entered as the pinNumber argument.

bud_tn()

Returns the budget equivalent output-noise temperature

Syntax

y = bud_tn(vIn, iIn, noisevIn, Zs, BW, pinNumber, "simName") or y = bud_tn("SourceName")

Arguments

Name	Description	Default	Range	Type	Required
vIn	voltage flowing into the input port	None	(-∞:∞)	Complex	Yes
iIn	current flowing into the input port	None	(-∞:∞)	Complex	Yes
noisevIn	noise input at the input port	None	(-∞:∞)	Complex	Yes
Zs	input source port impedance	50.0	[0:∞)	Real	No
BW †	bandwidth	1	[1:∞)	Real	No
pinNumber	Used to choose which pins of each network element are referenced ††	1	[1:∞)	Integer	No
simName	simulation instance name, such as "AC1" or "HB1", used to qualify the data when multiple simulations are performed.	None	None	String	No
SourceName	component name at the input port	None	None	String	Yes

† BW must be set as the value of Bandwidth used on the noise page of the AC controller
†† If 1 is passed as the pinNumber, the results at pin 1 of each element are returned; otherwise, the results for all pins of each element are returned. By default, the pinNumber is set to 1.

Examples

```
x = bud_tn(PORT1.t1.v, PORT1.t1.i, PORT1.t1.v.noise)
x = bud_tn("PORT1")
```

Defined in

$HPEESOF_DIR/expressions/ael/budget_fun.ael

See Also

bud_nf() (expmeas), *bud_nf_deg()* (expmeas)

Notes/Equations

Used in AC simulation

This is an equivalent output-noise temperature (in Kelvin) from the input port to the terminal(s) of each component, looking into that component. The noise analysis and noise contribution control parameters in the AC Simulation component must be selected: "Calculate Noise" and "Include port noise." For the source, the parameter "Noise" should be set to yes. The output-noise temperature is always calculated per IEEE standard definition with the input termination at 290 K.

Note
Remember that the budget function can refer only to the default dataset, that is, the dataset selected in the data display window.

Budget Path Measurements

Instead of all components in alphabetical order, this function can report its values just for the components selected in a budget path, and following the sequence in that path. To facilitate the budget path measurements the name of the budget path variable, as defined in the Schematic window, needs to be entered as the pinNumber argument.

bud_vswr()

Returns the budget voltage-standing-wave ratio

Syntax

y = bud_vswr(Zref, Plan, pinNumber, "simName")

Arguments

Name	Description	Default	Range	Type	Required
Zref	input source port impedance	50.0	[0:∞)	Real	No
Plan	number of the selected frequency plan(needed only for HB)	None	None	String	No
pinNumber	Used to choose which pins of each network element are referenced †	1	[1:∞)	Integer	No
simName	simulation instance name, such as "AC1" or "HB1", used to qualify the data when multiple simulations are performed.	None	None	String	No

† If 1 is passed as the pinNumber, the results at pin 1 of each element are returned; otherwise, the results for all pins of each element are returned. By default, the pinNumber is set to 1.

Examples

```
x = bud_vswr() returns the vswr at all frequencies
y = bud_vswr(75, 1) returns the vswr at reference frequencies in plan 1
```

Defined in

$HPEESOF_DIR/expressions/ael/budget_fun.ael

See Also

bud_gamma() (expmeas)

Notes/Equations

Used in AC and harmonic balance simulations
This is the VSWR looking into the terminal(s) of each component. Note that the fundamental frequency at different pins can be different, and therefore values are given for all frequencies unless a Plan is referenced.

Note
Remember that the budget function can refer only to the default dataset, that is, the dataset selected in the data display window.

Budget Path Measurements

Instead of all components in alphabetical order, this function can report its values just for the components selected in a budget path, and following the sequence in that path. To facilitate the budget path measurements the name of the budget path variable, as defined in the Schematic window, needs to be entered as the pinNumber argument.

Circuit Envelope Functions

This section describes the circuit envelope functions in detail. The functions are listed in alphabetical order.

- *ACPR ChPwr or EVM from 1tone swp()* (expmeas)
- *acpr vi()* (expmeas)
- *acpr vr()* (expmeas)
- *channel power vi()* (expmeas)
- *channel power vr()* (expmeas)
- *const evm()* (expmeas)
- *cross hist()* (expmeas)
- *delay path()* (expmeas)
- *evm wlan dsss cck pbcc()* (expmeas)
- *evm wlan ofdm()* (expmeas)
- *fs()* (expmeas)
- *Mod Data from 1tone swpUNI()* (expmeas)
- *peak pwr()* (expmeas)
- *peak to avg pwr()* (expmeas)
- *power ccdf()* (expmeas)
- *power ccdf ref()* (expmeas)
- *pwr vs t()* (expmeas)
- *relative noise bw()* (expmeas)
- *sample delay pi4dqpsk()* (expmeas)
- *sample delay qpsk()* (expmeas)
- *spectrum analyzer()* (expmeas)
- *total pwr()* (expmeas)
- *trajectory()* (expmeas)

Working with Envelope Data

Circuit Envelope Analysis produces complex frequency spectra as a function of time. A single envelope analysis can produce 2-dimensional data where the outermost independent variable is time and the innermost is frequency or harmonic number. Indexing can be used to look at a harmonic against time, or a spectrum at a particular time index.

ACPR_ChPwr_or_EVM_from_1tone_swp()

Returns an amplifier's adjacent or alternate channel power ratios, or main channel power, or error vector magnitude

Syntax

ACPR_ChPwr_or_EVM_from_1tone_swp(returnVal, algorithm, allowextrap, charVoltage, inputSig, sourceZ, loadZ, mainCh, lowerAdjCh, upperAdjCh, winType, winConst)

Arguments

Name	Description	Default	Range	Type	Required
returnVal	Specifies what the function is to return. Use "ACPR" for Adjacent (or Alternate) Channel Power Ratio (dBc), "MAINCHP" for Main Channel Power (dBm), or "EVM" for Error Vector Magnitude (%).	None	"ACPR", "MAINCHP", or "EVM"	String	Yes
algorithm	Specifies the algorithm to be used to model the vout-versus-vin data from the HB sweep. Use "CF" for Curve Fit or "LI" for Linear Interpolation.	None	"LI" or "CF"	String	Yes
allowextrap	Allow or disallow extrapolation when applying the scaled, modulated input signal to the vout-versus-vin model.	1	0, "No", "NO", "no", 1, "Yes", "YES", "yes"	String or Integer	No
charVoltage	This is the characterization voltage (the fundamental output voltage from the harmonic balance sweep.) Example: Vload_fund, where Vload_fund=Vload[1].	None	(-∞:∞)	Complex	Yes
inputSig	This is the input modulated signal (the envelope.) This signal should be a function of time, only.	None	(-∞:∞)	Complex	Yes
sourceZ	This is the source impedance. This can be a swept parameter.	None	(0:∞)	Complex	Yes
loadZ	This is the load impedance. This can be a swept parameter.	None	(0:∞)	Complex	Yes
mainCh	These are the main channel frequency limits, as an offset from the carrier frequency. Example: {(-3.84 MHz/2),(3.84 MHz/2)}	None	(-∞:∞)	1X2 matrix, Real or Integer	Yes
lowerAdjCh	These are the lower adjacent (or alternate) channel frequency limits as an offset from the carrier frequency. Example: MainLimits - (5 MHz)	None	(-∞:∞)	1X2 matrix, Real or Integer	Yes
upperAdjCh	These are the upper adjacent (or alternate) channel frequency limits as an offset from the carrier frequency. Example: MainLimits + (5 MHz)	None	(-∞:∞)	1X2 matrix, Real or Integer	Yes
winType	window type	"Kaiser"	†	string	No
winConst	window constant that affects the shape of the applied window.	depends on winType used	[0:∞)	Real	No

† winType can be: "none", "hamming", "hanning", "gaussian", "kaiser", "8510", "blackman","blackman-harris"

Examples

```
Vload_fundHB=Vload[1]
Vin_fund=_3GPPFDD_UE_Tx_12_2_SigGen..Vsource
Zsource=50
Zload=50
MainLimits={-3.84 MHz/2,3.84 MHz/2}
LoChLimits=MainLimits-(5 MHz)
UpChLimits=MainLimits+(5 MHz)
LoChLimitsAlt=MainLimits-(10 MHz)
UpChLimitsAlt=MainLimits+(10 MHz)
ACPR_dBc=ACPR_ChPwr_or_EVM_from_1tone_swp("ACPR", "LI", "Yes", Vload_fundHB,
```

```
Vin_fund, Zsource, Zload, MainLimits, LoChLimits, UpChLimits, "Kaiser",)
AltCPR_dBc=ACPR_ChPwr_or_EVM_from_1tone_swp("ACPR", "LI", "Yes", Vload_fundHB,
Vin_fund, Zsource, Zload, MainLimits, LoChLimitsAlt, UpChLimitsAlt, "Kaiser",)
Pout_dBm=ACPR_ChPwr_or_EVM_from_1tone_swp("MAINCHP", "LI", "Yes", Vload_fundHB,
Vin_fund, Zsource, Zload, MainLimits, LoChLimits, UpChLimits, "Kaiser",)
ACPR_vs_Pout=vs(ACPR_dBc,Pout_dBm)
AltCPR_vs_Pout=vs(AltCPR_dBc,Pout_dBm)
EVM_percent=ACPR_ChPwr_or_EVM_from_1tone_swp("EVM", "LI", "Yes", Vload_fundHB,
Vin_fund, Zsource, Zload, MainLimits, LoChLimits, UpChLimits, "Kaiser",)
EVM_vs_Pout=vs(EVM_percent,Pout_dBm)
```

See Also

Mod_Data_from_1toneSwpUNI() (expmeas)

Notes/Equations

This function returns the adjacent or alternate channel power ratio (in dBc), main channel power (in dBm), or EVM in percent.
It can be used in a measurement expression in a schematic or in the data display. While it is slower to use the ACPR_ChPwr_or_EVM_from_1tone_swp() function on the schematic than the Mod_Data_from_1tone_swpUNI() in the data display, the advantage is that the results are written into the dataset, and data displays that show these results open instantly.
However, when using ACPR_ChPwr_or_EVM_from_1tone_swp(), you have to call it once to get the adjacent channel power ratios, once
again to get the alternate channel power ratios, once again to get the main channel power, and once again to get the EVM. The big disadvantage of using the Mod_Data_from_1tone_swpUNI() is that this function will get executed each time you open a data display that contains it.

For EVM, this single-tone method is not specification-compliant. It just measures the “raw” EVM, computed at each time point. The EVM is computed after correcting for the average phase difference and RMS amplitude difference between the output and input modulated signals. If the modulated signal at the output of the amplifier has only a constant phase shift and a constant gain (meaning that neither vary with the amplitude of the input modulated signal), then the EVM will be zero. With this method, the EVM is computed at each time point, not at just the symbol times. There is no demodulation or decoding of the signal, so you can’t calculate the EVM of each sub-carrier, say for an LTE signal.

For ACPR, this single-tone method does not include any receive-side filtering. It just generates the spectrum at the output of the amplifier, integrates the power in the main, adjacent, and alternate channels, then computes the ratios. The single-tone method of computing EVM (and ACPR) will tend to become less accurate as the bandwidth of the signal gets larger. This is because this method assumes the response of the amplifier is constant across the modulation bandwidth (we’re modeling the nonlinearity by injecting a single tone at the carrier frequency, after all.)

acpr_vi()

Computes the adjacent-channel power ratio following a Circuit Envelope simulation

Syntax

ACPRvals = acpr_vi(voltage, current, mainCh, lowerAdjCh, upperAdjCh, winType, winConst)

Arguments

Name	Description	Default	Range	Type	Required
voltage	single complex voltage spectral component (for example, the fundamental) across a load versus time	None	(-∞:∞)	Complex	Yes
current	single complex current spectral component into the same load versus time	None	(-∞:∞)	Complex	Yes
mainCh	two-dimensional vector defining the main channel frequency limits (as an offset from the single voltage and current spectral component)	None	(-∞:∞)	Real	Yes
lowerAdjCh	the two-dimensional vector defining the lower adjacent-channel frequency limits (as an offset from the single voltage and current spectral component);	None	(-∞:∞)	Real	Yes
upperAdjCh	two-dimensional vector defining the upper adjacent channel frequency limits (as an offset from the single voltage and current spectral component);	None	(-∞:∞)	Real	Yes
winType	window type	None	†	string	No
winConst	window constant that affects the shape of the applied window.	0.75	[0:∞)	Real	No

† winType can be: "none", "hamming", "hanning", "gaussian", "kaiser", "8510", "blackman","blackman-harris"

Examples

```
VloadFund = vload[1]
IloadFund = iload.i[1]
mainlimits = {-16.4 kHz, 16.4 kHz}
UpChlimits = {mainlimits + 30 kHz}
LoChlimits = {mainlimits - 30 kHz}
TransACPR = acpr_vi(VloadFund, IloadFund, mainlimits, LoChlimits, UpChlimits,
"Kaiser")
where vload is the named connection at a load, and iload.i is the name of the
current probe that samples the current into the node. The {} braces are used to
define vectors, and the upper channel limit and lower channel limit frequencies
do not need to be defined by means of the vector that defines the main channel
limits.
examples/RF_Board/NADC_PA_wrk/NADC_PA_ACPRtransmitted.dds
```

Note
acpr_vi() function is intended to be used in Circuit Envelope design as MeasEqn or in resulting Circuit Envelope simulation's data display as an Eqn equation. If you want to do similar function in Ptolemy environment, recommend to look for ACPR/ACLR examples on support website and/or use the spec_power() function implement similar effect in Ptolemy.

Defined in

$HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael

See Also

acpr_vr() (expmeas), *channel_power_vi()* (expmeas), *channel_power_vr()* (expmeas), *relative_noise_bw()* (expmeas)

Notes/Equations

Used in Adjacent-channel power computations.
The user must supply a single complex voltage spectral component (for example, the fundamental) across a load versus time and a single complex current spectral component into the same load. The user must also supply the upper and lower adjacent-channel and main-channel frequency limits, as offsets from the spectral component frequency of the voltage and current. These frequency limits must be entered as two-dimensional vectors. An optional window and window constant may also be supplied, for use in processing non-periodic data.

acpr_vr()

Computes the adjacent-channel power ratio following a Circuit Envelope simulation.

Syntax

ACPRvals = acpr_vr(voltage, resistance, mainCh, lowerAdjCh, upperAdjCh, winType, winConst)

Arguments

Name	Description	Default	Range	Type	Required
voltage	single complex voltage spectral component (for example, the fundamental) across a load versus time	None	(-∞:∞)	Complex	Yes
resistance	load resistance in ohms	50	(-∞:∞)	Complex	No
mainCh	two-dimensional vector defining the main channel frequency limits (as an offset from the single voltage spectral component)	None	(-∞:∞)	Real	Yes
lowerAdjCh	the two-dimensional vector defining the lower adjacent-channel frequency limits (as an offset from the single voltage spectral component);	None	(-∞:∞)	Real	Yes
upperAdjCh	two-dimensional vector defining the upper adjacent channel frequency limits (as an offset from the single voltage spectral component);	None	(-∞:∞)	Real	Yes
winType	window type	None	†	string	No
winConst	window constant that affects the shape of the applied window.	0.75	[0:∞)	Real	No

† winType can be: "none", "hamming", "hanning", "gaussian", "kaiser", "8510", "blackman","blackman-harris"

Examples

```
Vfund = vOut[1]
mainlimits = {-(1.2288 MHz/2), (1.2288 MHz/2)}
UpChlimits = {885 kHz, 915 kHz}
LoChlimits = {-915 kHz, -885 kHz}
```

```
TransACPR = acpr_vr(VloadFund, 50, mainlimits, LoChlimits, UpChlimits,
"Kaiser")
where vOut is the named connection at a resistive load. The {} braces are used
to define vectors.
Note vOut is a named connection on the schematic. Assuming that a Circuit
Envelope simulation was run, vOut is output to the dataset as a two-dimensional
matrix. The first dimension is time, and there is a value for each time point
in the simulation. The second dimension is frequency, and there is a value for
each fundamental frequency, each harmonic, and each mixing term in the
analysis, as well as the baseband term.
vOut[1] is the equivalent of vOut[::, 1], and specifies all time points at the
lowest non-baseband frequency (the fundamental analysis frequency, unless a
multitone analysis has been run and there are mixing products). For former MDS
users, the notation "vOut[*, 2]" in MDS corresponds to the notation of
"vOut[1]".
examples/Tutorial/ModSources_wrk/IS95RevLinkSrc.dds
```

Note
acpr_vr() function is intended to be used in Circuit Envelope design as MeasEqn or in resulting Circuit Envelope simulation's data display as an Eqn equation. If you want to do similar function in Ptolemy environment, recommend to look for ACPR/ACLR examples on support website and/or use the spec_power() function implement similar effect in Ptolemy.

Defined in

$HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael

See Also

acpr_vi() (expmeas), *channel_power_vi()* (expmeas), *channel_power_vr()* (expmeas), *relative_noise_bw()* (expmeas)

Notes/Equations

Used in Adjacent-channel power computations.
The user must supply a single complex voltage spectral component (for example, the fundamental) across a resistive load versus time and the load resistance. The user must also supply the upper and lower adjacent-channel and main-channel frequency limits, as offsets from the spectral component frequency of the voltage. These frequency limits must be entered as two-dimensional vectors. An optional window and window constant may also be supplied, for use in processing non-periodic data.

channel_power_vi()

Computes the power (in watts) in an arbitrary frequency channel following a Circuit Envelope simulation.

Syntax

Channel_power = channel_power_vi(voltage, current, mainCh, winType, winConst)

Arguments

Name	Description	Default	Range	Type	Required
voltage	single complex voltage spectral component (for example, the fundamental) across a load versus time	None	(-∞:∞)	Complex	Yes
current	single complex current spectral component into the same load versus time	None	(-∞:∞)	Complex	Yes
mainCh	two-dimensional vector defining channel frequency limits (as an offset from the single voltage and current spectral component †	None	(-∞:∞)	Real	Yes
winType	window type	None	†	string	No
winConst	window constant that affects the shape of the applied window.	0.75	[0:∞)	Real	No

† note that these frequency limits do not have to be centered on the voltage and current spectral component frequency.
† winType can be: "none", "hamming", "hanning", "gaussian", "kaiser", "8510", "blackman","blackman-harris"

Examples

```
VloadFund = vload[1]
IloadFund = iload.i[1]
mainlimits = {-16.4 kHz, 16.4 kHz}
Main_Channel_Power = channel_power_vi(VloadFund, IloadFund, mainlimits,
"Kaiser")
where vload is the named connection at a load, and iload.i is the name of the
current probe that samples the current into the node. The {} braces are used to
define a vector. Note that the computed power is in watts. Use the following
equation to convert the power to dBm.
Main_Channel_Power_dBm = 10 * log(Main_Channel_Power) + 30
Do not use the dBm function, which operates on voltages.
examples/RF_Board/NADC_PA_wrk/NADC_PA_ACPRtransmitted.dds
```

Defined in

$HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael

See Also

acpr_vi() (expmeas), *acpr_vr()* (expmeas), *channel_power_vr()* (expmeas)

Notes/Equations

Used in Channel power computations.
The user must supply a single complex voltage spectral component (for example, the fundamental) across a load versus time, and a single complex current spectral component into the same load. The user must also supply the channel frequency limits, as offsets

from the spectral component frequency of the voltage and current. These frequency limits must be entered as a two-dimensional vector. An optional window and window constant may also be supplied, for use in processing non-periodic data.

channel_power_vr()

Computes the power (in watts) in an arbitrary frequency channel following a Circuit Envelope simulation.

Syntax

Channel_power = channel_power_vr(voltage, resistance, mainCh, winType, winConst)

Arguments

Name	Description	Default	Range	Type	Required
voltage	single complex voltage spectral component (for example, the fundamental) across a load versus time	None	(-∞:∞)	Complex	Yes
resistance	load resistance in ohms	50	(-∞:∞)	Complex	No
mainCh	two-dimensional vector defining the main channel frequency limits (as an offset from the single voltage spectral component) †	None	(-∞:∞)	Real	Yes
winType	window type	None	† †	string	No
winConst	window constant that affects the shape of the applied window.	0.75	[0:∞)	Real	No

† note that these frequency limits do not have to be centered on the voltage and current spectral component frequency
† † winType can be: "none", "hamming", "hanning", "gaussian", "kaiser", "8510", "blackman","blackman-harris"

Examples

```
Vmain_fund = Vmain[1]
mainlimits = {-16.4 kHz, 16.4 kHz}
Main_Channel_Power = channel_power_vr(Vmain_fund, 50, mainlimits, "Kaiser")
where Vmain is the named connection at a resistive load (50 ohms in this case.)
The {} braces are used to define a vector. Note that the computed power is in
watts. Use the equation
Main_Channel_Power_dBm = 10 * log(Main_Channel_Power) + 30
to convert the power to dBm. Do not use the dBm function, which operates on
voltages.
Example file
examples/RF_Board/NADC_PA_wrk/NADC_PA_ACPRreceived.dds
```

Defined in

$HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael

See Also

acpr_vi() (expmeas), *acpr_vr()* (expmeas), *channel_power_vi()* (expmeas)

Notes/Equations

Used in Channel power computations.
The user must supply a single complex voltage spectral component (for example, the fundamental) across a load versus time and the resistance of the load. The user must also supply the channel frequency limits, as offsets from the spectral component frequency of the voltage. These frequency limits must be entered as a two-dimensional vector. An optional window and window constant may also be supplied, for use in processing non-periodic data.

const_evm()

Takes the results of a Circuit Envelope simulation and generates data for the ideal and distorted constellation and trajectory diagrams, as well as the error vector magnitude, in percent, and a plot of the error vector magnitude versus time.

Syntax

data = const_evm(vfund_ideal, vfund_dist, symbol_rate, sampling_delay, rotation, transient_duration, path_delay)

Arguments

Name	Description	Default	Range	Type	Required
vfund_ideal	single complex voltage spectral component (for example the fundamental) that is ideal (undistorted). This could be constructed from two baseband signals instead, by using the function cmplx().	None	(-∞:∞)	Complex	Yes
vfund_dist	single complex voltage spectral component (for example, the fundamental) that has been distorted by the network being simulated. This could be constructed from two baseband signals instead, by using the function cmplx()	None	(-∞:∞)	Complex	Yes
symbol_rate	symbol rate of the modulation signal	None	[0:∞)	Integer, real	Yes
sampling_delay	sampling delay †	0	[0:∞)	Integer, real	No
rotation	parameter that rotates the constellations by that many radian ††	0	[0:∞)	Integer, real	No
transient_duration	time in seconds that causes this time duration of symbols to be eliminated from the error-vector-magnitude calculation †† †	0	[0:∞)	Integer, real	No
path_delay	time in seconds of the sum of all delays in the signal path †† ††	0	[0:∞)	Integer, real	No

† sampling_delay - (if nonzero) throws away the first delay = N seconds of data from the constellation and trajectory plots. It is also used to interpolate between simulation time points, which is necessary if the optimal symbol-sampling instant is not exactly at a simulation time point. Usually this parameter must be nonzero to generate a constellation diagram with the smallest grouping of sample points
†† rotation does not need to be entered, and it will not affect the error-vector-magnitude calculation, because both the ideal and distorted constellations will be rotated by the same

amount.
†† † Usually the filters in the simulation have transient responses, and the error-vector-magnitude calculation should not start until these transient responses have finished.
†† †† If the delay is 0, this parameter may be omitted. If it is non-zero, enter the delay value. This can be calculated by using the function delay_path().

Examples

```
rotation = -0.21
sampling_delay = 1/sym_rate[0, 0] - 0.5 * tstep[0, 0]
vfund_ideal = vOut_ideal[1]
vfund_dist = vOut_dist[1]
symbol_rate = sym_rate[0, 0]
data = const_evm(vfund_ideal, vfund_dist, symbol_rate, sampling_delay,
rotation, 1.5ms, path_delay)
where the parameter sampling_delay can be a numeric value, or in this case an
equation using sym_rate, the symbol rate of the modulated signal, and tstep,
the time step of the simulation. If these equations are to be used in a Data
Display window, sym_rate and tstep must be defined by means of a variable (VAR)
component, and they must be passed into the dataset as follows: Make the
parameter Other visible on the Envelope simulation component, and edit it so
that
Other = OutVar = sym_rate OutVar = tstep
In some cases, it may be necessary to experiment with the delay value to get
the constellation diagrams with the tightest points.
```

Note that const_evm() returns a list of data. So in the example above,
data[0]= ideal constellation
data[1]= ideal trajectory
data[2]= distorted constellation
data[3]= distorted trajectory
data[4]= error vector magnitude versus time
data[5]= percent error vector magnitude
Refer to the example file to see how these data are plotted.

Defined in

$HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael

See Also

constellation() (expmeas), *delay_path()* (expmeas), *sample_delay_pi4dqpsk()* (expmeas), *sample_delay_qpsk()* (expmeas)

Notes/Equations

Used in constellation and trajectory diagram generation and error-vector-magnitude calculation.

The user must supply a single complex voltage spectral component (for example, the fundamental) that is ideal (undistorted), as well as a single complex voltage spectral component (for example, the fundamental) that has been distorted by the network being simulated. These ideal and distorted complex voltage waveforms could be generated from baseband I and Q data. The user must also supply the symbol rate, a delay parameter, a rotation factor, and a parameter to eliminate any turn-on transient from the error-vector-magnitude calculation are optional parameters.

The error vector magnitude is computed after correcting for the average phase difference and RMS amplitude difference between the ideal and distorted constellations.

cross_hist()

Returns jitter histogram

Syntax

y = cross_hist(Vout_time, time_start, time_stop, level_low, level_high, number_of_bins, BitRate, No_of_Eye, Delay, steps)

Arguments

Name	Description	Default	Range	Type	Required
Vout_time	time domain voltage waveform	None	(-∞:∞)	Real	Yes
time_start	define the rectangular window points for jitter histogram plot	None	[0:∞)	Real	Yes
time_stop	define the rectangular window points for jitter histogram plot	None	[0:∞)	Real	Yes
level_low	define the rectangular window points for jitter histogram plot	None	[0:∞)	Real	Yes
level_high	define the rectangular window points for jitter histogram plot	None	[0:∞)	Real	Yes
number_of_bins	defines the number of bins on the time axis of eye diagram and controls the resolution of jitter histogram plot	None	[1:∞)	Integer	Yes
BitRate	bit rate of the channel and is expressed in frequency units	None	(0:∞)	Real	Yes
No_of_Eye	Used for multiple eye jitter histogram plots	None	[0:∞)	Integer	Yes
Delay	used to remove initial transient in the eye diagram and is expressed in time units	None	[0:∞)	Real	Yes
steps	represents the number of sampling points between level_low and level_high and is used for controlling the density of jitter histogram.	None	[1:∞)	Integer	Yes

Examples

```
Jitter_Histogram = cross_hist(vout, 0 ps, 100 ps, 0 V, 0.1V, 300, 10 GHz, 1,0,
20)
```

Defined in

Built in

See Also

eye_amplitude() (expmeas), *eye_closure()* (expmeas), *eye_fall_time()* (expmeas), *eye_rise_time()* (expmeas), *eye_height()* (expmeas)

Notes/Equations

Jitter histogram plots the jitter histogram of a time domain voltage waveform.

delay_path()

This function is used to determine the time delay and the constellation rotation angle between two nodal points along a signal path.

Syntax

y = delay_path(vin, vout)

Arguments

Name	Description	Default	Range	Type	Required
vin	envelope (I + j * Q) at the input node	None	(-∞:∞)	Complex	Yes
vout	I + j * Q at the output node	None	(-∞:∞)	Complex	Yes

Examples

```
x = delay_path(vin[1], vout[1])
where vin[1] and vout[1] are complex envelopes around the first carrier
frequency in envelope simulation. In return, x[0] is the time delay (in
seconds) between vin and vout. x[1] is the rotation angle (in radians) between
vin and vout constellations.
or
x = delay_path(T1, T2)
where T1 and T2 are instance names of two TimedSink components.
```

Defined in

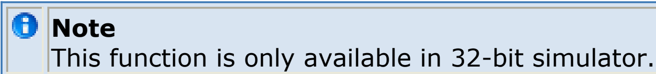
Built in

See Also

ber_pi4dqpsk() (expmeas), *ber_qpsk()* (expmeas), *const_evm()* (expmeas), *cross_corr()* (expmeas)

Notes/Equations

Used in Circuit Envelope simulation, Ptolemy simulation.
This function outputs an array of two values. The first value, data[0], is the time delay between vin and vout. The second value, data[1], is the rotation angle between vin-constellation and vout-constellation.



Note
This function is only available in 32-bit simulator.

evm_wlan_dsss_cck_pbcc()

Returns EVM (error vector magnitude) analysis results for WLAN DSSS/CCK/PBCC (IEEE 802.11b and IEEE 802.11g non-OFDM) voltage signals.

Syntax

evm = evm_wlan_dsss_cck_pbcc(voltage{, mirrorSpectrum, start, averageType, burstsToAverage, modulationFormat, searchTime, resultLengthType, resultLength, measurementOffset, measurementInterval, chipRate, clockAdjust, equalizationFilter, filterLength, descrambleMode, referenceFilter, referenceFilterBT, output})

Arguments

Name	Description	Default	Range	Type	Required
voltage	complex envelope WLAN DSSS/CCK/PBCC voltage signal	None	(-∞:∞)	Complex	Yes
mirrorSpectrum	specifies whether the input signal should be mirrored or not	NO	See Notes	String	No
start	specifies the start time for the EVM analysis	first point of the input data	[0:max(indep(voltage))]	Real	No
averageType	specifies what type of averaging is done	OFF	See Notes	String	No
burstsToAverage	number of bursts over which the results will be averaged	20	(0:∞)	Integer	No
modulationFormat	modulation format	"Auto Detect"	See Notes	String	No
searchTime	search time	550 usec	[0:max(indep(voltage))]	Real	No
resultLengthType	specifies how the result length is determined	"Auto Select"	See Notes	String	No
resultLength	result length in chips	2816	[1:108344]	Integer	No
measurementOffset	measurement offset in chips	22	(0:∞)	Integer	No
measurementInterval	measurement interval in chips	2794	(0:∞)	Integer	No
chipRate	chip rate in Hz	11e6 Hz	(0:∞)	Real	No
clockAdjust	clock adjustment as a fraction of a chip	0	[-0.5, 0.5]	Real	No
equalizationFilter	specifies whether an equalization filter is used or not	OFF	See Notes	String	No
filterLength	equalization filter length in chips	21	[3:∞) must be odd	Integer	No
descrambleMode	specifies how descrambling is done	On	See Notes	String	No
referenceFilter	specifies the reference filter to be used	Gaussian	See Notes	String	No
referenceFilterBT	BT value for the reference filter if a Gaussian reference filter is selected	0.3	[0.05:100]	Real	No
output	EVM analysis result to be returned	Avg_EVMrms_pct	See Notes	String	No

Examples

```
evmRMS = evm_wlan_dsss_cck_pbcc(Vout[1])
where Vout is a named node in a Circuit Envelope simulation, will return the
EVM rms value in percent for the voltage envelope at the fundamental frequency.
The voltage data Vout[1] must contain at least one complete burst. The EVM
result will be from the analysis of the first burst in the input signal.
evmPk = evm_wlan_dsss_cck_pbcc(Vout[1], , , "RMS (Video)", 10, , 300e-6, , , ,
, , , , , , "Gaussian", 0.5, "EVM_Pk_pct")
where Vout is a named node in a Circuit Envelope simulation, will return a
```

vector with 10 values each representing the peak EVM in percent for the first 10 bursts in the voltage envelope at the fundamental frequency. Since searchTime is set to 300 usec, the first 300 usec of Vout[1] must contain at least one complete burst. In addition, since 10 bursts need to be processed, Vout[1] must contain at least 10 complete bursts. A Gaussian filter with BT=0.5 will be used as a reference filter.

```
AvgMagErr = evm_wlan_dsss_cck_pbcc(Vout[1], , , "RMS (Video)", 3, , 400e-6, , , 100, 500, , , , , , , , "Avg_MagErr_rms_pct")
```

where Vout is a named node in a Circuit Envelope simulation, will return the average (over 3 bursts) magnitude error in percent for the voltage envelope at the fundamental frequency. Since searchTime is set to 400 usec, the first 400 usec of Vout[1] must contain at least one complete burst. In addition, since 3 bursts need to be averaged, Vout[1] must contain at least 3 complete bursts. Only the chips 101 to 600 (measurementOffset = 100 and measurementInterval = 500) will be considered for the EVM analysis.

Defined in

Built in

See Also

evm_wlan_ofdm() (expmeas)

Notes/Equations

Used in Circuit Envelope simulation and Data Flow simulation.
This expression can be used with input data of up to two dimensions. Only complex envelope input signals are allowed as input.

The evm_wlan_dsss_cck_pbcc() expression performs an EVM measurement for WLAN DSSS/CCK/PBCC (IEEE 802.11b and IEEE 802.11g non-OFDM) signals. Available Measurement Results for evm_wlan_dsss_cck_pbcc() displays the available measurement results.

Available Measurement Results for evm_wlan_dsss_cck_pbcc()

Measurement Result	Description
Avg_WLAN_80211b_1000_chip_Pk_EVM_pct	average EVM in percentage as specified by the standard (section 18.4.7.8 Transmit modulation accuracy in 802.11b specification; pages 55-57) except that the EVM value is normalized
WLAN_80211b_1000_chip_Pk_EVM_pct	EVM in percentage as specified by the standard (section 18.4.7.8 Transmit modulation accuracy in 802.11b specification; pages 55-57) with the exception that the EVM value is normalized versus burst
Avg_EVMrms_pct	average EVM rms in percentage as defined in the Agilent 89600 VSA
EVMrms_pct	EVM rms in percentage as defined in the Agilent 89600 VSA versus burst
EVM_Pk_pct	peak EVM in percentage versus burst
EVM_Pk_chip_idx	peak EVM chip index versus burst
Avg_MagErr_rms_pct	average magnitude error rms in percentage
MagErr_rms_pct	magnitude error rms in percentage versus burst
MagErr_Pk_pct	peak magnitude error in percentage versus burst
MagErr_Pk_chip_idx	peak magnitude error chip index versus burst
Avg_PhaseErr_deg	average phase error in degrees
PhaseErr_deg	phase error in degrees versus burst
PhaseErr_Pk_deg	peak phase error in degrees versus burst
PhaseErr_Pk_chip_idx	peak phase error chip index versus burst
Avg_FreqError_Hz	average frequency error in Hz
FreqError_Hz	frequency error in Hz versus burst
Avg_IQ_Offset_dB	average IQ offset in dB
IQ_Offset_dB	IQ offset in dB versus burst
Avg_SyncCorrelation	average sync correlation
SyncCorrelation	sync correlation versus burst

Results whose name is prefixed with " *Avg*"_ are averaged over the number of bursts specified by the user (if averageType is set to " *RMS* (*Video*)"). Results whose name is not prefixed with " *Avg*"_ are results versus burst.

The following is a brief description of the algorithm used (the algorithm used is the same as the one used in the Agilent 89600 VSA) and a detailed description of its arguments. Starting at the time instant specified by the start argument, a signal segment of length searchTime is acquired. This signal segment is searched in order for a complete burst to be detected. The burst search algorithm looks for both a *burst on* and a *burst off* transition. In order for the burst search algorithm to detect a burst, an idle part must exist between consecutive bursts and the bursts must be at least 15 dB above the noise floor.

If the acquired signal segment does not contain a complete burst, the algorithm will not detect any burst and the analysis that follows will most likely produce wrong results. Therefore, searchTime must be long enough to acquire at least one complete burst. Since the time instant specified by the start argument can be a little after the beginning of a burst, it is recommended that searchTime is set to a value approximately equal to 2 x burstLength, where burstLength is the duration of a burst in seconds including the duration of the idle part. If it is known that the time instant specified by the start argument is a little before the beginning of a burst, then searchTime can be set to burstLength.

After a burst is detected, synchronization is performed based on the preamble. The burst

is then demodulated. Finally, the burst is analyzed to get the EVM measurement results.

If averageType is set to *OFF* , only one burst is detected, demodulated, and analyzed.
If averageType is set to *RMS (Video)* , after the first burst is analyzed the signal segment corresponding to it is discarded and new signal samples are acquired to fill in the signal buffer of length searchTime. When the buffer is full again a new burst search is performed and when a burst is detected it is demodulated and analyzed. These steps repeat until burstsToAverage bursts are processed.

If for any reason a burst is misdetected the results from its analysis are discarded. The EVM results obtained from all the successfully detected, demodulated, and analyzed bursts are averaged to give the final result.
The mirrorSpectrum argument accepts the following strings: "NO" and "YES". This argument can be used to mirror (conjugate) the input signal before any other processing is done. Mirroring the input signal is necessary if the configuration of the mixers in your system has resulted in a mirrored signal compared to the one at the input of the up-converter and if the preamble and header are *short* format. In this case, if mirrorSpectrum is not set to " *YES* " the header bits (which carry the modulation format and length information) will not be recovered correctly so the demodulation of the PSDU part of the burst will most likely fail.

The start argument sets the starting point for acquiring the signal to be processed. By default, the starting point is the beginning of the input signal (voltage argument). However, if for any reason an initial part of the input signal needs to be omitted this can be done by setting the start argument appropriately.
The averageType argument accepts the following strings: "Off" and "RMS (Video)". This argument can be used to turn on/off video averaging. If set to "Off" the EVM result returned is from the processing of only one burst. Otherwise, multiple bursts are processed and the results are averaged.

The burstsToAverage argument set the number of bursts whose results will be averaged if averageType is set to "RMS (Video)". If averageType is set to "Off" this argument is ignored.

The modulationFormat argument accepts the following strings: "Auto Detect", "Barker 1", "Barker 2", "CCK 5.5", "CCK 11", "PBCC 5.5", "PBCC 11", "PBCC 22", "PBCC 33". This argument sets the modulation format used in the PSDU part of the burst. If modulationFormat is set to "Auto Detect", the algorithm will use the information detected in the PLCP header part of the burst to automatically determine the modulation format. Otherwise, the modulation format determined from the PLCP header is ignored and the modulation format specified by the modulationFormat argument is used in the demodulation of the PSDU part of the burst.

The searchTime argument sets the duration of the signal segment that is acquired and searched in order to detect a complete burst. Recommendations on how to set this argument are given in the brief description of the algorithm used by this expression earlier in these Notes/Equations section.

The resultLengthType argument accepts the following strings: "Auto Select" and "Manual Override". The resultLengthType and resultLength arguments control how much data is acquired and demodulated.

- When resultLengthType is set to " *Auto Select",* the measurement result length is

automatically determined from the information in the PLCP header part of the burst. In this case, the argument resultLength defines a maximum result length for the burst in symbol times; that is, if the measurement result length that is automatically detected is bigger than resultLength it will be truncated to resultLength. The maximum result length specified by the resultLength argument includes the PLCP preamble and PLCP header.
- When resultLengthType is set to " *Manual Override* ", the measurement result length is set to resultLength regardless of what is detected in the PLCP header part of the burst. The result length specified by the resultLength argument includes the PLCP preamble and PLCP header.

Measurement result length setting summarizes the differences between how *"Auto Select"* and " *Manual Override* " modes determine the measurement result length. The table lists the measurement result lengths actually used for " *Auto Select"* and " *Manual Override"* modes for three different values of the resultLength argument (3300, 2816 and 2200 chips). It is assumed that the input burst is 2816 symbols long.

Measurement result length setting

resultLengthType	resultLength	Measurement Result Length Actually Used
Auto Select	2200	2200
Auto Select	2816	2816
Auto Select	3300	2816
Manual Override	2200	2200
Manual Override	2816	2816
Manual Override	3300	3300

Note that when resultLengthType is set to " *Manual Override"* and resultLength=3300 (greater than the actual burst size) the algorithm will demodulate the full 3300 chips even though this is 484 chips beyond the burst width.

The measurementOffset and measurementInterval arguments can be used to isolate a specific segment of the burst for analysis. The values of measurementInterval and measurementOffset are in number of chips and are relative to the ideal starting point of the PLCP preamble portion of the burst. For a signal that uses the long PLCP format, the ideal starting point of the PLCP preamble is exactly 128 symbol times (128 x 11 chips) before the start of the SFD sync pattern. For a signal that uses the short PLCP format, the ideal starting point of the PLCP preamble is exactly 56 symbol times (56 x 11 chips) before the start of the SFD sync pattern.
The chipRate argument sets the fundamental chip rate of the signal to be analyzed. The default is 11 MHz, which matches the chip rate of 802.11b and 802.11g; however, this argument can be used when experimenting with signals that do not follow the standard specifications. A special case is the optional 802.11g 33 Mbit PBCC mode, where the chip rate of the transmitted signal starts at 11 MHz, but changes to 16.5 MHz in the middle of the burst. In this case chipRate should still be set to 11 MHz (the algorithm will automatically switch to 16.5 MHz at the appropriate place in the burst).

Although the algorithm synchronizes to the chip timing of the signal, it is possible for the synchronization to be slightly off. The clockAdjust argument allows the user to specify a timing offset which is added to the chip timing detected by the algorithm. This argument should only be used when trying to debug unusual signals.
The equalizationFilter argument accepts the following strings: "OFF" and "ON". This argument can be used to turn on/off the equalization filter. The filterLength argument sets

the equalization filter length (in number of chips). Using an equalization filter can dramatically improve the EVM results since the equalizer can compensate for ISI due to the transmit filter. However, it can also compensate the distortion introduced by the DUT. If the filter used in the transmitter is Gaussian, then having the equalizer off and selecting a Gaussian reference filter might be a better option.

The descrambleMode argument accepts the following strings: "Off", "Preamble Only", "Preamble & Header Only", "On". This argument can be used to control how descrambling is done.

- "Off" does no descrambling.
- "Preamble Only" descrambles only the PLCP preamble.
- "Preamble & Header Only" descrambles only the PLCP preamble and PLCP header.
- "On" descrambles all parts of the burst.
  Normally, 802.11b or 802.11g signals have all bits scrambled before transmission, so this parameter should normally be set to *"On"* . However, when debugging an 802.11b or 802.11g transmitter, it is sometimes helpful to disable scrambling in the transmitter, in which case you should disable descrambling in this component.

If the input signal's preamble is scrambled but you disable descrambling of the preamble (or vice versa), then the algorithm will not be able to synchronize to the signal properly. Similarly, if the input signal's header is scrambled but you disable descrambling of the header (or vice versa) then the algorithm will not be able to correctly identify the burst modulation type and burst length from the header.

The referenceFilter argument accepts the following strings: "Rectangular" and "Gaussian". This argument can be used to select a reference filter for the EVM analysis. Although, the IEEE 802.11b/g standards do not specify either a transmit filter or a receive filter, they do have a spectral mask requirement, and a transmitter must use some sort of transmit filter to meet the spectral mask. On the other hand, the description of the EVM measurement in the standard does not use any receive or measurement filter. The absence of the need to use any transmit or receive filter is partly because the standard has a very loose limit for EVM (35% peak on 1000 chips worth of data).

If the standard definition is followed when computing EVM, no measurement or reference filter should be used (referenceFilter must be set to "Rectangular"). However, what this means is that even a completely distortion-free input signal will still give non-zero EVM unless the input signal has a zero-ISI transmit filter. If a non-zero-ISI transmit filter is used and there is additional distortion added to the signal due to the DUT, then the EVM will measure the overall error due to both the transmit filter's ISI and the DUT distortion. Turning on the equalizer will remove most of the transmit filter's ISI but it can also remove some of the distortion introduced by the DUT. To get a better idea of the EVM due to the DUT distortion a reference filter that matches the transmit filter can be used. Currently, only "Rectangular" and "Gaussian" filters are available as reference filters.

The referenceFilterBT argument sets the BT (Bandwidth Time product) for the Gaussian reference filter. If referenceFilter is set to "Rectangular" this argument is ignored.

The output argument accepts the following strings (see Available Measurement Results for evm_wlan_dsss_cck_pbcc()): "Avg_WLAN_80211b_1000_chip_Pk_EVM_pct", "WLAN_80211b_1000_chip_Pk_EVM_pct", "Avg_EVMrms_pct", "EVMrms_pct", "EVM_Pk_pct", "EVM_Pk_chip_idx", "Avg_MagErr_rms_pct", "MagErr_rms_pct", "MagErr_Pk_pct", "MagErr_Pk_chip_idx". "Avg_PhaseErr_deg", "PhaseErr_deg",

"PhaseErr_Pk_deg", "PhaseErr_Pk_chip_idx", "Avg_FreqError_Hz", "FreqError_Hz", "Avg_IQ_Offset_dB", "IQ_Offset_dB", "Avg_SyncCorrelation", "SyncCorrelation". This argument selects which EVM analysis result will be returned.

Relationship Between WLAN_802_11b Source Parameters and evm_wlan_dsss_cck_pbcc() Expression Arguments summarizes how some of the arguments of the evm_wlan_dsss_cck_pbcc() expression should be set based on the parameter values of the WLAN_802_11b source.

Relationship Between WLAN_802_11b Source Parameters and evm_wlan_dsss_cck_pbcc() Expression Arguments

WLAN_802_11b	evm_wlan_dsss_cck_pbcc()	Comments
DataRate (default is 11 Mbps)	searchTime (default is 550 μsec)	The recommended searchTime is !expmeas-05-09-01.gif! . BurstLength= tRamp+tPLCP+tPSDU+IdleInterval, where, tRamp = !expmeas-05-09-02.gif! tPLCP = !expmeas-05-09-03.gif! tPSDU = !expmeas-05-09-04.gif!
PreambleFormat (default is Long)		
PwrRamp (default is None)		
IdleInterval (default is 10 μsec)		
DataLength (default is 100)		
MirrorSpectrum (default is NO)	mirrorSpectrum (default is NO)	If DUT introduces spectrum mirroring, then mirrorSpectrum must be set to "NO" ("YES") when MirrorSpectrum is set to "YES" ("NO"); otherwise mirrorSpectrum must be set to the same value as MirrorSpectrum.
FilterType (default is Gaussian)	referenceFilter (default is Gaussian)	When FilterType is set to "Gaussian", the recommended setting of referenceFilter is "Gaussian"; otherwise, the recommended setting is "Rectangular"
GaussianFilter_bT (default is 0.3)	referenceFilterBT (default is 0.3)	The recommended setting of referenceFilterBT is the same value as GaussianFilter_bT. This parameter is only used when referenceFilter is set to "Gaussian".

Note
This function is only available in 32-bit simulator.

evm_wlan_ofdm()

Returns EVM (error vector magnitude) analysis results for WLAN OFDM (IEEE 802.11a) voltage signals

Syntax

evm = evm_wlan_ofdm(voltage{, mirrorSpectrum, start, averageType, burstsToAverage, subcarrierModulation, guardInterval, searchTime, resultLengthType, resultLength, measurementOffset, measurementInterval, subcarrierSpacing, symbolTimingAdjust, sync, output})

Arguments

Name	Description	Default	Range	Type	Required
voltage	complex envelope WLAN OFDM (orthogonal frequency division multiplexing) voltage signal	None	(-∞:∞)	Complex	Yes
mirrorSpectrum	specifies whether the input signal should be mirrored or not	NO	See Notes	String	No
start	specifies the start time for the EVM analysis	first point of the input data	[0:max(indep(voltage))]	Real	No
averageType	specifies what type of averaging is done	OFF	See Notes	String	No
burstsToAverage	number of bursts over which the results will be averaged	20	(0:∞)	Integer	No
subcarrierModulation	data subcarrier modulation format	"Auto Detect"	See Notes	String	No
guardInterval	guard interval length for the OFDM symbols (as a fraction of the FFT time period)	0.25	[0:1]	Real	No
searchTime	search time	80 usec	[0:max(indep(voltage))]	Real	No
resultLengthType	specifies how the result length is determined	"Auto Select"	See Notes	String	No
resultLength	result length in OFDM symbols	60	[1:1367]	Integer	No
measurementOffset	measurement offset in OFDM symbols	0	[0:∞)	Integer	No
measurementInterval	measurement interval in OFDM symbols	11	(0:∞)	Integer	No
subcarrierSpacing	frequency spacing between the subcarriers	312.5 kH	(0:∞)	Real	No
symbolTimingAdjust	specifies (as a percent of the FFT time period) the timing adjustment done on the OFDM symbols before performing the FFT	-3.125	[-100*guardInterval:0]	Real	No
sync	preamble sequence that will be used for synchronization	Short Training Seq	See Notes	String	No
output	EVM analysis result to be returned	EVMrms_percent	See Notes	String	No

Examples

```
evmRMS = evm_wlan_ofdm(Vout[1])
where Vout is a named node in a Circuit Envelope simulation, will return the
evm rms value in percent for the voltage envelope at the fundamental frequency.
The voltage data Vout[1] must contain at least one complete OFDM burst.
iqOffset = evm_wlan_ofdm( Vout[1], , , "RMS (Video)", 5, , 0.125, 200e-6, , ,
5, 10, , , , "IQ_Offset_dB")
where Vout is a named node in a Circuit Envelope simulation, will return the IQ
offset in dB for the voltage envelope at the fundamental frequency. Five bursts
```

```
will be analyzed and their results averaged. The guard interval used in the
generation of the input signal must be 0.125. Since searchTime is set to 200
usec, the first 200 usec of Vout[1] must contain at least one complete OFDM
burst. In addition, since 5 bursts need to be averaged Vout[1] must contain at
least 5 complete OFDM bursts. Only the OFDM symbols 6 to 15 (measurementOffset
= 5 and measurementInterval = 10) will be considered for the EVM analysis.
```

Defined in

Built in

See Also

evm_wlan_dsss_cck_pbcc() (expmeas)

Notes/Equations

Used in Circuit Envelope simulation and Data Flow simulation.

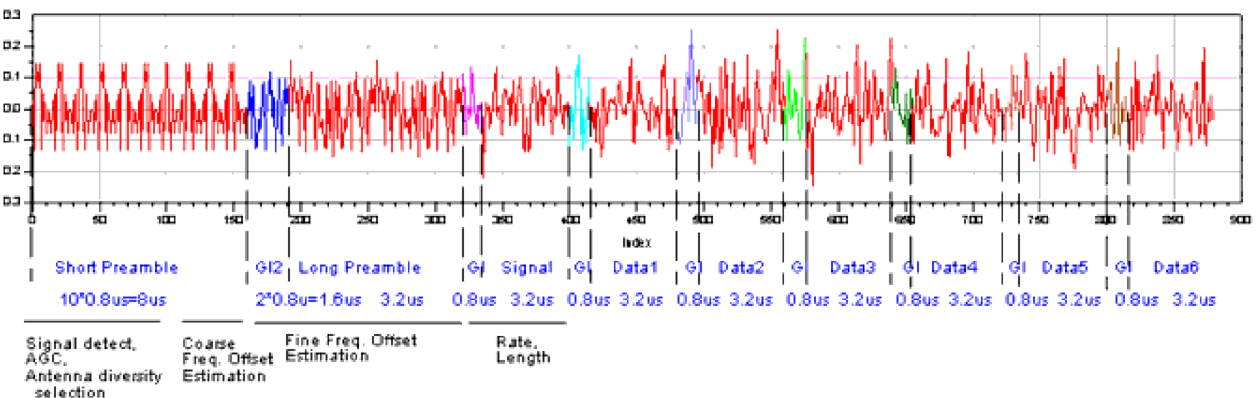
This expression can be used with input data of up to two dimensions. Only complex envelope input signals are allowed as input.

The evm_wlan_ofdm() expression performs an EVM measurement for WLAN OFDM (IEEE 802.11a) signals. Available Measurement Results for evm_wlan_ofdm() seen below, displays the available measurement results.

Available Measurement Results for evm_wlan_ofdm()

Measurement Result	**Description**
EVMrms_percent	average EVM rms in percentage
EVM_dB	average EVM in dB
PilotEVM_dB	average pilot EVM in dB
CPErms_percent	average Common Pilot Error rms in percentage
IQ_Offset_dB	average IQ offset in dB
SyncCorrelation	average sync correlation

The following is a brief description of the algorithm used (the algorithm used is the same as the one used in the Agilent 89600 VSA) and a detailed description of its arguments. Structure of an OFDM burst. shows the structure of an OFDM burst. Many of the terms mentioned later in these notes such as the preamble, SIGNAL symbol, DATA symbols, guard intervals (GI) are shown in this figure.



Structure of an OFDM burst.

Starting at the time instant specified by the start argument, a signal segment of length searchTime is acquired. This signal segment is searched in order for a complete burst to be detected. The burst search algorithm looks for both a *burst on* and a *burst off* transition. In order for the burst search algorithm to detect a burst, an idle part must exist between consecutive bursts and the bursts must be at least 15 dB above the noise floor.

If the acquired signal segment does not contain a complete burst, the algorithm will not detect any burst and the analysis that follows will most likely produce incorrect results. Therefore, searchTime must be long enough to acquire at least one complete burst. Since the time instant specified by the start argument can be a little after the beginning of a burst, it is recommended that searchTime is set to a value approximately equal to 2 x burstLength, where burstLength is the duration of a burst in seconds including the duration of the idle part. If it is known that the time instant specified by the start argument is a little before the beginning of a burst, then searchTime can be set to burstLength.

After a burst is detected, synchronization is performed based on the value of the sync argument. The burst is then demodulated. Finally, the burst is analyzed to get the EVM measurement results.

If averageType is set to *Off* , only one burst is detected, demodulated, and analyzed.

If averageType is set to *RMS (Video)* , after the first burst is analyzed the signal segment corresponding to it is discarded and new signal samples are acquired to fill in the signal buffer of length searchTime. When the buffer is full again a new burst search is performed and when a burst is detected it is demodulated and analyzed. These steps repeat until burstsToAverage bursts are processed.

If for any reason a burst is mis-detected, the results from its analysis are discarded. The EVM results obtained from all the successfully detected, demodulated, and analyzed bursts are averaged to give the final result.

The mirrorSpectrum argument accepts the following strings: "NO" and "YES". This argument can be used to mirror (conjugate) the input signal before any other processing is done. Mirroring the input signal is necessary if the configuration of the mixers in your system has resulted in a mirrored signal compared to the one at the input of the up-

converter. The demodulation process recovers a lot of the information about the burst from the burst preamble and SIGNAL symbol. If the input signal is mirrored, then some of this information may not be recovered correctly and the demodulation will most likely fail.

The start argument sets the starting point for acquiring the signal to be processed. By default, the starting point is the beginning of the input signal (voltage argument). However, if for any reason an initial part of the input signal needs to be omitted this can be done by setting the start argument appropriately.

The averageType argument accepts the following strings: "Off" and "RMS (Video)". This argument can be used to turn on/off video averaging. If set to "Off" the EVM result returned is from the processing of only one burst. Otherwise, multiple bursts are processed and the results are averaged.

The burstsToAverage argument set the number of bursts whose results will be averaged if averageType is set to "RMS (Video)". If averageType is set to "Off" this argument is ignored.

The subcarrierModulation argument accepts the following strings: "Auto Detect", "BPSK", "QPSK", "QAM 16", and "QAM 64". This argument sets the data subcarrier modulation format. If subcarrierModulation is set to " *Auto Detect"* , the algorithm will use the information detected within the OFDM burst (SIGNAL symbol - RATE data field) to automatically determine the data subcarrier modulation format. Otherwise, the format determined from the OFDM burst will be ignored and the format specified by the subcarrierModulation argument will be used in the demodulation for all data subcarriers. This argument has no effect on the demodulation of the pilot subcarriers and the SIGNAL symbol, whose format is always BPSK.

The guardInterval argument sets the guard interval (also called cyclic extension) length for the OFDM symbols. The value is expressed as a fraction of the FFT time period and so its valid range is [0, 1]. The value must match the guard interval length actually used in the generation of the input signal in order for the demodulation to work properly.

The searchTime argument sets the duration of the signal segment that is acquired and searched in order to detect a complete OFDM burst. Recommendations on how to set this argument are given in the brief description of the algorithm used by this expression earlier in these Notes/Equations section.

The resultLengthType argument accepts the following strings: "Auto Select" and "Manual Override". The resultLengthType and resultLength arguments control how much data is acquired and demodulated.

- When resultLengthType is set to " *Auto Select",* the measurement result length is automatically determined from the information in the decoded SIGNAL symbol (LENGTH data field). In this case, the argument resultLength defines a maximum result length for the burst in symbol times; that is, if the measurement result length that is automatically detected is bigger than resultLength it will be truncated to resultLength.
- When resultLengthType is set to " *Manual Override* ", the measurement result length is set to resultLength regardless of what is detected from the SIGNAL symbol of the burst. The value specified in resultLength includes the SIGNAL symbol but does not include any part of the burst preamble.

Measurement Result Length Setting table shown below, summarizes the differences between how *"Auto Select"* and " *Manual Override* " modes determine the measurement result length. The table lists the measurement result lengths actually used for "*Auto Select*" and " *Manual Override"* modes for three different values of the resultLength argument (30, 26 and 20 symbols). It is assumed that the input burst is 26 symbols long.

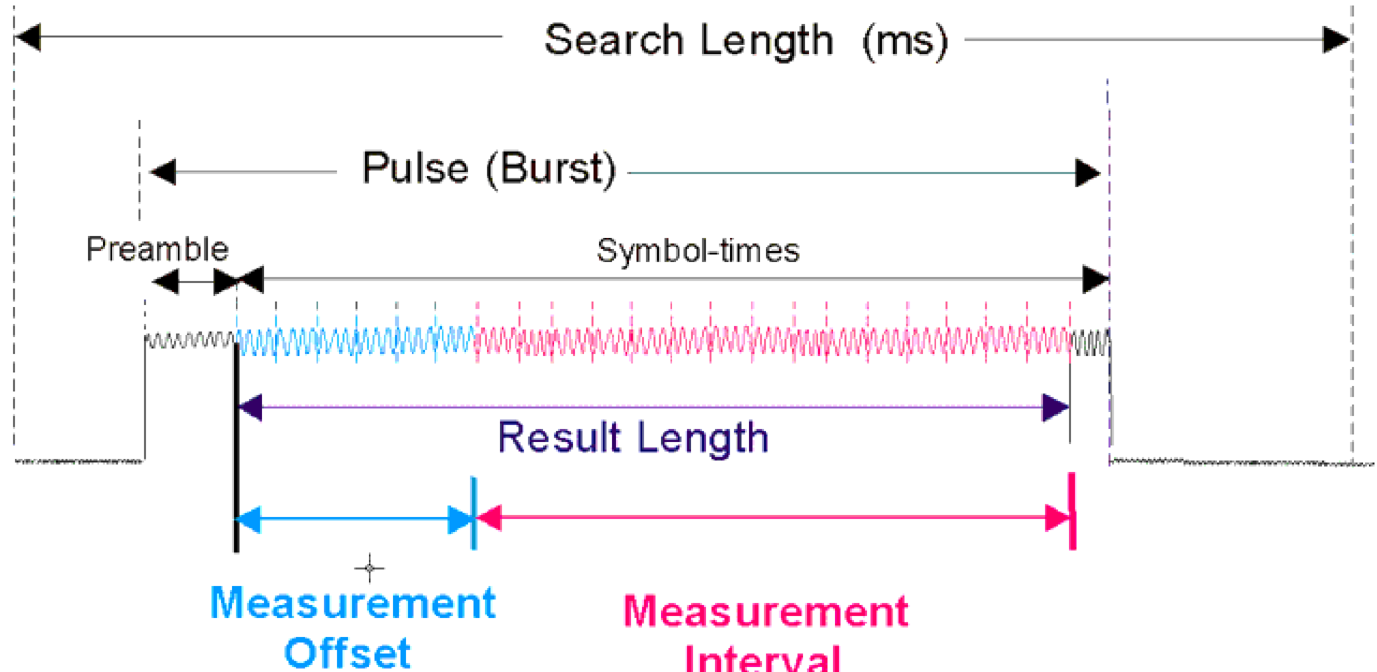
Measurement Result Length Setting

resultLengthType	resultLength	Measurement Result Length Actually Used
Auto Select	20	20
Auto Select	26	26
Auto Select	30	26
Manual Override	20	20
Manual Override	26	26
Manual Override	30	30

Note that when resultLengthType is set to " *Manual Override"* and resultLength=30 (greater than the actual burst size) the algorithm will demodulate the full 30 symbols even though this is 4 symbols beyond the burst width.

The measurementOffset and measurementInterval arguments can be used to isolate a specific segment of the burst for analysis. Both are expressed in number of OFDM symbols. The offset starts counting from the SIGNAL symbol. An offset of 0 will include the SIGNAL symbol in the EVM analysis, an offset of 1 will exclude the SIGNAL symbol, and an offset of 5 will exclude the SIGNAL symbol as well as the first 4 DATA symbols.

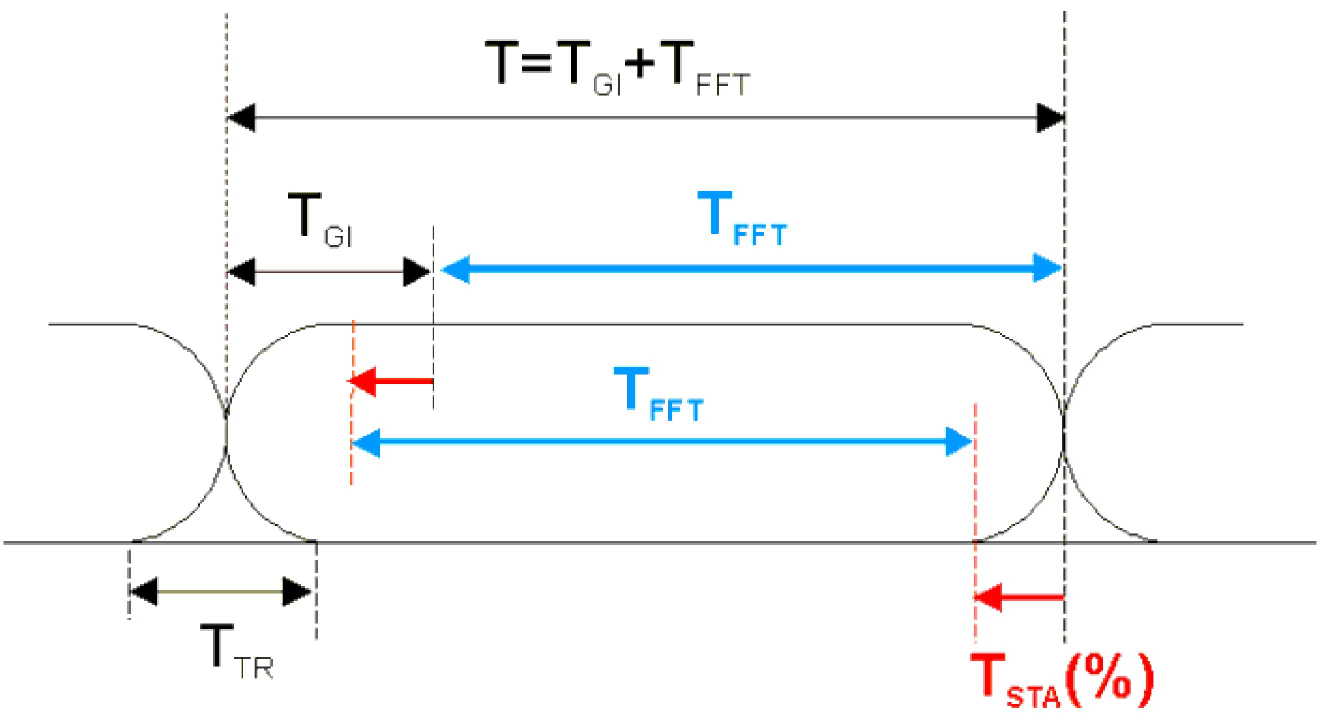
Interrelationship between searchTime, resultLength, measurementInterval, and measurementOffsetnshown below illustrates the interrelationship between searchTime, resultLength, measurementInterval, and measurementOffset.



Interrelationship between searchTime, resultLength, measurementInterval, and measurementOffset.

The subcarrierSpacing argument sets the frequency spacing between the subcarriers of the OFDM signal. The value must match the subcarrier spacing actually used in the generation of the input signal in order for the demodulation to work properly.

The symbolTimingAdjust argument sets the timing adjustment done on the OFDM symbols before performing the FFT. The value is expressed as a percent of the FFT time period. Its valid range is [-100*guardInterval, 0]. Normally, when demodulating an OFDM symbol, the guard interval is skipped and an FFT is performed on the last portion of the symbol. However, this means that the FFT will include the transition region between this symbol and the following symbol. To avoid this, it is generally beneficial to back away from the end of the symbol and use part of the guard interval. The symbolTimingAdjust argument controls how far the FFT part of the symbol is adjusted away from the end of the symbol. Note that the value of this argument is negative because the FFT start time is moved back by the amount specified by it. The definition of symbolTimingAdjust shown below, explains this concept graphically. When setting this argument, care should be taken to not back away from the end of the symbol too much because this may make the FFT include corrupt data from the transition region at the beginning of the symbol.



T = Symbol Time

T_{GI} = Guard Interval

T_{FFT} = FFT/IFFT Time Period

T_{TR} = Symbol Transition Time

T_{STA} = Symbol Timing Adjust (%)

Definition of symbolTimingAdjust.

The sync argument accepts the following strings: "Short Training Seq", "Channel Estimation Seq". This argument determines which preamble sequence will be used for synchronization.

The output argument accepts the following strings (see Available Measurement Results for evm_wlan_ofdm()): "EVMrms_percent", "EVM_dB", "PilotEVM_dB", "CPErms_percent", "IQ_Offset_dB", "SyncCorrelation". This argument selects which EVM analysis result will be returned.

Relationship Between WLAN_802_11a Source Parameters and evm_wlan_ofdm() Expression Arguments summarizes how some of the arguments of the evm_wlan_ofdm() expression should be set based on the parameter values of the WLAN_802_11a source.

Relationship Between WLAN_802_11a Source Parameters and evm_wlan_ofdm() Expression Arguments

WLAN_802_11a	evm_wlan_ofdm()	Comments
DataRate (default is 54 Mbps)	searchTime (default is 80 μsec)	The recommended searchTime is: 2 BurstLength. BurstLength= {[5 + (1 + OFDMSymbolsPerBurst)*(1 + GuardInterval)]*64} / Bandwidth + IdleInterval, where: OFDMSymbolsPerBurst = ceil[(22 + 8 *DataLength)* 250000 / DataRate]
Bandwidth (default is 20 MHz)		
DataLength (default is 100)		
IdleInterval (default is 4 μsec) †		
GuardInterval (default is 0.25)		
MirrorSpectrum (default is NO)	mirrorSpectrum (default is NO)	If DUT introduces spectrum mirroring, then mirrorSpectrum must be set to "NO" ("YES") when MirrorSpectrum is set to "YES" ("NO"); otherwise mirrorSpectrum must be set to the same value as MirrorSpectrum.
Bandwidth (default is 20 MHz)	subcarrierSpacing (default is 312.5 kHz)	subcarrierSpacing must be set to Bandwidth/64.
GuardInterval (default is 0.25)	guardInterval (default is 0.25)	guardInterval must be set to the same value as GuardInterval.
† The source IdleInterval must be >= 2 sec because the EVM measurement provided by the evm_wlan_ofdm() needs enough Idle Interval to detect the burst start time.		

fs()

Performs a time-to-frequency transform

Syntax

y = fs(x, fstart, fstop, numfreqs, dim, windowType, windowConst, tstart, tstop, interpOrder, transformMethod)

Arguments

Name	Description	Default	Range	Type	Required
x	Time-domain data to be transformed	None	(-∞:∞)	Real	Yes
fstart	starting frequency	0 †	[0:∞)	Real	No
fstop	stoping frequency	1/(2*newdeltat) †	[0:∞)	Real	No
numfreqs	number of frequencies	(fstop-fstart)*(tstop-tstart)+1	[1:∞)	Integer	No
dim	dimension to be transformed (not used currently)	highest dimension	[1:∞)	Integer	No
windowType	type of window to be applied to the data	0	[0:9] ††	Integer, string	No
windowConst	window constant †† †	0	[0:∞)	Integer, Real	No
tstart	start time †† ††	first time point in given data	[0:∞)	Integer, Real	No
tstop	stop time †† ††	last time point in given data	[0:∞)	Integer, Real	No
interpOrder	Interpolation Order	1	[1:3] †† †† †	Integer	No
transformMethod	Transformation method	1	[1:3] †† †† ††	Integer	No

† If data is real, fstart = 0, fstop=1/(2*newdeltat). If data is complex, fstart = -1/(time[startIndex+2]-time[stopIndex]) and fstop=1/(time[startIndex+2]-time[stopIndex]). Where newdeltat is the new uniform timestep of the resampled data, and startIndex and stopIndex are the index of tstart and tstop.

†† The window types and their default constants are:
0 = None
1 = Hamming 0.54
2 = Hanning 0.50
3 = Gaussian 0.75
4 = Kaiser 7.865
5 = 8510 6.0 (This is equivalent to the time-to-frequency transformation with normal gate shape setting in the 8510 series network analyzer.)
6 = Blackman
7 = Blackman-Harris
8 = 8510-Minimum 0
9 = 8510-Maximum 13

†† † windowConst is not used if windowType is 8510

†† †† If tstart or tstop lies between two time points in the data, then the values are interpolated for these values.

†† †† † If the tranorder variable is not present, or if the user wishes to override the interpolation scheme, then interpOrder may be set to a nonzero value:
1 = use only linear interpolation
2 = use quadratic interpolation
3 = use cubic polynomial interpolation

†† †† †† The time-to-frequency transform can be changed by using transformMethod:

1 = Chirp-z transform
2 = Discrete Fourier integral evaluated at each frequency
3 = Fast Fourier transform

Examples

```
The following example equations assume that a transient simulation was
performed from 0 to 5 ns with 176 timesteps, on a 1-GHz-plus-harmonics signal
called vOut:
y=fs(vOut) returns the spectrum (0, 0.2GHz, ... , 17.6GHz), evaluated from 0 to
5 ns.
y=fs(vOut, 0, 10GHz) returns the spectrum (0, 0.2GHz, ... , 10.0GHz), evaluated
from 0 to 5 ns.
y=fs(vOut, 0, 10GHz, 11) returns the spectrum (0, 1.0GHz, ... , 10.0GHz),
evaluated from 0 to 5 ns.
y=fs(vOut, , , , , , , 3ns, 5ns) returns the spectrum (0, 0.5GHz, ... ,
32.0GHz), evaluated from 3 to 5 ns.
y=fs(vOut, 0, 10GHz, 21, , , , 3ns, 5ns) returns the spectrum (0, 0.5GHz, ... ,
10.0GHz), evaluated from 3 to 5 ns.
y=fs(vOut, 0, 10GHz, 11, , "Blackman") returns the spectrum (0, 1.0GHz, ... ,
10.0GHz), evaluated from 0 to 5 ns after a Blackman window is applied.
```

Defined in

Built in

See Also

fft() (expmeas), *fspot()* (expmeas)

Notes/Equations

The dim argument is not used and should be left empty in the expression. Entering a value will have no impact on the results.

fs(x) returns the frequency spectrum of the vector x by using a chirp-z transform. The values returned are peak, complex values. The data to be transformed is typically from a transient, signal processing, or envelope analysis.

Transient simulation uses a variable timestep and variable order algorithm. The user sets an upper limit on the allowed timestep, but the simulator will control the timestep so that the local truncation error of the integration is also controlled. The non-uniformly sampled data are uniformly resampled for fs.

If the Gear integration algorithm is used, the order can also change during simulation. fs can use this information when resampling the data. This variable order integration depends on the presence of a special dependent variable, tranorder, which is output by the transient simulator. When the order varies, the Fourier integration will adjust the order of the polynomial it uses to interpolate the data between timepoints.

Only polynomials of degree one to three are supported. The polynomial is fit from the timepoint in question backwards over the last n points. This is because time-domain data are obtained by integrating forward from zero; previous data are used to determine future data, but future data can never be used to modify past data.

The data are uniformly resampled, with the number of points being determined by increasing the original number of points to the next highest power of two.

The data to be transformed default to all of the data. The user may specify tstart and tstop
to transform a subset of the data.

The starting frequency defaults to 0 and the stopping frequency defaults to 1/(2*newdeltat), where newdeltat is the new uniform timestep of the resampled data. The number of frequencies defaults to (fstop-fstart)*(tstop-tstart)+1. The user may change these by using fstart, fstop, and numfreqs. Note that numfreqs specifies the number of frequencies, not the number of increments. Thus, to get frequencies at (0, 1, 2, 3, 4, 5), numfreqs should be set to 6, not 5.

When the data to be operated on is of the baseband type, such as VO[0] from a Circuit Envelope analysis, where VO is an output node voltage and [0] is index for DC, then in order to obtain a single sided spectrum, only the real part of VO[0] should be used as the argument. i.e., x=fs(real(VO[0],...). This is necessary because the fs() function has no way of knowing the data VO[0] is baseband. Even though VO[0] contains an imaginary part of all zeroes, it is still represented by a complex data type. When the first argument of fs() is complex, the result will be a double-sided spectrum by default.

An alternative method of obtaining a single-sided spectrum from the above baseband data is to specify the frequencies ranges in the spectrum, using the fstart, fstop, and numfreqs parameters of the fs() function.

For example, y=fs(VO[0], 0, 25e3, 251). This will yield a spectrum from 0 to 25 kHz with 26 frequencies and 1 kHz spacing.

This does not apply to data from Transient analysis or Ptolemy simulation because voltage data from Transient and baseband data from Ptolemy are real.

For Envelope analysis, the transform is centered around the fundamental tone. For Signal Processing analysis, it is centered around the characterization frequency.

Differences Between the fs() and fft() Functions

For a periodic signal from t=0 to t=per, fs() requires the full data [0,per], where [] means 0 <= t <= per. The fft is defined as needing data [0,per), meaning 0 <= t < per. The last point at per should be the same as value at zero. fs() requires this point, fft() does not. That is why fs() really requires 2^{n+1} points and fft() requires 2^n . So for using the FFT option in the fs() function, there should be 2^{n+1} points. However, for the default Chirp Z-transform option, any number of points will work.
Conditions under which fs() may not work properly:

1. When deltaF equals 0. Where deltaF = (Fstop - Fstart)/(numFreqs).
2. When 1.0/deltaF > Tstop - Tstart. In this case there are not enough time data for requested frequency resolution.
3. When 2*numFreqs > numDataPts. In this case there are not enough data points for frequencies.

Mod_Data_from_1tone_swpUNI()

Returns an amplifier's adjacent and alternate channel power ratios, main channel power, and error vector magnitude

Syntax

Mod_Data_from_1tone_swpUNI(algorithm, allowextrap, charVoltage, inputSig, sourceZ, loadZ, mainCh, mainChForPout, lowerAdjCh, upperAdjCh, lowerAltCh, upperAltCh, winType, winConst)

Arguments

Name	Description	Default	Range	Type	Required
algorithm	Specifies the algorithm to be used to model the vout-versus-vin data from the HB sweep. Use "CF" for Curve Fit or "LI" for Linear Interpolation.	None	"LI" or "CF"	String	Yes
allowextrap	Allow or disallow extrapolation when applying the scaled, modulated input signal to the vout-versus-vin model.	1	0, "No", "NO", "no", 1, "Yes", "YES", "yes"	String or Integer	No
charVoltage	This is the characterization voltage (the fundamental output voltage from the harmonic balance sweep.) Example: Vload_fund, where Vload_fund=Vload[1].	None	(-∞:∞)	Complex	Yes
inputSig	This is the input modulated signal (the envelope.) This signal should be a function of time, only.	None	(-∞:∞)	Complex	Yes
sourceZ	This is the source impedance. This can be a swept parameter.	None	(0:∞)	Complex	Yes
loadZ	This is the load impedance. This can be a swept parameter.	None	(0:∞)	Complex	Yes
mainCh	These are the main channel frequency limits, as an offset from the carrier frequency. Example: {(-3.84 MHz/2),(3.84 MHz/2)}	None	(-∞:∞)	1X2 matrix, Real or Integer	Yes
mainChForPout	These are the frequency limits used for computing the modulated output power. Normally these would be the same as the mainCh argument, but this allows you to specify a different bandwidth for computing the modulated output power.	None	(-∞:∞)	1X2 matrix, Real or Integer	Yes
lowerAdjCh	These are the lower adjacent channel frequency limits as an offset from the carrier frequency. Example: MainLimits - (5 MHz)	None	(-∞:∞)	1X2 matrix, Real or Integer	Yes
upperAdjCh	These are the upper adjacent channel frequency limits as an offset from the carrier frequency. Example: MainLimits + (5 MHz)	None	(-∞:∞)	1X2 matrix, Real or Integer	Yes
lowerAltCh	These are the lower alternate channel frequency limits as an offset from the carrier frequency. Example: MainLimits - (10 MHz)	None	(-∞:∞)	1X2 matrix, Real or Integer	Yes
upperAltCh	These are the upper alternate channel frequency limits as an offset from the carrier frequency. Example: MainLimits + (10 MHz)	None	(-∞:∞)	1X2 matrix, Real or Integer	Yes
winType	window type	"Kaiser"	†	string	No
winConst	window constant that affects the shape of the applied window.	depends on winType used	[0:∞)	Real	No

† winType can be: "none", "hamming", "hanning", "gaussian", "kaiser", "8510", "blackman","blackman-harris"

Examples

```
Vload_fundHB=Vload[1]
Vin_fund=_3GPPFDD_UE_Tx_12_2_SigGen..Vsource
Zsource=50
Zload=50
MainLimits={-3.84 MHz/2,3.84 MHz/2}
MainLimitsForPout=MainLimits
LoChLimits=MainLimits-(5 MHz)
UpChLimits=MainLimits+(5 MHz)
LoChLimitsAlt=MainLimits-(10 MHz)
UpChLimitsAlt=MainLimits+(10 MHz)
Data_DDS=Mod_Data_from_1tone_swpUNI("LI", "Yes", Vload_fundHB, Vin_fund,
Zsource, Zload, MainLimits, MainLimitsForPout, LoChLimits, UpChLimits,
LoChLimitsAlt, UpChLimitsAlt, "Kaiser",)
ACPR_dBc=Data_DDS(0)
Pout_dBm=Data_DDS(2)
ACPR_vs_Pout=vs(ACPR_dBc,Pout_dBm)
AltCPR_dBc=Data_DDS(1)
AltCPR_vs_Pout=vs(AltCPR_dBc,Pout_dBm)
EVM_percent=Data_DDS(3)
EVM_vs_Pout=vs(EVM_percent,Pout_dBm)
```

See Also

ACPR_ChPwr_or_EVM_from_1toneSwp() (expmeas)

Notes/Equations

This function returns the adjacent and alternate channel power ratios, main channel power, and EVM. Because it returns multiple results, it cannot be used in a measurement expression on the schematic. It can only be used in the data display.
The advantage of using this function instead of ACPR_ChPwr_or_EVM_from_1tone_swp() is that it is more efficient for obtaining all these results. When using ACPR_ChPwr_or_EVM_from_1tone_swp(), you have to call it once to get the adjacent channel power ratios, once again to get the alternate channel power ratios, once again to get the main channel power, and once again to get the EVM. The big disadvantage of using the Mod_Data_from_1tone_swpUNI() is that this function will get executed each time you open a data display that contains it. While it is slower to use the ACPR_ChPwr_or_EVM_from_1tone_swp() function on the schematic, the advantage is that the results are written into the dataset, and data displays that show these results open instantly.

For EVM, the single-tone method is not specification-compliant. It just measures the “raw” EVM, computed at each time point. The EVM is computed after correcting for the average phase difference and RMS amplitude difference between the output and input modulated signals. If the modulated signal at the output of the amplifier has only a constant phase shift and a constant gain (meaning that neither vary with the amplitude of the input modulated signal), then the EVM will be zero. With this method, the EVM is computed at each time point, not at just the symbol times. There is no demodulation or decoding of the signal, so you can’t calculate the EVM of each sub-carrier, say for an LTE signal.

For ACPR, the single-tone method does not include any receive-side filtering. It just generates the spectrum at the output of the amplifier, integrates the power in the main, adjacent, and alternate channels, then computes the ratios. The single-tone method of

computing EVM (and ACPR) will tend to become less accurate as the bandwidth of the signal gets larger. This is because this method assumes the response of the amplifier is constant across the modulation bandwidth (we're modeling the nonlinearity by injecting a single tone at the carrier frequency, after all.)

peak_pwr()

Returns the peak power of the input voltage data

Syntax

peakP = peak_pwr(voltage, refR, percent, unit)

Arguments

Name	Description	Default	Range	Type	Required
voltage	baseband or complex envelope voltage signal	None	(-∞:∞)	Integer, Real, Complex	Yes
refR	reference resistance in Ohms	50.0	(0:∞)	Real	No
percent	percentage of time the returned power value is exceeded	0.0	[0:100]	Real	No
unit	power unit to be used	"W"	"W","dBm","dBW"	string	No

Examples

```
peakP = peak_pwr(Vout[1], 100, , "dBW")
where Vout is a named node in a Circuit Envelope simulation, will return the
peak power (in dBW) at the fundamental frequency using 100 Ohms as reference
resistance.
peakP = peak_pwr(T1, , 5)
where T1 is the name of a TimedSink component (in a DSP schematic), will return
the power level (in W) that is exceeded 5% of the time for the voltage signal
recorded in the TimedSink using 50 Ohms as reference resistance. If the signal
recorded by the TimedSink is complex envelope Gaussian noise with a standard
deviation of 30 mV for each of the I and Q envelopes, then peakP will be very
close to 5.39e-5 W ( -ln(0.05) / 0.032 / 50 ).
```

Defined in

$HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael

See Also

peak_to_avg_pwr() (expmeas), *power_ccdf()* (expmeas), *pwr_vs_t()* (expmeas), *total_pwr()* (expmeas)

Notes/Equations

Used in Circuit Envelope and Signal Processing simulations.
This expression can be used with input data (voltage) of any dimensions when the percent

argument is 0 and up to three dimensions when the percent argument is greater than 0. It can handle both baseband as well as complex envelope data.

When the percent argument is set to 0, the returned value is the maximum instantaneous power of the input voltage signal. When the percent argument is set to a value greater than 0, the returned value is the power level that is exceeded for percent amount of time. This argument is useful since some wireless standards specify the peak power not as the absolute maximum instantaneous power but as the power level that is exceeded for some percentage of time. For example, the 3GPP standard defines the maximum power as the power level that is exceeded for 1% of the time.

peak_to_avg_pwr()

Returns the peak to average power ratio of the input voltage data

Syntax

peak_avg_ratio = peak_to_avg_pwr(voltage, percent, unit)

Arguments

Name	Description	Default	Range	Type	Required
voltage	baseband or complex envelope voltage signal	None	(-∞:∞)	Integer, Real, Complex	Yes
percent	percentage of time the returned power value is exceeded	0.0	[0:100]	Real	No
unit	unit to be used	"dB"	"dB","ratio"	string	No

Examples

```
peak_avg_ratio = peak_to_avg_pwr(Vout[1], , "ratio")
where Vout is a named node in a Circuit Envelope simulation, will return the
peak to average power ratio (as a ratio) at the fundamental frequency.
peak_avg_ratio = peak_to_avg_pwr(T1, 5, "ratio")
where T1 is the name of a TimedSink component (in a DSP schematic), will return
the ratio of the power level that is exceeded 5% of the time to the average
power level for the voltage signal recorded in the TimedSink. If the signal
recorded by the TimedSink is complex envelope Gaussian noise with the same
standard deviation for both the I and Q envelopes, then peak_avg_ratio will be
very close to 2.99 ( -ln(0.05) ).
peak_avg_ratio = peak_to_avg_pwr(T1)
where T1 is the name of a TimedSink component (in a DSP schematic), will return
the peak to average power ratio (in dB) for the voltage signal recorded in the
TimedSink. If the signal recorded by the TimedSink is an ideal QPSK signal
(filtered using a raised cosine filter of ExcessBW 0.5), then peak_avg_ratio
will be very close to 3.95 dB. If the signal recorded by the TimedSink is an
ideal /4-DQPSK signal (filtered using a raised cosine filter of ExcessBW 0.5),
then peak_avg_ratio will be very close to 3.3 dB.
```

Defined in

$HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael

See Also

peak_pwr() (expmeas), *power_ccdf()* (expmeas), *pwr_vs_t()* (expmeas), *total_pwr()* (expmeas)

Notes/Equations

Used in Circuit Envelope and Signal Processing simulations.
This expression can be used with input data (voltage) of any dimensions when the percent argument is 0 and up to three dimensions when the percent argument is greater than 0. The expression can accommodate both baseband as well as complex envelope data.

The peak to average ratio is computed by calling the peak_pwr() and total_pwr() expressions and then taking the ratio of the two values returned by these expressions. The percent argument is used for the calculation of the peak power value. For the use and meaning of the percent argument see the *peak_pwr()* (expmeas) Notes/Equations.

Since a ratio of power values is calculated a reference resistance is not needed for this measurement.

power_ccdf()

Returns the power CCDF (Complementary Cumulative Distribution Function) curve for the input voltage/current data.

Syntax

pCCDF = power_ccdf(data, numBins)

Arguments

Name	Description	Default	Range	Type	Required
data	baseband or complex envelope voltage/current signal	None	(-∞:∞)	Integer, Real, Complex	Yes
numBins	number of points in the returned power CCDF curve	log2(numDataPoints)	[1:∞)	Integer	No

Examples

```
pCCDF = power_ccdf(Vout[1])
where Vout is a named node in a Circuit Envelope simulation, will return the
power CCDF curve for the voltage at the fundamental frequency. The returned
power CCDF curve will have the default number of points.
pCCDF = power_ccdf(T1, 10)
where T1 is the name of a TimedSink component (in a DSP schematic), will return
the power CCDF curve for the voltage signal recorded in the TimedSink. The
returned curve will have 10 points.
```

Defined in

$HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael

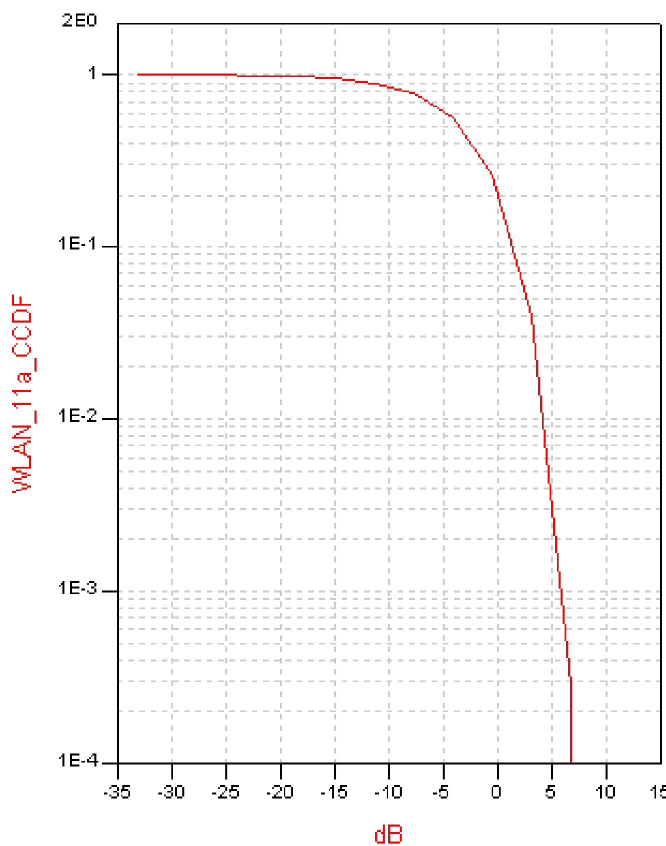
See Also

peak_pwr() (expmeas), *peak_to_avg_pwr()* (expmeas), *power_ccdf_ref()* (expmeas), *pwr_vs_t()* (expmeas), *total_pwr()* (expmeas)

Notes/Equations

Used in Circuit Envelope and Signal Processing simulations.
This expression can be used with input data of up to two dimensions. It can accommodate both baseband as well as complex envelope data.

The power CCDF (typically called just CCDF) measurement is a very common measurement performed on 2G and 3G wireless signals. The CCDF curve shows the probability that the instantaneous signal power will be higher than the average signal power by a certain amount of dB. The independent axis of the CCDF curve shows power levels in dB with respect to the signal average power level (0 dB corresponds to the signal average power level). The dependent axis of the CCDF curve shows the probability that the instantaneous signal power will exceed the corresponding power level on the independent axis. CCDF Curve for WLAN 802.11a 54 Mbps Signal shows the CCDF curve for a WLAN 802.11a 54 Mbps signal.



CCDF Curve for WLAN 802.11a 54 Mbps Signal

In CCDF Curve for WLAN 802.11a 54 Mbps Signal, you can see that the instantaneous signal power exceeds the average signal power (0 dB) for 20% of the time. You can also see that the instantaneous signal power exceeds the average signal power by 5 dB for only 0.3% of the time.

power_ccdf_ref()

Returns the power CCDF (Complementary Cumulative Distribution Function) curve for white gaussian noise.

Syntax

ccdfRef = power_ccdf_ref(indepPowerRatioValues)

Arguments

Name	Description	Default	Range	Type	Required
indepPowerRatioValues	power levels (in dB with respect to the average power level of a white gaussian noise signal) at which the power CCDF for white gaussian noise will be calculated	None	(-∞:∞)	Integer, Real	Yes

Examples

```
pCCDF = power_ccdf(T1)
ccdfRef = power_ccdf_ref( indep( pCCDF) )
where T1 is the name of a TimedSink component (in a DSP schematic), will return
the power CCDF curve for white gaussian noise evaluated at the same power
levels as the pCCDF curve.
ccdfRef = power_ccdf_ref( [-10::2::8] )
will return the power CCDF curve for white gaussian noise evaluated at the
power levels -10, -8, -6, -4, -2, 0, 2, 4, 6, 8 (in dB with respect to the
average power level of a white gaussian noise signal).
```

Defined in

$HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael

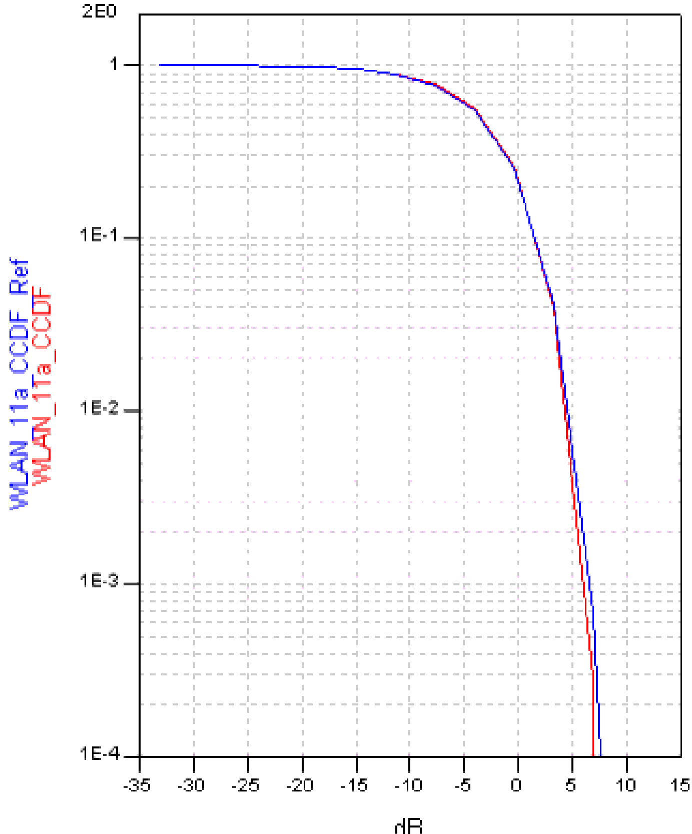
See Also

power_ccdf() (expmeas)

Notes/Equations

This expression can be used with input data of up to two dimensions.
The input data must be sorted in increasing or decreasing order. In addition, it is recommended that the input vector argument has uniformly spaced values. The vector "indep(X)", where X is the value returned by the power_ccdf() expression, is guaranteed to have uniformly spaced values sorted in increasing order. This would be the most typical value for the input argument of power_ccdf_ref().

A typical power CCDF measurement (see Notes/Equations for *power_ccdf()* (expmeas)) provides a reference CCDF curve along with the measured CCDF curve. This reference curve is typically the power CCDF curve for a white gaussian noise signal. This expression can generate this reference curve. CCDF Curve for WLAN 802.11a Signal with Reference Curve shows the CCDF curve for a WLAN 802.11a signal along with the reference curve.



CCDF Curve for WLAN 802.11a Signal with Reference Curve

pwr_vs_t()

Returns the power vs. time waveform for the input voltage data

Syntax

p_vs_t = pwr_vs_t(voltage, refR, unit)

Arguments

Name	Description	Default	Range	Type	Required
voltage	baseband or complex envelope voltage signal	None	(-∞:∞)	Integer, Real, Complex	Yes
refR	reference resistance in Ohms	50.0	(0:∞)	Real	No
unit	power unit to be used	"W"	"W","dBm","dBW"	string	No

Examples

```
p_vs_t = pwr_vs_t(Vout[1], 100, "dBm")
where Vout is a named node in a Circuit Envelope simulation, will return the
power (in dBm) vs. time waveform for the voltage at the fundamental frequency
using 100 Ohms as reference resistance.
p_vs_t = pwr_vs_t(T1)
where T1 is the name of a TimedSink component (in a DSP schematic), will return
the power (in W) vs. time waveform for the voltage signal recorded in the
TimedSink using 50 Ohms as reference resistance.
```

Defined in

$HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael

See Also

peak_pwr() (expmeas), *peak_to_avg_pwr()* (expmeas), *power_ccdf()* (expmeas), *total_pwr()* (expmeas)

Notes/Equations

Used in Circuit Envelope and Signal Processing simulations.
This expression can be used with input data (voltage) of any dimensions. It can accommodate both baseband as well as complex envelope data.

relative_noise_bw()

Computes the relative noise bandwidth of the smoothing windows used by the fs() function

Syntax

y = relative_noise_bw(winType, winConst)

Arguments

Name	Description	Default	Range	Type	Required
winType	window type	None	†	string	No
winConst	window constant that affects the shape of the applied window.	0.75	[0:∞)	Real	No

† winType can be: "none", "hamming", "hanning", "gaussian", "kaiser", "8510", "blackman","blackman-harris"

Examples

```
winType = "Kaiser"
winConst = 8
relNoiseBW = relative_noise_bw("Kaiser", 8) = 1.666
Vfund=vOut[1]
VoltageSpectralDensity = fs(Vfund, , , , , winType, winConst)
PowerSpectralDensity = 0.5 * mag(VoltageSpectralDensity**2)/50/relNoiseBW
```

```
where vOut is the named connection at a 50-ohm load, and it is an output from a
Circuit Envelope simulation.
Note vOut is a named connection on the schematic. Assuming that a Circuit
Envelope simulation was run, vOut is output to the dataset as a two-dimensional
matrix. The first dimension is time, and there is a value for each time point
in the simulation. The second dimension is frequency, and there is a value for
each fundamental frequency, each harmonic, and each mixing term in the
analysis, as well as the baseband term.
vOut[1] is the equivalent of vOut[::, 1], and specifies all time points at the
lowest non-baseband frequency (the fundamental analysis frequency, unless a
multitone analysis has been run and there are mixing products). For former MDS
users, the notation "vOut[*, 2]" in MDS corresponds to the notation of
"vOut[1]".
```

Defined in

$HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael

See Also

acpr_vi() (expmeas), *acpr_vr()* (expmeas), *channel_power_vi()* (expmeas), *channel_power_vr()* (expmeas), *fs()* (expmeas)

Notes/Equations

Used in The following functions: acpr_vi, acpr_vr, channel_power_vi, channel_power_vr.

The relative noise bandwidth function is used to account for the fact that as windows are applied, the effective noise bandwidth increases with respect to the normal resolution bandwidth. The resolution bandwidth is determined by the time span and not by the displayed frequency resolution.

sample_delay_pi4dqpsk()

This function calculates the optimal sampling point within a symbol for a given pi4dqpsk waveform. "Optimal" is defined as the sampling point that provides the lowest bit error rate.

Syntax

y = sample_delay_pi4dqpsk(vIQ, symbolRate, path_delay, timeResolution)

Arguments

Name	Description	Default	Range	Type	Required
vIQ	complex envelope (I + j * Q) of a pi/4 DQPSK signal	None	(-∞:∞)	Complex	Yes
symbolRate	symbol rate of the pi/4 DQPSK signal	None	(0:∞)	Real	Yes
path_delay	time delay on the waveform before the sampling starts †	None	(0:∞)	Real	Yes
timeResolution	time step (typically one-tenth of a symbol time or less) used to search for the best sampling point in a given symbol period	None	(0:∞)	Real	Yes

† If the delay is 0, this parameter may be omitted. If it is non-zero, enter the delay value. This can be calculated using the function delay_path().

Examples

```
a = sample_delay_pi4dqpsk(vout[1], 25e3, 1.5e-6, 0.15e-6)
```

Defined in

Built in

See Also

ber_pi4dqpsk() (expmeas), *ber_qpsk()* (expmeas), *const_evm()* (expmeas)

Notes/Equations

This function can be used only with 1-dimensional data.

sample_delay_qpsk()

This function calculates the optimal sampling point within a symbol for a given QPSK waveform."Optimal" is defined as the sampling point that provides the lowest bit error rate.

Syntax

y = sample_delay_qpsk(vIQ, symbolRate, path_delay, timeResolution)

Arguments

Name	Description	Default	Range	Type	Required
vIQ	complex envelope (I + j * Q) of a QPSK signal	None	(-∞:∞)	Complex	Yes
symbolRate	symbol rate of the QPSK signal	None	(0:∞)	Real	Yes
path_delay	time delay on the waveform before the sampling starts †	None	(0:∞)	Real	Yes
timeResolution	time step (typically one-tenth of a symbol time or less) used to search for the best sampling point in a given symbol period	None	(0:∞)	Real	Yes

† If the delay is 0, this parameter may be omitted. If it is non-zero, enter the delay value. This can be calculated using the function delay_path().

Examples

```
a = sample_delay_qpsk(vout[1], 25e3, 1.5e-6, 0.15e-6)
```

Defined in

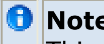
Built in

See Also

ber_pi4dqpsk() (expmeas), *ber_qpsk()* (expmeas), *const_evm()* (expmeas)

Notes/Equations

This function can be used only with 1-dimensional data.



Note
This function is only available in 32-bit simulator.

spectrum_analyzer()

Performs a spectrum analysis for the input voltage/current data

Syntax

spectrum = spectrum_analyzer(data, fCarrier, start, stop, window, resBW)

Arguments

Name	Description	Default	Range	Type	Required
data	baseband or complex envelope voltage/current signal	None	(-∞:∞)	Real, Complex	Yes
fCarrier	frequency around which the spectrum will be centered	0	[0:∞)	Real	No
start	start time for the spectrum analysis	first point of input data	[0:∞)	Integer, Real	No
stop	stop time for the spectrum analysis	last point of input data	[0:∞)	Integer, Real	No
window	type of window to be used	0	[0:7] †	Integer, string	No
resBW	resolution bandwidth	0	[0:∞)	Integer, Real	No

† See Notes for the window type.

Examples

```
spectrum = spectrum_analyzer(Vout[1])
where Vout is a named node in a Circuit Envelope simulation, will return the
```

```
voltage spectrum at the fundamental frequency. The spectrum will be centered
around 0 Hz. All the input data will be processed in one block resulting in the
highest resolution bandwidth possible. No windowing will be done.
spectrum = spectrum_analyzer(Vout[1], 3.5e9, , , "Kaiser 7.865", 30e3)
where Vout is a named node in a Circuit Envelope simulation, will return the
voltage spectrum at the fundamental frequency. The spectrum will be centered
around 3.5 GHz. The input signal will be broken down in smaller segments in
order to achieve 30 kHz of resolution bandwidth. All segments will be windowed
with a Kaiser 7.865 window. The spectra of all segments will be averaged.
spectrum = spectrum_analyzer(T1, , 1.0e-3, 2.0e-3, "Hanning 0.5")
where T1 is the name of a TimedSink component (in a DSP schematic), will return
the voltage spectrum for the segment between 1 msec and 2 msec of the signal
recorded in the TimedSink. Of course, the TimedSink component must have
recorded a signal that starts before 1 msec and ends after 2 msec. The spectrum
will be centered around 0 Hz. A Hanning 0.5 window will be used.
```

Defined in

Built in

See Also

fs() (expmeas)

Notes/Equations

Used in Circuit Envelope and Signal Processing simulations.
This expression can be used with input data of up to two dimensions. It can handle both baseband as well as complex envelope input data. Although non-uniformly sampled data is supported, it is recommended to use this expression with uniformly sampled data.

The spectrum_analyzer() expression returns the voltage or current spectrum of the input (voltage or current) signal. The returned values are the complex amplitude voltages or currents at the frequencies of the spectral tones. The returned values are not the powers at the frequencies of the spectral tones. However, one can calculate and display the power spectrum by using the *dbm()* (expmeas) expression (for voltage input data) or writing a simple equation (for current input data) in the data display window.

Note that, for baseband signals and for the frequency of 0 Hz, the dBm function returns a power value that is 3 dB less than the actual power. This is because the primary use of the dBm function is with RF signals, where the 0 Hz frequency corresponds to the carrier frequency and not really 0 Hz signal frequency. If the baseband signal has no significant power at DC, this 3 dB error is insignificant and can be ignored-otherwise, it must be considered.

The basis of the algorithm used by the spectrum_analyzer() expression is the *fs()* (expmeas) expression (the chirp-Z transform option of fs() is used). The input data can be processed as one block and the spectrum calculated over the entire input signal.

Alternatively, the input signal can be broken down in smaller blocks and the spectra of all blocks averaged.

The *fCarrier* argument sets the frequency around which the spectrum will be centered. This is only true for a complex envelope signal. For baseband signals, this argument is ignored. The spectrum span is:

- for complex envelope signals [fCarrier-0.5/TStep, fCarrier+0.5/TStep]
- for baseband signals [0, 0.5/TStep]
  where *TStep* is the simulation time step.

If it is not desirable to process all the input data, the *start* and *stop* arguments can be used to define a segment of the input data to be processed. This can be useful when trying to exclude parts of the signal where transients occur.
The *window* argument sets the window that will be used to window the input data. Windowing is often necessary in transform-based (chirp-Z, FFT) spectrum estimation. Without windowing, the estimated spectrum may suffer from spectral leakage that can cause misleading measurements or masking of weak signal spectral detail by spurious artifacts. If the input data is broken down in multiple blocks then the window is applied to each block.

The window argument accepts the following strings: "none", "Hamming 0.54", "Hanning 0.50", "Gaussian 0.75", "Kaiser 7.865", "8510 6.0", "Blackman", "Blackman-Harris". The equations defining these windows are given below:

- none:
  $$w(kT_s) = \begin{cases} 1.0 & 0 \le k \le N \\ 0.0 & otherwise \end{cases}$$
  where N is the window size
- Hamming 0.54:
  $$w(kT_s) = \begin{cases} 0.54 - 0.46\cos\left(\frac{2\pi k}{N}\right) & 0 \le k \le N \\ 0.0 & otherwise \end{cases}$$
  where N is the window size
- Hanning 0.5:
  $$w(kT_s) = \begin{cases} 0.5 - 0.5\cos\left(\frac{2\pi k}{N}\right) & 0 \le k \le N \\ 0.0 & otherwise \end{cases}$$
  where N is the window size
- Gaussian 0.75:
  $$w(kT_s) = \begin{cases} exp\left(-\left(0.75\pi\left(\frac{2k-N}{N}\right)\right)^2\right) & 0 \le k \le N \\ 0 & otherwise \end{cases}$$
  where N is the window size
- Kaiser 7.865:

$$w(kT_s) = \begin{cases} \dfrac{I_0\left(7.865\left[1-\left(\dfrac{k-\alpha}{\alpha}\right)^2\right]^{1/2}\right)}{I_0(7.865)} & 0 \le k \le N \\ 0.0 & otherwise \end{cases}$$

where N is the window size, α = N / 2, and $I_0(.)$ is the 0th order modified Bessel function of the first kind

- 8510 6.0 (Kaiser 6.0):

$$w(kT_s) = \begin{cases} \dfrac{I_0\left(6.0\left[1-\left(\dfrac{k-\alpha}{\alpha}\right)^2\right]^{1/2}\right)}{I_0(6.0)} & 0 \le k \le N \\ 0.0 & otherwise \end{cases}$$

where N is the window size, α = N / 2, and $I_0(.)$ is the 0th order modified Bessel function of the first kind

- Blackman:

$$w(kT_s) = \begin{cases} 0.42 - 0.5\cos\left(\dfrac{2\pi k}{N}\right) + 0.08\cos\left(\dfrac{4\pi k}{N}\right) & 0 \le k \le N \\ 0.0 & otherwise \end{cases}$$

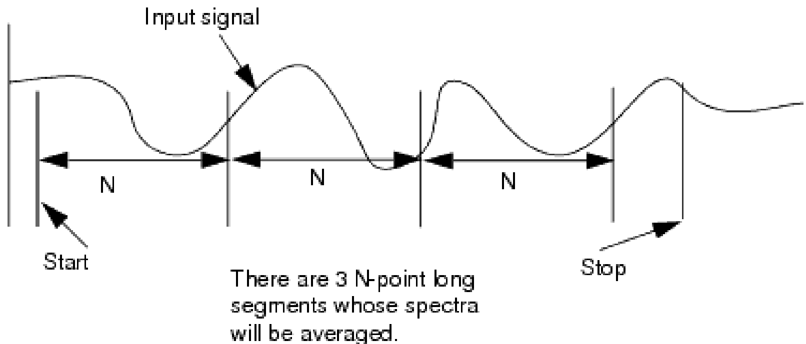
where N is the window size

- Blackman-Harris:

$$w(kT_s) = \begin{cases} 0.35875 - 0.48829\cos\left(\dfrac{2\pi k}{N}\right) + 0.14128\cos\left(\dfrac{4\pi k}{N}\right) - 0.01168\cos\left(\dfrac{6\pi k}{N}\right) & 0 \le k \le N \\ 0.0 & otherwise \end{cases}$$

where N is the window size

The *resBW* parameter can be used to set the spectrum measurement resolution bandwidth. If set to 0, it is ignored and the signal segment defined by start and stop is processed as one segment. In this mode of operation, the returned spectrum will have the highest possible resolution bandwidth (resolution bandwidth is inversely proportional to the input signal length). If resBW is set to a value greater than 0, then the input signal segment defined by start and stop is broken down in smaller subsegments of the appropriate length. The length of each segment is decided based on the value of resBW and the selected window. The spectrum of each subsegment is calculated and averaged with the spectra of the other subsegments. In this mode of operation, the resolution bandwidth achieved is resBW. Averaging multiple input signal segments. shows an example where the input signal is broken down in multiple segments. As can be seen in this figure, if an exact integer multiple of subsegments cannot fit in the segment defined by start and stop, then part of the signal may not be used.



Averaging multiple input signal segments.

In an analog swept spectrum analyzer, the resolution bandwidth is determined by the last in a series of analog IF filters. In contrast, the spectrum_analyzer() expression calculates the spectrum using DSP algorithms and so the resolution bandwidth is determined by the length of the input data segment the algorithm processes and the selected window.
ResBW = ENBW = NENBW / T
where ENBW is the window equivalent noise bandwidth
T is the length of the input data segment in seconds and
NENBW is the window normalized equivalent noise bandwidth The following table shows the NENBW values for the available windows).

Window	NENBW
none	1.0
Hamming 0.54	1.363
Hanning 0.5	1.5
Gaussian 0.75	1.883
Kaiser 7.865	1.653
8510 6.0	1.467
Blackman	1.727
Blackman-Harris	2.021

The equivalent noise bandwidth (ENBW) of a window is defined as the width of a rectangular filter that passes the same amount of white noise as the window. The normalized equivalent noise bandwidth (NENBW) is computed by multiplying ENBW with the time record length. For example, a Hanning window with a 0.5 second input data segment will result in an ENBW (as well as ResBW) of 3 Hz (1.5 / 0.5).

How to choose the right window

Every time a window is applied to a signal (Window = none effectively applies a rectangular window to the signal), leakage occurs, that is, power from one spectral component leaks into the adjacent ones. Leakage from strong spectral components can result in hiding/masking of nearby weaker spectral components. Even strong spectral components can be affected by leakage. For example, two strong spectral components close to each other can appear as one due to leakage.
Choosing the right window for a spectral measurement is very important. The choice of window depends on what is being measured and what the trade-offs between frequency resolution (ability to distinguish spectral components of comparable strength that are close to each other) and dynamic range (ability to measure signals with spectral components of widely varying strengths and distributed over a wide range) are.

As described above, windows can be characterized by their Normalized Equivalent Noise Bandwidth (NENBW). In general, for the same length of signal processed, the higher the NENBW of a window the higher its dynamic range (less leakage) and the poorer its frequency resolution.
Some general guidelines for choosing a window are given below:

- Do not use a window (Window = none) when analyzing transients.
- For periodic signals whose spectral components have comparable strengths and when the signal segment processed includes an exact integer multiple of periods, the best results are obtained if no window is used (Window = none, which is equivalent to using a rectangular window). Any start up transients should be excluded.
- For periodic signals whose spectral components have significantly different strengths and/or when the signal segment processed does not include an exact integer multiple of periods, the use of a window can improve the detection of the weaker spectral components. The higher the NENBW the more likely the weaker spectral components will be detected. However, this trades-off frequency resolution and so if the spectral components are very close to each other the weaker one might remain unresolved. To improve frequency resolution while still maintaining a good dynamic range use a window but process a longer signal segment.
- For aperiodic signals such as modulated signals (QPSK, QAM, GSM, EDGE, CDMA, OFDM) the use of a window is highly recommended. The window will attenuate the signal at both ends of the signal segment processed to zero. This makes the signal apear periodic and reduces leakage.

total_pwr()

Returns the total (average) power of the input voltage data

Syntax

totalP = total_pwr(voltage, refR, unit)

Arguments

Name	Description	Default	Range	Type	Required
voltage	baseband or complex envelope voltage signal	None	(-∞:∞)	Integer, Real, Complex	Yes
refR	reference resistance in Ohms	50.0	(0:∞)	Real	No
unit	power unit to be used	"W"	"W","dBm","dBW"	string	No

Examples

```
totalP = total_pwr(Vout[1], 100, "dBm")
where Vout is a named node in a Circuit Envelope simulation, will return the
total power (in dBm) at the fundamental frequency using 100 Ohms as reference
resistance.
totalP = total_pwr(T1)
where T1 is the name of a TimedSink component (in a DSP schematic), will return
the total power (in W) for the voltage signal recorded in the TimedSink using
50 Ohms as reference resistance. If the signal recorded by the TimedSink is
baseband Gaussian noise with a standard deviation of 30 mV, then totalP will be
very close to 1.8e-5 W ( 0.032 / 50 ).
```

Defined in

$HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael

See Also

peak_pwr() (expmeas), *peak_to_avg_pwr()* (expmeas), *power_ccdf()* (expmeas), *pwr_vs_t()* (expmeas)

Notes/Equations

Used in Circuit Envelope and Signal Processing simulations.
This expression can be used with input data (voltage) of any dimensions. It can accommodate both baseband as well as complex envelope data.

trajectory()

Generates the trajectory diagram from I and Q data, which are usually produced by a Circuit Envelope simulation.

Syntax

Traj = trajectory(i_data, q_data)

Arguments

Name	Description	Default	Range	Type	Required
i_data	in-phase component of data versus time of a single complex voltage spectral component (for example, the fundamental) †	None	(-∞:∞)	Complex	Yes
q_data	quadrature-phase component of data versus time of a single complex voltage spectral component (for example, the fundamental) †	None	(-∞:∞)	Complex	Yes

† This could be a baseband signal instead, but in either case it must be real valued versus time.

Examples

```
Rotation = -0.21
Vfund=vOut[1] *exp(j * Rotation)
Vimag = imag(Vfund)
Vreal = real(Vfund)
Traj = trajectory(Vreal, Vimag)
where Rotation is a user-selectable parameter that rotates the trajectory
diagram by that many radians and vOut is the named connection at a node.
Note vOut is a named connection on the schematic. Assuming that a Circuit
Envelope
simulation was run, vOut is output to the dataset as a two-dimensional matrix.
The first dimension is time, and there is a value for each time point in the
simulation. The second dimension is frequency, and there is a value for each
fundamental frequency, each harmonic, and each mixing term in the analysis, as
well as the baseband term.
vOut[1] is the equivalent of vOut[::, 1], and specifies all time points at the
```

```
lowest non-baseband frequency (the fundamental analysis frequency, unless a
multitone analysis has been run and there are mixing products). For former MDS
users, the notation "vOut[*, 2]" in MDS corresponds to the notation of
"vOut[1]".
```

Defined in

$HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael

See Also

constellation() (expmeas), *const_evm()* (expmeas)

Notes/Equations

Used in Trajectory diagram generation.
The I and Q data do not need to be baseband waveforms. For example, they could be the in-phase (real or I) and quadrature-phase (imaginary or Q) part of a modulated carrier. The user must supply the I and Q waveforms versus time.

Data Access Functions for Measurement Expressions

This section describes data access and data manipulation functions in detail. The functions are listed in alphabetical order.

You can use these functions to find information about a piece of data (e.g., independent values, size, type, attributes, etc.). You can also use some functions to generate data for plotting circles and contours.

> **Note**
> For information on how these functions are used, see the tutorial example *Using Expressions in the Data Display Window* (examples).

- *build subrange()* (expmeas)
- *chop()* (expmeas)
- *chr() Measurement* (expmeas)
- *circle()* (expmeas)
- *collapse()* (expmeas)
- *contour_ex()* (expmeas)
- *contour()* (expmeas)
- *contour polar()* (expmeas)
- *copy()* (expmeas)
- *create()* (expmeas)
- *dd_threshold()* (expmeas)
- *delete()* (expmeas)
- *expand()* (expmeas)
- *find()* (expmeas)
- *find index()* (expmeas)
- *generate()* (expmeas)
- *get attr()* (expmeas)
- *get indep values()* (expmeas)
- *indep()* (expmeas)
- *max index()* (expmeas)
- *min index()* (expmeas)
- *permute()* (expmeas)
- *plot vs()* (expmeas)
- *set attr()* (expmeas)
- *size()* (expmeas)
- *sort()* (expmeas)
- *sweep dim()* (expmeas)
- *sweep size()* (expmeas)
- *type()* (expmeas)
- *vs()* (expmeas)
- *what()* (expmeas)
- *write var()* (expmeas)

build_subrange()

Builds the subrange data according to the innermost independent range. Use with all swept data

Syntax

y = build_subrange(data, innermostIndepLow, innermostIndepHigh)

Arguments

Name	Description	Default	Range	Type	Required
data	analysis results or user input.	None	(-∞:∞)	Integer, Real, Complex	Yes
innermostIndepLow	lowest value of innermost independent	minimum value of the inner most independent variable	(-∞:∞)	Integer, Real, Complex	No
innermostIndepHigh	highesr value of innermost independent	maximum value of the inner most independent variable	(-∞:∞)	Integer, Real, Complex	No

Examples

```
Given S-parameter data swept as a function of frequency with a range of 100 MHz
to 500 MHz, find the values of S12 in the range of 200 MHz to 400 MHz.
subrange_S12 = build_subrange(S12, 200MHz, 400MHz)
```

Defined in

$HPEESOF_DIR/expressions/ael/statistical_fun.ael

chop()

Replace numbers in x with magnitude less than dx with 0

Syntax

y = chop(x, dx)

Arguments

Name	Description	Default	Range	Type	Required
x	numbers to replace	None	(-∞:∞)	Integer, real, complex	Yes
dx	value to compare to	1e-10	(-∞:∞)	Integer, real, complex	Yes

Examples

```
chop(1) returns 1
chop(1e-12) returns 0
chop(1+1e-12i) returns 1+0i
```

Defined in

$HPEESOF_DIR/expressions/ael/elementary_fun.ael

Notes/Equations

The chop() function acts independently on the real and complex components of x, comparing each to mag(dx). For example,

```
y = x if mag(x) >= mag(dx)
y = 0 if mag(x)
```

chr()

Returns the character representation of an integer

Syntax

y = chr(x)

Arguments

Name	Description	Default	Range	Type	Required
x	valid number representing a ASCII character	None	[0:127]	Integer	Yes

Examples

```
a = chr(64) returns @
a = chr(60) returns <
a = chr(117) returns u
```

Defined in

Built in

Note
The function name chr() is used for more than one type of expression. For comparison, see the AEL Function *chr() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

circle()

Used to draw a circle on a Data Display page. Accepts the arguments center, radius, and number of points. Can only be used on polar plots and Smith charts.

Syntax

a = circle(center, radius, numPts)

Arguments

Name	Description	Default	Range	Type	Required
center	center coordinate	None	(-∞:∞)	Integer, real, Complex	Yes
radius	radius	None	[0:∞)	Integer, real	Yes
numPts	number of points in the circle	None	[0:∞)	Integer	Yes

Examples

```
x = circle(1,1,500)
y = circle(1+j*1,1,500)
```

Defined in

Built in

collapse()

Collapses the inner independent variable and returns one dimensional data

Syntax

y = collapse(x)

Arguments

Name	Description	Default	Range	Type	Required
x	multi-dimensional data to be collapsed (dimension is larger than one and less than four)	None	(-∞:∞)	Integer, real, complex	Yes

Examples

```
Given monte carlo analysis results for the S11 of a transmission line:
It is two-dimensional data: the outer sweep is mcTrial; the inner sweep is the
frequency from 100 MHz to 300 MHz and is given in the following format:
mcTrial freq S11
1 100MHz 0.2
200MHz 0.4
300MHz 0.6
2 100MHz 0.3
200MHz 0.5
300MHz 0.7
Returns a one dimensional data with mcTrail, containing all of the previous
data:
collapsed_S11 = collapse(S11)
mcTrial S11
1 0.2
1 0.4
1 0.6
2 0.3
2 0.5
2 0.7
```

Defined in

$HPEESOF_DIR/expressions/ael/statistical_fun.ael

See Also

expand() (expmeas)

Notes/Equations

The collapse() function cannot be used to convert a matrix into an array or vector. Use the expand() function instead.

contour_ex()

Generates contours at desired levels that you specify, or at "round number" values on polar, or rectangular surface data

Syntax

Response1_Contours=contour_ex(data, rect_or_polar, step_size, num_lines, desired_levels, interp_type, min_or_max, data_format)

Arguments

Name	Description	Default	Range	Type	Required
data	two-dimensional data	None	(-∞:∞)	Real	Yes
rect_or_polar	specifies whether to use the contour() or contour_polar() function internally	"POLAR" or 1	(0, 1), or ("RECT", "POLAR") 0/"RECT" - use contour(), 1/"POLAR" - use contour_polar()	integer/string	No
step_size	step size between contour lines, a single real or integer number	1	(0:∞)	Real or Integer	No
num_lines	number of contour lines requested	5	[1:∞)	Real or Integer	No
desired_levels	specific desired contour levels	None	(-∞:∞)	Real or Integer single value or array	No
interp_type	specifies the type of interpolation to perform	2	(0, 1, 2) 0 - No Interpolation, 1 - Cubic Spline, 2 - B-Spline	integer	No
min_or_max	boolean variable that specifies whether the contours are to be drawn from the maximum value down or from the minimum value up	1	(0, 1), or ("Min", "Max") 0/"Min", from the minimum value up 1/"Max", from the maximum value down	integer or string	No
data_format	specifies data format. It is required for the polar contour plot	"RI"	"RI"/"MA"	string	No

Examples

```
polar contours
Response1=PAE
StepSize=1
Num_Lines=5
InterpType=2
Response1_Contours=contour_ex(Response1, StepSize, Num_Lines, [25,30,35],
InterpType)
```

Defined in

Built in

See Also

contour() (expmeas), *contour_polar()* (expmeas)

Notes/Equations

This function is an extended function for contour() and contour_polar(). It generates contours at desired levels that you specify, or at "round number" values on polar, or rectangular surface data.

contour()

Generates contour levels on surface data

Syntax

y = contour(data, contour_levels, interpolation_type)

Arguments

Name	Description	Default	Range	Type	Required
data	data to be contoured, which must be at least two-dimensional real, integer or implicit	None	(-∞:∞)	Integer, Real	Yes
contour_levels	one-dimensional quantity specifying the levels of the contours †	six levels equally spaced between the maximum and the minimum of the data	(-∞:∞)	Integer, Real	No
interpolation_type	specifies the type of interpolation to perform	0	[0,1,2] † †	Integer	No

† Normally specified by the sweep generator "[ ]," but can also be specified as a vector
† † Interpolation types are: 0 - No Interpolation,1 - Cubic Spline,2 - B-Spline

Examples

```
a = contour(dB(S11), [1::3::10])
a = contour(dB(S11), {1, 4, 7, 10})
produces a set of four equally spaced contours on a surface generated as a
function of, say, frequency and strip width.
a = contour(dB(S11), {1, 4, 7, 10}, 1)
produces the same set of contours as the above example, but with cubic spline
interpolation.
```

Defined in

Built in

See Also

contour_polar() (expmeas)

Notes/Equations

This function introduces three extra inner independents into the data. The first two are "level", the contour level, and "number", the contour number. For each contour level there may be n contours. The contour is an integer running from 1 to n. The contour is represented as an (x, y) pair with x as the inner independent.

contour_polar()

Generates contour levels on polar or Smith chart surface data

Syntax

y = contour_polar(data, contour_levels, InterpolationType, DataFormat)

Arguments

Name	Description	Default	Range	Type	Required
data	polar or Smith chart data to be contoured, (and therefore is surface data)	None	(-∞:∞)	Integer, Real, Complex	Yes
contour_levels	one-dimensional quantity specifying the levels of the contours †	six levels equally spaced between the maximum and the minimum of the data	(-∞:∞)	Integer, Real	No
InterpolationType	specifies the type of interpolation to perform	0	[0,1,2] † †	Integer	No
DataFormat	format of swept data	"RI"	"RI", "MA" ††	String	No

† Normally specified by the sweep generator "[ ]," but can also be specified as a vector
† † Interpolation types are: 0 - No Interpolation,1 - Cubic Spline,2 - B-Spline
†† DataFormat are: "RI" = real-imaginary, "MA" = magnitude-phase

Examples

```
a = contour_polar(data_polar, [1::4])
a = contour_polar(data_polar, {1, 2, 3, 4})
produces a set of four equally spaced contours on a polar or Smith chart
surface.
a = contour_polar(data_polar, {1, 2, 3, 4}, 2)
produces the same set of contours as the above example, but with B-spline
interpolation.
```

Defined in

$HPEESOF_DIR/expressions/ael/display_fun.ael

See Also

contour() (expmeas)

Notes/Equations

This function introduces three extra inner independents into the data. The first two are "level", the contour level, and "number", the contour number. For each contour level there may be n contours. The contour is an integer running from 1 to n. The contour is represented as an (x,y) pair with x as the inner independent.

copy()

Makes a copy of a multi-dimensional data variable

Syntax

y = copy(DataVar)

Arguments

Name	Description	Default	Range	Type	Required
DataVar	data variable or array that is to be copied	None	None	Boolean, integer, real, complex, string	Yes

Examples

```
result = copy(S21) returns the copy of the data stored in the data variable S21
The array or data variable created above can be used as follows:
indepV = indep(result,"Index");
result[0] = complex(1, 2); Sets the first value to a complex number
indepV[0] = 1GHz;
```

Defined in

Built in

See Also

create() (expmeas), *delete()* (expmeas)

Notes/Equations

Makes a copy of a multi-dimensional data variable, so that the contents of the copy can be manipulated. Data Variables in ADS are data structures that are used to hold multi-dimensional data. Internally they are not implemented as arrays, and therefore do not have the performance of an array. Accessing and setting data in these arrays are

performance intensive and should be noted.

create()

Creates a multi-dimensional data variable

Syntax

y = create(Dimensionality, DependDataType, IndepName, IndepType, NumRows, NumColumns)

Arguments

Name	Description	Default	Range	Type	Required
Dimensionality	dimensionality of the data variable or array	None	[1:∞)	Integer	Yes
DependDataType	Dependent data type	"Real"	†	String	No
IndepName	Name(s) of independent	"__i"	None	String	No
IndepType	Independent data type	"Real"	†	String	No
NumRows	Number of rows	0	[0:∞)	Integer	No
NumColumns	Number of columns	0	[0:∞)	Integer	No

† "Boolean", "Integer", "Real", "Complex", "String" or "Byte16"

Examples

```
result = create(1, "Complex", {"Index"}, {"Real"},1, 1) returns a 1 dimensional
data variable with dependent type Complex, independent name "Index" and type
real with 1 row and column
The array or data variable created above can be used as follows:
indepV = indep(result,"Index");
result[0] = 1.1;
indepV[0] = 1.0;
A 2-dimensional example is given in the Measurement Expression AEL function
below. Test the function by doing the following:
* Copy the code below into the file user_defined_fun.ael in the directory
$HOME/hpeesof/expressions/ael.
* Launch ADS. In a DDS window create an equation: cre2D=datavar_test_create2()
* Display cre2D to see the contents.
defun datavar_test_create2()

{
decl result = create(2, "Complex", {"Index1", "Index2"}, {"String", "Real"},1,
1);
decl idenp1V = indep(result, "Index1");
decl idenp2V = indep(result, "Index2");
decl iD1, iD2;
for (iD1=0; iD1 < 2; iD1++) {
for (iD2=0; iD2 < 3; iD2++) {
result[iD1,iD2] = complex(iD1, iD2);
idenp2V[iD2] = iD2;
} //for
idenp1V[iD1] = strcat("Val", iD1);
} //for
return result;
```

```
}
```

Defined in

Built in

See Also

copy() (expmeas), *delete()* (expmeas)

Notes/Equations

The create() function can only be used from the measurement expression AEL code and not from the DDS, the schematic, or PDE AEL.
See the *Examples* section.

This function is used to create multi-dimensional data variable or arrays. Data Variables in ADS are data structures that are used to hold multi-dimensional data. Internally they are not implemented as arrays, and therefore do not have the performance of an array. Accessing and setting data in these arrays are performance intensive and should be noted. The number of rows and columns are used in specifying the dimension of matrix data. For a scalar this would be 1, 1.

dd_threshold()

Returns the filtered S Parameter matrix

Syntax

S_Filtered = dd_threshold(S, rep_func, mod_func , relop,threshold,low_freq,high_freq)

Arguments

Name	Description	Default	Range	Type	Required
S	the input S matrix that needs to be filtered	None	(-∞:∞)	Real, Complex	Yes
rep_func	a function that reduces linear data to scalar data that can be compared with the threshold eg: dd_rms,min	None	(-∞:∞)	function	Yes
mod_func	a function that is used to modify the Sij data to comparable units with the threshold eg: dB	None	(-∞:∞)	function	Yes
relop	the relational operator, in double quotes as a string, to compare the Sij statistic and the threshold.: lhs = statistic rhs = threshold	">="	(">" , ">=" , "==", "!=" , "<=" ,"<")	String	Yes
threshold	the rhs value for the relational operator	None	(-∞:∞)	Real	Yes
low_freq	lower bound for frequency	None	(-∞)	Real	No
high_freq	higher bound for freq	None	(∞)	Real	No

Examples

```
S_Filtered = dd_threshold(S, dd_rms,dB ,">=", -30.0,10 Hz, 8GHz).
```

Defined in

$HPEESOF_DIR/expressions/ael/utility_fun.ael

See Also

find index() (expmeas), *mix()* (expmeas)

Notes/Equations

This function filteres out unwanted S parameters
This function can accept conditionals such as ==, !=, >, <, >= and <=, but not logical operators such as && and ||.

delete()

Deletes the multi-dimensional data variable

Syntax

y = delete(DataVar)

Arguments

Name	Description	Default	Range	Type	Required
DataVar	data variable or array that is to be deleted	None	None	Boolean, integer, real, complex, string	Yes

Examples

```
result = delete(S21) returns true or false depending on whether the data
variable was deleted or not
```

Defined in

Built in

See Also

copy() (expmeas), *create()* (expmeas)

expand()

Expands the dependent data of a variable into single points by introducing an additional inner independent variable

Syntax

y = expand(x)

Arguments

Name	Description	Default	Range	Type	Required
x	data to be expanded (dimension is larger than one and less than four)	None	(-∞:∞)	Integer, real, Complex	Yes

Examples

```
Given a dependent data A which has independent variables

B: If A is a 1 dimensional data containing 4 points (10, 20, 30, and 40) and
similarly B is made up of 4 points (1, 2, 3, and 4),
Eqn A = [10,20,30,40]
Eqn B = [1,2,3,4]
Eqn C = vs(A,B,"X")
Using expand(C) increases the dimensionality of the data by 1 where each inner
dependent variable ("X") consists of 1 point.
Eqn Y = expand(C)
```

X	C	A	B	Y
1234	10203040	10203040	1234	X=1 10X=2 20X=3 30X=4 40

Defined in

Built in

See Also

collapse() (expmeas)

Notes/Equations

In addition to the application above, the expand() function can also be used to convert a matrix into an array or vector. For example, an S-parameter matrix can be converted into an array by using the equation `arr=expand(S)` .

find()

Finds the indices of the conditions that are true. Use with all simulation data

Syntax

indices = find(condition)

Arguments

Name	Description	Default	Range	Type	Required
condition	condition	None	None	string	Yes

Examples

```
Given an S-parameter data swept as a function of frequency, find the value of
S11 at 1GHz:
index_1 = find(freq == 1GHz)
```

```
data = S11[index_1]
Given an S-parameter data swept as a function of frequency, find the values of
the frequencies where the
magnitude of S11 is greater than a given value.
lookupValue = 0.58
indices = find(mag(S11) > lookupValue))
firstPoint = indices[0]
lastPoint = indices[sweep_size(indices)-1]
freqDifference = freq[lastPoint]- freq[firstPoint]
The following examples assume a Harmonic Balance data vtime, and a marker m1.
Find the dependent value at the marker:
vVal = find(indep(vtime) >= indep(m1) && indep(vtime) <= indep(m1))
Find all the dependent values less than that of the m1 or the value at m1:
vVal = find(indep(vtime) < indep(m1) ,, indep(vtime) == indep(m1))
Find all the dependent values that are not equal to m1:
vVal = find(indep(vtime) != indep(m1))
```

Defined in

$HPEESOF_DIR/expressions/ael/utility_fun.ael

See Also

find_index() (expmeas), *mix()* (expmeas)

Notes/Equations

The find function will return all the indices of the conditions that are true. If none of the conditions are true, then a -1 is returned. The find function performs an exhaustive search on the given data. The supplied data can be an independent or dependent data. In addition, the dimension of the data that is returned will be identical to the dimension of the input data.
The find function can accept conditionals such as ==, !=, >, <, >= and <=, and logical operators such as && and ||.

find_index()

Finds the closest index for a given search value. Use with all simulation data

Syntax

index = find_index(data_sweep, search_value)

Arguments

Name	Description	Default	Range	Type	Required
data_sweep	data to search	None	(-∞:∞)	Integer, Real, complex, string	Yes
search_value	value to search	None	(-∞:∞)	Integer, Real, complex, string	Yes

Examples

Given S-parameter data swept as a function of frequency, find the value of S11 at 1 GHz:
index = find_index(freq, 1GHz)
a = S11[index]

Defined in

Built in

See Also

find() (expmeas), *mix()* (expmeas)

Notes/Equations

To facilitate searching, the find_index function finds the index value in a sweep that is closest to the search value. Data of type int or real must be monotonic. find_index also performs an exhaustive search of complex and string data types.

Note
For a more robust and versatile data-search tool, see the *find()* (expmeas) function.

generate()

This function generates a sequence of real numbers. The modern way to do this is to use the sweep generator "[ ]."

Syntax

y = generate(start, stop, npts)

Arguments

Name	Description	Default	Range	Type	Required
start	start value of sequence	None	(-∞:∞)	Integer, real	Yes
stop	stop value of sequence	None	(-∞:∞)	Integer, real	Yes
npts	Numper of points in the sequence	None	[2:∞)	Integer	Yes

Examples

```
a = generate(9, 4, 6) returns the sequence 9., 8., 7., 6., 5., 4
```

Defined in

$HPEESOF_DIR/expressions/ael/elementary_fun.ael

get_attr()

Gets a data attribute. This function only works with frequency swept variables.

Syntax

y = get_attr(data, "attr_name", eval)

Arguments

Name	Description	Default	Range	Type	Required
data	frequency swept variable	None	(-∞:∞)	Integer, Real, Complex	Yes
attr_name	name of the attribute	None	None	string	Yes
eval	specifies whether to evaluate the attribute	true	false,true	boolean	No

Examples

```
y = get_attr(data, "fc", true) returns 10GHz
y = get_attr(data, "dataType") returns "TimedData"
y = get_attr(data, "TraceType", false) returns "Spectral"
```

Defined in

Built in

See Also

set_attr() (expmeas)

get_indep_values()

Returns the independent values associated with the given dependent value as an array.

Syntax

indepVals = get_indep_values(Data, LookupValue)

Arguments

Name	Description	Default	Range	Type	Required
Data	1 to 5 dimensional array.	None	(-∞:∞)	Integer, Real, Complex	Yes
LookupValue	Dependent value for which the corresponding independent values have to be found.	None	(-∞:∞)	Real, Complex	Yes
Tolerance	tolerance to be used while comparing numbers	0	[0:∞)	Real	No
All	Finds all matches of the LookupValue. Default behavior is to return after the first match.	0	[0:1]	Integer	No

Examples

```
We assume that the data is 2-dimensional i.e. 2 independent variables created
from a Harmonic Balance Analysis with Pout being the output data.
indepVals = get_indep_values(Pout, max(max(Pout)) returns the values of the
indepenent as an array.
indepVals = get_indep_values(Pout, [m1,m2]) returns the indepenent values of
the markers m1 and m2.
```

Defined in

$HPEESOF_DIR/expressions/ael/utility_fun.ael

See Also

indep() (expmeas)

Notes/Equations

This function can be used only on 1 to 5 dimensional data. The independent values have to be real. The dependent value to be looked up can be a single value or multiple values.

indep()

Returns the independent attached to the data

Syntax

Y = indep(x, NumberOrName)

Arguments

Name	Description	Default	Range	Type	Required
x	data to access the independent values	None	(-∞:∞)	Integer, Real, Complex	Yes
NumberOrName	number or name of independent	0	[0:6]	Integer, string	No

Examples

```
Given S-parameters versus frequency and power: Frequency is the innermost
independent, so its index is 1. Power has index 2.
freq = indep(S, 1)
```

```
freq = indep(S, "freq")
power = indep(S, 2)
power = indep(S, "power")
```

Defined in

Built in

See Also

find_index() (expmeas), *get_indep_values()* (expmeas)

Notes/Equations

The indep() function returns the independent (normally the swept variable) attached to simulation data. When there is more than one independent, then the independent of interest may be specified by number or by name. If no independent specifications are passed, then indep() returns the innermost independent.

max_index()

Returns the index of the maximum

Syntax

max_index(x)

Arguments

Name	Description	Default	Range	Type	Required
x	data to find maximum index	None	(-∞:∞)	Integer, real or complex	Yes

Examples

```
y = max_index([1, 2, 3]) returns 2
y = max_index([3, 2, 1]) returns 0
```

Defined in

Built-in

See Also

min_index() (expmeas)

min_index()

Returns the index of the minimum

Syntax

y = min_index(x)

Arguments

Name	Description	Default	Range	Type	Required
x	data to find the minimum index	None	(-∞:∞)	Integer, real or complex	Yes

Examples

```
a = min_index([3, 2, 1]) returns 2
a = min_index([1, 2, 3]) returns 0
```

Defined in

Built in

See Also

max_index() (expmeas)

permute()

Permutes data based on the attached independents

Syntax

y = permute(data, permute_vector)

Arguments

Name	Description	Default	Range	Type	Required
data	any N-dimensional square data (all inner independents must have the same value N)	None	(-∞:∞)	Integer, Real, Complex	Yes
permute_vector	any permutation vector of the numbers 1 through N †	None	(-∞:∞)	Integer, Real, Complex	Yes

† The permute_vector defaults to {N::1}, representing a complete reversal of the data with respect to its independent variables. If permute_vector has fewer than N entries, the remainder of the vector, representing the outer independent variables, is filled in. In this way, expressions remain robust when outer sweeps are added

Examples

```
Lets assume that the variable data has 3 independent variables. Then:
a1 = permute(data)
```

```
reverses the (three inner independents of) the data.
a2 = permute(data, {3, 2, 1})
Same as above.
aOrig = permute(data, {1, 2, 3})
preserves the data.
In the example below lets assume that a DC analysis has been done with two
independent variables VGS and VDS, and IDS.i is the dependent variable. To see
a plot of IDS vs VGS for different values of VDS the data can be permuted as
follows:
permutedData=permute(IDS.i,{2,1})
See the example $HPEESOF_DIR/examples/Tutorial/sweep.dds
```

Defined in

Built in

See Also

plot_vs() (expmeas)

Notes/Equations

The permute() function is used to swap the order of the independent variables that are attached to a data variable. For example, a data could have two independent variables in a particular order. To swap the order so that it can be easily plotted, the order of the independents must be swapped. The permute() function can be used for this purpose.

The permute() function cannot be used to swap the rows and columns of a matrix. However, it can be used to swap the orders of the independent, even if the dependent is a matrix. For example, a parameter sweep of an S-parameter analysis.

plot_vs()

Attaches an independent to data for plotting

Syntax

y = plot_vs(dependent, independent)

Arguments

Name	Description	Default	Range	Type	Required
dependent	any N-dimensional square data (all inner independents must have the same value N)	None	(-∞:∞)	Integer, Real, Complex	Yes
independent	independent variable	None	(-∞:∞)	Integer, Real	Yes

Examples

```
Example 1:
```

```
a=[1, 2, 3]
b=[4, 5, 6]
c=plot_vs(a, b)
Builds c with independent b, and dependent a.
Example 2:
Assume that an S-parameter analysis has been done with one swept variable Cval
(of say 10 values) for 20 frequency points. The dependent data dbS11=db(S11) is
of 2 dimension and Dependency of [10, 20]. A standard plot would display dbS11
vs freq(the inner independent), for 10 values of Cval. Instead to plot dbS11 vs
Cval, the plot_vs() function can used as follows:
plot_vs(dbS11, Cval)
To plot dbS11 for half the values of Cval:
CvalH=Cval/2
plot_vs(dbS11, CvalH)
Example 3:
In the example below lets assume that a DC analysis has been done with two
independent variables Vgs and Vds, and Ids.i is the dependent variable. To see
a plot of Ids vs Vgs for different values of Vds the data can be plotted as
follows:
plot_vs(Ids.i, Vgs)
```

Defined in

$HPEESOF_DIR/expressions/ael/display_fun.ael

See Also

indep() (expmeas), *permute()* (expmeas), *vs()* (expmeas)

Notes/Equations

When using plot_vs(), the independent and dependent data should be the same size (i.e., not irregular). This function works as follows:

- Checks to see if the argument "independent", is an independent of argument "depend" or argument "independent" is dis-similar to independent of argument "depend".
- If one of the above conditions is met, then the data is swapped or sliced, and the new result formed with the argument "independent" is returned.

set_attr()

Sets the data attribute

Syntax

y = set_attr(data, "attr_name", attribute_value)

Arguments

Name	Description	Default	Range	Type	Required
data	data	None	(-∞:∞)	Integer, Real, Complex	Yes
attr_name	name of the attribute	None	None	string	Yes
attribute_value	value of the attribute	None	(-∞:∞)	boolean, integer, real, complex	Yes

Examples

```
a = set_attr(data, "TraceType", "Spectral")
a = set_attr(data, "TraceType", "Histogram")
```

Defined in

Built in

Notes: When a variable's attributes are set using set_attr, any other equations using that variable are not re-evaluated.

See Also

get_attr() (expmeas)

size()

Returns the row and column size of a vector or matrix

Syntax

y = size(x)

Arguments

Name	Description	Default	Range	Type	Required
x	data	None	(-∞:∞)	Integer, Real, Complex	Yes

Examples

```
Given 2-port S-parameters versus frequency, and given 10 frequency points. Then
for ten 2 2 matrices, size() returns the dimensions of the S-parameter matrix,
and its companion function sweep_size() returns the size of the sweep:
Y = size(S) returns {2, 2}
Y = sweep_size(S) returns 10
```

Defined in

Built in

See Also

sweep_size() (expmeas)

sort()

This measurement returns a sorted variable in ascending or descending order. The sorting can be done on the independent or dependent variables. String values are sorted by folding them to lower case.

Syntax

y = sort(data, sortOrder, indepName)

Arguments

Name	Description	Default	Range	Type	Required
data	data to be sorted (multidimensional scalar variable)	None	(-∞:∞)	Integer, real or complex	Yes
sortOrder	sorting order	"ascending"	"ascending" or "descending"	string	No
indepName	specify the name of the independent variable for sorting	dependent value †	None	string	No

† if indepName not specified, the sorting is done on the dependent.

Examples

```
a = sort(data)
a = sort(data, "descending", "freq")
```

Defined in

Built in

sweep_dim()

Returns the dimensionality of the data

Syntax

y = sweep_dim(x)

Arguments

Name	Description	Default	Range	Type	Required
x	data	None	(-∞:∞)	Integer, Real, Complex	Yes

Examples

```
a = sweep_dim(1) returns 0
a = sweep_dim([1, 2, 3]) returns 1
```

Defined in

Built in

See Also

sweep_size() (expmeas)

sweep_size()

Returns the sweep size of a data object

Syntax

y = sweep_size(x)

Arguments

Name	Description	Default	Range	Type	Required
x	data	None	(-∞:∞)	Integer, Real, Complex	Yes

Examples

```
Given 2-port S-parameters versus frequency, and given 10 frequency points,
there are then ten 2 X 2 matrices. sweep_size() is used to return the sweep
size of the S-parameter matrix, and its companion function size() returns the
dimensions of the S-parameter matrix itself:
a = sweep_size(S) returns 10
a = size(S) returns {2, 2}
Irregular data:
Assume that the data is 3 dimensional with the last dimension being irregular.
The independents are: Vsrc, size, time with dimension [3,2,irreg]. Then:
SwpSz=sweep_size(data) would return:
__SIZE SwpSz
Vsrc=1.0,size=1.0
1 20
Vsrc=1.0,size=2.0
1 21
Vsrc=2.0,size=1.0
1 59
Vsrc=2.0,size=2.0
1 61
Vsrc=3.0,size=1.0
1 76
Vsrc=3.0,size=2.0
1 78
where __SIZE is an independent added by the sweep_size() function.
sweep_size(SwpSz) would return the correct size of the two outer variables:
(1) (2) (3)
3 2 1
```

Defined in

Built in

See Also

size() (expmeas), *sweep_dim()* (expmeas)

Notes/Equations

For regular data, this function returns a vector with an entry corresponding to the length of each sweep. For irregular data, the function returns a multi-dimensional data, which needs to be processed further to get the size. See example above.

type()

Returns the type of the data.

Syntax

y = type(x)

Arguments

Name	Description	Default	Range	Type	Required
x	data to find the type	None	(-∞:∞)	Integer, Real, Complex, String	Yes

Examples

```
a = type(1) returns "Integer"
a = type(1.1) returns "Real"
a = type(1i) returns "Complex"
a = type("type") returns "String"
```

Defined in

Built in

See Also

what() (expmeas)

vs()

Attaches an independent to dependent data

Syntax

y = vs(dependent, independent, indepName)

Arguments

Name	Description	Default	Range	Type	Required
dependent	dependent values	None	(-∞:∞)	Integer, real	Yes
independent	independent values	None	(-∞:∞)	Integer, real, string, complex	Yes
indepName	independent name	None	None	string	No

Examples

```
a=[1, 2, 3]
b=[4, 5, 6]
c = vs(a, b)
```

Defined in

Built in

See Also

indep() (expmeas), *plot_vs()* (expmeas)

Notes/Equations

Use the plot_vs() function to plot the dependent with the order of independent changed. For example, to plot Ids vs Vgs, in a DC analysis data with two independent variables, Vgs and Vds, and a dependent variable Ids.i, use as below:
plot_vs(Ids.i, Vgs)

what()

Returns size and type of data

Syntax

y = what(x, DisplayBlockName)

Arguments

Name	Description	Default	Range	Type	Required
x	data	None	(-∞:∞)	Integer, Real, Complex, String	Yes
DisplayBlockName	Displays block name	0	[0:1] †	Integer	No

† If DisplayBlockName equals 0, no block name is specified (default behavior). If DisplayBlockName equals 1, then block name is displayed. If DisplayBlockName is not equal to 0 or 1, it defaults to 0.

Examples

```
x=[10,20,30,40]
y=what(x)
returns:
y
Dependency : [ ]
Num. Points : [4]
Matrix Size : scalar
Type : Integer
y=what(x, 1)
returns:
y
Dependency : [ ]
Num. Points : [4]
Matrix Size : scalar
Type : Integer
Block Name: __tmp_XX
```

Defined in

Built in

See Also

type() (expmeas)

Notes/Equations

This function is used to determine the dimensions of a piece of data, the attached independents, the type, and (in the case of a matrix) the number of rows and columns. Use what() by entering a listing column and using the trace expression what(x).

write_var()

Writes dataset variables to a file

Syntax

y = write_var(FileName, WriteMode, Comment, Delimiter, Format, Precision, Var1, Var2,...,VarN)

Arguments

Name	Description	Default	Range	Type	Required
FileName	Name of the output file	None	None	String	Yes
WriteMode	Describes the write mode - overwrite or append	None	"W","A" †	String	Yes
Comment	Text to be written at the top of the file	""	None	String	No
Delimiter	Delimiter that separates the data	"\t"	None	String	No
Format	Format of the data	"f"	"f","s" ††	String	No
Precision	precision of the data	6	[1:64]	Integer	No
Var1,...,VarN	Data variables to be written	None	None	dataset variable	Yes

† WriteMode: "W" - overwrite the file, "A" - append to the file
†† Format: "f" - full notation, "s" - scientific notation

Examples

```
write_var_f=write_var("output_S21.txt","W","! Freq real(S21) imag(S21)"," ",,
14, freq, S21)
writes S21 to the output file output_S21.txt as:
! Freq real(S21) imag(S21)
1000000000 0.60928892074273 -0.10958342264718
2000000000 0.52718867597783 -0.13319167002392
3000000000 0.4769067837712 -0.12080489345341
wv_ib=write_var("output_hbIb.txt","W","! HB Ib.i", " ",,,freq, Ib.i)
write the Harmonic Balance frequency and current Ib.i to the output file
output_hbIb.txt.
```

Defined in

$HPEESOF_DIR/expressions/ael/utility_fun.ael

See Also

indep() (expmeas)

Notes/Equations

This function can be used to write multiple dataset variables to a file. Currently only 1 dimensional data is supported. All variables that are to be written must be of the same size. Each variable data is written in column format. Complex data type is written in 2 columns as real and imaginary.

FrontPanel Eye Diagram Functions

This section describes the eye diagram FrontPanel and associated functions in detail. These functions are not generalized functions, they are provided specifically to support the eye diagram utility available from the Data Display.

- *eye binning()* (expmeas)
- *eye density()* (expmeas)
- *FrontPanel eye 2d indepvar maximum inner()* (expmeas)
- *FrontPanel eye()* (expmeas)
- *FrontPanel eye amplitude histogram()* (expmeas)
- *FrontPanel eye crossings()* (expmeas)
- *FrontPanel eye delay()* (expmeas)
- *FrontPanel eye fall trace()* (expmeas)
- *FrontPanel eye horizontal histogram()* (expmeas)
- *FrontPanel eye regular()* (expmeas)
- *FrontPanel eye risefall marker()* (expmeas)
- *FrontPanel eye rise trace()* (expmeas)
- *FrontPanel eye topbase()* (expmeas)
- *FrontPanel get histogram mean stddev()* (expmeas)
- *FrontPanel pp rms jitter()* (expmeas)
- *FrontPanel wave 1st falling edge period()* (expmeas)
- *FrontPanel wave 1st rising edge period()* (expmeas)
- *FrontPanel wave 1st transition fall time()* (expmeas)
- *FrontPanel wave 1st transition rise time()* (expmeas)
- *FrontPanel wave datarate()* (expmeas)
- *FrontPanel wave negative pulse width()* (expmeas)
- *FrontPanel wave positive pulse width()* (expmeas)
- *FrontPanel wave topbase()* (expmeas)

Working with the Eye Diagram FrontPanel

The eye diagram FrontPanel is available from the Data Display **Tools > Eye Diagram** menu. The FrontPanel features two modes of operation: *oscilloscope* and *eye/mask* . For more information on working with the eye diagram FrontPanel, refer to the *Data Display* documentation.

eye_binning()

Returns density binning data

Syntax

y = eye_binning(eye_data, Indep_bins, Dep_bins)

Arguments

Name	Description	Default	Range	Type	Required
eye_data	eye diagram data	None	(-∞:∞)	Real	Yes
Indep_bins	number of independent bins	None	[1:∞)	Integer	Yes
Dep_bins	number of dependent bins	None	[1:∞)	Integer	Yes

Examples

```
Eye_data=FrontPanel_eye(vout,10GHz)
Eye_bins = eye_binning(Eye_data, 100, 70);
```

Defined in

$HPEESOF_DIR/expressions/ael/DesignGuide_fun.ael

See Also

eye() (expmeas), *eye_density()* (expmeas), *FrontPanel_eye()* (expmeas), *FrontPanel_eye_2d_indepvar_maximum_inner()* (expmeas), *FrontPanel_eye_amplitude_histogram()* (expmeas), *FrontPanel_eye_crossings()* (expmeas), *FrontPanel_eye_delay()* (expmeas), *FrontPanel_eye_fall_trace()* (expmeas), *FrontPanel_eye_horizontal_histogram()* (expmeas), *FrontPanel_eye_regular()* (expmeas), *FrontPanel_eye_rise_trace()* (expmeas), *FrontPanel_eye_risefall_marker()* (expmeas), *FrontPanel_eye_topbase()* (expmeas), *FrontPanel_get_histogram_mean_stddev()* (expmeas), *FrontPanel_pp_rms_jitter()* (expmeas), *FrontPanel_wave_1st_falling_edge_period()* (expmeas), *FrontPanel_wave_1st_rising_edge_period()* (expmeas), *FrontPanel_wave_1st_transition_fall_time()* (expmeas), *FrontPanel_wave_1st_transition_rise_time()* (expmeas), *FrontPanel_wave_datarate()* (expmeas), *FrontPanel_wave_negative_pulse_width()* (expmeas), *FrontPanel_wave_positive_pulse_width()* (expmeas), *FrontPanel_wave_topbase()* (expmeas)

Notes/Equations

This function takes a multi-dimensional eye diagram plot and performs binning. The binning is performed by slicing the eye diagram on its X-axis (Indep_bins) and Y-axis (Dep_bins) and returning number of traces passing through every bin. The eye binning information can be used to calculate trace distribution in any region of the eye diagram.

eye_density()

Returns density trace.

Syntax

y = eye_density(eye_data, Indep_bins, Dep_bins)

Arguments

Name	Description	Default	Range	Type	Required
eye_data	eye diagram data	None	(-∞:∞)	Real	Yes
Indep_bins	number of independent bins	None	[1:∞)	Integer	Yes
Dep_bins	number of dependent bins	None	[1:∞)	Integer	Yes

Examples

```
Eye_data=FrontPanel_eye(vount,10GHz)
Eye_bins = eye_density(Eye_data, 100, 70);
```

Defined in

$HPEESOF_DIR/expressions/ael/DesignGuide_fun.ael

See Also

eye() (expmeas), *eye_binning()* (expmeas), *FrontPanel_eye()* (expmeas), *FrontPanel_eye_2d_indepvar_maximum_inner()* (expmeas), *FrontPanel_eye_amplitude_histogram()* (expmeas), *FrontPanel_eye_crossings()* (expmeas), *FrontPanel_eye_delay()* (expmeas), *FrontPanel_eye_fall_trace()* (expmeas), *FrontPanel_eye_horizontal_histogram()* (expmeas), *FrontPanel_eye_regular()* (expmeas), *FrontPanel_eye_rise_trace()* (expmeas), *FrontPanel_eye_risefall_marker()* (expmeas), *FrontPanel_eye_topbase()* (expmeas), *FrontPanel_get_histogram_mean_stddev()* (expmeas), *FrontPanel_pp_rms_jitter()* (expmeas), *FrontPanel_wave_1st_falling_edge_period()* (expmeas), *FrontPanel_wave_1st_rising_edge_period()* (expmeas), *FrontPanel_wave_1st_transition_fall_time()* (expmeas), *FrontPanel_wave_1st_transition_rise_time()* (expmeas), *FrontPanel_wave_datarate()* (expmeas), *FrontPanel_wave_negative_pulse_width()* (expmeas), *FrontPanel_wave_positive_pulse_width()* (expmeas), *FrontPanel_wave_topbase()* (expmeas)

FrontPanel_eye_2d_indepvar_maximum_inner()

Given two dimensional data, returns maximum value of the inner indpendent variable

Syntax

y = FrontPanel_eye_2d_indepvar_maximum_inner(eye_bin_data)

Arguments

Name	Description	Default	Range	Type	Required
eye_bin_data	eye binning data	None	(-∞:∞)	real	Yes

Examples

```
maximum_indep=FrontPanel_eye_2d_indepvar_maximum_inner(eye_bin_data)
```

Defined in

$HPEESOF_DIR/expressions/ael/FrontPanel_fun.ael

See Also

eye_binning() (expmeas), *eye_density()* (expmeas), *FrontPanel_eye()* (expmeas), *FrontPanel_eye_amplitude_histogram()* (expmeas), *FrontPanel_eye_crossings()* (expmeas), *FrontPanel_eye_delay()* (expmeas), *FrontPanel_eye_fall_trace()* (expmeas), *FrontPanel_eye_horizontal_histogram()* (expmeas), *FrontPanel_eye_regular()* (expmeas), *FrontPanel_eye_rise_trace()* (expmeas), *FrontPanel_eye_risefall_marker()* (expmeas), *FrontPanel_eye_topbase()* (expmeas), *FrontPanel_get_histogram_mean_stddev()* (expmeas), *FrontPanel_pp_rms_jitter()* (expmeas), *FrontPanel_wave_1st_falling_edge_period()* (expmeas), *FrontPanel_wave_1st_rising_edge_period()* (expmeas), *FrontPanel_wave_1st_transition_fall_time()* (expmeas), *FrontPanel_wave_1st_transition_rise_time()* (expmeas), *FrontPanel_wave_datarate()* (expmeas), *FrontPanel_wave_negative_pulse_width()* (expmeas), *FrontPanel_wave_positive_pulse_width()* (expmeas), *FrontPanel_wave_topbase()* (expmeas)

Notes/Equations

The FrontPanel_eye_2d_indepvar_maximum_inner() function essentially takes two dimensional binning data and returns maximum value of inner independent variable.

FrontPanel_eye()

Creates data for an eye diagram plot with one bit shift per trace, adding an additional cycle to the number of cycles defined

Syntax

y = FrontPanel_eye(NRZ_data, symbolRate, Cycles, Delay)

Arguments

Name	Description	Default	Range	Type	Required
NRZ_data	either numeric data or a time domain waveform, typically NRZ data	None	(-∞:∞)	Complex	Yes
symbolRate	bit rate of the channel	None	(0:∞)	Real	Yes
Cycles	number of cycles to repeat	1	[1:∞)	Integer	No
Delay	sampling delay	0	[0:∞)	Integer, Real	No

Examples

```
y = FrontPanel_eye(NRZ_data, symbol_rate)
```

Defined in

Built in

See Also

eye_binning() (expmeas), *eye_density()* (expmeas), *FrontPanel_eye_2d_indepvar_maximum_inner()* (expmeas), *FrontPanel_eye_amplitude_histogram()* (expmeas), *FrontPanel_eye_crossings()* (expmeas), *FrontPanel_eye_delay()* (expmeas), *FrontPanel_eye_fall_trace()* (expmeas), *FrontPanel_eye_horizontal_histogram()* (expmeas), *FrontPanel_eye_regular()* (expmeas), *FrontPanel_eye_rise_trace()* (expmeas), *FrontPanel_eye_risefall_marker()* (expmeas), *FrontPanel_eye_topbase()* (expmeas), *FrontPanel_get_histogram_mean_stddev()* (expmeas), *FrontPanel_pp_rms_jitter()* (expmeas), *FrontPanel_wave_1st_falling_edge_period()* (expmeas), *FrontPanel_wave_1st_rising_edge_period()* (expmeas), *FrontPanel_wave_1st_transition_fall_time()* (expmeas), *FrontPanel_wave_1st_transition_rise_time()* (expmeas), *FrontPanel_wave_datarate()* (expmeas), *FrontPanel_wave_negative_pulse_width()* (expmeas), *FrontPanel_wave_positive_pulse_width()* (expmeas), *FrontPanel_wave_topbase()* (expmeas)

Notes/Equations

This function will shift the eye trace by one bit when multiple eyes are plotted as opposed to eye() function.

FrontPanel_eye_amplitude_histogram()

Returns amplitude histogram from binning data

Syntax

y = FrontPanel_eye_amplitude_histogram(data)

Arguments

Name	Description	Default	Range	Type	Required
data	Eye binning data, usually the output of eye_binning function	None	(-∞:∞)	Real	Yes

Examples

```
Find_hist = FrontPanel_eye_amplitude_histogram(eye_bin_data)
```

Defined in

$HPEESOF_DIR/expressions/ael/FrontPanel_fun.ael

See Also

eye_binning() (expmeas), *eye_density()* (expmeas), *FrontPanel_eye()* (expmeas), *FrontPanel_eye_2d_indepvar_maximum_inner()* (expmeas), *FrontPanel_eye_crossings()* (expmeas), *FrontPanel_eye_delay()* (expmeas), *FrontPanel_eye_fall_trace()* (expmeas), *FrontPanel_eye_horizontal_histogram()* (expmeas), *FrontPanel_eye_regular()* (expmeas), *FrontPanel_eye_rise_trace()* (expmeas), *FrontPanel_eye_risefall_marker()* (expmeas), *FrontPanel_eye_topbase()* (expmeas), *FrontPanel_get_histogram_mean_stddev()* (expmeas), *FrontPanel_pp_rms_jitter()* (expmeas), *FrontPanel_wave_1st_falling_edge_period()* (expmeas), *FrontPanel_wave_1st_rising_edge_period()* (expmeas), *FrontPanel_wave_1st_transition_fall_time()* (expmeas), *FrontPanel_wave_1st_transition_rise_time()* (expmeas), *FrontPanel_wave_datarate()* (expmeas), *FrontPanel_wave_negative_pulse_width()* (expmeas), *FrontPanel_wave_positive_pulse_width()* (expmeas), *FrontPanel_wave_topbase()* (expmeas)

Notes/Equations

The FrontPanel_eye_amplitude_histogram() function calculates the amplitude histogram from eye binning data.

FrontPanel_eye_crossings()

Given lower boundary, top boundary and the data type returns the eye crossing points

Syntax

y = FrontPanel_eye_crossings(data,base_upper,top_lower,data_type)

Arguments

Name	Description	Default	Range	Type	Required
data	eye binning data	None	(-∞:∞)	real	Yes
base_upper	upper three sigma point of the base boundary	None	(-∞:∞)	Real	Yes
top_lower	lower three sigma point of the top boundary	None	(-∞:∞)	Real	Yes
data_type	data type, current version supports only "NRZ" data	None	String	Yes	

Examples

```
Eye_crossing_array = FrontPanel_eye_crossings(eye_bin_data,0.2,0.8,"NRZ")
Eye_crossing_1_level=Eye_crossing_array[0]
Eye_crossing_1_time=Eye_crossing_array[1]
Eye_crossing_2_level=Eye_crossing_array[3]
Eye_crossing_2_time=Eye_crossing_array[4]
```

Defined in

$HPEESOF_DIR/expressions/ael/FrontPanel_fun.ael

See Also

eye_binning() (expmeas), *eye_density()* (expmeas), *FrontPanel_eye()* (expmeas), *FrontPanel_eye_2d_indepvar_maximum_inner()* (expmeas), *FrontPanel_eye_amplitude_histogram()* (expmeas), *FrontPanel_eye_delay()* (expmeas), *FrontPanel_eye_fall_trace()* (expmeas), *FrontPanel_eye_horizontal_histogram()* (expmeas), *FrontPanel_eye_regular()* (expmeas), *FrontPanel_eye_rise_trace()* (expmeas), *FrontPanel_eye_risefall_marker()* (expmeas), *FrontPanel_eye_topbase()* (expmeas), *FrontPanel_get_histogram_mean_stddev()* (expmeas), *FrontPanel_pp_rms_jitter()* (expmeas), *FrontPanel_wave_1st_falling_edge_period()* (expmeas), *FrontPanel_wave_1st_rising_edge_period()* (expmeas), *FrontPanel_wave_1st_transition_fall_time()* (expmeas), *FrontPanel_wave_1st_transition_rise_time()* (expmeas), *FrontPanel_wave_datarate()* (expmeas), *FrontPanel_wave_negative_pulse_width()* (expmeas), *FrontPanel_wave_positive_pulse_width()* (expmeas), *FrontPanel_wave_topbase()* (expmeas)

Notes/Equations

The FrontPanel_eye_crossings() function essentially takes the eye binning data, the top and base boundary limits, and the data type, and returns crossing points as an array.

FrontPanel_eye_delay()

Returns delay required for eye pattern positioning

Syntax

y = FrontPanel_eye_delay(Vout_time,BitRate)

Arguments

Name	Description	Default	Range	Type	Required
Vout_time	time domain voltage waveform	None	(-∞:∞)	Real	Yes
BitRate	bit rate of the channel and is expressed in frequency units	None	[0:∞)	Real	Yes

Examples

```
Find_delay = FrontPanel_eye_delay(vout,1e9)
```

Defined in

$HPEESOF_DIR/expressions/ael/FrontPanel_fun.ael

See Also

eye_binning() (expmeas), *eye_density()* (expmeas), *FrontPanel_eye()* (expmeas), *FrontPanel_eye_2d_indepvar_maximum_inner()* (expmeas), *FrontPanel_eye_amplitude_histogram()* (expmeas), *FrontPanel_eye_crossings()* (expmeas), *FrontPanel_eye_fall_trace()* (expmeas),

FrontPanel_eye_horizontal_histogram() (expmeas), *FrontPanel_eye_regular()* (expmeas), *FrontPanel_eye_rise_trace()* (expmeas), *FrontPanel_eye_risefall_marker()* (expmeas), *FrontPanel_eye_topbase()* (expmeas), *FrontPanel_get_histogram_mean_stddev()* (expmeas), *FrontPanel_pp_rms_jitter()* (expmeas), *FrontPanel_wave_1st_falling_edge_period()* (expmeas), *FrontPanel_wave_1st_rising_edge_period()* (expmeas), *FrontPanel_wave_1st_transition_fall_time()* (expmeas), *FrontPanel_wave_1st_transition_rise_time()* (expmeas), *FrontPanel_wave_datarate()* (expmeas), *FrontPanel_wave_negative_pulse_width()* (expmeas), *FrontPanel_wave_positive_pulse_width()* (expmeas), *FrontPanel_wave_topbase()* (expmeas)

Notes/Equations

The FrontPanel_eye_delay() function determines eye mid crossing point of NRZ data and calculates the delay required to position 50% eye crossing at the center of unit interval.

FrontPanel_eye_fall_trace()

Returns eye falling edges

Syntax

y = FrontPanel_eye_fall_trace(eye_bin_data, level_zero, level_one, low_threshold, high_threshold)

Arguments

Name	Description	Default	Range	Type	Required
eye_bin_data	eye binning data	None	(-∞:∞)	real	Yes
level_zero	logic level zero amplitude	None	(-∞:∞)	Real	Yes
level_one	logic level one amplitude	None	(-∞:∞)	Real	Yes
low_threshold	low threshold percentage	None	(-∞:∞)	Real	Yes
high_threshold	high threshold precentage	None	(-∞:∞)	Real	Yes

Examples

```
Eye_falling_traces = FrontPanel_eye_fall_trace(eye_bin_data,0,1.8, 10, 90)
```

Defined in

$HPEESOF_DIR/expressions/ael/FrontPanel_fun.ael

See Also

eye_binning() (expmeas), *eye_density()* (expmeas), *FrontPanel_eye()* (expmeas),

FrontPanel_eye_2d_indepvar_maximum_inner() (expmeas), *FrontPanel_eye_amplitude_histogram()* (expmeas), *FrontPanel_eye_crossings()* (expmeas), *FrontPanel_eye_delay()* (expmeas), *FrontPanel_eye_horizontal_histogram()* (expmeas), *FrontPanel_eye_regular()* (expmeas), *FrontPanel_eye_rise_trace()* (expmeas), *FrontPanel_eye_risefall_marker()* (expmeas), *FrontPanel_eye_topbase()* (expmeas), *FrontPanel_get_histogram_mean_stddev()* (expmeas), *FrontPanel_pp_rms_jitter()* (expmeas), *FrontPanel_wave_1st_falling_edge_period()* (expmeas), *FrontPanel_wave_1st_rising_edge_period()* (expmeas), *FrontPanel_wave_1st_transition_fall_time()* (expmeas), *FrontPanel_wave_1st_transition_rise_time()* (expmeas), *FrontPanel_wave_datarate()* (expmeas), *FrontPanel_wave_negative_pulse_width()* (expmeas), *FrontPanel_wave_positive_pulse_width()* (expmeas), *FrontPanel_wave_topbase()* (expmeas)

Notes/Equations

The FrontPanel_eye_fall_trace() function essentially takes the eye binning data and returns the falling edges which crosses the lower and upper threshold points.

FrontPanel_eye_horizontal_histogram()

Given start, stop time and amplitude,returns horizontal histogram from the eye binning data

Syntax

y = FrontPanel_eye_horizontal_histogram(data,start_time,stop_time,start_amp,stop_amp)

Arguments

Name	Description	Default	Range	Type	Required
data	eye binning data	None	(-∞:∞)	real	Yes
start_time	start time of horizontal histogram	None	(-∞:∞)	Real	Yes
stop_time	stop time of horizontal histogram	None	(-∞:∞)	Real	Yes
start_amp	start amplitude of horizontal histogram	None	(-∞:∞)	Real	Yes
stop_amp	start amplitude of horizontal histogram	None	(-∞:∞)	Real	Yes

Examples

```
Eye_histogram = FrontPanel_eye_horizontal_histogram(eye_bin_data,0,10e-9, -0.1,
0.1)
```

Defined in

$HPEESOF_DIR/expressions/ael/FrontPanel_fun.ael

See Also

eye_binning() (expmeas), *eye_density()* (expmeas), *FrontPanel_eye()* (expmeas), *FrontPanel_eye_2d_indepvar_maximum_inner()* (expmeas), *FrontPanel_eye_amplitude_histogram()* (expmeas), *FrontPanel_eye_crossings()* (expmeas), *FrontPanel_eye_delay()* (expmeas), *FrontPanel_eye_fall_trace()* (expmeas), *FrontPanel_eye_regular()* (expmeas), *FrontPanel_eye_rise_trace()* (expmeas), *FrontPanel_eye_risefall_marker()* (expmeas), *FrontPanel_eye_topbase()* (expmeas), *FrontPanel_get_histogram_mean_stddev()* (expmeas), *FrontPanel_pp_rms_jitter()* (expmeas), *FrontPanel_wave_1st_falling_edge_period()* (expmeas), *FrontPanel_wave_1st_rising_edge_period()* (expmeas), *FrontPanel_wave_1st_transition_fall_time()* (expmeas), *FrontPanel_wave_1st_transition_rise_time()* (expmeas), *FrontPanel_wave_datarate()* (expmeas), *FrontPanel_wave_negative_pulse_width()* (expmeas), *FrontPanel_wave_positive_pulse_width()* (expmeas), *FrontPanel_wave_topbase()* (expmeas)

Notes/Equations

The FrontPanel_eye_horizontal_histogram() function essentially takes the eye binning data and returns horizontal histogram between start and stop points.

FrontPanel_eye_regular()

Given eye data and the resolution, returns uniformally spaced eye data

Syntax

y = FrontPanel_eye_regular(eye_data,resolution_x)

Arguments

Name	Description	Default	Range	Type	Required
eye_data	eye diagram data	None	(-∞:∞)	real	Yes
resolution_x	inner independent variable resolution	None	(-∞:∞)	real	Yes

Examples

```
eye_data=eye(vout,1GHz)
uniform_eye_data=FrontPanel_eye_regular(eye_data,450)
```

Defined in

$HPEESOF_DIR/expressions/ael/FrontPanel_fun.ael

See Also

eye_binning() (expmeas), *eye_density()* (expmeas), *FrontPanel_eye()* (expmeas), *FrontPanel_eye_2d_indepvar_maximum_inner()* (expmeas), *FrontPanel_eye_amplitude_histogram()* (expmeas), *FrontPanel_eye_crossings()* (expmeas), *FrontPanel_eye_delay()* (expmeas), *FrontPanel_eye_fall_trace()* (expmeas), *FrontPanel_eye_horizontal_histogram()* (expmeas), *FrontPanel_eye_rise_trace()* (expmeas), *FrontPanel_eye_risefall_marker()* (expmeas), *FrontPanel_eye_topbase()* (expmeas), *FrontPanel_get_histogram_mean_stddev()* (expmeas), *FrontPanel_pp_rms_jitter()* (expmeas), *FrontPanel_wave_1st_falling_edge_period()* (expmeas), *FrontPanel_wave_1st_rising_edge_period()* (expmeas), *FrontPanel_wave_1st_transition_fall_time()* (expmeas), *FrontPanel_wave_1st_transition_rise_time()* (expmeas), *FrontPanel_wave_datarate()* (expmeas), *FrontPanel_wave_negative_pulse_width()* (expmeas), *FrontPanel_wave_positive_pulse_width()* (expmeas), *FrontPanel_wave_topbase()* (expmeas)

Notes/Equations

The FrontPanel_eye_regular() function essentially takes eye data, and returns uniform spaced data. The first and last trace of the eye diagram will be removed when using this function.

FrontPanel_eye_risefall_marker()

Returns the threshold points from rising or falling edge data

Syntax

y = FrontPanel_eye_risefall_marker(data, level_zero, level_one, low_threshold, high_threshold)

Arguments

Name	Description	Default	Range	Type	Required
data	eye binning data	None	(-∞:∞)	real	Yes
level_zero	logic level zero amplitude	None	(-∞:∞)	Real	Yes
level_one	logic level one amplitude	None	(-∞:∞)	Real	Yes
low_threshold	low threshold percentage	None	(-∞:∞)	Real	Yes
high_threshold	high threshold precentage	None	(-∞:∞)	Real	Yes

Examples

```
Eye_rise_marker = FrontPanel_eye_risefall_marker(eye_bin_data,0,1.8, 10, 90)
```

Defined in

$HPEESOF_DIR/expressions/ael/FrontPanel_fun.ael

See Also

eye_binning() (expmeas), *eye_density()* (expmeas), *FrontPanel_eye()* (expmeas), *FrontPanel_eye_2d_indepvar_maximum_inner()* (expmeas), *FrontPanel_eye_amplitude_histogram()* (expmeas), *FrontPanel_eye_crossings()* (expmeas), *FrontPanel_eye_delay()* (expmeas), *FrontPanel_eye_fall_trace()* (expmeas), *FrontPanel_eye_horizontal_histogram()* (expmeas), *FrontPanel_eye_regular()* (expmeas), *FrontPanel_eye_rise_trace()* (expmeas), *FrontPanel_eye_topbase()* (expmeas), *FrontPanel_get_histogram_mean_stddev()* (expmeas), *FrontPanel_pp_rms_jitter()* (expmeas), *FrontPanel_wave_1st_falling_edge_period()* (expmeas), *FrontPanel_wave_1st_rising_edge_period()* (expmeas), *FrontPanel_wave_1st_transition_fall_time()* (expmeas), *FrontPanel_wave_1st_transition_rise_time()* (expmeas), *FrontPanel_wave_datarate()* (expmeas), *FrontPanel_wave_negative_pulse_width()* (expmeas), *FrontPanel_wave_positive_pulse_width()* (expmeas), *FrontPanel_wave_topbase()* (expmeas)

Notes/Equations

The FrontPanel_eye_risefall_marker() function essentially takes the eye binning data and returns the gaussian mean position at lower threshold, 50% crossing, and upper threshold crossing points as an array. The first element of the array will give the indepedent and dependent value at the lower threshold, the second element will give the 50% crossing point, and the third element will give the high threshold point.

FrontPanel_eye_rise_trace()

Returns eye rising edges

Syntax

y = FrontPanel_eye_rise_trace(data, level_zero, level_one, low_threshold, high_threshold)

Arguments

Name	Description	Default	Range	Type	Required
data	eye binning data	None	(-∞:∞)	real	Yes
level_zero	logic level zero amplitude	None	(-∞:∞)	Real	Yes
level_one	logic level one amplitude	None	(-∞:∞)	Real	Yes
low_threshold	low threshold percentage	None	(-∞:∞)	Real	Yes
high_threshold	high threshold precentage	None	(-∞:∞)	Real	Yes

Examples

```
Eye_rising_traces = FrontPanel_eye_rise_trace(eye_bin_data,0,1.8, 10, 90)
```

Defined in

$HPEESOF_DIR/expressions/ael/FrontPanel_fun.ael

See Also

eye_binning() (expmeas), *eye_density()* (expmeas), *FrontPanel_eye()* (expmeas), *FrontPanel_eye_2d_indepvar_maximum_inner()* (expmeas), *FrontPanel_eye_amplitude_histogram()* (expmeas), *FrontPanel_eye_crossings()* (expmeas), *FrontPanel_eye_delay()* (expmeas), *FrontPanel_eye_fall_trace()* (expmeas), *FrontPanel_eye_horizontal_histogram()* (expmeas), *FrontPanel_eye_regular()* (expmeas), *FrontPanel_eye_risefall_marker()* (expmeas), *FrontPanel_eye_topbase()* (expmeas), *FrontPanel_get_histogram_mean_stddev()* (expmeas), *FrontPanel_pp_rms_jitter()* (expmeas), *FrontPanel_wave_1st_falling_edge_period()* (expmeas), *FrontPanel_wave_1st_rising_edge_period()* (expmeas), *FrontPanel_wave_1st_transition_fall_time()* (expmeas), *FrontPanel_wave_1st_transition_rise_time()* (expmeas), *FrontPanel_wave_datarate()* (expmeas), *FrontPanel_wave_negative_pulse_width()* (expmeas), *FrontPanel_wave_positive_pulse_width()* (expmeas), *FrontPanel_wave_topbase()* (expmeas)

Notes/Equations

The FrontPanel_eye_rise_trace() function essentially takes the eye binning data and returns the rising edges which crosses the lower and upper threshold points.

FrontPanel_eye_topbase()

Given lower boundary, top boundary and the data type returns the eye crossing points

Syntax

y = FrontPanel_eye_topbase(data,eye_crossing_points,lower_threshold,upper_threshold,data_type)

Arguments

Name	Description	Default	Range	Type	Required
data	eye binning data	None	(-∞:∞)	real	Yes
eye_crossing_points	array of eye crossing points, typically FrontPanel_eye_crossings function returned value	None	(-∞:∞)	Real	Yes
lower_threshold	lower threshold point %	None	(-∞:∞)	Real	Yes
upper_threshold	upper threshold point %	None	(-∞:∞)	Real	Yes
data_type	data type, current version supports only "NRZ" data	None	String	Yes	

Examples

```
eye_crossing_points=FrontPanel_eye_crossings(eye_bin_data,-1.0,1,0,"NRZ")
eye_top_base =
FrontPanel_eye_topbase(eye_bin_data,eye_crossing_points,20,80,"NRZ")
```

Defined in

$HPEESOF_DIR/expressions/ael/FrontPanel_fun.ael

See Also

eye_binning() (expmeas), *eye_density()* (expmeas), *FrontPanel_eye()* (expmeas), *FrontPanel_eye_2d_indepvar_maximum_inner()* (expmeas), *FrontPanel_eye_amplitude_histogram()* (expmeas), *FrontPanel_eye_crossings()* (expmeas), *FrontPanel_eye_delay()* (expmeas), *FrontPanel_eye_fall_trace()* (expmeas), *FrontPanel_eye_horizontal_histogram()* (expmeas), *FrontPanel_eye_regular()* (expmeas), *FrontPanel_eye_rise_trace()* (expmeas), *FrontPanel_eye_risefall_marker()* (expmeas), *FrontPanel_get_histogram_mean_stddev()* (expmeas), *FrontPanel_pp_rms_jitter()* (expmeas), *FrontPanel_wave_1st_falling_edge_period()* (expmeas), *FrontPanel_wave_1st_rising_edge_period()* (expmeas), *FrontPanel_wave_1st_transition_fall_time()* (expmeas), *FrontPanel_wave_1st_transition_rise_time()* (expmeas), *FrontPanel_wave_datarate()* (expmeas), *FrontPanel_wave_negative_pulse_width()* (expmeas), *FrontPanel_wave_positive_pulse_width()* (expmeas), *FrontPanel_wave_topbase()* (expmeas)

Notes/Equations

The FrontPanel_eye_topbase() function essentially takes the eye binning data, the crossing points, the lower and upper threshold percentage value, and returned top and base levels.

Frontpanel_get_histogram_mean_stddev()

Given histogram data computes mean and standard deviation values

Syntax

y = Frontpanel_get_histogram_mean_stddev(data,start_bin,stop_bin)

Arguments

Name	Description	Default	Range	Type	Required
data	histogram data	None	(-∞:∞)	real	Yes
start_bin	start bin number	None	(-∞:∞)	real	Yes
stop_bin	stop bin number	None	(-∞:∞)	real	Yes

Examples

```
histogram_statistics=Frontpanel_get_histogram_mean_stddev(histogram_data,10,100)
histogram_mean=histogram_statistics[0]
histogram_stddev=histogram_statistics[1]
```

Defined in

$HPEESOF_DIR/expressions/ael/FrontPanel_fun.ael

See Also

eye_binning() (expmeas), *eye_density()* (expmeas), *FrontPanel_eye()* (expmeas), *FrontPanel_eye_2d_indepvar_maximum_inner()* (expmeas), *FrontPanel_eye_amplitude_histogram()* (expmeas), *FrontPanel_eye_crossings()* (expmeas), *FrontPanel_eye_delay()* (expmeas), *FrontPanel_eye_fall_trace()* (expmeas), *FrontPanel_eye_horizontal_histogram()* (expmeas), *FrontPanel_eye_regular()* (expmeas), *FrontPanel_eye_rise_trace()* (expmeas), *FrontPanel_eye_risefall_marker()* (expmeas), *FrontPanel_eye_topbase()* (expmeas), *FrontPanel_pp_rms_jitter()* (expmeas), *FrontPanel_wave_1st_falling_edge_period()* (expmeas), *FrontPanel_wave_1st_rising_edge_period()* (expmeas), *FrontPanel_wave_1st_transition_fall_time()* (expmeas), *FrontPanel_wave_1st_transition_rise_time()* (expmeas), *FrontPanel_wave_datarate()* (expmeas), *FrontPanel_wave_negative_pulse_width()* (expmeas), *FrontPanel_wave_positive_pulse_width()* (expmeas), *FrontPanel_wave_topbase()* (expmeas)

Notes/Equations

The FrontPanel_get_histogram_mean_stddev() function essentially takes histogram data and returns mean and standard deviation.

FrontPanel_pp_rms_jitter()

Given histogram data computes peak to peak jitter value and RMS jitter value

Syntax

y = FrontPanel_pp_rms_jitter(histogram_data)

Arguments

Name	Description	Default	Range	Type	Required
histogram_data	histogram data	None	(-∞:∞)	real	Yes

Examples

```
eye_jitter=FrontPanel_pp_rms_jitter(histogram_data)
peak2peak_jitter=eye_jitter[0]
rms_jitter=eye_jitter[1]
```

Defined in

$HPEESOF_DIR/expressions/ael/FrontPanel_fun.ael

See Also

eye_binning() (expmeas), *eye_density()* (expmeas), *FrontPanel_eye()* (expmeas), *FrontPanel_eye_2d_indepvar_maximum_inner()* (expmeas), *FrontPanel_eye_amplitude_histogram()* (expmeas), *FrontPanel_eye_crossings()* (expmeas), *FrontPanel_eye_delay()* (expmeas), *FrontPanel_eye_fall_trace()* (expmeas), *FrontPanel_eye_horizontal_histogram()* (expmeas), *FrontPanel_eye_regular()* (expmeas), *FrontPanel_eye_rise_trace()* (expmeas), *FrontPanel_eye_risefall_marker()* (expmeas), *FrontPanel_eye_topbase()* (expmeas), *FrontPanel_get_histogram_mean_stddev()* (expmeas), *FrontPanel_wave_1st_falling_edge_period()* (expmeas), *FrontPanel_wave_1st_rising_edge_period()* (expmeas), *FrontPanel_wave_1st_transition_fall_time()* (expmeas), *FrontPanel_wave_1st_transition_rise_time()* (expmeas), *FrontPanel_wave_datarate()* (expmeas), *FrontPanel_wave_negative_pulse_width()* (expmeas), *FrontPanel_wave_positive_pulse_width()* (expmeas), *FrontPanel_wave_topbase()* (expmeas)

Notes/Equations

The FrontPanel_pp_rms_jitter() function essentially takes histogram data, and returns peak to peak jitter by determining non-zero start and stop values and one standard deviation. The first element of the array represents peak-to-peak value and the second element represents one standard deviation.

FrontPanel_wave_1st_falling_edge_period()

Given time domain voltage waveform, the logic levels, and the threshold levels, returns first falling edge period

Syntax

y = FrontPanel_wave_1st_falling_edge_period(data,base,top,base_threshold_pct,top_threshold_pct)

Arguments

Name	Description	Default	Range	Type	Required
data	time domain voltage waveform	None	(-∞:∞)	real	Yes
base	logic level zero voltage level	None	(-∞:∞)	real	Yes
top	logic level one voltage level	None	(-∞:∞)	real	Yes
base_threshold_pct	low threshold point	None	(-∞:∞)	real	Yes
high_threshold_pct	high threshold point	None	(-∞:∞)	real	Yes

Examples

```
get_rise_time=FrontPanel_wave_1st_falling_edge_period(data,-1.0,1.0,20,80)
```

Defined in

$HPEESOF_DIR/expressions/ael/FrontPanel_fun.ael

See Also

eye_binning() (expmeas), *eye_density()* (expmeas), *FrontPanel_eye()* (expmeas), *FrontPanel_eye_2d_indepvar_maximum_inner()* (expmeas), *FrontPanel_eye_amplitude_histogram()* (expmeas), *FrontPanel_eye_crossings()* (expmeas), *FrontPanel_eye_delay()* (expmeas), *FrontPanel_eye_fall_trace()* (expmeas), *FrontPanel_eye_horizontal_histogram()* (expmeas), *FrontPanel_eye_regular()* (expmeas), *FrontPanel_eye_rise_trace()* (expmeas), *FrontPanel_eye_risefall_marker()* (expmeas), *FrontPanel_eye_topbase()* (expmeas), *FrontPanel_get_histogram_mean_stddev()* (expmeas), *FrontPanel_pp_rms_jitter()* (expmeas), *FrontPanel_wave_1st_rising_edge_period()* (expmeas), *FrontPanel_wave_1st_transition_fall_time()* (expmeas), *FrontPanel_wave_1st_transition_rise_time()* (expmeas), *FrontPanel_wave_datarate()* (expmeas), *FrontPanel_wave_negative_pulse_width()* (expmeas), *FrontPanel_wave_positive_pulse_width()* (expmeas), *FrontPanel_wave_topbase()* (expmeas)

Notes/Equations

Period is a measure of the time between the first detected edge of a waveform and the next occurrence of the same type of edge. The time between the edges of the waveform used for measurement is taken at the middle threshold crossings. The FrontPanel_wave_1st_falling_edge_period() function essentially takes time domain data and returns the first falling edge period.

FrontPanel_wave_1st_rising_edge_period()

Given time domain voltage waveform, the logic levels, and the threshold levels, returns first rising edge period

Syntax

y = FrontPanel_wave_1st_rising_edge_period(data,base,top,base_threshold_pct,top_threshold_pct)

Arguments

Name	Description	Default	Range	Type	Required
data	time domain voltage waveform	None	(-∞:∞)	real	Yes
base	logic level zero voltage level	None	(-∞:∞)	real	Yes
top	logic level one voltage level	None	(-∞:∞)	real	Yes
base_threshold_pct	low threshold point	None	(-∞:∞)	real	Yes
high_threshold_pct	high threshold point	None	(-∞:∞)	real	Yes

Examples

```
get_rising_period=FrontPanel_wave_1st_rising_edge_period(data,-1.0,1.0,20,80)
```

Defined in

$HPEESOF_DIR/expressions/ael/FrontPanel_fun.ael

See Also

eye_binning() (expmeas), *eye_density()* (expmeas), *FrontPanel_eye()* (expmeas), *FrontPanel_eye_2d_indepvar_maximum_inner()* (expmeas), *FrontPanel_eye_amplitude_histogram()* (expmeas), *FrontPanel_eye_crossings()* (expmeas), *FrontPanel_eye_delay()* (expmeas), *FrontPanel_eye_fall_trace()* (expmeas), *FrontPanel_eye_horizontal_histogram()* (expmeas), *FrontPanel_eye_regular()* (expmeas), *FrontPanel_eye_rise_trace()* (expmeas), *FrontPanel_eye_risefall_marker()* (expmeas), *FrontPanel_eye_topbase()* (expmeas), *FrontPanel_get_histogram_mean_stddev()* (expmeas), *FrontPanel_pp_rms_jitter()* (expmeas), *FrontPanel_wave_1st_falling_edge_period()* (expmeas), *FrontPanel_wave_1st_transition_fall_time()* (expmeas), *FrontPanel_wave_1st_transition_rise_time()* (expmeas), *FrontPanel_wave_datarate()* (expmeas), *FrontPanel_wave_negative_pulse_width()* (expmeas), *FrontPanel_wave_positive_pulse_width()* (expmeas), *FrontPanel_wave_topbase()* (expmeas)

Notes/Equations

Period is a measure of the time between the first detected edge of a waveform and the next occurrence of the same type of edge. The time between the edges of the waveform used for measurement is taken at the middle threshold crossings. The FrontPanel_wave_1st_rising_edge_period() function essentially takes time domain data

and returns the first rising edge period.

FrontPanel_wave_1st_transition_fall_time()

Given time domain voltage waveform, the logic levels, and the threshold levels, returns wave first negative transition fall time

Syntax

y = FrontPanel_wave_1st_transition_fall_time(data,base,top,base_threshold_pct,top_threshold_pct)

Arguments

Name	Description	Default	Range	Type	Required
data	time domain voltage waveform	None	(-∞:∞)	real	Yes
base	logic level zero voltage level	None	(-∞:∞)	real	Yes
top	logic level one voltage level	None	(-∞:∞)	real	Yes
base_threshold_pct	low threshold point	None	(-∞:∞)	real	Yes
high_threshold_pct	high threshold point	None	(-∞:∞)	real	Yes

Examples

```
get_rise_time=FrontPanel_wave_1st_transition_fall_time(data,-1.0,1.0,20,80)
```

Defined in

$HPEESOF_DIR/expressions/ael/FrontPanel_fun.ael

See Also

eye_binning() (expmeas), *eye_density()* (expmeas), *FrontPanel_eye()* (expmeas), *FrontPanel_eye_2d_indepvar_maximum_inner()* (expmeas), *FrontPanel_eye_amplitude_histogram()* (expmeas), *FrontPanel_eye_crossings()* (expmeas), *FrontPanel_eye_delay()* (expmeas), *FrontPanel_eye_fall_trace()* (expmeas), *FrontPanel_eye_horizontal_histogram()* (expmeas), *FrontPanel_eye_regular()* (expmeas), *FrontPanel_eye_rise_trace()* (expmeas), *FrontPanel_eye_risefall_marker()* (expmeas), *FrontPanel_eye_topbase()* (expmeas), *FrontPanel_get_histogram_mean_stddev()* (expmeas), *FrontPanel_pp_rms_jitter()* (expmeas), *FrontPanel_wave_1st_falling_edge_period()* (expmeas), *FrontPanel_wave_1st_rising_edge_period()* (expmeas), *FrontPanel_wave_1st_transition_rise_time()* (expmeas), *FrontPanel_wave_datarate()* (expmeas), *FrontPanel_wave_negative_pulse_width()* (expmeas), *FrontPanel_wave_positive_pulse_width()* (expmeas), *FrontPanel_wave_topbase()* (expmeas)

Notes/Equations

The FrontPanel_wave_1st_transition_rise_time() function essentially takes time domain data and measures the fall time of the first negative edge of the waveform. The fall time will not be measured until the falling edge completes the transition through the upper and lower threshold.

FrontPanel_wave_1st_transition_rise_time()

Given time domain voltage waveform, the logic levels, and the threshold levels, returns wave first transition rise time

Syntax

y = FrontPanel_wave_1st_transition_rise_time(data,base,top,base_threshold_pct,top_threshold_pct)

Arguments

Name	Description	Default	Range	Type	Required
data	time domain voltage waveform	None	(-∞:∞)	real	Yes
base	logic level zero voltage level	None	(-∞:∞)	real	Yes
top	logic level one voltage level	None	(-∞:∞)	real	Yes
base_threshold_pct	low threshold point	None	(-∞:∞)	real	Yes
high_threshold_pct	high threshold point	None	(-∞:∞)	real	Yes

Examples

```
get_rise_time=FrontPanel_wave_1st_transition_rise_time(data,-1.0,1.0,20,80)
```

Defined in

$HPEESOF_DIR/expressions/ael/FrontPanel_fun.ael

See Also

eye_binning() (expmeas), *eye_density()* (expmeas), *FrontPanel_eye()* (expmeas), *FrontPanel_eye_2d_indepvar_maximum_inner()* (expmeas), *FrontPanel_eye_amplitude_histogram()* (expmeas), *FrontPanel_eye_crossings()* (expmeas), *FrontPanel_eye_delay()* (expmeas), *FrontPanel_eye_fall_trace()* (expmeas), *FrontPanel_eye_horizontal_histogram()* (expmeas), *FrontPanel_eye_regular()* (expmeas), *FrontPanel_eye_rise_trace()* (expmeas), *FrontPanel_eye_risefall_marker()* (expmeas), *FrontPanel_eye_topbase()* (expmeas), *FrontPanel_get_histogram_mean_stddev()* (expmeas), *FrontPanel_pp_rms_jitter()* (expmeas), *FrontPanel_wave_1st_falling_edge_period()* (expmeas), *FrontPanel_wave_1st_rising_edge_period()* (expmeas), *FrontPanel_wave_1st_transition_fall_time()* (expmeas), *FrontPanel_wave_datarate()*

(expmeas), *FrontPanel_wave_negative_pulse_width()* (expmeas), *FrontPanel_wave_positive_pulse_width()* (expmeas), *FrontPanel_wave_topbase()* (expmeas)

Notes/Equations

The FrontPanel_wave_1st_transition_rise_time() function essentially takes time domain data and measures the rise time of the first positive edge of the waveform. The rise time will not be measured until the rising edge completes the transition through the lower and upper threshold.

FrontPanel_wave_datarate()

Estimate data rate from NRZ bitstream

Syntax

y = FrontPanel_wave_datarate(Vout_time,Data_Type)

Arguments

Name	Description	Default	Range	Type	Required
Vout_time	time domain voltage waveform	None	(-∞:∞)	Real	Yes

Examples

```
Find_datarate = FrontPanel_wave_datarate(vout,"NRZ")
```

Defined in

$HPEESOF_DIR/expressions/ael/FrontPanel_fun.ael

See Also

eye_binning() (expmeas), *eye_density()* (expmeas), *FrontPanel_eye()* (expmeas), *FrontPanel_eye_2d_indepvar_maximum_inner()* (expmeas), *FrontPanel_eye_amplitude_histogram()* (expmeas), *FrontPanel_eye_crossings()* (expmeas), *FrontPanel_eye_delay()* (expmeas), *FrontPanel_eye_fall_trace()* (expmeas), *FrontPanel_eye_horizontal_histogram()* (expmeas), *FrontPanel_eye_regular()* (expmeas), *FrontPanel_eye_rise_trace()* (expmeas), *FrontPanel_eye_risefall_marker()* (expmeas), *FrontPanel_eye_topbase()* (expmeas), *FrontPanel_get_histogram_mean_stddev()* (expmeas), *FrontPanel_pp_rms_jitter()* (expmeas), *FrontPanel_wave_1st_falling_edge_period()* (expmeas), *FrontPanel_wave_1st_rising_edge_period()* (expmeas), *FrontPanel_wave_1st_transition_fall_time()* (expmeas), *FrontPanel_wave_1st_transition_rise_time()* (expmeas), *FrontPanel_wave_negative_pulse_width()* (expmeas), *FrontPanel_wave_positive_pulse_width()* (expmeas), *FrontPanel_wave_topbase()* (expmeas)

Notes/Equations

The FrontPanel_wave_datarate() estimate the data rate of the NRZ bit stream.

FrontPanel_wave_negative_pulse_width()

Given time domain voltage waveform, the logic levels, and the threshold levels, returns negative pulse width of the first cycle

Syntax

y = FrontPanel_wave_negative_pulse_width(data,base,top,base_threshold_pct,top_threshold_pct)

Arguments

Name	Description	Default	Range	Type	Required
data	time domain voltage waveform	None	(-∞:∞)	real	Yes
base	logic level zero voltage level	None	(-∞:∞)	real	Yes
top	logic level one voltage level	None	(-∞:∞)	real	Yes
base_threshold_pct	low threshold point	None	(-∞:∞)	real	Yes
high_threshold_pct	high threshold point	None	(-∞:∞)	real	Yes

Examples

```
get_negative_pulse_width=FrontPanel_wave_negative_pulse_width(data,-
1.0,1.0,20,80)
```

Defined in

$HPEESOF_DIR/expressions/ael/FrontPanel_fun.ael

See Also

eye_binning() (expmeas), *eye_density()* (expmeas), *FrontPanel_eye()* (expmeas), *FrontPanel_eye_2d_indepvar_maximum_inner()* (expmeas), *FrontPanel_eye_amplitude_histogram()* (expmeas), *FrontPanel_eye_crossings()* (expmeas), *FrontPanel_eye_delay()* (expmeas), *FrontPanel_eye_fall_trace()* (expmeas), *FrontPanel_eye_horizontal_histogram()* (expmeas), *FrontPanel_eye_regular()* (expmeas), *FrontPanel_eye_rise_trace()* (expmeas), *FrontPanel_eye_risefall_marker()* (expmeas), *FrontPanel_eye_topbase()* (expmeas), *FrontPanel_get_histogram_mean_stddev()* (expmeas), *FrontPanel_pp_rms_jitter()* (expmeas), *FrontPanel_wave_1st_falling_edge_period()* (expmeas), *FrontPanel_wave_1st_rising_edge_period()* (expmeas), *FrontPanel_wave_1st_transition_fall_time()* (expmeas),

FrontPanel_wave_1st_transition_rise_time() (expmeas), *FrontPanel_wave_datarate()* (expmeas), *FrontPanel_wave_positive_pulse_width()* (expmeas), *FrontPanel_wave_topbase()* (expmeas)

Notes/Equations

Negative pulse width is defined as the time from the mid-threshold of the first falling edge to the mid-threshold of the next rising edge. The FrontPanel_wave_negative_pulse_width() function essentially takes time domain data and returns the first negative pulse width.

FrontPanel_wave_positive_pulse_width()

Given time domain voltage waveform, the logic levels, and the threshold levels, returns positive pulse width of the first cycle

Syntax

y = FrontPanel_wave_positive_pulse_width(data,base,top,base_threshold_pct,top_threshold_pct)

Arguments

Name	Description	Default	Range	Type	Required
data	time domain voltage waveform	None	(-∞:∞)	real	Yes
base	logic level zero voltage level	None	(-∞:∞)	real	Yes
top	logic level one voltage level	None	(-∞:∞)	real	Yes
base_threshold_pct	low threshold point	None	(-∞:∞)	real	Yes
high_threshold_pct	high threshold point	None	(-∞:∞)	real	Yes

Examples

```
get_negative_pulse_width=FrontPanel_wave_positive_pulse_width(data,-
1.0,1.0,20,80)
```

Defined in

$HPEESOF_DIR/expressions/ael/FrontPanel_fun.ael

See Also

eye_binning() (expmeas), *eye_density()* (expmeas), *FrontPanel_eye()* (expmeas), *FrontPanel_eye_2d_indepvar_maximum_inner()* (expmeas), *FrontPanel_eye_amplitude_histogram()* (expmeas), *FrontPanel_eye_crossings()* (expmeas), *FrontPanel_eye_delay()* (expmeas), *FrontPanel_eye_fall_trace()* (expmeas), *FrontPanel_eye_horizontal_histogram()* (expmeas), *FrontPanel_eye_regular()* (expmeas),

FrontPanel_eye_rise_trace() (expmeas), *FrontPanel_eye_risefall_marker()* (expmeas), *FrontPanel_eye_topbase()* (expmeas), *FrontPanel_get_histogram_mean_stddev()* (expmeas), *FrontPanel_pp_rms_jitter()* (expmeas), *FrontPanel_wave_1st_falling_edge_period()* (expmeas), *FrontPanel_wave_1st_rising_edge_period()* (expmeas), *FrontPanel_wave_1st_transition_fall_time()* (expmeas), *FrontPanel_wave_1st_transition_rise_time()* (expmeas), *FrontPanel_wave_datarate()* (expmeas), *FrontPanel_wave_negative_pulse_width()* (expmeas), *FrontPanel_wave_topbase()* (expmeas)

Notes/Equations

Positive pulse width is defined as the time from the mid-threshold of the first rising edge to the mid-threshold of the next falling edge. The FrontPanel_wave_positive_pulse_width() function essentially takes time domain data and returns the first positive pulse width.

FrontPanel_wave_topbase()

Given amplitude histogram of a time domain waveform, returns logic level one and zero statistics

Syntax

y = FrontPanel_wave_topbase(data,data_type)

Arguments

Name	Description	Default	Range	Type	Required
data	amplitude histogram data	None	(-∞:∞)	real	Yes
data_type	data type, only "NRZ" data is supported in current release	None	(-∞:∞)	real	Yes

Examples

```
get_level=FrontPanel_wave_topbase(amplitude_histogram,"NRZ")
top_data=get_level[0::3]
level_one_mean=top_data[0]
level_one_stddev=top_data[1]
level_one_start=top_data[2]
level_one_stop=top_data[3]
base_data=get_level[4::7]
level_zero_mean=base_data[0]
level_zero_stddev=base_data[1]
level_zero_start=base_data[2]
level_zero_stop=base_data[3]
```

Defined in

$HPEESOF_DIR/expressions/ael/FrontPanel_fun.ael

See Also

eye_binning() (expmeas), *eye_density()* (expmeas), *FrontPanel_eye()* (expmeas), *FrontPanel_eye_2d_indepvar_maximum_inner()* (expmeas), *FrontPanel_eye_amplitude_histogram()* (expmeas), *FrontPanel_eye_crossings()* (expmeas), *FrontPanel_eye_delay()* (expmeas), *FrontPanel_eye_fall_trace()* (expmeas), *FrontPanel_eye_horizontal_histogram()* (expmeas), *FrontPanel_eye_regular()* (expmeas), *FrontPanel_eye_rise_trace()* (expmeas), *FrontPanel_eye_risefall_marker()* (expmeas), *FrontPanel_eye_topbase()* (expmeas), *FrontPanel_get_histogram_mean_stddev()* (expmeas), *FrontPanel_pp_rms_jitter()* (expmeas), *FrontPanel_wave_1st_falling_edge_period()* (expmeas), *FrontPanel_wave_1st_rising_edge_period()* (expmeas), *FrontPanel_wave_1st_transition_fall_time()* (expmeas), *FrontPanel_wave_1st_transition_rise_time()* (expmeas), *FrontPanel_wave_datarate()* (expmeas), *FrontPanel_wave_negative_pulse_width()* (expmeas), *FrontPanel_wave_positive_pulse_width()* (expmeas)

Notes/Equations

The FrontPanel_wave_topbase() function essentially takes amplitude histogram data and returns logic level one and zero statistics as an array.

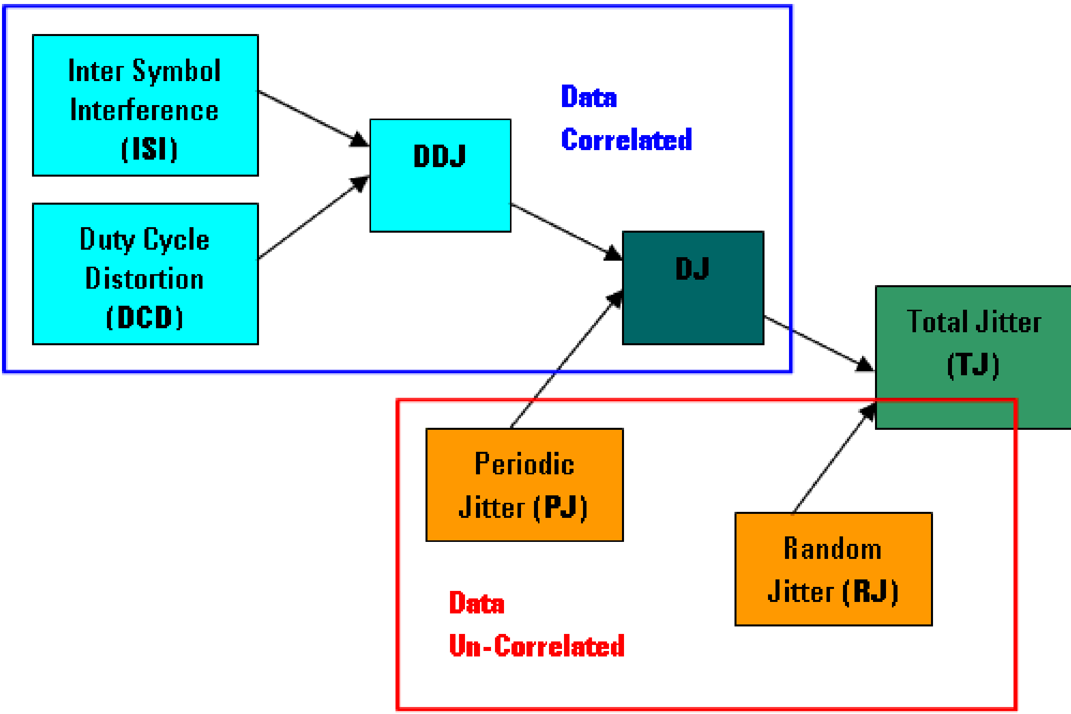
Jitter Analysis Functions

This section describes the jitter analysis concepts and functions in detail. These are not generalized functions, they are provided specifically to support jitter analysis.

- *bathtub()* (expmeas)
- *jitter separation()* (expmeas)

Working with Jitter Analysis Data

Jitter Analysis is used to decompose aggregate total jitter of serial data into the individual jitter components, random jitter (RJ), and deterministic jitter (DJ) as shown in Jitter Components. Jitter analysis leverages the techniques from the DCA-J, equivalent-time sampling, and the Infiniium DSO80000 real-time oscilloscopes.



Jitter Components

Definitions

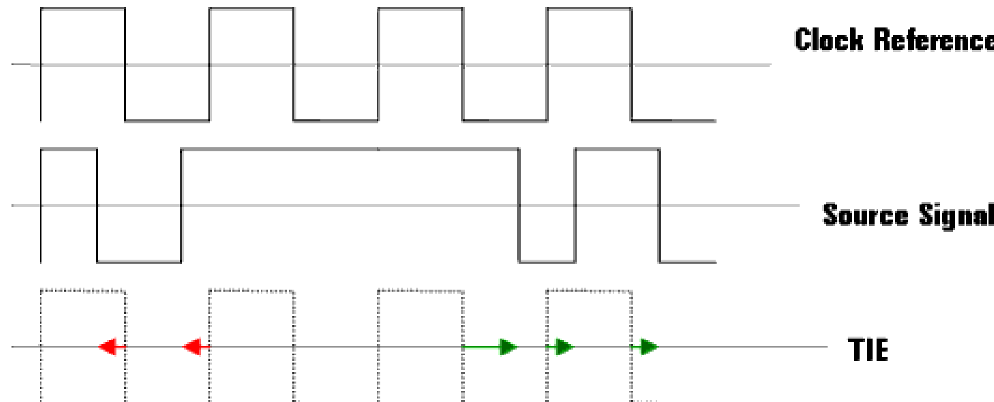
BER	Bit Error Rate
DCD	Duty Cycle Distortion. Derived from the Composite DDJ Histogram graph. It is the absolute value of the difference between the mean of the histogram of the rising edge positions and the mean of the histogram of the falling edge positions.
DDJ	Data Dependent Jitter. DDJ is the difference in the position of the earliest edge (rising or falling) and the latest (rising or falling) edge. If DCD = zero, DDJ = ISI. If ISI = zero, DDJ = DCD.
DDJ_{pp}	Data dependent jitter, peak-to-peak value of the jitter that is correlated to the data pattern. It is the difference in the position of the earliest edge (rising or falling) and the latest (rising or falling) edge. If DCD=0, DDJ is just ISI. If ISI=0, then DDJ is just DCD.
DJ	Deterministic Jitter. DJ is bounded by a finite magnitude. It can be broken into jitter which is correlated to the data sequence and jitter that occurs independent of data.
DJ(δδ)	Delta-Delta Deterministic Jitter of the bimodal equivalent model used to represent all aggregate deterministic jitter as defined in the dual-Dirac model for total jitter. Deterministic jitter is defined by the dual-Dirac jitter model as all those components of total jitter that do not fit a Gaussian probability density function. It is given by the time delay separation of the two delta functions.
ISI_{pp}	Inter-Symbol Interference peak-to-peak (p-p) range of the jitter that is correlated to rising edges or the jitter that is correlated to falling edges (whichever is greater). ISI is the largest of the difference between the earliest falling/rising and latest falling/rising edges, determined from measuring the average position of each bit in the pattern. When doing a jitter analysis if the rising or falling edge modes are selected, then only the specified edges are used in the calculation of ISIpp.
MinSeqDDJ	Minimum number of sequences needed for a successful DDJ separation.
MinNTIEs	Minimum number of TIEs that are needed if the corresponding number of Nbps is used.
MinNSeqs	Minimum number of sequences that can be used for given NTIEs.
MaxNSeqs	Maximum number of sequences that can be used for given MinSeqsDDJ.
Nbps	Number of bits per sequence
Nbpp	Number of bits per pattern
NTIEs	Number of TIEs
NumSeqs	Number of sequences
Nwpps	Number of whole patterns per sequence
PJ	Periodic Jitter. PJ represents all of the periodic jitter that is uncorrelated from the data pattern. There are 2 types:
PJ(δδ)	Periodic Jitter delta-delta, is the jittered magnitude required to make RJ PDFs match the Dual-Dirac model with the measured or simulated RJ and PJ.
PJ_{rms}	The root-mean-square value of the uncorrelated periodic jitter.
RJ	Random Jitter. RJ follows a Gaussian distribution and is represented by the rms value of the RJ distribution, RJ_{rms}. RJ is the baseline noise floor of the aliased power spectrum. Peaks (anything above a particular threshold) are identified and removed.
RJ_{rms}	Random Jitter follows a Gaussian distribution. RJ is the baseline noise floor of the aliased power spectrum. Anything above it is periodic jitter and is removed in calculating RJ.
TJ	Total Jitter. TJ is interpreted as total eye closure at a specified BER (10-12 default). If the closure threshold is the default and TJ=50ps, the likelihood that and edge will be 25ps late or 25ps early is 1 in a billion.
TJ_{pp}	The peak-to-peak value of the total jitter calculated at a specific bit error rate (BER). The BER level specified as one of the arguments during jitter analysis identifies the specified BER for which the TJ value was calculated. This TJpp value is calculated as an estimate of the true total jitter, as defined by the dual-Dirac jitter model.

Jitter Analysis Process

This section describes steps involved in jitter analysis.

Time Interval Error

The first step in the jitter analysis/decomposition process is finding the time interval error (TIE), which is the time difference between the serial data signal relative to a reference signal (usually a clock signal) as shown in Time Interval Error. TIE is then used in the jitter decomposition.



Time Interval Error

Sub-Sampled Decimation for DDJ Separation

After the TIE data has been calculated, the next step is DDJ separation. Jitter analysis can be done on periodic or arbitrary data. Currently, only periodic data is supported. In order to decompose jitter, the TIE calculated is associated with the specific bit in the source signal's logical bit sequence. The TIE data is decimated into sub-sampled TIE data, where the value in the sub-sampled sequence corresponds to a specific bit within the pattern. The number of original samples skipped during decimation is a function of the RJ bandwidth. The narrow-band mode maximizes the decimation ratio, and the wide-band minimizes the decimation. The next step is to perform a FFT on the sub-sampled data. The first value of each jitter spectrum (DC component) is the DDJ for the particular bit of the repeating pattern.

RJ PJ Separation

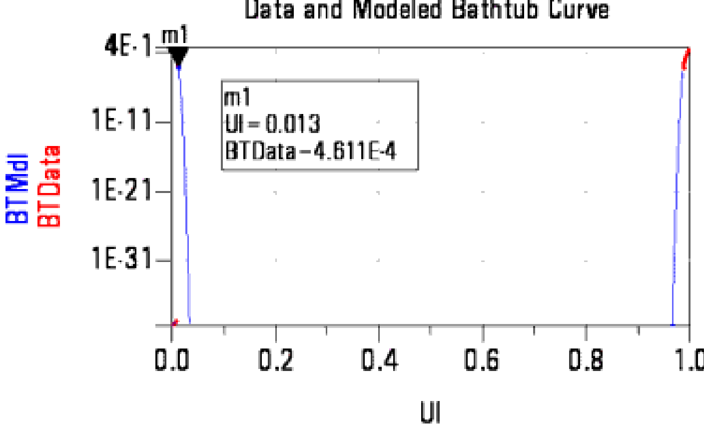
Once the DDJ component has been subtracted from TJ, the remaining jitter spectrum is comprised of RJ and PJ. The power spectrum density (PSD) of the RJ/PJ spectrum is calculated. All of the individual RJ/PJ spectrums are averaged together (as well as averaged with spectrums from previous sequences) to form the averaged PSD (APSD). All APSD's frequency components that have a value above a threshold are removed as PJ.

The remaining APSD are then combined to obtain RJrms. Refer to Reference 1 in References for more details.

Viewing Results

Jitter Analysis FrontPanel is part of the Data Display. For information on viewing Jitter Analysis simulation results, see "Jitter Analysis FrontPanel" in the *Data Display* documentation.

BER Bathtub Graph

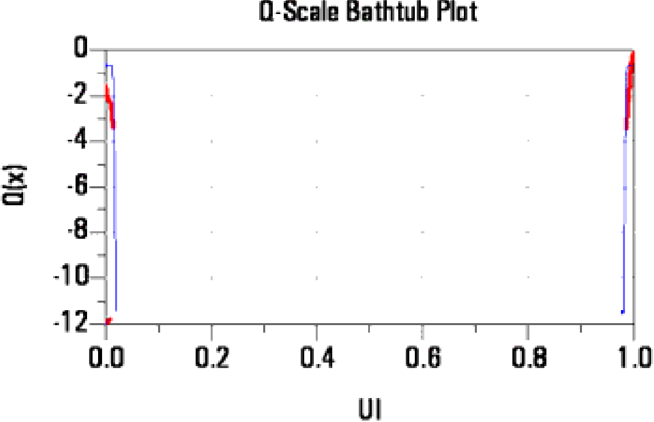


Bathtub

The BER Bathtub graph plots the sampling time (in UI) of a serial data signal on the X-axis versus bit error rate on the Y-axis. The trace in red represents BER values that are calculated directly from the TIE data, and the trace in blue represents BER values that are extrapolated using the calculated values of RJ and DJ.

The BER value for the TIE data is calculated by integrating the TJ Histogram. The extrapolated values are obtained from the Dual-Dirac model of the way RJ and DJ combine into a CDF. RJ is modeled as a Gaussian distribution with standard deviation RJ, and DJ is modeled as two Dirac-Delta functions, separated by distance DJ. This function is basically the integral of the function defined by Dual-Dirac PDF model, except that because this is intended to model BER curves, this CDF has been modified to peak at the transition density rather than 1.0, for a maximum BER equivalent to the transition density.

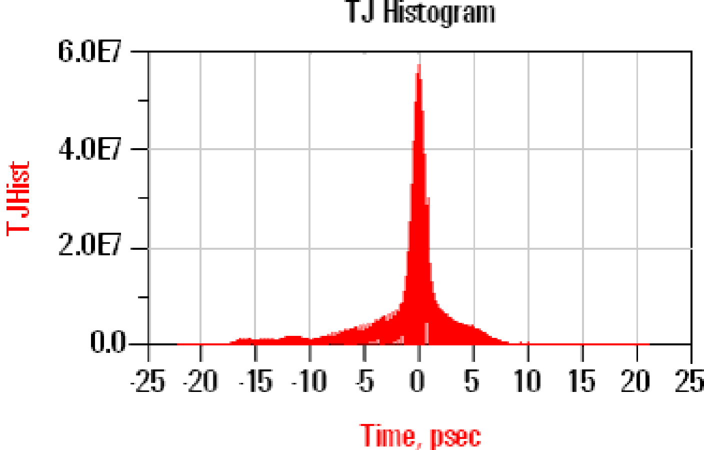
The Q of BER bathtub graph plots the sampling time (in UI) versus the Q of BER.



Q of BER Bathtub

Jitter Histograms

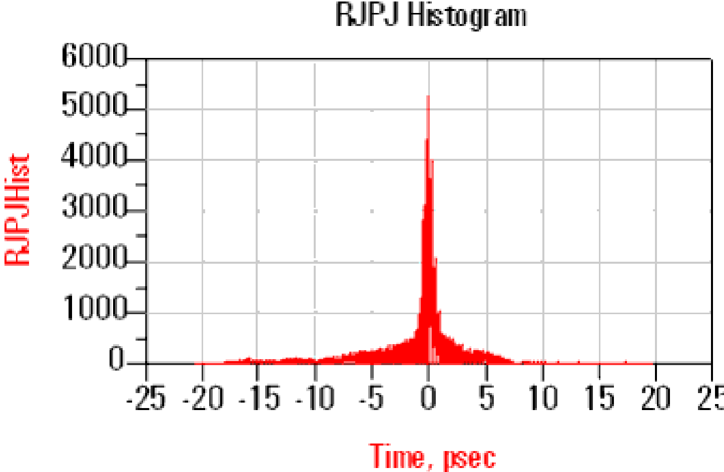
TJ Histogram



Total Jitter (TJ) Histogram

The Total Jitter (TJ) Histogram shows the combined Random Jitter (RJ), Periodic Jitter (PJ) and Data Dependent Jitter (DDJ) probability density functions. The TJ histogram is calculated by cross-correlating the RJ, PJ histogram with the DDJ histogram. It is the a histogram of all of the measured jitter, both correlated to the data pattern and uncorrelated to the data pattern, combined in a single histogram. The graph's horizontal axis indicates negative time for samples that occur earlier than expected and positive time for samples that occur later than expected.

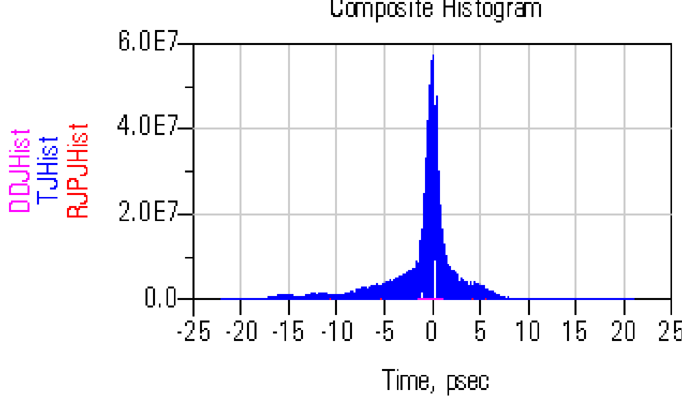
RJ PJ Histogram



Random Jitter, Periodic Jitter (RJ, PJ) Histogram

The RJ, PJ Jitter Histogram shows the histogram of all uncorrelated jitter. The graph's horizontal axis indicates negative time for samples that occur earlier than expected and positive time for samples that occur later than expected.

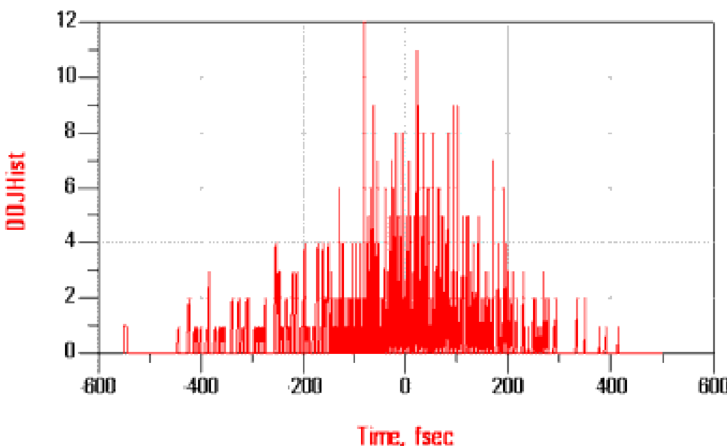
Composite TJ Histogram



Composite TJ Histogram

The Composite TJ Histogram shows separate graphs of Total Jitter (TJ), Data-Dependent Jitter (DDJ), and the combined histogram of uncorrelated Random Jitter (RJ) and uncorrelated Periodic Jitter (PJ). The graph's horizontal axis indicates negative time for samples that occur earlier than expected and positive time for samples that occur later than expected.

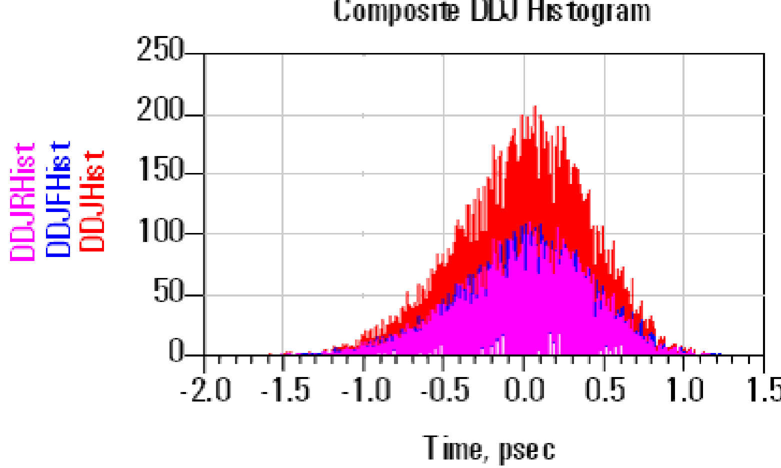
Data Dependent Jitter Histogram



Data Dependent Jitter (DDJ) Histogram

The Data Dependent Jitter (DDJ) histogram displays the jitter that is correlated to the data pattern. The graph's horizontal axis indicates negative time for samples that occur earlier than expected and positive time for samples that occur later than expected. For data-type source waveform signal, the mean of the histogram of DDJ from all edges is always equal to zero.

Composite DDJ Histogram

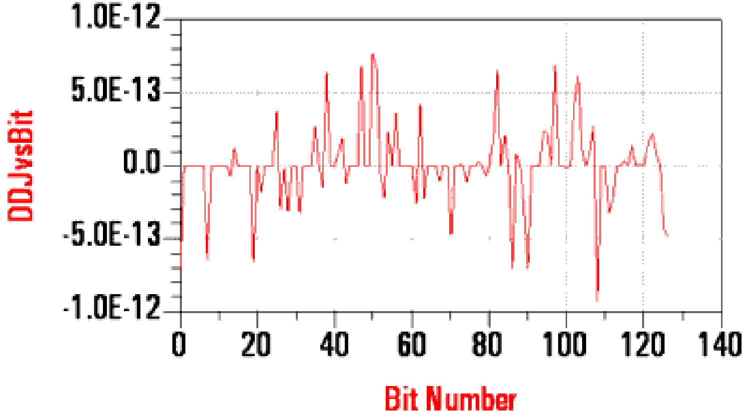


Composite Data Dependent Jitter (DDJ) Histogram

The Composite Data Dependent Jitter (DDJ) Histogram shows three histograms of correlated jitter based on data from all edges, rising edges, and falling edges. The peak-to-peak spread of the all-edges histogram represents the DDJ. The peak-to-peak spread of the rising-edges histogram or the falling-edges histogram, whichever is greater, represents Inter-Symbol Interference (ISI). The difference between the mean of the rising

edge positions and the mean of the falling edge positions represents the Duty Cycle Distortion (DCD). The graph's horizontal axis indicates negative time for samples that occur earlier than expected and positive time for samples that occur later than expected.

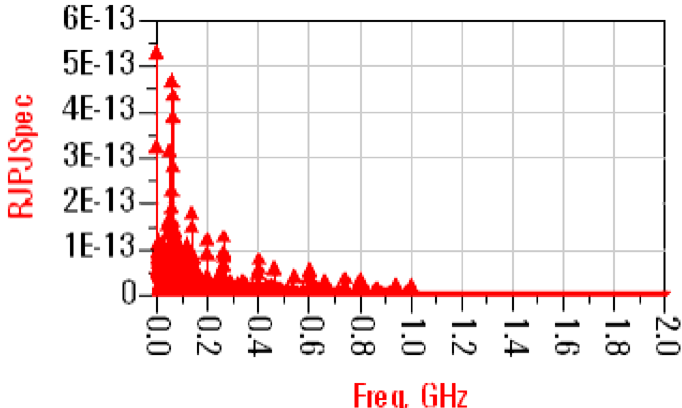
DDJ versus Bits Graph



Data Dependent Jitter (DDJ) versus Relative Bit Position

The graph of Data-Dependant Jitter (DDJ) versus Relative Bit Position shows relative bit position on the horizontal axis. The vertical axis indicates negative time for samples that occur earlier than expected and positive time for samples that occur later than expected. The bit numbers displayed on the horizontal axis are relative values only and may change each time the logical bit pattern of the source waveform is recalculated.

RJ PJ Spectrum



RJ, PJ Spectrum Graph

The RJ, PJ (Random Jitter, Periodic Jitter) Spectrum graph shows the discrete Fourier

transform of the combined RJ and PJ. The vertical axis represents the magnitude of each spectral jitter component and the horizontal axis identifies the frequency. The displayed magnitude spectrum is calculated independently for each sequenced waveform and then averaged with magnitude spectrums from previous sequences.
The frequency resolution of the RJ, PJ Spectrum is improved by increasing Nbps, sequence record length. However, increasing the Nbps can significantly affect calculation time. Overall calculation time can be improved by not displaying the RJ, PJ Spectrum graph (not using MeasType equal 4).

References

1. *Precision Jitter Analysis Using the Agilent 86100C DCA-J* - Agilent Literature Number 5989-1146EN
2. *Jitter Analysis: The dual-Dirac Model, RJ/DJ, and Q-Scale* - Agilent White Paper 5989-3206EN
3. *MJSQ - Methodologies for Jitter and Signal Quality Specification* - T11.2/Project 1316-DT
4. *Analyzing Jitter Using Agilent EZJIT Plus Software* - Application Note 1563
5. *Selecting RJ Bandwidth in EZJIT Plus Software* - Application Note 1577
6. *EZJIT and EZJIT Plus Jitter Analysis Software for Infiniium Serial Oscilloscopes* - Data Sheet 5989-0109EN
7. U *nderstanding Jitter and Wander Measurements and Standards* - 5988-6254EN
8. *Jitter Separation - 50 Mb/s to Over 40 Gb/s Using the Agilent 86100C Infiniium DCA-J,* whitepaper, Agilent Technologies, Inc. http://www.eeplace.com/dm/2802/tw/DCAjwhitepaper3.pdf

bathtub()

This measurement returns the data based and extrapolated bathtub curves and Q curves.

Syntax

BathTub = bathtub(vJitterSig, vRefClkSig, Nbpp, BitPeriod, BERLevel, DataType, Pattern, Nbps, NumSeq, RJBWMode, EdgeType)

Arguments

Name	Description	Default	Range	Type	Required
vJitterSig	time-domain based jittered signal	None	(-∞:∞)	Real	Yes
vRefClkSig	time-domain based reference clock signal or bit-period	None	(0:∞)	Real	No
Nbpp	number of bits per pattern	None	2 to 2^17	integer	Yes
BitPeriod	bit period	None	0 < BitPeriod < 1e-3	Real	Yes
BERLevel	BER level at which to measure TJ	1e-12	1e-40 < BERLevel < 1e-1	Real	No
DataType	data-type	1	1 (Periodic), 2(Random) †	Integer	No
Pattern	bit pattern vector or bit pattern file-name	None	0 and 1	Array, String	No
Nbps	number of bits per sequence	† †	[2:Inf]	Integer	No
NumSeq	number of sequences of data to be used in creating the bathtub	† † †	[1:Inf]	Integer	No
RJBWMode	RJ Bandwidth Mode	1	1(Narrow), 2(Wide)	Integer	No
EdgeType	Data Edge Type	3	1(Rising), 2(Falling), 3(Both)	Integer	No

Examples

```
The following example measures the bathtub of a jittered signal that uses a
PRBS10 source:
BathTub = bathtub(vJitSig, vRefClkSig, 1023, 50 ps, 1e-12, 1,, 5000, 95,,)
The returned value BathTub is a list of four measurements - data based BER
curve, extrapolated BER curve, data based Q BER curve and extrapolated Q BER
curve.
DataBath = BathTub[0]
MdlBath = BathTub[1]
DataQBath = BathTub[2]
MdlQBath = BathTub[3]
```

Where:

`vJitterSig` is the time-domain jittered signal. The argument jittered signal is the serial data (time versus amplitude) used in jitter separation. This signal is used in calculating the zero crossings, and the serial data is calculated from the zero crossings. If there are no zero crossings, then jitter separation will terminate with an error message. Currently, only periodic pattern serial data is supported, and the pattern must be repeatable. Arbitrary or random serial data is not supported.

`vRefClkSig` can be a reference clock signal (time versus amplitude) or the bit period. If the clock signal is given, the zero crossings are used to find the clock serial. This zero crossing is used along with the zero crossings of the jittered data in calculating the Time Interval Error (TIE). If the bit period is given, then the clock zero crossings are calculated from the bit period. If not given, the fourth argument bit period is used.

`Nbpp` is the pattern length and is required in order to automatically detect the pattern. At minimum this should be 2 (a clock signal) and a maximum of 2^{17}. Note that for larger pattern length the number of serial bits needed would be large and jitter separation would take a long time. See *Viewing Results* (expmeas) for more information.

`BitPeriod` = Bit period must be greater than 0 and less than 1e-3.

`BERLevel` is the level at which to calculate TJ, RJpp. This argument is optional and the default value is 1e-12. For example if the system is being designed for a BER of 1e-12, then doing a jitter separation at 1e-12 would calculate the TJ at 1e-12.

`Pattern` can be an array of 0s and 1s or the name of a file containing the pattern. An example of a pattern in array format is [1,0,0,0,0,0,0,0,1,0]. A pattern must have a 0 and a 1. In addition, the minimum pattern is a [0, 1] - a clock signal. If the pattern is given in a file, the pattern bits must be space separated and in a single line. If pattern is not given, the pattern is automatically detected from the signal bits. For each sequence of data, the pattern is detected and compared with the previous sequence patter or the given pattern. If the pattern matches, that particular sequence of data is used.
`Nbps` indicates the number of bits-per-acquisition or sequence. Nbps is optional and the default is 2 * Minimum # Whole Pattern per sequence * Nbpp = 2 * 64 * Nbpp. In most cases this default value would work, but in some cases this value might need to be set manually since this argument has a direct bearing on the number of serial bits needed for a valid jitter separation and the RJPJ separation method. See *Viewing Results* (expmeas) for more information.

`NumSeqs` can be used to control the amount of data to be used in jitter separation. The default value is (# TIE Points)/Nbps, and in most cases this would work. But if increased control is required over the number of bits to be used, this argument can be set to a different value.
`RJBWMode` Jitter analysis uses a spectral technique to separate RJ from PJ. In the RJ, PJ Spectrum the noise floor or baseline depicts RJ. The narrow spikes above RJ depict PJ (see RJ, PJ Spectrum graph). This separation works well for wide bandwidth RJ, having a uniform PSD across the entire jitter spectrum. But in some cases, this is not the norm and the PJ components appear much broader. In such cases PJ can be mis-represented as RJ and this affects the TJ (since TJ is a multiplier of RJ). Setting RJ bandwidth mode to **Wide** or **White** treats RJ as flat. See *References* (expmeas) for more information.

Defined in

$HPEESOF_DIR/expressions/ael/JitAnalysis.ael

See Also

jitter_separation() (expmeas)

Notes and Equations

1. Default value for Nbps = 2 * Minimum # Whole Pattern per sequence * Nbpp = 2 * 64 * Nbpp
   For Periodic data type if the number of whole pattern per sequence, Nwpps = Nbps / Ntpp, is less than 64, the method used for RJPJ Separation is random (which is different than the data type or pattern being random). For PRBS15 if Nbps=60000, Ntpp=16384, NTIEs=2621359 then Nwpps=3. If the number of sequences to use is less than minimum number of sequences needed for DDJ separation, MinSeqsDDJ = 100/Nwpps, then DDJ cannot be separated and the initial sequences are not used. For example if Nwpps=3, then MinSeqsDDJ =100/3 = 34. In this case, increase the number of TIE to (MinSeqsDDJ+1) * Nbps.
   Some calculations for the PRBS15 example are shown below:

Nbps = BitMultiplier * Nbpp
Nwpps = floor(Nbps/Ntpp)
MinSeqsDDJ = ceil(100/Nwpps)
MinNTIEs = (MinSeqsDDJ + 1) * Nbps
MinNSeqs = floor(NTIEs/Nbps)
MaxNSeqs = MinNTIEs/Nbps
Where:
MinNTIEs is the minimum number of TIEs that are needed if the corresponding number of Nbps is used.
MinSeqsDDJ is the minimum number of sequences that are needed for a successful DDJ separation.
MinNSeqs is the minimum number of sequences that can be used for given NTIEs.
MaxNSeqs is the maximum number of sequences that can be used for given MinSeqsDDJ.
So in the above example, if Nbps = 13 * Nbpp = 425971, then Nwpps = 25.
For this Nbps, the minimum number of sequences that can be used for a valid DDJ separation is 4. This is less than the number of sequences, 6 that is possible with NTIEs= 2621359. So a valid DDJ separation can be done. In the table below, for Nwpps ≤63 the RJPJ separation used is random. For Nwpps ≥64, the RJPJ separation used is periodic.

BitMultiplier	Nbps	Nwpps	MinNTIEs	MinSeqsDDJ	MaxNSeqs	MinNSeqs
1	32767	1	3309467	100	101	79
7	229369	13	2064321	8	9	11
13	425971	25	2129855	4	5	6
19	622573	37	2490292	3	4	4
25	819175	49	3276700	3	4	3
31	1015777	61	3047331	2	3	2
32	1048544	63	3145632	2	3	2
34	1114078	67	3342234	2	3	2
36	1179612	71	3538836	2	3	2
38	1245146	75	3735438	2	3	2
40	1310680	79	3932040	2	3	1

In addition the frequency resolution of the RJ, PJ Spectrum is improved by increasing Nbps. But increasing Nbps can significantly increase the RJ, PJ spectrum calculation time.

2. NumSeqs default value = (# TIE Points)/Nbps.
   In order to get a valid jitter separation the number of data points should be at least have 32 complete data patterns, otherwise jitter separation would terminate with an error. With number of bits between 32 and 128, jitter separation will be performed, but the results are questionable. For accurate results, use at least 128 patterns. Since the separation algorithm is a statistical procedure, the results would correlate better with more patterns.

PJ is separated from RJ by inspecting the spectral content of jitter. The robustness of this separation methodology depends on having sufficient frequency resolution in the jitter spectrum. When the frequency resolution drops below a threshold it could start to cause a reduction in RJ/PJ separation accuracy. This happens when the number of complete data patterns in the serial data falls below 128 patterns. When this happens, examine the RJ, PJ spectrum. If the spectrum is comprised primarily of random noise with very few tall PJ spikes, then the questionable results are accurate. If there are a large number of PJ spikes then the jitter measurement should be repeated several times while varying the sequence

record length. If the RJ and PJ results change significantly with record length, then the results are correct.

jitter_separation()

This function does a jitter analysis and separates the jitter components.

Syntax

JitRes = jitter_separation(vJitterSig, vRefClkSig, Nbpp, BitPeriod, BERLevel, DataType, Pattern, Nbps, NumSeq, RJBWMode, EdgeType, InterpType, MeasType)

Arguments

Name	Description	Default	Range	Type	Required
vJitterSig	time-domain based jittered signal	None	(-∞:∞)	Real	Yes
vRefClk	time-domain based reference clock signal or bit period	None	(0:∞)	Real	No
Nbpp	number of bits per pattern	None	2 to 2^17	integer	Yes
BitPeriod	bit period	None	0 < BitPeriod < 1e-3	Real	Yes
BERLevel	BER level at which to measure TJ	1e-12	1e-40 < BERLevel < 1e-1	Real	No
DataType	data-type	1	1 (Periodic), 2(Random) †	Integer	No
Pattern	bit pattern vector or bit pattern file-name	None	0 and 1	Array, String	No
Nbps	number of bits per sequence	† †	[2:Inf]	Integer	No
NumSeq	number of sequences of data to be used in jitter analysis	† † †	[1:Inf]	Integer	No
RJBWMode	RJ Bandwidth Mode	1	1(Narrow or Pink), 2(Wide or White)	Integer	No
EdgeType	Data Edge Type	3	1(Rising), 2(Falling), 3(Both)	Integer	No
InterType	Interpolation type for holes in TIE for jitter spectrum	1	1(None), 2(Linear)	Integer	No
MeasType	Specifies the jitter components and graphs to calculate.	3	† † † †	Integer	No

† Note that Random data type is not supported in this release.
† † The default for Nbps is 2 * minimum number of whole patterns per sequence * Nbpp.
† † † NumSeq defaults is calculated as number of TIE points/Nbps.
† † † † MeasType can be one of the following:
1 Calculate TJpp,RJrms,DJdd
2 Returns TJpp,RJrms,DJdd,PJdd,PJrms,ISIpp,DCD,DDJpp.
3 In addition to measurements in type 2, calculate TJ,RJPJ,DDJ,DDJR,DDJF Histograms & Bathtub Plot & DDJ vs Bit
4 In addition to measurements in type 3, calculate RJPJ spectrum.

Examples

```
The following example does a jitter separation of a jittered signal that uses a
PRBS10 source:
JitRes = jitter_separation(vJitSig, vRefClk, 1023, 50 ps, 1e-12, 1,, 5000,
95,,,4)
The returned value JitRes is a list of 18 different measurements.
TJpp = JitRes[0]
RJrms = JitRes[1]
```

```
DJdd = JitRes[2]
PJdd = JitRes[3]
PJrms = JitRes[4]
ISIpp = JitRes[5]
DCD = JitRes[6]
DDJpp = JitRes[7]
TJHist = JitRes[8]
RJPJHist = JitRes[9]
DDJHist = JitRes[10]
DDJFHist = JitRes[11]
DDJRHist = JitRes[12]
DataBath = JitRes[13]
MdlBath = JitRes[14]
DataQBath = JitRes[15]
MdlQBath = JitRes[16]
DDJvsBit = JitRes[17]
RJPJSpec = JitRes[18]
```

Where:
`vJitterSig` is the time-domain jittered signal. The argument jittered signal is the serial data (time versus amplitude) used in jitter separation. This signal is used in calculating the zero crossings, and the serial data is calculated from the zero crossings. If there are no zero crossings, then jitter separation will terminate with an error message. Currently, only periodic pattern serial data is supported, and the pattern must be repeatable. Arbitrary or random serial data is not supported.
`vRefClkSig` can be a reference clock signal (time versus amplitude) or the bit period. If the clock signal is given, the zero crossings are used to find the clock serial. This zero crossing is used along with the zero crossings of the jittered data in calculating the Time Interval Error (TIE). If the bit period is given, then the clock zero crossings are calculated from the bit period. If not given, the fourth argument bit period is used.
`Nbpp` is the pattern length and is required in order to automatically detect the pattern. At minimum this should be 2 (a clock signal) and a maximum of 2^{17}. Note that for larger pattern length the number of serial bits needed would be large and jitter separation would take a long time. See *Viewing Results* (expmeas) for more information.

`BitPeriod` = Bit period must be greater than 0 and less than 1e-3.
`BERLevel` is the level at which to calculate TJ, RJpp. This argument is optional and the default value is 1e-12. For example if the system is being designed for a BER of 1e-12, then doing a jitter separation at 1e-12 would calculate the TJ at 1e-12.
`Pattern` can be an array of 0s and 1s or the name of a file containing the pattern. An example of a pattern in array format is [1,0,0,0,0,0,0,0,1,0]. A pattern must have a 0 and a 1. In addition, the minimum pattern is a [0, 1] - a clock signal. If the pattern is given in a file, the pattern bits must be space separated and in a single line. If pattern is not given, the pattern is automatically detected from the signal bits. For each sequence of data, the pattern is detected and compared with the previous sequence patter or the given pattern. If the pattern matches, that particular sequence of data is used.
`Nbps` indicates the number of bits-per-acquisition or sequence. Nbps is optional and the default is 2 * Minimum # Whole Pattern per sequence * Nbpp = 2 * 64 * Nbpp. In most cases this default value would work, but in some cases this value might need to be set manually since this argument has a direct bearing on the number of serial bits needed for a valid jitter separation and the RJPJ separation method. See *Viewing Results* (expmeas) for more information.

`NumSeqs` can be used to control the amount of data to be used in jitter separation. The default value is (# TIE Points)/Nbps, and in most cases this would work. But if increased control is required over the number of bits to be used, this argument can be set to a different value.
`RJBWMode` Jitter analysis uses a spectral technique to separate RJ from PJ. In the RJ, PJ Spectrum the noise floor or baseline depicts RJ. The narrow spikes above RJ depict PJ (see RJ, PJ Spectrum graph). This separation works well for wide bandwidth RJ, having a uniform PSD across the entire jitter spectrum. But in some cases, this is not the norm and the PJ components appear much broader. In such cases PJ can be mis-represented as RJ and this affects the TJ (since TJ is a multiplier of RJ). Setting RJ bandwidth mode to **Wide** or **White** treats RJ as flat. See *References* (expmeas) for more information.

`MeasType` specifies the jitter components and graphs to be calculated and displayed. There are four values that can be used for MeasType:

- 1 (RJ DJ TJ): Calculate TJpp, RJrms, DJdd.
- 2 (Add PJ DDJ): TJpp, RJrms, DJdd, PJdd, PJrms, ISIpp, DCD, DDJpp
- 3 (Add Histograms, Bathtub, DDJ versus Bits): Measurements in MeasType=2 + TJ, RJPJ, DDJ, DDJR, DDJF Histograms, Bathtub graphs, and DDJ versus Bits graph.
- 4 (Add full RJPJ spectrum): Measurements in mode 3 + RJPJ spectrum.

Note
Type 4 measurements can take an exceptionally long time to complete.

`InterType` For clock-type signals, the DFT is calculated from the uniformly spaced RJ, PJ time record, where each value in the RJ, PJ time record corresponds to a voltage transition in the clock-type waveform. For NRZ data-type signals, the RJ, PJ time record is not comprised of uniformly spaced jitter values. For these signals, the RJ, PJ time record contains "holes" caused by consecutive logical ones or zeros. The lack of information about the jitter at times corresponding to these holes makes it impossible to determine the true RJ, PJ spectrum.
There are two options for displaying the RJ, PJ spectrum. If `InterType` , the Data TIE Interpolation mode is set to none, then the spectrum is calculated as if the holes were all set to a value of zero. In this case, the resulting spectrum appears to be modulated (convolved in the frequency domain) by the data pattern. If `InterpType` is set to linear, then the spectrum is calculated as if the unknown values (holes) were determined using linear interpolation. In this second case, the resulting spectrum is calculated by filtering out the higher possible spectral components with a time-variant low-pass filter.

Defined in

$HPEESOF_DIR/expressions/ael/JitAnalysis.ael

See Also

bathtub() (expmeas)

Notes and Equations

1. This function requires the *ads_si_verification* license.

> **Caution**
> Due to memory limitations, large datasets created for performing jitter analysis can cause instabilities in ADS. This can result in the Data Display window crashing without saving the DDS file. To avoid losing any setup information, save the DDS file before performing a jitter analysis.

2. Default value for Nbps = 2 * Minimum # Whole Pattern per sequence * Nbpp = 2 * 64 * Nbpp
For Periodic data type if the number of whole pattern per sequence, Nwpps = Nbps / Ntpp, is less than 64, the method used for RJPJ Separation is random (which is different than the data type or pattern being random). For PRBS15 if Nbps=60000, Ntpp=16384, NTIEs=2621359 then Nwpps=3. If the number of sequences to use is less than minimum number of sequences needed for DDJ separation, MinSeqsDDJ = 100/Nwpps, then DDJ cannot be separated and the initial sequences are not used. For example if Nwpps=3, then MinSeqsDDJ =100/3 = 34. In this case, increase the number of TIE to (MinSeqsDDJ+1) * Nbps.
Some calculations for the PRBS15 example are shown below:
Nbps = BitMultiplier * Nbpp
Nwpps = floor(Nbps/Ntpp)
MinSeqsDDJ = ceil(100/Nwpps)
MinNTIEs = (MinSeqsDDJ + 1) * Nbps
MinNSeqs = floor(NTIEs/Nbps)
MaxNSeqs = MinNTIEs/Nbps
Where:
MinNTIEs is the minimum number of TIEs that are needed if the corresponding number of Nbps is used.
MinSeqsDDJ is the minimum number of sequences that are needed for a successful DDJ separation.
MinNSeqs is the minimum number of sequences that can be used for given NTIEs.
MaxNSeqs is the maximum number of sequences that can be used for given MinSeqsDDJ.
So in the above example, if Nbps = 13 * Nbpp = 425971, then Nwpps = 25.
For this Nbps, the minimum number of sequences that can be used for a valid DDJ separation is 4. This is less than the number of sequences, 6 that is possible with NTIEs= 2621359. So a valid DDJ separation can be done. In the table below, for Nwpps 63 the RJPJ separation used is random. For Nwpps 64, the RJPJ separation used is periodic.

BitMultiplier	Nbps	Nwpps	MinNTIEs	MinSeqsDDJ	MaxNSeqs	MinNSeqs
1	32767	1	3309467	100	101	79
7	229369	13	2064321	8	9	11
13	425971	25	2129855	4	5	6
19	622573	37	2490292	3	4	4
25	819175	49	3276700	3	4	3
31	1015777	61	3047331	2	3	2
32	1048544	63	3145632	2	3	2
34	1114078	67	3342234	2	3	2
36	1179612	71	3538836	2	3	2
38	1245146	75	3735438	2	3	2
40	1310680	79	3932040	2	3	1

In addition the frequency resolution of the RJ, PJ Spectrum is improved by increasing

Nbps. But increasing Nbps can significantly increase the RJ, PJ spectrum calculation time.

3. NumSeqs default value = (# TIE Points)/Nbps.
   In order to get a valid jitter separation the number of data points should be at least have 32 complete data patterns, otherwise jitter separation would terminate with an error. With number of bits between 32 and 128, jitter separation will be performed, but the results are questionable. For accurate results, use at least 128 patterns. Since the separation algorithm is a statistical procedure, the results would correlate better with more patterns.
4. PJ is separated from RJ by inspecting the spectral content of jitter. The robustness of this separation methodology depends on having sufficient frequency resolution in the jitter spectrum. When the frequency resolution drops below a threshold it could start to cause a reduction in RJ/PJ separation accuracy. This happens when the number of complete data patterns in the serial data falls below 128 patterns. When this happens, examine the RJ, PJ spectrum. If the spectrum is comprised primarily of random noise with very few tall PJ spikes, then the questionable results are accurate. If there are a large number of PJ spikes then the jitter measurement should be repeated several times while varying the sequence record length. If the RJ and PJ results change significantly with record length, then the results are correct.
5. Jitter separation algorithms are sensitive to input parameters in the following ways:
   - Accurate Bit period is essential in determining the Time Interval Errors. Linear interpolation is used to determine the threshold crossing, and linear regression is used to determine the bit period/frequency and phase of the input jittered signal. The Agilent Infinium oscilloscopes use sinc() interpolation to detect threshold crossings which is much more accurate than the interpolation method used in jitter_separation(). In some cases, with the same input signal, results from the oscilloscope and from jitter_separation() in ADS may be different.
   - In general it is better to use a longer signal sequence per acquisition (Nbps) to increase confidence in the separation results. Longer sequences help to acquire more information about the low-frequency components of jitter.
   - In some cases, the jitter separation function may produce different results for the same input signal when treating it as a Periodic Sequence compared to treating it as an Arbitrary Sequence. One of the major contributors to any such difference is the length of each sequence. When the sequence is longer, the results are more similar.

FrontPanel S-Parameter TDR Functions

This section describes the S-Parameter TDR FrontPanel and associated functions in detail. The majority of these functions are not generalized functions; they are provided specifically to support the S-Parameter TDR FrontPanel utility available from the Data Display. Any function with the prefix "FrontPanel_TDR" requires a special data structure for input parameters.

- *FrontPanel_TDRExtrapolate()* (expmeas)
- *FrontPanel_TDREye()* (expmeas)
- *FrontPanel_TDRFreqMode()* (expmeas)
- *FrontPanel_TDRFreqScale()* (expmeas)
- *FrontPanel_TDRFreqSweep()* (expmeas)
- *FrontPanel_TDRGate()* (expmeas)
- *FrontPanel_TDRIFW()* (expmeas)
- *FrontPanel_TDRInversePeeling()* (expmeas)
- *FrontPanel_TDRPeeling()* (expmeas)
- *FrontPanel_TDRPortExt()* (expmeas)
- *FrontPanel_TDRPortMap()* (expmeas)
- *FrontPanel_TDRSmooth()* (expmeas)
- *FrontPanel_TDRTimeScale()* (expmeas)
- *FrontPanel_TDRTimeSweep()* (expmeas)
- *FrontPanel_TDRWindow()* (expmeas)
- *tdr_inverse_peeling()* (expmeas)
- *tdr_peeling()* (expmeas)

Note
Any function with the prefix "FrontPanel_TDR" requires a special data structure for input parameters and is used exclusively by the S-Parameter TDR FrontPanel.

FrontPanel_TDRExtrapolate()

Extrapolates frequency data with the ability to add a DC and low frequency points

Syntax

y = FrontPanel_TDRExtrapolate(Data, Flag, DCFlag, OptFlag, DCValue, Step, PolyList)

Arguments

Name	Description	Default	Range	Type	Required
Data	frequency transfer function †	None	(-∞:∞)	Complex Array	Yes
Flag	boolean flag if set FALSE will return the original waveform	None	[0:1]	Real	Yes
DCFlag	boolean flag if set TRUE a DC point will be added before Extrapolate	None	[0:1]	Real	Yes
OptFlag	boolean flag if set TRUE an optimizer is run on extrapolated data	None	[0:1]	Real	Yes
DCValue	array of values at DC (one value for each S parameter)	None	(-∞:∞)	Real	Yes
Step	frequency step after Extrapolate	None	(0:∞)	Real	Yes
PolyList	list data type representing parameters for extrapolation - list(even order, even num points, odd order, odd num points)	None	(1:∞)	List	Yes

† The multi-dimensional S parameter array can be created with FP_TDRFreq1Dto3D(S).

Examples

```
ExtrapolateOut = FrontPanel_TDRExtrapolate(FP_TDRFreq1Dto3D(S), TRUE, TRUE,
TRUE, [0,1,1,0], 1e9, list(3,5,3,5))
```

Note
Any function with the prefix "FrontPanel_TDR" requires a special data structure for input parameters and is used exclusively by the S-Parameter TDR FrontPanel.

FrontPanel_TDREye()

Returns the eye waveform produced by running a randomly generated NRZ trace through the system described by the input S parameter

Syntax

y = FrontPanel_TDREye(Data, NumBits, DataRate, BitSpan, Vmax)

Arguments

Name	Description	Default	Range	Type	Required
Data	frequency transfer function †	None	(-∞:∞)	Complex Array	Yes
NumBits	number of bits in NRZ waveform	None	[1:∞)	Real	Yes
DataRate	data rate of NRZ wafeform	None	[1:∞)	Real	Yes
BitSpan	number of samples per bit of NRZ waveform	None	[1:∞)	Real	Yes
Vmax	maximum voltage for NRZ waveform	None	(-∞:∞)	Real	Yes
Vmin	minimum voltage for NRZ waveform	None	(-∞:∞)	Real	Yes

† The multi-dimensional S parameter array can be created with FP_TDRFreq1Dto3D(S).

Examples

```
EyeOut = FrontPanel_TDREye(S11, 101, 1 GHz, 5, 1)
```

Note
Any function with the prefix "FrontPanel_TDR" requires a special data structure for input parameters and is used exclusively by the S-Parameter TDR FrontPanel.

FrontPanel_TDRFreqMode()

Returns single-ended or mixed-mode S parameters for 4-port datasets

Syntax

y = FrontPanel_TDRFreqMode(Data, Mode)

Arguments

Name	Description	Default	Range	Type	Required
Data	frequency transfer function †	None	(-∞:∞)	Complex Array	Yes
Mode	type of frequency mode † †	None	[0:4]	Real	Yes

† The multi-dimensional S parameter array can be created with FP_TDRFreq1Dto3D(S).
† † The different type of modes are:
Mode = 0 Single-Ended
Mode = 1 Differential
Mode = 2 Common
Mode = 3 Differential-Common
Mode = 4 Common-Differential
Any Mode > 0 is only valid for 4-port S parameters

Examples

```
FreqModeOut = FrontPanel_TDRFreqMode(FP_TDRFreq1Dto3D(S), 0)
```

Note
Any function with the prefix "FrontPanel_TDR" requires a special data structure for input parameters and is used exclusively by the S-Parameter TDR FrontPanel.

FrontPanel_TDRFreqScale()

Scales raw frequency data

Syntax

y = FrontPanel_TDRFreqScale(Data, Scale)

Arguments

Name	Description	Default	Range	Type	Required
Data	frequency transfer function †	None	(-∞:∞)	Complex Array	Yes
Scale	type of scaling † †	None	[0:4]	Real	Yes

† The multi-dimensional S parameter array can be created with FP_TDRFreq1Dto3D(S).
† † The available scale types are:
Scale = 0 dB
Scale = 1 Magnitude
Scale = 2 Phase
Scale = 3 Real

Scale = 4 Imaginary

Examples

```
FreqScaleOut = FrontPanel_TDRFreqScale(FP_TDRFreq1Dto3D(S), 1)
```

Note
Any function with the prefix "FrontPanel_TDR" requires a special data structure for input parameters and is used exclusively by the S-Parameter TDR FrontPanel.

FrontPanel_TDRFreqSweep()

Performs the time-to-frequency transform across multiple dimensions

Syntax

y = FrontPanel_TDRFreqSweep(Data, Flag, Mode, Start, Stop, NumPoints, FArray)

Arguments

Name	Description	Default	Range	Type	Required
Data	multi-dimensional time waveform †	None	(-∞:∞)	Complex Array	Yes
Flag	boolean flag if set FALSE default sweep values are used	None	[0:1]	Real	Yes
Mode	type of time mode (0 = Lowpass Impulse, 1 = Lowpass Step, 2 = Bandpass Impulse, 3 = Bandpass Step)	None	[0:2]	Real	Yes
Start	start frequency	None	[0:∞)	Real	Yes
Stop	stop frequency	None	[0:∞)	Real	Yes
NumPoints	number of frequency points	None	[1:∞)	Real	Yes
FArray	frequency sweep of the initial S parameter waveform	None	(-∞:∞)	Real Array	Yes

† The input data is a multi-dimensional temporal array representing the time transform of an S parameter array.
The multi-dimensional S parameter array can be created with FP_TDRFreq1Dto3D(S).

Examples

```
FreqSweepOut = FrontPanel_TDRFreqSweep(ts(FP_TDRFreq1Dto3D(S)), TRUE, 0, 1 GHz,
101, 0)
```

Note
Any function with the prefix "FrontPanel_TDR" requires a special data structure for input parameters and is used exclusively by the S-Parameter TDR FrontPanel.

FrontPanel_TDRGate()

Returns a gated time waveform

Syntax

y = FrontPanel_TDRGate(Data, Flag, Start, Length, OutFlag, ResponseType, sm, sn)

Arguments

Name	Description	Default	Range	Type	Required
Data	multi-dimensional time waveform †	None	(-∞:∞)	Real Array	Yes
Flag	boolean flag if set FALSE will return the original waveform	None	[0,1]	Real	Yes
Start	gate start time for each time waveform	None	(-∞:∞)	Real Array	Yes
Length	gate length time for each time waveform	None	(0:∞)	Real Array	Yes
OutFlag	boolean flag if set TRUE will zero values inside the gate	None	[0,1]	Real	Yes
ResponseType	gate response type: (0) standard gate for impulse modes or (1) flat, (2) linear, or (3) hyperbolic for step modes	None	[0:3]	Real	Yes

† The input data is a multi-dimensional temporal array representing the time transform of an S parameter array.
The multi-dimensional S parameter array can be created with FP_TDRFreq1Dto3D(S).

Examples

```
GateOut = FrontPanel_TDRGate(ts(FP_TDRFreq1Dto3D(S)), GateMask, TRUE, FALSE)
```

Note
Any function with the prefix "FrontPanel_TDR" requires a special data structure for input parameters and is used exclusively by the S-Parameter TDR FrontPanel.

FrontPanel_TDRIFW()

Converts an IFW dataset to S parameters

Syntax

y = FrontPanel_TDRIFW(DatasetName)

Arguments

Name	Description	Default	Range	Type	Required
DatasetName	name of IFW dataset	None	None	String	Yes

Examples

```
IFWOut = FrontPanel_TDRIFW("my_sym.ds")
```

FrontPanel_TDRInversePeeling()

Note
Any function with the prefix "FrontPanel_TDR" requires a special data structure for input parameters and is used exclusively by the S-Parameter TDR FrontPanel.

FrontPanel_TDRInversePeeling() is a wrapper function for *tdr_inverse_peeling()* (expmeas).

Note
Any function with the prefix "FrontPanel_TDR" requires a special data structure for input parameters and is used exclusively by the S-Parameter TDR FrontPanel.

FrontPanel_TDRPeeling()

Applies the peeling algorithm to all ports of an S parameter array

Syntax

y = FrontPanel_TDRPeeling(Data, Flag, Step)

Arguments

Name	Description	Default	Range	Type	Required
Data	frequency transfer function †	None	(-∞:∞)	Complex Array	Yes
Flag	boolean flag if set FALSE will return the original waveform	None	[0:1]	Real	Yes
Step	voltage step value	None	(0:∞)	Real	Yes

† The multi-dimensional S parameter array can be created with FP_TDRFreq1Dto3D(S).

Examples

```
PeelingOut = FrontPanel_TDRPeeling(FP_TDRFreq1Dto3D(S), TRUE, 1)
```

Note
Any function with the prefix "FrontPanel_TDR" requires a special data structure for input parameters and is used exclusively by the S-Parameter TDR FrontPanel.

FrontPanel_TDRPortExt()

Adds a virtual length of ideal transmission line to each port of S parameter data

Syntax

y = FrontPanel_TDRPortExt(Data, Flag, Length, Velocity, LossType, Loss1, Loss2, Freq1, Freq2)

Arguments

Name	Description	Default	Range	Type	Required
Data	frequency transfer function †	None	(-∞:∞)	Complex Array	Yes
Flag	boolean flag if set FALSE default sweep values are used	None	[0:1]	Real	Yes
Length	array representing virtual lengths for each port † †	None	(-∞:∞)	Real Array	Yes
Velocity	normalized velocity	None	(-∞:∞)	Real Array	Yes
LossType	array representing loss types for each port (0 = Lossless, 1 = Constant, 2 = Linear) † †	None	(-∞:∞)	Real Array	Yes
Loss1	array representing loss in dB for each port (only used for LossType > 0) † †	None	(-∞:∞)	Real Array	Yes
Loss2	array representing loss in dB for each port (only used for LossType > 1) † †	None	(-∞:∞)	Real Array	Yes
Freq1	array representing frequency bin of loss for each port (only used for LossType > 0) † †	None	(-∞:∞)	Real Array	Yes
Freq2	array representing frequency bin of loss for each port (only used for LossType > 1) † †	None	(-∞:∞)	Real Array	Yes

† The multi-dimensional S parameter array can be created with FP_TDRFreq1Dto3D(S).
† † Each parameter for port extensions is the same length as the number of ports in the input S parameter

Examples

```
PortExtOut = FrontPanel_TDRPortExt(FP_TDRFreq1Dto3D(S), TRUE, [1,1], [1,1],
[0,0], [3,3], [3,3], [1 Ghz, 1 GHz], [2 GHz, 2 GHz])
```

Note
Any function with the prefix "FrontPanel_TDR" requires a special data structure for input parameters and is used exclusively by the S-Parameter TDR FrontPanel.

FrontPanel_TDRPortMap()

Maps input and output ports of an S parameter array

Syntax

y = FrontPanel_TDRPortMap(Data, Flag, MapArray)

Arguments

Name	Description	Default	Range	Type	Required
Data	frequency transfer function †	None	(-∞:∞)	Complex Array	Yes
Flag	boolean flag if set FALSE will return the original waveform	None	[0:1]	Real	Yes
MapArray	array where indicies represent output ports and values represent input ports	None	[1:∞)	Real	Yes

† The multi-dimensional S parameter array can be created with FP_TDRFreq1Dto3D(S).

Examples

```
PortMapOut = FrontPanel_TDRPortMap(FP_TDRFreq1Dto3D(S), TRUE, [1,2,3,4])
```

Note
Any function with the prefix "FrontPanel_TDR" requires a special data structure for input parameters and is used exclusively by the S-Parameter TDR FrontPanel.

FrontPanel_TDRSmooth()

Performs the specified smoothing operation on S parameter data

Syntax

y = FrontPanel_TDRSmooth(Data, Flag, Type, ParmList)

Arguments

Name	Description	Default	Range	Type	Required
Data	frequency transfer function †	None	(-∞:∞)	Complex Array	Yes
Flag	boolean flag if set FALSE will return the original waveform	None	[0:1]	Real	Yes
Type	specifies the type of smoothing (0 = moving average, 1 = alpha-beta)	None	[1:∞)	Real	Yes
ParmList	filter specific parameters (e.g. list(10) for N=10 width moving average filter)	None	(0:∞)	Real	Yes

† The multi-dimensional S parameter array can be created with FP_TDRFreq1Dto3D(S).

Examples

```
SmoothOut = FrontPanel_TDRSmooth(FP_TDRFreq1Dto3D(S), TRUE, 0, list(10))
```

Note
Any function with the prefix "FrontPanel_TDR" requires a special data structure for input parameters and is used exclusively by the S-Parameter TDR FrontPanel.

FrontPanel_TDRTimeScale()

Scales raw time data and sweep

Syntax

y = FrontPanel_TDRTimeScale(Data, Scale, Units, Z0, VelocityFactor, TimeModeFlag, FreqModeFlag, sm, sn)

Arguments

Name	Description	Default	Range	Type	Required
Data	multi-dimensional time waveform †	None	(-∞:∞)	Real Array	Yes
Scale	type of time scale (Scale = 1 for Impedance, Scale = 5 for Ref. Coeff.)	None	[1,5]	Real	Yes
Units	type of sweep units "seconds" or "meters"	None	[0,1]	Real	Yes
Z0	characteristic impedance	50	[1::∞)	Real	Yes
VelocityFactor	speed of light scale factor	None	[1::∞)	Real	Yes
TimeModeFlag	time mode flag (see FrontPanel_TDRTimeSweep())	None	[0:1]	Real	Yes
FreqModeFlag	freq mode flag (see FrontPanel_TDRFreqMode())	None	[0:1]	Real	Yes

† The input data is a multi-dimensional temporal array representing the time transform of an S parameter array.
The multi-dimensional S parameter array can be created with FP_TDRFreq1Dto3D(S).

Examples

```
TimeScaleOut = FrontPanel_TDRTimeScale(ts(FP_TDRFreq1Dto3D(S)), "seconds", 50)
```

Note
Any function with the prefix "FrontPanel_TDR" requires a special data structure for input parameters and is used exclusively by the S-Parameter TDR FrontPanel.

FrontPanel_TDRTimeSweep()

Transforms S parameter data into a time waveform

Syntax

y = FrontPanel_TDRTimeSweep(Data, Flag, Mode, Start, Stop, NumPoints)

Arguments

Name	Description	Default	Range	Type	Required
Data	multi-dimensional frequency transfer function †	None	(-∞:∞)	Complex Array	Yes
Flag	boolean flag if set FALSE default sweep values are used	None	[0:1]	Real	Yes
Mode	type of time mode (0 = Lowpass Impulse, 1 = Lowpass Step, 2 = Bandpass Impulse, 3 = Bandpass Step)	None	[0:2]	Real	Yes
Start	start time	None	(-∞:∞)	Real	Yes
Stop	stop time	None	(-∞:∞)	Real	Yes
NumPoints	number of time points	None	[1:∞)	Real	Yes

† The multi-dimensional S parameter array can be created with FP_TDRFreq1Dto3D(S).

Examples

```
TimeSweepOut = FrontPanel_TDRTimeSweep(FP_TDRFreq1Dto3D(S), TRUE, 1, 0, 10 ns,
101)
```

Note
Any function with the prefix "FrontPanel_TDR" requires a special data structure for input parameters and is used exclusively by the S-Parameter TDR FrontPanel.

FrontPanel_TDRWindow()

Returns windowed S parameter data

Syntax

y = FrontPanel_TDRWindow(Data, Flag, Type, ParmArray)

Arguments

Name	Description	Default	Range	Type	Required
Data	multi-dimensional frequency transfer function †	None	(-∞:∞)	Complex Array	Yes
Flag	boolean flag if set FALSE will return the original waveform	None	[0:1]	Real	Yes
Type	specifies the type of window † †	None	[0:6]	Real	Yes
ParmArray	list containing window parameter values: list(TimeMode, Parm1, Parm2)	None	None	Real Array	Yes

† The multi-dimensional S parameter array can be created with FP_TDRFreq1Dto3D(S).
† † The window types are enumerated by the following:
0 = Rectangular
1 = Hanning
2 = Hamming
3 = Gaussian
4 = Kaiser
5 = Blackman

Examples

```
WindowOut = FrontPanel_TDRWindow(FP_TDRFreq1Dto3D(S), TRUE, 2, list(0, .42))
```

tdr_inverse_peeling()

Undoes the layer peeling algorithm on a lossless, peeled, time-domain step response.

Syntax

y = tdr_inverse_peeling(Peeled_Data)

Arguments

Name	Description	Range	Type	Default	Required
Peeled_Data	peeled time-domain waveform	(-1, 1)	real		yes

Examples

```
y = tdr_inverse_peeling(PeeledResponse)
```

Defined in

Built in

See Also

tdr_peeling() (expmeas)

Notes/Equations

This function is the inverse of the peeling function; thus, excluding numerical sensitivity, y = tdr_inverse_peeling(tdr_peeling(y)).

tdr_peeling()

Performs TDR peeling or inverse scattering.

Syntax

yTDR = tdr_peeling(TDR_data, TDR_step)

Arguments

Name	Description	Default	Range	Type	Required
TDR_data	the TDR data to be impedance peeled	None	(-∞:∞)	Real	Yes
TDR_step	Step height	0.5	[0:1]	Real	No

Examples

```
peeled=tdr_peeling(vout, 0.5)
```

Defined in

Built in

See Also

tdr_inverse_peeling() (expmeas)

Notes/Equations

A more detailed explanation of the peeling algorithm can be found in the *S-Parameter TDR FrontPanel* (data) documentation.

Harmonic Balance Functions For Measurement Expressions

This section describes the Harmonic Balance functions in detail. The functions are listed in alphabetical order.

- *carr to im()* (expmeas)
- *cdrange()* (expmeas)
- *dc to rf()* (expmeas)
- *ifc()* (expmeas)
- *ip3 in()* (expmeas)
- *ip3 out()* (expmeas)
- *ipn()* (expmeas)
- *it()* (expmeas)
- *mix()* (expmeas)
- *pae()* (expmeas)
- *pfc()* (expmeas)
- *phase gain()* (expmeas)
- *pspec()* (expmeas)
- *pt()* (expmeas)
- *remove noise()* (expmeas)
- *sfdr()* (expmeas)
- *snr()* (expmeas)
- *spur track()* (expmeas)
- *spur track with if()* (expmeas)
- *thd func()* (expmeas)
- *ts()* (expmeas)
- *vfc()* (expmeas)
- *vspec()* (expmeas)
- *vt()* (expmeas)

Working with Harmonic Balance Data

Harmonic Balance (HB) Analysis produces complex voltages and currents as a function of frequency or harmonic number. A single analysis produces 1-dimensional data. Individual harmonic components can be indexed by means of "[ ]". Multi-tone HB also produces 1-dimensional data. Individual harmonic components can be indexed as usual by means of "[ ]". However, the function *mix()* (expmeas) provides a convenient way to select a particular mixing component.

carr_to_im()

This measurement gives the suppression (in dB) of a specified IMD product below the fundamental power at the output port.

Syntax

y = carr_to_im(vOut, fundFreq, imFreq, Mix)

Arguments

Name	Description	Default	Range	Type	Required
vOut	signal voltage at the output port	None	[0:∞)	Real, Complex	Yes
fundFreq	harmonic frequency indices for the fundamental frequency	None	(-∞:∞)	Integer array	Yes
imFreq	harmonic frequency indices for the IMD product of interest	None	(-∞:∞)	Integer array	Yes
Mix	consists of all possible vectors of harmonic frequency (mixing terms) in the analysis †	None	(-∞:∞)	Integer array	No

† It is required whenever the first argument is a spectrum obtained from an expression that operates on the voltage and/or current spectrums.

Examples

```
a = carr_to_im(out, {1, 0}, {2, -1})
```

Defined in

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

See Also

ip3_out() (expmeas)

cdrange()

Returns compression dynamic range

Syntax

y = cdrange(nf, inpwr_lin, outpwr_lin, outpwr)

Arguments

Name	Description	Default	Range	Type	Required
nf	noise figure at the output port	None	[0:∞)	Real	Yes
inpwr_lin	input power in the linear region	None	[0:∞)	Real	Yes
outpwr_lin	output power in the linear region	None	[0:∞)	Real	Yes
outpwr	output power at 1 dB compression	None	[0:∞)	Real	Yes

Examples

```
a = cdrange(nf2, inpwr_lin, outpwr_lin, outpwr)
```

Defined in

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

See Also

sfdr() (expmeas)

Notes/Equations

Used in XDB simulation. The compressive dynamic range ratio identifies the dynamic range from the noise floor to the 1-dB gain-compression point. The noise floor is the noise power with respect to the reference bandwidth.

dc_to_rf()

This measurement computes the DC-to-RF efficiency of any part of the network

Syntax

y = dc_to_rf(vPlusRF, vMinusRF, vPlusDC, vMinusDC, currentRF, currentDC, harm_freq_index, Mix)

Arguments

Name	Description	Default	Range	Type	Required
vPlusRF	voltage at the positive terminal	None	(-∞:∞)	Real, Complex	Yes
vMinusRF	voltage at the negative terminal	None	(-∞:∞)	Real, Complex	Yes
vPlusDC	DC voltage at the positive terminal	None	(-∞:∞)	Real, Complex	Yes
vMinusDC	DC voltage at the negative terminal	None	(-∞:∞)	Real, Complex	Yes
currentRF	RF current for power calculation	None	(-∞:∞)	Real, Complex	Yes
currentDC	DC current for power calculation	None	(-∞:∞)	Real, Complex	Yes
harm_freq_index	harmonic index of the RF frequency at the output por	None	(-∞:∞)	Integer array	Yes
Mix	consists of all possible vectors of harmonic frequency (mixing terms) in the analysis †	None	(-∞:∞)	Matrix	No

† It is required whenever the first argument is a spectrum obtained from an expression that operates on the voltage and/or current spectrums.

Examples

```
a = dc_to_rf(vrf, 0, vDC, 0, I_Probe1.i, SRC1.i, {1,0})
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

ifc()

This measurement gives the RMS current value of one frequency-component of a harmonic balance waveform

Syntax

y = ifc(iOut, harm_freq_index, Mix)

Arguments

Name	Description	Default	Range	Type	Required
iOut	current through a branch	None	(-∞:∞)	Real, Complex	Yes
harm_freq_index	harmonic index of the desired frequency †	None	(-∞:∞)	Integer array	Yes
Mix	consists of all possible vectors of harmonic frequency (mixing terms) in the analysis † †	None	(-∞:∞)	Matrix	No

† Note that the harm_freq_index argument's entry should reflect the number of tones in the harmonic balance controller. For example, if one tone is used in the controller, there should be one number inside the braces; two tones would require two numbers separated by a comma.
† † Mix is required whenever the first argument is a spectrum obtained from an expression that operates on the voltage and/or current spectrums.

Examples

```
The following example is for two tones in the harmonic balance controller:
y = ifc(I_Probe1.i, {1, 0})
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

pfc() (expmeas), *vfc()* (expmeas)

Notes/Equations

This function should not be used for DC measurements. If used, the results will be less by a factor of sqrt(2.0).

ip3_in()

This measurement determines the input third-order intercept point (in dBm) at the input port with reference to a system output port.

Syntax

y = ip3_in(vOut, ssGain, fundFreq, imFreq, zRef, Mix)

Arguments

Name	Description	Default	Range	Type	Required
vOut	signal voltage at the output port	None	[0:∞)	Real, Complex	Yes
ssGain	small signal gain in dB	None	[0:∞)	Real	Yes
fundFreq	harmonic frequency indices for the fundamental frequency	None	(-∞:∞)	Integer array	Yes
imFreq	harmonic frequency indices for the intermodulation frequency	None	(-∞:∞)	Integer array	Yes
zRef	reference impedance	50.0	(-∞:∞)	Real, Complex	No
Mix	consists of all possible vectors of harmonic frequency (mixing terms) in the analysis †	None	(-∞:∞)	Integer array	No

† It is required whenever the first argument is a spectrum obtained from an expression that operates on the voltage and/or current spectrums.

Examples

```
y = ip3_in(vOut, 22, {1, 0}, {2, -1}, 50)
```

Defined in

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

See Also

ip3_out() (expmeas), *ipn()* (expmeas)

Notes/Equations

To measure the third-order intercept point, you must setup a Harmonic Balance simulation with the input signal driving the circuit in the linear range. Input power is typically set 10 dB below the 1 dB gain compression point. If you simulate the circuit in the nonlinear region, the calculated results will be incorrect.

ip3_out()

This measurement determines the output third-order intercept point (in dBm) at the system output port.

Syntax

y = ip3_out(vOut, fundFreq, imFreq, zRef, Mix)

Arguments

Name	Description	Default	Range	Type	Required
vOut	signal voltage at the output port	None	[0:∞)	Real, Complex	Yes
fundFreq	harmonic frequency indices for the fundamental frequency	None	(-∞:∞)	Integer array	Yes
imFreq	harmonic frequency indices for the intermodulation frequency	None	(-∞:∞)	Integer array	Yes
zRef	reference impedance	50.0	(-∞:∞)	Real, Complex	No
Mix	consists of all possible vectors of harmonic frequency (mixing terms) in the analysis †	None	(-∞:∞)	Integer array	No

† It is required whenever the first argument is a spectrum obtained from an expression that operates on the voltage and/or current spectrums.

Examples

```
y = ip3_out(vOut, {1, 0}, {2, -1}, 50)
```

Defined in

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

See Also

ip3_in() (expmeas), *ipn()* (expmeas)

Notes/Equations

To measure the third-order intercept point, you must setup a Harmonic Balance simulation with the input signal driving the circuit in the linear range. Input power is typically set 10 dB below the 1 dB gain compression point. If you simulate the circuit in the nonlinear region, the calculated results will be incorrect.

ipn()

This measurement determines the output nth-order intercept point (in dBm) at the system output port

Syntax

y = ipn(vPlus, vMinus, iOut, fundFreq, imFreq, n, Mix)

Arguments

Name	Description	Default	Range	Type	Required
vPlus	voltage at the positive output terminal	None	(-∞:∞)	Real, Complex	Yes
vMinus	voltage at the negative output terminal	None	(-∞:∞)	Real, Complex	Yes
iOut	current through a branch	None	(-∞:∞)	Real, Complex	Yes
fundFreq	harmonic indices of the fundamental frequency	None	(-∞:∞)	Integer array	Yes
imFreq	harmonic indices of the intermodulation frequency	None	(-∞:∞)	Integer array	Yes
n	order of the intercept	None	[1:∞)	Integer	Yes
Mix	consists of all possible vectors of harmonic frequency (mixing terms) in the analysis †	None	(-∞:∞)	Matrix	No

† It is required whenever the first argument is a spectrum obtained from an expression that operates on the voltage and/or current spectrums.

Examples

```
y = ipn(vOut, 0, I_Probe1.i, {1, 0}, {2, -1}, 3)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

ip3_in() (expmeas), *ip3_out()* (expmeas)

Notes/Equations

To measure the third-order intercept point, you must setup a Harmonic Balance simulation with the input signal driving the circuit in the linear range. Input power is typically set 10 dB below the 1 dB gain compression point. If you simulate the circuit in the nonlinear region, the calculated results will be incorrect.

it()

This measurement converts a harmonic-balance current frequency spectrum to a time-domain current waveform.

Syntax

it(iOut, tmin, tmax, numOfPnts)

Arguments

Name	Description	Default	Range	Type	Required
iOut	current through a branch	None	(-∞:∞)	Real, Complex	Yes
tmin	start time	0	[0:∞)	Real	No
tmax	stop time	2*cycle time	[0:∞)	Real	No
numOfPnts	number of points	101	[0:∞)	Integer	No

Examples

```
y = it(I_Probe1.i, 0, 10nsec, 201)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

vt() (expmeas)

mix()

Returns a component of a spectrum based on a vector of mixing indices

Syntax

mix(xOut, harmIndex, Mix)

Arguments

Name	Description	Default	Range	Type	Required
xOut	voltage or a current spectrum	None	(-∞:∞)	Real, Complex	Yes
harmIndex	desired vector of harmonic frequency indices (mixing terms)	None	(-∞:∞)	Integer array	Yes
Mix	variable consisting of all possible vectors of harmonic frequency indices (mixing terms) in the analysis	†	(-∞:∞)	Matrix	No †

† Mix, is required whenever the first argument is a spectrum obtained from an expression that operates on the voltage and/or current spectrums. This is not required if the voltage is a named node or if the current is from a probe.

Examples

```
In the example below, vOut is the voltage at a named node. Therefore the third
argument Mix is not required.
y = mix(vOut, {2, -1})
In the example below, vExp is an expression. Therefore in the mix() function,
the third argument Mix is required.
vExp=vOut*vOut/50
z = mix(vExp, {2, -1}, Mix)
Obtain the frequency corresponding to the mixing term {2, -1}
f = mix(freq, {2, -1})
```

```
In the examples below, Vload is the load voltage from a 2-tone Harmonic Balance
analysis with sweep input power.
f1=mix(freq[0,::],{1,0},Mix[0,::]) returns the fundamental frequency at the
first power point.
freI=vs(freq,freq)
f2=mix(freI,{1,0},Mix) returns the fundamental frequency vs the input power.
```

Defined in

Built in

See Also

find_index() (expmeas)

Notes/Equations

Used in Harmonic Balance analysis.
It is used to obtain the mixing component of a voltage or a current spectrum corresponding to particular harmonic frequency indices or mixing terms.

pae()

This measurement computes the power-added efficiency (in percent) of any part of the circuit

Syntax

y = pae(vPlusOut, vMinusOut, vPlusIn, vMinusIn, vPlusDC, vMinusDC, iOut, iIn, iDC, outFreq, inFreq)

Arguments

Name	Description	Default	Range	Type	Required
vPlusOut	output voltage at the positive terminal	None	(-∞:∞)	Real, Complex	Yes
vMinusOut	output voltage at the negative terminal	None	(-∞:∞)	Real, Complex	Yes
vPlusIn	input voltage at the positive terminal	None	(-∞:∞)	Real, Complex	Yes
vMinusIn	input voltage at the negative terminal	None	(-∞:∞)	Real, Complex	Yes
vPlusDC	DC voltage at the positive terminal	None	(-∞:∞)	Real, Complex	Yes
vMinusDC	DC voltage at the negative terminal	None	(-∞:∞)	Real, Complex	Yes
iOut	output current	None	(-∞:∞)	Real, Complex	Yes
iIn	input current	None	(-∞:∞)	Real, Complex	Yes
iDC	DC current	None	(-∞:∞)	Real, Complex	Yes
outFreq	harmonic indices of the fundamental frequency at the output port	None	(-∞:∞)	Integer array	Yes
inFreq	harmonic indices of the fundamental frequency at the input port	None	(-∞:∞)	Integer array	Yes
Mix	consists of all possible vectors of harmonic frequency (mixing terms) in the analysis †	None	(-∞:∞)	Matrix	No

† It is required whenever the first argument is a spectrum obtained from an expression that operates on the voltage and/or current spectrums.

Examples

```
y = pae(vOut, 0, vIn, 0, v1, 0, I_Probe1.i, I_Probe2.i, I_Probe3.i, 1, 1)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

db() (ael), *dbm()* (ael)

pfc()

This measurement gives the RMS power value of one frequency component of a harmonic balance waveform

Syntax

y = pfc(vPlus, vMinus, iOut, harm_freq_index)

Arguments

Name	Description	Default	Range	Type	Required
vPlus	voltage at the positive output terminal	None	(-∞:∞)	Real, Complex	Yes
vMinus	voltage at the negative output terminal	None	(-∞:∞)	Real, Complex	Yes
iOut	current through a branch	None	(-∞:∞)	Real, Complex	Yes
harm_freq_index	harmonic index of the desired frequency †	None	(-∞:∞)	Integer array	Yes
Mix	consists of all possible vectors of harmonic frequency (mixing terms) in the analysis † †	None	(-∞:∞)	Matrix	No

† Note that the harm_freq_index argument's entry should reflect the number of tones in the harmonic balance controller. For example, if one tone is used in the controller, there should be one number inside the braces; two tones would require two numbers separated by a comma.
† † Mix is required whenever the first argument is a spectrum obtained from an expression that operates on the voltage and/or current spectrums.

Examples

```
y = pfc(vOut, 0, I_Probe1.i, {1, 0})
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

ifc() (expmeas), *vfc()* (expmeas)

Notes/Equations

This function should not be used for DC measurements. If used, the results will be less by a factor of 2.0.

phase_gain()

Returns the gain associated with the phase (normally zero)
crossing at associated power. Can be used in Harmonic Balance Analysis of
an oscillator to get the loop-gain. Returns an array of gains.

Syntax

y = phase_gain(Gain, DesiredPhase)

Arguments

Name	Description	Default	Range	Type	Required
Gain	Two dimensional data representing gain. E.g. Loop-gain of an oscillator.	None	(-∞:∞)	Complex	Yes
DesiredPhase	A single value representing the desired phase.	0	(-∞:∞)	Real	No

Examples

```
We assume that a Harmonic Balance analysis has been performed at different
power.
gainAtZeroPhase = phase_gain(Vout/Vin, 0) returns the gain at zero phase.
```

Defined in

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

pspec()

This measurement gives a power frequency spectrum in harmonic balance analyses

Syntax

y = pspec(vPlus, vMinus, iOut)

Arguments

Name	Description	Default	Range	Type	Required
vPlus	voltage at the positive node	None	(-∞:∞)	Real, Complex	Yes
vMinus	voltage at the negative node	None	(-∞:∞)	Real, Complex	Yes
iOut	current through a branch	0	(-∞:∞)	Real	No

Examples

```
a = pspec(vOut, 0, I_Probe1.i)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

pt() (expmeas), *ispec()* (expmeas), *vspec()* (expmeas)

pt()

This measurement calculates the total power of a harmonic balance frequency spectrum.

Syntax

y = pt(vPlus, vMinus, iOut)

Arguments

Name	Description	Default	Range	Type	Required
vPlus	voltage at the positive node	None	(-∞:∞)	Real, Complex	Yes
vMinus	voltage at the negative node	None	(-∞:∞)	Real, Complex	Yes
iOut	current through a branch	0	(-∞:∞)	Real	No

Examples

```
y = pt(vOut, 0, I_Probe1.i)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

pspec() (expmeas)

remove_noise()

Removes noise floor data from noise data and returns an array.

Syntax

nd = remove_noise(NoiseData, NoiseFloor)

Arguments

Name	Description	Default	Range	Type	Required
NoiseData	Two dimensional array representing noise data	None	(-∞:∞)	Real, Complex	Yes
NoiseFloor	Single dimensional array representing noise floor	None	(-∞:∞)	Real, Complex	Yes

Examples

```
nd = remove_noise(vnoise, noiseFloor) returns the noise data with the noise
floor removed
```

Defined in

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

Notes/Equations

Used in Harmonic Balance analysis.
NoiseData is [m,n] where m is receive frequency and n is interference offset frequency. If NoiseData is [m,n], then NoiseFloor must be [m]. If NoiseData - NosieFloor is less than zero, then -200 dBm is used.

sfdr()

Returns the spurious-free dynamic range

Syntax

y = sfdr(vOut, ssGain, nf, noiseBW, fundFreq, imFreq, zRef{, Mix})

Arguments

Name	Description	Default	Range	Type	Required
vOut	signal voltage at the output port	None	[0:∞)	Real, Complex	Yes
ssGain	small signal gain in dB	None	[0:∞)	Real	Yes
nf	noise figure at the output port	None	[0:∞)	Real	Yes
fundFreq	harmonic frequency indices for the fundamental frequency	None	(-∞:∞)	Integer array	Yes
imFreq	harmonic frequency indices for the intermodulation frequency	None	(-∞:∞)	Integer array	Yes
zRef	reference impedance	50.0	(-∞:∞)	Real, Complex	No
Mix	consists of all possible vectors of harmonic frequency (mixing terms) in the analysis †	None	(-∞:∞)	Integer array	No

† Mix is required whenever the first argument is a spectrum obtained from an expression that operates on the voltage and/or current spectrums.

Examples

```
a = sfdr(vIn, 12, nf2, , {1, 0}, {2, -1}, 50)
```

Defined in

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

See Also

ip3_out() (expmeas)

Notes/Equations

Used in a Harmonic Balance and Small-signal S-parameter. It appears in the HB Simulation palette.

This measurement determines the spurious-free dynamic-range ratio for noise power with respect to the reference bandwidth.

To measure the third-order intercept point, you must setup a Harmonic Balance simulation with the input signal driving the circuit in the linear range. Input power is typically set 10 dB below the 1 dB gain compression point. If you simulate the circuit in the nonlinear region, the calculated results will be incorrect.

For the NoiseBW argument, you normally have to use the default value of 1 Hz. If you increase it, as per the bandwidth of a filter you are using, dynamic range can be greatly reduced by the resulting rise in the noise floor.

snr()

This measurement gives the ratio of the output signal power (at the fundamental frequency for a harmonic balance simulation) to the total noise power (in dB).

Syntax

y = snr(vOut, vOut.noise, fundFreq, Mix)

Arguments

Name	Description	Default	Range	Type	Required
vOut	signal voltage at the output port	None	[0:∞)	Real, Complex	Yes
vOut.noise	noise voltage at the output port	None	[0:∞)	Real, Complex	Yes
fundFreq	harmonic frequency indices for the fundamental frequency †	None	(-∞:∞)	Integer array	Yes
Mix	consists of all possible vectors of harmonic frequency (mixing terms) in the analysis † †	None	(-∞:∞)	Integer array	No

† Note that fundFreq is not optional; it is required for harmonic balance simulations, but it is not applicable in AC simulations.
† † Mix is required whenever the first argument is a spectrum obtained from an expression that operates on the voltage and/or current spectrums.

Examples

```
a = snr(vOut, vOut.noise, {1, 0})
returns the signal-to-power noise ratio for a harmonic balance simulation.
a = snr(vOut, vOut.noise)
returns the signal-to-power noise ratio for an AC simulation.
```

Defined in

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

See Also

ns_pwr_int() (expmeas), *ns_pwr_ref_bw()* (expmeas)

Notes/Equations

If the second argument is of higher dimension than the first, the noise bandwidth used for the purpose of computing snr will be equal to the frequency spacing of the innermost dimension of the noise data, instead of the standard value of 1 Hz.

spur_track()

Returns the maximum power of all signals appearing in a user-specifiable IF band, as a single RF input signal is stepped. If there is no IF signal appearing in the specified band, for a particular RF input frequency, then the function returns an IF signal power of -500 dBm.

Syntax

IFspur = spur_track(vs(vout, freq), if_low, if_high, rout)

Arguments

Name	Description	Default	Range	Type	Required
vout	IF output node name	None	None	String	Yes
if_low	lowest frequency in the IF band	None	[0:∞)	Real	Yes
if_high	highest frequency in the IF band	None	[0:∞)	Real	Yes
rout	load resistance connected to the IF port, necessary for computing power delivered to the load	None	[0:∞)	Real	Yes

Examples

```
IFspur = spur_track(vs(HB.VIF1, freq), Fiflow[0, 0], Fifhigh[0, 0], 50)
where
VIF1 is the named node at the IF output.
Fiflow is the lowest frequency in the IF band.
Fifhigh is the highest frequency in the IF band.
50 is the IF load resistance.
Fiflow and Fifhigh are passed parameters from the schematic page (although they
can be defined on the data display page instead.) These parameters, although
single-valued on the schematic, become matrices when passed to the dataset,
where each element of the matrix has the same value. The [0, 0] syntax just
selects one element from the matrix.
```

Defined in

$HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael

See Also

spur_track_with_if() (expmeas)

Notes/Equations

Used in Receiver spurious response simulations.
IFspur computed above will be the power in dBm of the maximum signal appearing in the IF band, versus RF input frequency. Note that it would be easy to modify the function to compute dBV instead of dBm.

This function is meant to aid in testing the response of a receiver to RF signals at various frequencies. This function shows the maximum power of all signals appearing in a user-specifiable IF band, as a single RF input signal is stepped. There could be fixed, interfering tones present at the RF input also, if desired. The maximum IF signal power may be plotted or listed versus the stepped RF input signal frequency. If there is no IF signal appearing in the specified band, for a particular RF input frequency, then the function returns an IF signal power of -500 dBm.

spur_track_with_if()

Returns the maximum power of all signals appearing in a user-specifiable IF band, as a single RF input signal is stepped. In addition, it shows the IF frequencies and power levels of each signal that appears in the IF band, as well as the corresponding RF signal frequency.

Syntax

IFspur = spur_track_with_if(vs(vout, freq), if_low, if_high, rout)

Arguments

Name	Description	Default	Range	Type	Required
vout	IF output node name	None	None	String	Yes
if_low	lowest frequency in the IF band	None	[0:∞)	Real	Yes
if_high	highest frequency in the IF band	None	[0:∞)	Real	Yes
rout	load resistance connected to the IF port, necessary for computing power delivered to the load	None	[0:∞)	Real	Yes

Examples

```
IFspur=spur_track_with_if(vs(HB.VIF1, freq), Fiflow[0, 0], Fifhigh[0, 0], 50)
where
VIF1 is the named node at the IF output.
Fiflow is the lowest frequency in the IF band.
Fifhigh is the highest frequency in the IF band.
50 is the IF load resistance.
Fiflow and Fifhigh are passed parameters from the schematic page (although they
can be defined on the data display page instead.) These parameters, although
single-valued on the schematic, become matrices when passed to the dataset,
where each element of the matrix has the same value. The [0, 0] syntax just
selects one element from the matrix.
```

Defined in

$HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael

See Also

spur_track() (expmeas)

Notes/Equations

Used in Receiver spurious response simulations.

IFspur computed above will be the power in dBm of the maximum signal appearing in the IF band, versus RF input frequency. Note that it would be easy to modify the function to compute dBV instead of dBm.

This function is meant to aid in testing the response of a receiver to RF signals at various frequencies. This function, similar to the spur_track function, shows the maximum power of all signals appearing in a user-specifiable IF band, as a single RF input signal is stepped. In addition, it shows the IF frequencies and power levels of each signal that appears in the IF band, as well as the corresponding RF signal frequency. There could be fixed, interfering tones present at the RF input also, if desired. The maximum IF signal power may be plotted or listed versus the stepped RF input signal frequency.

thd_func()

This measurement returns the Total Harmonic Distortion percentage.

Syntax

y = thd_func(v)

Arguments

Name	Description	Default	Range	Type	Required
v	voltage	None	(-∞:∞)	Real, Complex	Yes

Examples

```
y = thd_func(Vload)
```

Defined In

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

ts()

Performs a frequency-to-time transform

Syntax

y= ts(x, tstart, tstop, numtpts, dim, windowType, windowConst, nptsspec)

Arguments

Name	Description	Default	Range	Type	Required
x	frequency-domain data to be transformed	None	(-∞:∞)	Real	Yes
tstart	starting time	0	[0:∞)	Integer, Real	No
tstop	stoping time	tstop = tstart + 2.0/fabs(freq[0])	[0:∞)	Integer, Real	No
numtpts	number of time points	101	[1:∞)	Integer	No
dim	dimension to be transformed (not used currently)	highest dimension	[1:∞)	Integer	No
windowType	type of window to be applied to the data	0	[0:9] †	Integer, string	No
windowConst	window constant ††	0	[0:∞)	Integer, Real	No
nptsspec	number of first harmonics to be transformed	1	[1:NumFreqs]	Integer	No

† The window types and their default constants are:
0 = None
1 = Hamming 0.54
2 = Hanning 0.50
3 = Gaussian 0.75
4 = Kaiser 7.865
5 = 8510 6.0 (This is equivalent to the time-to-frequency transformation with normal gate shape setting in the 8510 series network analyzer.)
6 = Blackman
7 = Blackman-Harris
8 = 8510-Minimum 0
9 = 8510-Maximum 13
†† windowConst is not used if windowType is 8510

Examples

```
The following examples of ts assume that a harmonic balance simulation was
performed with a fundamental frequency of 1 GHz and order = 8:
Y=ts(vOut) returns the time series (0, 20ps, ... , 2ns)
Y=ts(vOut, 0, 1ns) returns the time series (0, 10ps, ..., 1ns)
Y=ts(vOut, 0, 10ns, 201) returns the time series (0, 50ps, ... , 10ns)
Y=ts(vOut, , , , , , , 3) returns the time series (0, 20ps, ... , 2ns), but
only uses harmonics from 1 to 3 GHz
```

Defined in

Built in

See Also

fft() (expmeas), *fs()* (expmeas), *fspot()* (expmeas)

Notes/Equations

Used in Harmonic Balance and Circuit Envelope simulations.

The default number of time points is computed as follows:
if the number of frequencies is < 2, it is set to 101
else use the following formula

```
newdefault_numtpts = fabs(16.0 * 2.0 * freq[ numfreqs-1] / freq[1]);
    if the new value is less than 101
           it is set to 101
    if it is greater than 10001,
           it is set to 10001.
```

So the new default is a value between 101 and 10001, and varies based on the number of frequencies, and the actual frequency values.

The dim argument is not used and should be left empty in the expression. Entering a value will have no impact on the results.

ts(x) returns the time domain waveform from a frequency spectrum. When x is a multidimensional vector, the transform is evaluated for each vector in the specified dimension. For example, if x is a matrix, then ts(x) applies the transform to every row of the matrix. If x is three dimensional, then ts(x) is applied in the lowest dimension over the remaining two dimensions. The dimension over which to apply the transform may be specified by dimension; the default is the lowest dimension (dimension=1). ts() originated in MDS and is similar to vt().

> **Note**
> Where x is multi-dimensional, the ts (x) function expects frequency to be the most frequently-changing variable in the input data, and the function may produce unpredictable results if this is not the case. For a dataset which does not have frequency as the most frequently-changing variable, the problem can be eliminated by using the permute (x) function to change the dimension order, as follows:
>
> ```
> permx = permute(x)
> ts(permx)
> ```

x must be numeric. It will typically be data from a Harmonic Balance analysis.

By default, two cycles of the waveform are produced with 101 points, starting at time zero, based on the lowest frequency in the input spectrum. These may be changed by setting tstart, tstop, or numtpts.

All of the harmonics in the spectrum will be used to generate the time domain waveform. When the higher-order harmonics are known not to contribute significantly to the time domain waveform, only the first n harmonics may be requested for the transform, by setting nptsspec = n.

ts(x) can be used to process more than Harmonic Balance. For example, ts(x) can be used to convert AC simulation data to a time domain waveform using only one frequency point in the AC simulation.

Note that if the data does not have an explicit independent variable "freq", it is assumed to be starting at 0.0 and incremented in steps of 1. In some cases, this might lead to an incorrect time waveform. For example to obtain the time waveform of the second tone in a

single tone analysis, using ts(Vout[2]) would give incorrect results. In this case use ts(Vout[2::3],,,,,,,1) to obtain the correct waveform.

In harmonic balance analysis if variables are swept, the dataset saved has an inner most independent *harmindex* and a first dependent *freq* . In the case of argument *tstop* not being given the default is calculated using the dependent variable *freq* . But if the argument *x* in the ts() function is arithmetically operated or sub-indexed, the dependent *freq* is not maintained and in such cases the ts() function returns incorrect *time* values. This can be prevented by first using the ts() function on such data and then obtaining the necessary data. The example below illustrates this point.
Assume that the harmonic balance analysis has swept variable *Pin* . In this case the data has two independents [Pin, harmindex]. If *Idd.i* and *Iout.i* are 2 currents then the expression below:
```
tsERR = ts(Idd.i - Iout.i)
```
would return the incorrect time axis values. This can be solved by the expression:
```
tsWORKS = ts(Idd.i) - ts(Iout.i)
```
Similarly the expression:
```
tsERR1 = ts(Idd.i[0,::])
```
would return the incorrect time axis values. This can be solved by the expressions:
```
ts_Idd = ts(Idd.i)
ts_Idd_0 = ts_Idd[0,::]
```

vfc()

This measurement gives the RMS voltage value of one frequency-component of a harmonic balance waveform

Syntax

y = vfc(vPlus, vMinus, harm_freq_index)

Arguments

Name	Description	Default	Range	Type	Required
vPlus	voltage at the positive output terminal	None	(-∞:∞)	Real, Complex	Yes
vMinus	voltage at the negative output terminal	None	(-∞:∞)	Real, Complex	Yes
harm_freq_index	harmonic index of the desired frequency †	None	(-∞:∞)	Integer array	Yes
Mix	consists of all possible vectors of harmonic frequency (mixing terms) in the analysis † †	None	(-∞:∞)	Matrix	No

† Note that the harm_freq_index argument's entry should reflect the number of tones in the harmonic balance controller. For example, if one tone is used in the controller, there should be one number inside the braces; two tones would require two numbers separated by a comma.
† † Mix is required whenever the first argument is a spectrum obtained from an expression that operates on the voltage and/or current spectrums.

Examples

```
a = vfc(vOut, 0, {1, 0})
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

ifc() (expmeas), *pfc()* (expmeas)

Notes/Equations

This function should not be used for DC measurements. If used, the results will be less by a factor of sqrt(2.0).

vspec()

Returns the voltage frequency spectrum

Syntax

y = vspec(vPlus, vMinus)

Arguments

Name	Description	Default	Range	Type	Required
vPlus	voltage at the positive node	None	(-∞:∞)	Real, Complex	Yes
vMinus	voltage at the negative node	None	(-∞:∞)	Real, Complex	Yes

Examples

```
a = vspec(v1, v2)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

ispec() (expmeas), *pspec()* (expmeas)

Notes/Equations

Used in Harmonic Balance analysis.
This measurement gives a voltage frequency spectrum across any two nodes. The measurement gives a set of RMS voltages at each frequency.

vt()

This measurement converts a harmonic-balance voltage frequency spectrum to a time-domain voltage waveform.

Syntax

y = vt(vPlus, vMinus, tmin, tmax, numOfPnts)

Arguments

Name	Description	Default	Range	Type	Required
vPlus	voltage at the positive node	None	(-∞:∞)	Real, Complex	Yes
vMinus	voltage at the negative node	None	(-∞:∞)	Real, Complex	Yes
tmin	start time	0	[0:∞)	Real	No
tmax	stop time	2*cycle time	[0:∞)	Real	No
numOfPnts	number of points	101	[0:∞)	Integer	No

Examples

```
a = vt(vOut, 0, 0, 10nsec, 201)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

it() (expmeas), *ts()* (expmeas)

Notes/Equations

The vt() function originated in Series IV and simply calls the frequency to time domain transformer function, ts(). In some cases, if default values are used for tmin, tmax, and numOfPnts, the proper results may not be obtained due to an insufficient number of points. In such cases, the appropriate values for tmin, tmax, and numOfPnts need to be used. This function uses default values for window type and window constant, and in certain cases the correct results may not be obtained due to this fact. In this situation, use the ts() function instead with the proper windowing. For more information, refer to the Notes/Equations section for the ts() function.

Math Functions For Measurement Expressions

This section describes the math functions in detail. The functions are listed in alphabetical order.

- *abs() Measurement* (expmeas)
- *acos() Measurement* (expmeas)
- *acosh() Measurement* (expmeas)
- *acot() Measurement* (expmeas)
- *acoth() Measurement* (expmeas)
- *asin() Measurement* (expmeas)
- *asinh() Measurement* (expmeas)
- *atan2() Measurement* (expmeas)
- *atan() Measurement* (expmeas)
- *atanh() Measurement* (expmeas)
- *ceil() Measurement* (expmeas)
- *cint() Measurement* (expmeas)
- *cmplx() Measurement* (expmeas)
- *complex() Measurement* (expmeas)
- *conj() Measurement* (expmeas)
- *convBin() Measurement* (expmeas)
- *convHex() Measurement* (expmeas)
- *convInt()* (expmeas)
- *convOct() Measurement* (expmeas)
- *cos() Measurement* (expmeas)
- *cosh() Measurement* (expmeas)
- *cot() Measurement* (expmeas)
- *coth() Measurement* (expmeas)
- *cum prod()* (expmeas)
- *cum sum()* (expmeas)
- *db() Measurement* (expmeas)
- *dbm() Measurement* (expmeas)
- *dbmtow() Measurement* (expmeas)
- *deg() Measurement* (expmeas)
- *diagonal()* (expmeas)
- *diff()* (expmeas)
- *erf()* (expmeas)
- *erfc()* (expmeas)
- *erfcinv()* (expmeas)
- *erfinv()* (expmeas)
- *exp() Measurement* (expmeas)
- *fft()* (expmeas)
- *fix() Measurement* (expmeas)
- *float() Measurement* (expmeas)
- *floor() Measurement* (expmeas)
- *fmod() Measurement* (expmeas)
- *hypot() Measurement* (expmeas)
- *identity()* (expmeas)
- *im() Measurement* (expmeas)
- *imag() Measurement* (expmeas)

- *int() Measurement* (expmeas)
- *integrate()* (expmeas)
- *interp()* (expmeas)
- *interpolate()* (expmeas)
- *inverse()* (expmeas)
- *jn() Measurement* (expmeas)
- *ln() Measurement* (expmeas)
- *log10() Measurement* (expmeas)
- *log() Measurement* (expmeas)
- *mag() Measurement* (expmeas)
- *max2() Measurement* (expmeas)
- *max() Measurement* (expmeas)
- *max outer()* (expmeas)
- *min2() Measurement* (expmeas)
- *min() Measurement* (expmeas)
- *min outer()* (expmeas)
- *num() Measurement* (expmeas)
- *ones()* (expmeas)
- *phase() Measurement* (expmeas)
- *phasedeg() Measurement* (expmeas)
- *phaserad() Measurement* (expmeas)
- *polar() Measurement* (expmeas)
- *pow() Measurement* (expmeas)
- *prod()* (expmeas)
- *rad() Measurement* (expmeas)
- *re() Measurement* (expmeas)
- *real() Measurement* (expmeas)
- *rms()* (expmeas)
- *round() Measurement* (expmeas)
- *sgn() Measurement* (expmeas)
- *sin() Measurement* (expmeas)
- *sinc() Measurement* (expmeas)
- *sinh() Measurement* (expmeas)
- *sqr()* (expmeas)
- *sqrt() Measurement* (expmeas)
- *step() Measurement* (expmeas)
- *sum() Measurement* (expmeas)
- *tan() Measurement* (expmeas)
- *tanh() Measurement* (expmeas)
- *transpose()* (expmeas)
- *wtodbm() Measurement* (expmeas)
- *xor() Measurement* (expmeas)
- *zeros()* (expmeas)

Note
You can generally use these functions with data from any type of analysis. They consist of traditional math (e.g., trigonometric functions and matrix operations) and other functions.

abs()

Returns the absolute value of a integer, real or complex number

Syntax

y = abs(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	(-∞:∞)	Integer, real, complex	Yes

Examples

```
a = abs(-45) returns 45
```

Defined in

Built in

See Also

cint() (expmeas), *exp()* (expmeas), *float()* (expmeas), *int()* (expmeas), *log()* (expmeas), *log10()* (expmeas), *pow()* (expmeas), *sgn()* (expmeas), *sqrt()* (expmeas)

Note
The function name abs() is used for more than one type of expression. For comparison, see the Simulator Expression *abs() Expression* (expsim) and the AEL Function *abs() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

Notes/Equations

In the case of a complex number, the abs function accepts one complex argument and returns the magnitude of its complex argument as a positive real number.

acos()

Returns the inverse cosine, or arc cosine, in radians

Syntax

y = acos(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	(-∞:∞)	Integer, real, complex	Yes

Examples

```
a = acos(-1) returns 3.142
```

Defined in

Built in

See Also

asin() (expmeas), *atan()* (expmeas), *atan2()* (expmeas)

Note
The function name acos() is used for more than one type of expression. For comparison, see the Simulator Expression *acos() Expression* (expsim) and the AEL Function *acos() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

Notes/Equations

Returned value ranges from 0 to pi.

acosh()

Returns the inverse hyperbolic cosine

Syntax

y = acosh(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	(-∞:∞)	Integer, real, complex	Yes

Examples

```
a = acosh(1.5) returns 0.962
```

Defined in

Built in

See Also

acos() (expmeas), *asin()* (expmeas), *atan()* (expmeas), *atan2()* (expmeas)

Note
The function name acosh() is used for more than one type of expression. For comparison, see the Simulator Expression *acosh() Expression* (expsim) and the AEL Function *acosh() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

acot()

Returns the inverse cotangent

Syntax

y = acot(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	(-∞:∞)	Integer, real, complex	Yes

Examples

```
a = acot(1.5) returns 0.588
```

Defined in

Built in

See Also

asin() (expmeas), *atan()* (expmeas), *atan2()* (expmeas)

Note
The function name acot() is used for more than one type of expression. For comparison, see the AEL Function *acot() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

acoth()

Returns the inverse hyperbolic cotangent

Syntax

y = acoth(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	(-∞:∞)	Integer, real, complex	Yes

Examples

```
a = acoth(1.5) returns 0.805
```

Defined in

Built in

See Also

acot() (expmeas), *asin()* (expmeas), *atan()* (expmeas), *atan2()* (expmeas)

Note
The function name acoth() is used for more than one type of expression. For comparison, see the AEL Function *acoth() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

asin()

Returns the inverse sine, or arc sine, in radians

Syntax

y = asin(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	(-∞:∞)	Integer, real, complex	Yes

Examples

```
a = asin(-1) returns -1.571
```

Defined in

Built in

See Also

acos() (expmeas), *atan()* (expmeas), *atan2()* (expmeas)

Note
The function name asin() is used for more than one type of expression. For comparison, see the Simulator Expression *asin() Expression* (expsim) and the AEL Function *asin() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

asinh()

Returns the inverse hyperbolic sine

Syntax

y = asinh(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	(-∞:∞)	Integer, real, complex	Yes

Examples

```
a = asinh(.5) returns 0.481
```

Defined in

Built in

See Also

asin() (expmeas), *acos()* (expmeas), *atan()* (expmeas), *atan2()* (expmeas)

> **Note**
> The function name asinh() is used for more than one type of expression. For comparison, see the Simulator Expression *asinh() Expression* (expsim). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

atan2()

Returns the inverse tangent, or arc tangent, of the rectangular coordinates y and x

Syntax

y = atan2(y, x)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	(-∞:∞)	Integer, real, complex	Yes
y	number	None	(-∞:∞)	Integer, real, complex	Yes

Examples

```
a = atan2(1, 0) returns 1.571
```

Defined in

Built in

See Also

acos() (expmeas), *asin()* (expmeas), *atan()* (expmeas)

Notes/Equations

Returned value range is -pi to pi. atan2(0,0) returns -pi/2.

Note
The function name atan2() is used for more than one type of expression. For comparison, see the Simulator Expression *atan2() Expression* (expsim) and the AEL Function *atan2() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

atan()

Returns the inverse tangent, or arc tangent, in radians

Syntax

y = atan(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	(-∞:∞)	Integer, real, complex	Yes

Examples

```
a = atan(-1) returns -0.785
```

Defined in

Built in

See Also

acos() (expmeas), *asin()* (expmeas), *atan2()* (expmeas)

Notes/Equations

Returned value range is -pi/2 to pi/2.

Note
The function name atan() is used for more than one type of expression. For comparison, see the Simulator Expression *atan() Expression* (expsim) and the AEL Function *atan() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

atanh()

Returns the inverse hyperbolic tangent

Syntax

y = atanh(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	(-∞:∞)	Integer, real, complex	Yes

Examples

```
a = atanh(.5) returns 0.549
```

Defined in

Built in

See Also

acos() (expmeas), *asin()* (expmeas), *atan()* (expmeas), *atan2()* (expmeas)

Notes/Equations

Returned value ranges from 0 to pi.

Note
The function name atanh() is used for more than one type of expression. For comparison, see the Simulator Expression *atanh() Expression* (expsim) and the AEL Function *atanh() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

ceil()

Given a real number, returns the smallest integer not less than its argument; that is, its argument rounded to the next highest number.

Syntax

y = ceil(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	(-∞:∞)	real	Yes

Examples

```
a = ceil(5.27) returns 6
```

Defined in

Built in

Note
The function name ceil() is used for more than one type of expression. For comparison, see the Simulator Expression *ceil() Expression* (expsim) and the AEL Function *ceil() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

cint()

Given a non-integer real number, returns a rounded integer.

Syntax

y = cint(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	(-∞:∞)	real	Yes

Examples

```
a = cint(45.6) returns 46
a = cint(-10.7) returns -11
```

Defined in

Built in

See Also

abs() (expmeas), *exp()* (expmeas), *float()* (expmeas), *int()* (expmeas), *log()* (expmeas), *log10()* (expmeas), *pow()* (expmeas), *sgn()* (expmeas), *sqrt()* (expmeas)

Notes/Equations

0.5 rounds up, -0.5 rounds down (up in magnitude).

Note
The function name cint() is used for more than one type of expression. For comparison, see the AEL Function *cint() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

cmplx()

Returns complex number given real and imaginary

Syntax

y = cmplx(x, y)

Arguments

Name	Description	Default	Range	Type	Required
x	real part of complex number	None	(-∞:∞)	Integer, Real	Yes
y	imaginary part of complex number	None	(-∞:∞)	Integer, Real	Yes

Examples

```
a = cmplx(2, -1) returns 2 - 1j
```

Defined in

Built in

See Also

complex() (expmeas), *imag()* (expmeas), *real()* (expmeas)

> **Note**
> The function name cmplx() is used for more than one type of expression. For comparison, see the AEL Function *cmplx() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

complex()

Returns complex number given real and imaginary

Syntax

y = complex(x, y)

Arguments

Name	Description	Default	Range	Type	Required
x	real part of complex number	None	(-∞:∞)	Integer, Real	Yes
y	imaginary part of complex number	None	(-∞:∞)	Integer, Real	Yes

Examples

```
a = complex(2, -1) returns 2 - 1j
```

Defined in

$HPEESOF_DIR/expressions/ael/elementary_fun.ael

See Also

cmplx() (expmeas), *imag()* (expmeas), *real()* (expmeas)

Note
The function name complex() is used for more than one type of expression. For comparison, see the Simulator Expression *complex() Expression* (expsim). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

conj()

Returns the conjugate of a complex number

Syntax

y = conj(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	(-∞:∞)	complex	Yes

Examples

```
a = conj(3-4*j) returns 3.000 + j4.000
or 5.000 / 53.130 in magnitude / degrees
```

Defined in

Built in

See Also

mag() (expmeas)

Note
The function name conj() is used for more than one type of expression. For comparison, see the Simulator Expression *conj() Expression* (expsim) and the AEL Function *conj() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

convBin()

Returns a binary string of an integer with n-digits

Syntax

y = convBin(val, num)

Arguments

Name	Description	Default	Range	Type	Required
val	integer to be converted to a binary string	None	(-∞:∞)	Integer, real	Yes
num	number of digits in the binary string	None	[0:∞)	Integer	Yes

Examples

```
a = convBin(1064, 8) returns 00101000
```

Defined in

Built in

See Also

convHex() (expmeas), *convInt()* (expmeas), *convOct()* (expmeas)

Note
The function name convBin() is used for more than one type of expression. For comparison, see the AEL Function *convBin() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

convHex()

Returns a hexadecimal string of an integer with n-digits

Syntax

y = convHex(val, num)

Arguments

Name	Description	Default	Range	Type	Required
val	integer to be converted to a hexadecimal string	None	(-∞:∞)	Integer, real	Yes
num	number of digits in the hexadecimal string	None	[0:∞)	Integer	Yes

Examples

```
a = convHex(1064, 8) returns 00000428
```

Defined in

Built in

See Also

convBin() (expmeas), *convOct()* (expmeas), *convInt()* (expmeas)

Note
The function name convHex() is used for more than one type of expression. For comparison, see the AEL Function *convHex() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

convInt()

Returns an integer of a binary, octal or hexadecimal number

Syntax

y = convInt(val, base)

Arguments

Name	Description	Default	Range	Type	Required
val	string representation of the binary, octal or hexadecimal number to be converted	None	None	String	Yes
base	base of the conversion	None	2,8,16 †	Integer	Yes

† base values: 2:binary, 8:octal, 16:hexadecimal

Examples

```
b2I = convInt("11100", 2) returns 28
o2I = convInt("34", 8) returns 28
h2I = convInt("1c", 16) returns 28
```

Defined in

Built in

See Also

convBin() (expmeas), *convHex()* (expmeas), *convOct()* (expmeas)

convOct()

Returns an octal string of an integer with n-digits

Syntax

y = convOct(val, num)

Arguments

Name	Description	Default	Range	Type	Required
val	integer to be converted to a octal string	None	(-∞:∞)	Integer, real	Yes
num	number of digits in the octal string	None	[0:∞)	Integer	Yes

Examples

```
a = convOct(1064, 8) returns 00002050
```

Defined in

Built in

See Also

convBin() (expmeas), *convHex()* (expmeas), *convInt()* (expmeas)

Note
The function name convOct() is used for more than one type of expression. For comparison, see the the AEL Function *convOct() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

cos()

Returns the cosine

Syntax

y = cos(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number in radians	None	(-∞:∞)	Integer, real, complex	Yes

Examples

```
y = cos(pi/3) returns 0.500
```

Defined in

Built in

See Also

sin() (expmeas), *tan()* (expmeas)

Note
The function name cos() is used for more than one type of expression. For comparison, see the Simulator Expression *cos() Expression* (expsim) and the AEL Function *cos() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

cosh()

Returns the hyperbolic cosine

Syntax

y = cosh(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number in radians	None	(-∞:∞)	Integer, real, complex	Yes

Examples

```
y = cosh(0) returns 1
y = cosh(1) returns 1.543
```

Defined in

Built in

See Also

sinh() (expmeas), *tanh()* (expmeas)

Note
The function name cosh() is used for more than one type of expression. For comparison, see the Simulator Expression *cosh() Expression* (expsim) and the AEL Function *cosh() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

cot()

Returns the cotangent

Syntax

y = cot(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	(-∞:∞)	Integer, real, complex	Yes

Examples

```
a = cot(1.5) returns 0.071
```

Defined in

Built in

See Also

tan() (expmeas), *tanh()* (expmeas)

Note
The function name cot() is used for more than one type of expression. For comparison, see the Simulator Expression *cot() Expression* (expsim) and the AEL Function *cot() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

coth()

Returns the hyperbolic cotangent

Syntax

y = coth(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	(-∞:∞)	Integer, real, complex	Yes

Examples

```
a = coth(1.5) returns 1.105
```

Defined in

Built in

See Also

cot() (expmeas), *tan()* (expmeas), *tanh()* (expmeas)

Note
The function name coth() is used for more than one type of expression. For comparison, see the Simulator Expression *coth() Expression* (expsim) and the AEL Function *coth() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

cum_prod()

Returns the cumulative product

Syntax

y = cum_prod(x)

Arguments

Name	Description	Default	Range	Type	Required
x	data to find cumulative product	None	(∞:∞)	Integer, real or complex	Yes

Examples

```
y = cum_prod(1) returns 1.000
y = cum_prod([1, 2, 3]) returns [1.000, 2.000, 6.000]
y = cum_prod([i, i]) returns [i, i2]
```

Defined in

Built in

See Also

cum_sum() (expmeas), *max()* (expmeas), *mean()* (expmeas), *min()* (expmeas), *prod()* (expmeas), *sum()* (expmeas)

cum_sum()

Returns the cumulative sum

Syntax

y = cum_sum(x)

Arguments

Name	Description	Default	Range	Type	Required
x	data to find cumulative sum	None	(-∞:∞)	Integer, real or complex	Yes

Examples

```
y = cum_sum([1, 2, 3]) returns [1.000, 3.000, 6.000]
y = cum_sum([i, i] returns [i,2i]
```

Defined in

Built in

See Also

cum_prod() (expmeas), *max()* (expmeas), *mean()* (expmeas), *min()* (expmeas), *prod()* (expmeas), *sum()* (expmeas)

db()

Returns the decibel measure of a voltage ratio

Syntax

y = db(r, z1, z2)

Arguments

Name	Description	Default	Range	Type	Required
r	voltage ratio (vOut/vIn)	None	(-∞:∞)	Integer, real, complex	Yes
z1	source impedance	50.0	(-∞:∞)	Integer, real, complex	No
z2	load impedance	50.0	(-∞:∞)	Integer, real, complex	No

Examples

```
y = db(100) returns 40
y = db(8-6*j) returns 20
```

Defined in

Built in

See Also

dbm() (expmeas), *pae()* (expmeas)

Notes/Equations

dbValue = 20 log(mag(r)) - 10 log(zOutfactor/zInfactor)
zOutfactor = mag(z2)**2 / real (z2)
zInfactor = mag(z1)**2 / real (z1).

> **Note**
> The function name db() is used for more than one type of expression. For comparison, see the Simulator Expression *db() Expression* (expsim) and the AEL Function *dB() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

dbm()

Returns the decibel measure of a voltage referenced to a 1 milliwatt signal

Syntax

y = dbm(v, z)

Arguments

Name	Description	Default	Range	Type	Required
v	voltage (the peak voltage)	None	(-∞:∞)	Integer, real, complex	Yes
z	impedance	50.0	(-∞:∞)	Integer, real, complex	No

Examples

```
y = dbm(100) returns 50
y = dbm(8-6*j) returns 30
```

Defined in

Built in

See Also

db() (expmeas), *pae()* (expmeas)

Notes/Equations

The voltage is assumed to be a peak value. Signal voltages stored in the dataset from AC and harmonic balance simulations are in peak volts. However, noise voltages obtained from AC and HB simulations are in rms volts. Using the dbm() function with noise voltages will yield a result that is 3 dB too low unless the noise voltage is first converted to peak:
noise_power = dbm(vout.noise * sqrt(2));

Understanding the dbm() Function

Given a power Po in Watts, the power in *dB* is:

Po_dBW	=	10*log(mag(Po/(1 W)))

while the power in *dBm* is:

Po_dBm	=	10*log(mag(Po/(1 mW)))
=	10*log(mag(Po/(1 W)))+ 30	
=	Po_dB + 30	

Given a voltage Vo in Volts, the voltage in *dB* is:

V_dBV	=	20*log(mag((Vo/(1 V)))

This is the *db()* (expmeas) function - voltage in *dB* relative to 1V. Although dB is a dimensionless quantity, it is normal to attach dB to a value in order to differentiate it from the absolute value.

Given a real impedance Zo, the power-voltage relation is:

Po	=	$(Vo)^2/(2*Zo)$

Using the above, Po in *dBm* is then:

Po_dBm	=	10*log(mag(Po/(1 W))) + 30
=	$10*log(mag((Vo/(1 V))^2/(2*Zo/(1 Ohm)))) + 30$	
=	$10*log(mag((Vo/(1 V))^2)) - 10*log(mag(2*Zo/(1 Ohm))) + 30$	
=	20*log(mag(Vo/(1 V))) - 10*log(mag(Zo/(50 Ohm))) + 10	

This is the *dbm()* function - voltage in dBm in a Zo environment.

Note
The function name dbm() is used for more than one type of expression. For comparison, see the Simulator Expression *dbm() Expression* (expsim) and the AEL Function *dBm() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

dbmtow()

Converts dBm to watts

Syntax

wValue = dbmtow(P)

Arguments

Name	Description	Default	Range	Type	Required
P	power expressed in dBm	None	[0:∞)	Real	Yes

Examples

```
y = dbmtow(0) returns .001 W
y = dbmtow(-10) returns 1.000E-4 W
```

Defined in

$HPEESOF_DIR/expressions/ael/elementary_fun.ael

See Also

dbm() (expmeas), *wtodbm()* (expmeas)

Note
The function name dbmtow() is used for more than one type of expression. For comparison, see the Simulator Expression *dbmtow() Expression* (expsim). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

deg()

Converts radians to degrees

Syntax

y = deg(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number in radians	None	(-∞:∞)	Integer, Real	Yes

Examples

```
y = deg(1.5708) returns 90
y = deg(pi) returns 180
```

Defined in

Built in

See Also
rad() (expmeas)

Note
The function name deg() is used for more than one type of expression. For comparison, see the Simulator Expression *deg() Expression* (expsim) and the AEL Function *deg() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

diagonal()

Returns the diagonal of a square matrix as a matrix

Syntax

y = diagonal(Matrix)

Arguments

Name	Description	Default	Range	Type	Required
Matrix	square matrix to find the diagonal	None	(-∞:∞)	Integer, Real or Complex	Yes

Examples

```
mat={{1,2,3},{4,5,6},{7,8,9}}
```

```
diag=diagonal(mat) returns {1,5,9}
For a 2-port S-parameter analysis of 10 freq points:
diagS=diagonal(S) would return S11 and S22 for each frequency point
```

Defined in

Built In

See Also

transpose() (expmeas), *inverse()* (expmeas)

diff()

Calculates the simple numerical first derivative. Can be used to calculate group delay.

Syntax

y = diff(data, pad)

Arguments

Name	Description	Default	Range	Type	Required
data	data to find numerical derivative	None	(-∞:∞)	Real, Complex	Yes
pad	pad the differentiated data with an extra value	0	[0:1] †	Integer	No

† If pad is 1, then the differentiated data is padded with an extra value (last value of differentiated data) to make it the same length as the data to be differentiated. If 0 (default) then the length of the differentiated data is one less than the length of data to be differentiated.

Examples

```
group_delay = -diff(unwrap(phaserad(S21),pi))/(2*pi)
```

Defined in

$HPEESOF_DIR/expressions/ael/elementary_fun.ael

See Also

dev_lin_phase() (expmeas), *integrate()* (expmeas), *phasedeg()* (expmeas), *phaserad()* (expmeas), *ripple()* (expmeas), *unwrap()* (expmeas)

Notes/Equations

This function calculates the first derivative of the dependent data with respect to the inner independent value i.e. dy/dx. The function uses the simple forward finite-divided-difference formulas of 2 values. The error is O(h), where h is the independent step size. The error decreases with smaller values of h. If the data to be differentiated does not have an explicit independent-name, the differentiated data is given an independent name "diffX'.

erf()

Calculates the error function, the area under the Gaussian curve exp(-x**2)

Syntax

y = erf(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	(-∞:∞)	Integer, real	Yes

Examples

```
a = -erf(0.1) returns 0.112
a = -erf(0.2) returns 0.223
```

Defined in

Built in

See Also

erfc() (expmeas)

erfc()

Calculates the complementary error function, or 1 minus the error function. For large x, this can be calculated more accurately than the plain error function

Syntax

y = erfc(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	(-∞:∞)	Integer, real	Yes

Examples

```
a = erfc(0.1) returns 0.888
a = erfc(0.2) returns 0.777
```

Defined in

Built in

See Also

erf() (expmeas)

erfcinv()

Returns the inverse complementary error function as a real number

Syntax

y = erfcinv(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	[0:2] †	Integer, real	Yes

† For numbers outside the range, erfcinv returns <-+∞>. If val is 0, erfcinv returns <∞>. If val is 2, then <-∞>

Examples

```
res= erfcinv(0.5) returns 0.477
res= erfcinv(1.9) returns -1.163
```

Defined in

Built In

See Also

erfc() (expmeas)

erfinv()

Returns the inverse error function as a real number

Syntax

y = erfinv(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	[-1:1] †	Integer, real	Yes

† For numbers outside the range, erfinv returns <-+∞>. If x is +1, erfinv returns <∞>. If x is -1, then <-∞>.

Examples

```
res= erfinv(-0.4) returns -0.371
res= erfinv(0.8) returns 0.906
```

Defined in

Built In

See Also

erf() (expmeas)

exp()

The exponential function is used to calculate powers of e. Given a complex number, x, the exp(x) function calculates e to the power of x (i.e. ex)

Syntax

y = exp(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	(-∞:∞)	Integer, real, complex	Yes

Examples

```
a = exp(1) returns 2.71828
b = exp(1+j1) returns 1.469 + j*2.287
```

Defined in

Built in

See Also

abs() (expmeas), *cint()* (expmeas), *float()* (expmeas), *int()* (expmeas), *log()* (expmeas), *log10()* (expmeas), *pow()* (expmeas), *sgn()* (expmeas), *sqrt()* (expmeas)

Notes/Equations

If
```
x = a+j*b
```
then
```
ex = ea+j*b = (ea)*(ej*b) = (ea)*(cos(b)+j*sin(b))
```

> **Note**
> The function name exp() is used for more than one type of expression. For comparison, see the Simulator Expression *exp() Expression* (expsim) and the AEL Function *exp() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

fft()

Performs the discrete Fourier transform

Syntax

y = fft(x, length)

Arguments

Name	Description	Default	Range	Type	Required
x	data to be transformed	None	(-∞:∞)	Integer, real, complex	Yes
length	length of the transform	None	[1:∞)	Integer	Yes

Examples

```
fft([1, 1, 1, 1]) returns [4+0i, 0+0i]
fft([1, 0, 0, 0] returns [1+0i, 1+0i]
fft(1, 4) returns [1+0i, 1+0i]
```

Defined in

Built in

See Also

fs() (expmeas), *ts()* (expmeas)

Notes/Equations

The fft() function uses a high-speed radix-2 fast Fourier transform when the length of x is a power of two. fft(x, n) performs an n-point discrete Fourier transform, truncating x if length(x) > n and padding x with zeros if length(x) < n.

fft() uses a real transform if x is real and a complex transform if x is complex. If the length of x is not a power of two, then a mixed radix algorithm based on the prime factors of the length of x is used.
The fft() function is designed to work with uniformly spaced waveforms. If a non-uniform waveform is input, then the output spectrum will be incorrect. For non-uniformly spaced data, use the fs() function.

fix()

Takes a real number argument, truncates it, and returns an integer value

Syntax

y = fix(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	(-∞:∞)	Integer, real	Yes

Examples

```
a = fix(5.9) returns 5
```

Defined in

Built in

Note
The function name fix() is used for more than one type of expression. For comparison, see the AEL Function *fix() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

float()

Converts an integer to a real (floating-point) number

Syntax

y = float(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number to convert	None	(-∞:∞)	Integer	Yes

Examples

```
a = float(10)
```

Defined in

Built in

See Also

abs() (expmeas), *cint()* (expmeas), *int()* (expmeas), *log10()* (expmeas), *pow()* (expmeas), *sgn()* (expmeas), *sqrt()* (expmeas)

Note
The function name float() is used for more than one type of expression. For comparison, see the AEL Function *float() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

floor()

Returns the largest integer not more than its argument from a real number

Syntax

y = floor(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number to convert	None	(-∞:∞)	Real	Yes

Examples

```
a = floor(4.3) returns 4
```

Defined in

Built in

Note
The function name floor() is used for more than one type of expression. For comparison, see the Simulator Expression *floor() Expression* (expsim) and the AEL Function *floor() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

fmod()

Returns the remainder of the division of two real numbers

Syntax

y = fmod(fNum, fDenom)

Arguments

Name	Description	Default	Range	Type	Required
fNum	Value of numerator	None	(-∞:∞)	Integer, Real	Yes
fDenom	Value of denominator	None	(-∞:∞)	Integer, Real	Yes

Examples

```
y = fmod(4.2, 2.0) returns 0.2
```

Defined In

$HPEESOF_DIR/expressions/ael/elementary_fun.ael

> **Note**
> The function name fmod() is used for more than one type of expression. For comparison, see the Simulator Expression *fmod() Expression* (expsim). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

hypot()

Returns the hypotenuse

Syntax

y = hypot(xVal, yVal)

Arguments

Name	Description	Default	Range	Type	Required
xVal	Value of X	None	(-∞:∞)	Real, Complex	Yes
yVal	Value of Y	None	(-∞:∞)	Real, Complex	Yes

Examples

```
y = hypot(2, 1) returns 5
```

Defined In

$HPEESOF_DIR/expressions/ael/elementary_fun.ael

> **Note**
> The function name hypot() is used for more than one type of expression. For comparison, see the Simulator Expression *hypot() Expression* (expsim). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

identity()

Returns the identity matrix

Syntax

y = identity(rows, columns)

Arguments

Name	Description	Default	Range	Type	Required
rows	number of rows †	None	[i:∞)	Integer	Yes
columns	number of columns †	None	[i:∞)	Integer	No

† If one argument is supplied, then a square matrix is returned with ones on the diagonal and zeros elsewhere. If two arguments are supplied, then a matrix with size rows cols is returned, again with ones on the diagonal.

Examples

```
a = identity(2)
```

Defined in

Built in

See Also

ones() (expmeas), *zeros()* (expmeas)

im()

Returns the imaginary component of a complex number

Syntax

y = im(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	(-∞:∞)	complex	Yes

Examples

```
y = imag(1-1*j) returns -1.000
```

Defined in

Built in

See Also

imag() (expmeas), *cmplx()* (expmeas), *real()* (expmeas)

Note
The function name im() is used for more than one type of expression. For comparison, see the AEL Function *im() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

imag()

Returns the imaginary component of a complex number

Syntax

y = imag(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	(-∞:∞)	complex	Yes

Examples

```
y = imag(1-1*j) returns -1.000
```

Defined in

Built in

See Also

cmplx() (expmeas), *im()* (expmeas), *real()* (expmeas)

Note
The function name imag() is used for more than one type of expression. For comparison, see the Simulator Expression *imag() Expression* (expsim) and the AEL Function *imag() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

int()

Converts an real to an integer

Syntax

y = int(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number to convert	None	(-∞:∞)	Real	Yes

Examples

```
a = int(4.3) returns 4

b = int(334.235) returns integer 334
c = int(-.45e3) returns integer -450
```

Defined in

Built in

See Also

abs() (expmeas), *cint()* (expmeas), *exp()* (expmeas), *float()* (expmeas), *log10()* (expmeas), *pow()* (expmeas), *sgn()* (expmeas), *sqrt()* (expmeas)

Note
The function name int() is used for more than one type of expression. For comparison, see the Simulator Expression *int() Expression* (expsim) and the AEL Function *int() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

integrate()

Returns the integral of data

Syntax

y = integrate(data, start, stop, incr)

Arguments

Name	Description	Default	Range	Type	Required
data	data to be intergated	None	(-∞:∞)	Integer, real	Yes
start	starting value of the integration	first point in the data	(-∞:∞)	Integer, real	No
stop	stop value of the integration	last point in the data	(-∞:∞)	Integer, real †	No
incr	increment	(stop - start)/(# data points - 1)	[0:∞)	Integer, real	No

† stop can be an array.

Examples

```
x = [0::0.01::1.0]
y = vs(2*exp(-x*x) / sqrt(pi), x)
z= integrate(y, 0.1, 0.6, 0.001) returns 0.491
xx=[1::0.1::2]
yy=vs(sin(xx),xx)
Stop=[1.9,2.0]
intgYY=integrate(yy,1,Stop,0.1) returns [0.767, 0.958]
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

diff() (expmeas)

Notes/Equations

Returns the integral of data from start to stop with increment incr using the composite trapezoidal rule on uniform subintervals. The Stop limit can be an array of values. In this case, the function returns integration for limits [start, stop[0]], [start, stop[1]], etc.

If no incr parameter is given, then no increment (or no interpolation) will be performed.

interp()

Returns linearly interpolated data between start and stop with increment.

Syntax

y = interp(Data, Start, Stop, Increment)

Arguments

Name	Description	Default	Range	Type	Required
Data	Data to be interpolated	None	(-∞:∞)	Integer, Real	Yes
Start	Independent value specifying start	First data point	(-∞:∞)	Integer, Real	No
Stop	Independent value specifying stop	Last data point	(-∞:∞)	Integer, Real	No
Increment	Increment between interpolated data points	(Stop-Start)/(NumDataPoints-1)	(0:∞)	Integer, Real	No

Examples

```
x=[1::0.5::2]
y=vs(sin(x),x)
interpP=interp(y,1,1.5,0.1) returns
1.0 0.841
1.1 0.873
1.2 0.904
1.3 0.935
1.4 0.966
1.5 0.997
y = interp(Data, Start, Stop, Increment)
```

Defined in

Built in

See also

interpolate() (expmeas)

interpolate()

Interpolates data

Syntax

y = interpolate(InterpType,Data,iVar1,iVal1,iVar2,iVal2,....,iVarN,iValN)

Arguments

Name	Description	Default	Range	Type	Required
InterpType	Type of interpolation	None	"linear", "cubic", "spline"	String	Yes
Data	Data to interpolate	None	None	Integer, Real	Yes
iVar1, iVar2,...	Dimension of independent variable to interpolate	None	None	Integer	Yes
iVal1, iVal2,...	values to interpolate	None	None	Integer, real	Yes

Examples

```
linI = interpolate("linear", colY, 1,[1::0.5::4])
cubicI = interpolate("cubic", colY, 1,[1::0.2::2])
splineI = interpolate("spline", colY, 1,[-2::0.2::2])
For a 2-D data with 2 independents, the following interpolates dimension 1
(inner-most)
from 1 to 4 in steps of 0.5 and second dimension at 0.5:
linI = interpolate("linear", colY, 1,[1::0.5::4],2,0.5)
```

Defined in

Built in

See also

interp() (expmeas)

Notes/Equations

This function can interpolate data over multiple dimensions.

inverse()

Returns the inverse of a matrix

Syntax

y = inverse(Matrix)

Arguments

Name	Description	Default	Range	Type	Required
Matrix	square matrix to find the inverse	None	(-∞:∞)	Integer, Real or Complex	Yes

Examples

`inverse({{1, 2}, {3, 4})` returns `-2, 1}, {1.5, -0.5`}}

Defined in

Built in

See Also

diagonal() (expmeas), *transpose()* (expmeas)

jn()

Computes the Bessel function of the first kind and returns a real number

Syntax

y = jn(n, x)

Arguments

Name	Description	Default	Range	Type	Required
n	order	None	(-∞:∞)	Integer	Yes
x	value for which the Bessel value is to be found	None	(-∞:∞)	Real	Yes

Examples

```
jn0_15 = jn(0, 15) returns -0.014
jn1_xV = jn(1, 5.23) returns -0.344
jn10_15 = jn(10, 15) returns -0.09
```

Defined in

in-built

Note
The function name jn() is used for more than one type of expression. For comparison, see the Simulator Expression *jn() Expression* (expsim). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

ln()

Returns the natural logarithm (ln)

Syntax

y = ln(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	(-∞:∞)	Integer, real, complex	Yes

Examples

```
a = ln(e) returns 1
```

Defined in

Built in

See Also

abs() (expmeas), *cint()* (expmeas), *exp()* (expmeas), *float()* (expmeas), *int()* (expmeas), *pow()* (expmeas), *sgn()* (expmeas), *sqrt()* (expmeas)

Note
The function name ln() is used for more than one type of expression. For comparison, see the Simulator Expression *ln() Expression* (expsim) and the AEL Function *ln() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

log10()

Returns the base 10 logarithm of an integer or real number

Syntax

y = log10(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	(-∞:∞)	Integer, real, complex	Yes

Examples

```
a = log10(10) returns 1
a = log10(0+0i) returns NULL and an error message "log of zero"
```

Defined in

Built in

See Also

abs() (expmeas), *cint()* (expmeas), *exp()* (expmeas), *float()* (expmeas), *int()* (expmeas), *log()* (expmeas), *pow()* (expmeas), *sgn()* (expmeas), *sqrt()* (expmeas)

> **Note**
> The function name log10() is used for more than one type of expression. For comparison, see the Simulator Expression *log10() Expression* (expsim) and the AEL Function *log10() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

log()

Returns the base 10 logarithm of an integer or real number

Syntax

y = log(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	(-∞:∞)	Integer, real, complex	Yes

Examples

```
a = log(10) returns 1
a = log(0+0i) returns NULL and an error message "log of zero"
```

Defined in

Built in

See Also

abs() (expmeas), *cint()* (expmeas), *exp()* (expmeas), *float()* (expmeas), *int()* (expmeas), *log10()* (expmeas), *pow()* (expmeas), *sgn()* (expmeas), *sqrt()* (expmeas)

Note
The function name log() is used for more than one type of expression. For comparison, see the Simulator Expression *log() Expression* (expsim) and the AEL Function *log() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

mag()

Returns the magnitude

Syntax

y = mag(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	(-∞:∞)	Integer, real, complex	Yes

Examples

```
a = mag(3-4*j) returns 5.000
```

Defined in

Built in

See Also

conj() (expmeas)

Note
The function name mag() is used for more than one type of expression. For comparison, see the Simulator Expression *mag() Expression* (expsim) and the AEL Function *mag() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

max2()

Returns the larger value of two numeric values, or NULL if parameters are invalid

Syntax

y = max2(x, y)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	(-∞:∞)	Integer, real, complex	Yes
y	number	None	(-∞:∞)	Integer, real, complex	Yes

Examples

```
a = max2(1.5, -1.5) returns 1.500
```

Defined in

Built in

See Also

cum_prod() (expmeas), *cum_sum()* (expmeas), *max()* (expmeas), *mean()* (expmeas), *min()* (expmeas), *prod()* (expmeas), *sum()* (expmeas)

Note
The function name max2() is used for more than one type of expression. For comparison, see the AEL Function *max2() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

max()

Returns the maximum value.

Syntax

y = max(x)

Arguments

Name	Description	Default	Range	Type	Required
x	data to find max	None	(-∞:∞)	Integer, real or complex	Yes

Examples

```
a = max([1, 2, 3]) returns 3
```

Defined in

Built in

See Also

cum_prod() (expmeas), *cum_sum()* (expmeas), *max2()* (expmeas), *mean()* (expmeas), *min()* (expmeas), *prod()* (expmeas), *sum()* (expmeas)

Note
The function name max() is used for more than one type of expression. For comparison, see the Simulator Expression *max() Expression* (expsim). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

max_outer()

Computes the maximum across the outer dimension of two-dimensional data

Syntax

y = max_outer(data)

Arguments

Name	Description	Default	Range	Type	Required
data	2-dimensional data to find max	None	(-∞:∞)	Integer, Real, Complex	Yes

Examples

```
y = max_outer(data)
```

Defined in

$HPEESOF_DIR/expressions/ael/statistcal_fun.ael

See Also

fun_2d_outer() (expmeas), *mean_outer()* (expmeas), *min_outer()* (expmeas)

Notes/Equations

The max function operates on the inner dimension of two-dimensional data. The max_outer function just calls the fun_2d_outer function, with max being the applied operation. As an example, assume that a Monte Carlo simulation of an amplifier was run, with 151 random sets of parameter values, and that for each set the S-parameters were simulated over 26 different frequency points. S21 becomes a [151 Monte Carlo iteration X 26 frequency] matrix, with the inner dimension being frequency, and the outer dimension being Monte Carlo index. Now, assume that it is desired to know the maximum value of the S-parameters at each frequency. Inserting an equation max(S21) computes the maximum value of S21 at each Monte Carlo iteration. If S21 is simulated from 1 to 26 GHz, it computes the maximum value over this frequency range, which usually is not very useful. Inserting an equation max_outer(S21) computes the maximum value of S21 at each frequency.

min2()

Returns the lesser value of two numeric values, or NULL if parameters are invalid

Syntax

y = min2(x, y)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	(-∞:∞)	Integer, real, complex	Yes
y	number	None	(-∞:∞)	Integer, real, complex	Yes

Examples

```
a = min2(1.5, -1.5) returns -1.500
```

Defined in

Built in

See Also

cum_prod() (expmeas), *cum_sum()* (expmeas), *max()* (expmeas), *max2()* (ael), *mean()* (expmeas), *min()* (expmeas), *prod()* (expmeas), *sum()* (expmeas)

Note
The function name min2() is used for more than one type of expression. For comparison, see the AEL Function *min2() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

min()

Returns the minimum value.

Syntax

y = min(x)

Arguments

Name	Description	Default	Range	Type	Required
x	data to find min	None	(-∞:∞)	Integer, real or complex	Yes

Examples

```
a = min([1, 2, 3]) returns 1
```

Defined in

Built in

See Also

cum_prod() (expmeas), *cum_sum()* (expmeas), *max()* (expmeas), *max2()* (expmeas), *mean()* (expmeas), *prod()* (expmeas), *sum()* (expmeas)

> **Note**
> The function name min() is used for more than one type of expression. For comparison, see the Simulator Expression *min() Expression* (expsim). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

min_outer()

Computes the minimum across the outer dimension of two-dimensional data

Syntax

y = min_outer(data)

Arguments

Name	Description	Default	Range	Type	Required
data	2-dimensional data to find min	None	(-∞:∞)	Integer, Real, Complex	Yes

Examples

```
y = min_outer(data)
```

Defined in

$HPEESOF_DIR/expressions/ael/statistcal_fun.ael

See Also

fun_2d_outer() (expmeas), *max_outer()* (expmeas), *mean_outer()* (expmeas)

Notes/Equations

The min function operates on the inner dimension of two-dimensional data. The min_outer function just calls the fun_2d_outer function, with min being the applied operation. As an example, assume that a Monte Carlo simulation of an amplifier was run, with 151 random sets of parameter values, and that for each set the S-parameters were simulated over 26 different frequency points. S21 becomes a [151 Monte Carlo iteration X 26 frequency] matrix, with the inner dimension being frequency, and the outer dimension being Monte Carlo index. Now, assume that it is desired to know the minimum value of the S-parameters at each frequency. Inserting an equation min(S21) computes the minimum value of S21 at each Monte Carlo iteration. If S21 is simulated from 1 to 26 GHz, it computes the minimum value over this frequency range, which usually is not very useful.

Inserting an equation min_outer(S21) computes the minimum value of S21 at each frequency.

num()

Returns an integer that represents an ASCII numeric value of the first character in the specified string.

Syntax

y = num(str)

Arguments

Name	Description	Default	Range	Type	Required
str	string to convert to integer	None	None	String	Yes

Examples

```
a = num("/users/myhome/fullpath") returns 47
a = num("alpha") returns 97
```

Defined in

Built in

Note
The function name num() is used for more than one type of expression. For comparison, see the AEL Function *num() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

ones()

Returns ones matrix

Syntax

y = ones(rows, columns)

Arguments

Name	Description	Default	Range	Type	Required
rows	number of rows †	None	[i:∞)	Integer	Yes
columns	number of columns †	None	[i:∞)	Integer	No

† If only one argument is supplied, then a square matrix is returned. If two are supplied, then a matrix of ones with size rows X cols is returned.

Examples

```
a = ones(2) returns {{1, 1}, {1, 1}}
```

Defined in

Defined in

Built in

See Also

identity() (expmeas), *zeros()* (expmeas)

phase()

Phase in degrees

Syntax

y = phase(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	(-∞:∞)	Complex	Yes

Examples

```
a = phase(1*i) returns 90
a = phase(1+1i) returns 45
```

Defined in

Built-in

See Also

phaserad() (expmeas)

> **Note**
> The function name phase() is used for more than one type of expression. For comparison, see the Simulator Expression *phase() Expression* (expsim) and the AEL Function *phase() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

phasedeg()

Phase in degrees

Syntax

y = phasedeg(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	(-∞:∞)	Complex	Yes

Examples

```
a = phasedeg(1*i) returns 90
a = phasedeg(1+1i) returns 45
```

Defined in

Built-in

See Also

dev_lin_phase() (expmeas), *diff()* (expmeas), *phase()* (expmeas), *phaserad()* (expmeas), *ripple()* (expmeas), *unwrap()* (expmeas)

Note
The function name phasedeg() is used for more than one type of expression. For comparison, see the Simulator Expression *phasedeg() Expression* (expsim) and the AEL Function *phasedeg() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

phaserad()

Phase in radians

Syntax

y = phaserad(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	(-∞:∞)	Complex	Yes

Examples

```
a = phaserad(1*i) returns 1.5708
a = phaserad(1+1i) returns 0.785398
```

Defined in

Built in

See Also

dev_lin_phase() (expmeas), *diff()* (expmeas), *phase()* (expmeas), *phasedeg()* (expmeas),

ripple() (expmeas), *unwrap()* (expmeas)

Note
The function name phaserad() is used for more than one type of expression. For comparison, see the Simulator Expression *phaserad() Expression* (expsim) and the AEL Function *phaserad() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

polar()

Builds a complex number from magnitude and angle (in degrees)

Syntax

y = polar(mag, angle)

Arguments

Name	Description	Default	Range	Type	Required
mag	magnitude part of complex number	None	(-∞:∞)	Integer, Real	Yes
angle	angle part of complex number	None	(-∞:∞)	Integer, Real	Yes

Examples

```
a = polar(1, 90) returns 0+1i
a = polar(1, 45) returns 0.707107+0.707107i
```

Defined in

Built in

Note
The function name polar() is used for more than one type of expression. For comparison, see the Simulator Expression *polar() Expression* (expsim) and the AEL Function *polar() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

pow()

Raises a number to a given power

Syntax

yPow = pow(x, y)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	(-∞:∞)	Integer, real	Yes
y	exponent	None	(-∞:∞)	Integer, real	Yes

Examples

```
a = pow(4, 2) returns 16
a = pow(1+j*1,2+j*2) returns -0.266+j*0.32
```

Defined in

Built in

See Also

abs() (expmeas), *cint()* (expmeas), *exp()* (expmeas), *float()* (expmeas), *int()* (expmeas), *log10()* (expmeas), *sgn()* (expmeas), *sqrt()* (expmeas)

Note
The function name pow() is used for more than one type of expression. For comparison, see the Simulator Expression *pow() Expression* (expsim) and the AEL Function *pow() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

prod()

Returns the product

Syntax

y = prod(x)

Arguments

Name	Description	Default	Range	Type	Required
x	data to find product	None	(∞:∞)	Integer, real or complex	Yes

Examples

```
a = prod([1, 2, 3] returns 6
a = prod([4, 4, 4]) returns 64
```

Defined in

Built-in

See Also

sum() (expmeas)

rad()

Converts degrees to radians

Syntax

y = rad(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	(-∞:∞)	Integer, Real	Yes

Examples

```
a = rad(90) returns 1.5708
a = rad(45) returns 0.785398
```

Defined in

Built in

See Also

deg() (expmeas)

Note
The function name rad() is used for more than one type of expression. For comparison, see the Simulator Expression *rad() Expression* (expsim) and the AEL Function *rad() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

re()

Returns the real component of a complex number

Syntax

y = re(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	(-∞:∞)	complex	Yes

Examples

```
y = re(1-1*j) returns 1.000
```

Defined in

Built in

See Also

cmplx() (expmeas), *imag()* (expmeas), *real()* (expmeas)

Note
The function name re() is used for more than one type of expression. For comparison, see the AEL Function *re() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

real()

Returns the real component of a complex number

Syntax

y = real(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	(-∞:∞)	complex	Yes

Examples

```
y = real(1-1*j) returns 1.000
```

Defined in

Built in

See Also

cmplx() (expmeas), *imag()* (expmeas), *re()* (expmeas)

Note
The function name real() is used for more than one type of expression. For comparison, see the Simulator Expression *real() Expression* (expsim) and the AEL Function *real() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

rms()

Returns the root mean square value

Syntax

y = rms(Value)

Arguments

Name	Description	Default	Range	Type	Required
Value	Value to find RMS	None	(-∞:∞)	Integer, Real, Complex	Yes

Examples

```
rmsR = rms(2) returns 1.414
rmsR = rms(complex(3, 10)) returns 7.382/73.301
rmsR = rms( [1, 2, 3, 4, 5,] ) returns [0.707, 1.414, 2.121, 2.828, 3.536]
```

Defined in

$HPEESOF_DIR/expressions/ael/elementary_fun.ael

Notes/Equations

The rms() function calculates the root mean square value. If the data's inner independent is freq, and if frequency equals 0 (DC), then the function returns mag() rather than rms value.

round()

Rounds to the nearest integer

Syntax

y = round(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	(-∞:∞)	real	Yes

Examples

```
a = round(0.1) returns 0
a = round(0.5) returns 1
a = round(0.9) returns 1
a = round(-0.1) returns 0
a = round(-0.5) returns -1
a = round(-0.9) returns -1
```

Defined in

Built in

See Also

int() (expmeas)

Note
The function name round() is used for more than one type of expression. For comparison, see the AEL Function *round() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

sgn()

Returns the integer sign of an integer or real number, as either 1 or -1

Syntax

y = sgn(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	(-∞:∞)	Integer, real	Yes

Examples

```
a = sgn(-1) returns -1
a = sgn(1) returns 1
```

Defined in

Built in

See Also

abs() (expmeas), *cint()* (expmeas), *exp()* (expmeas), *float()* (expmeas), *int()* (expmeas), *log10()* (expmeas), *pow()* (expmeas), *sqrt()* (expmeas)

Note
The function name sgn() is used for more than one type of expression. For comparison, see the Simulator Expression *sgn() Expression* (expsim) and the AEL Function *sgn() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

sin()

Returns the sine

Syntax

y = sin(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number in radians	None	(-∞:∞)	Integer, real, complex	Yes

Examples

```
y = sin(pi/2) returns 1
```

Defined in

Built in

See Also

cos() (expmeas), *tan()* (expmeas)

Note
The function name sin() is used for more than one type of expression. For comparison, see the Simulator Expression *sin() Expression* (expsim) and the AEL Function *sin() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

sinc()

Returns the sinc of a number.

Syntax

y = sinc(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	(-∞:∞)	Integer, real, complex	Yes

Examples

```
a = sinc(0.5) returns 0.959
```

Defined in

Built in

See Also

sin() (expmeas)

Notes/Equations

The sinc function is defined as sinc(x) = sin(pi*x)/ (pi*x) and sinc(0)=1.

Note
The function name sinc() is used for more than one type of expression. For comparison, see the Simulator Expression *sinc() Expression* (expsim) and the AEL Function *sinc() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

sinh()

Returns the hyperbolic sine

Syntax

y = sinh(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	(-∞:∞)	Integer, real, complex	Yes

Examples

```
a = sinh(0) returns 0
a = sinh(1) returns 1.1752
```

Defined in

Built in

See Also

cosh() (expmeas), *tanh()* (expmeas)

Note
The function name sinh() is used for more than one type of expression. For comparison, see the Simulator Expression *sinh() Expression* (expsim) and the AEL Function *sinh() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

sqr()

Returns the square of a number

Syntax

y = sqr(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number to square	None	(-∞:∞)	Integer, real, complex	Yes

Examples

```
y = sqr(2) returns 4
```

Defined In

$HPEESOF_DIR/expressions/ael/elementary_fun.ael

sqrt()

Returns the square root of number

Syntax

y = sqrt(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	(-∞:∞)	Integer, real, complex	Yes

Examples

```
a = sqrt(4) returns 2
a = sqrt(2+j*1) returns 1.455+j*0.344
```

Defined in

Built in

See Also

abs() (expmeas), *cint()* (expmeas), *exp()* (expmeas), *float()* (expmeas), *int()* (expmeas), *log10()* (expmeas), *pow()* (expmeas), *sgn()* (expmeas)

Note
The function name sqrt() is used for more than one type of expression. For comparison, see the Simulator Expression *sqrt() Expression* (expsim) and the AEL Function *sqrt() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

step()

Returns 0, 0.5, or 1

Syntax

y= step(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	(-∞:∞)	Integer, real, complex	Yes

Examples

```
a = step(-1.5) returns 0.000
a = step(0) returns 0.500
a = step(1.5) returns 1.000
```

Defined in

Built in

Note
The function name step() is used for more than one type of expression. For comparison, see the Simulator Expression *step() Expression* (expsim) and the AEL Function *step() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

sum()

Returns the sum

Syntax

y = sum(x)

Arguments

Name	Description	Default	Range	Type	Required
x	data to find sum	None	(-∞:∞)	Integer, real or complex	Yes

Examples

```
a = sum([1, 2, 3]) returns 6
```

Defined in

Built in

See Also

max() (expmeas), *mean()* (expmeas), *min()* (expmeas)

Note
The function name sum() is used for more than one type of expression. For comparison, see the Measurement Expression *sum() Measurement* (expmeas). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

tan()

Returns the tangent

Syntax

y = tan(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number in radians	None	(-∞:∞)	Integer, real, complex	Yes

Examples

```
a = tan(pi/4) returns 1
a = tan(+/-pi/2) returns +/- 1.633E16
```

Defined in

Built in

See Also

cos() (expmeas), *sin()* (expmeas)

Note
The function name tan() is used for more than one type of expression. For comparison, see the Simulator Expression *tan() Expression* (expsim) and the AEL Function *tan() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

tanh()

Returns the hyperbolic tangent

Syntax

y = tanh(x)

Arguments

Name	Description	Default	Range	Type	Required
x	number in radians	None	(-∞:∞)	Integer, Real, complex	Yes

Examples

```
a = tanh(0) returns 0
a = tanh(1) returns 0.761594
a = tanh(-1) returns -0.761594
```

Defined in

Built in

See Also

cosh() (expmeas), *sinh()* (expmeas)

Note
The function name tanh() is used for more than one type of expression. For comparison, see the Simulator Expression *tanh() Expression* (expsim) and the AEL Function *tanh() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

transpose()

Returns the transpose of a matrix

Syntax

y = transpose(Matrix)

Arguments

Name	Description	Default	Range	Type	Required
Matrix	square matrix to find the transpose	None	(-∞:∞)	Integer, Real or Complex	Yes

Examples

```
a={{1, 2}, {3, 4}}
b=transpose(a)
returns {{1, 3}, {2, 4}}
```

Defined in

Built in

See Also

diagonal() (expmeas), *inverse()* (expmeas)

wtodbm()

Converts Watts to dBm and returns a real or complex number

Syntax

dbmVal = wtodbm(Value)

Arguments

Name	Description	Default	Range	Type	Required
Value	Value in Watts	None	(-∞:∞)	Real, Complex	Yes

Examples

```
wtodbm01_M=wtodbm(0.01) returns 10
wtodbm1_M=wtodbm(1) returns 30
wtodbmC_M=wtodbm(complex(10,2)) returns 40.094/1.225
```

Defined in

$HPEESOF_DIR/expressions/ael/elementary_fun.ael

See Also

dbmtow() (expmeas)

Note
The function name wtodbm() is used for more than one type of expression. For comparison, see the Simulator Expression *wtodbm() Expression* (expsim). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

xor()

Returns an integer that represents the exclusive OR between arguments

Syntax

yXor = xor(x, y)

Arguments

Name	Description	Default	Range	Type	Required
x	number	None	(-∞:∞)	Integer	Yes
y	number	None	(-∞:∞)	Integer	Yes

Examples

```
a = xor(16, 32) returns 48
```

Defined in

Built in

Note
The function name xor() is used for more than one type of expression. For comparison, see the AEL Function *xor() Function* (ael). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

zeros()

Returns zeros matrix

Syntax

y = zeros(rows, columns)

Arguments

Name	Description	Default	Range	Type	Required
rows	number of rows †	None	[i:∞)	Integer	Yes
columns	number of columns †	None	[i:∞)	Integer	Yes

† If only one argument is supplied, then a square matrix is returned. If two are supplied, then a matrix of zeros with size rows X cols is returned.

Examples

```
a=zeros(2) returns {{0, 0}, {0, 0}}
b=(2, 3) returns {{0, 0, 0}, {0, 0, 0}}
```

Defined in

Built in

See Also

identity() (expmeas), *ones()* (expmeas)

Signal Processing Functions

This section describes the signal processing functions in detail. The functions are listed in alphabetical order.

- *add rf()* (expmeas)
- *ber pi4dqpsk()* (expmeas)
- *ber qpsk()* (expmeas)
- *eye()* (expmeas)
- *eye amplitude()* (expmeas)
- *eye closure()* (expmeas)
- *eye fall time()* (expmeas)
- *eye height()* (expmeas)
- *eye rise time()* (expmeas)
- *spec power()* (expmeas)

add_rf()

Returns the sum of two Timed Complex Envelope signals defined by the triplet in-phase (real or I(t)) and quadrature-phase (imaginary or Q(t)) part of a modulated carrier frequency(Fc).

Syntax

y = add_rf(T1, T2)

Arguments

Name	Description	Default	Range	Type	Required
T1	Timed Complex Envelope signals at carrier frequencies Fc1	None	(-∞:∞)	Complex	Yes
T2	Timed Complex Envelope signals at carrier frequencies Fc2	None	(-∞:∞)	Complex	Yes

Examples

```
y= add_rf(T1, T2)
```

Defined in

$HPEESOF_DIR/expressions/ael/signal_proc_fun.ael

Notes/Equations

Used in Signal processing designs that output Timed Signals using Timed Sinks
This equation determines the sum of two Timed Complex Envelope at a new carrier frequency Fc3. Given Fc1 and Fc2 as the carrier frequencies of the two input waveforms, the output carrier frequency Fc3 will be the greater of the two.

ber_pi4dqpsk()

Returns the symbol probability of error versus signal-to-noise ratio per bit for pi/4 DQPSK

modulation.

Syntax

data = ber_pi4dqpsk(vIn, vOut, symRate, noise, samplingDelay, rotation, tranDelay, pathDelay)

Arguments

Name	Description	Default	Range	Type	Required
vIn	complex envelope voltage signals at the input node	None	(-∞:∞)	Complex	Yes
vOut	complex envelope voltage signals at the output node	None	(-∞:∞)	Complex	Yes
symRate	symbol rate (real) of the modulation signal	None	(0:∞)	Real	Yes
noise	RMS noise vector	None	(0:∞)	Real	Yes
samplingDelay	clock phase in seconds	calculated	[0:∞)	Real	Yes
rotation	carrier phase in radians	calculated	[0:∞)	Real	Yes
tranDelay	time in seconds that causes this time duration of symbols to be eliminated from the bit error rate calculation †	calculated	[0:∞)	Real	Yes
pathDelay	delay from input to output in seconds	calculated	[0:∞)	Real	Yes

† Usually the filters in the simulation have transient responses, and the bit error rate calculation should not start until these transient responses have finished.

Examples

```
y = ber_pi4dqpsk(videal[1], vout[1], 0.5e6, {0.1::-0.01::0.02})
```

Defined in

$HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael

See Also

ber_qpsk() (expmeas), *constellation()* (expmeas)

Notes/Equations

Used in Circuit Envelope and Signal Processing simulations.
The arguments vIn and vOut usually come from a circuit envelope simulation, while noise usually comes from a harmonic balance simulation, and is assumed to be additive white Gaussian. It can take a scalar or vector value. The function uses the quasi-analytic approach for estimating BER: for each symbol, Eb / N0 and BER are calculated analytically; then the overall BER is the average of the BER values for the symbols.
Note that ber_pi4dqpsk returns a list of data:
data[0]= symbol probability of error versus Eb / N0
data[1]= path delay in seconds

data[2]= carrier phase in radians
data[3]= clock phase in seconds
data[4]= complex(Isample, Qsample)

ber_qpsk()

Returns the symbol probability of error versus signal-to-noise ratio per bit for QPSK modulation

Syntax

data = ber_qpsk(vIn, vOut, symRate, noise{, samplingDelay, rotation, tranDelay, pathDelay})

Arguments

Name	Description	Default	Range	Type	Required
vIn	complex envelope voltage signals at the input node	None	(-∞:∞)	Complex	Yes
vOut	complex envelope voltage signals at the output node	None	(-∞:∞)	Complex	Yes
symRate	symbol rate (real) of the modulation signal	None	(0:∞)	Real	Yes
noise	RMS noise vector	None	(0:∞)	Real	Yes
samplingDelay	clock phase in seconds	calculated	[0:∞)	Real	Yes
rotation	carrier phase in radians	calculated	[0:∞)	Real	Yes
tranDelay	time in seconds that causes this time duration of symbols to be eliminated from the bit error rate calculation †	calculated	[0:∞)	Real	Yes
pathDelay	delay from input to output in seconds	calculated	[0:∞)	Real	Yes

† Usually the filters in the simulation have transient responses, and the bit error rate calculation should not start until these transient responses have finished.

Examples

```
y = ber_qpsk(videal[1], vout[1], 1e6, {0.15::-0.01::0.04})
```

Defined in

$HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael

See Also

ber_pi4dqpsk() (expmeas), *constellation()* (expmeas)

Notes/Equations

Used in Circuit Envelope and Signal Processing simulations.
The arguments vIn and vOut usually come from a circuit envelope simulation, while noise usually comes from a harmonic balance simulation, and is assumed to be additive white

Gaussian. It can take a scalar or vector value. The function uses the quasi-analytic approach for estimating BER: for each symbol, Eb / N0 and BER are calculated analytically; then the overall BER is the average of the BER values for the symbols.
Note that ber_qpsk returns a list of data:
data[0]= symbol probability of error versus Eb / N0
data[1]= path delay in seconds
data[2]= carrier phase in radians
data[3]= clock phase in seconds
data[4]= complex(Isample, Qsample)

eye()

Creates data for an eye diagram plot

Syntax

y = eye(data, symbolRate, Cycles, Delay)

Arguments

Name	Description	Default	Range	Type	Required
data	either numeric data or a time domain waveform typically from the I or Q data channel	None	(-∞:∞)	Complex	Yes
symbolRate	symbol rate of the channel. For numeric data, the symbol rate is the reciprocal of the number of points in one cycle; for a waveform, it is the frequency	None	(0:∞)	Real	Yes
Cycles	number of cycles to repeat	1	[1:∞)	Integer	No
Delay	sampling delay	0	[0:∞)	Integer, Real	No

Examples

```
y = eye(I_data, symbol_rate)
```

Defined in

Built in

See Also

constellation() (expmeas), *eye_amplitude()* (expmeas), *eye_closure()* (expmeas), *eye_fall_time()* (expmeas), *eye_height()* (expmeas), *eye_rise_time()* (expmeas)

eye_amplitude()

Returns eye amplitude

Syntax

y = eye_amplitude(Vout_time, Delay, BitRate, SamplePoint, WindowPct)

Arguments

Name	Description	Default	Range	Type	Required
Vout_time	time domain voltage waveform	None	(-∞:∞)	Complex	Yes
Delay	used to remove initial transient in the eye diagram and is expressed in time units	None	[0:∞)	Real	Yes
BitRate	bit rate of the channel and is expressed in frequency units	None	[0:∞)	Real	Yes
SamplePoint	marker name placed on the eye diagram measurement. The independent value of this marker is used to determine the symbol offset.	None	None	string	Yes
WindowPct	used to determine level '1' and level '0' of the time domain waveform and its typical value is 0.2.	None	[0:1]	Real	Yes

Examples

```
Eye_amp = eye_amplitude(vout,12 ps,10 GHz, m1, 0.2)
```

Defined in

$HPEESOF_DIR/expressions/ael/DesignGuide_fun.ael

See Also

cross_hist() (expmeas), *eye()* (expmeas), *eye_closure()* (expmeas), *eye_fall_time()* (expmeas), *eye_height()* (expmeas), *eye_rise_time()* (expmeas)

Notes/Equations

The eye_amplitude() function essentially takes the vertical histogram of the eye voltages and subtracts the "0" level mean from "1" level mean within a given measurement window.

eye_closure()

Returns eye closure

Syntax

y = eye_closure(Vout_time, Delay, BitRate, SamplePoint, WindowPct)

Arguments

Name	Description	Default	Range	Type	Required
Vout_time	time domain voltage waveform	None	(-∞:∞)	Complex	Yes
Delay	used to remove initial transient in the eye diagram and is expressed in time units	None	[0:∞)	Real	Yes
BitRate	bit rate of the channel and is expressed in frequency units	None	[0:∞)	Real	Yes
SamplePoint	marker name placed on the eye diagram measurement. The independent value of this marker is used to determine the symbol offset.	None	None	string	Yes
WindowPct	used to determine level '1' and level '0' of the time domain waveform and its typical value is 0.2.	None	[0:1]	Real	Yes

Examples

```
Eye_Close = eye_closure(vout,12 ps,10 GHz, m1, 0.2)
```

Defined in

$HPEESOF_DIR/expressions/ael/DesignGuide_fun.ael

See Also

cross_hist() (expmeas), *eye()* (expmeas), *eye_amplitude()* (expmeas), *eye_fall_time()* (expmeas), *eye_height()* (expmeas), *eye_rise_time()* (expmeas)

Notes/Equations

Computes the ratio of eye height to eye amplitude to provide eye closure.

eye_fall_time()

Returns eye fall time

Syntax

y = eye_fall_time(Vout_time,Delay,BitRate,SamplePoint,WindowPct)

Arguments

Name	Description	Default	Range	Type	Required
Vout_time	time domain voltage waveform	None	(-∞:∞)	Complex	Yes
Delay	used to remove initial transient in the eye diagram and is expressed in time units	None	[0:∞)	Real	Yes
BitRate	bit rate of the channel and is expressed in frequency units	None	[0:∞)	Real	Yes
SamplePoint	marker name placed on the eye diagram measurement. The independent value of this marker is used to determine the symbol offset.	None	None	string	Yes
WindowPct	used to determine level '1' and level '0' of the time domain waveform and its typical value is 0.2.	None	[0:1]	Real	Yes

Examples

```
Eye_Fall = eye_fall_time(vout,12 ps,10 GHz, m1, 0.2)
```

Defined in

$HPEESOF_DIR/expressions/ael/DesignGuide_fun.ael

See Also

cross_hist() (expmeas), *eye()* (expmeas), *eye_amplitude()* (expmeas), *eye_closure()* (expmeas), *eye_height()* (expmeas), *eye_rise_time()* (expmeas)

Notes/Equations

Computes 20% - 80% fall time of a time domain waveform.

eye_height()

Returns eye height

Syntax

y = eye_height(Vout_time, Delay, BitRate, SamplePoint, WindowPct)

Arguments

Name	Description	Default	Range	Type	Required
Vout_time	time domain voltage waveform	None	(-∞:∞)	Complex	Yes
Delay	used to remove initial transient in the eye diagram and is expressed in time units	None	[0:∞)	Real	Yes
BitRate	bit rate of the channel and is expressed in frequency units	None	[0:∞)	Real	Yes
SamplePoint	marker name placed on the eye diagram measurement. The independent value of this marker is used to determine the symbol offset.	None	None	string	Yes
WindowPct	used to determine level '1' and level '0' of the time domain waveform and its typical value is 0.2.	None	[0:1]	Real	Yes

Examples

```
Eye_Ht = eye_height(vout,12 ps,10 GHz, m1, 0.2)
```

Defined in

$HPEESOF_DIR/expressions/ael/DesignGuide_fun.ael

See Also

cross_hist() (expmeas), *eye()* (expmeas), *eye_amplitude()* (expmeas), *eye_closure()* (expmeas), *eye_fall_time()* (expmeas), *eye_rise_time()* (expmeas)

Notes/Equations

The eye_height() function essentially takes the vertical histogram of the eye voltages and computes inner bounds of the eye opening.

eye_rise_time()

Returns eye rise time

Syntax

y = eye_rise_time(Vout_time,Delay,BitRate,SamplePoint,WindowPct)

Arguments

Name	Description	Default	Range	Type	Required
Vout_time	time domain voltage waveform	None	(-∞:∞)	Complex	Yes
Delay	used to remove initial transient in the eye diagram and is expressed in time units	None	[0:∞)	Real	Yes
BitRate	bit rate of the channel and is expressed in frequency units	None	[0:∞)	Real	Yes
SamplePoint	marker name placed on the eye diagram measurement. The independent value of this marker is used to determine the symbol offset.	None	None	string	Yes
WindowPct	used to determine level '1' and level '0' of the time domain waveform and its typical value is 0.2.	None	[0:1]	Real	Yes

Examples

```
Eye_Rise = eye_rise_time(vout,12 ps,10 GHz, m1, 0.2)
```

Defined in

$HPEESOF_DIR/expressions/ael/DesignGuide_fun.ael

See Also

cross_hist() (expmeas), *eye()* (expmeas), *eye_amplitude()* (expmeas), *eye_closure()* (expmeas), *eye_fall_time()* (expmeas), *eye_height()* (expmeas)

Notes/Equations

Computes 20% - 80% rise time of a time domain waveform.

spec_power()

Returns the integrated signal power (dBm) of a spectrum

Syntax

y = spec_power(spectralData{, lowerFrequencyLimit, upperFrequencyLimit})

Arguments

Name	Description	Default	Range	Type	Required
spectralData	spectral data in dBm	None	None	Real	Yes
lowerFrequencyLimit	lower frequency limit to be used in calculating the integrated power	min(indep(spectralData))	(-∞:∞)	Real	No
upperFrequencyLimit	upper frequency limit to be used in calculating the integrated power	max(indep(spectralData))	[lowerFrequencyLimit:∞)	Real	No

Examples

```
total_power = spec_power(dBm(Mod_Spectrum), 60 MHz, 71 MHz)
where Mod_Spectrum is the instance name of a SpectrumAnalyzer sink component,
will return the integrated power between 60 and 71 MHz.
total_power = spec_power(dBm(fs(Vout[1])), indep(m1), indep(m2))
where Vout is a named node in a Circuit Envelope simulation, will return the
integrated power between markers 1 and 2.
```

Defined in

$HPEESOF_DIR/expressions/ael/signal_proc_fun.ael

Notes/Equations

Used in Circuit Envelope and Signal Processing simulations.
This expression can be used with spectral data of up to 4 dimensions (frequency should be the inner dimension).

The spec_power() function returns the power (in dBm) of a spectrum integrated between the lower and upper frequency limits specified. If no lower (upper) limit is specified, the lowest (highest) frequency in the spectral data is used instead.

The input spectral data must be in dBm. Spectral data can be generated in several different ways, such as applying the fs() expression on voltage or current time domain data or using the SpectrumAnalyzer sink component.

The fs() expression returns the voltage or current spectrum of the input data and so the dBm() expression should be applied to the fs() output before it is passed to the spec_power() expression. The frequency axis values in the spectral data returned by fs() are in Hz and so if lower and/or upper frequency limits are to be passed to spec_power(),

they should be specified in Hz.
Similarly, the SpectrumAnalyzer sink component returns the voltage spectrum of the input signal and so the dBm() expression should be applied to the spectral data generated by SpectrumAnalyzer before it is passed to the spec_power() expression. The frequency axis values in the spectral data generated by SpectrumAnalyzer are in Hz, so if lower and/or upper frequency limits are to be passed to spec_power(), they should be specified in Hz.

S-Parameter Analysis Functions for Measurement Expressions

This section describes the S-parameter analysis functions in detail. The functions are listed in alphabetical order.

- *abcdtoh()* (expmeas)
- *abcdtos()* (expmeas)
- *abcdtoy()* (expmeas)
- *abcdtoz()* (expmeas)
- *bandwidth func()* (expmeas)
- *center freq()* (expmeas)
- *dev lin gain()* (expmeas)
- *dev lin phase()* (expmeas)
- *ga circle()* (expmeas)
- *gain comp()* (expmeas)
- *gl circle()* (expmeas)
- *gp circle()* (expmeas)
- *gs circle()* (expmeas)
- *htoabcd()* (expmeas)
- *htos()* (expmeas)
- *htoy()* (expmeas)
- *htoz()* (expmeas)
- *ispec()* (expmeas)
- *l stab circle()* (expmeas)
- *l stab circle center radius()* (expmeas)
- *l stab region()* (expmeas)
- *map1 circle()* (expmeas)
- *map2 circle()* (expmeas)
- *max gain()* (expmeas)
- *mu()* (expmeas)
- *mu prime()* (expmeas)
- *ns circle()* (expmeas)
- *ns pwr int()* (expmeas)
- *ns pwr ref bw()* (expmeas)
- *phase comp()* (expmeas)
- *pwr gain()* (expmeas)
- *ripple() Measurement* (expmeas)
- *sm gamma1()* (expmeas)
- *sm gamma2()* (expmeas)
- *sm y1()* (expmeas)
- *sm y2()* (expmeas)
- *sm z1()* (expmeas)
- *sm z2()* (expmeas)
- *s stab circle()* (expmeas)
- *s stab circle center radius()* (expmeas)
- *s stab region()* (expmeas)
- *stab fact()* (expmeas)
- *stab meas()* (expmeas)
- *stoabcd()* (expmeas)
- *stoh()* (expmeas)

- *stos()* (expmeas)
- *stot()* (expmeas)
- *stoy()* (expmeas)
- *stoz()* (expmeas)
- *tdr_step_impedance()* (expmeas)
- *tdr sp gamma()* (expmeas)
- *tdr sp imped()* (expmeas)
- *tdr step imped()* (expmeas)
- *ttos()* (expmeas)
- *unilateral figure()* (expmeas)
- *unwrap()* (expmeas)
- *v dc()* (expmeas)
- *volt gain()* (expmeas)
- *volt gain max()* (expmeas)
- *vswr()* (expmeas)
- *write snp()* (expmeas)
- *yin()* (expmeas)
- *yopt()* (expmeas)
- *ytoabcd()* (expmeas)
- *ytoh()* (expmeas)
- *ytos()* (expmeas)
- *ytoz()* (expmeas)
- *zin()* (expmeas)
- *zopt()* (expmeas)
- *ztoabcd()* (expmeas)
- *ztoh()* (expmeas)
- *ztos()* (expmeas)
- *ztoy()* (expmeas)

abcdtoh()

This measurement transforms the chain (ABCD) matrix of a 2-port network to a hybrid matrix.

Syntax

h=abcdtoh(a)

Arguments

Name	Description	Default	Range	Type	Required
A	chain (ABCD) matrix of a 2-port network	None	(-∞:∞)	Complex	Yes

Examples

```
h=abcdtoh(a)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

abcdtos() (expmeas), *abcdtoy()* (expmeas), *abcdtoz()* (expmeas), *htoabcd()* (expmeas), *stoabcd()* (expmeas), *ytoabcd()* (expmeas), *ztoabcd()* (expmeas)

abcdtos()

This measurement transforms the chain (ABCD) matrix of a 2-port network to a scattering matrix.

Syntax

sp = abcdtos(A, zRef)

Arguments

Name	Description	Default	Range	Type	Required
A	chain (ABCD) matrix of a 2-port network	None	(-∞:∞)	Complex	Yes
zRef	reference impedance	50.0	(-∞:∞)	Integer, real or complex	No

Examples

```
sp = abcdtos(a, 50)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

abcdtoh() (expmeas), *abcdtoy()* (expmeas), *abcdtoz()* (expmeas), *htoabcd()* (expmeas), *stoabcd()* (expmeas), *ytoabcd()* (expmeas), *ztoabcd()* (expmeas)

abcdtoy()

This measurement transforms the chain (ABCD) matrix of a 2-port network to an admittance matrix.

Syntax

y = abcdtoy(a)

Arguments

Name	Description	Default	Range	Type	Required
A	chain (ABCD) matrix of a 2-port network	None	(-∞:∞)	Complex	Yes

Examples

```
y = abcdtoy(a)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

abcdtoh() (expmeas), *abcdtos()* (expmeas), *abcdtoz()* (expmeas), *htoabcd()* (expmeas), *stoabcd()* (expmeas), *ytoabcd()* (expmeas), *ztoabcd()* (expmeas)

abcdtoz()

This measurement transforms the chain (ABCD) matrix of a 2-port network to impedance matrix.

Syntax

z = abcdtoz(a)

Arguments

Name	Description	Default	Range	Type	Required
A	chain (ABCD) matrix of a 2-port network	None	(-∞:∞)	Complex	Yes

Examples

```
z = abcdtoz(a)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

abcdtoh() (expmeas), *abcdtos()* (expmeas), *abcdtoy()* (expmeas), *htoabcd()* (expmeas), *stoabcd()* (expmeas), *ytoabcd()* (expmeas), *ztoabcd()* (expmeas)

bandwidth_func()

Returns the bandwidth at the specified level as a real number. Typically used in filter application to calculate the 1, 3 dB bandwidth.

Syntax

bw = bandwidth_func(Data, DesiredValue, Type)

Arguments

Name	Description	Default	Range	Type	Required
Data	data (usually gain) to find the bandwidth	None	(-∞:∞)	Integer, Real	Yes
DesiredValue	A single value representing the desired bandwidth level.	None	(-∞:∞)	Real	Yes
Type	Type of response	0	[0:3] †	Integer	No

† Type:
0 - Band-pass
1 - Band-stop
2 - Low-pass
3 - High-pass

Examples

```
We assume that a S-Parameter analysis has been performed.
bw3dB = bandwidth_func(db(S21), 3) returns the 3dB bandwidth
```

Defined in

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

See Also

center_freq() (expmeas)

Notes/Equations

This function returns the bandwidth of a filter response. Bandwidth is defined as the difference between the upper and lower frequency at which the amplitude response is *DesiredValue* dB below the maximum amplitude. It uses an iterative process to find the bandwidth on non-ideal responses for the band limited responses.
Data can be from 1 to 4 dimensions.

center_freq()

Returns the center frequency at the specified level as a real number.
Typically used in filter application to calculate the center frequency at 1, 3 dB bandwidth.

Syntax

fc = center_freq(Data, ReferenceBW)

Arguments

Name	Description	Default	Range	Type	Required
Data	data (usually gain) to find the center frequency	None	(-∞:∞)	Complex	Yes
ReferenceBW	a single value representing the reference bandwidth	None	(-∞:∞)	Real	Yes

Examples

```
We assume that a S-Parameter analysis has been performed.
fc3 = center_freq(db(S21), 3) returns the center frequency using the 3dB point
as reference
```

Defined in

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

See Also

bandwidth_func() (expmeas)

Notes/Equations

This function returns the center frequency of a filter response. Linear interpolation is performed between data points. The function uses an iterative process to find the bandwidth on non-ideal responses. The data can be from 1 to 4 dimensions.

dev_lin_gain()

Given a variable sweep over a frequency range, a linear least-squares fit is performed on the gain of the variable, and the deviation from this linear fit is calculated at each frequency point.

Syntax

y = dev_lin_gain(voltGain)

Arguments

Name	Description	Default	Range	Type	Required
voltGain	gain as a function of frequency	None	[0:∞)	Complex	Yes

Examples

```
a = dev_lin_gain(volt_gain(S,PortZ(1),PortZ(2)))
```

Defined in

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

See Also

dev_lin_phase() (expmeas), *diff()* (expmeas), *phasedeg()* (ael), *phaserad()* (ael), *pwr_gain()* (expmeas), *ripple()* (expmeas), *unwrap()* (expmeas), *volt_gain()* (expmeas)

dev_lin_phase()

Given a variable sweep over a frequency range, a linear least-squares fit is performed on the phase of the variable, and the deviation from this linear fit is calculated at each frequency point.

Syntax

y = dev_lin_phase(voltGain)

Arguments

Name	Description	Default	Range	Type	Required
voltGain	function of frequency	None	[0:∞)	Complex	Yes

Examples

```
a = dev_lin_phase(S21)
```

Defined in

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

See Also

dev_lin_gain() (expmeas), *diff()* (expmeas), *phasedeg()* (expmeas), *phaserad()* (expmeas), *pwr_gain()* (expmeas), *ripple()* (expmeas), *unwrap()* (expmeas), *volt_gain()* (expmeas)

Notes/Equations

In order to use this function, the *Group Delay* option must be enabled in the S-parameter analysis setup. For more information, refer to " *Calculating Group Delay* " in your " *S-Parameter Simulation* " documentation.

ga_circle()

Generates an available gain circle.

Syntax

y = ga_circle(S, gain, numOfPts, numCircles, gainStep)

Arguments

Name	Description	Default	Range	Type	Required
S	scattering matrix of a 2-port network.	None	(-∞:∞)	Complex	Yes
gain	specified gain in dB	†	[0:∞)	Integer or Real array	No
numOfPts	desired number of points per circle	51	[1:∞)	Integer	No
numCircles	number of desired circles. This is used if gain is not specified.	None	[0:∞)	Integer	No
gainStep	gain step size. This is used if gain is not specified.	1.0	[0:∞)	Integer or Real	No

† Default value for gain is min(max_gain(S)) - {1, 2, 3}

Examples

```
circleData = ga_circle(S, 2, 51)
circleData = ga_circle(S, {2, 3, 4}, 51) return the points on the circle(s).
circleData = ga_circle(S, , 51, 5, 0.5) return the points on the circle(s) for
5 circles at maxGain - {0,0.5,1.0,1.5,2.0}
circleData = ga_circle(S, , , 2, 1.0) return the points on the circle(s) for 2
circles at maxGain - {0,1.0}
```

Defined in

$HPEESOF_DIR/expressions/ael/circle_fun.ael

See Also

gl_circle() (expmeas), *gp_circle()* (expmeas), *gs_circle()* (expmeas)

Notes/Equations

This function is used in Small-signal S-parameter simulations.
The function generates the constant available-gain circle resulting from a source mismatch. The circle is defined by the loci of the source-reflection coefficients resulting in the specified gain.

A gain circle is created for each value of the swept variable(s). Multiple gain values can be specified for a scattering parameter that has dimension less than four. This measurement is supported for 2-port networks only.
If gain and numCircles are not specified, gain circles are drawn at min(max_gain(S)) - {0,1,2,3}. That is, gain is calculated at a loss of 0,1,2,3 dB from maxGain.

If gain is not specified and numCircles is given, then numCircles gain circles are drawn at gainStep below max_gain(). Gain is also limited by max_gain(S). That is, if gain > max_gain(S), then the circle is generated at max_gain(S).

$$Ca(Center) = \frac{gaC_1^*}{(1 + ga(|S_{11}|^2 - |\Delta|^2))}$$

$$Ra(radius) = \frac{\sqrt{1 - 2K|S_{12}S_{21}|ga + |S_{12}S_{21}|^2 ga^2}}{|1 + ga(|S_{11}|^2 - |\Delta|^2)|}$$

Where:
K = Stability Factor

$ga(G) = (Ga/|S_{21}|^2)$ (Ga is Desired Gain in absolute terms)

$C_1 = S_{11} - |\Delta|^2 S_{22}^*$

gain_comp()

Returns gain compression

Syntax

y = gain_comp(Sji)

Arguments

Name	Description	Default	Range	Type	Required
Sji	Sji is a power-dependent complex transmission coefficient obtained from large-signal S-parameter simulation.	None	(-∞:∞)	Complex	Yes

Examples

```
gc = gain_comp(S21[::, 0])
```

Defined in

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

See Also

phase_comp() (expmeas)

Notes/Equations

Used in Large-signal S-parameter simulations.
This measurement calculates the small-signal minus the large-signal power gain, in dB. The first power point (assumed to be small) is used to calculate the small-signal power gain.

gl_circle()

Returns a load-mismatch gain circle.

Syntax

y = gl_circle(S, gain, numOfPts, numCircles, gainStep)

Arguments

Name	Description	Default	Range	Type	Required
S	scattering matrix of a 2-port network.	None	(-∞:∞)	Complex	Yes
gain	specified gain in dB	maxGain - {0, 1, 2, 3} †	[0:∞)	Integer or Real array	No
numOfPts	desired number of points per circle	51	[1:∞)	Integer	No
numCircles	number of desired circles. This is used if gain is not specified.	None	[0:∞)	Integer	No
gainStep	gain step size. This is used if gain is not specified.	1.0	[0:∞)	Integer or Real	No

† Where maxGain = 10*log(1 / (1 - mag(S22)**2))

Examples

```
circleData = gl_circle(S, 2, 51)
circleData = gl_circle(S, {2, 3, 4}, 51) return the points on the circle(s).
circleData = gl_circle(S, , 51, 5, 0.5) return the points on the circle(s) for
5 circles at maxGain - {0,0.5,1.0,1.5,2.0}
circleData = gl_circle(S, , , 2, 1.0) return the points on the circle(s) for 2
circles at maxGain - {0,1.0}
```

Defined in

$HPEESOF_DIR/expressions/ael/circle_fun.ael

See Also

***ga_circle()* (expmeas), *gp_circle()* (expmeas), *gs_circle()* (expmeas)**

Notes/Equations

Used in Small-signal S-parameter simulations.
This function generates the unilateral gain circle resulting from a load mismatch. The circle is defined by the loci of the load-reflection coefficients that result in the specified gain.

A gain circle is created for each value of the swept variable(s). Multiple gain values can be specified for a scattering parameter that has dimension less than four. This measurement is supported for 2-port networks only.

If gain and numCircles are not specified, gain circles are drawn at maxGain - {0,1,2,3}. That is, gain is calculated at a loss of 0,1,2,3 dB from the maximum gain. If gain is not specified and numCircles is given, then numCircles gain circles are drawn at gainStep below maxGain. Gain is also limited by maxGain. That is, if gain > maxGain, then the circle is generated at maxGain.

$$Cl(Center) = \frac{(1 - |S_{22}|^2 Gabs)|S_{22}|^*}{1 - |S_{22}|^2 (1 - |S_{22}|^2 Gabs)}$$

$$Rl(radius) = \frac{\sqrt{|1 - (1 - |S_{22}|^2 Gabs)|}(1 - |S_{22}|^2)}{1 - |S_{22}|^2 (1 - |S_{22}|^2 Gabs)}$$

Where *Gabs* is the absolute gain, and is given by:

$$Gabs = (10^{dBGain})/10.0$$

gp_circle()

Generates a power gain circle.

Syntax

y = gp_circle(S, gain, numOfPts, numCircles, gainStep)

Arguments

Name	Description	Default	Range	Type	Required
S	scattering matrix of a 2-port network.	None	(-∞:∞)	Complex	Yes
gain	specified gain in dB	†	[0:∞)	Integer or Real array	No
numOfPts	desired number of points per circle	51	[1:∞)	Integer	No
numCircles	number of desired circles. This is used if gain is not specified.	None	[0:∞)	Integer	No
gainStep	gain step size. This is used if gain is not specified.	1.0	[0:∞)	Integer or Real	No

† Default value for gain is min(max_gain(S)) - {1, 2, 3}

Examples

```
circleData = gp_circle(S, 2, 51)
circleData = gp_circle(S, {2, 3, 4}, 51) return the points on the circle(s).
circleData = gp_circle(S, , 51, 5, 0.5) return the points on the circle(s) for
5 circles at maxGain - {0,0.5,1.0,1.5,2.0}
circleData = gp_circle(S, , , 2, 1.0) return the points on the circle(s) for 2
circles at maxGain - {0,1.0}
```

Defined in

$HPEESOF_DIR/expressions/ael/circle_fun.ael

See Also

ga_circle() (expmeas), *gl_circle()* (expmeas), *gs_circle()* (expmeas)

Notes/Equations

Used in Small-signal S-parameter simulations.
This function generates a constant-power-gain circle resulting from a load mismatch. The circle is defined by the loci of the output-reflection coefficients that result in the specified gain.

A gain circle is created for each value of the swept variable(s). Multiple gain values can be specified for a scattering parameter that has dimension less than four. This measurement is supported for 2-port networks only.

If gain and numCircles are not specified, gain circles are drawn at min(max_gain(S)) - {0,1,2,3}. That is, gains are calculated at a loss of 0,1,2,3 dB from the maximum gain. If gain is not specified and numCircles is given, then numCircles gain circles are drawn at gainStep below max_gain(). Gain is also limited by max_gain(S). That is, if gain > max_gain(S), then the circle is generated at max_gain(S)).

$$Cp(Center) = \frac{gpC_2^*}{(1 + gp(|S_{22}|^2 - |\Delta|^2))}$$

$$Rp(radius) = \frac{\sqrt{1 - 2K|S_{12}S_{21}|gp + |S_{12}S_{21}|^2 gp^2}}{|1 + gp(|S_{22}|^2 - |\Delta|^2)|}$$

Where:
K = Stability Factor

$gp(G) = (Gp/|S_{21}|^2)$ (Gp is Desired Gain in absolute terms)

$C_2 = S_{22} - |\Delta|^2 S_{11}^*$

gs_circle()

Returns a source-mismatch gain circle.

Syntax

y = gs_circle(S, gain, numOfPts, numCircles, gainStep)

Arguments

Name	Description	Default	Range	Type	Required
S	scattering matrix of a 2-port network.	None	(-∞:∞)	Complex	Yes
gain	specified gain in dB	maxGain - {0, 1, 2, 3} †	[0:∞)	Integer or Real array	No
numOfPts	desired number of points per circle	51	[1:∞)	Integer	No
numCircles	number of desired circles. This is used if gain is not specified.	None	[0:∞)	Integer	No
gainStep	gain step size. This is used if gain is not specified.	1.0	[0:∞)	Integer or Real	No

† Where maxGain = 10*log(1 / (1 - mag(S11)**2))

Examples

```
circleData = gs_circle(S, 2, 51)
circleData = gs_circle(S, {2, 3, 4}, 51) return the points on the circle(s).
circleData = gs_circle(S, , 51, 5, 0.5) return the points on the circle(s) for
5 circles at maxGain - {0,0.5,1.0,1.5,2.0}
circleData = gs_circle(S, , , 2, 1.0) return the points on the circle(s) for 2
circles at maxGain - {0,1.0}
```

Defined in

$HPEESOF_DIR/expressions/ael/circle_fun.ael

See Also

ga_circle() (expmeas), *gl_circle()* (expmeas), *gp_circle()* (expmeas)

Notes/Equations

Used in Small-signal S-parameter simulations.
This function generates the unilateral gain circle resulting from a source mismatch. The circle is defined by the loci of the source-reflection coefficients that result in the specified gain. A gain circle is created for each value of the swept variable(s). Multiple gain values can be specified for a scattering parameter that has dimension less than four. This measurement is supported for 2-port networks only.

If gain and numCircles are not specified, gain circles are drawn at maxGain - {0,1,2,3}. That is, gain values are calculated at a loss of 0,1,2,3 dB from the maximum gain. If gain is not specified and if numCircles is given, then numCircles gain circles are drawn at gainStep below maxGain. Gain is also limited by maxGain. That is, if gain > maxGain, then the circle is generated at maxGain.

$$Cs(Center) = \frac{(1-|S_{11}|^2 Gabs)|S_{11}|^*}{1-|S_{11}|^2(1-|S_{11}|^2 Gabs)}$$

$$Rs(radius) = \frac{\sqrt{|1-(1-|S_{11}|^2 Gabs)|}(1-|S_{11}|^2)}{1-|S_{11}|^2(1-|S_{11}|^2 Gabs)}$$

Where *Gabs* is the absolute gain, and is given by:

$$Gabs = (10^{dBGain})/10.0$$

htoabcd()

This measurement transforms the hybrid matrix of a 2-port network to a chain (ABCD) matrix.

Syntax

a = htoabcd(H)

Arguments

Name	Description	Default	Range	Type	Required
H	hybrid matrix of a 2-port network	None	(-∞:∞)	Complex	Yes

Examples

```
a = htoabcd(H)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

abcdtoh() (expmeas), *htoz()* (expmeas), *ytoh()* (expmeas)

htos()

This measurement transforms the hybrid matrix of a 2-port network to a scattering matrix.

Syntax

sp = htos(h, zRef)

Arguments

Name	Description	Default	Range	Type	Required
H	hybrid matrix of a 2-port network	None	(-∞:∞)	Complex	Yes
zRef	reference impedance	50.0	(-∞:∞)	Integer, real or complex	No

Examples

```
sp = htos(h, 50)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

htoy() (expmeas), *htoz()* (expmeas), *stoh()* (expmeas)

htoy()

This measurement transforms the hybrid matrix of a 2-port network to an admittance matrix.

Syntax

y = htoy(H)

Arguments

Name	Description	Default	Range	Type	Required
H	hybrid matrix of a 2-port network	None	(-∞:∞)	Complex	Yes

Examples

```
y = htoy(H)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

htos() (expmeas), *htoz()* (expmeas), *ytoh()* (expmeas)

htoz()

This measurement transforms the hybrid matrix of a 2-port network to an impedance matrix.

Syntax

z = htoz(H)

Arguments

Name	Description	Default	Range	Type	Required
H	hybrid matrix of a 2-port network	None	(-∞:∞)	Complex	Yes

Examples

```
z = htoz(h)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

htos() (expmeas), *htoy()* (expmeas), *ytoh()* (expmeas)

ispec()

Returns the current frequency spectrum

Syntax

y = ispec(current)

Arguments

Name	Description	Default	Range	Type	Required
current	current	None	(-∞:∞)	Real, Complex	Yes

Examples

```
a = ispec(i1)
```

Defined In

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

pspec() (expmeas), *vspec()* (expmeas)

Notes/Equations

This measurement gives a current frequency spectrum. The measurement gives a set of RMS currents at each frequency.

l_stab_circle()

Returns load (output) stability circles.

Syntax

y = l_stab_circle(S, numOfPts)

Arguments

Name	Description	Default	Range	Type	Required
S	scattering matrix of a 2-port network.	None	(-∞:∞)	Complex	Yes
numOfPts	desired number of points per circle	51	[1:∞)	Integer	No

Examples

```
circleData = l_stab_circle(S, 51) returns the points on the circle(s).
```

Defined in

$HPEESOF_DIR/expressions/ael/circle_fun.ael

See Also

l_stab_circle_center_radius() (expmeas), *l_stab_region()* (expmeas), *s_stab_circle()* (expmeas), *s_stab_region()* (expmeas)

Notes/Equations

Used in Small-signal S-parameter simulations.
The function generates a load stability circle. The circle is defined by the loci of load-reflection coefficients where the magnitude of the source-reflection coefficient is 1.

A circle is created for each value of the swept variable(s). This measurement is supported for 2-port networks only.
Use the function _l_stab_circle_center_radius()_ to find the center and radius of the stability circle.

Use the function _l_stab_region(S)_ to determine the region of stability.

l_stab_circle_center_radius()

Returns the center and radius of the load (output) stability circle

Syntax

l_cr = l_stab_circle_center_radius(S, Type)

Arguments

Name	Description	Default	Range	Type	Required
S	2-port S-Parameters	None	None	Complex	Yes
Type	Type of parameter to return	"both"	["both","center","radius"]	String	No

Examples

```
1. l_cr=l_stab_circle_center_radius(S) returns a 1X2 matrix containing center
and radius
lCirc=expand(circle(l_cr(1), real(l_cr(2)), 51)) returns data for the load
stability circle
2. l_radius=l_stab_circle_center_radius(S, "radius") returns the radius
```

Defined in

$HPEESOF_DIR/expressions/ael/circle_fun.ael

See Also

l_stab_circle() (expmeas), *s_stab_circle()* (expmeas), *s_stab_circle_center_radius()* (expmeas)

Notes/Equations

Used in Small-signal S-parameter simulations.
If the argument *Type* is not specified, the function returns complex data. Although radius is of type real, the values are returned as complex. Therefore, when using the returned radius, use the real part. To obtain the radius as a real number, set the *Type* argument to *radius* .

l_stab_region()

This expression returns a string identifying the region of stability of the corresponding load stability circle.

Syntax

y = l_stab_region(S)

Arguments

Name	Description	Default	Range	Type	Required
S	scattering matrix of a 2-port network.	None	(-∞:∞)	Complex	Yes

Examples

```
region = l_stab_region(S) returns "Outside" or "Inside".
```

Defined in

$HPEESOF_DIR/expressions/ael/circle_fun.ael

See Also

l_stab_circle() (expmeas), *s_stab_circle()* (expmeas), *s_stab_region()* (expmeas)

Notes/Equations

Used in Small-signal S-parameter simulations.

map1_circle()

Returns source-mapping circles from port 2 to port 1.

Syntax

circleData=map1_circle(S, numOfPts)

Arguments

Name	Description	Default	Range	Type	Required
S	scattering matrix of a 2-port network.	None	(-∞:∞)	Complex	Yes
numOfPts	desired number of points per circle	51	[1:∞)	Integer	No

Examples

```
circleData=map1_circle(S, 51) returns the points on the circle(s).
```

Defined in

$HPEESOF_DIR/expressions/ael/circles_fun.ael

See Also

map2_circle() (expmeas)

Notes/Equations

Used in Small-signal S-parameter simulations
The function maps the set of terminations with unity magnitude at port 2 to port 1. The circles are defined by the loci of terminations on one port as seen at the other port. A source-mapping circle is created for each value of the swept variable(s). This measurement is supported for 2-port networks only.

map2_circle()

Returns load-mapping circles from port 1 to port 2.

Syntax

circleData=map2_circle(S, numOfPts)

Arguments

Name	Description	Default	Range	Type	Required
S	scattering matrix of a 2-port network.	None	(-∞:∞)	Complex	Yes
numOfPts	desired number of points per circle	51	[1:∞)	Integer	No

Examples

```
circleData=map2_circle(S, 51) returns the points on the circle(s).
```

Defined in

$HPEESOF_DIR/expressions/ael/circle_fun.ael

See Also

map1_circle() (expmeas)

Notes/Equations

Used in Small-signal S-parameter simulations
The function maps the set of terminations with unity magnitude at port 1 to port 2. The circles are defined by the loci of terminations on one port as seen at the other port. A source-mapping circle is created for each value of the swept variable(s). This measurement is supported for 2-port networks only.

max_gain()

Given a 2 x 2 scattering matrix, this measurement returns the maximum available and stable gain (in dB) between the input and the measurement ports.

Syntax

y = max_gain(S)

Arguments

Name	Description	Default	Range	Type	Required
S	scattering matrix of 2-port network	None	(-∞:∞)	Complex	Yes

Examples

```
y = max_gain(S)
```

Defined in

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

See Also

sm_gamma1() (expmeas), *sm_gamma2()* (expmeas), *stab_fact()* (expmeas), *stab_meas()* (expmeas)

Notes/Equations

This function should be used in a stable system, i.e. when the stability factor (k) is greater than 1. The function actually works for both k > 1 and k <= 1. If k > 1, then k is used in

the max gain calculations. However, if k <= 1, then k = 1 is used in the max gain calculations.

mu()

Returns the geometrically derived stability factor for the load.

Syntax

y = mu(S)

Arguments

Name	Description	Default	Range	Type	Required
S	scattering matrix of a 2-port network	None	(-∞:∞)	Complex	Yes

Examples

```
x=mu(S)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

mu_prime() (expmeas)

Notes/Equations

1. This measurement gives the distance from the center of the Smith chart to the nearest output (load) stability circle. This stability factor is given by:
   $mu = \{1-|S_{11}|^2\} / \{|S_{22} - conj(S_{11})*Delta | + |S_{12}*S_{21}| \}$
   where Delta is the determinant of the S-parameter matrix. Having mu > 1 is the single necessary and sufficient condition for unconditional stability of the 2-port network.
2. The Mu component is available in the *Simulation-S_Param* palette.

Reference

1. M. L. Edwards and J. H. Sinsky, "A new criterion for linear 2-port stability using geometrically derived parameters", IEEE Transactions on Microwave Theory and Techniques, Vol. 40, No. 12, pp. 2303-2311, Dec. 1992.

mu_prime()

Returns the geometrically derived stability factor for the source.

Syntax

y = mu_prime(S)

Arguments

Name	Description	Default	Range	Type	Required
S	scattering matrix of a 2-port network	None	(-∞:∞)	Complex	Yes

Examples

```
x=mu_prime(S)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

mu() (expmeas)

Notes/Equations

This measurement gives the distance from the center of the Smith chart to the nearest unstable-input (source) stability circle. This stability factor is given by:

mu_prime = $\{1-|S_{22}|^2\}$ / $\{|S_{11}$ - conj(S_{22})*Delta | + $|S_{21}*S_{12}|$ }

where Delta is the determinant of the S-parameter matrix. Having mu_prime > 1 is the single necessary and sufficient condition for unconditional stability of the 2-port network.

Reference

M. L. Edwards and J. H. Sinsky, "A new criterion for linear 2-port stability using geometrically derived parameters", IEEE Transactions on Microwave Theory and Techniques, Vol. 40, No. 12, pp. 2303-2311, Dec. 1992.

ns_circle()

Returns noise-figure circles.

Syntax

y = ns_circle(nf2, NFmin, Sopt, rn, numOfPts, numCircles, NFStep)

Arguments

Name	Description	Default	Range	Type	Required
nf2	specified noise figure	†	(-∞:∞)	Real	No
NFmin	minimum noise figure	None	[0:∞)	Integer or Real	Yes
Sopt	optimum mismatch	None	[0:∞)	Complex	Yes
rn	equivalent normalized noise resistance of a 2-port network ††	None	[0:∞)	Complex	Yes
numOfPts	desired number of points per circle	51	[1:∞)	Integer	No
numCircles	number of desired circles. This is used if nf2 is not specified.	None	[0:∞)	Integer	No
NFStep	nf step size. This is used if nf2 is not specified.	1.0	[0:∞)	Integer or Real	No

† If nf2 is NULL or not specified the default is max(NFmin)+{0,1,2,3}.
†† rn = Rn/zRef where Rn is the equivalent noise resistance and zRef is the reference impedance.

Examples

```
circleData = ns_circle(0+NFmin, NFmin, Sopt, Rn/50, 51)
circleData = ns_circle(NULL, NFmin, Sopt,Rn/50,51) return the points on the
circle for 4 circles at max(NFmin)+{0,1,2,3}
Returns the points on the circle(s):
circleData = ns_circle({0, 1}+NFmin, NFmin, Sopt, Rn/50, 51)
Returns the points on the circle(s) for 3 circles at max(NFmin) + {0, 0.5,
1.0}:
circleData = ns_circle(, NFmin, Sopt, Rn/50, 51, 3, 0.5)
Returns the points on the circle(s) for 3 circles at max(NFmin) + {0, 1, 2.0}:
circleData = ns_circle(, NFmin, Sopt, Rn/50, , 3)
```

Defined in

$HPEESOF_DIR/expressions/ael/circle_fun.ael

Notes/Equations

Used in Small-signal S-parameter simulations and Harmonic Balance analysis.
The expression generates constant noise-figure circles. The circles are defined by the loci of the source-reflection coefficients that result in the specified noise figure. NFmin, Sopt, and Rn are generated from noise analysis.
A circle is created for each value of the swept variable(s).

If both nf2 and numCircles are specified, then circles are drawn at nf2 values (numCircles is not used).

If nf2 and numCircles are not specified, then nf2 circles are drawn at max(NFmin) + {0,1,2,3}.

If nf2 is not specified, and numCircles is given, then numCircles nf2 circles are drawn at

NFStep above max(NFmin).

If nf2 is specified, and numCircles is not specified, then circles are drawn at nf2 values.

ns_pwr_int()

Returns the integrated noise power

Syntax

y = ns_pwr_int(Sji, nf, resBW, stop)

Arguments

Name	Description	Default	Range	Type	Required
Sji	complex transmission coefficient	None	(-∞:∞)	Complex	Yes
nf	noise figure at the output port (in dB)	None	(-∞:∞)	Complex	Yes
resBW	user-defined resolution bandwidth	None	(-∞:∞)	Complex	Yes
stop	stop value (works for nonuniform delta frequency)	None	[0:∞)	Real	No

Examples

```
Y = ns_pwr_int(S21, nf2, 1MHz)
```

Defined in

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

See Also

ns_pwr_ref_bw() (expmeas), *snr()* (expmeas)

Notes/Equations

Used in Small-signal S-parameter simulation
This is the integrated noise power (in dBm) calculated by integrating the noise power over the entire frequency sweep. The noise power at each frequency point is calculated by multiplying the noise spectral density by a user-defined resolution bandwidth.

ns_pwr_ref_bw()

Returns noise power in a reference bandwidth

Syntax

y = ns_pwr_ref_bw(Sji, nf, resBW)

Arguments

Name	Description	Default	Range	Type	Required
Sji	complex transmission coefficient	None	(-∞:∞)	Complex	Yes
nf	noise figure at the output port (in dB)	None	(-∞:∞)	Complex	Yes
resBW	user-defined resolution bandwidth	None	(-∞:∞)	Complex	Yes

Examples

```
Y = ns_pwr_ref_bw(S21, nf2, 1MHz) returns the noise power with respect to the
reference bandwidth.
```

Defined in

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

See Also

ns_pwr_int() (expmeas), *snr()* (expmeas)

Notes/Equations

Used in Small-signal S-parameter simulation.
This is the noise power calculated by multiplying the noise spectral density at a frequency point by a user-defined resolution bandwidth. Unlike NsPwrInt, this gives the noise power (in dBm) at each frequency sweep.

The ns_pwr_ref_bw() function ignores noise from the load. It will always take into account the noise from the source resistor to be fixed at 290K. You can raise the temperature of the source resistor (Term), or set Noise to `no` in the source resistor (Term), or set Noise to `yes` in the load resistor. These actions will not change the ns_pwr_ref_bw() results.

phase_comp()

Returns the phase compression (phase change)

Syntax

y = phase_comp(Sji)

Arguments

Name	Description	Default	Range	Type	Required
Sji	power-dependent parameter obtained from large-signal S-parameters simulation	None	(-∞:∞)	Integer, real, complex	Yes

Examples

```
a = phase_comp(S21[::, 0])
```

Defined in

$HPEESOF_DIR/expressions/ael/rf_systems_fun.ael

See Also

gain_comp() (expmeas)

Notes/Equations

Used in Large-signal S-parameter simulations
This measurement calculates the small-signal minus the large-signal phase, in degrees.
The first power point (assumed to be small) is used to calculate the small-signal phase.
Phase compression (change) is only available for 1-D power sweep.

pwr_gain()

This measurement is used to determine the transducer power gain (in dB) and is defined as the ratio of the power delivered to the load, to the power available from the source. (where power is in dBm).

Syntax

y = pwr_gain(S, Zs, Zl, Zref)

Arguments

Name	Description	Default	Range	Type	Required
S	2 X 2 scattering matrix	None	(-∞:∞)	Complex	Yes
Zs	input impedance	None	(-∞:∞)	Real, Complex	Yes
Zl	Output impedance	None	(-∞:∞)	Real, Complex	Yes
Zref	reference impedance	50.0	(-∞:∞)	Real, Complex	Yes

Examples

```
pGain = pwr_gain(S,50,75) - Zref defaults to 50 ohms
pGain1 = pwr_gain(S, 50, 75, 75) - Zref = 75 ohms
```

Defined in

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

See Also

stos() (expmeas), *volt_gain()* (expmeas), *volt_gain_max()* (expmeas)

ripple()

This function measures the deviation of x from the average of x

Syntax

y = ripple(x, fstart, fstop)

Arguments

Name	Description	Default	Range	Type	Required
x	can be a gain or group delay data over a given frequency range	None	(-∞:∞)	Real	Yes
fstart	start frequency	None	(0:∞)	Real	No
fstop	stop frequency	None	(0:∞)	Real	No

Examples

```
a = ripple(pwr_gain(S21))
a1 = ripple(pwr_gain(S21), 1GHz, 3GHz)
```

Defined in

$HPEESOF_DIR/expressions/ael/elementary_fun.ael

See Also

dev_lin_gain() (expmeas), *dev_lin_phase()* (expmeas), *diff()* (expmeas), *mean()* (expmeas), *phasedeg()* (ael), *phaserad()* (ael), *unwrap()* (expmeas)

Notes/Equations

In order to use this function, the *Group Delay* option must be enabled in the S-parameter analysis setup. For more information, refer to " *Calculating Group Delay* " in your " *S-Parameter Simulation* " documentation.
This function supports data up to four dimensions.

Note
The function name ripple() is used for more than one type of expression. For comparison, see the Simulator Expression *ripple() Expression* (expsim). Also, for more information on the different expression types and the contexts in which they are used, see *Duplicated Expression Names* (expmeas).

sm_gamma1()

Returns the simultaneous-match input-reflection coefficient.

Syntax

y = sm_gamma1(S)

Arguments

Name	Description	Default	Range	Type	Required
S	scattering matrix of 2-port network	None	(-∞:∞)	Complex	Yes

Examples

```
a = sm_gamma1(S)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

max_gain() (expmeas), *sm_gamma2()* (expmeas), *stab_fact()* (expmeas), *stab_meas()* (expmeas)

Notes/Equations

This complex measurement determines the reflection coefficient that must be presented to the input (port 1) of the network to achieve simultaneous input and output reflections. If the Rollett stability factor stab_fact(S) is less than unity for the analyzed circuit, then sm_gamma1(S) returns 1+j*0.

sm_gamma2()

Returns the simultaneous-match output-reflection coefficient.

Syntax

y = sm_gamma2(S)

Arguments

Name	Description	Default	Range	Type	Required
S	scattering matrix of 2-port network	None	(-∞:∞)	Complex	Yes

Examples

```
a = sm_gamma2(S)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

max_gain() (expmeas), *sm_gamma1()* (expmeas), *stab_fact()* (expmeas), *stab_meas()* (expmeas)

Notes/Equations

This complex measurement determines the reflection coefficient that must be presented to the output (port 2) of the network to achieve simultaneous input and output reflections. If the Rollett stability factor stab_fact(S) is less than unity for the analyzed circuit, then sm_gamma2(S) returns1+j*0.

sm_y1()

This complex measurement determines the admittance that must be presented to the input (port 1) of the network to achieve simultaneous input and output reflections.

Syntax

y = sm_y1(S, Z)

Arguments

Name	Description	Default	Range	Type	Required
S	scattering matrix of a 2-port network	None	(-∞:∞)	Complex	Yes
Z	port impedance	50.0	(-∞:∞)	Integer, real or Complex	Yes

Examples

```
a = sm_y1(S, 50)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

sm_y2() (expmeas)

sm_y2()

This complex measurement determines the admittance that must be presented to the input (port 1) of the network to achieve simultaneous input and output reflections.

Syntax

y = sm_y2(S, Z)

Arguments

Name	Description	Default	Range	Type	Required
S	scattering matrix of a 2-port network	None	(-∞:∞)	Complex	Yes
Z	port impedance	50.0	(-∞:∞)	Integer, real or Complex	Yes

Examples

```
a = sm_y2(S, 50)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

sm_y1() (expmeas)

sm_z1()

This complex measurement determines the impedance that must be presented to the input (port 1) of the network to achieve simultaneous input and output reflections.

Syntax

y = sm_z1(S, Z)

Arguments

Name	Description	Default	Range	Type	Required
S	scattering matrix of a 2-port network	None	(-∞:∞)	Complex	Yes
Z	port impedance	50.0	(-∞:∞)	Integer, real or Complex	Yes

Examples

```
a = sm_z1(S, 50)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

sm_z2() (expmeas)

sm_z2()

This complex measurement determines the impedance that must be presented to the output (port 2) of the network to achieve simultaneous input and output reflections.

Syntax

y = sm_z2(S, Z)

Arguments

Name	Description	Default	Range	Type	Required
S	scattering matrix of a 2-port network	None	(-∞:∞)	Complex	Yes
Z	port impedance	50.0	(-∞:∞)	Integer, real or Complex	Yes

Examples

```
a = sm_z2(S, 50)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

sm_z1() (expmeas)

s_stab_circle()

Returns source (input) stability circles.

Syntax

y = s_stab_circle(S, numOfPts)

Arguments

Name	Description	Default	Range	Type	Required
S	scattering matrix of a 2-port network.	None	(-∞:∞)	Complex	Yes
numOfPts	desired number of points per circle	51	[1:∞)	Integer	No

Examples

```
circleData = s_stab_circle(S, 51) returns the points on the circle(s).
```

Defined in

$HPEESOF_DIR/expressions/ael/circle_fun.ael

See Also

l_stab_circle() (expmeas), *l_stab_region()* (expmeas), *s_stab_circle_center_radius()* (expmeas), *s_stab_region()* (expmeas)

Notes/Equations

Used in Small-signal S-parameter simulations.
This expression generates source stability circles. The circles are defined by the loci of source-reflection coefficients where the magnitude of the load-reflection coefficient is 1. A circle is created for each value of the swept variable(s). This measurement is supported for 2-port networks only.
To find the center and radius of the stability circle use the following expressions:
cir=s_stab_circle(S,2)
cir_center=sum((cir[0::1]) /2)
cir_radius=abs(cir[1]-cir[0]) /2
Alternately, the function s_stab_circle_center_radius() can be used. Use the function s_stab_region(S) to determine the region of stability.

s_stab_circle_center_radius()

Returns the center and radius of the source stability circle

Syntax

s_cr = s_stab_circle_center_radius(S, Type)

Arguments

Name	Description	Default	Range	Type	Required
S	2-port S-Parameters	None	None	Complex	Yes
Type	Type of parameter to return	"both"	["both","center","radius"]	String	No

Examples

```
1. s_cr=s_stab_circle_center_radius(S) returns a 1X2 matrix containing center
and radius
sCirc=expand(circle(s_cr(1), real(s_cr(2)), 51)) returns data for the source
stability circle
2. s_radius=s_stab_circle_center_radius(S, "radius") returns the radius
```

Defined in

$HPEESOF_DIR/expressions/ael/circle_fun.ael

See Also

l_stab_circle() (expmeas), *s_stab_circle()* (expmeas), *l_stab_circle_center_radius()* (expmeas)

Notes/Equations

Used in Small-signal S-parameter simulations
If the argument *Type* is not specified, the function returns complex data. Although radius is of type real, the values are returned as complex. Therefore, when using the returned radius, use the real part. To obtain the radius as a real number, set *Type* to *radius* .

s_stab_region()

This expression returns a string identifying the region of stability of the corresponding source stability circle.

Syntax

y = s_stab_region(S)

Arguments

Name	Description	Default	Range	Type	Required
S	scattering matrix of a 2-port network.	None	(-∞:∞)	Complex	Yes

Examples

```
region = s_stab_region(S) returns "Outside" or "Inside".
```

Defined in

$HPEESOF_DIR/expressions/ael/circle_fun.ael

See Also

l_stab_circle() (expmeas), *l_stab_region()* (expmeas), *s_stab_region()* (expmeas)

Notes/Equations

Used in Small-signal S-parameter simulations

stab_fact()

Returns the Rollett stability factor.

Syntax

k = stab_fact(S)

Arguments

Name	Description	Default	Range	Type	Required
S	scattering matrix of 2-port network	None	(-∞:∞)	Complex	Yes

Examples

```
k = stab_fact(S)
```

Defined in

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

See Also

max_gain() (expmeas), *sm_gamma1()* (expmeas), *sm_gamma2()* (expmeas), *stab_meas()* (expmeas)

Notes/Equations

Given a 2 x 2 scattering matrix between the input and measurement ports, this function calculates the stability factor. The Rollett stability factor is given by:

$k = \{1- |S_{11}|^2- |S_{22}|^2 + |S_{11}*S_{22} - S_{12}*S_{21}|^2\} / \{2*|S_{12}*S_{21}|\}$

The necessary and sufficient conditions for unconditional stability are that the stability factor is greater than unity and the stability measure is positive.

Reference

1. Guillermo Gonzales, Microwave Transistor Amplifiers, second edition, Prentice-Hall, 1997.

stab_meas()

Returns the stability measure

Syntax

k = stab_meas(S)

Arguments

Name	Description	Default	Range	Type	Required
S	scattering matrix of 2-port network	None	(-∞:∞)	Complex	Yes

Examples

```
k = stab_meas(S)
```

Defined in

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

See Also

max_gain() (expmeas), *sm_gamma1()* (expmeas), *sm_gamma2()* (expmeas), *stab_fact()* (expmeas)

Notes/Equations

Given a 2 x 2 scattering matrix between the input and measurement ports, this function calculates the stability measure. The stability measure is given by:

$b = 1 + |S_{11}|^2 - |S_{22}|^2 - |S_{11} * S_{22} - S_{12} * S_{21}|^2$

The necessary and sufficient conditions for unconditional stability are that the stability factor is greater than unity and the stability measure is positive.

Reference

1. Guillermo Gonzales, Microwave Transistor Amplifiers, second edition, Prentice-Hall, 1997.

stoabcd()

This measurement transforms the scattering matrix of a 2-port network to a chain (ABCD) matrix

Syntax

y = stoabcd(S, zRef)

Arguments

Name	Description	Default	Range	Type	Required
S	scattering matrix of a 2-port network	None	(-∞:∞)	Complex	Yes
zRef	reference impedance	50.0	(-∞:∞)	Integer, real or complex	No

Examples

```
a = stoabcd(S, 50)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

abcdtoh() (expmeas), *stoh()* (expmeas), *stoy()* (expmeas)

stoh()

This measurement transforms the scattering matrix of a 2-port network to a hybrid matrix

Syntax

y = stoh(S, zRef)

Arguments

Name	Description	Default	Range	Type	Required
S	scattering matrix of a 2-port network	None	(-∞:∞)	Complex	Yes
zRef	reference impedance	50.0	(-∞:∞)	Integer, real or complex	No

Examples

```
h = stoh(S, 50)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

htos() (expmeas), *stoabcd()* (expmeas), *stoy()* (expmeas)

stos()

Changes the normalizing impedance of a scattering matrix.

Syntax

y = stos(S, zRef, zNew, zy)

Arguments

Name	Description	Default	Range	Type	Required
S	scattering matrix	None	(-∞:∞)	Complex	Yes
zRef	normalizing impedance	50.0	(-∞:∞)	Integer, real or complex	No
zNew	new normalizing impedance	50.0	(-∞:∞)	Integer, real or complex	No
zy	directs the conversion through the Z- or Y-matrix. †	1 (Y-matrix)	[0:1]	Integer	No

† f zy=0, the S-to-S conversion is performed through the Z-matrix. If zy=1, the S-to-S conversion is performed through the Y-matrix

Examples

```
Converts the 50 ohm terminated S-parameters to 75 ohm terminated S-parameters
through the Y-matrix:
a = stos(S, 50, 75, 1)
Converts the 75 ohm terminated S-parameters to 50 ohm terminated S-parameters
through the Y-matrix:
a = stos(S, 75)
Assume that a two-port S-parameter analysis has been done with port 1
terminated in50 ohms, and port 2 in 75 ohms. The expression below converts the
S-parameters at a 50 ohm impedance termination at both ports:
S50 = stos(S, PortZ, 50, 1)
The above converted S-parameters can than be written to a S2P file using the
function write_snp().
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

stoy() (expmeas), *stoz()* (expmeas)

stot()

This function transforms the scattering matrix of a 2-port network to a chain scattering matrix.

Syntax

t = stot(S)

Arguments

Name	Description	Default	Range	Type	Required
S	scattering matrix of a 2-port network	None	(-∞:∞)	Complex	Yes

Examples

```
Tparams = stot(S)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

ttos() (expmeas)

stoy()

This measurement transforms a scattering matrix to an admittance matrix.

Syntax

y = stoy(S, zRef)

Arguments

Name	Description	Default	Range	Type	Required
S	scattering matrix of a 2-port network	None	(-∞:∞)	Complex	Yes
zRef	reference impedance	50.0	(-∞:∞)	Integer, real or complex	No

Examples

```
y = stoy (S, 50.0)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

stoh() (expmeas), *stoz()* (expmeas), *ytos()* (expmeas)

stoz()

This measurement transforms a scattering matrix to an impedance matrix.

Syntax

z = stoz(S, zRef)

Arguments

Name	Description	Default	Range	Type	Required
S	scattering matrix of a 2-port network	None	(-∞:∞)	Complex	Yes
zRef	reference impedance	50.0	(-∞:∞)	Integer, real or complex	No

Examples

```
z = stoz(S, 50)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

stoh() (expmeas), *stoy()* (expmeas), *ztos()* (expmeas)

tdr_step_impedance()

Returns time domain Impedance. This function essentially takes
voltage reflection coefficient and calculates impedance versus time.

Syntax

y = tdr_step_impedance(VSource, Zref, Vincident)

Arguments

Name	Description	Default	Range	Type	Required
VSource	Step voltage source	None	(-∞:∞)	Complex	Yes
zRef	Reference impedance	None	[0:∞)	real	Yes
Vincident	time domain pulse TDR waveform	None	(-∞:∞)	Complex	Yes

Examples

```
x = tdr_step_impedance(VSrc, 50, Vin)
```

Defined in

$HPEESOF_DIR/expressions/ael/DesignGuide_fun.ael

See Also

tdr_sp_gamma() (expmeas), *tdr_sp_imped()* (expmeas)

Notes/Equations

This function takes a time domain pulse TDR waveform, and computes the impedance
versus time. The function requires that a step impulse was applied to the DUT.
Use this function instead of *tdr_step_imped()* (expmeas).

tdr_sp_gamma()

Returns step response. This function calculates time domain response from the S-
parameter measurement directly. Normalization is taken into account.

Syntax

y = tdr_sp_gamma(Sii, delay, Tstart, Tstop, NumPts, window)

Arguments

Name	Description	Default	Range	Type	Required
Sii	Complex reflection coefficient	None	(-∞:∞)	Complex	Yes
delay	delay value	None	[0:∞)	real	Yes
Tstart	Start Time	0	[0:∞)	real	No
Tstop	Stop Time	2 cycles	[0:∞)	real	No
NumPts	Number of points	101	[0:∞)	real ot string	No
Window	Windowing to be applied	0	[0:9] †	real	No

† The window types allowed and their default constants are:

0 = None
1 = Hamming 0.54
2 = Hanning 0.50
3 = Gaussian 0.75
4 = Kaiser 7.865
5 = 8510 6.0 (This is equivalent to the frequency-to-time transformation with normal gate window setting in the 8510 series network analyzer.)
6 = Blackman
7 = Blackman-Harris
8 = 8510-Minimum 0
9 = 8510-Maximum 13

Examples

```
x = tdr_sp_gamma(S(1,1), 0.05ns, -0.2 ns, 3.8 ns, 401, "Hamming")
```

Defined in

$HPEESOF_DIR/expressions/ael/DesignGuide_fun.ael

See Also

tdr_sp_imped() (expmeas), *tdr_step_imped()* (expmeas)

Notes/Equations

This function takes an S-parameter dataset and changes it to a step response. The step response is normalized and used to calculate reflection coefficient vs. time.

tdr_sp_imped()

Returns step response. This function calculates time domain response from the S-parameter measurement directly. Normalization is taken into account.

Syntax

y = tdr_sp_imped(Sii, delay, zRef, Tstart, Tstop, NumPts, window)

Arguments

Arguments

Name	Description	Default	Range	Type	Required
Sii	Complex reflection coefficient	None	(-∞:∞)	Complex	Yes
delay	delay value	None	[0:∞)	real	Yes
zRef	Reference impedance	None	[0:∞)	real	Yes
Tstart	Start Time	0	[0:∞)	real	No
Tstop	Stop Time	2 cycles	[0:∞)	real	No
NumPts	Number of points	101	[0:∞)	real ot string	No
Window	Windowing to be applied	0	[0:7] †	real	No

† The window types allowed and their default constants are:

0 = None
1 = Hamming 0.54
2 = Hanning 0.50
3 = Gaussian 0.75
4 = Kaiser 7.865
5 = 8510 6.0 (This is equivalent to the frequency-to-time transformation with normal gate window setting in the 8510 series network analyzer.)
6 = Blackman
7 = Blackman-Harris
8 = 8510-Minimum 0
9 = 8510-Maximum 13

Examples

```
x = tdr_sp_imped(S(1,1), 0.05ns, 50, -0.2 ns, 3.8 ns, 401, "Hamming")
```

Defined in

$HPEESOF_DIR/expressions/ael/DesignGuide_fun.ael

See Also

tdr_sp_gamma() (expmeas), *tdr_step_imped()* (expmeas)

Notes/Equations

This function takes an S-parameter dataset and changes it to a step response. The step response is normalized and used to calculate reflection coefficient vs. time. This gamma is then used to calculate impedance versus time.

tdr_step_imped()

Returns time domain Impedance. This function essentially takes voltage reflection coefficient and calculates impedance versus time.

Syntax

y = tdr_step_imped(time_waveform, zRef)

Arguments

Name	Description	Default	Range	Type	Required
time_waveform	time domain pulse TDR waveform	None	(-∞:∞)	Complex	Yes
zRef	Reference impedance	None	[0:∞)	real	Yes

Examples

```
x = tdr_step_imped(vout, 50)
```

Defined in

$HPEESOF_DIR/expressions/ael/DesignGuide_fun.ael

See Also

tdr_sp_gamma() (expmeas), *tdr_sp_imped()* (expmeas)

Notes/Equations

1. This function takes a time domain pulse TDR waveform, and computes the impedance versus time. The function also assumes that a step impulse was applied to the DUT, since it normalizes the impedance data to the last time point.
2. Please use *tdr_step_impedance()* (expmeas) function instead of this function.

ttos()

This function transforms the chain scattering matrix of a 2-port network to a scattering matrix.

Syntax

sp = ttos(T)

Arguments

Name	Description	Default	Range	Type	Required
T	scattering matrix of a 2-port network	None	(-∞:∞)	Complex	Yes

Examples

```
sp = ttos(t)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

stot() (expmeas)

unilateral_figure()

Returns the unilateral figure as a real number

Syntax

U = uniltaeral_figure(SParam)

Arguments

Name	Description	Default	Range	Type	Required
SParam	2-Port S-Parameters	None	None	Complex	Yes

Examples

```
sMat={{polar(0.55,-50), polar(0.02,10)}, {polar(3.82,80),polar(0.15,-20)}}
U=uniltaeral_figure(sMat) returns 0.009
U_plus=10*log(1/(1-U)**2) returns 0.081
U_minus=10*log(1/(1+U)**2) returns -0.08
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

Notes/Equations

Used in Small-signal S-parameter simulations.
This function is used to calculate the Unilateral Figure of Merit, which determines whether the simplification can be made in neglecting the effect of S_{12} (unilateral behavior of device). It is calculated as:

$$U = \frac{|S_{11}||S_{12}||S_{21}||S_{22}|}{(1-|S_{11}|^2)(1-|S_{22}|^2)}$$

The error limit on unilateral figure or merit, U is:

$$\frac{1}{(1+U)^2} < \frac{GT}{GTU} < \frac{1}{(1-U)^2}$$

where:
GT is Transducer Gain
GTU is Unilateral Transducer Gain
This function can be used only with 2-Port S-parameters and works only on 1-dimensional or single swept parameter data.

unwrap()

This measurement unwraps a phase by changing an absolute jump greater than jump to its 2*jump complement

Syntax

y = unwrap(phase, jump)

Arguments

Name	Description	Default	Range	Type	Required
phase	swept real variable	None	(-∞:∞)	Real	Yes
jump	absolute jump	180.0	(-∞:∞)	Integer, Real	No

Examples

```
a = unwrap(phase(S21))
a = unwrap(phaserad(S21), pi)
```

Defined in

Built-in

See Also

dev_lin_phase() (expmeas), *diff()* (expmeas), *phase()* (ael), *phasedeg()* (ael), *phaserad()* (ael), *ripple()* (expmeas)

Notes/Equations

The unwrap function requires that the difference between two succesive data should be less than or greater than 2*jump. Otherwise no jump is made for that phase and the original data is maintained. And, if the number of phase data points is one, no phase is unwrapped and the original data is maintained.

v_dc()

Returns the voltage difference.

Syntax

y = v_dc(vPlus, vMinus)

Arguments

Name	Description	Default	Range	Type	Required
vPlus	voltage at the positive output terminal	None	(-∞:∞)	Real, Complex	Yes
vMinus	voltage at the negative output terminal	None	(-∞:∞)	Real, Complex	Yes

Examples

```
y = v_dc(vp, vm)
```

Defined In

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

volt_gain()

Returns the voltage gain

Syntax

y= volt_gain(S, Zs, Zl, Zref)

Arguments

Name	Description	Default	Range	Type	Required
S	2 2 scattering matrix measured with equal terminations of Zref	None	None	Complex	Yes
Zs	input impedance	None	(-∞:∞)	Real, Complex	Yes
Zl	Output impedance	None	(-∞:∞)	Real, Complex	Yes
Zref	reference impedance	50.0	(-∞:∞)	Real, Complex	Yes

Examples

```
a = volt_gain(S, 50, 75)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

pwr_gain() (expmeas), *volt_gain_max()* (expmeas)

Notes/Equations

This function calculates the ratio of the voltage across the load impedance to the voltage applied at the input port of the network. The network-parameter transformation function

stos() can be used to change the normalizing impedance of the scattering matrix.

$$\text{volt_gain} = \frac{S_{21}}{2} \cdot \frac{(1-\Gamma_S)\cdot(1+\Gamma_L)}{(S_{11}\cdot\Gamma_S-1)\cdot(S_{22}\cdot\Gamma_L-1)-S_{12}\cdot S_{21}\cdot\Gamma_S\cdot\Gamma_L}$$

Signal Flow Diagram used for volt_gain Calculation illustrates the volt_gain measurement.

Signal Flow Diagram used for volt_gain Calculation

1. In the S-parameter simulation setup, the source and load impedances must be identical.
2. For a case of unequal source and load impedances, S-parameter analysis should be performed with identical source and load impedances. Voltage gain can then be computed with the actual source and load impedances as the second and third arguments.
   For example, compute voltage gain with Zs=100 and Zl=50. Perform an S-parameter analysis with both the Zs=Zl=50 ohms. The voltage gain is computed as follows:
   volt_gain(S, 100, 50, 50)
   This expression gives the voltage gain when the source impedance is 100 ohms and the load impedance is 50 ohms. The fourth argument in volt_gain is the reference impedance, which is the value of the Z parameter of the Term components used in the S-parameter analysis.

volt_gain_max()

This measurement determines the ratio of the voltage across the load to the voltage available from the source at maximum power transfer.

Syntax

y = volt_gain_max(S, Zs, Zl, Zref)

Arguments

Name	Description	Default	Range	Type	Required
S	2 X 2 scattering matrix	None	(-∞:∞)	Complex	Yes
Zs	input impedance	None	(-∞:∞)	Real, Complex	Yes
Zl	Output impedance	None	(-∞:∞)	Real, Complex	Yes
Zref	reference impedance	50.0	(-∞:∞)	Real, Complex	Yes

Examples

```
vGain = volt_gain_max(S,50,75) - Zref defaults to 50 ohms
vGain1 = volt_gain(S, 50, 75, 75) - Zref = 75 ohms
```

Defined in

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

See Also

pwr_gain() (expmeas), *volt_gain()* (expmeas)

Note/Equations

The network-parameter transformation function stos() can be used to change the normalizing impedance of the scattering matrix.

vswr()

Given a complex reflection coefficient, this measurement returns the voltage standing wave ratio.

Syntax

y = vswr(Sii)

Arguments

Name	Description	Default	Range	Type	Required
Sii	complex reflection coefficient	None	(-∞:∞)	Complex	Yes

Examples

```
y = vswr(S11)
```

Defined in

$HPEESOF_DIR/expressions/ael/rf_system_fun.ael

See Also

yin() (expmeas), *zin()* (expmeas)

write_snp()

Write S-Parameters in Touchstone SnP file format. Returns True or False.

Syntax

y = write_snp(FileName, S, Comment, FreqUnit, DataFormat, Zref, Znorm, ZorY, Precision, Delimiter)

Arguments

Name	Description	Default	Range	Type	Required
FileName	Name or full-path of the S-Parameter file.	None	None	String	Yes
S	S-parameter matrix variable	None	None	Real, Complex	Yes
Comment	Text that is to be written at the top of file.	""	None	String	No
FreqUnit	Frequency Unit	GHz	"Hz", "KHz", "MHz", "GHz", "THz"	String	No
DataFormat	Format of S-Parameter that is to be output	"MA"	"MA", "DB", "RI" †	String	No
Zref	Reference impedance (scalar or vector)	50	(0:∞)	Real	No
Znorm	Normalizing impedance (a scalar value)	50	(0:∞)	Real	No
ZorY	Directs the conversion through Z or Y transform ††	1 (Y-matrix)	[0:1]	Integer	No
Precision	precision of the data	6	[1:64]	Integer	No
Delimiter	Delimiter that separates the data	"\t"	None	String	No

† "MA" = magnitude-phase, "DB" = dB-phase, "RI" = real-imaginary
†† If ZorY=0, the S-to-S conversion is performed through the Z-matrix. If ZorY=1, the S-to-S conversion is performed through the Y-matrix

Examples

```
We assume that a S-Parameter Analysis has been performed.
write_snp("spar_ts.s2p", S, "S-par simulation data", "GHz", "MA", 50)
writes the S-Parameters to the file spar_ts.s2p in mag-phase format
write_snp("spar_ts_1.s2p", S, "S-par simulation data")
writes the S-Parameters to the file spar_ts_1.s2p in default "GHz", mag-phase
format and referenece impedance of 50.0.
We assume that a 2-port S-Parameter Analysis has been performed with source
terminated in 60 ohms and load terminated in 70 ohms:
write_snp("spar_norm_ts.s2p", S, "S-par simulation data", "GHz", "MA", PortZ,
50, 1,9," ")
writes the S-Parameters to the file spar_norm_ts.s2p in "GHz", mag-phase
format, with 9 digit precision and delimited by " " and normalized impedance of
```

```
50.0.
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

stos() (expmeas)

Notes/Equations

The function supports only 1-dimensional S-parameter data. S-parameters can be from 1 to 99 ports. The S-parameters to be written can be normalized by a different impedance through the arguments *Znorm* and *ZorY* . If the argument Znorm is not specified, then the S-parameters are normalized to 50 ohms.
When using this function from the schematic page, the output file will be written in the workspace data directory. When used from the Data Display, the file will be written to the workspace directory.

yin()

Given a reflection coefficient and the reference impedance, this measurement returns the input admittance looking into the measurement ports.

Syntax

y = yin(Sii, Z)

Arguments

Name	Description	Default	Range	Type	Required
Sii	complex reflection coefficient	None	(-∞:∞)	Complex	Yes
zRef	reference impedance	50.0	(-∞:∞)	Integer, real or complex	No

Examples

```
yIN = yin(S11, 50)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

vswr() (expmeas), *zin()* (expmeas)

yopt()

Returns optimum admittance for noise match

Syntax

y = yopt(gammaOpt, zRef)

Arguments

Name	Description	Default	Range	Type	Required
gammaOpt	optimum reflection coefficient	None	(-∞:∞)	Complex	Yes
zRef	reference impedance	50.0	(-∞:∞)	Real, Complex	No

Examples

```
a = yopt(Sopt, 50)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

zopt() (expmeas)

Notes/Equations

Used in Small-signal S-parameter simulations
This complex measurement produces the optimum source admittance for noise matching.
gammaOpt is the optimum reflection coefficient that must be presented at the input of the network to realize the minimum noise figure (NFmin).

ytoabcd()

This measurement transforms an admittance matrix of a 2-port network into a hybrid matrix.

Syntax

a = ytoabcd(Y)

Arguments

Name	Description	Default	Range	Type	Required
Y	2-port admittance matrix	None	(-∞:∞)	Complex	Yes

Examples

```
a = ytoabcd(Y)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

abcdtoh() (expmeas), *htoabcd()* (expmeas)

ytoh()

This measurement transforms an admittance matrix of a 2-port network into a hybrid matrix.

Syntax

h = ytoh(Y)

Arguments

Name	Description	Default	Range	Type	Required
Y	2-port admittance matrix	None	(-∞:∞)	Complex	Yes

Examples

```
h = ytoh(Y)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

htoy() (expmeas), *ytoabcd()* (expmeas)

ytos()

This measurement transforms an admittance matrix into a scattering matrix.

Syntax

z = ytos(Y, zRef)

Arguments

Name	Description	Default	Range	Type	Required
Y	admittance matrix	None	(-∞:∞)	Complex	Yes
zRef	reference impedance	50.0	(-∞:∞)	Integer, real or complex	No

Examples

```
s = ytos(Y, 50.0)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

stoy() (expmeas), *ytoz()* (expmeas)

ytoz()

This measurement transforms an admittance matrix to an impedance matrix.

Syntax

Z = ytoz(Y)

Arguments

Name	Description	Default	Range	Type	Required
Y	admittance matrix	None	(-∞:∞)	Complex	Yes

Examples

```
Z = ytoz(Y)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

ytos() (expmeas), *ztoy()* (expmeas)

zin()

Given a reflection coefficient and the reference impedance, this measurement returns the input impedance looking into the measurement ports.

Syntax

z = zin(Sii, Z)

Arguments

Name	Description	Default	Range	Type	Required
Sii	complex reflection coefficient.	None	(-∞:∞)	Complex	Yes
zRef	reference impedance	50.0	(-∞:∞)	Integer, real or complex	No

Examples

```
zIN = zin(S11, 50.0)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

vswr() (expmeas), *yin()* (expmeas)

zopt()

Returns optimum impedance for noise match

Syntax

y = zopt(gammaOpt, zRef)

Arguments

Name	Description	Default	Range	Type	Required
gammaOpt	optimum reflection coefficient	None	(-∞:∞)	Complex	Yes
zRef	reference impedance	50.0	(-∞:∞)	Real, Complex	No

Examples

```
a = zopt(Sopt, 50)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

yopt() (expmeas)

Notes/Equations

Used in Small-signal S-parameter simulations.
This complex measurement produces the optimum source impedance for noise matching.
gammaOpt is the optimum reflection coefficient that must be presented at the input of the network to realize the minimum noise figure (NFmin).

ztoabcd()

This measurement transforms an impedance matrix of a 2-port network into a chain (ABCD) matrix.

Syntax

a = ztoabcd(Z)

Arguments

Name	Description	Default	Range	Type	Required
Z	2-port impedance matrix	None	(-∞:∞)	Complex	Yes

Examples

```
a = ztoabcd(Z)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

abcdtoz() (expmeas), *ytoabcd()* (expmeas), *ztoh()* (expmeas)

ztoh()

This measurement transforms an impedance matrix of a 2-port network into a hybrid matrix.

Syntax

h = ztoh(Z)

Arguments

Name	Description	Default	Range	Type	Required
Z	2-port impedance matrix	None	(-∞:∞)	Complex	Yes

Examples

```
h = ztoh(Z)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

htoz() (expmeas), *ytoh()* (expmeas), *ztoabcd()* (expmeas)

ztos()

This measurement transforms an impedance matrix to a scattering matrix.

Syntax

sp = ztos(Z, zRef)

Arguments

Name	Description	Default	Range	Type	Required
Z	impedance matrix	None	(-∞:∞)	Complex	Yes
zRef	reference impedance	50.0	(-∞:∞)	Integer, real or complex	No

Examples

```
s = ztos(Z, 50.0)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

stoz() (expmeas), *ytoz()* (expmeas), *ztoy()* (expmeas)

ztoy()

This measurement transforms an impedance matrix to an admittance matrix.

Syntax

y = ztoy(Z)

Arguments

Name	Description	Default	Range	Type	Required
Z	impedance matrix	None	(-∞:∞)	Complex	Yes

Examples

```
y = ztoy(Z)
```

Defined in

$HPEESOF_DIR/expressions/ael/network_fun.ael

See Also

stoz() (expmeas), *ytos()* (expmeas), *ztoy()* (expmeas)

Statistical Analysis Functions

This section describes the statistical analysis functions in detail. The functions are listed in alphabetical order.

- *cdf()* (expmeas)
- *cross corr()* (expmeas)
- *fun 2d outer()* (expmeas)
- *histogram()* (expmeas)
- *histogram multiDim()* (expmeas)
- *histogram sens()* (expmeas)
- *histogram stat()* (expmeas)
- *interpolate swept data()* (expmeas)
- *lognorm dist1D()* (expmeas)
- *lognorm dist inv1D()* (expmeas)
- *mean()* (expmeas)
- *mean outer()* (expmeas)
- *median()* (expmeas)
- *moving average()* (expmeas)
- *norm dist1D()* (expmeas)
- *norm dist inv1D()* (expmeas)
- *norms dist1D()* (expmeas)
- *norms dist inv1D()* (expmeas)
- *pdf()* (expmeas)
- *stddev()* (expmeas)
- *stddev outer()* (expmeas)
- *uniform dist1D()* (expmeas)
- *uniform dist inv1D()* (expmeas)
- *yield sens()* (expmeas)

cdf()

Returns the cumulative distribution function (CDF)

Syntax

y = cdf(data, numBins, minBin, maxBin)

Arguments

Name	Description	Default	Range	Type	Required
data	the signal	None	(-∞:∞)	Real	Yes
numBins	number of subintervals or bins used to measure CDF	log(numOfPts)/log(2.0)	[1:∞)	Real	No
minBin	beginning of the evaluation of the CDF	minimum value of the data	(-∞:∞)	Real	No
maxBin	end of the evaluation of the CDF	maximum value of the data	(-∞:∞)	Real	No

Examples

```
y = cdf(data)
y = cdf(data, 20)
```

Defined in

$HPEESOF_DIR/expressions/ael/statistical_fun.ael

See Also

histogram() (expmeas), *pdf()* (expmeas), *yield_sens()* (expmeas)

Notes/Equations

This function measures the cumulative distribution function of a signal. This function can only be used by entering an equation (Eqn) in the Data Display window.

cross_corr()

Returns the cross-correlation

Syntax

y = cross_corr(v1, v2)

Arguments

Name	Description	Default	Range	Type	Required
v1	one-dimensional data	None	(-∞:∞)	Real	Yes
v2	one-dimensional data	None	(-∞:∞)	Real	Yes

Examples

```
v1 = qpsk..videal[1]
v2 = qpsk..vout[1]
x_corr_v1v2 = cross_corr(v1, v2)
auto_corr_v1 = cross_corr(v1, v1)
```

Defined in

$HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael

fun_2d_outer()

Applies a function to the outer dimension of two-dimensional data

Syntax

y = fun_2d_outer(data, fun)

Arguments

Name	Description	Default	Range	Type	Required
data	two-dimensional data	None	(-∞:∞)	Integer, Real, Complex	Yes
fun	name of function (usually mean, max, or min) that will be applied to the outer dimension of the data	None	None	string	Yes

Examples

```
y = fun_2d_outer(data, min)
```

Defined in

$HPEESOF_DIR/expressions/ael/statistcal_fun.ael

See Also

max_outer() (expmeas), *mean_outer()* (expmeas), *min_outer()* (expmeas)

Notes/Equations

Used in max_outer(), mean_outer(), min_outer() functions.
Functions such as mean, max, and min operate on the inner dimension of two-dimensional data. The function fun_2d_outer enables these functions to be applied to the outer dimension. As an example, assume that a Monte Carlo simulation of an amplifier was run, with 151 random sets of parameter values, and that for each set the S-parameters were simulated over 26 different frequency points. S21 becomes a [151 Monte Carlo iteration X 26 frequency] matrix, with the inner dimension being frequency, and the outer dimension being Monte Carlo index. Now, assume that it is desired to know the mean value of the S-parameters at each frequency. Inserting an equation mean(S21) computes the mean value of S21 at each Monte Carlo iteration. If S21 is simulated from 1 to 26 GHz, it computes the mean value over this frequency range, which usually is not very useful. The function fun_2d_outer allows the mean to be computed over each element in the outer dimension.

histogram()

Generates a histogram representation. This function creates a histogram that represents data

Syntax

y = histogram(data, numBins, minBin, maxBin)

Arguments

Name	Description	Default	Range	Type	Required
data	signal(must be one-dimensional)	None	(-∞:∞)	Integer, Real	Yes
numBins	number of subintervals or bins used to measure the histogram.	log(numOfPts)/log(2.0)	[1:∞)	Integer	No
minBin	beginning of the evaluation of the histogram	minimum value of the data	(-∞:∞)	Real	No
maxBin	end of the evaluation of the histogram	maximum value of the data	(-∞:∞)	Real	No

Examples

```
y = histogram(data)
y = histogram(data, 20)
If you have performed a parameter sweep such that the first argument (data) in
the histogram function is a function of two independent variables, then you
must reduce the dimensionality of data before using it in the histogram
function. For example, if you run a Monte Carlo simulation on the S-parameters
of a circuit, S21 would be a function of both the Monte Carlo index and the
frequency (assuming you have swept frequency).
So, you could plot the histogram of S21 at the 100th frequency in the sweep by
using:
y = histogram(dB(S21[::,99]))
See also $HPEESOF_DIR/examples/Tutorial/yldex1_wrk
(see measurement_hist.dds and worstcase_measurement_hist.dds)
```

Defined in

Built in

See Also

cdf() (expmeas), *pdf()* (expmeas), *yield_sens()* (expmeas), *histogram_multiDim()* (expmeas), *histogram_stat()* (expmeas)

Notes/Equations

This function can only be used by entering an equation (Eqn) in the Data Display window. Use the *histogram_multiDim()* function for multi-dimensional data.

histogram_multiDim()

Collapses the multi-dimensional data down to one-dimensional data and applies the histogram to the one-dimensional data.

Syntax

y = histogram_multiDim(data, normalized, numBins, minBin, maxBin)

Arguments

Name	Description	Default	Range	Type	Required
data	the signal	None	(-∞:∞)	Real	Yes
normalized	sets normalization of data †	no	"no", "yes"	String	No
numBins	number of subintervals or bins used to measure CDF	log(numOfPts)/log(2.0)	[1:∞)	Real	No
minBin	beginning of the evaluation of the CDF	minimum value of the data	(-∞:∞)	Real	No
maxBin	end of the evaluation of the CDF	maximum value of the data	(-∞:∞)	Real	No

† When normalized is set to "yes", the histogram is generated with percent on the Y-axis instead of the number of outcomes.

Examples

```
Given monte carlo analysis results for the S12. It is two-dimensional data: the
outer sweep is mcTrial; the inner sweep is the frequency from 100 MHz to 500
MHz.
Histogram_multiDim_S12 = histogram_multiDim(S12)
See also $HPEESOF_DIR/examples/Tutorial/yldex1_wrk
(see measurement_hist.dds and worstcase_measurement_hist.dds)
```

Defined in

$HPEESOF_DIR/expressions/ael/statistical_fun.ael

See Also

collapse() (expmeas), *histogram()* (expmeas)

histogram_sens()

Produces the yield sensitivity histogram displaying the sensitivity of a measurement statistical response to a selected statistical variable. The function is mainly applied to the statistical analysis (Monte Carlo/Yield/YieldOpt) results.

Syntax

y = histogram_sens(data, sensitivityVar, goalMin, goalMax, innermostIndepLow, innermostIndepHigh, numBins)

Arguments

Name	Description	Default	Range	Type	Required
data	statistical response	None	(-∞:∞)	Real	Yes
sensitivityVar	selected statistical variable	None	None	String	Yes
goalMin	specifies the performance range †	None	(-∞:∞)	Real	No
goalMax	specifies the performance range †	None	(-∞:∞)	Real	No
innermostIndepLow	specifies the low value of the subrange of data with the inner most sweep variable	None	(-∞:∞)	Real	Yes
innermostIndepHigh	specifies the high value of the subrange of data with the inner most sweep variable	None	(-∞:∞)	Real	Yes
numBins	number of sub-intervals or bins used to measure the histogram.	log(numOfPts)/log(2.0)	(1:∞)	Real	No

† The yield is 1 inside the range, and the yield is zero outside the range. Note that while goalMin and goalMax are optional arguments, at least one of them must be specified.

Examples

```
Given monte carlo analysis results for the S11. It is two-dimensional data: the
outer sweep is mcTrial; the inner sweep is the frequency from 100 MHz to 500
MHz.
The design wants the maximum of db(S11) is -18.0dB in the frequency range of
200 MHz to 400 MHz. The yield sensitivity of such performance to statistical
variable "C1v" can be calculated as:
Histogram_sens_S11 = histogram_sens(dB(S11),C1v,,-18.0,200MHz,400MHz)
```

Defined in

$HPEESOF_DIR/expressions/ael/statistical_fun.ael

See Also

histogram() (expmeas), *histogram_multiDim()* (expmeas), *histogram_stat()* (expmeas), *build_subrange()* (expmeas), *yield_sens()* (expmeas)

histogram_stat()

Reduces the histogram of a subrange of the multi-dimension data. It first calls build_subrange() to build the subrange of a mulit-dimension data, then calls histogram_multiDim() to produce the histogram.

Syntax

y = histogram_stat(data, normalized, innermostIndepLow, innermostIndepHigh, numBins, minBin, maxBin)

Arguments

Name	Description	Default	Range	Type	Required
data	statistical data to be analyzed	None	(-∞:∞)	Real	Yes
normalized	sets normalization of data †	no	"no", "yes"	String	No
innermostIndepLow	specifies the low value of the subrange of data with the inner most sweep variable	None	(-∞:∞)	Real	Yes
innermostIndepHigh	specifies the high value of the subrange of data with the inner most sweep variable	None	(-∞:∞)	Real	Yes
numBins	number of subintervals or bins used to measure histogram	log(numOfPts)/log(2.0)	[1:∞)	Real	No
minBin	beginning of the evaluation of the histogram	minimum value of the data	(-∞:∞)	Real	No
maxBin	end of the evaluation of the histogram	maximum value of the data	(-∞:∞)	Real	No

† When normalized is set to "yes", the histogram is generated with percent on the Y-axis instead of the number of outcomes.

Examples

```
Given monte carlo analysis results for the S12. It is two-dimensional data: the
outer sweep is mcTrial; the inner sweep is the frequency from 100 MHz to 500
MHz.
Histogram_stat_S12 = histogram_stat(S12,,200MHz,400MHz)
```

Defined in

$HPEESOF_DIR/expressions/ael/statistical_fun.ael

See Also

histogram() (expmeas), *histogram_multiDim()* (expmeas), *build_subrange()* (expmeas)

interpolate_swept_data()

Returns a function or trace's y-axis value or values corresponding to a specific x-axis value

Syntax

Interpolated_Pout=interpolate_swept_data(original_data_Y_vs_X, desired_X, interpStepSize, interpType)

Arguments

Name	Description	Default	Range	Type	Required
original_data_Y_vs_X	one-, two-, or three-dimensional data	None	(-∞:∞)	Real	Yes
desired_X	X-axis search value, a single real or integer number	None	(-∞:∞)	Real or Integer	Yes
interpStepSize	interpolation step size, a single real or integer number	0.05	(-∞:∞)	Real or Integer	No
interpType	type of interpolation	None	"linear", "spline", or "cubic"	String	Yes

Examples

```
Pout=dBm(Vload_fundHB)
Pin=indep(Vload_fundHB)
Gain_HB=Pout-Pin
Gain_vs_Pout=vs(Gain,Pout)
Search_Value=30
interpStepSize=0.05
interpType="spline"
Gain_at_Specified_Pout=interpolate_swept_data(Gain_vs_Pout, Search_Value,
interpStepSize, interpType)
```

Notes/Equations

This function would be useful for the following application. You have run a swept input power simulation of an amplifier.
You plot a trace that is gain versus output power or gain compression versus output power. Use this function to find the gain or gain compression at a particular output power.
This function is applicable even if you have run a Monte Carlo analysis or swept some parameter. It enables you to see the distribution of the gain or gain compression when the amplifier is delivering a
particular output power.

lognorm_dist_1D()

Returns a lognormal distribution: either its probability density function (pdf), or cumulative distribution function (cdf).

Syntax

y = lognorm_dist1D(data, m, s, absS, is_cpf)

Arguments

Name	Description	Default	Range	Type	Required
data	number or a one-dimensional data	None	(-∞:∞)	Real	Yes
m	mean for the lognormal distribution	None	(-∞:∞)	Integer, Real	Yes
s	standard deviation for the lognormal distribution	None	(-∞:∞)	Integer, Real	Yes
absS	specifies absolute or relative standard deviation	1	[0:∞) †	Integer	No
is_cpf	specifies cdf or pdf og lognormal distribution	0	[0:∞) † †	Integer	No

† When absS is not equal to "0", s is the absolute standard deviation; otherwise, s is the relative standard deviation
† † When is_cdf is not equal to "0", the function returns the cdf of the lognormal

distribution. Otherwise, it returns the pdf of the lognormal distribution.

Examples

```
X = 0.5
X_pdf= lognorm_dist1D(X, 2.0,0.2, 1, 0)
X_cdf = lognorm_dist1D(X,2.0,0.1, 0, 1)
XX=[-3.9::0.1::3.9]
XX_pdf = lognorm_dist1D(XX,2.0, 0.2,1,0)
XX_cdf = lognorm_dist1D(XX,2.0,0.2,0,1)
```

Defined in

$HPEESOF_DIR/expressions/ael/statistical_fun.ael

lognorm_dist_inv1D()

Returns the inverse of the cumulative distribution function (cdf) for a lognormal distribution

Syntax

y = lognorm_dist_inv1D(data, m, s, absS)

Arguments

Name	Description	Default	Range	Type	Required
data	real number representing the cumulative probability	None	[0:1]	Real	Yes
m	mean for the lognormal distribution	None	(-∞:∞)	Integer, Real	Yes
s	standard deviation for the lognormal distribution	None	(-∞:∞)	Integer, Real	Yes
absS	specifies absolute or relative standard deviation	1	[0:∞) †	Integer	No

† absS is not equal to "0", s is the absolute standard deviation; otherwise, s is the relative standard deviation

Examples

```
X_cpf = 0.5
X= lognorm_dist_inv1D(X_cpf, 2.0, 0.2, 1)
XX_cpf=[0.0::0.01::1.0]
XX= lognorm_dist_inv1D(XX_cpf, 2.0, 0.2, 1)
```

Defined in

$HPEESOF_DIR/expressions/ael/statistical_fun.ael

mean()

Returns the mean

Syntax

y = mean(x)

Arguments

Name	Description	Default	Range	Type	Required
x	data to find mean	None	(-∞:∞)	Integer, Real, Complex	Yes

Examples

```
a = mean([1, 2, 3]) returns 2
```

Defined in

Built in

See Also

cum_prod() (expmeas), *cum_sum()* (expmeas), *max()* (expmeas), *min()* (expmeas), *prod()* (expmeas), *sum()* (expmeas)

mean_outer()

Computes the mean across the outer dimension of two-dimensional data

Syntax

y = mean_outer(data)

Arguments

Name	Description	Default	Range	Type	Required
data	2-dimensional data to find mean	None	(-∞:∞)	Integer, Real, Complex	Yes

Examples

```
a = mean_outer(data)
```

Defined in

$HPEESOF_DIR/expressions/ael/statistcal_fun.ael

See Also

fun_2d_outer() (expmeas), *max_outer()* (expmeas), *min_outer()* (expmeas)

Notes/Equations

The mean function operates on the inner dimension of two-dimensional data. The mean_outer function just calls the fun_2d_outer function, with mean being the applied operation. As an example, assume that a Monte Carlo simulation of an amplifier was run, with 151 random sets of parameter values, and that for each set the S-parameters were simulated over 26 different frequency points. S21 becomes a [151 Monte Carlo iteration X 26 frequency] matrix, with the inner dimension being frequency, and the outer dimension being Monte Carlo index. Now, assume that it is desired to know the mean value of the S-parameters at each frequency. Inserting an equation mean(S21) computes the mean value of S21 at each Monte Carlo iteration. If S21 is simulated from 1 to 26 GHz, it computes the mean value over this frequency range, which usually is not very useful. Inserting an equation mean_outer(S21) computes the mean value of S21 at each Monte Carlo frequency.

median()

Returns the median

Syntax

y = median(x)

Arguments

Name	Description	Default	Range	Type	Required
data	data to find the median	None	(-∞:∞)	Real	Yes

Examples

```
a = median([1, 2, 3, 4]) returns 2.5
```

Defined in

$HPEESOF_DIR/expressions/ael/statistcal_fun.ael

See Also

mean() (expmeas), *sort()* (expmeas)

Notes/Equations

This function can only be used by entering an equation (Eqn) in the Data Display window.

moving_average()

Returns the moving_average of a sequence of data

Syntax

y = moving_average(Data, NumPoints)

Arguments

Name	Description	Default	Range	Type	Required
Data	one or multi-dimensional sequence of numbers	None	(-∞:∞)	Integer, real or complex	Yes
NumPoints	Number of points to be averaged together	None	[1:∞) †	Integer	Yes

† NumPoints must be an odd number. If even, the value is increased to the next odd number. If greater or equal to the number of data points, the value is set to number of data points - 1 for even number of data points. For odd number of data points, NumPoints is set to number of data points.

Examples

```
a = moving_average([1, 2, 3, 7, 5, 6, 10], 3) returns [1, 2, 4, 5, 6, 7, 10]
```

Defined in

Built in

Notes/Equations

The first value of the smoothed sequence is the same as the original data. The second value is the average of the first three. The third value is the average of data elements 2, 3, and 4, etc. If NumPoints were set to 7, for example, then the first value of the smoothed sequence would be the same as the original data. The second value would be the average of the first three original data points. The third value would be the average of the first five data points, and the fourth value would be the average of the first seven data points. Subsequent values in the smoothed array would be the average of the seven closest neighbors. The last points in the smoothed sequence are computed in a way similar to the first few points. The last point is just the last point in the original sequence. The second from last point is the average of the last three points in the original sequence. The third from the last point is the average of the last five points in the original sequence, etc.

norm_dist_1D()

Returns a normal distribution: either its probability density function (pdf), or cumulative distribution function (cdf)

Syntax

y = norm_dist1D(data, m, s, absS, is_cdf)

Arguments

Name	Description	Default	Range	Type	Required
data	number or a one-dimensional data	None	(-∞:∞)	Real	Yes
m	mean for the normal distribution	None	(-∞:∞)	Integer, Real	Yes
s	standard deviation for the normal distribution	None	(-∞:∞)	Integer, Real	Yes
absS	specifies absolute or relative standard deviation	1	[0:∞) †	Integer	No
is_cdf	specifies cdf or pdf of normal distribution	0	[0:∞) † †	Integer	No

† When absS is not equal to "0", s is the absolute standard deviation; otherwise, s is the relative standard deviation
† † When is_cdf is not equal to "0", the function returns the cdf of the normal distribution. Otherwise, it returns the pdf of the normal distribution.

Examples

```
X = 0.5
X_pdf= norm_dist1D(X, 0, 1, 1, 0)
X_cdf = norm_dist1D(X,0, 1, 1, 1)
XX=[-3.9::0.1::3.9]
XX_pdf = norm_dist1D(XX,5.0, 0.5,1,0)
XX_cdf = norm_dist1D(XX,5.0,0.1,0,1)
```

Defined in

$HPEESOF_DIR/expressions/ael/statistical_fun.ael

See Also

norm_dist_inv1D() (expmeas), *norms_dist_inv1D()* (expmeas), *norms_dist1D()* (expmeas), *lognorm_dist_inv1D()* (expmeas), *lognorm_dist1D()* (expmeas), *uniform_dist_inv1D()* (expmeas), *uniform_dist1D()* (expmeas)

norm_dist_inv1D()

Returns the inverse of the cumulative distribution function (cdf) for a normal distribution

Syntax

y = norm_dist_inv1D(data, m, s, absS)

Arguments

Name	Description	Default	Range	Type	Required
data	represents the cumulative probability	None	[0:1]	Real	Yes
m	mean for the normal distribution	None	(-∞:∞)	Integer, Real	Yes
s	standard deviation for the normal distribution	None	(-∞:∞)	Integer, Real	Yes
absS	specifies absolute or relative standard deviation	1	[0:∞) †	Integer	No

† When absS is not equal to "0", s is the absolute standard deviation; otherwise, s is the

relative standard deviation

Examples

```
X_cpf = 0.5
X= norm_dist_inv1D(X_cpf, 0, 1, 1) will be equal to 0.0
XX_cpf=[0.0::0.01::1.0]
XX= norm_dist_inv1D(XX_cpf, 5.0, 0.5, 1)
```

Defined in

$HPEESOF_DIR/expressions/ael/statistical_fun.ael

See Also

norm_dist1D() (expmeas), *norms_dist_inv1D()* (expmeas), *norms_dist1D()* (expmeas), *lognorm_dist_inv1D()* (expmeas), *lognorm_dist1D()* (expmeas), *uniform_dist_inv1D()* (expmeas), *uniform_dist1D()* (expmeas)

norms_dist1D()

Returns the standard normal distribution: either its probability density function (pdf), or cumulative distribution function (cdf).

Syntax

y = norms_dist1D(data, is_cdf)

Arguments

Name	Description	Default	Range	Type	Required
data	number or a one-dimensional data	None	(-∞:∞)	Real	Yes
is_cdf	specifies cdf or pdf of standard normal distribution	0	[0:∞) †	Integer	No

† When is_cdf is not equal to "0", the function returns the cdf of the standard normal distribution. Otherwise, it returns the pdf of the standard normal distribution.

Examples

```
X = 0.5
X_pdf= norms_dist1D(X, 0)
X_cdf = norms_dist1D(X,1)
XX=[-3.9::0.1::3.9]
XX_pdf = norms_dist1D(XX,0)
XX_cdf = norms_dist1D(XX,1)
```

Defined in

$HPEESOF_DIR/expressions/ael/statistical_fun.ael

See Also

norm_dist1D() (expmeas), *norm_dist_inv1D()* (expmeas), *norms_dist_inv1D()* (expmeas), *lognorm_dist_inv1D()* (expmeas), *lognorm_dist1D()* (expmeas), *uniform_dist_inv1D()* (expmeas), *uniform_dist1D()* (expmeas)

norms_dist_inv1D()

Returns the inverse of the cumulative distribution function (cdf) for a standard normal distribution

Syntax

y = norms_dist_inv1D(data)

Arguments

Name	Description	Default	Range	Type	Required
data	number represents the cumulative probability	None	[0:1]	Integer, Real	Yes

Examples

```
X_cpf = 0.5
X= norms_dist_inv1D(X_cpf) will be equal to 0.0
XX_cpf=[0.0::0.01::1.0]
XX= norms_dist_inv1D(XX_cpf)
```

Defined in

$HPEESOF_DIR/expressions/ael/statistical_fun.ael

See Also

norm_dist1D() (expmeas), *norm_dist_inv1D()* (expmeas), *norms_dist1D()* (expmeas), *lognorm_dist_inv1D()* (expmeas), *lognorm_dist1D()* (expmeas), *uniform_dist_inv1D()* (expmeas), *uniform_dist1D()* (expmeas)

pdf()

Returns a probability density function (PDF)

Syntax

y = pdf(data, numBins, minBin, maxBin)

Arguments

Name	Description	Default	Range	Type	Required
data	the signal	None	(-∞:∞)	Real	Yes
numBins	number of subintervals or bins used to measure PDF	log(numOfPts)/log(2.0)	[1:∞)	Real	No
minBin	beginning of the evaluation of the PDF	minimum value of the data	(-∞:∞)	Real	No
maxBin	end of the evaluation of the PDF	maximum value of the data	(-∞:∞)	Real	No

Examples

```
y = pdf(data)
y = pdf(data, 20)
```

Defined in

$HPEESOF_DIR/expressions/ael/statistical_fun.ael

See Also

cdf() (expmeas), *histogram()* (expmeas), *yield_sens()* (expmeas)

Notes/Equations

This function measures the probability density function of a signal. This function can only be used by entering an equation (Eqn) in the Data Display window.

stddev()

This function calculates the standard deviation of the data

Syntax

y = stddev(data, flag)

Arguments

Name	Description	Default	Range	Type	Required
data	data to find the stddev	None	(-∞:∞)	Real	Yes
flag	indicates how stddev normalizes	0	[0:1] †	Integer	No

† When flag equals 0, the stddev normalizes by N-1, where N is the length of the data sequence. Otherwise, stddev normalizes by N.

Examples

```
a = stddev(data)
a = stddev(data, 1)
```

Defined in

$HPEESOF_DIR/expressions/ael/statistical_fun.ael

See Also

mean() (expmeas)

Notes/Equations

This function can only be used by entering an equation (Eqn) in the Data Display window.

stddev_outer()

Computes the stddev across the outer dimension of two-dimensional data

Syntax

y = stddev_outer(x, flag)

Arguments

Name	Description	Default	Range	Type	Required
data	data to find the stddev	None	(-∞:∞)	Real	Yes
flag	indicates how stddev normalizes	0	[0:1] †	Integer	No

† When flag equals 0, the stddev normalizes by N-1, where N is the length of the data sequence. Otherwise, stddev normalizes by N.

Examples

```
a = stddev_outer(data)
a = stddev_outer(data, 1)
```

Defined In

$HPEESOF_DIR/expressions/ael/statistical_fun.ael

See Also

fun_2d_outer() (expmeas), *max_outer()* (expmeas), *mean_outer()* (expmeas), *min_outer()* (expmeas)

Notes/Equations

The stddev_outer() function operates on the inner dimension of two-dimensional data. This function just calls the fun_2d_outer function, with stddev being the applied operation. As an example, assume that a Monte Carlo simulation of an amplifier was run, with 151 random sets of parameter values, and that for each set the S-parameters were simulated over 26 different frequency points. S21 becomes a [151 Monte Carlo iteration X 26 frequency] matrix, with the inner dimension being frequency, and the outer dimension being Monte Carlo index. Now, assume that it is desired to know the stddev value of the S-parameters at each frequency. Inserting an equation stddev(S21) computes the stddev value of S21 at each Monte Carlo iteration. If S21 is simulated from 1 to 26 GHz, it computes the stddev value over this frequency range, which usually is not very useful. Inserting an equation stddev_outer(S21) computes the stddev value of S21 at each Monte Carlo frequency.

uniform_dist1D()

Returns a uniform distribution: either its probability density function (pdf), or cumulative distribution function (cdf)

Syntax

y = uniform_dist1D(data, A, B, is_cdf)

Arguments

Name	Description	Default	Range	Type	Required
data	number or a one-dimensional data	None	(-∞:∞)	Integer, Real	Yes
A	uniform distributed range	None	[0:∞)	Integer, real	Yes
B	uniform distributed range	None	[0:∞)	Integer, real	Yes
is_cdf	specifies cdf or pdf of standard uniform distribution	0	[0:∞) †	Integer	No

† When is_cdf is not equal to "0", the function returns the cdf of the uniform distribution. Otherwise, it returns the pdf of the uniform distribution.

Examples

```
X = 0.5
X_pdf= uniform_dist1D(X, 0.0,1.0, 0)
X_cdf = uniform_dist1D(X,0.0,1.0, 1)
XX=[-3.9::0.1::3.9]
XX_pdf = uniform_dist1D(XX,0.0,5.0,0)
XX_cdf = uniform_dist1D(XX,0.0,5.0,1)
```

Defined in

$HPEESOF_DIR/expressions/ael/statistical_fun.ael

See Also

norm_dist1D() (expmeas), *norm_dist_inv1D()* (expmeas), *norms_dist_inv1D()* (expmeas), *norms_dist1D()* (expmeas), *lognorm_dist_inv1D()* (expmeas),

lognorm_dist1D() (expmeas), *uniform_dist_inv1D()* (expmeas)

uniform_dist_inv1D()

Returns the inverse of the cumulative distribution function (cdf) for a uniform distribution

Syntax

y = uniform_dist_inv1D(data, A, B)

Arguments

Name	Description	Default	Range	Type	Required
data	represents the cumulative probability	None	[0:1]	Integer, Real	Yes
A	uniform distributed range	None	[0:∞)	Integer, real	Yes
B	uniform distributed range	None	[0:∞)	Integer, real	Yes

Examples

```
X_cpf = 0.5
X= uniform_dist_inv1D(X_cpf, 0.0, 1.5)
XX_cpf=[0.0::0.01::1.0]
XX= uniform_dist_inv1D(XX_cpf, 0.0, 1.5)
```

Defined in

$HPEESOF_DIR/expressions/ael/statistical_fun.ael

See Also

norm_dist1D() (expmeas), *norm_dist_inv1D()* (expmeas), *norms_dist_inv1D()* (expmeas), *norms_dist1D()* (expmeas), *lognorm_dist_inv1D()* (expmeas), *lognorm_dist1D()* (expmeas), *uniform_dist1D()* (expmeas)

yield_sens()

Returns the yield as a function of a design variable

Syntax

y = yield_sens(pf_data, numBins)

Arguments

Name	Description	Default	Range	Type	Required
pf_data	binary-valued scalar data set indicating the pass/fail status of each value of a companion independent variable	None	[0-1]	Integer	Yes
numBins	number of subintervals or bins used to measure yield_sens	log(numOfPts)/log(2.0)	[1:∞)	Real	No

Examples

```
a = yield_sens(pf_data)
```

```
a = yield_sens(pf_data, 20)
```

Defined in

$HPEESOF_DIR/expressions/ael/statistical_fun.ael

See Also

cdf() (expmeas), *histogram()* (expmeas), *pdf()* (expmeas)

Notes/Equations

Used in Monte Carlo simulation.
This function measures the yield as a function of a design variable. For more information and an example refer to " *Creating a Sensitivity Histogram* " in the *Optimization and Statistical Design* documentation.
This function can only be used by entering an equation (Eqn) in the Data Display window.

Transient Analysis Functions

This section describes the transient analysis functions in detail. The functions are listed in alphabetical order.

- *constellation()* (expmeas)
- *cross()* (expmeas)
- *fspot()* (expmeas)
- *ifc tran()* (expmeas)
- *ispec tran()* (expmeas)
- *pfc tran()* (expmeas)
- *pspec tran()* (expmeas)
- *pt tran()* (expmeas)
- *vfc tran()* (expmeas)
- *vspec tran()* (expmeas)
- *vt tran()* (expmeas)

Working with Transient Data

Transient analysis produces real voltages and currents as a function of time. A single analysis produces 1-dimensional data. Sections of time-domain waveforms can be indexed by using a sequence within "[ ]".

constellation()

Generates the constellation diagram from Circuit Envelope, Transient, or Ptolemy simulation I and Q data

Syntax

Const = constellation(i_data, q_data, symbol_rate, delay)

Arguments

Name	Description	Default	Range	Type	Required
i_data	in-phase component of data versus time of a single complex voltage spectral component (for example, the fundamental) †	None	(-∞:∞)	Complex	Yes
q_data	quadrature-phase component of data versus time of a single complex voltage spectral component (for example, the fundamental) †	None	(-∞:∞)	Real	Yes
symbol_rate	symbol rate of the modulation signal	None	[0:∞)	Real	Yes
delay	delay value † †	None	[0:∞)	Real	No

† this could be a baseband signal instead, but in either case it must be real-valued versus time.
† † (if nonzero) throws away the first delay = N seconds of data from the constellation plot. It is also used to interpolate between simulation time points, which is necessary if the optimal symbol-sampling instant is not exactly at a simulation time point. Usually this parameter must be nonzero to generate a constellation diagram with the smallest grouping of sample points.

Examples

```
Rotation = -0.21
Vfund =vOut[1] * exp(j * Rotation)
delay =1/sym_rate[0, 0] - 0.5 * tstep[0, 0]
Vimag = imag(Vfund)
Vreal = real(Vfund)
Const = constellation(Vreal, Vimag, sym_rate[0, 0], delay)
```

where Rotation is a user-selectable parameter that rotates the constellation by that many radians, and vOut is the named connection at a node. The parameter delay can be a numeric value, or in this case an equation using sym_rate, the symbol rate of the modulated signal, and tstep, the time step of the simulation. If these equations are to be used in a Data Display window, sym_rate and tstep must be defined by means of a variable (VAR) component, and they must be passed into the dataset as follows: Make the parameter Other visible on the Envelope simulation component, and edit it so that
Other = OutVar = sym_rate OutVar = tstep
In some cases, it may be necessary to experiment with the value of delay to get the constellation diagram with the tightest points.
Note vOut is a named connection on the schematic. Assuming that a Circuit Envelope simulation was run, vOut is output to the dataset as a two-dimensional matrix. The first dimension is time, and there is a value for each time point in the simulation. The second dimension is frequency, and there is a value for each fundamental frequency, each harmonic, and each mixing term in the analysis, as well as the baseband term.
vOut[1] is the equivalent of vOut[::, 1], and specifies all time points at the lowest non-baseband frequency (the fundamental analysis frequency, unless a multitone analysis has been run and there are mixing products). For former MDS users, the notation "vOut[*, 2]" in MDS corresponds to the notation of "vOut[1]".

Defined in

$HPEESOF_DIR/expressions/ael/digital_wireless_fun.ael

See Also

const_evm() (expmeas)

Notes/Equations

Used in Constellation diagram generation.
The I and Q data do not need to be baseband waveforms. For example, they could be the in-phase (real or I) and quadrature-phase (imaginary or Q) part of a modulated carrier. The user must supply the I and Q waveforms versus time, as well as the symbol rate. A delay parameter is optional. The i_data and q_data must be of the same dimension, and up to 5-dimensional data is supported.

cross()

Computes the zero crossings of a signal, interval between successive zero crossings or slope at the crossing.

Syntax

yCross = cross(signal, direction, slope)

Arguments

Name	Description	Default	Range	Type	Required
signal	the signal for which the zero crossing is to be found	None	(-∞:∞)	Real	Yes
direction	type of zero crossing †	0	[-1:1]	Integer	Yes
slope	specifies if slope is to be calculated, rather than interval between zero crossing	0 (no slope)	[0:1]	Integer	No

† If direction = +1, compute positive going zero crossings.
If direction = -1, compute negative going zero crossings.
If direction = 0, compute all zero crossings.

Examples

```
period=cross(vosc-2.0, 1)
This computes the period of each cycle of the vosc signal. The period is
measured from each positive-going transition through 2.0V.
period = cross(vosc-2.0, 1, 1) returns the zero crossings and the slope at the
zero crossings.
```

Defined In

Built in

Notes/Equations

The independent axis returns the time when the crossing occurred. If the third argument is set to 1, the dependent axis returns the slope at zero crossing. Otherwise the dependent axis returns the time interval since the last crossing (default behavior).

fspot()

Performs a single-frequency time-to-frequency transform

Syntax

y = fspot(x, fund, harm, windowType, windowConst, interpOrder, tstart)

Arguments

Name	Description	Default	Range	Type	Required
x	time domain signal	None	(-∞:∞)	Real	Yes
fund	period 1/fund for the Fourier transform	period that matches the length of the independent axis of x	[1:∞)	Real	No
harm	harmonic number †	1	[1:∞)	Integer	No
windowType	type of window to apply to the data	0	[0:9] ††	Integer, String	No
windowConst	window constant †† †	1	[0:∞)	Real	No
interpOrder	interpolation scheme	None	[1:3] †† ††	Integer	No
tstart	start time	1	[0:∞)	Real	No

† harm=0 will compute the dc component of x.
†† The window types and their default constants are:
0 = None
1 = Hamming 0.54
2 = Hanning 0.50
3 = Gaussian 0.75
4 = Kaiser 7.865
5 = 8510 6.0
6 = Blackman
7 = Blackman-Harris
8 = 8510-Minimum 0
9 = 8510-Maximum 13
windowType can be specified either by the number or by the name.

†† † windowConst is not used if windowType is 8510

†† †† If the tranorder variable is not present, or if the user wishes to override the interpolation scheme, then interpOrder may be set to a nonzero value:
1 = use only linear interpolation
2 = use quadratic interpolation
3 = use cubic polynomial interpolation

Examples

```
The following example equations assume that a transient simulation was
performed from 0 to 5 ns on a 1-GHz-plus-harmonics signal called vOut:
fspot(vOut)
returns the 200-MHz component, integrated from 0 to 5 ns.
fspot(vOut, , 5)
returns the 1-GHz component, integrated from 0 to 5 ns.
fspot(vOut, 1GHz, 1)
returns the 1-GHz component, integrated from 4 to 5 ns.
fspot(vOut, 0.5GHz, 2, , , , 2.5ns)
returns the 1-GHz component, integrated from 2.5 to 4.5 ns.
fspot(vOut, 0.25GHz, 4, "Kaiser")
returns the 1-GHz component, integrated from 1 to 5 ns, after applying the
default Kaiser window to this range of data.
fspot(vOut, 0.25GHz, 4, 3, 2.0)
returns the 1-GHz component, integrated from 1 to 5 ns, after applying a
Gaussian window with a constant of 2.0 to this range of data.
```

Defined in

Built in

See Also

fft() (expmeas), *fs()* (expmeas)

Notes/Equations

fspot(x) returns the discrete Fourier transform of the vector x evaluated at one specific frequency. The value returned is the peak component, and it is complex. The harmth harmonic of the fundamental frequency fund is obtained from the vector x. The Fourier transform is applied from time tstop-1/fund to tstop, where tstop is the last timepoint in x.

When x is a multidimensional vector, the transform is evaluated for each vector in the specified dimension. For example, if x is a matrix, then fspot(x) applies the transform to every row of the matrix. If x is three dimensional, then fspot(x) is applied in the lowest dimension over the remaining two dimensions. The dimension over which to apply the transform may be specified by dim; the default is the lowest dimension (dim=1). x must be numeric. It will typically be data from a transient, signal processing, or envelope analysis.
By default, the transform is performed at the end of the data from tstop-1/fund to tstop. By using tstart, the transform can be started at some other point in the data. The transform will then be performed from tstart to tstart+1/fund.
Unlike with fft() or fs(), the data to be transformed are not zero padded or resampled. fspot() works directly on the data as specified, including non-uniformly sampled data from a transient simulation.

Transient simulation uses a variable timestep and variable order algorithm. The user sets an upper limit on the allowed timestep, but the simulator will control the timestep so the local truncation error of the integration is controlled. If the Gear integration algorithm is used, the order can also be changed during simulation. fspot() can use all of this information when performing the Fourier transform. The time data are not resampled; the Fourier integration is performed from timestep to timestep of the original data.

When the order varies, the Fourier integration will adjust the order of the polynomial it uses to compute the shape of the data between timepoints.
This variable order integration depends on the presence of a special dependent variable, tranorder, which is output by the transient simulator. If this variable is not present, or if the user wishes to override the interpolation scheme, then interpOrder may be set to a nonzero value.

Only polynomials of degree one to three are supported. The polynomial is fit because time domain data are obtained by integrating forward from zero; previous data are used to determine future data, but future data can never be used to modify past data.

ifc_tran()

Returns frequency-selective current in Transient analysis

Syntax

y = ifc_tran(iOut, fundFreq, harmNum)

Arguments

Name	Description	Default	Range	Type	Required
iOut	current through a branch	None	(-∞:∞)	Real, Complex	Yes
fundFreq	fundamental frequency	None	[0:∞)	Real	Yes
harmNum	harmonic number of the fundamental frequency	None	[0:∞)	Integer	Yes

Examples

```
y = ifc_tran(I_Probe1.i, 1GHz, 1)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

pfc_tran() (expmeas), *vfc_tran()* (expmeas)

Notes/Equations

This measurement gives RMS current, in current units, for a specified branch at a particular frequency of interest. *fundFreq* determines the portion of the time-domain waveform to be converted to the frequency domain. This is typically one full period corresponding to the lowest frequency in the waveform. *harmNum* is the harmonic number of the fundamental frequency at which the current is requested.

ispec_tran()

Returns current spectrum

Syntax

y = ispec_tran(iOut, fundFreq, numHarm, windowType, windowConst)

Arguments

Name	Description	Default	Range	Type	Required
iOut	current through a branch	None	(-∞:∞)	Real, Complex	Yes
fundFreq	fundamental frequency	None	[0:∞)	Real	Yes
numHarm	number of harmonics of fundamental frequency	None	[0:∞)	Integer	Yes
windowType	type of window to be applied to the data	0	[0:9] †	Integer, string	No
windowConst	window constant †† †	0	[0:∞)	Integer, Real	No

† The window types and their default constants are:
0 = None
1 = Hamming 0.54
2 = Hanning 0.50
3 = Gaussian 0.75
4 = Kaiser 7.865
5 = 8510 6.0 (This is equivalent to the time-to-frequency transformation with normalgate shape setting in the 8510 series network analyzer.)
6 = Blackman
7 = Blackman-Harris
8 = 8510-Minimum 0
9 = 8510-Maximum 13

Examples

```
y = ispec_tran(I_Probe1.i, 1GHz, 8)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

pspec_tran() (expmeas), *vspec_tran()* (expmeas)

Notes/Equations

This measurement gives a current spectrum for a specified branch. The measurement gives a set of RMS current values at each frequency. The *fundFreq* argument determines the portion of the time-domain waveform to be converted to frequency domain. This is typically one full period corresponding to the lowest frequency in the waveform. The *numHarm* argument is the number of harmonics of fundamental frequency to be included in the currents spectrum.

pfc_tran()

Returns frequency-selective power

Syntax

y = pfc_tran(vPlus, vMinus, iOut, fundFreq, harmNum)

Arguments

Name	Description	Default	Range	Type	Required
vPlus	voltage at the positive terminal	None	(-∞:∞)	Real, Complex	Yes
vMinus	voltage at the negative terminal	None	(-∞:∞)	Real, Complex	Yes
iOut	current through a branch measured for power calculation	None	(-∞:∞)	Real, Complex	Yes
fundFreq	fundamental frequency	None	[0:∞)	Real	Yes
harmNum	harmonic number of the fundamental frequency	None	[0:∞)	Integer	Yes

Examples

```
a = pfc_tran(v1, v2, I_Probe1.i, 1GHz, 1)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

ifc_tran() (expmeas), *vfc_tran()* (expmeas)

Notes/Equations

This measurement gives RMS power, delivered to any part of the circuit at a particular frequency of interest. *fundFreq* determines the portion of the time-domain waveform to be converted to frequency domain. This is typically one full period corresponding to the lowest frequency in the waveform. *harmNum* is the harmonic number of the fundamental frequency at which the power is requested.

pspec_tran()

Returns transient power spectrum

Syntax

y = pspec_tran(vPlus, vMinus, iOut, fundFreq, numHarm, windowType, windowConst)

Arguments

Name	Description	Default	Range	Type	Required
vPlus	voltage at the positive terminal	None	(-∞:∞)	Real, Complex	Yes
vMinus	voltage at the negative terminal	None	(-∞:∞)	Real, Complex	Yes
iOut	current through a branch measured for power calculation	None	(-∞:∞)	Real, Complex	Yes
fundFreq	fundamental frequency	None	[0:∞)	Real	Yes
numHarm	number of harmonics of fundamental frequency	None	[0:∞)	Integer	Yes
windowType	type of window to be applied to the data	0	[0:9] †	Integer, string	No
windowConst	window constant †† †	0	[0:∞)	Integer, Real	No

† The window types and their default constants are:
0 = None
1 = Hamming 0.54
2 = Hanning 0.50
3 = Gaussian 0.75
4 = Kaiser 7.865
5 = 8510 6.0 (This is equivalent to the time-to-frequency transformation with normalgate shape setting in the 8510 series network analyzer.)
6 = Blackman
7 = Blackman-Harris
8 = 8510-Minimum 0
9 = 8510-Maximum 13

Examples

```
a = pspec_tran(v1, v2, I_Probe1.i, 1GHz, 8)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

ispec_tran() (expmeas), *vspec_tran()* (expmeas)

Notes/Equations

This measurement gives a power spectrum, delivered to any part of the circuit. The measurement gives a set of RMS power values at each frequency. The *fundFreq* argument is the fundamental frequency that determines the portion of the time-domain waveform to be converted to frequency domain (typically one full period corresponding to the lowest frequency in the waveform). The *numHarm* argument is the number of harmonics of the fundamental frequency to be included in the power spectrum.

pt_tran()

This measurement produces a transient time-domain power waveform for specified nodes.

Syntax

y = pt_tran(vPlus, vMinus, current, fundFreq)

Arguments

Name	Description	Default	Range	Type	Required
vPlus	voltage at the positive terminal	None	(-∞:∞)	Real, Complex	Yes
vMinus	voltage at the negative terminal	None	(-∞:∞)	Real, Complex	Yes
current	current	None	(-∞:∞)	Real, Complex	Yes
fundFreq	fundamental frequency	None	[0:∞)	Real	Yes

Examples

```
a = pt_tran(v1, v2, i1, 1GHz)
```

Defined In

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

vt() (expmeas), *vt_tran()* (expmeas)

Notes/Equations

DC-to-RF efficiency is based on HB analysis.

vfc_tran()

Returns the transient frequency-selective voltage

Syntax

y = vfc_tran(vPlus, vMinus, fundFreq, harmNum)

Arguments

Name	Description	Default	Range	Type	Required
vPlus	voltage at the positive terminal	None	(-∞:∞)	Real, Complex	Yes
vMinus	voltage at the negative terminal	None	(-∞:∞)	Real, Complex	Yes
fundFreq	fundamental frequency	None	[0:∞)	Real	Yes
harmNum	harmonic number of the fundamental frequency	None	[0:∞)	Integer	Yes

Examples

```
a = vfc_tran(vOut, 0, 1GHz, 1)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

ifc_tran() (expmeas), *pfc_tran()* (expmeas)

Notes/Equations

This measurement gives the RMS voltage across any two nodes at a particular frequency of interest. The fundamental frequency determines the portion of the time-domain waveform to be converted to frequency domain. This is typically one full period corresponding to the lowest frequency in the waveform. The harmonic number is the fundamental frequency at which the voltage is requested (positive integer value only).

vspec_tran()

Returns the transient voltage spectrum

Syntax

y = vspec_tran(vPlus, vMinus, fundFreq, numHarm, windowType, windowConst)

Arguments

Name	Description	Default	Range	Type	Required
vPlus	voltage at the positive terminal	None	(-∞:∞)	Real, Complex	Yes
vMinus	voltage at the negative terminal	None	(-∞:∞)	Real, Complex	Yes
fundFreq	fundamental frequency	None	[0:∞)	Real	Yes
numHarm	number of harmonics of fundamental frequency	None	[0:∞)	Integer	Yes
windowType	type of window to be applied to the data	0	[0:9] †	Integer, string	No
windowConst	window constant	0	[0:∞)	Integer, Real	No

† The window types and their default constants are:
0 = None
1 = Hamming 0.54
2 = Hanning 0.50
3 = Gaussian 0.75
4 = Kaiser 7.865
5 = 8510 6.0 (This is equivalent to the time-to-frequency transformation with normalgate shape setting in the 8510 series network analyzer.)
6 = Blackman
7 = Blackman-Harris

8 = 8510-Minimum 0
9 = 8510-Maximum 13

Examples

```
a = vspec_tran(v1, v2, 1GHz, 8)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

ispec_tran() (expmeas), *pspec_tran()* (expmeas)

Notes/Equations

This measurement gives a voltage spectrum across any two nodes. The measurement gives a set of RMS voltages at each frequency. The fundamental frequency determines the portion of the time-domain waveform to be converted to the frequency domain. This is typically one full period corresponding to the lowest frequency in the waveform. The *numHarm* argument is the number of harmonics of the fundamental frequency to be included in the voltage spectrum.

vt_tran()

This measurement produces a transient time-domain voltage waveform for specified nodes. vPlus and vMinus are the nodes across which the voltage is measured.

Syntax

y = vt_tran(vPlus, vMinus)

Arguments

Name	Description	Default	Range	Type	Required
vPlus	voltage at the positive terminal	None	(-∞:∞)	Real, Complex	Yes
vMinus	voltage at the negative terminal	None	(-∞:∞)	Real, Complex	Yes

Examples

```
a = vt_tran(v1, v2)
```

Defined in

$HPEESOF_DIR/expressions/ael/circuit_fun.ael

See Also

vt() (expmeas)

Utility Functions for Measurement Expressions

This section describes the utility functions in detail.

- *amodelb_snp()* (expmeas)
- *design_name()* (expmeas)

amodelb_snp()

Returns the cartesian difference between Two S Parameter matrices

Syntax

modelGoal = amodelb_snp(measured..S, model..S)

Arguments

Name	Description	Default	Range	Type	Required
Target	the input S matrix that needs to be matched	None	(-∞:∞)	Real	Yes
Model	input S matrix to compare	None	(-∞:∞)	Real	Yes

Examples

```
modelGoal = abodelb_snp( measured..S, model..S)
```

design_name()

Returns design name given by TopDesignName in netlist.log

Syntax

design = design_name()

Arguments

Name	Description	Default	Range	Type	Required
None	None	None	None	None	No

Notes

The design_name() function returns the value of the TopDesignName parameter in the netlist.log file. It allows the design name to be saved in the dataset for documentation.

Duplicated Expression Names

Expressions used in ADS are of three types:

- *Simulator Expressions* (expsim) are used *during* a simulation; they are the expressions that are added to a VAR block on a schematic page.
- *Measurement Expressions* (expmeas) are used *after* a simulation, for post-processing of simulation results. There are two ways to use them:
  - As MeasEqn blocks on a schematic page; although these blocks are on the schematic, they execute only after the actual simulation has finished (except in the case of Optimization; then the expression can be evaluated repeatedly, once for each try). The MeasEqn result is also written to the dataset.
  - As EQN blocks on a DDS page; after the simulation has finished, these blocks can be used to process the data for display or further calculations.
- *AEL Expressions* (ael) are employed by users to develop custom functions. They can be used wherever AEL is used. AEL expressions are valid even on a DDS page.

In many cases, the same function name is used for different types of expressions, and the expressions can differ. For example, there is a step() Measurement Expression, a step() Simulator Expression, and a step() AEL expression. Although these three expressions operate similarly, the Simulator Expression assumes that the value passed in represents time, and cannot be a complex number; that is not the case with the other two expressions.

Because expressions with the same function name do not necessarily operate in the same way, it is important to know which expression type will actually be used in a given situation, and to be aware of any differences between that expression and others with the same name.

Which Type is Used?

Where an expression name is duplicated, ADS uses the expression type that is appropriate to the place in which the expression name appears:
Measurement expressions include all AEL expressions, plus some new functions that are usable only in a measurement context and not in general AEL code. That is, AEL Expression functions are a proper subset of Measurement Expression functions.

- If the expression name appears in a VAR block on a schematic page, ADS interprets it as a Simulator Expression. If no Simulator Expression is found by that name, an Error is shown.
- If the expression name appears in a MeasEqn block on a schematic page, ADS interprets it as a Measurement Expression. If no Measurement Expression is found by that name, but an AEL Expression is found, it is used, otherwise an Error is shown.
- If the expression name appears in an EQN block on a DDS page, ADS interprets it as a Measurement Expression. If no Measurement Expression is found by that name, but an AEL Expression is found, it is used, otherwise an Error is shown.
- If the expression name appears in an AEL file, ADS interprets it as an AEL Expression. If no AEL Expression is found by that name, an Error is shown.

Expression Types and Expression Names

This table lists the duplicated expression names. Follow the links to find information about

the individual expressions.

Measurement Expression	**AEL Expression**	**Simulator Expression**
abs() Measurement (expmeas)	*abs() Function* (ael)	*abs() Expression* (expsim)
acos() Measurement (expmeas)	*acos() Function* (ael)	*acos() Expression* (expsim)
acosh() Measurement (expmeas)	*acosh() Function* (ael)	*acosh() Expression* (expsim)
acot() Measurement (expmeas)	*acot() Function* (ael)	
acoth() Measurement (expmeas)	*acoth() Function* (ael)	
asin() Measurement (expmeas)	*asin() Function* (ael)	*asin() Expression* (expsim)
asinh() Measurement (expmeas)		*asinh() Expression* (expsim)
atan() Measurement (expmeas)	*atan() Function* (ael)	*atan() Expression* (expsim)
atan2() Measurement (expmeas)	*atan2() Function* (ael)	*atan2() Expression* (expsim)
atanh() Measurement (expmeas)	*atanh() Function* (ael)	*atanh() Expression* (expsim)
ceil() Measurement (expmeas)	*ceil() Function* (ael)	*ceil() Expression* (expsim)
chr() Measurement (expmeas)	*chr() Function* (ael)	
cint() Measurement (expmeas)	*cint() Function* (ael)	
cmplx() Measurement (expmeas)	*cmplx() Function* (ael)	
complex() Measurement (expmeas)		*complex() Expression* (expsim)
conj() Measurement (expmeas)	*conj() Function* (ael)	*conj() Expression* (expsim)
convBin() Measurement (expmeas)	*convBin() Function* (ael)	
convHex() Measurement (expmeas)	*convHex() Function* (ael)	
convOct() Measurement (expmeas)	*convOct() Function* (ael)	
cos() Measurement (expmeas)	*cos() Function* (ael)	*cos() Expression* (expsim)
cosh() Measurement (expmeas)	*cosh() Function* (ael)	*cosh() Expression* (expsim)
cot() Measurement (expmeas)	*cot() Function* (ael)	*cot() Expression* (expsim)
coth() Measurement (expmeas)	*coth() Function* (ael)	*coth() Expression* (expsim)
db() Measurement (expmeas)	*dB() Function* (ael)	*db() Expression* (expsim)
dbm() Measurement (expmeas)	*dBm() Function* (ael)	*dbm() Expression* (expsim)
dbmtow() Measurement (expmeas)		*dbmtow() Expression* (expsim)
deg() Measurement (expmeas)	*deg() Function* (ael)	*deg() Expression* (expsim)
exp() Measurement (expmeas)	*exp() Function* (ael)	*exp() Expression* (expsim)
fix() Measurement (expmeas)	*fix() Function* (ael)	
float() Measurement (expmeas)	*float() Function* (ael)	
floor() Measurement (expmeas)	*floor() Function* (ael)	*floor() Expression* (expsim)
fmod() Measurement (expmeas)		*fmod() Expression* (expsim)
hypot() Measurement (expmeas)		*hypot() Expression* (expsim)
im() Measurement (expmeas)	*im() Function* (ael)	
imag() Measurement (expmeas)	*imag() Function* (ael)	*imag() Expression* (expsim)
	index() Function (ael)	*index() Expression* (expsim)
int() Measurement (expmeas)	*int() Function* (ael)	*int() Expression* (expsim)
jn() Measurement (expmeas)		*jn() Expression* (expsim)
	list() Function (ael)	*list() Expression* (expsim)
ln() Measurement (expmeas)	*ln() Function* (ael)	*ln() Expression* (expsim)
log() Measurement (expmeas)	*log() Function* (ael)	*log() Expression* (expsim)
log10() Measurement (expmeas)	*log10() Function* (ael)	*log10() Expression* (expsim)
mag() Measurement (expmeas)	*mag() Function* (ael)	*mag() Expression* (expsim)

max2() Measurement (expmeas)	*max2() Function* (ael)	
max() Measurement (expmeas)		*max() Expression* (expsim)
min() Measurement (expmeas)		*min() Expression* (expsim)
min2() Measurement (expmeas)	*min2() Function* (ael)	
num() Measurement (expmeas)	*num() Function* (ael)	
phase() Measurement (expmeas)	*phase() Function* (ael)	*phase() Expression* (expsim)
phasedeg() Measurement (expmeas)	*phasedeg() Function* (ael)	*phasedeg() Expression* (expsim)
phaserad() Measurement (expmeas)	*phaserad() Function* (ael)	*phaserad() Expression* (expsim)
polar() Measurement (expmeas)	*polar() Function* (ael)	*polar() Expression* (expsim)
pow() Measurement (expmeas)	*pow() Function* (ael)	*pow() Expression* (expsim)
rad() Measurement (expmeas)	*rad() Function* (ael)	*rad() Expression* (expsim)
re() Measurement (expmeas)	*re() Function* (ael)	
real() Measurement (expmeas)	*real() Function* (ael)	*real() Expression* (expsim)
ripple() Measurement (expmeas)		*ripple() Expression* (expsim)
round() Measurement (expmeas)	*round() Function* (ael)	
sgn() Measurement (expmeas)	*sgn() Function* (ael)	*sgn() Expression* (expsim)
sin() Measurement (expmeas)	*sin() Function* (ael)	*sin() Expression* (expsim)
sinc() Measurement (expmeas)	*sinc() Function* (ael)	*sinc() Expression* (expsim)
sinh() Measurement (expmeas)	*sinh() Function* (ael)	*sinh() Expression* (expsim)
	sprintf() Function (ael)	*sprintf() Expression* (expsim)
sqrt() Measurement (expmeas)	*sqrt() Function* (ael)	*sqrt() Expression* (expsim)
step() Measurement (expmeas)	*step() Function* (ael)	*step() Expression* (expsim)
	strcat() Function (ael)	*strcat() Expression* (expsim)
sum() Measurement (expmeas)		*sum() Expression* (expsim)
tan() Measurement (expmeas)	*tan() Function* (ael)	*tan() Expression* (expsim)
tanh() Measurement (expmeas)	*tanh() Function* (ael)	*tanh() Expression* (expsim)
wtodbm() Measurement (expmeas)		*wtodbm() Expression* (expsim)
xor() Measurement (expmeas)	*xor() Function* (ael)	