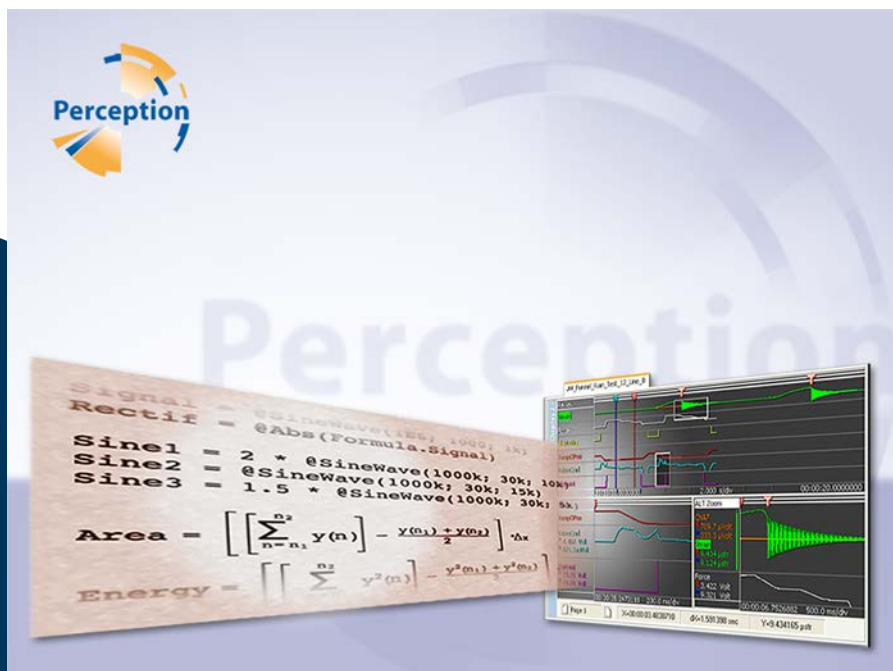


English

User Manual



Perception Analysis Option

Document version 10.0 - February 2025

For Perception 7.14 or higher

For HBM's Terms and Conditions visit www.hbm.com/terms

Hottinger Brüel & Kjaer GmbH
Im Tiefen See 45
64293 Darmstadt
Germany
Tel: +49 6151 80 30
Fax: +49 6151 8039100
Email: info@hbm.com
www.hbm.com/highspeed

Copyright © 2025

All rights reserved. No part of the contents of this document may be reproduced or transmitted in any form or by any means without the written permission of the publisher.

LICENSE AGREEMENT AND WARRANTY

LICENSE AGREEMENT and WARRANTY

For information about LICENSE AGREEMENT AND WARRANTY refer to:
www.hbm.com/terms

TABLE OF CONTENTS

ToC - Overview

1	Analysis Option	10
1.1	Introduction	10
1.1.1	How to install the Analysis option	10
1.2	Formula database sheet	12
1.3	Definitions	15
1.3.1	Constants	15
1.3.2	Variables	15
1.3.3	Functions	16
1.4	Modifying the layout	17
1.4.1	Add, delete and clear rows	17
1.4.2	Moving around	18
1.5	Creating formulas	19
1.5.1	Function creator	19
1.5.2	Entering comment	25
1.5.3	Entering formulas	25
1.5.4	Problems in formulas	26
1.6	Formula menu	30
1.6.1	Load Formulas	30
1.6.2	Save Formulas	31
1.6.3	Print Formulas	31
1.6.4	Move Sheet	32
2	Formula Database Functions	33
2.1	General	33
2.2	Overview	35
2.2.1	Function Overview	35
2.2.2	Function Overview – HIC section	42
2.2.3	Function Overview – Cycle Math section	43
3	Arithmetic Operations	45
3.1	+ (Addition)	45
3.2	- (Subtraction)	47
3.3	* (Multiplication)	49
3.4	/ (Division)	51
3.5	– (Unary minus)	53
3.6	@Modulo	54
4	Reference Guide	55

TABLE OF CONTENTS

4.1	@ABS	55
4.2	@ACosine	56
4.3	@And	57
4.4	@Area	59
4.5	@ASine	61
4.6	@ATan	62
4.7	@ATan2	63
4.8	@BlockFFT	66
4.9	@Clip	68
4.10	@Comparator	70
4.11	@Cos	72
4.12	@CurveFitting	73
4.13	@Cut	75
4.14	@Cycles	77
4.15	@Diff	79
4.16	@DQ0Transformation	81
4.17	@Energy	85
4.18	@EqualTo	87
4.19	@Exp	88
4.20	@ExpWave	89
4.21	@FallTime	91
4.22	@FilterBiquad	93
4.23	@FilterButterworthLP	96
4.24	@FilterButterworthHP	98
4.25	@FilterButterworthBP	100
4.26	@FilterButterworthBS	102
4.27	@FilterBesselLP	104
4.28	@FilterBesselHP	106
4.29	@FilterBesselBP	108
4.30	@FilterBesselBS	110
4.31	@FilterChebyshevLP	112
4.32	@FilterChebyshevHP	115
4.33	@FilterChebyshevBP	117
4.34	@FilterChebyshevBS	119
4.35	@Frequency	121
4.36	@FormatDate	123
4.37	@FormatTime	125
4.38	@GetNumberFromString	127

TABLE OF CONTENTS

4.39	@GreaterEqualThan	128
4.40	@GreaterThan	129
4.41	@Histogram	130
4.42	@HSFundamentalFrequency	132
4.43	@HSOneHarmonicAmplitude	133
4.44	@HSOneHarmonicPhase	138
4.45	@IIF	140
4.46	@Integrate	142
4.47	@IntLookUp	143
4.48	@IntLookUp12	146
4.49	@IsNaN	148
4.50	@Join	149
4.51	@Length	151
4.52	@LessEqualThan	152
4.53	@LessThan	153
4.54	@Ln	154
4.55	@Log	156
4.56	@Max	157
4.57	@MaxNum	159
4.58	@MaxPos	160
4.59	@Mean	162
4.60	@MedianFilter	164
4.61	@Min	167
4.62	@MinNum	169
4.63	@MinPos	170
4.64	@NextHillPos	172
4.65	@NextLvlCross	174
4.66	@NextValleyPos	176
4.67	@Noise	178
4.68	@Not	179
4.69	@NumberOfSweeps	180
4.70	@Or	182
4.71	@Period	184
4.72	@Pow	186
4.73	@PrevHillPos	187
4.74	@PrevLvlCross	189
4.75	@PrevValleyPos	191
4.76	@Pulse	193

TABLE OF CONTENTS

4.77	@PulseWidth	195
4.78	@Ramp	197
4.79	@ReadAsciiFile	199
4.80	@ReadLogFile	201
4.81	@Reduce	205
4.82	@RefCheck	206
4.83	@RelativeTime2Local	208
4.84	@RelativeTime2UTC	210
4.85	@RemoveGlitch	212
4.86	@Res2	213
4.87	@RiseTime	214
4.88	@RMS	216
4.89	@SAEJ211Filter	218
4.90	@Sin	219
4.91	@SineWave	220
4.92	@Smooth	222
4.93	@SpaceVectorInverseTransformation	224
4.94	@SpaceVectorTransformation	228
4.95	@Sqrt	231
4.96	@SquareWave	232
4.97	@StdDev	233
4.98	@Sweep	235
4.99	@SweepEndTime	236
4.100	@SweepNumberAtTime	237
4.101	@SweepSource	238
4.102	@SweepStartTime	239
4.103	@SweptSineWave	240
4.104	@Tan	242
4.105	@TimeMaxAbove	243
4.106	@TimeMaxAboveBegin	245
4.107	@TimeMaxBelow	247
4.108	@TimeMaxBelowBegin	249
4.109	@TimeMinAbove	251
4.110	@TimeMinAboveBegin	253
4.111	@TimeMinBelow	255
4.112	@TimeMinBelowBegin	257
4.113	@TimeTotalAbove	259
4.114	@TimeTotalAboveBegin	261

TABLE OF CONTENTS

4.115	@TimeTotalBelow	263
4.116	@TimeTotalBelowBegin	265
4.117	@TriggerTime	267
4.118	@TriggerTimeToText	269
4.119	@TrueFrequency	271
4.120	@TrueRMS	273
4.121	@TrueRMSRef	274
4.122	@Value	276
4.123	@XDelta	277
4.124	@XDeltaHigh	278
4.125	@XDeltaLow	279
4.126	@XFirst	280
4.127	@XLast	281
4.128	@XShift	282
4.129	@XYArray	283
4.130	@YArray	285
5	Reference Guide - HIC	286
5.1	Introduction - HIC	286
5.2	@Con3ms	287
5.3	@Con3ms_T	289
5.4	@Cum3ms	290
5.5	@Cum3ms_T	292
5.6	@HIC	293
5.7	@HICStartTime	295
5.8	@HICEndTime	296
5.9	@HIC15	297
5.10	@HIC15StartTime	298
5.11	@HIC15EndTime	299
5.12	@HIC36	300
5.13	@HIC36StartTime	301
5.14	@HIC36EndTime	302
6	Reference Guide - Cycle Math	303
6.1	Introduction - Cycle Math	303
6.2	@CycleArea	307
6.3	@CycleCount	309
6.4	@CycleCrestFactor	310
6.5	@CycleDetect	312

TABLE OF CONTENTS

6.6	@CycleEnergy	314
6.7	@CycleFrequency	316
6.8	@CycleFundamental	318
6.9	@CycleFundamentalPhase	320
6.10	@CycleFundamentalRMS	322
6.11	@CycleInterval	324
6.12	@CycleLevel	326
6.13	@CycleMax	328
6.14	@CycleMean	329
6.15	@CycleMin.	331
6.16	@CyclePeriod	332
6.17	@CyclePhase.....	333
6.18	@CycleRPM.....	335
6.19	@CycleRMS	336
6.20	@CycleStdDev.....	338
6.21	@CycleTHD	340
7	Pulse Measurement and Analysis	342
7.1	General	342
8	IIR Filters.....	346
8.1	Introduction.....	346
8.1.1	Bessel	347
8.1.2	Butterworth.....	348
8.1.3	Chebyshev (Type I).....	348
8.1.4	Magnitude Spectrum.....	349
8.1.5	Impulse Response	352
8.1.6	Step Response	353
8.1.7	Phaseless Filtering.....	354
8.1.8	Importance of sampling rate and cutoff frequency.....	355

1 Analysis Option

1.1 Introduction

The Perception Analysis option allows you to perform calculations on measured data. A variety of built-in functions get you on the right track for analysis, ranging from basic statistics to advanced mathematics.

The analysis option comprises two main parts:

- A variety of functions
- The visible formula database

Using the formula database you can create your own set of additional functions without programming or sequencing. Just type in the required calculation and the result will be displayed. The results of new data is updated automatically, not only in the formula database but also directly on the displays and within the reports. Once defined the formulas can be saved for later use.

The formula database allows for an unlimited number of formulas, each with a name and units. A formula can be created using arithmetic operations on waveforms and scalars and combined with one of the built-in functions, cursor information, or the result of another formula. Auto-complete and in-line help guide you through the various options.

For ease of use we refer to the analysis option including the functions and the formula database as "the formula database".

1.1.1 How to install the Analysis option

The Perception software requires a HASP key. HASP (Hardware Against Software Piracy) is a hardware-based (hardware key) software copy protection system that prevents unauthorized use of software applications.

Each HASP key contains a unique ID number used for personalization of the application according to the features and options purchased. The key is also used for storing licensing parameters, applications and customer-specific data. If you have purchased the Analysis option as a separate item, you will receive a personalized "key file". Use this file to unlock the additional features.

You can find the serial number of your key in **Help ► About Perception**

To update the key information:

1. Choose **Help ► Update Key...**

1 ANALYSIS OPTION

2. In the Open dialog locate the Key File (*.pKey) and click **Open**.
3. If everything is OK you will see the following message:



Fig. 1.1 Software copy protection dialog

4. Click **OK**.

After the installation you can go to **Help ► About Perception ► More...** to see all installed options.

You will need to restart the program before the changes take effect. The Analysis option is now available.

1.2 Formula database sheet

The Formula database sheet is used to create and edit formulas.

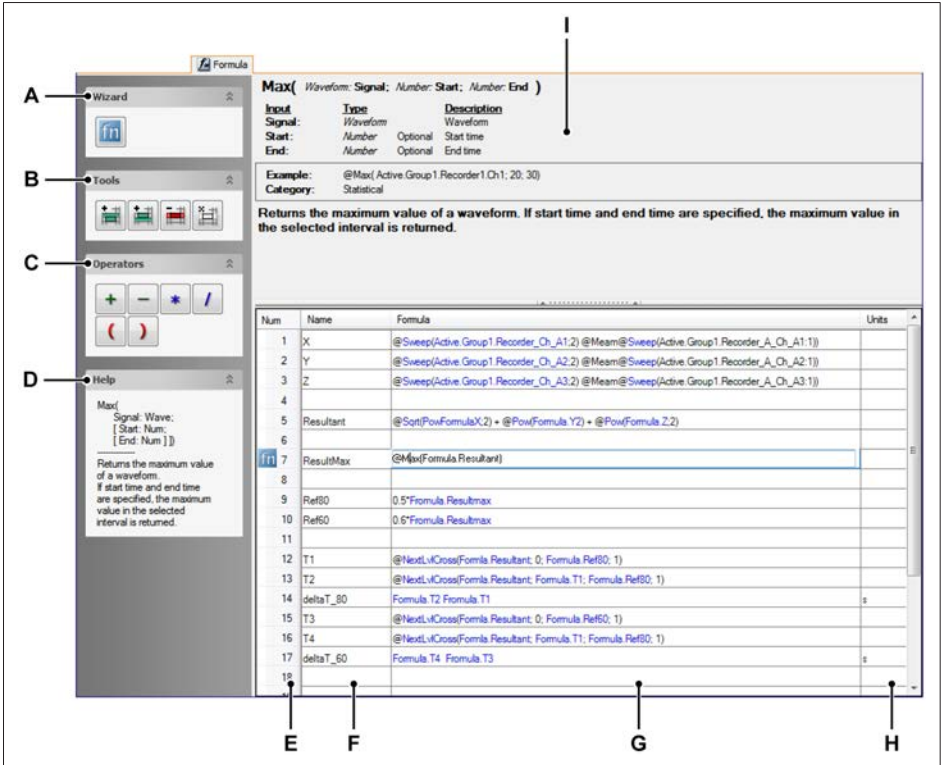


Fig. 1.2 Formula database sheet

- A Wizard
- B Tools
- C Operators
- D Default help
- E Formula database row numbers with row pointer (function creator icon)
- F Formula name column
- G Formula column
- H Formula units
- I Additional help

- A **Wizard** The function creator allows the user to enter and verify a function in a “guided” way. The function creator operates on the currently selected row. The currently selected row is denoted by the function creator icon in the row number column.



Fig. 1.3 Function creator icon

- 1 Calls on the function creator (see Fig. 1.7)
- B **Tools** Tools are provided to add, delete and clear lines with formulas. The tools operate on the currently selected row. The currently selected row is denoted by the function creator icon in the row number column.

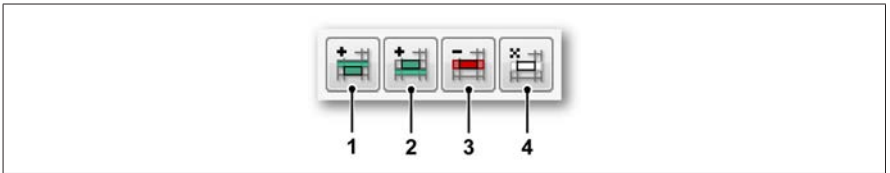


Fig. 1.4 Formula sheet tools

- 1 Insert row above current row
- 2 Insert row below current row
- 3 Delete row
- 4 Clear row

1 ANALYSIS OPTION

C **Operators** Use the operator buttons to insert basic operators.

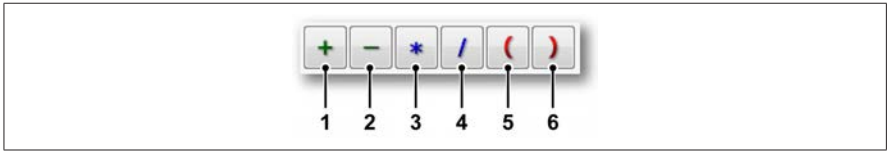


Fig. 1.5 Formula sheet operators

- 1 Add
 - 2 Subtract
 - 3 Multiply
 - 4 Divide
 - 5 Opening parenthesis
 - 6 Closing parenthesis
- D **Default help** This area displays basic help information on the selected function.
- E **Row numbers** For easy reference each row has a row number. The currently active row has a function creator icon in front of the number.
- F - H **Formulas** Each formula has a name, body and units. This area is also called the “formula editor”.
- I **Additional help** For more detailed help information the additional help area is optionally available.

You may want to show or hide the additional help.

To show or hide the additional help:

- Click on the grip that is located on top of the formulas area.



1.3 Definitions

The formula database uses operators and functions. Operators require variables, functions require parameters.

1.3.1 Constants

Constants are numerical or string values which are entered straight into the formula. Numbers can be entered using floating point or integer notation. Internally all numerical values are floating point numbers. Strings are entered with quotation marks, like "text".

1.3.2 Variables

Variables can be any variable (waveform, numeric or string) from the data source list. A variable can be selected using the data sources. Previously defined formulas are also available as variables.

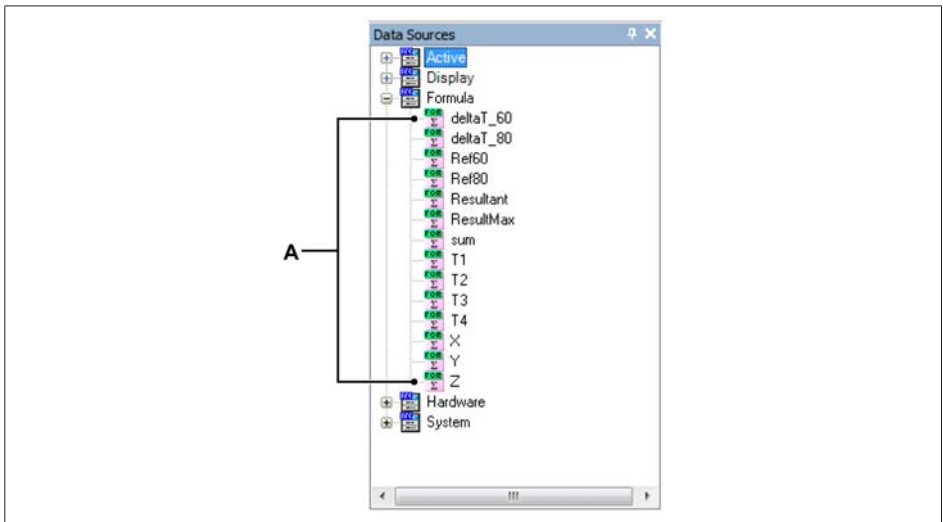


Fig. 1.6 Data Sources navigator with formula results

A Formula results

In the formula body a variable is referenced by the complete path of the variable in the data sources. For example in the diagram above the formula result T2 is referenced as **Formula.T2**

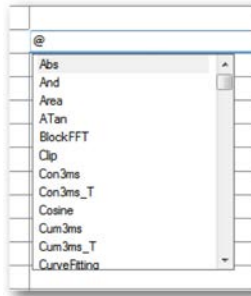
1 ANALYSIS OPTION

1.3.3 Functions

A function takes a number of parameters (which can be variables and/or constants) and uses them to produce a result. The type of the parameters and the type of the results depends on the function.

To enter a function:

1. Click once in the formula body of an empty row to select it or click twice in the formula body of an empty row to enter the edit mode.
2. Type the @ sign.
 - A list of available functions comes up.



3. To select a function from the list do the following:
 - Use the scroll bar to scroll through the list of available functions.
 - Click with the mouse on the function.
 - Use the arrow keys to select a function and press Enter.

The complete description of the selected function comes up in the Help area.

Notice

As long as the formula has no name it is considered to be a comment line.

1.4 Modifying the layout

You can modify the layout of the rows by adding, deleting and clearing rows. You can use the tools directly or a right mouse click to call up a context menu with the same commands.

1.4.1 Add, delete and clear rows



You can add, delete and clear rows. You can do this on one row or on multiple rows.

To select multiple rows:

To select consecutive rows:



- Click the first row and drag to the last row.
- Click the first row, press and hold down SHIFT, and then click the last row.
- To select nonconsecutive rows, press and hold down CTRL, and then click each row.

To add one or more rows:

1. Select one or more rows.
 - The row pointer will move to the last selected row.
2. Click on the appropriate tool:
 - **Insert row(s) above**  to insert row(s) above the currently selected row.
 - **Insert row(s) below**  to insert row(s) below the currently selected row.
 - Empty row(s) will be inserted.

To remove one or more rows:

Select one or more rows and do the following:

- To remove a row and the contents click the **Delete selected row(s)**  tool.
- To remove the contents of the row but keep the row click the **Clear selected row(s)**  tool.
- In the confirmation dialog that comes up click **OK**.

To clear a field:

You can clear a single field as follows:

- Select the field you want to clear and press **Del**.
- Select the field you want to clear and press **Enter**: the field is open for editing and the complete text is selected. Press **Del**.

1.4.2 Moving around

You can move through the fields by using the **Tab** and the **Arrow** keys.

1.5 Creating formulas

Creating formulas and comment is easy. Using the function creator or various auto-completion techniques you are guided through the definition of formulas without too much typing. This system also reduces the possibility of errors.

The formula editor has a database alike layout with rows that represent records. Each row has three editable fields to enter comment and formulas.

1.5.1 Function creator

The function creator is a one-dialog feature that allows the user to enter and verify a function in a “guided” way.


To call on the function creator:

Select the row where you want to create or edit a function. There are two ways to open the function creator:

- Click on the function creator icon in the number column of the row.



OR

- Click on the function creator icon  in the Wizard panel (see Fig. 1.2 on page 12).

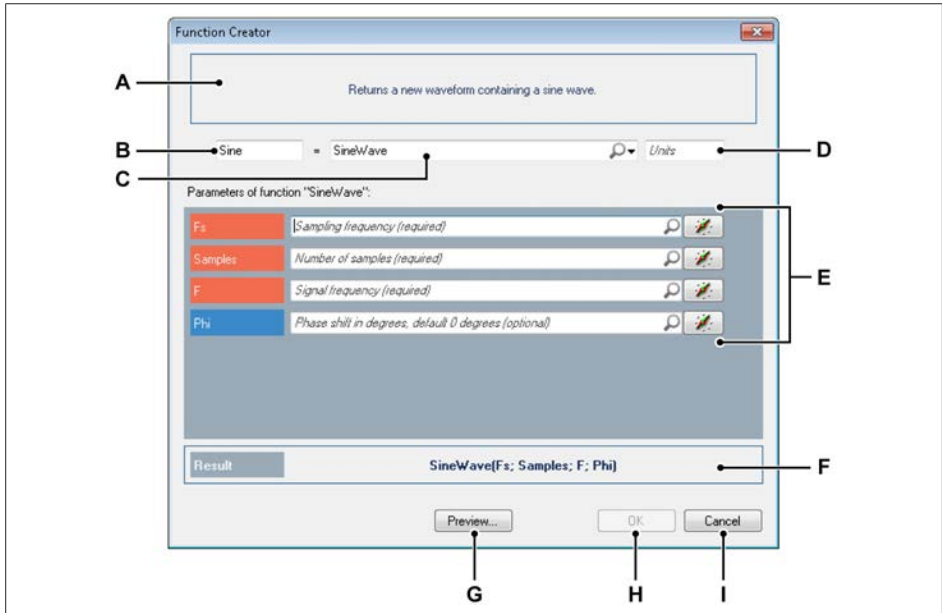


Fig. 1.7 Function Creator dialog

- A Default help
- B Function name
- C Function selection/search control
- D Function units
- E Function parameters
- F Function result
- G Show/hide preview
- H OK button
- I Cancel button

- A **Default help** This area displays basic help information on the selected function.
- B **Function name** The name of the function as it is shown in the data sources navigator.

- C Function selection/search control** Type in a part of the function's name you are looking for. The functions containing the search query are listed.

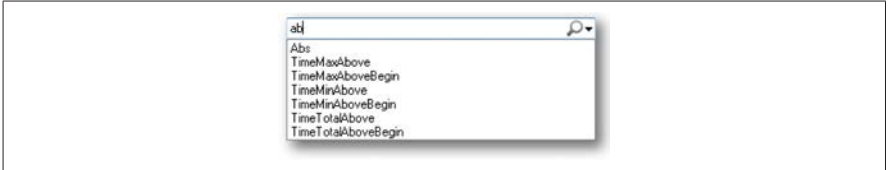
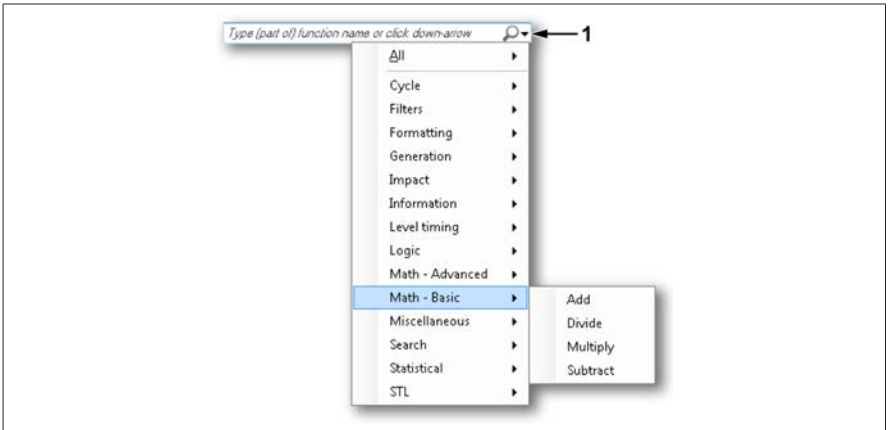


Fig. 1.8 Function selection – Automatically completed list

The needed function can be selected with the mouse or the up- and down-arrow keys and confirmed with enter.

If the drop-down arrow of the search control is pressed, the available functions are shown per group.



1 Drop-down arrow

- D Function units** The units of the function.
If the function has created its own unit (for example by performing an “Abs” on a waveform with the unit “Hz”), this is shown in the entry field as a default.

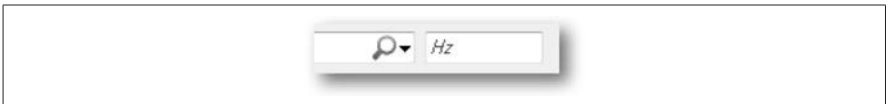


Fig. 1.9 Default function unit

- E **Function parameters** If a function is selected, the parameters for this method are added here.

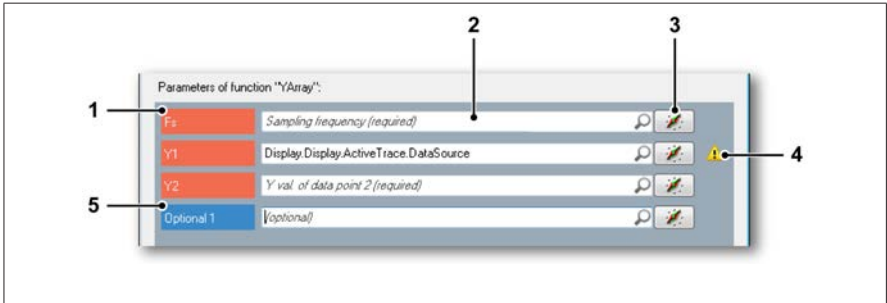




Fig. 1.10 Function parameters area

- 1 Required function parameter
A parameter for the function which is required. This is indicated by the red color and the text "required".
- 2 Function parameter value
Set the needed value for this parameter. Valid data sources containing the entered text will also be shown in the drop down list.
- 3 Data source button 
Opens the data source selection dialog, in which the valid data sources are shown. Calls on the default data source selector, which only shows the data sources which are allowed.
- 4 Invalid parameter indication 
If the value of the parameter is invalid, this is shown by a warning icon behind the parameter.
- 5 Optional function parameter
A parameter for the function which is optional. This is indicated by the blue color and the text "optional".
Some functions allow you to add optional parameters. If you enter a value in the last optional parameter, a new optional parameter is added, and the previous parameter becomes required (see Fig. 1.11).

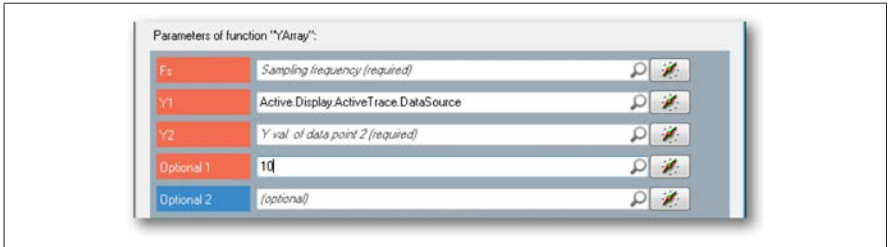


Fig. 1.11 Function parameters area with new optional parameter

F **Function result** Shows the result of the function created by the function creator.

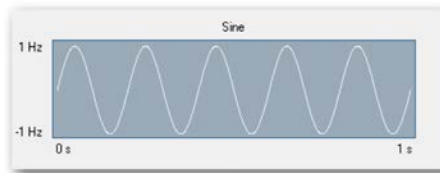
1 ANALYSIS OPTION

G **Show/hide preview** Expands or collapses the dialog to show or hide the preview. There are three kinds of previews:

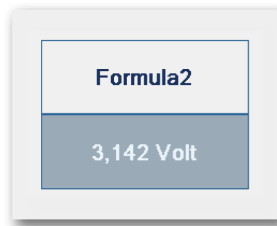
1 "Invalid" preview:



2 "Waveform" preview:



3 "Value" preview:



H Click the **OK** button to apply the current function and to close the dialog.

Notice

*The **OK** button is only enabled when the function is set up correctly. All required parameters and the function name must be filled in.*

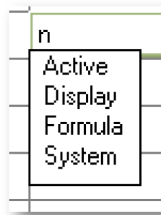
I Click the **Cancel** button to discard the changes and to close the dialog.

1.5.2 Entering comment

You can enter comment in the formula editor to clarify the formulas you make. Comment is displayed in a green color. A comment row has no name.

To enter comment:

1. Click once in the formula body of an empty row to select it or click twice in the formula body of an empty row to enter the edit mode.
2. Start typing your text. Note that after typing the first character a list comes up:



3. Ignore this list. It will disappear when you entered your first word. Continue typing until done.
4. Press **Enter** when done or click anywhere else in the formula editor.

Do not type text in the name column of the comment row.

1.5.3 Entering formulas

Formulas are rules how to create new waveforms, numeric values or strings from existing data sources, variables and constants. A formula is a mathematical expression which can contain data sources, constants and functions. You can use brackets to change the precedence of the operators.

Formulas have a name, a body and optionally units. Formulas are displayed in blue (known keywords) and black.

To enter a formula:

1. Enter a name for the formula:
 - a. Click once in the name field of an empty row to select it or click twice in the name field of an empty row to enter the edit mode.
 - b. Enter a descriptive name.

3. Navigate to the body field of the formula as follows:
 - Press the TAB.
 - Press the right-arrow key.
4. Enter the body of the formula by using one or more of the following techniques:
 - Type the @ sign to get a list of available functions and pick your function.
 - Type any character: a drop down list comes up with a list of possible data sources roots. Select a root and type a dot: a new list drops down with branches from the selected root. Select a branch and type a dot, etc. until you have selected the variable or data source you want.
 - Type in the full path of the variable / data source directly.
 - Use operators and brackets to create more complex formulas.
 - When available, help text is displayed in the Help area(s).
5. When done navigate to the Unit field and enter the units if applicable.



Tip

You can also drag data sources directly into a formula in the formula database. This allows you to quickly insert constants and variables into a function without the need to know the complete path of that variable. E.g simply drag a cursor X-position into your formula without typing the complete path like: Display.Display1.Cursor1.XPosition.

1.5.4 Problems in formulas

A lot of detailed input needs to be corrected when creating formulas in order for the formulas to work well. Typing errors or using incompatible types of input can cause problems in formulas. As the system of formulas grows, it becomes more difficult to pin-point the cause of problems. To help find these problems in the formulas, the formula database automatically finds and marks most of the problems.

Problems found in the formulas are divided into in two categories:



Errors: These are problems that need to be solved actively.



Warnings: These are problems in the fomulas that could possibly be fixed at a later time automatically. For example, if input signals from a recording were missing, the problem would be solved simply by loading the recording into the system later.

Errors found in the functions are divided into two categories. The difference between the two types of errors is the timepoint when the error can be detected.

- **Syntax errors:** These errors can be detected when new formulas are added to the formula database. A common example of a syntax error is when incorrect separators are used between the parameters of a formula.

Notice

Typing errors can be avoided by using the function wizard.

- **Runtime errors:** These errors are detected when the result of a formula is used. This happens when the result of the formula is used on a display, on a meter, in a report, or by another formula in use. Runtime errors are typically more difficult to detect than syntax errors are, since more knowledge of the measurement, data-sources and formulas used is required. It can be difficult to determine if a formula or system of formulas contain an error, especially in a complex measurement environment.

Consider the following examples:

- **Syntax error:** Incorrect separator

Num	Name	Formula
A → 1	SyntaxError	@SineWave(42.1k??100)

Fig. 1.12 Syntax error (Incorrect separator)

A Error message

SineWave normally uses three (3) arguments that are separated by a semi-colon. To illustrate this issue, the second semi-colon is mistyped as “??” in this example. The formula is now marked with an error icon. Additional information about the error can be retrieved by clicking on the icon. This will show the following detailed information for this example:

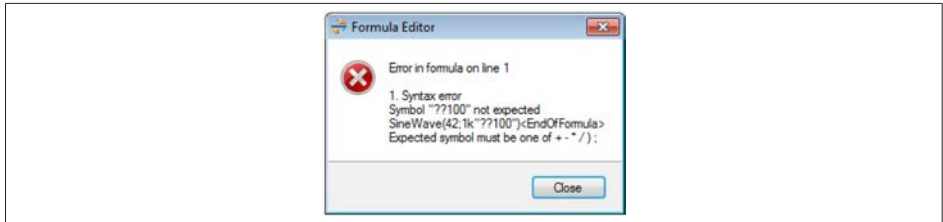


Fig. 1.13 Formula Editor – Error message

- **Runtime error:** Incompatible sample rates

Runtime errors are only detected when a formula is used. Therefore, two erroneous formulas could be added, but only one of the erroneous formulas might be marked with an error. In the following example, two functions are created to add two waveforms. The error here is that the two waveforms are sampled at different speeds. Thus, the samples are not aligned and cannot be added. “Add1” (A) is used because it is shown on screen, whereas “Add2” (B) is defined but it has not used anywhere in Perception yet.

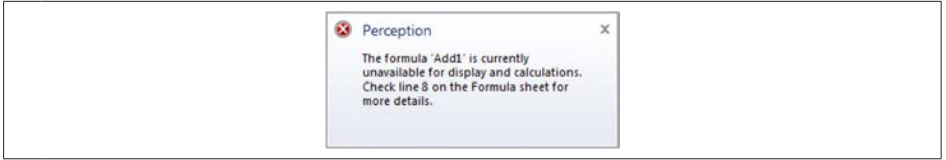
	6	Sine1Hz	@SineWave[1; 1000; 42]
	7	Sine3Hz	@SineWave[3; 1000; 42]
A	8	Add1	Formula.Sine1Hz + Formula.Sine3Hz
B	9	Add2	Formula.Sine1Hz + Formula.Sine3Hz

Fig. 1.14 Runtime error message

Even though both formulas are exactly the same, only the formula in use is marked with an error.

This problem is detected when the formula is used for the first time. It is very likely that the formula sheet is not visible at that time. If there is an error, no output is generated by the formula. This could be confusing. Therefore, Perception displays a notification in this case to provide as much feedback as possible.

1 ANALYSIS OPTION



This should enable you to find most common problems in the formulas.

Notice

Not all problems in the formulas can be detected automatically. Perception can and will not show if a wrong signal was used as input for a formula when the incorrect signal is of the right type.

1.6 Formula menu

The Formula menu lists commands related to Formula file handling. For layout and content management use the tools that are provided in the task pane on the left hand side of the formula editor.

The Formula menu is a dynamic menu and only available when the Formula sheet is on top, that is visible.

In the menu there are possibilities to save the formulas in a separate file. In general the formula database settings:

- comprises all formulas/functions as specified in the formulas sheet,
- can be stored in a separate file with the file extension **.pFormulas**,
- are stored automatically when a workbench is saved and as part of a recording,
- are loaded automatically as part of a complete workbench,
- can be extracted / loaded out of a workbench or recording as separate settings,
- can be saved into a workbench or recording as separate settings.

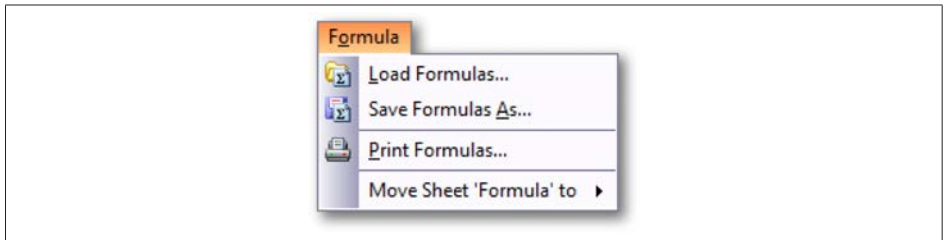


Fig. 1.15 Formula menu

1.6.1 Load Formulas

You can load formulas from a variety of sources.

To load formulas:

To load formulas from an external source proceed as follows:

1. Do one of the following:
 - In the **Formula menu** click **Load Formulas...**
 - When available in the **toolbar** click e **Load Formulas...** button 


2. In the Load Formulas dialog that comes up select your file type if required:
 - Formula File (*.pFormulas)
 - Formulas out of a Virtual Workbench (*.pVWB)
 - Formulas out of an Experiment (*.pNRF)
3. Select the file you want to use.
4. Click **Open**.

1.6.2 Save Formulas

Much in the same way as you can load formulas you can also save formulas. You can also save into an existing virtual workbench or experiment. By doing so you will replace the formulas within that file. No other data will be altered.

To save formulas:


To save formulas into an external file proceed as follows:

1. Do one of the following:
 - In the **Formulas** menu click **Save Formulas As...**
 - When available in the **toolbar** click the **Save Formulas As...** button 
2. In the **Save Formulas As** dialog that comes up select your file type if required:
 - Formula File (*.pFormulas)
 - Formulas out of a Virtual Workbench (*.pVWB)
 - Formulas out of an Experiment (*.pNRF)
3. Select the file you want to save into/replace or type a name for a new file.
4. Click **Save**.

1.6.3 Print Formulas

You can make a copy on the printer of the formulas.

To print a copy of the formulas:

1. Do one of the following:
 - In the **Formulas** menu click **Print Formulas...**
 - When available in the **toolbar** click the **Print Formulas...** button 
2. In the Print dialog that comes up enter your preferences.
3. Click **Print**.

1.6.4 Move Sheet

When the multiple workbook option is installed you can move the Formula sheet to another workbook.

To move the formula sheet to another workbook:

1. In the **Formulas** menu point to **Move Sheet 'Formula' to >**
2. In the sub menu that comes up select a workbook.

2 **Formula Database Functions**

2.1 **General**

This document describes in full detail all functions available in the Perception formula database.

For a description of the formula database itself refer to the appropriate section in the User Manual.

Details on pulse characteristics, measurement and analysis can be found in chapter: “Pulse Measurement and Analysis” on page 324.

Formulas in the formula database are defined as:

output = formula		
-------------------------	--	--

The output is available in the Data Sources navigator as a variable. You can find it as an item in the Formula branch.

The output can be used as parameter in another function, regardless of the physical order in the database.

You can enter formulas just as you would record on paper. Standard mathematical rules are applicable.

Exception

No expressions are allowed as parameter(s) of an “@”-function.

Example:

	Angle	=	33
Correct	AngleRad	=	System.Constants.Pi * Formula.Angle / 180
	CosAngle	=	@Cos (Formula.AngleRad)
Incorrect	CosAngle	=	@Cos (System.Constants.Pi * Formula.Angle / 180)

2 FORMULA DATABASE FUNCTIONS

Notice

All the functions work on static, single-time base, single-sweep data. The functions also work on multi-time base and/or multi-sweep data. However, the results may be unpredictable due to the nature of the function used. Some functions also work on dynamic (real-time) data. When applicable this is noted in the description of the function.

2 FORMULA DATABASE FUNCTIONS

2.2 Overview

This section gives an alphabetical overview of all functions, including their name, a short description of the function and the page number.

2.2.1 Function Overview

Function Overview	
Name	Description
+ (Add)	Add two expressions: "+ (Addition)" on page 45
- (Subtract)	Subtract two expressions: "- (Subtraction)" on page 47
* (Multiply)	Multiply two expressions: "* (Multiplication)" on page 49
/ (Divide)	Divide two expressions: "/" (Division)" on page 50
– (Invert)	Invert an expression: "– (Unary minus)" on page 53
@Modulo	Calculate the modulo: "@Modulo" on page 54
@Abs	Absolute value of a numerical value or a waveform: "@ABS" on page 55
@ACosine	Calculate the arccosine: "@ACosine" on page 56
@And	Logical AND: "@And" on page 57
@Area	Area under curve of a waveform: "@Area" on page 59
@ASine	Calculate the arcsine: "@ASine" on page 61
@ATan	Calculate the arctangent: "@ATan" on page 62
@ATan2	Calculate the arctangent of two parameters: "@ATan2" on page 63
@BlockFFT	Calculate the main frequency within a block of data: "@BlockFFT" on page 66
@Clip	Clip the amplitude of a waveform: "@Clip" on page 68
@Comparator	Compare values of two waveforms or one waveform and a numerical value: "@Comparator" on page 70
@Cos	Calculate the cosine: "@Cos" on page 72
@CurveFitting	Fit a linear or parabolic waveform to a given waveform: "@CurveFitting" on page 73
@Cut	Cut out a specific segment of a waveform: "@Cut" on page 75
@Cycles	Count the number of cycles in a waveform: "@Cycles" on page 77
@Diff	Differentiate a waveform: "@Diff" on page 79

2 FORMULA DATABASE FUNCTIONS

Function Overview	
Name	Description
@DQ0Transformation	Apply a DQ0Transformation on the input waveforms: " @DQ0Transformation " on page 81
@Energy	Calculate the energy under curve of a waveform: " @Energy " on page 85
@EqualTo	Equal-to evaluation: " @EqualTo " on page 87
@Exp	Exponential: calculate the power (base e) of the input: " @Exp " on page 88
@ExpWave	Generate an exponential waveform: " @ExpWave " on page 89
@FallTime	Determine the falltime of a pulse in a waveform: " @FallTime " on page 91
@FilterBiquad	Returns a waveform filtered using a biquad filter: " @FilterBiquad " on page 93
@FilterButterworthLP	Filters the input signal: " @FilterButterworthLP " on page 96
@FilterButterworthHP	Filters the input signal: " @FilterButterworthHP " on page 98
@FilterButterworthBP	Filters the input signal: " @FilterButterworthBP " on page 100
@FilterButterworthBS	Filters the input signal: " @FilterButterworthBS " on page 102
@FilterBesselLP	Filters the input signal: " @FilterBesselLP " on page 104
@FilterBesselHP	Filters the input signal: " @FilterBesselHP " on page 106
@FilterBesselBP	Filters the input signal: " @FilterBesselBP " on page 108
@FilterBesselBS	Filters the input signal: " @FilterBesselBS " on page 110
@FilterChebyshevLP	Filters the input signal: " @FilterChebyshevLP " on page 112
@FilterChebyshevHP	Filters the input signal: " @FilterChebyshevHP " on page 115
@FilterChebyshevBP	Filters the input signal: " @FilterChebyshevBP " on page 117
@FilterChebyshevBS	Filters the input signal: " @FilterChebyshevBS " on page 119
@FormatDate	Formats a date data source into a string: " @FormatDate " on page 123
@FormatTime	Formats a time data source into a string: " @FormatTime " on page 125

Function Overview	
Name	Description
@Frequency	Determine the frequency of a waveform: "@Frequency" on page 121
@GetNumberFromString	Returns a number from the text value of the input string: "@GetNumberFromString" on page 127
@GreaterEqualThan	Greater-than-or-equal-to evaluation: "@GreaterEqualThan" on page 128
@GreaterThan	Greater-than evaluation: "@GreaterThan" on page 129
@Histogram	Calculate amplitude histogram: "@Histogram" on page 130
@HSFundamentalFrequency	Returns a waveform containing the fundamental frequency: "@HSFundamentalFrequency" on page 132
@HSOneHarmonicAmplitude	Returns a waveform containing the amplitude of a single harmonic: "@HSOneHarmonicAmplitude" on page 133
@HSOneHarmonicPhase	Returns a waveform containing the phase of a single harmonic: "@HSOneHarmonicPhase" on page 138
@IIF	Conditional result: "@IIF" on page 140
@IntLookUp	Modify a waveform by using the waveform data as an index to an external conversion table: "@IntLookUp" on page 143
@IntLookUp12	Modify a waveform by using the waveform data as an index to a conversion table. Optimized for 12-bit data: "@IntLookUp12" on page 146
@IsNaN	Determine if the input parameter is not a number: "@IsNaN" on page 148
@Join	Concatenate two or more waveforms: "@Join" on page 149
@Length	Return the length (in samples) of a waveform: "@Length" on page 151
@LessEqualThan	Less-than-or-equal-to evaluation: "@LessEqualThan" on page 152
@LessThan	Less-than evaluation: "@LessThan" on page 153
@Ln	Calculate the natural logarithm: "@Ln" on page 154
@Log	Calculate the logarithm base 10: "@Log" on page 156
@Max	Determine the maximum value (amplitude) of a waveform: "@Max" on page 157
@MaxNum	Determine the maximum value of a range of numerical values: "@MaxNum" on page 159

Function Overview	
Name	Description
@MaxPos	Return the position of the waveform maximum: "@MaxPos" on page 160
@Mean	Calculate the mean value of a waveform: "@Mean" on page 162
@MedianFilter	Filter a waveform using a median filter. "@MedianFilter" on page 164
@Min	Determine the minimum value (amplitude) of a waveform: "@Min" on page 167
@MinNum	Return the position of the waveform minimum: "@MinNum" on page 169
@MinPos	Return the position – in time – of the absolute minimum: "@MinPos" on page 170
@NextHillPos	Determine the position of the next local maximum in a waveform: "@NextHillPos" on page 172
@NextLvlCross	Determine the position of the next crossing of a waveform with a specified signal level: "@NextLvlCross" on page 174
@NextValleyPos	Determine the position of the next local minimum in a waveform: "@NextValleyPos" on page 176
@Noise	Generate a waveform containing noise: "@Noise" on page 178
@Not	Logical NOT: "@Not" on page 179
@NumberOfSweeps	Returns the number of sweeps: "@NumberOfSweeps" on page 180
@Or	Logical OR: "@Or" on page 182
@Period	Determine the period of a waveform: "@Period" on page 184
@Pow	Exponentiation, base raised to the power exponent: "@Pow" on page 186
@PrevHillPos	Determine the position of the previous local maximum in a waveform: "@PrevHillPos" on page 187
@PrevLvlCross	Determine the position of the previous crossing of a waveform with a specified signal level: "@PrevLvlCross" on page 189
@PrevValleyPos	Determine the position of the previous local minimum in a waveform: "@PrevValleyPos" on page 191

Function Overview	
Name	Description
@PulseWidth	Determine the width of a pulse in a waveform: "@PulseWidth" on page 195
@Ramp	Generate a linear ramp waveform: "@Ramp" on page 197
@ReadAsciiFile	Read waveform data from an ASCII (text) file: "@ReadAsciiFile" on page 199
@ReadLogFile	Read a sequence of numerical values from a Perception log file and creates a waveform: "@ReadLogFile" on page 201
@Reduce	Reduce the number of samples in a waveform by resampling: "@Reduce" on page 205
@RefCheck	Verify a waveform against one or two waveform envelopes: "@RefCheck" on page 206
@RelativeTime2Local	Returns a string representing the absolute local time corresponding to the elapsed time: "@RelativeTime2Local" on page 208
@RelativeTime2UTC	Returns a string representing the absolute UTC time corresponding to the elapsed time: "@RelativeTime2UTC" on page 210
@RemoveGlitch	Remove undesirable samples in a waveform: "@RemoveGlitch" on page 212
@Res2	Sample a waveform so that its length becomes a power of two: "@Res2" on page 213
@RiseTime	Determine the risetime of a pulse in a waveform: "@RiseTime" on page 214
@RMS	Calculate the Root Mean Square value of a waveform: "@RMS" on page 216
@SAEJ211Filter	Filter waveform based on SAE J211 recommendations: "@SAEJ211Filter" on page 218
@Sin	Calculate the sine: "@Sin" on page 219
@SineWave	Generate a sine wave: "@SineWave" on page 220
@Smooth	Smooth a waveform over a selected number of samples: "@Smooth" on page 222
@SpaceVectorInverseTransformation	Calculate a Space Vector Inverse waveform: "@SpaceVectorInverseTransformation" on page 224
@SpaceVectorTransformation	Calculate a Space Vector waveform: "@SpaceVectorTransformation" on page 228

Function Overview	
Name	Description
@Sqrt	Calculate the square root: "@Sqrt" on page 231
@SquareWave	Generate a square wave: "@SquareWave" on page 232
@StdDev	Calculate the standard deviation of a waveform: "@ StdDev" on page 233
@Sweep	Select a sweep in a multi-sweep recording: "@Sweep" on page 235
@SweepEndTime	Returns the end time of a specific sweep: "@SweepEnd-Time" on page 236
@SweepNumberAt-Time	Returns the sweep index: "@SweepNumberAtTime" on page 237
@SweepSource	Returns the source of the trigger of a specific sweep from a multi-sweep signal: "@SweepSource" on page 238
@SweepStartTime	Returns the start time of a specific sweep: "@SweepStart-Time" on page 239
@SweptSineWave	Generates a waveform described by the sine function sweeping from a start frequency to an end frequency: "@SweptSineWave" on page 240
@Tan	Calculate the tangent: "@Tan" on page 242
@TimeMaxAbove	Return a number representing the time interval of the longest period that the signal is above a specified level: "@TimeMaxAbove" on page 243
@TimeMaxAboveBegin	Return a number representing the start time of the longest period that the signal is above a specified level: "@TimeMaxAboveBegin" on page 245
@TimeMaxBelow	Return a number representing the time interval of the longest period that the signal is below a specified level: "@TimeMaxBelow" on page 247
@TimeMaxBelowBegin	Return a number representing the start time of the longest period that the signal is Below a specified level: "@TimeMaxBelowBegin" on page 249
@TimeMinAbove	Return a number representing the time interval of the shortest period that the signal is above a specified level: "@TimeMinAbove" on page 251
@TimeMinAboveBegin	Return a number representing the start time of the shortest period that the signal is above a specified level: "@TimeMinAboveBegin" on page 253

Function Overview	
Name	Description
@TimeMinBelow	Return a number representing the time interval of the shortest period that the signal is below a specified level: " @TimeMinBelow " on page 255
@TimeMinBelowBegin	Return a number representing the start time of the shortest period that the signal is above a specified level: " @TimeMinBelowBegin " on page 257
@TimeTotalAbove	Return a number representing the total amount of time that the signal is above a specified level: " @TimeTotalAbove " on page 259
@TimeTotalAboveBegin	Return a number representing the start of the first period where the signal is above the specified level: " @TimeTotalAboveBegin " on page 261
@TimeTotalBelow	Return a number representing the total amount of time that the signal is below a specified level: " @TimeTotalBelow " on page 263
@TimeTotalBelowBegin	Return a number representing the start of the first period where the signal is below the specified level: " @TimeTotalBelowBegin " on page 265
@TriggerTime	Return the trigger position: " @TriggerTime " on page 267
@TriggerTimeToText	Return the trigger position in a time-date formatted string: " @TriggerTimeToText " on page 269
@TrueRMS	Calculate RMS value: " @TrueRMS " on page 273
@TrueRMSRef	Returns a waveform that represents the true RMS per number of cycles: " @TrueRMSRef " on page 274
@Value	Return the amplitude value of a waveform at a specified x-position: " @Value " on page 276
@XDelta	Return the sampling interval of a waveform: " @XDelta " on page 277
@XDeltaHigh	Return the maximum sampling interval in a multitimebase recording: " @XDeltaHigh " on page 278
@XDeltaLow	Return the minimum sampling interval in a multitimebase recording: " @XDeltaLow " on page 279
@XFirst	Return the x-coordinate (with respect to the trigger point) of the first sample in a waveform: " @XFirst " on page 280
@XLast	Return the x-coordinate (with respect to the trigger point) of the last sample in a waveform: " @XLast " on page 281

2 FORMULA DATABASE FUNCTIONS

Function Overview	
Name	Description
@XShift	Shift a waveform in time: "@XShift" on page 282
@XYArray	Create a waveform from a list of X/Y-value pairs: "@XYArray" on page 283
@YArray	Create a waveform from a list of Y-values: "@YArray" on page 285

2.2.2 Function Overview – HIC section

Function Overview – HIC section	
Name	Description
@Con3ms	Return the highest acceleration level: "@Con3ms" on page 287
@Con3ms_T	Return the start time of the highest acceleration level: "@Con3ms_T" on page 289
@Cum3ms	Return the highest acceleration level: "@Cum3ms" on page 290
@Cum3ms_T	Return the start time of the highest acceleration level: "@Cum3ms_T" on page 292
@HIC	Calculate the HIC of a waveform: "@HIC" on page 293
@HICStartTime	Return the start of the interval: "@HICStartTime" on page 295
@HICEndTime	Return the end of the interval: "@HICEndTime" on page 296
@HIC15	Return the maximum HIC value with a fixed 15 millisecond interval: "@HIC15" on page 297
@HIC15StartTime	Return the start of the interval during the HIC15 calculations: "@HIC15StartTime" on page 298
@HIC15EndTime	Return the end of the interval during the HIC15 calculations: "@HIC15EndTime" on page 299
@HIC36	Return the maximum HIC value with a fixed 36 millisecond interval: "@HIC36" on page 300
@HIC36StartTime	Return the start of the interval during the HIC36 calculations: "@HIC36StartTime" on page 301
@HIC36EndTime	Return the end of the interval during the HIC36 calculations: "@HIC36EndTime" on page 302

2.2.3 Function Overview – Cycle Math section

Function Overview – Cycle Math section	
Name	Description
@CycleArea	Calculate the area of every cycle detected on a reference signal: "@CycleArea" on page 307
@CycleCount	Count the number of cycles detected on a reference signal: "@CycleCount" on page 309
@CycleCrestFactor	Calculates the "Crest Factor" of every cycle detected on a reference signal: "@CycleCrestFactor" on page 310
@CycleDetect	Perform level crossing detection on the input waveform: "@CycleDetect" on page 312
@CycleEnergy	Calculate the energy of cycles detected on a reference signal: "@CycleEnergy" on page 314
@CycleFrequency	Calculate the frequency of cycles detected on the reference signal: "@CycleFrequency" on page 316
@CycleFundamental	Generate the Waveform of the Cycle Fundamental: "@CycleFundamental" on page 318
@CycleFundamental-Phase	Calculate the phase difference between the Cycle Fundamentals: "@CycleFundamentalPhase" on page 320
@ CycleFundamentalRMS	Calculate the RMS value of the Cycle Fundamental: "@CycleFundamentalRMS" on page 322
@CycleInterval	Generate a square waveform with a specified interval between the high and low states: "@CycleInterval" on page 324
@CycleLevel	Return a waveform that uses the value of the input signal at the start of a cycle of the reference signal and repeat that value during half, one or multiple cycles of that reference signal: "@CycleLevel" on page 326
@CycleMax	Calculate the maximum value in cycles detected on a reference signal: "@CycleMax" on page 328
@CycleMean	Calculate the mean value in cycles detected on a reference signal: "@CycleMean" on page 329
@CycleMin	Calculate the minimum value in cycles detected on a reference signal: "@CycleMin" on page 331
@CyclePeriod	Calculate the period of cycles detected on a reference signal: "@CyclePeriod" on page 332

Function Overview – Cycle Math section	
Name	Description
@CyclePhase	Calculates the phase difference θ between two waveforms: "@CyclePhase" on page 333
@CycleRPM	Calculates the revolutions per minute: "@CycleRPM" on page 335
@CycleRMS	Calculate the root mean square value of cycles detected on a reference signal: "@CycleRMS" on page 336
@CycleStdDev	Calculate the standard deviation of every cycle detected on a reference signal: "@CycleStdDev" on page 338
@CycleTHD	Calculates the Total Harmonic Distortion: "@CycleTHD" on page 340

3 Arithmetic Operations

3.1 + (Addition)

Function

Adds left and right expression.

Syntax

Expression1 + *Expression2*

Parameters

<i>Expression1</i>	Left expression
<i>Expression2</i>	Right expression

Output

The result is the sum of the left and right expression.

Description

An expression can be:

- Waveform variable
- Function call
- Numerical variable
- Constant value

Normal operator precedence (first multiplication and division, then addition and subtraction) applies. Parentheses can be used to change the operator precedence in more complicated expressions.

Example

Some examples of valid expressions using addition are:

- $2 + 3$
- $\text{var1} + 4$
- $5 + \text{var2}$
- $1000 + @Noise(1E6; 1000)$
- $(\text{Var1} + \text{Var2}) * (\text{Var3} - \text{Var4})$

When adding numerical values, the output is a numerical value.

When adding a waveform and a numerical value, the output is a waveform. The number of points (length) of the output waveform is equal to the length of the input waveform.

When adding two waveforms, the output is a waveform formed by a point-by-point addition of the two input waveforms. The x-scaling of the output waveform is equal to the x-scaling of the left waveform. The length of the output waveform is equal to the shortest of the lengths of the two input waveforms. For a meaningful result, the x-scaling of both waveforms should be equal.

See also

"* (Multiplication)" on page 49, "-" (Subtraction)" on page 47, "/" (Division)" on page 51

3.2 - (Subtraction)

Function

Subtracts left and right expression.

Syntax

Expression1 – *Expression2*

Parameters

<i>Expression1</i>	Left expression
<i>Expression2</i>	Right expression

Output

The result is the difference of the left and right expression.

Description

An expression can be:

- Waveform variable
- Function call
- Numerical variable
- Constant value

Normal operator precedence (first multiplication and division, then addition and subtraction) applies. Parentheses can be used to change the operator precedence in more complicated expressions.

Example

Some examples of valid expressions using subtraction are:

- 3 - 2
- var1 - 4
- 5 - var2
- @SineWave(1E6; 1000; 1k) - 1
- (Var1 + Var2) * (Var3 - Var4)

When subtracting numerical values, the output is a numerical value.

When subtracting a waveform and a numerical value, the output is a waveform. The number of points (length) of the output waveform is equal to the length of the input waveform.

When subtracting two waveforms, the output is a waveform formed by a point-by-point subtraction of the two input waveforms. The x-scaling of the output waveform is equal to the x-scaling of the left waveform. The length of the output waveform is equal to the shortest of the lengths of the two input waveforms. For a meaningful result, the x-scaling of both waveforms should be equal.

See also

"* (Multiplication)" on page 49, "+ (Addition)" on page 45 "/" (Division)" on page 51

3.3 * (Multiplication)

Function

Multiplies left and right expression.

Syntax

Expression1 * *Expression2*

Parameters

<i>Expression1</i>	Left expression
<i>Expression2</i>	Right expression

Output

The result is the product of the left and right expression.

Description

An expression can be:

- Waveform variable
- Function call
- Numerical variable
- Constant value

Normal operator precedence (first multiplication and division, then addition and subtraction) applies. Parentheses can be used to change the operator precedence in more complicated expressions.

Example

Some examples of valid expressions using multiplication are:

- $2 * 3$
- $\text{var1} * 4$
- $5 * \text{var2}$
- $10 * \text{@SineWave}(1\text{E}6; 1000; 1\text{k})$
- $(\text{Var1} - \text{AvgVar1}) * (\text{Var2} - \text{AvgVar2})$

When multiplying numerical values, the output is a numerical value.

When multiplying a waveform and a numerical value, the output is the scaled waveform. The number of points (length) of the output waveform is equal to the length of the input waveform.

When multiplying two waveforms, the output is a waveform formed by a point-by-point multiplication of the two input waveforms. The x-scaling of the output waveform is equal to the x-scaling of the left waveform. The length of the output waveform is equal to the shortest of the lengths of the two input waveforms. For a meaningful result, the x-scaling of both waveforms should be equal.

See also

"+" (Addition)" on page 45, "-" (Subtraction)" on page 47, "/" (Division)" on page 51

3.4 / (Division)

Function

Divides left and right expression.

Syntax

Expression1 / *Expression2*

Parameters

<i>Expression1</i>	Left expression (dividend)
<i>Expression2</i>	Right expression (divisor)

Output

The result is the quotient of the left and right expression.

Description

An expression can be:

- Waveform variable
- Function call
- Numerical variable
- Constant value

Normal operator precedence (first multiplication and division, then addition and subtraction) applies. Parentheses can be used to change the operator precedence in more complicated expressions.

Example

Some examples of valid expressions using division are:

- $3 / 2$
- $\text{var1} / 4$
- $5 / \text{var2}$
- $\text{@SineWave}(1\text{E}6; 1000; 1\text{k}) / 10$
- $(\text{Var1} + \text{Var2}) / (\text{Var3} - \text{Var4})$

When dividing numerical values, the output is a numerical value.

When dividing a waveform and a numerical value, the output is a waveform. The number of points (length) of the output waveform is equal to the length of the input waveform.

When dividing two waveforms, the output is a waveform formed by a point-by-point division of the two input waveforms. The x-scaling of the output waveform is equal to the x-scaling of the left waveform. The length of the output waveform is equal to the shortest of the lengths of the two input waveforms. For a meaningful result, the x-scaling of both waveforms should be equal.

Returns an undefined (unknown) value when the divisor is a numerical value equal to zero or when the denominator is a waveform containing a sample with the value zero.

See also

"* (Multiplication)" on page 49, "+ (Addition)" on page 45 "- (Subtraction)" on page 47

3.5 – (Unary minus)

Function

Inverts the sign of an expression.

Syntax

– Expression

Parameters

Expression Expression to be inverted

Output

The result is the expression multiplied by -1 (minus one).

Description

The expression can be any expression containing:

- Waveform variable
- Function call
- Numerical variable
- Constant value

Normal operator precedence (first multiplication and division, then addition and subtraction) applies. Parentheses can be used to change the operator precedence in more complicated expressions.

Example

Some examples of valid expressions using unary minus are:

- - 2
- - var1
- 5 * var2
- - @SineWave(1E6; 1000; 1k) - 1)
- - ((Var1 + Var2) * (Var3 - Var4))

When inverting a numerical value, the output is a numerical value.

When inverting a waveform, the output is the negated waveform. The number of points (length) of the output waveform is equal to the length of the input waveform.

See also

"* (Multiplication)" on page 49, "+ (Addition)" on page 45 "- (Subtraction)" on page 47, "/" (Division)" on page 51

3.6 @Modulo

Function

Returns a waveform or numerical value with the **modulo** result.

Syntax

`@Modulo(Par1, Par2)`

Parameters

<i>Par1</i>	Input waveform or numerical value.
<i>Par2</i>	Numerical value

Output

Waveform or numerical value containing the modulo operation of the input.

Description

The modulo operation divides the input with the supplied argument *Par2*. The remainder of that division is returned.

Example

The below would divide the input signal `Formula.MyAngle` by 360 and return the remainder. In this case it is ensured the angle signal never exceeds the 360 value.

```
Angle = @Modulo(Formula.MyAngle; 360 )
```

4 Reference Guide

4.1 @ABS

Function

Calculates the **absolute** value of the parameter.

Syntax

`@Abs(Par)`

Parameters

Par Input waveform or numerical value.

Output

Absolute value of the waveform or numerical value.

Description

Calculates the absolute value of the input waveform or numerical value. Positive values remain unchanged, negative values change their sign. This function can be used to rectify signals or to force positive values for results.

Example

The following example creates a sine wave and rectifies this signal:

```
Signal = @SineWave(1E6; 1000; 1k)
Rectif = @Abs(Formula.Signal)
```

4.2 @ACosine

Function

Returns a waveform or numerical value representing the **arccosine** of a waveform or a number. The result is in radians.

Syntax

@ACosine(*Input*)

Parameters

Input Input waveform or numerical value.

Output

Waveform or numerical value containing the arccosine of the input.

Description

The arccosine function returns the angle for which the cosine equals the argument. The angle returned is in radians. The arccosine is the inverse trigonometric function of the cosine function. That is the function:

$$y = \arccosine(x)$$

is defined so that:

$$\text{sine}(y) = x$$

The input domain of arccosine is:

$$-1 \leq x \leq 1$$

for real result values in the range:

$$0 \leq y \leq \pi$$

Example

The function below would calculate the angle in radians for which the cosine would amount to 0.

CosValue = @ACos(1)

See also

"@Cos" on page 72

4.3 @And

Function

Performs a logical **AND** evaluation on the input parameters.

Syntax

`@And(Param1; ...; ParamN)`

Parameters

Param1 Number: first parameter used for the AND evaluation.
ParamN Last parameter used for the AND evaluation. With N >= 2.

Output

The output is 1 or a 0.

Description

The @And function performs a logical AND evaluation on the input parameters. Depending on the evaluation the result will be 1 or 0. A numerical value not equal to 0 corresponds to a logical "True" and a numerical 0 corresponds to a logical "False".

The truth table of the AND function is:

Param1	Param2	Result
True	True	True
True	False	False
False	True	False
False	False	False

Example

The following is a list of examples and their return value.

```
AndExamp11 = @And(1; 1; 1)      => 1 (=true)
AndExamp12 = @And(1; 4; 10)     => 1 (=true)
AndExamp13 = @And(1; 4; 0)      => 0 (=false)
AndExamp14 = @And(0; 0; 0)      => 0 (=false)
```

See also

"@Not" on page 179 and "@Or" on page 182

4.4 @Area

Function

Calculates the **area** under curve of a waveform.

Syntax

```
@Area(Waveform)
@Area(Waveform; Begin)
@Area(Waveform; Begin; End)
```

Parameters

<i>Waveform</i>	Input waveform for which the area under curve is to be calculated.
<i>Begin</i>	Number: segment begin
<i>End</i>	Number: segment end

Output

The output is a numerical value.

Description

The area under curve is calculated using the following formula:

$$\text{Area} = \left[\left[\sum_{n=n_1}^{n_2} y(n) \right] - \frac{y(n_1) + y(n_2)}{2} \right] \cdot \Delta x$$

n_1 = first sample with $x \geq \text{Begin}$

n_2 = last sample with $x \leq \text{End}$

Δx = x - difference between two samples

The segment limits (Begin and End) are used to select a range of samples. If no segment limits are specified, the complete waveform is used. When only Begin is specified, the waveform segment from Begin to the end of the waveform is used.

Notice

The **Begin** and **End** parameters have to be specified in the units of the horizontal axis (for example time) and not in samples.

The numerical integration of the curve is performed assuming linear interpolation of the curve between the samples.

Example

The following example creates a sine wave of 50 Hz and calculates the area under curve of the first half period of the signal:

```
Signal = @SineWave(50k; 1000; 50)
Area   = @Area(Formula.Signal; 0; 10m)
```

See also

"@Energy" on page 85 and "@Mean" on page 162

4.5 @ASine

Function

Returns a waveform or numerical value representing the **arcsine** of a waveform or a number. The result is in radians.

Syntax

@ASine(*Input*)

Parameters

Input Input waveform or numerical value.

Output

Waveform or numerical value containing the arcsine of the input.

Description

The arcsine function returns the angle for which the sine equals the argument. The angle returned is in radians. The arcsine is the inverse trigonometric function of the sine function. That is the function:

$$y = \arcsine(x)$$

is defined so that:

$$\text{sine}(y) = x$$

The input domain of arcsine is:

$$-1 \leq x \leq 1$$

for real result values in the range:

$$-\pi/2 \leq y \leq \pi/2$$

Example

The function below would calculate the angle in radians for which the sine would amount to 1.

```
CosValue = @ASine( 1 )
```

See also

"@Sin" on page 219

4.6 @ATan

Function

Calculates the **arctangent** of the input parameter.

Syntax

@ATan(*Par*)

Parameters

Par Input waveform or numerical value.

Output

Waveform or numerical value containing the arctangent of the input.

Description

The arctangent function returns the angle for which the tangent equals the argument. The angle is returned in radians. The arctangent is the inverse trigonometric function of the tangent.

Example

The following example calculates Pi by multiplying ATan(1) by four:

```
Pi = 4 * @ATan(1)
```

See also

"@Cos" on page 72, "@Sin" on page 219 and "@Tan" on page 242

4.7 @ATan2

Function

Calculates the **arctangent** of the two input parameters.

Syntax

`@ATan2(Waveform1; Par2)`

Parameters

<i>Waveform1</i>	First input waveform
<i>Par2</i>	Second input waveform or numerical value.

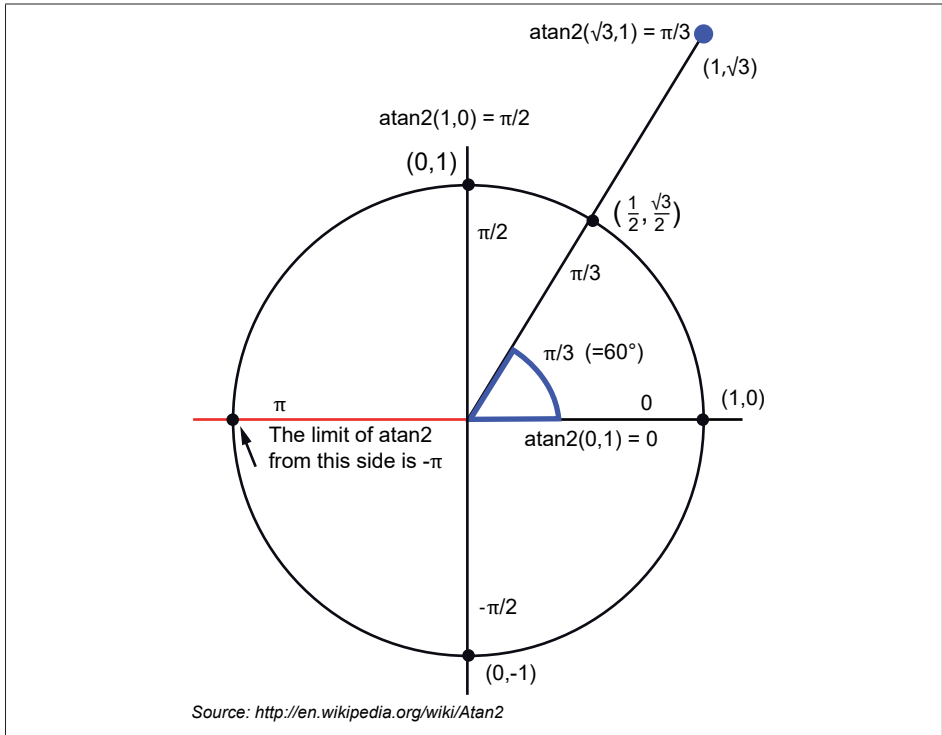
Output

A waveform containing the arctangent of the two input parameters (either two input waveforms or an input waveform and a numerical value). The values of the output waveform are in radians.

Description

The `Atan2` function is the arctangent function with two arguments. The purpose of using two arguments instead of one (`Atan` function) is to gather information on the signs of the inputs in order to return the appropriate quadrant of the computed angle, which is not possible for the single-argument arctangent function.

For any arguments x and y not both equal to zero, `atan2(y, x)` is the angle in radians between the positive x -axis of a plane and the point given by the coordinates (x, y) on it. The angle is positive for counter-clockwise angles (upper half-plane, $y > 0$), and negative for clockwise angles (lower half-plane, $y < 0$).



Result:

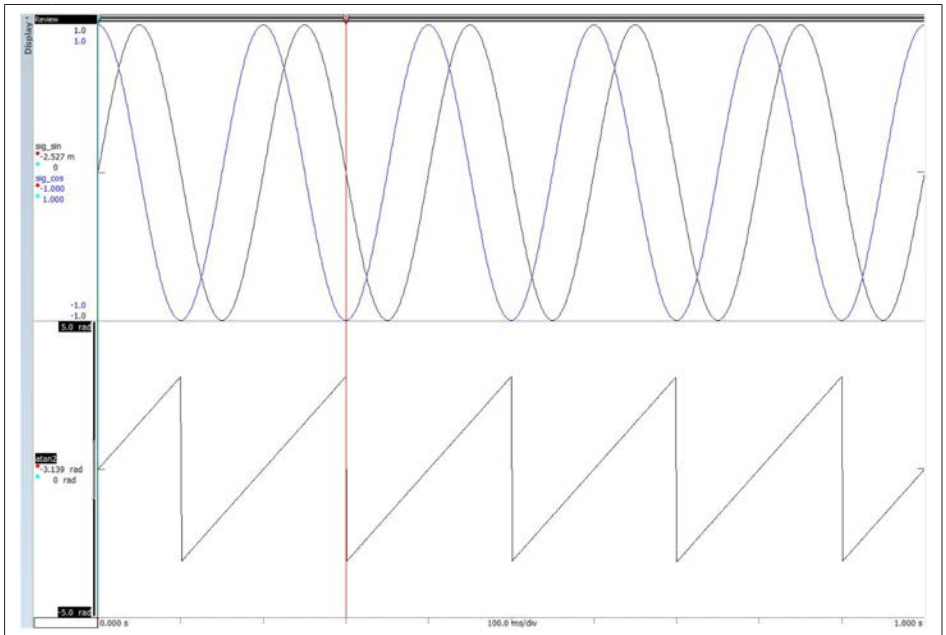


Fig. 4.2 Result of ATan2 Example

See also

"@ATan" on page 62

4.8 @BlockFFT

Function

Returns a waveform representing the maximum frequency detected per **block** of the input waveform.

Syntax

@BlockFFT(*Waveform; Size; Space*)
@BlockFFT(*Waveform; Size; Space; Begin*)
@BlockFFT(*Waveform; Size; Space; Begin; End*)

Parameters

<i>Waveform</i>	Input waveform
<i>Size</i>	Number: block size in milliseconds.
<i>Space</i>	Number: spacing between start of two successive blocks in milliseconds.
<i>Begin</i>	Number: start position of BlockFFT function.
<i>End</i>	Number: end position of BlockFFT function.

Output

Waveform containing frequency versus time.

Description

This function calculates the maximum frequency per block using an FFT algorithm. The parameter *Size* determines the size (length) of the block in milliseconds. The *Space* parameter determines the spacing between the start of two successive blocks in milliseconds.

Refer to the following diagram as an example for the relation between *Space* and *Block*.

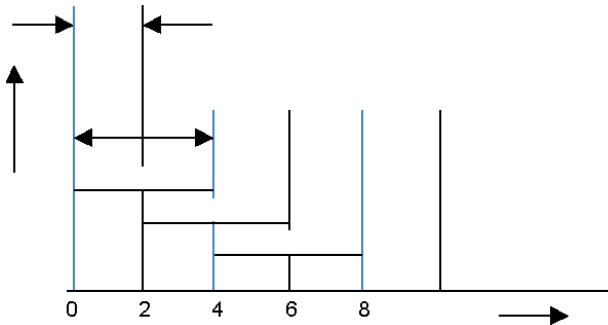


Fig. 4.3 Example – Relation between spacing and block size

The output of the function is a waveform containing a single maximum frequency per block as function of time. The sample spacing of this output waveform is equal to the Space parameter.

The segment limits (Begin and End) are used to select a range of the waveform in which BlockFFT's are calculated. If no segment limits are specified, the complete waveform is used. When only Begin is specified, the waveform segment from Begin to the end of the waveform is used.

Example

The following example calculates the BlockFFT of a composed waveform:

```
Sine1  = 2 * @SineWave(1000k; 30k; 10k)
Sine2  = @SineWave(1000k; 30k; 15k)
Sine3  = 1.5 * @SineWave(1000k; 30k; 5k)
Signal = @Join(Formula.Sine1; Formula.Sine2;
               Formula.Sine3)
Result = @BlockFFT(Formula.Signal; 4; 2)
```

4.9 @Clip

Function

Clips a waveform between the lower and upper bound specified.

Syntax

`@Clip(Waveform; LowerBound; UpperBound)`

Parameters

<i>Waveform</i>	Waveform whose amplitude range is to be clipped.
<i>LowerBound</i>	Number: lower bound value used for clipping.
<i>UpperBound</i>	Number: upper bound value used for clipping.

Output

Waveform with all sample values clipped between the lower and upper bound.

Description

For each sample in the waveform a comparison is made against the lower and upper bound of the clipping range.

If the sample value is between these two values, it is not modified.

If the sample value is larger than the upper bound, it is set to the upper bound.

If the sample value is lower than the lower bound, it is set to the lower bound.

Example

The example clips the 1.2 V sine wave to the limits -1 and 1 to simulate input overflow.

```
Signal      = 1.2 * @SineWave(20k; 1000; 50)
InpSignal = @Clip(Formula.Signal; -1; 1)
```

The following example determines the cumulative time an acceleration signal is above 150 g. The technique used is to first clip the signal between 150 g and 0.001 g more than 150 g. From the clipped signal 150 is subtracted and the clipped signal is scaled up by a factor of 1000. This technique results in 0 when the acceleration is below 150 g and in 1 when it is above 150 g. The area under this curve is the cumulative time above 150 g.

4 REFERENCE GUIDE

```
Accel      = 150 + @SineWave(20k; 100; 50)
Temp       = 1000 * (@Clip(Formula.Accel; 150; 150.001) - 150)
CumTime    = @Area(Formula.Temp)
```

See also

"@Cut" on page 75

4.10 @Comparator

Function

Compares the sample values of two waveforms or of one waveform and a numerical value.

Syntax

@Comparator(*Waveform1*; *Par2*)

Parameters

<i>Waveform1</i>	First input waveform
<i>Par</i>	Second input waveform or numerical value.

Output

A waveform that is the result of the comparison between the sample values of the first waveform and the second waveform or the numerical value.

When $INP1 < INP2$ then the output value is -1.

When $INP1$ is equal to $INP2$ then the output value is 0.

When $INP1 > INP2$ then the output value is 1.

Description

The Comparator function compares the two input waveforms with each other and returns -1, 0 or 1 dependent on the relationship between the input values.

The second parameter can also be a numerical value. In that case this value is used for the comparison.

Example

The following example calculates a waveform that indicates the relationship between the two input signals.

```
sig_sin = @SineWave(1k;1k+1;5)
sig_cos = @SineWave(1k;1k+1;5;90)
comp    = @Comparator(Formula.sig_sin; Formula.sig_cos)
```

Result:

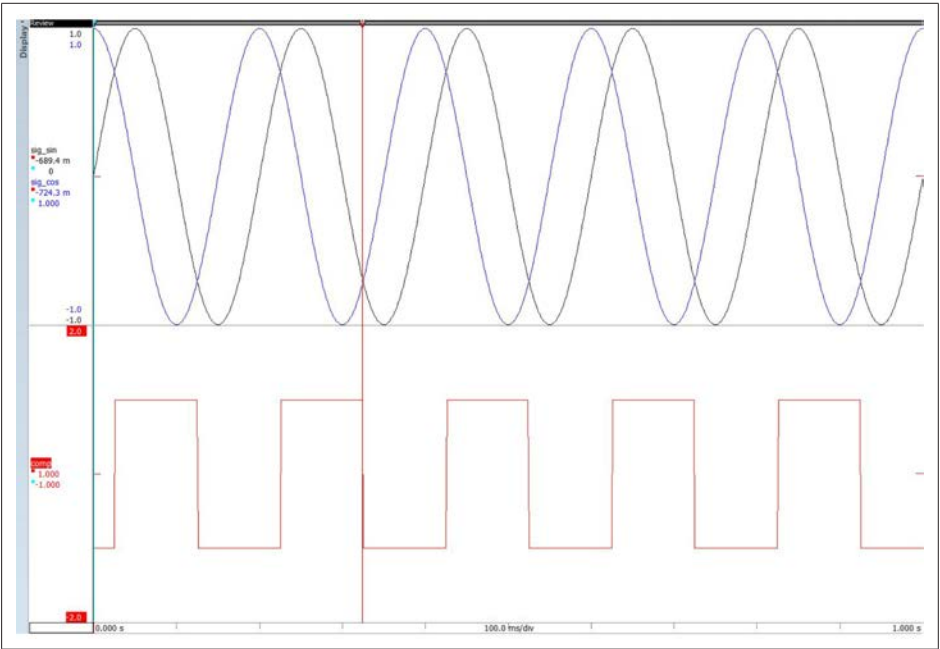


Fig. 4.4 Result of Comparator example

4.11 @Cos

Function

Calculates the **cosine** of the input parameter.

Syntax

`@Cos(Par)`

Parameters

Par Input waveform or numerical value.

Output

Waveform or numerical value containing the cosine of the input.

Description

The trigonometric function cosine is calculated assuming the input parameter is the angle in radians. When a waveform parameter is used, the cosine is calculated for each individual sample.

Example

The following example calculates the cosine of the variable "Angle" specified in degrees:

```
Angle      = 33
AngleRad   = System.Constants.Pi * Formula.Angle / 180
CosAngle   = @Cos(Formula.AngleRad)
```

See also

"@ATan" on page 62, "@Sin" on page 219 and "@Tan" on page 242

4.12 @CurveFitting

Function

Returns a waveform that has the best fit to a series of data points from the input signal, using linear interpolation or a parabolic regression.

Syntax

`@CurveFitting(Waveform)`
`@CurveFitting(Waveform; Order)`
`@CurveFitting(Waveform; Order; BeginIntv)`
`@CurveFitting(Waveform; Order; BeginIntv; EndIntv)`
`@CurveFitting(Waveform; Order; BeginIntv; EndIntv; Begin)`
`@CurveFitting(Waveform; Order; BeginIntv; EndIntv; Begin; End)`

Parameters

<i>Waveform</i>	Input waveform
<i>Order</i>	Number: regression order. 1 = linear, 2 = parabolic. Default is linear regression.
<i>BeginIntv</i>	Number: begin of the interval used for interpolation. Default the start of the input waveform is used.
<i>EndIntv</i>	Number: end of the interval used for interpolation. Default the end of input waveform is used.
<i>Begin</i>	Number: begin time from where the output curve starts. Default the start time of the input waveform is used.
<i>End</i>	Number: end time to where the output curve ends. Default the end time of the input waveform is used.

Output

A waveform being the linear or parabolic regressing of the input waveform or a part of it.

Description

This function seeks a curve that has the best fit to a series of data points from the input signal. Depending on the parameters all data points or just a limited interval of data points from the input signal are used.

If the Order parameter is set to 1 a linear regression will be done: the output signal will be a straight line.

If the Order parameter is set to 2 a parabolic regression will be done: the output signal will be a parabola.

The least squares algorithm is used to do the calculations.

Example

The following examples create a linear and a parabolic fit:

```
Signal          = @SineWave(8000; 8001; 5)
LineFit         = @CurveFitting(Formula.Signal; 1; 90m; 110m)
ParabolicFit    = @CurveFitting(Formula.Signal; 2; 125m; 175m)
```

4.13 @Cut

Function

Cuts out a specific part of a waveform.

Syntax

@Cut(Waveform; Begin; End)

Parameters

Waveform	Input waveform from which a segment is to be selected.
Begin	Number: segment begin
End	Number: segment end

Output

Waveform segment.

Description

A specific part of a waveform can be selected for further processing. The function calculates the number of samples for the output based on the Begin and End values. The first sample is the sample with an x-coordinate nearest to Begin. The last sample is the sample with an x-coordinate nearest to End.

Notice

The **Begin** and **End** parameters have to be specified in the units of the horizontal axis (for example time) and not in samples. If the segment limits are located outside the x-range of the waveform, these values are limited to that range.

Example

The following example cuts out the segment of a signal between 100 ms and 200 ms:

```
Signal      = @SineWave(10000; 10000; 100)
Segment     = @Cut(Formula.Signal; 100m; 200m)
```

If a specific range of samples is required, the information functions can be used to calculate the proper range to select.

The following example selects the first 1024 points of the previous signal, and uses -1E20 as a very small value:

```
XEnd      = @XBegin(Formula.Signal)
           + 1023 * @XDelta(Formula.Signal)
First1024 = @Cut(Formula.Signal; -1E20; Formula.XEnd)
```

See also

"@Join" on page 149, "@Length" on page 151, "@XFirst" on page 280, "@XDelta" on page 277 and "@XLast" on page 281

4.14 @Cycles

Function

Calculates the number of **cycles** in a waveform or waveform segment.

Syntax

@Cycles(Waveform)
@Cycles(Waveform; Begin; End)
@Cycles(Waveform; Begin)

Parameters

<i>Waveform</i>	Input waveform from which the cycles are calculated.
<i>Begin</i>	Number: segment begin
<i>End</i>	Number: segment end

Output

The number of cycles.

Description

The @Cycles function calculates the number of times a periodically repeated sequence of samples (=Cycle) occurs in the waveform. The 50% level between the maximum and minimum amplitude values is used to count the level crossings to get the number of cycles.

The segment limits (Begin and End) are used to select a range of samples in which cycles are calculated. If no segment limits are specified, the complete waveform is used. When only Begin is specified, the waveform segment from Begin to the end of the waveform is used.

Notice

The **Begin** and **End** parameters have to be specified in the units of the horizontal axis (for example: time) and not in samples.

Example

The following example calculates the number of cycles of a complete signal, and between the two cursors within a display named "display".

```
Signal      = @SineWave(10k; 10001; 50)
Cycles      = @Cycles(Formula.Signal)
Start       = Display.Display.Cursor1.XPosition
End         = Display.Display.Cursor2.XPosition
Cycles_BC   = @Cycles(Formula.Signal;
                      Formula.Start; Formula.End)
```

See also

"@Frequency" on page 121 and "@Period" on page 184

4.15 @Diff

Function

Differentiates a waveform.

Syntax

@Diff(*Waveform*)

Parameters

Waveform Waveform to be differentiated.

Output

Differentiated waveform

Description

The derivative is a measurement of how a function changes when the values of its inputs change. Loosely speaking, a derivative can be thought of as how much a quantity is changing at some given point. The process of finding a derivative is called differentiation. Otherwise stated differentiation is used to determine the slope of a signal instead of the value. The slope is calculated by calculating the difference between adjacent samples and dividing this difference by the sampling interval:

$$\text{Diff}(n) = \frac{y(n) - y(n-1)}{\Delta x} \quad \text{for } n = 2, \dots, N$$
$$\text{Diff}(1) = 0$$

Δx = x - difference between two samples

N = number of samples

Differentiation enhances the high frequency components like high frequency noise and the digitization errors caused by rounding effects. To get a better estimate for the slope, smoothing of the waveform (either before or after differentiation) is recommended.

Example

The following example creates a noisy sine wave and differentiates this signal. The resulting waveform is smoothed to get a more accurate estimate of the slope of the sine wave.

4 REFERENCE GUIDE

```
Signal = @SineWave(10k; 1000; 50)
Differ = @Diff(Formula.Signal)
Slope  = @Smooth(Formula.Differ; 7)
```

See also

"@Integrate" on page 142 and "@Energy" on page 85

4.16 @DQ0Transformation

Function

Applies the **DQ0 Transformation** on the input signals.

Syntax

@DQ0Transformation(*Signal1; Signal2; Signal3; ReferenceAngle; Component; Transformation; OrientationRule*)

Parameters

<i>Signal1</i>	Input waveform of phase1 signal.
<i>Signal2</i>	Input waveform of phase2 signal.
<i>Signal3</i>	Input waveform of phase3 signal.
<i>Reference Angle</i>	Waveform or Value of Reference Angle [rad]
<i>Component</i>	Required output Component (0="direct", 1="quadrature", 2="zero")
<i>Transformation Optional</i>	Invariant Transformation Rule (0="amplitude", 1="power") default is "amplitude"
<i>Orientation Rule Optional</i>	Orientation Rule (0="wiki", 1="matlab") default is "wiki"

Output

The waveform which results for the required component, transformation and required orientation rule transformations.

Description

The direct-quadrature-zero (DQ0) transformation (also known as Park's transformation) is a mathematical transformation that is used to simplify the analysis of three-phase power systems.

The DQ0 Transformation is defined by the following matrix and can be applied for any three-phase quantities (e.g. voltage, currents, etc.)

For the “wiki” orientation rule, the following formula is applied for the amplitude-invariant transformation:

$$\begin{bmatrix} I_{Ad} \\ I_{Aq} \\ I_{A0} \end{bmatrix} = \frac{2}{3} \begin{bmatrix} \cos(\theta) & \cos\left(\theta - \frac{2\pi}{3}\right) & \cos\left(\theta + \frac{2\pi}{3}\right) \\ -\sin(\theta) & -\sin\left(\theta - \frac{2\pi}{3}\right) & -\sin\left(\theta + \frac{2\pi}{3}\right) \\ 1/2 & 1/2 & 1/2 \end{bmatrix} \begin{bmatrix} I_u \\ I_v \\ I_w \end{bmatrix}$$

And for the power-invariant transformation, the following formula is applied:

$$\begin{bmatrix} I_{Pd} \\ I_{Pq} \\ I_{P0} \end{bmatrix} = \sqrt{\frac{2}{3}} \begin{bmatrix} \cos(\theta) & \cos\left(\theta - \frac{2\pi}{3}\right) & \cos\left(\theta + \frac{2\pi}{3}\right) \\ -\sin(\theta) & -\sin\left(\theta - \frac{2\pi}{3}\right) & -\sin\left(\theta + \frac{2\pi}{3}\right) \\ 1/\sqrt{2} & 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}$$

For the “MATLAB” orientation rule, the following formula is applied for the amplitude-invariant transformation:

$$\begin{bmatrix} I_{Ad} \\ I_{Aq} \\ I_{A0} \end{bmatrix} = \frac{2}{3} \begin{bmatrix} \sin(\theta) & \sin\left(\theta - \frac{2\pi}{3}\right) & \sin\left(\theta + \frac{2\pi}{3}\right) \\ \cos(\theta) & \cos\left(\theta - \frac{2\pi}{3}\right) & \cos\left(\theta + \frac{2\pi}{3}\right) \\ 1/2 & 1/2 & 1/2 \end{bmatrix} \begin{bmatrix} I_u \\ I_v \\ I_w \end{bmatrix}$$

And for the power-invariant transformation, the following formula is applied:

$$\begin{bmatrix} I_{Pd} \\ I_{Pq} \\ I_{P0} \end{bmatrix} = \sqrt{\frac{2}{3}} \begin{bmatrix} \sin(\theta) & \sin\left(\theta - \frac{2\pi}{3}\right) & \sin\left(\theta + \frac{2\pi}{3}\right) \\ \cos(\theta) & \cos\left(\theta - \frac{2\pi}{3}\right) & \cos\left(\theta + \frac{2\pi}{3}\right) \\ 1/\sqrt{2} & 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix} \begin{bmatrix} I_u \\ I_v \\ I_w \end{bmatrix}$$

Where	
I_u	Value of the phase1 signal
I_v	Value of the phase2 signal
I_w	Value of the phase3 signal
θ	Value of the reference angle
I_{Ad}	Value of the amplitude-invariant direct component
I_{Aq}	Value of the amplitude-invariant quadrature component
I_{AO}	Value of the amplitude-invariant zero component
I_{Pd}	Value of the power-invariant direct component
I_{Pq}	Value of the power-invariant quadrature component
I_{PO}	Value of the power-invariant zero component

Notes:

- The *Transformation* parameter is optional and has the default value 0="amplitude".
- The *Orientation Rule* parameter is optional and has the default value 0="wiki".
- The input waveforms (*phase1*, *phase2* & *phase3*) all required the same sampling frequency. The same applies to the *Reference Angle* when it is a Waveform.
- This function is only available in the Formula database sheet (see "Formula database sheet" on page 12) if the eDrive option (1-PERC-OP-EDR) is part of your license.

Example

Given that the three-phase input signals are available as *Formula.Phase1*, *Formula.Phase2*, *Formula.Phase3* and the *Reference Waveform* *Formula.RefVal*, the following DQ0 amplitude-invariant transformation waveforms can be defined:

Name	Formula
DQ0ATd	= @DQ0Transformation(Formula.phase1;Formula.phase2;Formula.phase3;Formula.refAng;0)
DQ0ATq	= @DQ0Transformation(Formula.phase1;Formula.phase2;Formula.phase3;Formula.refAng;1)
DQ0ATz	= @DQ0Transformation(Formula.phase1;Formula.phase2;Formula.phase3;Formula.refAng;2)

The Reference Waveform may also be specified as a constant value :

Name	Formula
DQ0ATd_2	= @DQ0Transformation(Formula.phase1;Formula. phase2;Formula.phase3; System.Constants.Pi; 0)

Or the Reference Waveform has a value of 0 (zero):

Name	Formula
DQ0ATd_3	= @DQ0Transformation(Formula.phase1;Formula. phase2;Formula.phase3;0;0)

Likewise the power-invariant variations are:

Name	Formula
DQ0PTd	= @DQ0Transformation(Formula.phase1;Formula. phase2;Formula.phase3;Formula.refAng;0;1)
DQ0PTq	= @DQ0Transformation(Formula.phase1;Formula. phase2;Formula.phase3;Formula.refAng;1;1)
DQ0PTz	= @DQ0Transformation(Formula.phase1;Formula. phase2;Formula.phase3;Formula.refAng;2;1)

Name	Formula
DQ0PTd_2	@DQ0Transformation(Formula.phase1;Formula. phase2;Formula.phase3;System.Constants.Pi; 0;1)

Name	Formula
DQ0PTd_3	= @DQ0Transformation(Formula.phase1;Formula. phase2;Formula.phase3;0;0;1)

All of the above transformations use the default “wiki” *Orientation Rule*. For example, to use the “MATLAB” *Orientation Rule* for the power-invariant transformation, the following would be used:

Name	Formula
DQ0PTdm	= @DQ0Transformation(Formula.phase1;Formula. phase2;Formula.phase3; Formula.refAng;0;1;1)

See also

“@SpaceVectorTransformation” on page 228

4.17 @Energy

Function

Calculates the **energy** under curve of a waveform.

Syntax

@Energy(Waveform)

@Energy(Waveform; Begin)

@Energy(Waveform; Begin; End)

Parameters

<i>Waveform</i>	Input waveform for which the energy under curve is to be calculated.
<i>Begin</i>	Number: segment begin
<i>End</i>	Number: segment end

Output

The output is a numerical value.

Description

The energy under curve is calculated using the following formula:

$$\text{Energy} = \left[\left[\sum_{n=n_1}^{n_2} y^2(n) \right] - \frac{y^2(n_1) + y^2(n_2)}{2} \right] \cdot \Delta x$$

n_1 = first sample with $x \geq \text{Begin}$

n_2 = last sample with $x \leq \text{End}$

Δx = x - difference between two samples

The segment limits (Begin and End) are used to select a range of samples. If no segment limits are specified, the complete waveform is used. When only Begin is specified, the waveform segment from Begin to the end of the waveform is used. The numerical integration is performed assuming linear interpolation of the squared curve between the samples.

Example

The following example creates a sine wave of 50 Hz and calculates the energy in one period of the signal:

```
Signal = @SineWave(20k; 1000; 50)
E1      = @Energy(Formula.Signal; 0; 20m)
```

See also

"@Area" on page 59 and "@RMS" on page 216

4.18 @EqualTo

Function

This function performs an **equal-to** (=) evaluation on the two numerical input parameters.

Syntax

`@EqualTo(Param1; Param2)`

Parameters

<i>Param1</i>	Number: first parameter used for evaluation
<i>Param2</i>	Number: second parameter used for evaluation

Output

The output is 1 or 0

Description

The EqualTo function performs an 'equal-to' evaluation on the input parameters. If Param 1 = Param 2 then the return value will be 1 (true) else the return value will be 0 (false).

The EqualTo function is typically used in combination with the IIF function.

Example

The following example compares input parameters and provides a result that depends on the outcome:

```
EqualToExamp11 = @EqualTo(5; 5)      => 1 (true)
EqualToExamp12 = @EqualTo(12; 10)    => 0 (false)
IIFExample      = @IIF(Formula.EqualToExamp12; "TRUE";
                      "FALSE")
```

See also

"@IIF" on page 140, "@GreaterEqualThan" on page 128, "@GreaterThan" on page 129, "@LessEqualThan" on page 152 and "@LessThan" on page 153

4.19 @Exp

Function

Exponential function, the mathematical operation, written e^n , involving two parameters: the base “e” (2.7...) and the exponent “n”.

Syntax

`@Exp(Par)`

Parameters

Par Input waveform or numerical value.

Output

Waveform or numerical value containing the exponentiation (base e) of the input.

Description

The exponential function calculates the power (base e) of the input. When a waveform parameter is used, the exponential function is calculated for each individual sample. This function is the inverse function of @ln.

Example

The exponential function can also be achieved with the function @Pow. The following formulas are equivalent, assuming a formula database variable named Input:

```
Result = @Exp(Formula.Input)
```

```
Result = @Pow(System.Constants.e; Formula.Input)
```

The following example shows an alternative for the system variable System.Constants.e:

```
Euler = @Exp(1)
```

See also

“@ExpWave” on page 89, “@Ln” on page 154 and “@Pow” on page 186

4.20 @ExpWave

Function

Generates a waveform containing an **Exponential** function.

Syntax

@ExpWave(*Rate*; *Count*; *Alpha*; *Beta*)

@ExpWave(*Rate*; *Count*; *Alpha*; *Beta*; *X0*)

Parameters

<i>Rate</i>	Number: sampling frequency
<i>Count</i>	Number of samples
<i>Alpha</i>	Number: amplitude multiplier
<i>Beta</i>	Number: exponent multiplier
<i>X0</i>	Number: exponent multiplier modifier

Output

The output is a waveform containing an exponentially growing signal.

Description

This function generates a waveform using the formula:

$$f(x) = \alpha \cdot e^{(\beta \cdot (x - x_0))}$$

The sampling frequency, number of samples, rate of growth and multiplication factor can be specified. The length of a generated waveform is limited to 1 GigaSamples (1 000 000 000).

The natural logarithm $\ln(x)$ is the inverse function of the exponential function. The possibility to generate an exponential wave function can be used to synthesize a variety of waveforms. The simulated data can be used as input for other analysis functions.

Example

The following example creates an exponential waveform.

```
Signal = @ExpWave(1; 100; 2; 100m)
```

See also

"@Ln" on page 154, "@Pulse" on page 193, "@Ramp" on page 197 and
"@SquareWave" on page 232

4.21 @FallTime

Function

Determines the **fall time** of a pulse in a waveform.

Syntax

```
@FallTime(Waveform)
@FallTime(Waveform; Begin)
@FallTime(Waveform; Begin; End)
```

Parameters

<i>Waveform</i>	Input waveform containing the trailing edge of a pulse.
<i>Begin</i>	Number: segment begin
<i>End</i>	Number: segment end

Output

The output is a numerical value. This function works “while recording” when the Begin and End parameters are set. The result will be calculated when the data between Begin and End is available.

Description

The fall time is computed by taking the time difference between the distal point (90% magnitude transition) and the proximal point (10% magnitude transition) on the first trailing edge of a pulse in the waveform (or waveform segment).

The segment limits (Begin and End) are used to select a range of samples. If no segment limits are specified, the complete waveform is used. When only Begin is specified, the waveform segment from Begin to the end of the waveform is used.

Example

The following example creates a pulse and calculates the 90%-10% fall time of the trailing edge. The result is 40 ms:

```
Sig1      = @Ramp(1k; 100; 0; 0)
Sig2      = @Ramp(1k; 51; 0; 10)
Sig3      = @Ramp(1k; 100; 10; 10)
```

```
Sig4      = @Ramp(1k; 51; 10; 0)
Signal    = @Join(Formula.Sig1; Formula.Sig2;
                  Formula.Sig3; Formula.Sig4; Formula.Sig1)
Falltime  = @FallTime(Formula.Signal)
```

See also

"@PulseWidth" on page 195 and "@RiseTime" on page 214

4.22 @FilterBiquad

Function

Returns a waveform filtered using a **biquad filter**, the filter coefficients are in the Laplace or s-domain.

$$H_c(s) = \frac{b_0 + b_1s + b_2s^2}{1 + a_1s + a_2s^2}$$

Syntax

@ FilterBiquad(*Signal*; a_1 ; a_2 ; b_0 ; b_1 ; b_2 ; f_p)

Parameters

<i>Signal</i>	Input waveform
a_1	First coefficient denominator
a_2	Second coefficient denominator
b_0	Zero coefficient numerator
b_1	First coefficient numerator
b_2	Second coefficient numerator
f_p	Frequency in the analogue domain that should be mapped onto the same frequency in the digital domain (Hz), recommended is the cutoff frequency. Default 0, then no warping is used

Output

Filtered input signal

Description

This function returns the filtered signal. The used filter is defined by the following formula:

$$H_c(s) = \frac{b_0 + b_1s + b_2s^2}{1 + a_1s + a_2s^2}$$

This function gives the user the possibility to use very specific functions. Although this is a 2nd order filter, it is possible to get higher order filters by cascade this **@FilterBiquad()** function. This means make multiple lines in the formula database and use the output of a **FilterBiquad** as the input of the next **FilterBiquad** formula.

Example

In the example below the biquad coefficients are calculated for a 1st and a 2nd order low pass Butterworth function with cutoff frequency of 5 kHz.

The signal is also filtered by using the existing **@FilterButterworthLP()** function. This is done to show the correctness of the **FilterBiquad** filter.

Num	Name	Formula	Unit
1	F	5k	Hz
2	Fs	10M	Hz
3	Duration	0.001	s
4	Fc	5k	
5	Signal	@SineWave(Formula Fs; Formula Duration * Formula Fs; Formula F)	V
6	Inv2PIFc	1 / (2 * System.Constants.PI * Formula Fc)	
7		1st order	
8	a1	Formula.Inv2PIFc	
9	a2	0	
10	b0	1	
11	b1	0	
12	b2	0	
13	Fd	5k	
14			
15	Filtered_1	@FilterBiquad(Formula.Signal; Formula.a1; Formula.a2; Formula.b0; Formula.b1; Formula.b2; Formula.Fd)	
16	FilBut_1	@FilterButterworthLP(Formula.Signal; 1; Formula.Fc)	
17			
18		2nd order	
19	A1	@Sqrt(2) * Formula.Inv2PIFc	
20	A2	@Pow(Formula.Inv2PIFc; 2)	
21	B0	1	
22	B1	0	
23	B2	0	
24	Filtered_2	@FilterBiquad(Formula.Signal; Formula.A1; Formula.A2; Formula.B0; Formula.B1; Formula.B2; Formula.Fd)	
25	FilBut_2	@FilterButterworthLP(Formula.Signal; 2; Formula.Fc)	

Fig. 4.5 Example of FilterBiquad coefficients

This results in the following filtered signals Filtered_2 and FilBut_2, as expected these signals are the same (see Fig. 4.6)

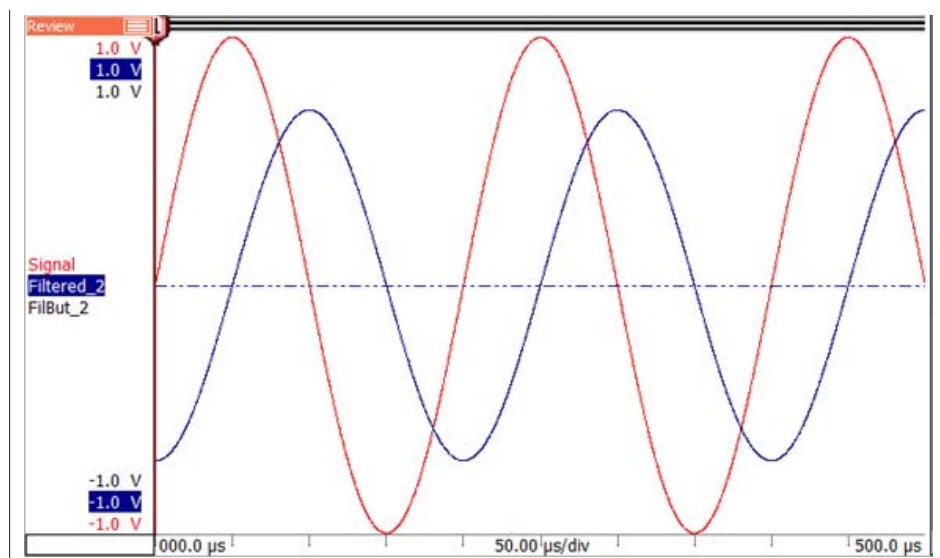


Fig. 4.6 FilterBiquad signal

See also

„@FilterButterworthLP“ on page 96

4.23 @FilterButterworthLP

Function

Filters the input signal with a direct form IIR lowpass Butterworth filter.

Syntax

`@FilterButterworthLP(Signal; Order; Fc)`

`@FilterButterworthLP(Signal; Order; Fc; Phaseless)`

Parameters

<i>Signal</i>	Input waveform
<i>Order</i>	Number: Filter order
<i>Fc</i>	Number: The cut off Frequency in Hz
<i>Phaseless</i>	Number: Filter method 0 Filter does not work phase less (default) 1 Filter works phase less

Output

The output is the filtered waveform.

Description

This function performs a direct form IIR lowpass Butterworth filtering.

The cut off frequency is the frequency at which the magnitude of the response is -3 dB.

More detailed information can be found in the "IIR Filters" on page 346

Example

```
Signal = @FilterButterworthLP(Formula.Signal; 2; 200)
```


If the *Formula.Signal* is sampled at 20 kHz than the frequency and phase response looks like:

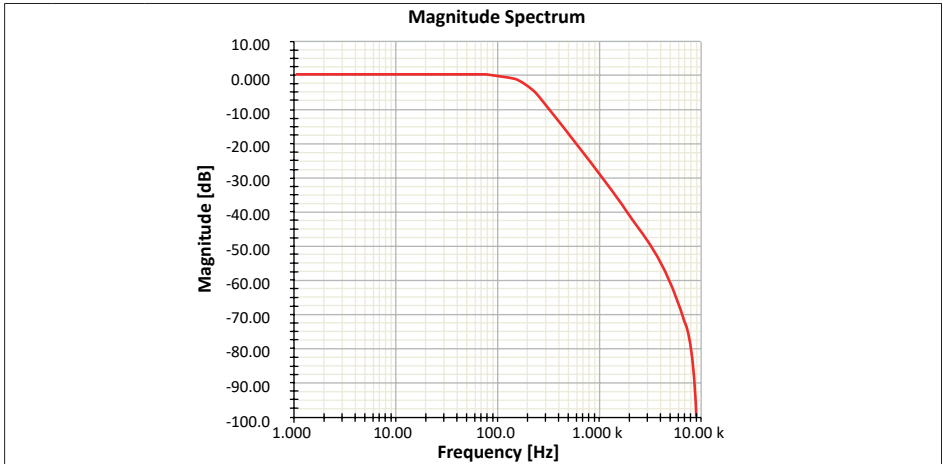


Fig. 4.7 Magnitude Spectrum - FilterButterworthLP

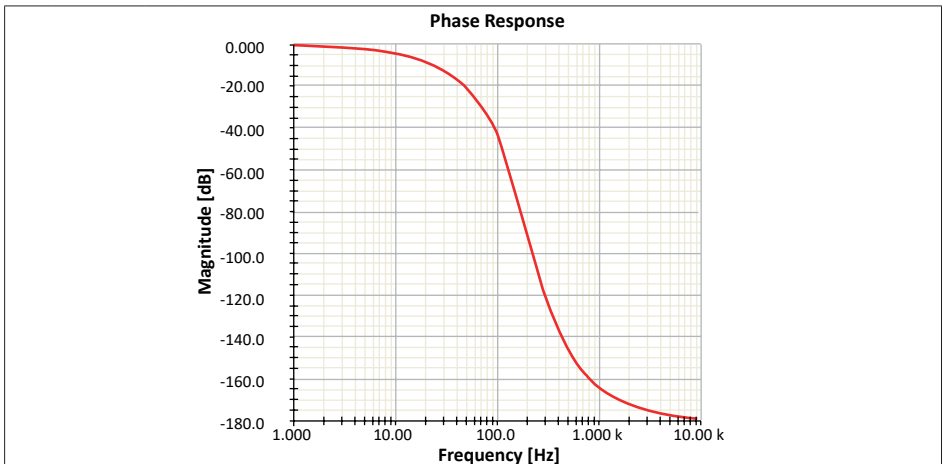


Fig. 4.8 Phase Response - FilterButterworthLP

See also

"@FilterButterworthBS" on page 102, "@FilterButterworthBP" on page 100 and
 "@FilterButterworthHP" on page 98

4.24 @FilterButterworthHP

Function

Filters the input signal with a direct form IIR highpass Butterworth filter.

Syntax

`@FilterButterworthHP(Signal; Order; Fc)`

`@FilterButterworthHP(Signal; Order; Fc; Phaseless)`

Parameters

<i>Signal</i>	Input waveform
<i>Order</i>	Number: Filter order
<i>Fc</i>	Number: The cut off Frequency in Hz
<i>Phaseless</i>	Number: Filter method 0 Filter does not work phase less (default) 1 Filter works phase less

Output

The output is the filtered waveform.

Description

This function performs a direct form IIR highpass Butterworth filtering.

The cut off frequency is the frequency at which the magnitude of the response is -3 dB.

More detailed information can be found in the "IIR Filters" on page 346

Example

```
Signal = @FilterButterworthHP(Formula.Signal; 2; 200)
```

If the *Formula.Signal* is sampled at 20 kHz than the frequency and phase response looks like:

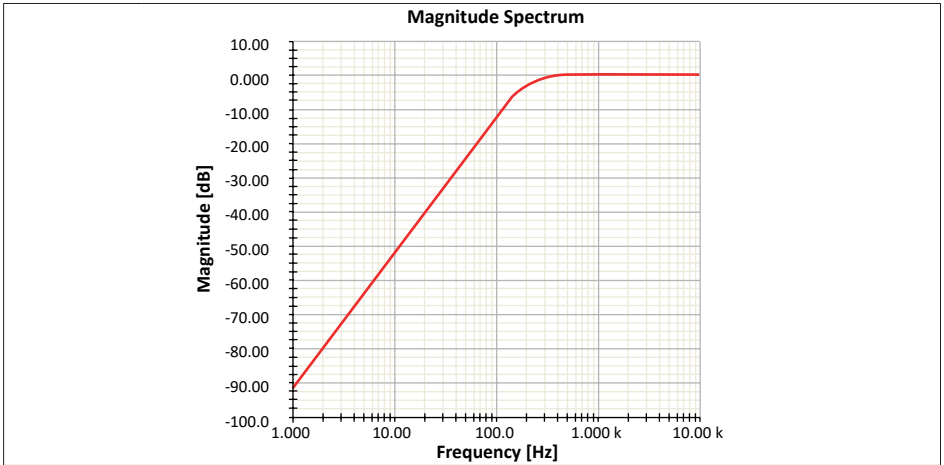


Fig. 4.9 Magnitude Spectrum - FilterButterworthHP

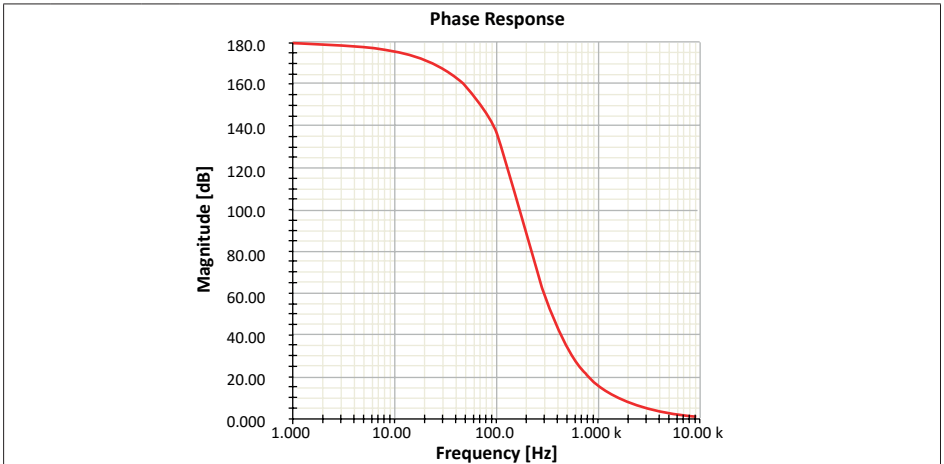


Fig. 4.10 Phase Response - FilterButterworthHP

See also

"@FilterButterworthBS" on page 102, "@FilterButterworthBP" on page 100 and "@FilterButterworthLP" on page 96

4.25 @FilterButterworthBP

Function

Filters the input signal with a direct form IIR bandpass Butterworth filter.

Syntax

`@FilterButterworthBP(Signal; Order; LowFc; UpFc)`

`@FilterButterworthBP(Signal; Order; LowFc; UpFc; Phaseless)`

Parameters

<i>Signal</i>	Input waveform
<i>Order</i>	Number: Filter order
<i>LowFc</i>	Number: The lower cut off Frequency in Hz
<i>UpFc</i>	Number: The upper cut off Frequency in Hz
<i>Phaseless</i>	Number: Filter method 0 Filter does not work phase less (default) 1 Filter works phase less

Output

The output is the filtered waveform.

Description

This function performs a direct form IIR bandpass Butterworth filtering.

The cut off frequencies are the frequencies at which the magnitude of the response is -3 dB.

More detailed information can be found in the "IIR Filters" on page 346

Example

```
Signal = @FilterButterworthBP(Formula.Signal; 2; 200; 1000)
```

If the *Formula.Signal* is sampled at 20 kHz than the frequency and phase response looks like:

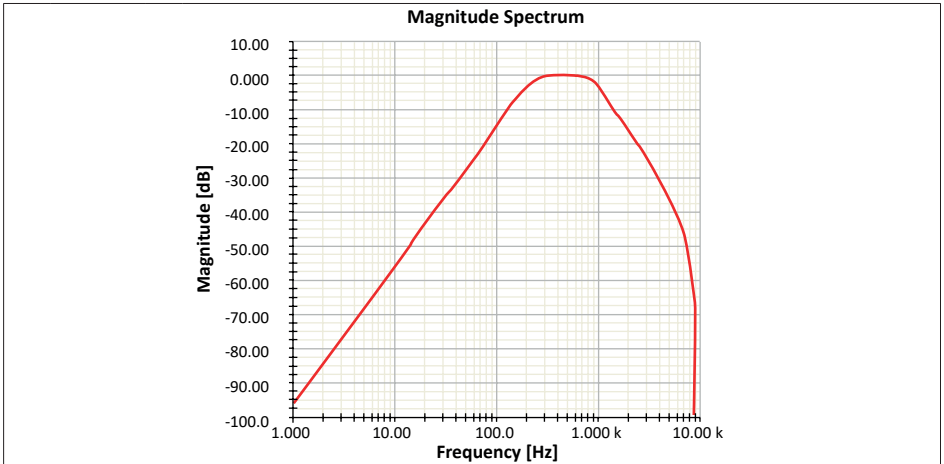


Fig. 4.11 Magnitude Spectrum - FilterButterworthBP

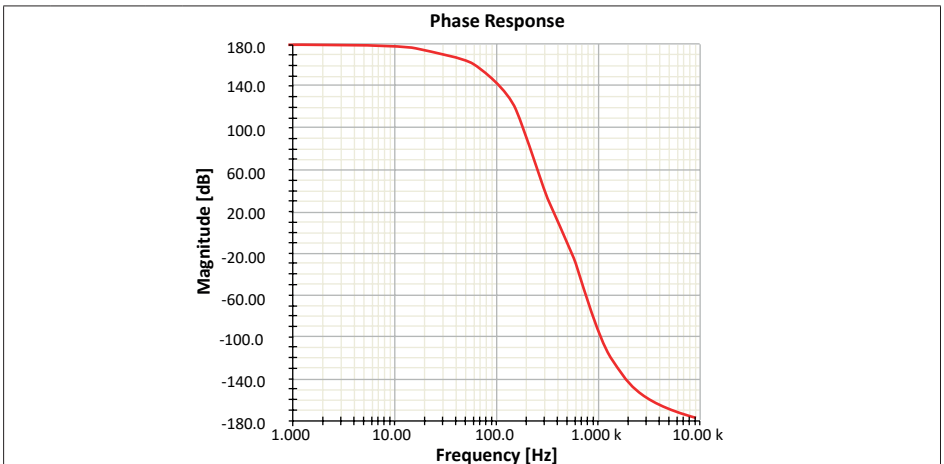


Fig. 4.12 Phase Response - FilterButterworthBP

See also

"@FilterButterworthBS" on page 102, "@FilterButterworthHP" on page 98 and "@FilterButterworthLP" on page 96

4.26 @FilterButterworthBS

Function

Filters the input signal with a direct form IIR bandstop Butterworth filter.

Syntax

@FilterButterworthBS(*Signal*; *Order*; *LowFc*; *UpFc*)

@FilterButterworthBS(*Signal*; *Order*; *LowFc*; *UpFc*; *Phaseless*)

Parameters

<i>Signal</i>	Input waveform
<i>Order</i>	Number: Filter order
<i>LowFc</i>	Number: The lower cut off Frequency in Hz
<i>UpF</i>	Number: The upper cut off Frequency in Hz
<i>Phaseless</i>	Number: Filter method 0 Filter does not work phase less (default) 1 Filter works phase less

Output

The output is the filtered waveform.

Description

This function performs a direct form IIR bandstop Butterworth filtering.

The cut off frequencies are the frequencies at which the magnitude of the response is -3 dB.

More detailed information can be found in the "IIR Filters" on page 346

Example

```
Signal = @FilterButterworthBS(Formula.Signal; 2; 200; 1000)
```

If the *Formula.Signal* is sampled at 20 kHz than the frequency and phase response looks like:

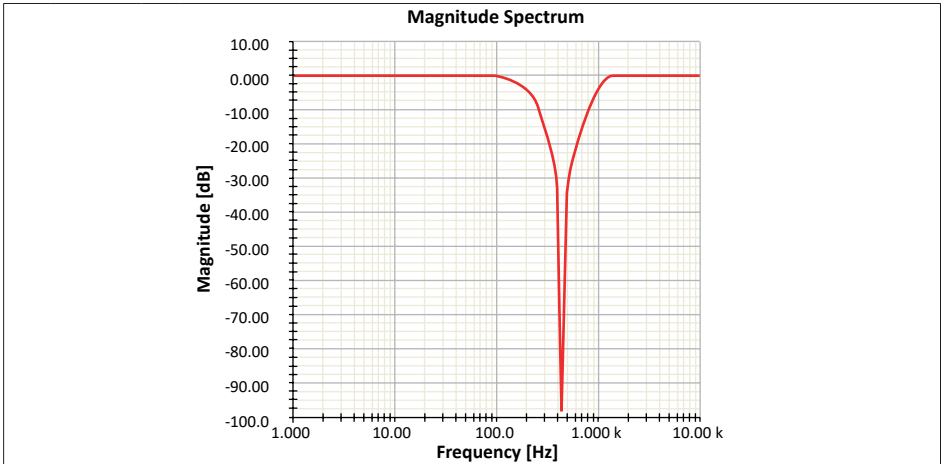


Fig. 4.13 Magnitude Spectrum - FilterBesselLP

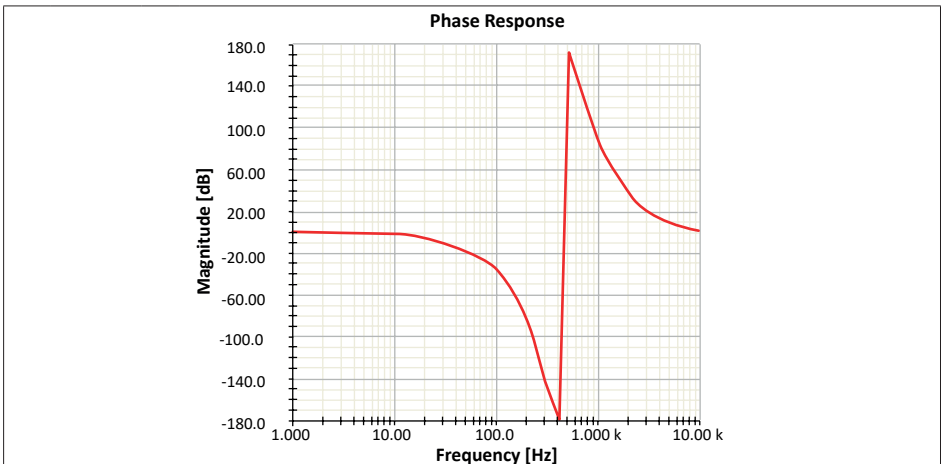


Fig. 4.14 Phase Response - FilterBesselLP

See also

"@FilterButterworthBP" on page 100, "@FilterButterworthHP" on page 98 and
 "@FilterButterworthLP" on page 96

4.27 @FilterBesselLP

Function

Filters the input signal with a direct form IIR lowpass Bessel filter.

Syntax

`@FilterBesselLP(Signal; Order; Fc)`

`@FilterBesselLP(Signal; Order; Fc; Phaseless)`

Parameters

<i>Signal</i>	Input waveform
<i>Order</i>	Number: Filter order
<i>Fc</i>	Number: The cut off Frequency in Hz
<i>Phaseless</i>	Number: Filter method 0 Filter does not work phase less (default) 1 Filter works phase less

Output

The output is the filtered waveform.

Description

This function performs a direct form IIR lowpass Bessel filtering.

The cut off frequencies are the frequencies at which the magnitude of the response is -3 dB.

More detailed information can be found in the "IIR Filters" on page 346

Example

```
Signal = @FilterBesselLP(Formula.Signal; 2; 200)
```


If the *Formula.Signal* is sampled at 20 kHz than the frequency and phase response looks like:

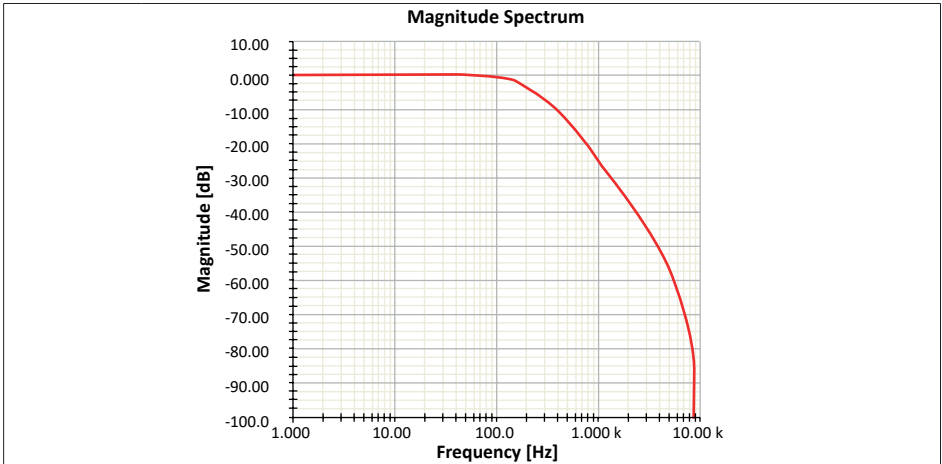


Fig. 4.15 Magnitude Spectrum - FilterBesselLP

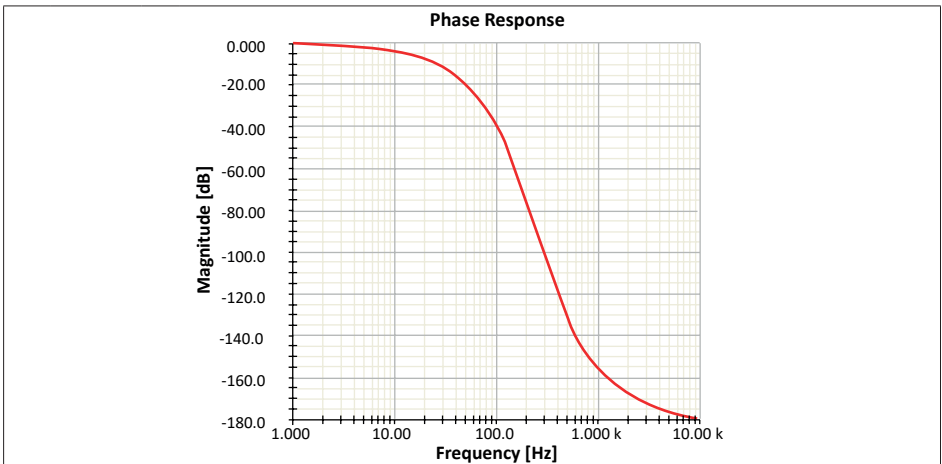


Fig. 4.16 Phase Response - FilterBesselLP

See also

"@FilterBesselBP" on page 108,"@FilterBesselBS" on page 110 and
 "@FilterBesselHP" on page 106

4.28 @FilterBesselHP

Function

Filters the input signal with a direct form IIR highpass Bessel filter.

Syntax

```
@FilterBesselHP(Signal; Order; Fc)
@FilterBesselHP(Signal; Order; Fc; Phaseless)
```

Parameters

<i>Signal</i>	Input waveform
<i>Order</i>	Number: Filter order
<i>Fc</i>	Number: The cut off Frequency in Hz
<i>Phaseless</i>	Number: Filter method 0 Filter does not work phase less (default) 1 Filter works phase less

Output

The output is the filtered waveform.

Description

This function performs a direct form IIR highpass Bessel filtering.

The cut off frequency is the frequency at which the magnitude of the response is -3 dB.

More detailed information can be found in the "IIR Filters" on page 346

Example

```
Signal = @FilterBesselHP(Formula.Signal; 2; 200)
```

If the *Formula.Signal* is sampled at 20 kHz than the frequency and phase response looks like:

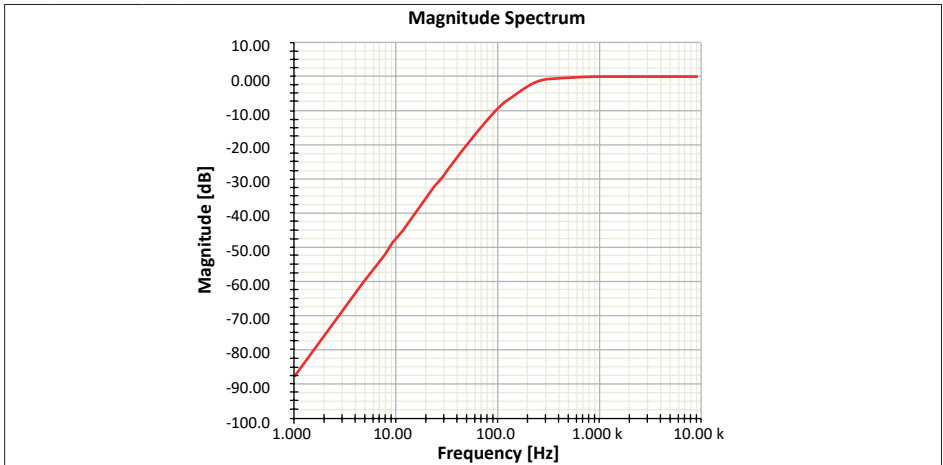


Fig. 4.17 Magnitude Spectrum - FilterBesselBP

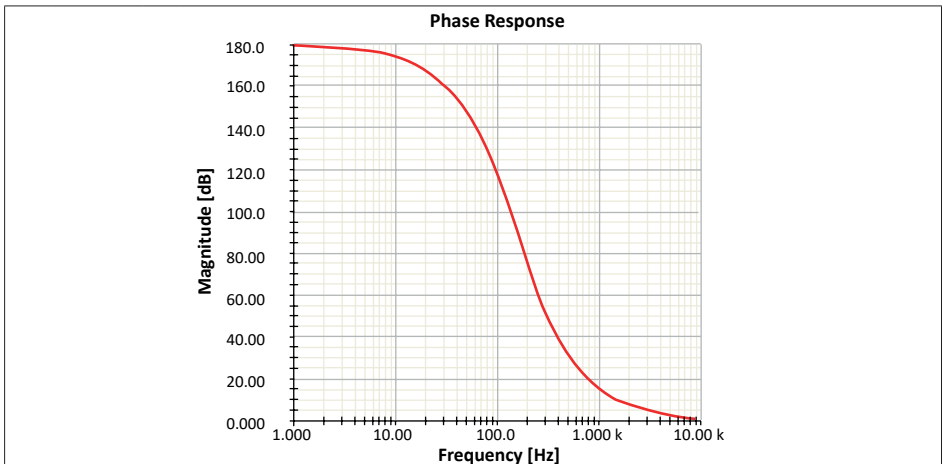


Fig. 4.18 Phase Response - FilterBesselHP

See also

"@FilterBesselBP" on page 108,"@FilterBesselBS" on page 110 and
 "@FilterBesselLP" on page 104

4.29 @FilterBesselBP

Function

Filters the input signal with a direct form IIR bandpass Bessel filter.

Syntax

`@FilterBesselBP(Signal; Order; LowFc; UpFc)`

`@FilterBesselBP(Signal; Order; LowFc; UpFc; Phaseless)`

Parameters

<i>Signal</i>	Input waveform
<i>Order</i>	Number: Filter order
<i>LowFc</i>	Number: The lower cut off Frequency in Hz
<i>UpFc</i>	Number: The upper cut off Frequency in Hz
<i>Phaseless</i>	Number: Filter method 0 Filter does not work phase less (default) 1 Filter works phase less

Output

The output is the filtered waveform.

Description

This function performs a direct form IIR bandpass Bessel filtering.

The cut off frequencies are the frequencies at which the magnitude of the response is -3 dB.

More detailed information can be found in the "IIR Filters" on page 346

Example

```
Signal = @FilterBesselBP(Formula.Signal; 2; 200; 1000)
```

If the *Formula.Signal* is sampled at 20 kHz than the frequency and phase response looks like:

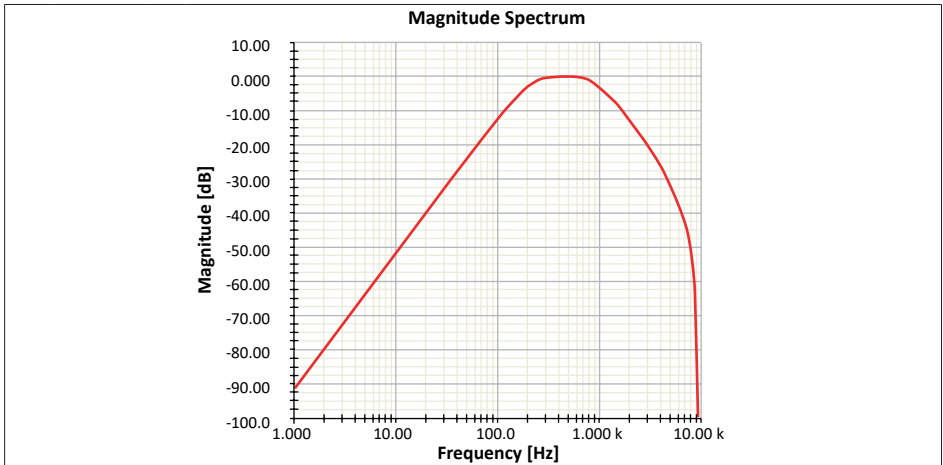


Fig. 4.19 Magnitude Spectrum - FilterBesselBP

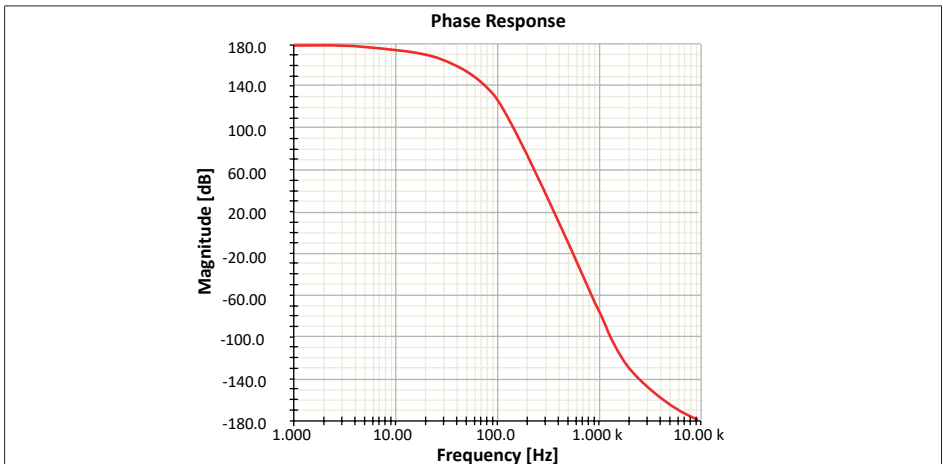


Fig. 4.20 Phase Response - FilterBesselBP

See also

"@FilterBesselBS" on page 110, "@FilterBesselHP" on page 106 and
 "@FilterBesselLP" on page 104

4.30 @FilterBesselBS

Function

Filters the input signal with a direct form IIR bandstop Bessel filter.

Syntax

`@FilterBesselBS(Signal; Order; LowFc; UpFc)`

`@FilterBesselBS(Signal; Order; LowFc; UpFc; Phaseless)`

Parameters

<i>Signal</i>	Input waveform
<i>Order</i>	Number: Filter order
<i>LowFc</i>	Number: The lower cut off Frequency in Hz
<i>UpFc</i>	Number: The upper cut off Frequency in Hz
<i>Phaseless</i>	Number: Filter method 0 Filter does not work phase less (default) 1 Filter works phase less

Output

The output is the filtered waveform.

Description

This function performs a direct form IIR bandstop Bessel filtering.

The cut off frequencies are the frequencies at which the magnitude of the response is -3 dB.

More detailed information can be found in the "IIR Filters" on page 346

Example

```
Signal = @FilterBesselBS(Formula.Signal; 2; 200; 1000)
```

If the *Formula.Signal* is sampled at 20 kHz than the frequency and phase response looks like:

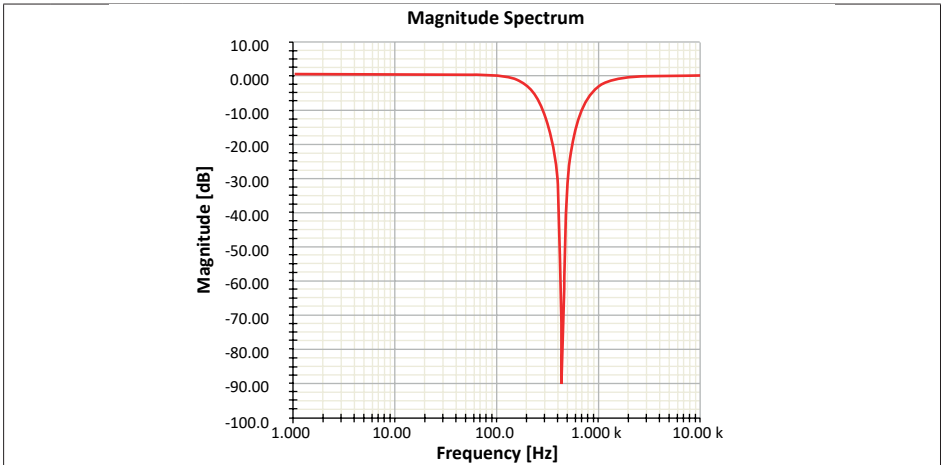


Fig. 4.21 Magnitude Spectrum - FilterBesselBS

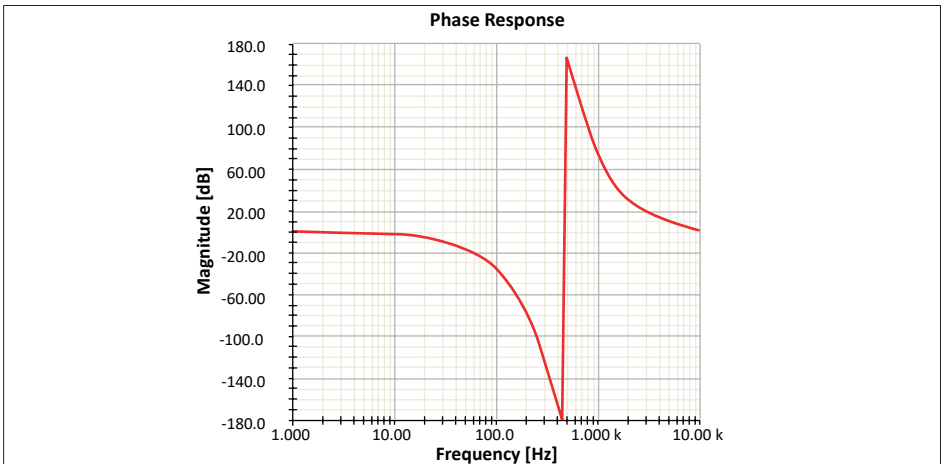


Fig. 4.22 Phase Response - FilterBesselBS

See also

"@FilterBesselBP" on page 108, "@FilterBesselHP" on page 106 and
 "@FilterBesselLP" on page 104

4.31 @FilterChebyshevLP

Function

Filters the input signal with a direct form IIR lowpass Chebyshev filter.

Syntax

```
@FilterChebyshevLP(Signal; Order; Fc)
@FilterChebyshevLP(Signal; Order; Fc; Ripple)
@FilterChebyshevLP(Signal; Order; Fc; Ripple; Phaseless)
```

Parameters

<i>Signal</i>	Input waveform
<i>Order</i>	Number: Filter order
<i>Fc</i>	Number: The cut off Frequency in Hz
<i>Ripple</i>	Number: The amplitude of the stopband ripple in decibels, the default value is 1 dB.
<i>Phaseless</i>	Number: Filter method 0 Filter does not work phase less (default) 1 Filter works phase less

Output

The output is the filtered waveform.

Description

This function performs a direct form IIR lowpass Chebyshev filtering.

The cut off frequency is not the -3 dB point as defined at the Butterworth and Bessel filters but it is the highest frequency at which the magnitude of the response is equal to the specified ripple below the maximum magnitude. If for example the ripple is defined as 2 dB and Fc is 200 Hz then the magnitude at 200 Hz will be -2 dB, see Fig. 4.23

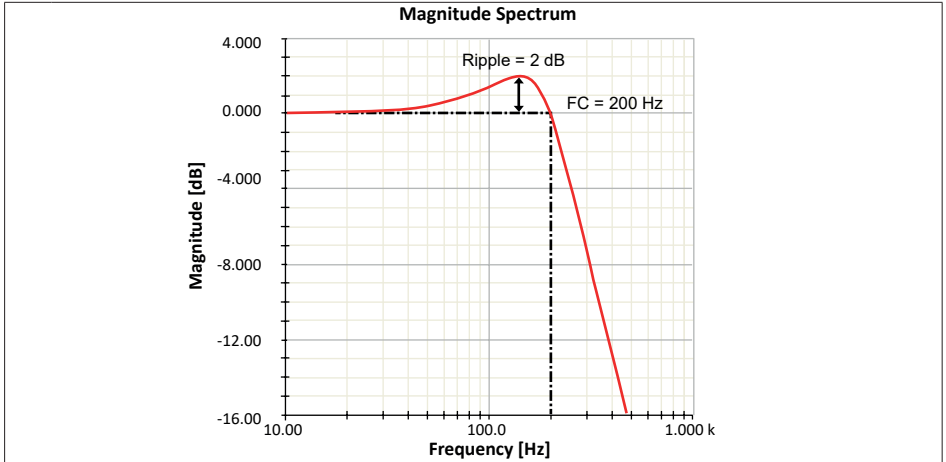


Fig. 4.23 Magnitude Spectrum - FilterChebyshevLP (Ripple)

More detailed information can be found in the "IIR Filters" on page 346

Example

```
Signal = @FilterChebyshevHP(Formula.Signal; 2; 200)
```

If the *Formula.Signal* is sampled at 20 kHz than the frequency and phase response looks like:

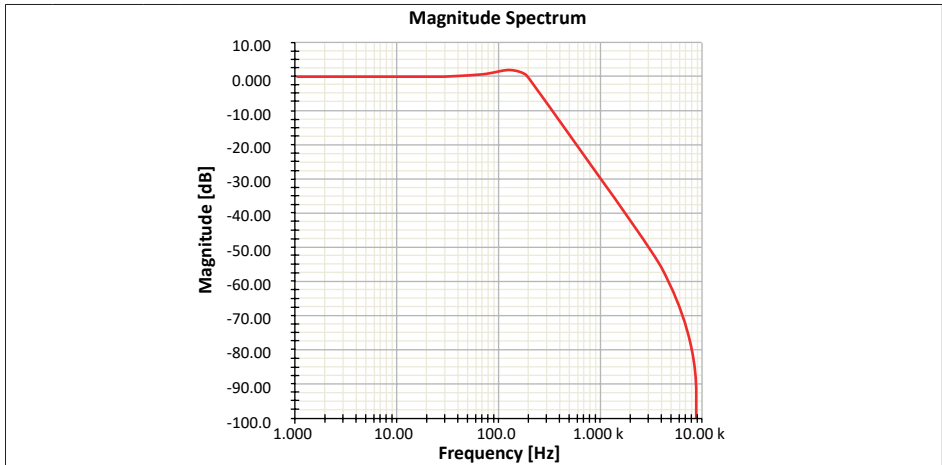


Fig. 4.24 Magnitude Spectrum - FilterChebyshevLP

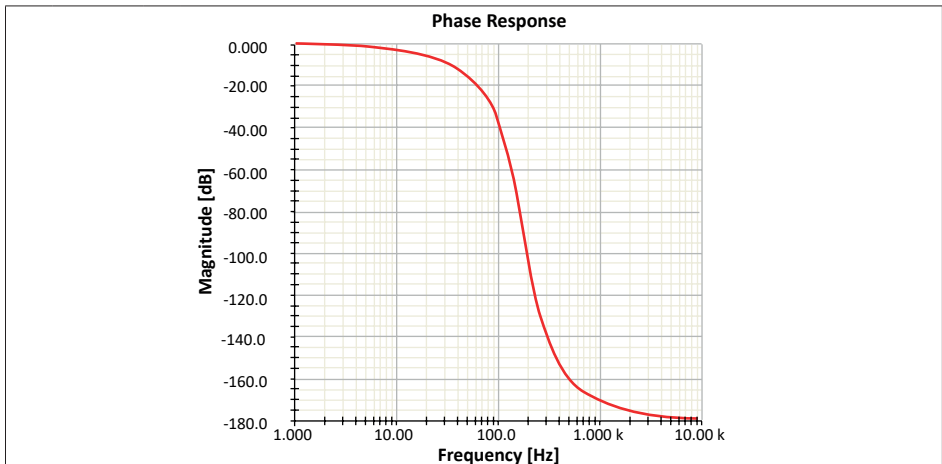


Fig. 4.25 Phase Response - FilterChebyshevLP

See also

"@FilterChebyshevBP" on page 117, "@FilterChebyshevBS" on page 119 and
 "@FilterChebyshevLP" on page 112

4.32 @FilterChebyshevHP

Function

Filters the input signal with a direct form IIR highpass Chebyshev filter.

Syntax

```
@FilterChebyshevHP(Signal; Order; Fc)
@FilterChebyshevHP(Signal; Order; Fc; Ripple)
@FilterChebyshevHP(Signal; Order; Fc; Ripple; Phaseless)
```

Parameters

<i>Signal</i>	Input waveform
<i>Order</i>	Number: Filter order
<i>Fc</i>	Number: The cut off Frequency in Hz
<i>Ripple</i>	Number: The amplitude of the stopband ripple in decibels, the default value is 1 dB.
<i>Phaseless</i>	Number: Filter method 0 Filter does not work phase less (default) 1 Filter works phase less

Output

The output is the filtered waveform.

Description

This function performs a direct form IIR highpass Chebyshev filtering.

The cut off frequency is not the -3 dB point as defined at the Butterworth and Bessel filters but it is the lowest frequency at which the magnitude of the response is equal to the specified ripple below the maximum magnitude. If for example the ripple is defined as 2 dB and Fc is 200 Hz then the magnitude at 200 Hz will be - 2dB. More detailed information can be found in the "IIR Filters" on page 346

Example

```
Signal = @FilterChebyshevHP(Formula.Signal; 2; 200)
```

If the *Formula.Signal* is sampled at 20 kHz than the frequency and phase response looks like:

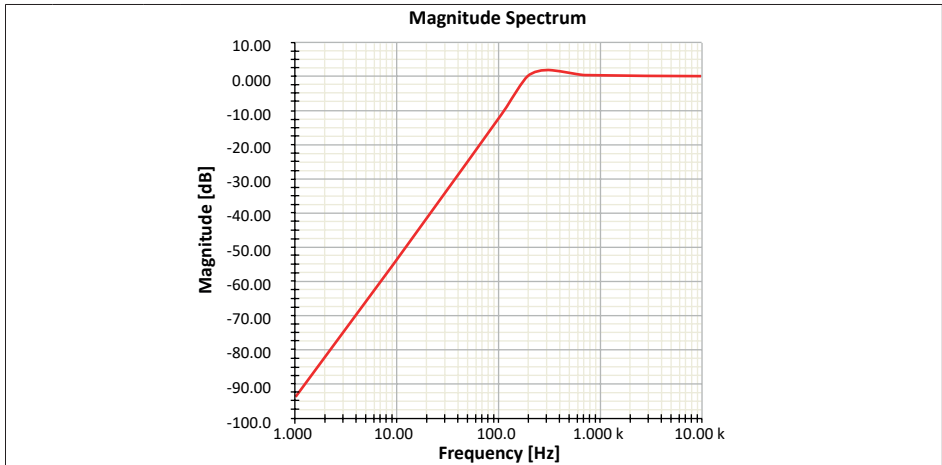


Fig. 4.26 Magnitude Spectrum - FilterChebyshev

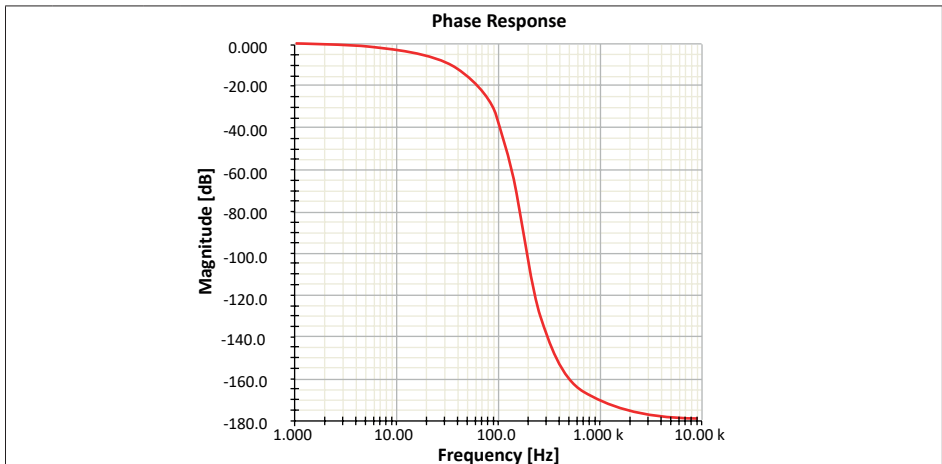


Fig. 4.27 Phase Response - FilterChebyshev

See also

"@FilterChebyshevBP" on page 117, "@FilterChebyshevBS" on page 119 and
 "@FilterChebyshevLP" on page 112

4.33 @FilterChebyshevBP

Function

Filters the input signal with a direct form IIR bandpass Chebyshev filter.

Syntax

```
@FilterChebyshevBP(Signal; Order; LowFc; UpFc)
@FilterChebyshevBP(Signal; Order; LowFc; UpFc; Ripple)
@FilterChebyshevBP(Signal; Order; LowFc; UpFc; Ripple; Phaseless)
```

Parameters

<i>Signal</i>	Input waveform
<i>Order</i>	Number: Filter order
<i>LowFc</i>	Number: The lower cut off Frequency in Hz
<i>UpFc</i>	Number: The upper cut off Frequency in Hz
<i>Ripple</i>	Number: The amplitude of the stopband ripple in decibels, the default value is 1 dB.
<i>Phaseless</i>	Number: Filter method 0 Filter does not work phase less (default) 1 Filter works phase less

Output

The output is the filtered waveform.

Description

This function performs a direct form IIR bandpass Chebyshev filtering. The cut off frequencies are not the -3d B points as defined at the Butterworth and Bessel filters but they are the lowest or highest frequencies at which the magnitude of the response is equal to the specified ripple below the maximum magnitude. If for example the ripple is defined as 2 dB and LowFc is 200 Hz than the magnitude at 200 Hz will be -2 dB.

More detailed information can be found in the "IIR Filters" on page 346

Example

```
Signal = @FilterChebyshevBP(Formula.Signal; 2; 200; 1000; 2)
```

If the *Formula.Signal* is sampled at 20 kHz than the frequency and phase response looks like:

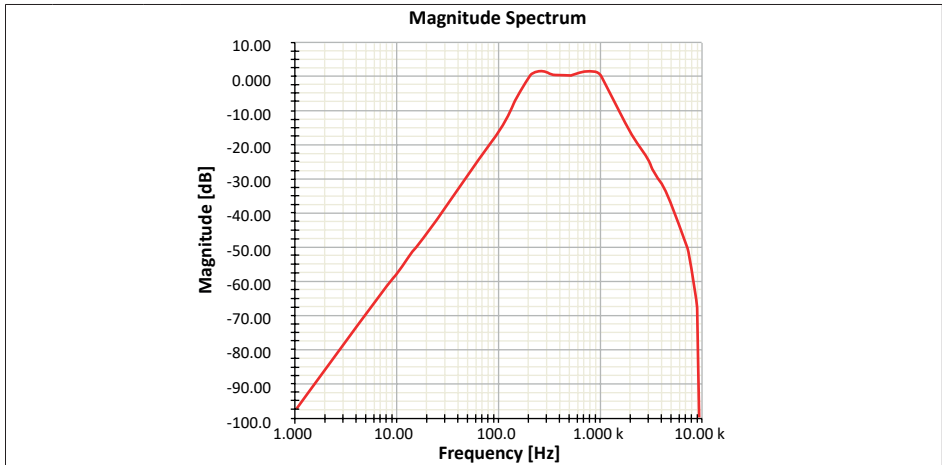


Fig. 4.28 Magnitude Spectrum - FilterChebyshevBP

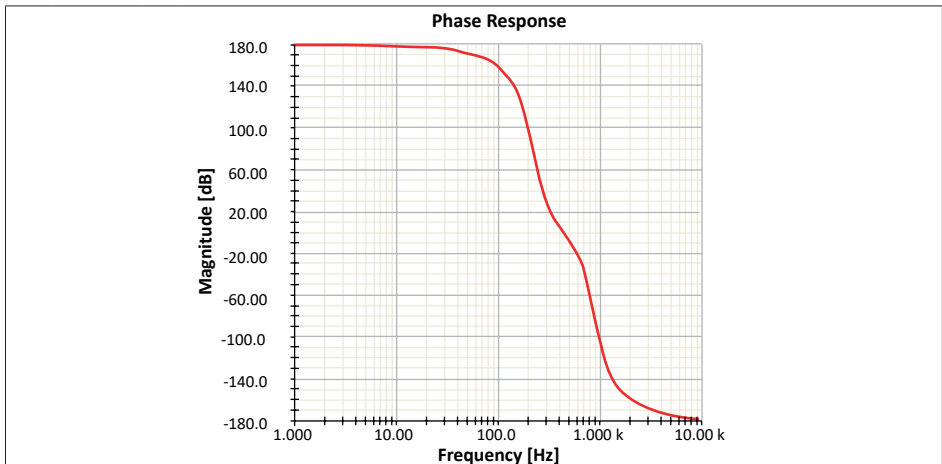


Fig. 4.29 Phase Response - FilterChebyshevBS

See also

"@FilterChebyshevBS" on page 119, "@FilterChebyshevHP" on page 115 and "@FilterChebyshevLP" on page 112

4.34 @FilterChebyshevBS

Function

Filters the input signal with a direct form IIR bandstop Chebyshev filter.

Syntax

```
@FilterChebyshevBS(Signal; Order; LowFc; UpFc)
@FilterChebyshevBS(Signal; Order; LowFc; UpFc; Ripple)
@FilterChebyshevBS(Signal; Order; LowFc; UpFc; Ripple; Phaseless)
```

Parameters

<i>Signal</i>	Input waveform
<i>Order</i>	Number: Filter order
<i>LowFc</i>	Number: The lower cut off Frequency in Hz
<i>UpFc</i>	Number: The upper cut off Frequency in Hz
<i>Ripple</i>	Number: The amplitude of the stopband ripple in decibels, the default value is 1 dB.
<i>Phaseless</i>	Number: Filter method 0 Filter does not work phase less (default) 1 Filter works phase less

Output

The output is the filtered waveform.

Description

This function performs a direct form IIR bandstop Chebyshev filtering. The cut off frequencies are not the -3 dB points as defined at the Butterworth and Bessel filters but they are the lowest or highest frequencies at which the magnitude of the response is equal to the specified ripple below the maximum magnitude. If for example the ripple is defined as 2 dB and LowFc is 200 Hz than the magnitude at 200 Hz will be -2 dB.

If for example the ripple is defined as 2 dB and Fc is 200 Hz than the magnitude at 200 Hz will be -2 dB.

More detailed information can be found in the "IIR Filters" on page 346

Example

```
Signal = @FilterChebyshevBS(Formula.Signal; 2; 200; 1000)
```

If the *Formula.Signal* is sampled at 20 kHz than the frequency and phase response looks like:

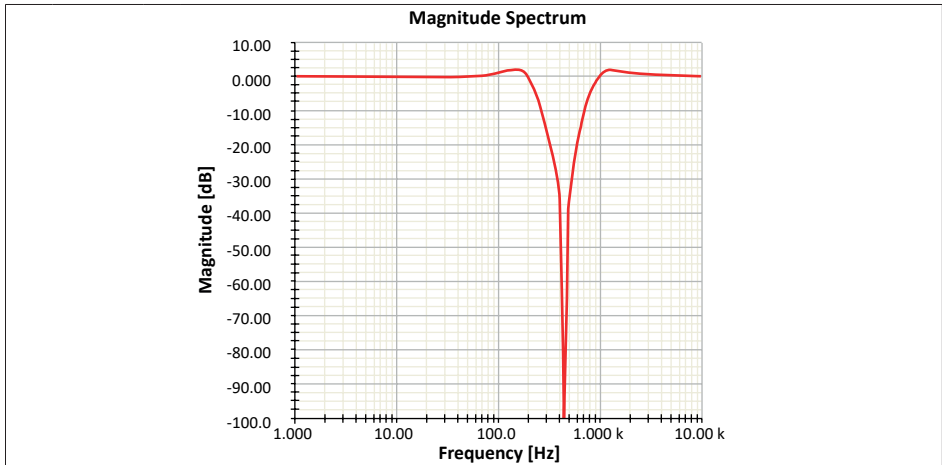


Fig. 4.30 Magnitude Spectrum - FilterChebyshevBS

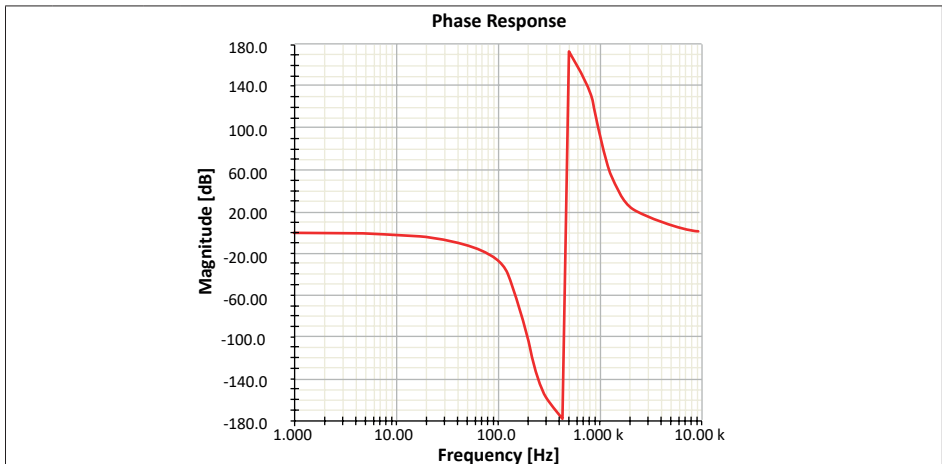


Fig. 4.31 Phase Response - FilterChebyshevBS

See also

"@FilterChebyshevBP" on page 117, "@FilterChebyshevHP" on page 115 and "@FilterChebyshevLP" on page 112

4.35 @Frequency

Function

Determines the **frequency** of a waveform.

Syntax

@Frequency(*Waveform*)

@Frequency(*Waveform*; *Begin*)

@Frequency(*Waveform*; *Begin*; *End*)

Parameters

<i>Waveform</i>	Input waveform for which the frequency is to be determined.
<i>Begin</i>	Number: segment begin
<i>End</i>	Number: segment end

Output

The output is a numerical value. This function works also “while recording” when both the *Begin* and *End* parameters are set. The result will be calculated continuously from the data that is readily available.

Description

The frequency of the waveform or waveform segment is determined. The segment limits (*Begin* and *End*) are used to select a range of samples. If no segment limits are specified, the complete waveform is used. When only *Begin* is specified, the waveform segment from *Begin* to the end of the waveform is used.

Notice

The ***Begin*** and ***End*** parameters have to be specified in the units of the horizontal axis (for example time) and not in samples.

The frequency is determined as follows:

- First the minimum and maximum values of the waveform are determined. The mean of these is considered to be the zero level and the number of zero crossings in the range *Begin* to *End* is calculated. The crossings should be in the same direction as the first one. The crossings are determined using a +/-5% hysteresis around the zero level in order to suppress the effects of noise.

- From the time difference between the first and last zero crossing in the same direction and the number of zero crossings in between, the period of the waveform is determined. Frequency is the reciprocal of the period.

Example

The following example determines the frequency of a 50 Hz signal:

```
Signal = @SineWave(20k; 1000; 50)
Freq   = @Frequency(Formula.Signal)
```

See also

"@Period" on page 184

4.36 @FormatDate

Function

Returns a **date data** source as a string, this date string can be formatted.

Syntax

@FormatDate(*Date*; *Format*)

Parameters

Date Input date data source
Format Date format string

Output

A string representing the input date in a user specified format.

Description

This function is used to format a date data source. The input data source can be the UTCDate of the RecordingInformation of an opened recording. The format string can use the following formatting codes:

FormatDate formula	
Format	Description
M	display month as a numerical (1 .. 12)
MM	display month as a numerical (01 .. 12)
MMM	display three-letter month (Apr)
MMMM	display complete name of month (April)
d	display day of the MONTH (1 .. 31)
dd	display day of the MONTH (01 .. 31)
ddd	display three-letter day of the WEEK (Tue)
dddd	display complete name of day of the WEEK (Tuesday)
yy	display two-digit year (15)
yyyy	display four-digit year (2015)

An example of some formulas:

Name	Formula
FormattedDate_1	= @FormatDate(Active.Recording Information.UTCDate;"ddd ddMMM-yyyy")
FormattedDate_2	= @FormatDate(Active.Recording Information.UTCDate;"dd-MM-yy")
FormattedDate_3	= @FormatDate(Active.Recording Information.UTCDate;"MMMM-d-yyyy")

The formula above give the following result:

Data source	Output	Format
FormattedDate_1	= Tue 20JAN-2015	"ddd ddMMMyyyy"
FormattedDate_2	= 200115	"dd-MM-yy"
FormattedDate_3	= January-20-2015	"MMMM-d-yyyy"

See also

"@FormatTime" on page 125

4.37 @FormatTime

Function

Returns a **time data** source as a string, this time string can be formatted.

Syntax

@FormatTime(*Time*; *Format*)

Parameters

Time Input time data source
Format Time format string

Output

A string representing the input time in a user specified format.

Description

This function is used to format a time data source. The input data source can be the UTCtime of the RecordingInformation of an opened recording. The format string can use the following formatting codes:

FormatTime formula	
Format	Description
HH:mm	13:47
HH:mm:ss	13:47:08
hh:mm:ss	01:47:08
h:mm:ss	1:47:08
t	1:47 PM
T	1:47:08 PM

An example of some formulas:

Name	Formula
FormattedTime_1	= @FormatTime(Active.Recording Information.UTCtime; "HH:mm:ss")
FormattedTime_2	= @FormatTime(Active.Recording Information.UTCtime; "T")

The formula above give the following result:

Data source	Output	Format
FormattedDate_1	= 10:08:37	"HH:mm:ss"
FormattedDate_2	= 10:08:37 AM	"T"

See also

"@FormatDate" on page 123

4.38 @GetNumberFromString

Function

Returns a **number from** the text value of the input **string**. This function can be used to **get a number from** a recording name: 'Recording023' => 23.

Syntax

`@GetNumberFromString(InputString; StartIndex; Length)`

Parameters

<i>InputString</i>	Input string e.g. 'Active.Recordinginformation_Title'
<i>StartIndex</i>	(Optional) The start location for the substring to be used for getting the number. The default value is 1
<i>Length</i>	(Optional) Length of the substring to be used for getting the number. The default is to use the whole string.

Output

A number extracted from the input string.

Description

This function returns a number which is contained by the input string.
This function can be used in combination with the user-key script actions: "Post User Table column to Excel" and "Post to Excel".
For more information read the help coming with these user-key actions in Perception.

Example

`@GetNumberFromString("Recording003")` results in 3
`@GetNumberFromString("Recording3025"; 11)` results in 25
`@GetNumberFromString("Recording3025"; 10; 3)` results in 302

4.39 @GreaterEqualThan

Function

This function performs a **greater-than-or-equal-to** (\geq) evaluation on the two numerical input parameters.

Syntax

`@GreaterEqualThan(Param1; Param2)`

Parameters

<i>Param1</i>	Number: first parameter used for evaluation.
<i>Param2</i>	Number: second parameter used for evaluation.

Output

The output is 1 or 0.

Description

The GreaterEqualThan function performs a “greater-than-or-equal-to” evaluation on the input parameters.

If $\text{Param 1} \geq \text{Param 2}$ then the return value will be 1 (= true) else the return value will be 0 (=false).

The **GreaterEqualThan** function is typically used in combination with the IIF function.

Example

The following example compares input parameters and provides a result that depends on the outcome:

```
GETExam1    = @GreaterEqualThan(5; 5)    => 0 (false)
GETExam2    = @GreaterEqualThan(12; 10) => 1 (true)
IIFExample  = @IIF(Formula.GETExam2; "TRUE"; "FALSE")
```

See also

"@EqualTo" on page 87, "@GreaterThan" on page 129, "@IIF" on page 140, "@LessEqualThan" on page 152 and "@LessThan" on page 153

4.40 @GreaterThan

Function

This function performs a **greater-than** (>) evaluation on the two numerical input parameters.

Syntax

@GreaterThan(*Param1*; *Param2*)

Parameters

<i>Param1</i>	Number: first parameter used for evaluation.
<i>Param2</i>	Number: second parameter used for evaluation.

Output

The output is 1 or 0.

Description

The GreaterThan function performs a 'greater-than' evaluation on the input parameters.

If Param 1 > Param 2 then the return value will be 1 (true) else the return value will be 0 (false).

The **GreaterThan** function is typically used in combination with the IIF function.

Example

The following example compares input parameters and provides a result that depends on the outcome:

```
GTEExam1      = @GreaterThan(5; 100)    => 0 (false)
GTEExam2      = @GreaterThan(12; 10)    => 1 (true)
IIFExample    = @IIF(Formula.GTEExam2; "TRUE"; "FALSE")
```

See also

"@EqualTo" on page 87, "@GreaterEqualThan" on page 128, "@IIF" on page 140, "@LessEqualThan" on page 152 and "@LessThan" on page 153

4.41 @Histogram

Function

Calculates amplitude **histogram**.

Syntax

@Histogram(*Waveform*; *YLow*; *YHigh*; *Number*)

Parameters

<i>Waveform</i>	Input waveform from which the histogram is to be calculated.
<i>YLow</i>	Number: lowest amplitude to include in the histogram.
<i>YHigh</i>	Number: highest amplitude to include in the histogram.
<i>Number</i>	Number of divisions

Output

Waveform containing the amplitude histogram.

Description

The histogram function determines the number of occurrences of amplitude values (values of a waveform) in each amplitude division. The amplitude divisions are specified by dividing the range between YLow and YHigh into Number divisions. The width of each division is $(YHigh - YLow) / Number$. Each value of the waveform is sorted to one of these divisions. The histogram shows how many values were sorted in each division. The maximum allowed number of divisions is 4096. Typically values between 100 and 1000 are used.

The parameter YLow and YHigh determine the range of amplitudes included in the histogram. Any value outside this range is discarded. To ensure that out-of-range values are included in a division of the histogram, use the @Clip function to limit the amplitude values before calculating the histogram.

Example

The following example calculates an amplitude histogram between -5.5 V and +5.5 V using 11 divisions. This means each division has a width of 1 V.

```
Signal = 5 * @SineWave(5000; 1000; 10)
Histo  = @Histogram(Formula.Signal; -5.5; 5.5; 11)
```

The following example calculates a histogram of 100 divisions between the minimum and maximum of the waveform, assuming a formula database variable `Signal`:

```
MinSignal = @Min(Formula.Signal)
MaxSignal = @Max(Formula.Signal)
Histo     = @Histogram(Formula.Signal;
                        Formula.MinSignal;
                        Formula.MaxSignal; 100)
```

See also

"@Clip" on page 68, "@Max" on page 157 and "@Min" on page 167

4.42 @HSFundamentalFrequency

Function

Returns a waveform containing the **fundamental frequency** over time extracted from a spectral input signal.

Syntax

@HSFundamentalFrequency(*Spectrum*; *Start time*; *End time*)

Parameters

<i>Spectrum</i>	Input signal containing the spectral information to be processed
<i>Start time</i>	(Optional) The time from which the analysis should start
<i>End time</i>	(Optional) The time at which the analysis should end

Output

A waveform containing the fundamental frequency over time.

Description

The input for this function is a Spectrum. This spectrum can be created e.g. using the real-time formula @HarmonicsIEC6100(). The recorded signal then creates a spectrum that can be accessible via the data-source name 'Active.RTFormulas.Results.I_ha).

The output is a value of the fundamental frequency over time found in this spectrum. The y-units is Hz.

For more information see also the function @HSOneHarmonicAmplitude()

Example

An example of the usage of the @HSFundamentalFrequency() function:

```
FundamentalFreq = @HSFundamentalFrequency(Active.RTFormulas.  
Results.I_ha)
```

See also

"@HSOneHarmonicAmplitude" on page 133 and

"@HSOneHarmonicPhase" on page 138

4.43 @HSOneHarmonicAmplitude

Function

Returns a waveform containing the **amplitude** of a single **harmonic** over time extracted from a spectral input signal.

Syntax

@HSOneHarmonicAmplitude(*Spectrum*; *Order*; *Grouping*; *Start time*; *End time*)

Parameters

<i>Spectrum</i>	Input signal containing the spectral information to be processed
<i>Order</i>	The order of the requested harmonic (a non-negative integer)
<i>Grouping</i>	(Optional) Indicates if grouping should be done to generate the amplitude: 0-no grouping, 1-sideband grouping, 2-full grouping. Default is no grouping.
<i>Start time</i>	(Optional) The time from which the analysis should start
<i>End time</i>	(Optional) The time at which the analysis should end

Output

A waveform containing the amplitude of a single harmonic over time.

Description

The input for this function is a Spectrum. This spectrum can be created e.g. using the real-time formula @HarmonicsIEC6100(). The recorded signal then creates a spectrum that can be accessible via the data-source name 'Active.RTFormulas.Results.l_ha)

The output is a amplitude of the selected harmonic over time found in this spectrum. The y-units is the same as the units from the recorded signal. E.g 'A' or 'Volt'. The grouping works like the grouping used in the Perception Harmonic Display. Fig. 4.32 shows sideband and full grouping for the 2nd and 4th harmonics respectively.

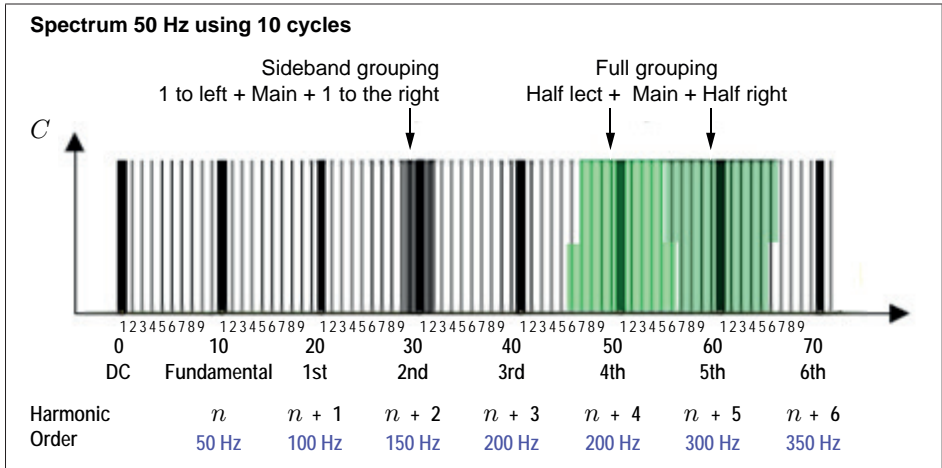


Fig. 4.32 Sideband and Full grouping (2nd and 4th Harmonic grouping)

Result 2nd Harmonic with sideband grouping

$$\sqrt{C_{29}^2 + C_{30}^2 + C_{31}^2}$$

Result 4th Harmonic with full grouping

$$\sqrt{(0.5 * C_{45}^2 + C_{46}^2 + C_{47}^2 + C_{48}^2 + C_{49}^2 + C_{50}^2 + C_{51}^2 + C_{52}^2 + C_{53}^2 + C_{45}^2 + C_{47}^2 + C_{47}^2 + 0.5 * C_{55}^2)}$$

4 REFERENCE GUIDE

Example

Fig. 4.33 shows an example on how the Harmonic spectra functions can be used. The following real-time formulas can be used to create your own signal containing harmonic information.

Real-time Data Formula Database (RT-FDB)	Name	Expression	Unit	Result type	Storage	Color
GEN3.Jos						
1	i	Recorder_A.i	A	SyncAnalog	✓	
2		*****	(-)	(-)	(-)	(-)
3		*****	(-)	(-)	(-)	(-)
4		*****	(-)	(-)	(-)	(-)
5	Cycle_source	Recorder_A.i	A	SyncAnalog	✗	
6	Cycle_count	1		Scalar	✗	
7	Cycle_level	0	A	Scalar	✗	
8	Cycle_hyst_mode	1		Scalar	✗	
9	Cycle_hyst	1	%	Scalar	✗	
10	Cycle_holdoff	0.001	s	Scalar	✗	
11	Cycle_filter_type	1		Scalar	✗	
12	Cycle_cutoff_frequency	5000	Hz	Scalar	✗	
13	Cycle_direction	0		Scalar	✗	
14	Cycle_timeout	1	s	Scalar	✗	
15	Cycle_source_filt	@HWFilt (RTFormulas.Cycle_source ; RTFormulas.Cycle_filter_type ; RTFormulas.Cycle_cutoff_fre	A	mcFilteredAnalog	✗	
16		End of cycle parameters	(-)	(-)	(-)	(-)
17		*****	(-)	(-)	(-)	(-)
18		*****	(-)	(-)	(-)	(-)
19		*****	(-)	(-)	(-)	(-)
20		*****	(-)	(-)	(-)	(-)
21	Cycle_Master	@CycleDetect (RTFormulas.Cycle_source_filt ; RTFormulas.Cycle_count ; RTFormulas.Cycle_level ;		Cycles	✓	
22		END of Computing the CYCLE MASTER	(-)	(-)	(-)	(-)
23	I_ha	@HarmonicsIEC61000 (RTFormulas ; RTFormulas.Cycle_Master ; 50)		Spectrum	✓	
24			(-)	(-)	(-)	(-)

Fig. 4.33 Example of Real-time formulas - Harmonic spectrum functions

The input data-source **Active.RTFormulas.Results.I_ha** contains the spectral information of the recorded signal **Recorder_A.i** (see Fig. 4.34).

Num	Name	Formula	Unit
1		Returns the frequency of the fundamental in Hz	
2	FundamentalFreq	@HSFundamentalFrequency(Active.RTFormulas.Results.I_ha)	Hz
3		Returns the amplitude of the fundamental (= 1st harmonic)	
4	FundamentalAmp	@HSOneHarmonicAmplitude(Active.RTFormulas.Results.I_ha; 1)	A
5		Returns the phase of the fundamental (= 1st harmonic)	
6	FundamentalPhase	@HSOneHarmonicPhase(Active.RTFormulas.Results.I_ha; 1)	

Fig. 4.34 Example of spectral information from signal

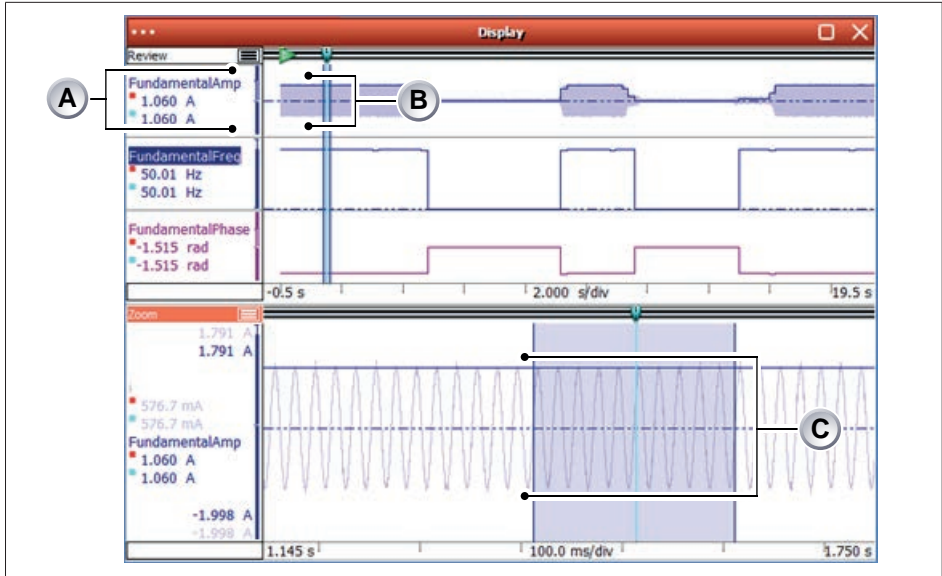


Fig. 4.35 Harmonic display - Spectral functions

- A FundamentalAmp
- B Cursor/Trace of sinusoidal input signal
- C Zoom area of the sinusoidal input signal

The calculated signals from the various HS functions is displayed in Fig. 4.35. The blue cursor (B) is the original sinusoidal input signal. In the bottom area the full trace is zoomed (C).

The value at the cursor of the calculated FundamentalAmp (A) signal is 1.060 A

This is the same value as shown in the Harmonic Display as the Amplitude of the 1st order spectra (see Fig. 4.36)

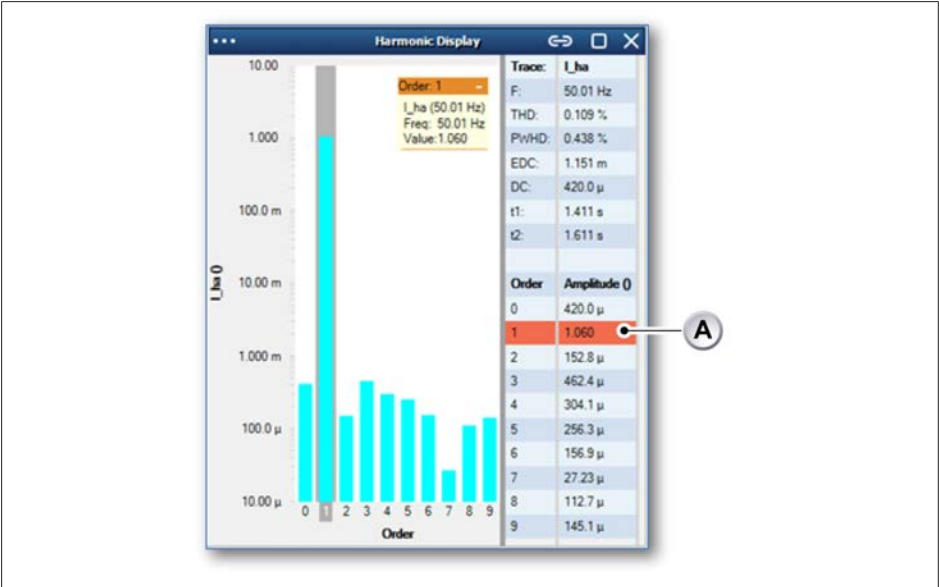


Fig. 4.36 Harmonic Display - Trace/Order values

A Amplitude value

See also

"@HSFundamentalFrequency" on page 132 and "@HSOneHarmonicPhase" on page 138

4.44 @HSOneHarmonicPhase

Function

Returns a waveform containing the **phase** of a single **harmonic** over time extracted from a spectral input signal.

Syntax

@HSOneHarmonicPhase(*Spectrum; Order; Grouping; Start time; End time*)

Parameters

<i>Spectrum</i>	Input signal containing the spectral information to be processed
<i>Order</i>	The order of the requested harmonic (a non-negative integer)
<i>Grouping</i>	(Optional) Indicates if grouping should be done to generate the amplitude: 0-no grouping, 1-sideband grouping, 2-full grouping. Default is no grouping.
<i>Start time</i>	(Optional) The time from which the analysis should start
<i>End time</i>	(Optional) The time at which the analysis should end

Output

A waveform containing the phase of a single harmonic over time over time.

Description

The input for this function is a Spectrum. This spectrum can be created e.g. using the real-time formula @HarmonicsIEC6100(). The recorded signal then creates a spectrum that can be accessible via the data-source name 'Active.RTFormulas.Results.l_ha)

The output is a value of the phase over time found in this spectrum. The y-units is rad.

For more information see also the function @HSOneHarmonicAmplitude()

Example

An example of the usage of the @HSOneHarmonicPhase() function:

4 REFERENCE GUIDE

```
FundamentalPhase = @HSONeHarmonicPhase(Active.RTFormulas.  
Results.I_ha; 1)
```

See also

"@HSONeHarmonicAmplitude" on page 133 and "@HSFundamentalFrequency" on page 132

4.45 @IIF

Function

This function performs an Immediate If (IIF) and returns one value if a condition evaluates to TRUE and another value if it evaluates to FALSE.

Syntax

@IIF(*Param*; *IfTrue*; *IfFalse*)

Parameters

- Param* Waveform/Number/String to be evaluated.
- IfTrue* Waveform/Number/String output if the evaluation of *Param* returns true.
- IfFalse* Waveform/Number/String output if the evaluation of *Param* returns false.

Output

The output can be a waveform, number or string.

Description

The IIF function performs an evaluation on the first input parameter; the sort of evaluation depends on the type of this parameter and is described in the table below. The evaluation will be done and the result is either true or false. If the result is true then the second parameter will be returned as output of the IIF function else the third parameter will be returned.

The following table describes the evaluation of the first parameter:

IIF formula functions		
Parameter type	True	False
Numeric	Not 0	Equal to 0
String	Not empty	Empty
Waveform	Not Null	Null

Example

The following example uses a generated sine wave. If this signal is shown in the display with name Display then the IIF function will return a string "Above 0.5" if the waveform value at the active cursor position is above 0.5 else the string "Below 0.5" will be returned.

The GreaterThan function is used to compare the waveform value at the active cursor position against the 0.5 value. The result is 1 (true) if the value is greater than 0.5 else the result is 0 (false). This result is the first parameter of the IIF function.

```
Signal          = @SineWave(8000; 8001; 5)
GreaterHalf     = @GreaterThan
                  (Display.Display.ActiveCursor.YValue; 0.5)
IIFExample      = @IIF(Formula.GreaterHalf; "Above 0.5";
                      "Below 0.5")
```

See also

"@EqualTo" on page 87, "@GreaterEqualThan" on page 128, "@GreaterThan" on page 129, "@LessEqualThan" on page 152 and "@LessThan" on page 153

4.46 @Integrate

Function

Integrates a waveform.

Syntax

@Integrate(*Waveform*)

Parameters

Waveform Waveform to be integrated.

Output

Integrated waveform

Description

Integration calculates the running sum of all sample values:

$$\begin{aligned}i(1) &= y(1) \\ i(n) &= i(n-1) + y(n) \cdot \Delta x \text{ for } n = 2, \dots, N\end{aligned}$$

The integral calculates the area under curve of the waveform from the starting point up to the end point.

Example

The following example integrates the instantaneous power to obtain energy as a function of time:

```
Signal = @SineWave(10k; 1000; 50)
Power  = (Formula.Signal * Formula.Signal) / 600
Energy = @Integrate(Formula.Power)
```

See also

"@Diff" on page 79 and "@Energy" on page 85

4.47 @IntLookUp

Function

Uses the data from a waveform as an index (pointer) into a Conversion Look Up Table (CLUT). The CLUT is stored in a separate file.

Syntax

@IntLookUp(*Waveform*; *CLUT*)

@IntLookUp(*Waveform*; *CLUT*; *IndexOffset*)

Parameters

<i>Waveform</i>	Input waveform
<i>CLUT</i>	String: the full path giving the name and the unique location of the CLUT-file.
<i>IndexOffset</i>	Number: offset to the index pointer.

Output

Converted waveform

Description

This function uses the data from a waveform as an index to a Conversion Look Up Table. The data can be 16-bit integer.

The Conversion Look Up Table (CLUT) is stored in an ASCII (text) file.

Each line in this file contains a floating-point number. The line number is used as the index for the CLUT.

The first line is associated by default with index 0. If the function however uses the optional 'IndexOffset' parameter then the first line is associated with – IndexOffset. This allows for working with negative values.

When using 16-bit integer data, the CLUT needs to contain $2^{16} = 65536$ points. When the data is signed (positive and negative) use an offset of 32768. Now the first point in the CLUT is associated with –32768.

Example

Assume the following look-up table:

Look-up functions	
Line Number	Value
1	11.3
2	21.4
3	31.5
4	41.4
5	51.2
6	61.3
7	71.5
8	81.6
9	91.2
10	101.4
11	111.2

Assume the original waveform to be: 6, 2, 7, 8

The first value 6 is associated with line 7 (index is zero-based). The corresponding value from the CLUT table is 71.5

The converted output waveform looks like:

71.5, 31.5, 81.6, 91.2

If the optional "IndexOffset" parameter is used and set to 2:

The first value 6 is associated with line 5 (index 6 – 2, zero-based). The corresponding value from CLUT table is 51.2

The converted output waveform looks like:

51.2, 11.3, 61.3, 71.5

If the input waveform contains an index greater than the maximum index stored in the CLUT then the last value of the CLUT is used. If the input waveform contains an index smaller than the minimum index stored in the CLUT then the first value of the CLUT is used.

To use the external lookup table "Lookup.asc" located in the directory "C:\CalibData":

```
Signal = @Ramp(1000; 2001; 0; 4096; 11)
Calib  = @IntLookUp
        (Formula.Signal; "C:\CalibData\Lookup.asc")
```

See also

"@IntLookUp12" on page 146

4.48 @IntLookUp12

Function

Uses the data from a waveform as an index (pointer) into a Conversion Look Up Table (CLUT).

Syntax

@IntLookUp12(*Waveform*; *Clut*)

Parameters

<i>Waveform</i>	Waveform to be converted.
<i>Clut</i>	Conversion Look Up Table

Output

Converted waveform

Description

This function uses the data from a waveform as an index (pointer) to a Conversion Look Up Table.

Notice

This function is optimized for 12-bit integer data.

The Conversion Look Up Table has 4096 values.

Each value of the original waveform is used as a pointer into the CLUT. The corresponding value in the CLUT is used to produce the output.

The CLUT can be created:

- using a high level programming language and the Perception CSI option
- with a (combination of) formula database function(s)

The following diagram shows an example of the use of this function.

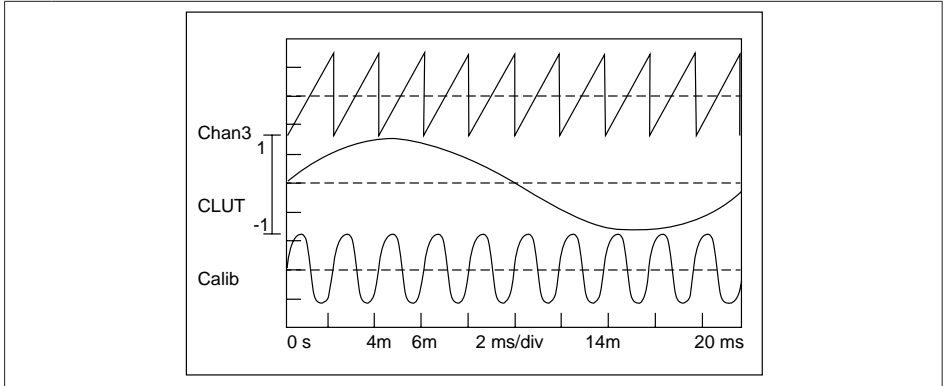


Fig. 4.37 Diagram - @IntLookUp12 Function

- The top trace shows the original recording.
- The middle trace shows the contents of the CLUT.
- The bottom trace shows the converted data.

To create this diagram the following formulas are used:

```
CLUT      = @SineWave(200k; 4096; 48.8)
Org       = @Ramp(2M; 40960; 4095; 10)
Calib     = @IntLookUp12(Formula.Org; Formula.CLUT)
```

First a sine wave is generated of 4096 points with a frequency of 48.8 Hz. With a sample rate of 200 kHz one cycle fits in the waveform.

The result named “Calib” is produced by taking the generated waveform Formula.Org and using this waveform as an index for the CLUT.

Notice

The original data should be integer data (2 bytes) in range -32767 to + 32768 with 12-bit resolution. Before the data is used, the integer value is divided by 16 to obtain the correct index. Therefore also 14- or 16-bit data can be used, but this will result in a loss of the least significant bits and may produce unpredictable or unexpected results.

See also

“@IntLookUp” on page 143

4.49 @IsNaN

Function

This function determines if the input parameter is Not a Number (NaN).

Syntax

@IsNaN(*Param*)

Parameters

Param Number: parameter to be validated.

Output

The Output is 1 or 0.

Description

When the input parameter is Not a Number (NaN) then the return value will be 1 (true) else the return value will be 0 (false).

Example

The following example will return the value 1 (true):

```
isValid = @IsNaN(10.0/0.0)
```

See also

"@GreaterThan" on page 129, "@LessThan" on page 153, "@EqualTo" on page 87, "@GreaterEqualThan" on page 128 and "@LessEqualThan" on page 152

4.50 @Join

Function

Joins two or more waveforms by concatenation.

Syntax

`@Join(Waveform1; ...; WaveformN)`

Parameters

<i>Waveform1</i>	First waveform
<i>WaveformN</i>	Last waveform, with $N \geq 2$

Output

Concatenated waveform

Description

This function concatenates two or more waveforms to form a new waveform. The waveforms are joined by concatenation of the end of a waveform with the begin of the next waveform. The horizontal axis is determined by the first waveform. The x-coordinate of the first sample of the output waveform is equal to the x-coordinate of the first sample of the first waveform. The samples from the other waveform(s) are simply concatenated to the samples of the first waveform. The length of the output waveform is the sum of the lengths of the waveforms specified as parameters.

When the input waveforms have different time bases, the output waveform will also contain multiple time bases.

Example

This function is especially useful for generating simulated signals as shown in the following example:

```
Seg1    = @Ramp(1k; 100; 0; 0)
Seg2    = @Ramp(1k; 50; 0; 1)
Seg3    = @Ramp(1k; 100; 1; 1)
Signal  = @Join(Formula.Seg1; Formula.Seg2; Formula.Seg3)
```

The Join function takes the first valid data point of a waveform and concatenates that to the previous waveform. This means that the original end and start time of the two concatenated waveforms may differ. However, in the output waveform they will be one (1) sample interval apart.

The following example illustrates this:

```
Seg1    = @Ramp(1k; 100; 0; 0)
Seg2    = @Ramp(1k; 100; 0; 0)
```

Shift second segment by 100 horizontal units using @XShift:

```
Seg3    = @XShift(Formula.Seg2; 100)
Signal  = @Join(Formula.Seg1; Formula.Seg3)
```

See also

"@Cut" on page 75

4.51 @Length

Function

Returns the **number of samples** in a waveform.

Syntax

`@Length(Waveform)`

Parameters

Waveform Input waveform

Output

The output is a numerical value.

Description

This function returns the number of samples that is available within the specified input waveform. This value can be used as input parameter for other functions.

Example

The following example creates a sine wave consisting of 1000 samples. The variable `NSamples` receives the length of this signal (1000).

```
Signal      = 25 * @SineWave(20k; 1000; 50)
NSamples    = @Length(Formula.Signal)
```

See also

"@XDelta" on page 277, "@XFirst" on page 280 and "@XShift" on page 282

4.52 @LessEqualThan

Function

This function performs a **less-than-or-equal-to** (\leq) evaluation on the two numerical input parameters.

Syntax

@LessEqualThan(*Param1*; *Param2*)

Parameters

<i>Param1</i>	Number: first parameter used for evaluation.
<i>Param2</i>	Number: second parameter used for evaluation.

Output

The output is 1 or 0.

Description

The LessEqualThan function performs a 'less-than-or-equal-to' evaluation on the input parameters.

If $\text{Param } 1 \leq \text{Param } 2$ then the return value will be 1 (true) else the return value will be 0 (false).

The LessEqualThan function is typically used in combination with the IIF function.

Example

The following example compares input parameters and provides a result that depends on the outcome:

```
LETExam1    = @LessEqualThan(5; 5)           => 1 (true)
LETExam2    = @LessEqualThan(12; 10)        => 0 (false)
IIFExample  = @IIF(Formula.LETExam2; "true"; "false")
```

See also

"@EqualTo" on page 87, "@GreaterEqualThan" on page 128, "@GreaterThan" on page 129, "@IIF" on page 140 and "@LessThan" on page 153

4.53 @LessThan

Function

This function performs a **less-than** (<) evaluation on the two numerical input parameters.

Syntax

`@LessThan(Param1; Param2)`

Parameters

<i>Param1</i>	Number: first parameter used for evaluation.
<i>Param2</i>	Number: second parameter used for evaluation.

Output

The output is 1 or 0.

Description

The LessThan function performs a 'less-than' evaluation on the input parameters. If Param 1 < Param 2 then the return value will be 1 (true) else the return value will be 0 (false).

The LessThan function is typically used in combination with the IIF function.

Example

The following example compares input parameters and provides a result that depends on the outcome:

```
LessThanExamp11 = @LessThan(5; => 1 (true) 100)
LessThanExamp12 = @LessThan(12; => 0 (false)10)
IIFExample      = @IIF(Formula.LessThanExamp12; "TRUE";
                      "FALSE")
```

See also

"@EqualTo" on page 87, "@GreaterEqualThan" on page 128, "@IIF" on page 140 and "@LessEqualThan" on page 152

4.54 @Ln

Function

Natural logarithm of a given number, (or logarithm to base “e”), written as $\ln(x)$ or $\log_e(x)$: the power to which you need to raise the base (e) in order to get the number.

Syntax

`@Ln(Par)`

Parameters

Par Input waveform or numerical value.

Output

Waveform or numerical value containing the natural logarithm of the input.

Description

The logarithm base e is calculated ($e = 2.718\dots$). This logarithm is called the natural logarithm. When a waveform parameter is used, the log function is calculated for each individual sample.

If a number is smaller than $1E-30$, the value is set to $\ln(1E-30) = -69.08$.

Formally, $\ln(a)$ may be defined as the area under the graph (integral) of $1/x$ from 1 to a , that is:

$$\ln(a) = \int_1^a \frac{1}{x} dx.$$

This function is the inverse function of @Exp.

Example

The following example shows that the natural logarithm of the system variable `System.Constants.e` equals 1:

```
One = @Ln(System.Constants.e)
```

The following example shows how the natural logarithm can be used on a waveform:

4 REFERENCE GUIDE

```
Wave      = @Ramp(10k; 1k; 1E-5; 10)
```

```
WaveLn    = @Ln(Formula.Wave)
```

See also

"@Exp" on page 88 and "@Log" on page 156

4.55 @Log

Function

Logarithm of a given number to base 10, written as $\log_{10}(x)$: the power to which you need to raise the base (10) in order to get the number.

Syntax

@Log(*Par*)

Parameters

Par Input waveform or numerical value.

Output

Waveform or numerical value containing the logarithm of the input.

Description

The logarithm base 10 is calculated. When a waveform parameter is used, the log function is calculated for each individual sample.

If a number is smaller than 1E-30, the value is set to $\log(1E-30) = -30$.

Example

If $10^x = 100$ then $x = \log_{10}(100)$

$x = @Log(100)$

The result will be $x = 2$

See also

"@Ln" on page 154 and "@Pow" on page 186

4.56 @Max

Function

Determines the **maximum** value of a waveform.

Syntax

@Max(Waveform)

@Max(Waveform; Begin)

@Max(Waveform; Begin; End)

Parameters

Waveform	Input waveform for which the maximum is to be determined.
Begin	Number: segment begin
End	Number: segment end

Output

The output is a numerical value. This function works also “while recording” when both the Begin and End parameters are set. The result will be calculated continuously from the data that is readily available.

Description

The maximum of the waveform or waveform segment is determined. The segment limits (Begin and End) are used to select a range of samples. If no segment limits are specified, the complete waveform is used. When only Begin is specified, the waveform segment from Begin to the end of the waveform is used.

Notice

*The **Begin** and **End** parameters have to be specified in the units of the horizontal axis (for example time) and not in samples.*

Example

The following example creates a 50 Hz sine wave. The maximum value of the first 20 ms (one period of 50 Hz) of the signal is calculated to determine the (peak) amplitude:

```
Signal = 25 * @SineWave(20k; 1000; 50)
Ampl   = @Max(Formula.Signal; 0; 20m)
```

If the position of the maximum is to be determined, use the @MaxPos function instead. The maximum value at the position found can then be obtained using the @Value function:

```
TMax   = @MaxPos(Formula.Signal; 0; 20m)
Ampl   = @Value(Formula.Signal; Formula.TMax)
```

See also

"@MaxPos" on page 160, "@Min" on page 167, "@MinPos" on page 170 and
"@Value" on page 276

4.57 @MaxNum

Function

Determines the **maximum** value of two or more **numerical** values.

Syntax

`@MaxNum(Par1; ...; ParN)`

Parameters

<i>Par1</i>	First numerical value
<i>ParN</i>	Last numerical value, N >= 2

Output

Maximum of all numerical values.

Description

This function returns the largest numerical value of all parameters specified. Each parameter must be a numerical value containing a valid value. If one of the parameters has the incorrect type or contains no value, the function returns no value.

Example

The following example calculates the minimum of a waveform but clips the value if it is below zero. The waveform is assumed to be a real-world signal:

```
Signal      = Active.Group1.Recorder_A.Ch_A1
MinValue    = @Min(Formula.Signal)
ClippedMin  = @MaxNum(Formula.MinValue; 0)
```

The following example returns -10:

```
Ten         = @MaxNum(-10; 3; 2.4; 10.0; -9.9)
```

See also

"@MinNum" on page 169

4.58 @MaxPos

Function

Determines the **position** of the **maximum** of a waveform.

Syntax

@MaxPos(Waveform)

@MaxPos(Waveform; Begin)

@MaxPos(Waveform; Begin; End)

Parameters

<i>Waveform</i>	Input waveform for which the position of the maximum is to be determined.
<i>Begin</i>	Number: segment begin
<i>End</i>	Number: segment end

Output

The output is the x-position of the maximum. This function works also “while recording” when both the Begin and End parameters are set. The result will be calculated continuously from the data that is readily available.

Description

The x-position of the maximum of the waveform or waveform segment is determined. The segment limits (Begin and End) are used to select a range of samples. If no segment limits are specified, the complete waveform is used. When only Begin is specified, the waveform segment from Begin to the end of the waveform is used.

Notice

*The **Begin** and **End** parameters have to be specified in the units of the horizontal axis (for example time) and not in samples.*

Notice

The returned value is the position on the x-axis of the waveform in terms of x-position, not in terms of a sample number. The units of this numerical value are the x-units of the waveform.

Example

The following example determines the time at which the pressure reaches its maximum value. The actual maximum pressure is determined using the @Value function for this time.

The pressure is assumed to be a real-world signal.

```
Pressure    = Active.Group1.Recorder_A.Ch_A1
TimeOfMax   = @MaxPos(Formula.Pressure)
MaxPress    = @Value(Formula.Pressure;Formula.TimeOfMax)
```

See also

"@Max" on page 157, "@Min" on page 167 and "@MinPos" on page 170

4.59 @Mean

Function

Calculates the arithmetic **mean** value of a waveform.

Syntax

@Mean(Waveform)
 @Mean(Waveform; Begin)
 @Mean(Waveform; Begin; End)

Parameters

<i>Waveform</i>	Input waveform for which the mean value is to be calculated.
<i>Begin</i>	Number: segment begin
<i>End</i>	Number: segment end

Output

The output is a numerical value. This function works also “while recording” when both the Begin and End parameters are set. The result will be calculated continuously from the data that is readily available.

Description

The mean value is calculated using the following formula:

$$\text{Mean} = \frac{1}{N} \sum_{n=n_1}^{n_2} y(n) \text{ with } N = (n_2 - n_1 + 1)$$

The segment limits Begin (n_1) and End (n_2) are used to select a range of samples. If no segment limits are specified, the complete waveform is used. When only Begin is specified, the waveform segment from Begin to the end of the waveform is used.

Notice

The **Begin** and **End** parameters have to be specified in the units of the horizontal axis (for example time) and not in samples.

Example

The following example creates a signal (constant value 100) with hum and noise added. The mean value of the first 20 ms (one period of 50 Hz) of the signal is calculated:

```
Signal = 100 + 0.5 * @SineWave(20k; 1000; 50)
        + @Noise(20k; 1000)
Avg     = @Mean(Formula.Signal; 0; 20m)
```

The value of Avg is a much more accurate reading of the signal than each individual sample in the waveform because of the averaging performed.

See also

"@RMS" on page 216 and "@ StdDev" on page 233

4.60 @MedianFilter

Function

Filters a waveform using a **median** filter.

Syntax

@MedianFilter(*Waveform*)

@MedianFilter(*Waveform*; *N*)

Parameters

Waveform Waveform to be filtered.

N The window size of the median filter. The default is 3.
Range: 3 ... 1001.

Output

Median filtered waveform.

Description

The median filter iterates through the input waveform sample by sample and takes the median of the sample and its neighboring samples. A sample together with its neighbors is called the “window”. The number of samples included in the window is the window size (*N*).

The median is the middle value after the samples in the window have been sorted numerically.

The median filter can be expressed with the following formula:

$$\text{Output}(n) = \text{Median}(\text{Sort}(\text{Input}(n-M) \dots \text{Input}(n+M)))$$

With:

N = Window size

M = (*N*-1)/2

n = Sample number in range 1 to 'end-of-waveform'

The window size *N* controls the number of samples used before sorting and taking the median. If *N* is not an odd integer, it will be rounded to the nearest odd integer first. If the value is smaller than 3, it is set to 3. If the value is larger than 1001, it is set to 1001.

Median filtering is a smoothing technique, similar to a moving average filter (See “@Smooth” on page 222). Smoothing techniques are effective at removing noise in smooth regions of an input waveform, but adversely affect edges. To reduce the edge effects, additional samples (‘M’) are added before the first resp. after the last input sample.

The median filter filters out high frequency components i.e. noise. The best filtering is achieved if the window size is equal to some integer multiple of the period of the disturbances. Due to the sorting, in combination with taking the median, the characteristic waveform features of the waveform are preserved. At locations where the input signal goes from a rising signal to a falling signal and vice versa, some filtering effects might be visible however. The length of the output waveform is the same as the length of the input waveform.

Example

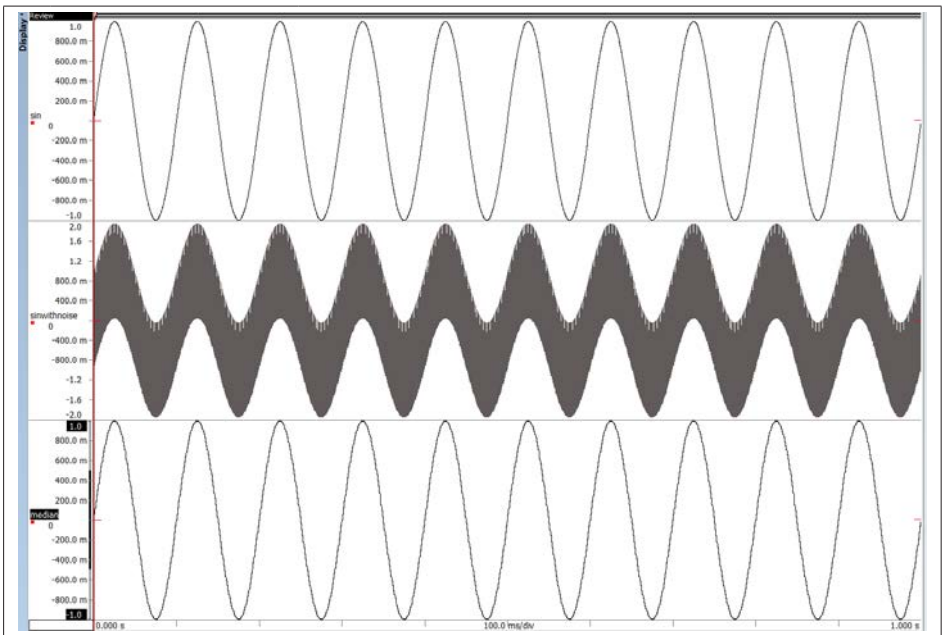


Fig. 4.38 Median filtering on a noisy sinusoidal waveform

Where:

```
sin                = @SineWave(10k;10k+1;10)
sinwithnoise      = Formula.sin + @SineWave(10k;10k+1;1000)
median            = @MedianFilter(Formula.sinwithnoise;51)
```

See also

"@Smooth" on page 222

4.61 @Min

Function

Determines the **minimum** value of a waveform.

Syntax

@Min(Waveform)

@Min(Waveform; Begin)

@Min(Waveform; Begin; End)

Parameters

Waveform	Input waveform for which the minimum is to be determined.
Begin	Number: segment begin
End	Number: segment end

Output

The output is a numerical value. This function works also “while recording” when both the Begin and End parameters are set. The result will be calculated continuously from the data that is readily available.

Description

The minimum of the waveform or waveform segment is determined. The segment limits (Begin and End) are used to select a range of samples. If no segment limits are specified, the complete waveform is used. When only Begin is specified, the waveform segment from Begin to the end of the waveform is used.

Notice

*The **Begin** and **End** parameters have to be specified in the units of the horizontal axis (for example time) and not in samples.*

Example

The following example creates a 50 Hz sine wave on top of a DC value. The maximum and minimum value of the first 20 ms (one period of 50 Hz) of the signal is calculated to determine the (peak-to-peak) amplitude of the 50 Hz component:

```
Signal    = 100 + 25 * @SineWave(20k; 1000; 50)
MinAmpl   = @Min(Formula.Signal; 0; 20m)
MaxAmpl   = @Max(Formula.Signal; 0; 20m)
Vpp       = Formula.MaxAmpl - Formula.MinAmpl
```

If the position of the minimum is to be determined, use the @MinPos function instead. The minimum value at the position found can then be obtained using the @Value function:

```
TMin      = @MinPos(Signal; 0; 20m)
Minim     = @Value(Formula.Signal; Formula.TMin)
```

See also

"@Max" on page 157, "@MaxPos" on page 160 and "@MinPos" on page 170

4.62 @MinNum

Function

Calculates the **minimum** value of two or more **numerical** values.

Syntax

`@MinNum(Par1; ...; ParN)`

Parameters

<i>Par1</i>	First numerical value
<i>ParN</i>	Last numerical value. N >= 2

Output

Minimum of all numerical values.

Description

This function returns the smallest numerical value of all parameters specified. Each parameter must be a numerical value containing a valid value. If one of the parameters has the incorrect type or contains no value, the function returns no value.

Example

The following example calculates the maximum of a waveform but clips the value if it is larger than 100. The waveform is assumed to be a real-world signal:

```
Signal      = Active.Group1.Recorder_A.Ch_A1
MaxValue    = @Max(Formula.Signal)
ClippedMax  = @MinNum(Formula.MaxValue; 100)
```

The following example returns -10:

```
MinusTen    = @MinNum(45.0; 10; -3.3; 20; -10.0; 5)
```

See also

"@MaxNum" on page 159

4.63 @MinPos

Function

Determines the **position** of the **minimum** of a waveform.

Syntax

@MinPos(*Waveform*)

@MinPos(*Waveform*; *Begin*)

@MinPos(*Waveform*; *Begin*; *End*)

Parameters

<i>Waveform</i>	Input waveform for which the position of the minimum is to be determined.
<i>Begin</i>	Number: segment begin
<i>End</i>	Number: segment end

Output

The output is the x-position of the minimum.

Description

The x-position of the minimum of the waveform or waveform segment is determined. The segment limits (*Begin* and *End*) are used to select a range of samples. If no segment limits are specified, the complete waveform is used. When only *Begin* is specified, the waveform segment from *Begin* to the end of the waveform is used.

Notice

The **Begin** and **End** parameters have to be specified in the units of the horizontal axis (for example time) and not in samples.

Notice

The returned value is the position on the x-axis of the waveform in terms of x-position, not a sample number. The units of this numerical value are the x-units of the waveform.

Example

The following example determines the time at which the pressure reaches its minimum value. The actual minimum pressure is determined using the @Value function for this time. The variable Pressure is assumed to be a real-world signal.

```
Pressure    = Active.Group1.Recorder_A.Ch_A1  
TimeOfMin   = @MinPos(Formula.Pressure)  
MinPress    = @Value(Formula.Pressure; Formula.TimeOfMin)
```

See also

"@Max" on page 157, "@MaxPos" on page 160 and "@Min" on page 167

4.64 @NextHillPos

Function

This function searches for the **position** of the **next local maximum** in a waveform.

Syntax

@NextHillPos(*Waveform*; *StartPos*; *Hysteresis*)

Parameters

<i>Waveform</i>	Input waveform for which the position of the next local maximum is to be determined.
<i>StartPos</i>	Number: start position in x-units where searching should start (forwards).
<i>Hysteresis</i>	Number: noise suppression hysteresis value.

Output

The output is the x-position of the next local maximum.

Description

The x-position of the next local maximum in the waveform is determined.

Searching starts from the x-position specified by StartPos to the right. If StartPos is before the beginning of the waveform, the start of the waveform is used as start position for the search.

The hysteresis value is used to suppress the effects of noise on the signal. If for instance 100 mV peak-to-peak noise is present on a signal, specifying 200 mV as hysteresis value will prevent the algorithm from determining small noise peaks as local maximum.

Notice

The returned value is the position on the x-axis of the waveform in terms of x-position, not a sample number. The units of this numerical value are the x-units of the waveform.

Example

The following example determines the position of a local maximum on a 1 V sine wave with 10 mV noise, starting from 500 ms searching forwards, using a hysteresis of 40 mV:

```
Signal = @SineWave(10k; 10000; 50 ) + 0.01 * @Noise(10k;  
10000)  
NHPos = @NextHillPos(Formula.Signal; 500m; 40m)
```

See also

"@NextValleyPos" on page 176, "@PrevHillPos" on page 187 and
"@PrevValleyPos" on page 191

4.65 @NextLvlCross

Function

Determines the position of the **next crossing** of a waveform with a specified signal level.

Syntax

@NextLvlCross(*Waveform*; *StartPos*; *Level*)

@NextLvlCross(*Waveform*; *StartPos*; *Level*; *Slope*)

Parameters

<i>Waveform</i>	Input waveform for which the position of the next level crossing is to be determined.
<i>StartPos</i>	Number: start position in x-units where searching should start.
<i>Level</i>	Number: amplitude level to search for
<i>Slope</i>	Number: direction of the slope (-1, 0, 1)

Output

The output is the x-position of the next level crossing.

Description

The x-position of the next level crossing of the waveform with the specified level in the specified direction is determined.

Searching starts at the x-position specified by *StartPos* and goes to the right in the positive time direction. If *StartPos* is before the beginning of the waveform, the start of the waveform is used as start position for the search.

The *Slope* parameter controls the type of level crossing that is searched for:

- Slope = 1: Positive level crossing (from below level to above level)
- Slope = -1: Negative level crossing (from above level to below level)
- Slope = 0: Any crossing (either positive or negative)

Notice

The returned value is the position on the x-axis of the waveform in terms of x-position, not a sample number. The units of this numerical value are the x-units of the waveform.

Example

The following example determines the start and end time of the first TTL pulse present in the TTLSignal that is assumed to be available as real-world signal in "Active.Group1.Recorder_A.Ch_A1".

```
TTSignal    = Active.Group1.Recorder_A.Ch_A1
TimeStart   = @NextLvlCross(Formula.TTSignal; -1E20; 2.5; 1)
TimeEnd     = @NextLvlCross(Formula.TTSignal;
                             Formula.TimeStart; 2.5; -1)
```

See also

"@PrevLvlCross" on page 189

4.66 @NextValleyPos

Function

This function searches for the **position** of the **next local minimum** in a waveform.

Syntax

@NextValleyPos(Waveform; StartPos; Hysteresis)

Parameters

<i>Waveform</i>	Input waveform for which the position of the next local minimum is to be determined.
<i>StartPos</i>	Number: start position in x-units where searching should start (forwards).
<i>Hysteresis</i>	Number: noise suppression hysteresis value.

Output

The output is the x-position of the next local minimum.

Description

The x-position of the next local minimum in the waveform is determined.

Searching starts from the x-position specified by StartPos to the right. If StartPos is before the beginning of the waveform, the start of the waveform is used as start position for the search.

The hysteresis value is used to suppress the effects of noise on the signal. If for instance 100 mV peak-to-peak noise is present on a signal, specifying 200 mV as hysteresis value will prevent the algorithm from determining small negative noise peaks as local minimum.

Notice

The returned value is the position on the x-axis of the waveform in terms of x-position, not a sample number. The units of this numerical value are the x-units of the waveform.

Example

The following example determines the position of a local minimum on a 1 V sine wave with 10 mV noise, starting from 500 ms searching forwards, using a hysteresis of 40 mV:

```
Signal = @SineWave(10k; 10000; 50) + 0.01 * @Noise(10k; 10000)
NVPos  = @NextValleyPos(Formula.Signal; 500m; 40m)
```

See also

"@NextHillPos" on page 172, "@PrevHillPos" on page 187 and "@PrevValleyPos" on page 191

4.67 @Noise

Function

Generates a waveform containing **noise**.

Syntax

`@Noise(FSampling; NSamples)`

Parameters

<i>FSampling</i>	Number: sampling frequency
<i>NSamples</i>	Number of samples

Output

Waveform containing noise.

Description

This function creates a waveform containing random noise with an amplitude between -1 and 1. The sampling frequency and number of samples to generate must be specified.

The possibility to generate noise can be very useful to investigate the influence of noise on the analysis of a measured or generated waveform.

Example

The following example generates a 5 V amplitude sine wave hidden in 10 V peak-to-peak noise.

```
Signal = 5 * @SineWave(10k; 1000; 50) + 10 *  
         @Noise(10k; 1000)
```

See also

"@Pulse" on page 193, "@Ramp" on page 197, "@SineWave" on page 220 and "@SquareWave" on page 232

4.68 @Not

Function

This function performs a logical **NOT** evaluation on the input parameters.

Syntax

@Not(*Param1*)

Parameters

Param1 Number used for the NOT evaluation.

Output

The output is a 1 or a 0.

Description

The @Not function performs a logical NOT evaluation on the input parameter. Depending on the evaluation the result will be 1 or 0. A numerical value not equal to 0 corresponds to a logical "True" and a numerical 0 corresponds to a logical "False".

The truth table of the NOT function is:

NOT function True/False values:	
Param1	Result
True	False
False	True

Example

The following is a list of examples and their return value.

```
NotExamp11 = @Not(1)           => 0 (= false)
NotExamp12 = @Not(4)           => 0 (= false)
NotExamp13 = @Not(0)           => 1 (= true)
```

See also

"@And" on page 57 and "@Or" on page 182

4.69 @NumberOfSweeps

Function

Returns the **number** of **sweeps**.

Syntax

@NumberOfSweeps(*Signal*)

@NumberOfSweeps(*Signal*; *Begin*)

@NumberOfSweeps(*Signal*; *Begin*; *End*)

Parameters

<i>Signal</i>	Waveform: multi-sweep signal
<i>Begin</i>	Number: segment begin
<i>End</i>	Number: segment end

Output

Numerical value representing the number of sweeps.

Description

The number of sweeps of the waveform or waveform segment is determined. The segment limits (*Begin* and *End*) are used to select a range of samples. If no segment limits are specified, the complete waveform is used. When only *Begin* is specified, the waveform segment from *Begin* to the end of the waveform is used.

If the begin or end of the segment is part of a sweep then this sweep is also counted.

Notice

*The **Begin** and **End** parameters need to be specified in the units of the horizontal axis (for example time) and not as samples.*

Example

The example is using the following signal:

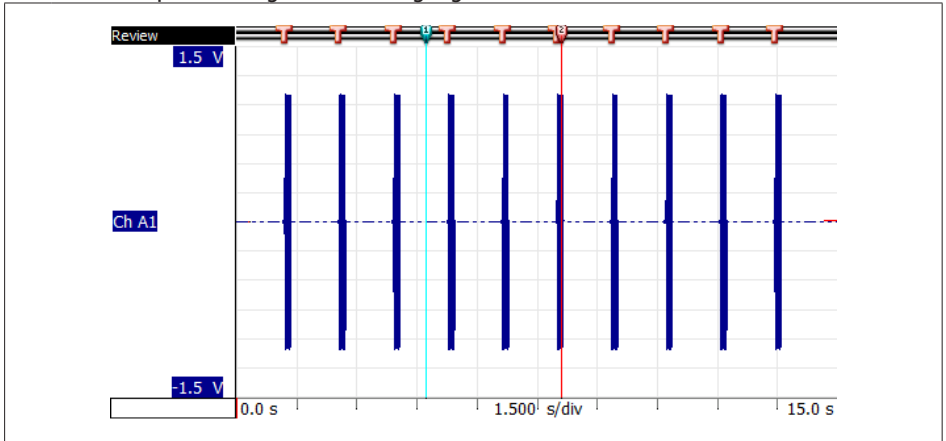


Fig. 4.39 Number of sweeps

The following formula returns the total number of sweeps of the signal
Formula.Ch_A1

```
TotalNumberOfSweeps           = @NumberOfSweeps
                                (Formula.Ch_A1)
NumberOfSweepsBetweenCursors = @NumberOfSweeps
                                (Formula.Ch_A1;
                                Display.Display.Cursor1.
                                XPosition;
                                Display.Display.Cursor2.
                                XPosition)
```

In the example above the results are:

```
TotalNumberOfSweeps           = 10
NumberOfSweepsBetweenCursors = 3
```

See also

"@Sweep" on page 235, "@SweepEndTime" on page 236,
"@SweepNumberAtTime" on page 237, "@SweepSource" on page 238,
"@SweepStartTime" on page 239

4.70 @Or

Function

This function performs a logical **OR** evaluation on the input parameters.

Syntax

@Or(*Param1*; ...; *ParamN*)

Parameters

Param1 Number: first parameter used for the OR evaluation.
ParamN Number: last parameter used for the OR evaluation.
N >= 2

Output

The output is a 1 or a 0.

Description

The @Or function performs a logical OR evaluation on the input parameters. Depending on the evaluation the result will be 1 or 0. A numerical value not equal to 0 corresponds to a logical "True" and a numerical 0 corresponds to a logical "False".

The truth table of the OR function is:

OR function True/False values:		
Param1	Param2	Result
True	True	True
True	False	True
False	True	True
False	False	False

Example

The following is a list of examples and their return value.

OrExamp11 = @Or(1; 1) => 1 (= true)
OrExamp12 = @Or(1;4;10) => 1 (= true)
OrExamp13 = @Or(1;4;0;6) => 1 (= true)

```
OrExamp14 = @Or(0;0)      => 0 (= false)
```

See also

"@And" on page 57 and "@Not" on page 179

4.71 @Period

Function

Determines the **period** of a waveform. The period is the reciprocal of the frequency.

Syntax

@Period(*Waveform*)

@Period(*Waveform*; *Begin*)

@Period(*Waveform*; *Begin*; *End*)

Parameters

<i>Waveform</i>	Input waveform for which the period is to be determined.
<i>Begin</i>	Number: segment begin
<i>End</i>	Number: segment end

Output

The output is a numerical value.

Description

The period of the waveform or waveform segment is determined. The segment limits (*Begin* and *End*) are used to select a range of samples. If no segment limits are specified, the complete waveform is used. When only *Begin* is specified, the waveform segment from *Begin* to the end of the waveform is used.

Notice

The ***Begin*** and ***End*** parameters have to be specified in the units of the horizontal axis (for example time) and not in samples.

The period is determined using base and top magnitude algorithms. The mean of these magnitudes is considered to be the zero level. The number of zero crossings in the range *Begin* to *End* is calculated. The crossings should be in the same direction as the first one. The crossings are determined with a hysteresis around the zero level to suppress the effects of noise. The hysteresis is +/- 5% of the difference between top and base.

From the time difference between the first and last zero crossing in the same direction and the number of zero crossings in between, the period of the waveform is determined.

Example

The following example determines the period of a 50 Hz signal:

```
Signal = @SineWave(10k; 10k; 50) + 0.01 * @Noise(10k; 10k)
Period = @Period(Formula.Signal)
Freq    = 1 / Formula.Period
```

See also

"@Frequency" on page 121

4.72 @Pow

Function

Exponentiation, the mathematical operation, written a^n , involving two parameters: the base “a” and the exponent “n”.

Syntax

`@Pow(Base; Exponent)`

Parameters

<i>Base</i>	Base waveform or numerical value.
<i>Exponent</i>	Exponent waveform or numerical value.

Output

Waveform or numerical value containing the exponentiation of base and exponent.

Description

The exponential function calculates the base raised to the power exponent.

If both the base and the exponent are numerical values, the result is a numerical value.

If either the base or the exponent is a waveform, the exponentiation function is calculated for each individual sample in combination with the numerical value. The result is a waveform.

Notice

It is not possible to use a waveform for both base and exponent.

Example

The following results are equivalent:

```
Input      = 2 * @SineWave(10k; 10k; 10)
Result1    = @Exp(Formula.Input)
Result2    = @Pow(System.Constants.e; Formula.Input)
```

See also

“@Exp” on page 88

4.73 @PrevHillPos

Function

This function searches for the position of the **previous local maximum** in a waveform.

Syntax

@PrevHillPos(*Waveform*; *StartPos*; *Hysteresis*)

Parameters

<i>Waveform</i>	Input waveform for which the position of the previous local maximum is to be determined.
<i>StartPos</i>	Number: start position in x-units where searching should start (backwards).
<i>Hysteresis</i>	Number: noise suppression hysteresis value.

Output

The output is the x-position of the previous local maximum.

Description

The x-position of the previous local maximum in the waveform is determined.

Searching starts from the x-position specified by StartPos to the left. If StartPos is past the end of the waveform, the end of the waveform is used as start position for the search.

The hysteresis value is used to suppress the effects of noise on the signal. If for instance 100 mV peak-to-peak noise is present on a signal, specifying 200 mV as hysteresis value will prevent the algorithm from determining small noise peaks as local maximum.

Notice

The returned value is the position on the x-axis of the waveform in terms of x-position, not a sample number. The units of this numerical value are the x-units of the waveform.

Example

The following example determines the position of a local maximum on a 1 V sine wave with 10 mV noise, starting from 500 ms searching backwards, using a hysteresis of 40 mV:

```
Signal = @SineWave(10k; 10000; 50) + 0.01 * @Noise(10k;  
10000)  
PHPos = @PrevHillPos(Formula.Signal; 500m; 40m)
```

See also

"@NextHillPos" on page 172, "@NextValleyPos" on page 176 and "@PrevValleyPos" on page 191

4.74 @PrevLvlCross

Function

Determines the position of the **previous crossing** of a waveform with a specified signal **level**.

Syntax

@PrevLvlCross(Waveform; StartPos; Level)

@PrevLvlCross(Waveform; StartPos; Level; Slope)

Parameters

<i>Waveform</i>	Input waveform for which the position of the previous level crossing is to be determined.
<i>StartPos</i>	Number: start position in x-units where searching should start (backwards).
<i>Level</i>	Number: amplitude level to search for
<i>Slope</i>	Number: direction of the slope (-1, 0, 1)

Output

The output is the x-position of the previous level crossing.

Description

The x-position of the previous level crossing of the waveform with the specified level in the specified direction is determined.

Searching starts at the x-position specified by StartPos and goes to the left in the negative time direction.

The Slope parameter controls the type of level crossing to search for:

- Slope = 1: Positive level crossing (from below level to above level)
- Slope = -1: Negative level crossing (from above level to below level)
- Slope = 0: Any crossing (either positive or negative)

Notice

The returned value is the position on the x-axis of the waveform in terms of x-position, not a sample number. The units of this numerical value are the x-units of the waveform.

Example

The following example determines the start and end time of the last TTL pulse present in the TTLSignal. This signal is assumed to be a real-world signal.

```
TTLSignal = Active.Group1.Recorder_A.Ch_A1
TimeEnd   = @PrevLvlCrossing(Formula.TTLSignal; 1E20; 2.5; 1)
TimeStart = @PrevLvlCrossing(Formula.TTLSignal;
                             Formula.TimeEnd; 2.5; -1)
```

See also

"@NextLvlCross" on page 174

4.75 @PrevValleyPos

Function

This function searches for the position of the **previous local minimum** in a waveform.

Syntax

@PrevValleyPos(*Waveform*; *StartPos*; *Hysteresis*)

Parameters

<i>Waveform</i>	Input waveform for which the position of the previous local minimum is to be determined.
<i>StartPos</i>	Number: start position in x-units where searching should start (backwards).
<i>Hysteresis</i>	Number: noise suppression hysteresis value.

Output

The output is the x-position of the previous local minimum.

Description

The x-position of the previous local minimum in the waveform is determined.

Searching starts from the x-position specified by StartPos to the left. If StartPos is past the end of the waveform, the end of the waveform is used as start position for the search.

The hysteresis value is used to suppress the effects of noise on the signal. If for instance 100 mV peak-to-peak noise is present on a signal, specifying 200 mV as hysteresis value will prevent the algorithm from determining small negative noise peaks as local minimum.

Notice

The returned value is the position on the x-axis of the waveform in terms of x-position, not a sample number. The units of this numerical value are the x-units of the waveform.

Example

The following example determines the position of a local minimum on a 1 V sine wave with 10 mV noise, starting from 500 ms searching backwards, using a hysteresis of 40 mV:

```
Signal = @SineWave(10k; 10000; 50) + 0.01 * @Noise(10k;  
10000)  
PVPos = @PrevValleyPos(Formula.Signal; 500m; 40m)
```

See also

"@NextHillPos" on page 172, "@NextValleyPos" on page 176 and "@PrevHillPos" on page 187

4.76 @Pulse

Function

Generates a waveform containing a single pulse.

Syntax

`@Pulse(FSampling; NSamples; PulseStart)`

`@Pulse(FSampling; NSamples; PulseStart; PulseWidth)`

Parameters

<i>FSampling</i>	Number: sampling frequency
<i>NSamples</i>	Number of samples
<i>PulseStart</i>	Number: Sample where pulse starts
<i>PulseWidth</i>	Number: Pulse width in samples the default value is set to 1.

Output

Waveform containing a single pulse.

Description

This function creates a single pulse waveform. The sampling frequency, number of samples to generate the single pulse can be specified. There is no limit on the length of the waveform. The amplitude of the pulse wave is 1 V.

The pulse function can be used to determine the impulse and step response of a filter function.

You can use this function to synthesize a variety of waveforms. This data can be used as input for other analysis functions.

Example

```
SignalPulse      = @Pulse(80k; 80k; 20k)
ImpulseResponse  = @FilterButterworthLP
                  (Formula.SignalPulse; 2; 200)
SignalStep       = @Pulse(80k; 80k; 20k; 10k)
StepResponse     = @FilterButterworthLP
                  (Formula.SignalStep; 2; 200)
```

If you are only interested in a part of the signal than you can use the pulse function to make all values zero outside of a specified interval. This interval can be defined by the *PulseStart* and *PulseWidth* parameters of the Pulse function.

```
SignalInterval = Formula.SignalStep *  
                Active.Group1.Recorder_A.Ch_A1
```

See also

"@Noise" on page 178, "@SineWave" on page 220, "@SquareWave" on page 232 and "@Ramp" on page 197

4.77 @PulseWidth

Function

Determines the **pulse width** of a pulse in a waveform.

Syntax

@PulseWidth(*Waveform*)

@PulseWidth(*Waveform*; *Begin*)

@PulseWidth(*Waveform*; *Begin*; *End*)

Parameters

Waveform Input waveform containing the pulse.

Begin Number: segment begin

End Number: segment end

Output

The output is a numerical value.

This function works “while recording” when the *Begin* and *End* parameters are set. The result will be calculated when the data between *Begin* and *End* is available.

Description

The pulse width is determined by taking the time difference between the first and second 50% magnitude transitions of the pulse in the waveform (or waveform segment).

Magnitude percentage crossings are calculated using the base and top magnitudes. The base and top magnitudes are determined by searching two dominant populations in the waveform (or waveform segment) magnitude histogram and taking the mean values of these two populations.

The segment limits (*Begin* and *End*) are used to select a range of samples. If no segment limits are specified, the complete waveform is used.

When only *Begin* is specified, the waveform segment from *Begin* to the end of the waveform is used.

Notice

The **Begin** and **End** parameters have to be specified in the units of the horizontal axis (for example time) and not in samples.

Example

The following example creates a pulse and calculates the 50%-50% pulse width. The result is 150 ms:

```
Sig1    = @Ramp(1k; 100; 0; 0)
Sig2    = @Ramp(1k; 51; 0; 10)
Sig3    = @Ramp(1k; 99; 10; 10)
Sig4    = Sig4 x
Signal  = @Join(Formula.Sig1; Formula.Sig2;
                Formula.Sig3; Formula.Sig4; Formula.Sig1)
Width   = @PulseWidth(Formula.Signal)
```

See also

"@FallTime" on page 91 and "@RiseTime" on page 214

4.78 @Ramp

Function

Generates a waveform containing one or more linear **ramps**.

Syntax

`@Ramp(FSampling; NSamples; YStart; YEnd)`

`@Ramp(FSampling; NSamples; YStart; YEnd; Count)`

Parameters

<i>FSampling</i>	Number: sampling frequency
<i>NSamples</i>	Number: number of samples
<i>YStart</i>	Number: value of first sample
<i>YEnd</i>	Number: value of last sample
<i>Count</i>	Number: number of ramps

Output

Waveform containing one or more linear ramps.

Description

This function creates a waveform containing one or more linear ramps starting at *YStart* and ending at *YEnd*. The sampling frequency and number of samples to generate can also be specified. The length of the waveform is limited to 1 GigaSamples.

When specifying the optional parameter *Count*, the specified number of ramps will be generated. Using the *Count* value results in a sawtooth waveform. When *Count* is not specified, the value 0 will be used. This will create a single ramp only, not a single sawtooth.

The possibility to generate a ramp function in combination with the `@Join` function and other waveform generation functions is used to synthesize a variety of waveforms. The simulated data can be used as input for other analysis functions.

Example

The following example generates a simulated pulse of 10 V amplitude with 4 V peak-to-peak noise added:

```
Sig1    = @Ramp(1k; 100; 0; 0)
Sig2    = @Ramp(1k; 51; 0; 10)
Sig3    = @Ramp(1k; 100; 10; 10)
Sig4    = @Ramp(1k; 51; 10; 0)
Signal  = @Join(Formula.Sig1; Formula.Sig2;
                Formula.Sig3; Formula.Sig4) + 2 * @Noise(1k; 302)
```

See also

"@Noise" on page 178, "@Pulse" on page 193, "@SineWave" on page 220 and
"@SquareWave" on page 232

4.79 @ReadAsciiFile

Function

Imports waveform data from an **ASCII** (text) file.

Syntax

@ReadAsciiFile(*Filename*)

Parameters

Filename String: the full path giving the name and the unique location of the ASCII file.

Output

The output is a waveform.

Description

This function imports waveform data from an ASCII file. The samples of the imported data are interpreted as being equidistant.

The ASCII file must contain a header and a data part. Two different types of headers are supported: a **short** version and a **long** version.

Short Header:		
Line	Description	Example
1	Number of header rows	5
2	Data delimiter (dot, comma, tab or semi-column)	;
3	Number of data pairs	n
4	Scale factor for x and y (x; y)	5.0E-8;2.44E+0
5	Units for x and y (x; y)	s;A

Long Header:		
Line	Description	Example
1	Number of header rows	12
2	Data delimiter (dot, comma, tab or semi-column)	;
3	Number of data pairs	n
4	Date of data generation	17.03.00
5	Time of data generation	23:59
6	Extra information about producer of the data, this line can be blank. Only for compatibility reasons.	TDG 0.5
7	Comments (max. 80 characters)	First example: Test 1;
8	Scale factor for x and y (x; y)	5.0E-8;2.44E+0
9	Units for x and y (x; y)	s;A
10	Name of scaled units (x; y)	time;current
11	Resolution of y-data in bit	12
12	Use of dynamic range in %	80

Data:

The data comes after the header and starts at line 6 or line 13, depending on the size of the header.

Each data line contains an x, y pair. The separator between the x and y value is defined at line 2 of the header.

The samples of the imported data are interpreted as being equidistant.

The first data line contains the first (lowest) x-value.

Example

The following example shows how the file "ReferenceCurve.asc" is read.

```
Signal = @ReadAsciiFile("d:\data\ReferenceCurve.asc")
```


4.80 @ReadLogFile

Function

Reads a sequence of numerical values from a Perception **log file** and creates a waveform from those values.

Syntax

```
@ReadLogFile(Filename; Fieldname)  
@ReadLogFile(Filename; Fieldname; X-Units)  
@ReadLogFile(Filename; Fieldname; X-Units; SampleInterval)  
@ReadLogFile(Filename; Fieldname; X-Units; SampleInterval; X-Start)
```

Parameters

<i>Filename</i>	String: the full path giving the name and the unique location of the Perception log file to be read.
<i>Fieldname</i>	String: the name of the numerical field in the Perception logging file to be read.
<i>X-Units</i>	String: The X-Units of the new created waveform data source; the default value is empty.
<i>SampleInterval</i>	Number: The sample interval of the new created waveform data source; the sample interval is expressed in X-Units; the default value is 1.
<i>X-Start</i>	Number: The start of the new created waveform data source expressed in X-Units; the default value is 0

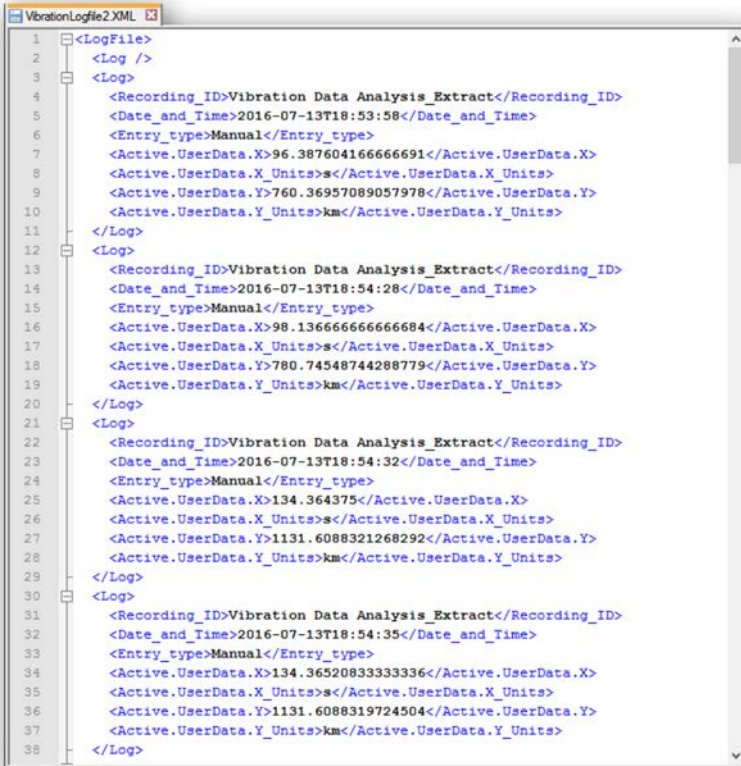
Output

The output is a waveform.

Description

This function reads numerical logged values from a Perception log file and creates a waveform from these values. Perception has the ability to save numerical values into a log file; these loggings can be done manually or as part of an automation setup ; for example after each recording. For more information on Perception logging, please refer to the Perception Data Acquisition Software manual.

The log file is an xml structured file. See Fig. 4.40 for an example of such a file.



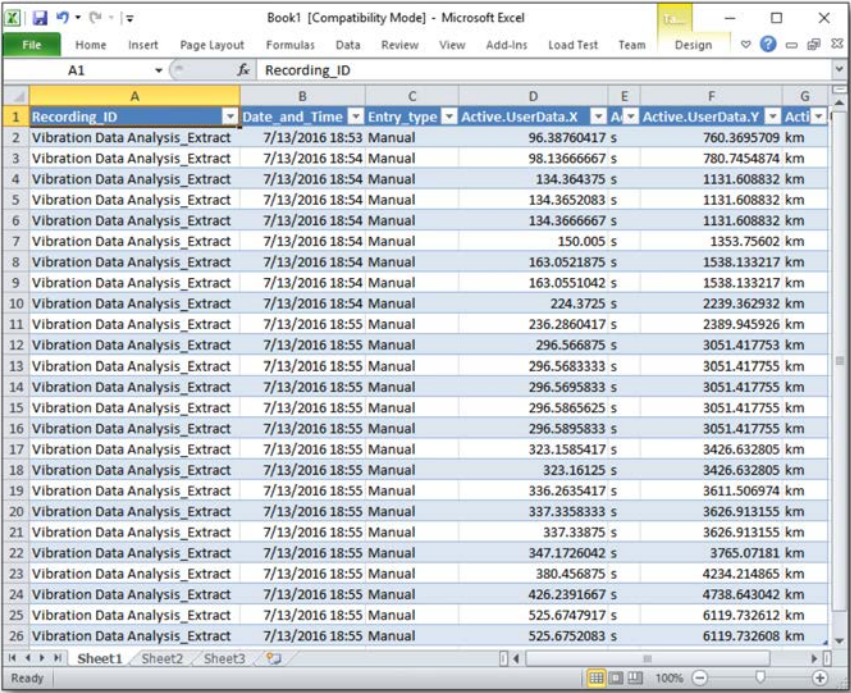
```

1  <LogFile>
2  <Log />
3  <Log>
4  <Recording_ID>Vibration Data Analysis_Extract</Recording_ID>
5  <Date_and_Time>2016-07-13T18:53:58</Date_and_Time>
6  <Entry_type>Manual</Entry_type>
7  <Active.UserData.X>96.387604166666691</Active.UserData.X>
8  <Active.UserData.X_Units>s</Active.UserData.X_Units>
9  <Active.UserData.Y>760.36957089057978</Active.UserData.Y>
10 <Active.UserData.Y_Units>km</Active.UserData.Y_Units>
11 </Log>
12 <Log>
13 <Recording_ID>Vibration Data Analysis_Extract</Recording_ID>
14 <Date_and_Time>2016-07-13T18:54:28</Date_and_Time>
15 <Entry_type>Manual</Entry_type>
16 <Active.UserData.X>98.136666666666684</Active.UserData.X>
17 <Active.UserData.X_Units>s</Active.UserData.X_Units>
18 <Active.UserData.Y>780.74548744288779</Active.UserData.Y>
19 <Active.UserData.Y_Units>km</Active.UserData.Y_Units>
20 </Log>
21 <Log>
22 <Recording_ID>Vibration Data Analysis_Extract</Recording_ID>
23 <Date_and_Time>2016-07-13T18:54:32</Date_and_Time>
24 <Entry_type>Manual</Entry_type>
25 <Active.UserData.X>134.364375</Active.UserData.X>
26 <Active.UserData.X_Units>s</Active.UserData.X_Units>
27 <Active.UserData.Y>1131.6088321268292</Active.UserData.Y>
28 <Active.UserData.Y_Units>km</Active.UserData.Y_Units>
29 </Log>
30 <Log>
31 <Recording_ID>Vibration Data Analysis_Extract</Recording_ID>
32 <Date_and_Time>2016-07-13T18:54:35</Date_and_Time>
33 <Entry_type>Manual</Entry_type>
34 <Active.UserData.X>134.36520833333336</Active.UserData.X>
35 <Active.UserData.X_Units>s</Active.UserData.X_Units>
36 <Active.UserData.Y>1131.6088319724504</Active.UserData.Y>
37 <Active.UserData.Y_Units>km</Active.UserData.Y_Units>
38 </Log>

```

Fig. 4.40 Log file example

The same log file can also be opened into Microsoft® Excel:



Recording_ID	Date_and_Time	Entry_type	Active.UserData.X	Active.UserData.Y	Active.UserData.Z
1	7/13/2016 18:53	Manual	96.38760417 s	760.3695709 km	
2	7/13/2016 18:54	Manual	98.13666667 s	780.7454874 km	
3	7/13/2016 18:54	Manual	134.364375 s	1131.608832 km	
4	7/13/2016 18:54	Manual	134.3652083 s	1131.608832 km	
5	7/13/2016 18:54	Manual	134.3666667 s	1131.608832 km	
6	7/13/2016 18:54	Manual	150.005 s	1353.75602 km	
7	7/13/2016 18:54	Manual	163.0521875 s	1538.133217 km	
8	7/13/2016 18:54	Manual	163.0551042 s	1538.133217 km	
9	7/13/2016 18:54	Manual	224.3725 s	2239.362932 km	
10	7/13/2016 18:55	Manual	236.2860417 s	2389.945926 km	
11	7/13/2016 18:55	Manual	296.566875 s	3051.417755 km	
12	7/13/2016 18:55	Manual	296.5683333 s	3051.417755 km	
13	7/13/2016 18:55	Manual	296.5695833 s	3051.417755 km	
14	7/13/2016 18:55	Manual	296.5865625 s	3051.417755 km	
15	7/13/2016 18:55	Manual	296.5895833 s	3051.417755 km	
16	7/13/2016 18:55	Manual	323.1585417 s	3426.632805 km	
17	7/13/2016 18:55	Manual	323.161225 s	3426.632805 km	
18	7/13/2016 18:55	Manual	336.2635417 s	3611.506974 km	
19	7/13/2016 18:55	Manual	337.3358333 s	3626.913155 km	
20	7/13/2016 18:55	Manual	337.33875 s	3626.913155 km	
21	7/13/2016 18:55	Manual	347.1726042 s	3765.07181 km	
22	7/13/2016 18:55	Manual	380.456875 s	4234.214865 km	
23	7/13/2016 18:55	Manual	426.2391667 s	4738.643042 km	
24	7/13/2016 18:55	Manual	525.6747917 s	6119.732612 km	
25	7/13/2016 18:55	Manual	525.6752083 s	6119.732608 km	

Fig. 4.41 Microsoft® Excel logfile

As can be seen each logging is a row in the Microsoft® Excel sheet. Via the **ReadLogFile()** function it is now possible to select one of the numerical columns and read the column data into Perception as a waveform data source. The selection is done by the second parameter the so called Fieldname; for example: *Active.UserData.X*

It is also possible to use *UserData.X* or even *X*; but if the field name is abbreviated, take care that the abbreviation is unique and does not conflict with other field names.

The **ReadLogFile()** function always compares the abbreviated name with the ending part of the original field name, therefore *X* is correct for *Active.UserData.X*, however *UserData* is incorrect and will result in an empty waveform.

When the logging rows are not equidistant in time, therefore the X-Units of the new created waveform is not set to time but is left empty.

Use for example “Setpoints” if a logging was done for each set-point whatever this might be. If there are however time equidistant loggings then you can use the SampleInterval to set the time between the loggings and set the X-Units to s.

Example

```
Y = @ReadLogFile("C:\Users\Public\Documents
\SharedLogfiles\VibrationLogfile2.XML";
"Active.UserData.Y")
X = @ReadLogFile("C:\Users\Public\Documents
\SharedLogfiles\VibrationLogfile2.XML"; "X")
```

The new X and Y waveforms can be shown in the Perception Y-t display or in an X-Y display (see Fig. 4.42).

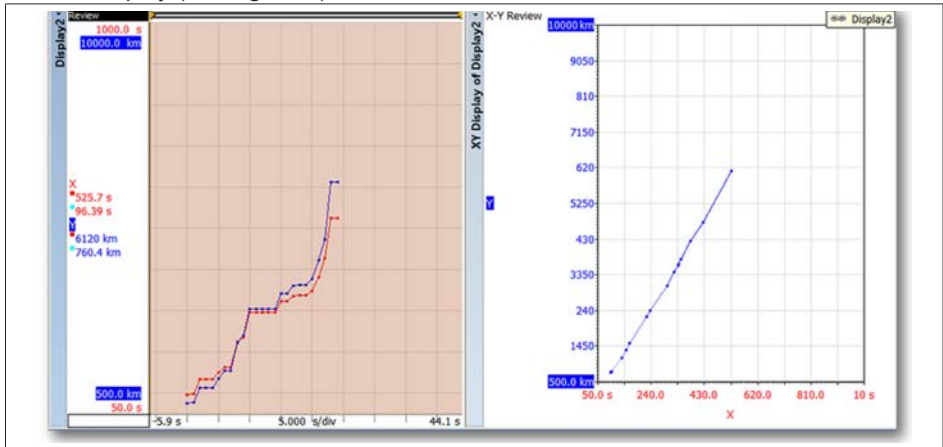


Fig. 4.42 X and Y waveforms in Perception

See also

"@ReadAsciiFile" on page 199

4.81 @Reduce

Function

Reduces the number of samples in a waveform by resampling.

Syntax

`@Reduce(Waveform; Factor)`

Parameters

<i>Waveform</i>	Waveform to be resampled.
<i>Factor</i>	Number: resampling factor

Output

Resampled waveform

Description

A waveform is resampled to reduce the amount of data. The resampling factor is rounded to the nearest integer N and should be in range 2, 3, ..., (waveform length)/2. The output waveform receives sample 1, sample $1+N$, sample $1+2N$, etc. until no more samples are available. The length of the output waveform is N times smaller than as the length of the input waveform. The sampling interval of the output waveform is N times the sampling interval of the input waveform. The x-coordinate of the first sample is not changed.

Resampling can cause aliasing effects when high frequency components are present in the waveform. In this case, use the smoothing function `@Smooth` for lowpass filtering of the data before resampling.

Example

The following example resamples a sine wave by a factor of 10, effectively lowering the sampling rate by a factor of 10. The number of samples is reduced from 1000 to 100 samples.

```
Sine1000 = @SineWave(10k; 1000; 50)
Sine100  = @Reduce(Formula.Sine1000; 10)
```

See also

"@Res2" on page 213 and "@Smooth" on page 222

4.82 @RefCheck

Function

Compares a waveform or waveform segment against one or two reference waveforms.

Syntax

```
@RefCheck(Waveform; UpperEnvelope)
@RefCheck(Waveform; UpperEnvelope; LowerEnvelope)
@RefCheck(Waveform; UpperEnvelope; LowerEnvelope; Begin)
@RefCheck(Waveform; UpperEnvelope; LowerEnvelope; Begin; End)
```

Parameters

<i>Waveform</i>	Input waveform to be compared.
<i>UpperEnvelope</i>	Waveform containing the upper envelope.
<i>LowerEnvelope</i>	Waveform containing the lower envelope.
<i>Begin</i>	Number: begin position of segment for comparison.
<i>End</i>	Number: end position of segment for comparison.

Output

Number indicating success (between the reference waveforms) or failure.

Description

This function is used to compare a waveform against one or two reference waveforms. The output is the x position of the first level crossing of the input waveform with the Upper or Lower waveform.

If the input waveform is between the upper and lower waveform then the return value will be a "Not-a-number" (Nan) value.

The segment limits (Begin and End) are used to select a range of samples. If no segment limits are specified, the complete waveform is used.

When only Begin is specified, the waveform segment from Begin to the end of the waveform is used.

Example

The following example creates a sine wave and two 'envelope' sine waves. The noise is used to create a success or failure. A higher noise value will result in failure.

```
Signal  = @SineWave(8000; 8001; 5)
Upper   = Formula.Signal + 0.1
Lower   = Formula.Signal - 0.1
Noise   = 0.1 * @Noise(8000; 8001)
Signal2  = Formula.Signal + Formula.Noise
Check    = @RefCheck(Formula.Signal2; Formula.Upper;
                  Formula.Lower)
Result   = @IIF(Formula.Check; "NOT OK"; "OK")
```

4.83 @RelativeTime2Local

Function

Returns a string representing the absolute local time corresponding to the time elapsed since the start of the signal.

Syntax

@RelativeTime2Local(*Signal*; *Rel Time*)

@RelativeTime2Local(*Signal*; *Rel Time*; *Decimals*)

@RelativeTime2Local(*Signal*; *Rel Time*; *Decimals*; *DateFormat*)

Parameters

<i>Signal</i>	Input waveform
<i>Rel Time</i>	Number: input time elapsed since start (e.g. cursor X value) in seconds.
<i>Decimals</i>	Number: number of decimals to display the fractional part of the seconds. Default 3. Number has to be between 0 and 9.
<i>DateFormat</i>	String: date format: "None", "Short" or "Long". Default "None" means no date information.

Output

A string representing the absolute local time corresponding to the time elapsed since the start of the signal.

The output can look like:

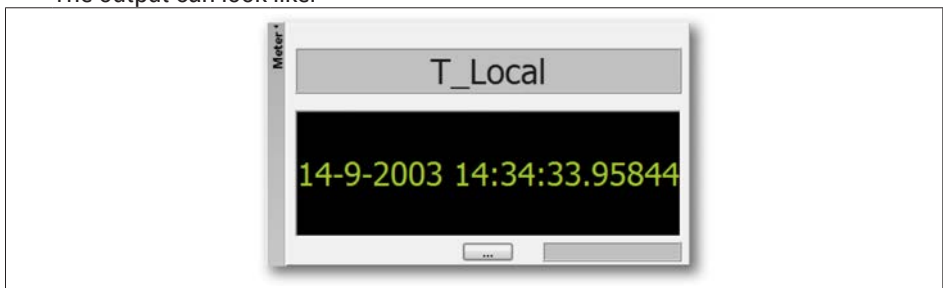


Fig. 4.43 @RelativeTime2Local - output

Description

This function is used to determine the absolute local time of a data sample. This function is often used to determine the absolute time at the cursor position.

Example

This example shows the absolute local time of the X-location of cursor 1.

```
T_Local = @RelativeTime2Local  
        (Active.Group1.Recorder_A.Input_shaft_vib;  
         Display.Display.Cursor1.XPosition; 5; "Short")
```

See also

"@RelativeTime2UTC" on page 210

4.84 @RelativeTime2UTC

Function

Returns a string representing the absolute UTC time corresponding to the time elapsed since the start of the signal.

Syntax

@RelativeTime2UTC(*Signal*; *Rel Time*)

@RelativeTime2UTC(*Signal*; *Rel Time*; *Decimals*)

@RelativeTime2UTC(*Signal*; *Rel Time*; *Decimals*; *DateFormat*)

Parameters

<i>Signal</i>	Input waveform
<i>Rel Time</i>	Number: input time elapsed since start (e.g. cursor X value) in seconds.
<i>Decimals</i>	Number: number of decimals to display the fractional part of the seconds. Default 3. Number has to be between 0 and 9.
<i>DateFormat</i>	String: date format: "None", "Short" or "Long". Default "None" means no date information.

Output

A string representing the absolute UTC time corresponding to the time elapsed since the start of the signal.

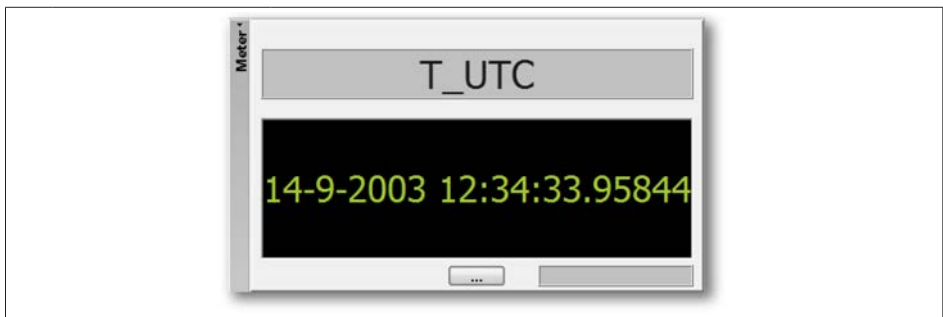


Fig. 4.44 @RelativeTime2UTC - Output

Description

This function is used to determine the absolute UTC time of a data sample. This function is often used to determine the absolute time at the cursor position.

Example

This example shows the absolute local time of the X-location of cursor 1.

```
T.UTC = @RelativeTime2UTC
      (Active.Group1.Recorder_A.Input_shaft_vib;
       Display.Display.Cursor1.XPosition; 5;
       "Short" )
```

See also

"@RelativeTime2Local" on page 208

4.85 @RemoveGlitch

Function

Returns a waveform with new data between a specified start and end time by using a straight line between those two points.

Syntax

@RemoveGlitch(Waveform; Begin; End)

Parameters

Waveform	Input waveform for which the glitch is to be removed.
Begin	Number: glitch start time
End	Number: glitch end time

Output

The output is a waveform with the same length as the original waveform.

Description

The resulting waveform creates a straight line between the Begin and End times. The rest of the original waveform remains unchanged.

Notice

If the formula is used to remove a glitch while recording, then the Begin and End parameters must be within the bounds of the available data. If either the Begin or End time does not have a corresponding data value in the waveform, then the RemoveGlitch function cannot be calculated.

Example

The following example creates a 50 Hz sine wave with duration of 2 seconds. Then the function creates a new 2-second-long waveform, where the samples in the interval between 1 second and 1.5 seconds are replaced with a straight line

```
Signal = @SineWave(1k; 2000; 50)
Cropped = @RemoveGlitch(Formula.Signal; 1000m; 1500m)
```

4.86 @Res2

Function

Resamples a waveform so that its length becomes a power of two.

Syntax

@Res2(*Waveform*)

Parameters

Waveform Waveform to be resampled.

Output

Resampled waveform

Description

This function resamples a waveform so that the new number of samples is equal to the next power of two greater than or equal to the original length. This function can be used effectively for calculating spectra of any desired segment of a waveform. The number of samples in such a segment will typically not be equal to a power of two, which is required by an FFT algorithm. By using the @Res2 function on this segment first, a new waveform is created containing the same signal but sampled at a higher sampling rate in such a way that the number of samples will be a power of two. This resampled waveform can be used for FFT calculations without the need to truncate or zero-pad the waveform.

Because the resampling factor will typically not be an integer, values of the waveform between two samples have to be generated. This is done using linear interpolation. The sampling interval of the waveform is corrected for the resampling factor.

Example

The following example generates a sine wave of 2500 samples and resamples it to a waveform with 4096 samples. The value of Length will be 4096.

```
Signal      = @SineWave(10k; 2500; 100)
Resampled   = @Res2(Formula.Signal)
Length      = @Length(Formula.Resampled)
```

See also

"@Reduce" on page 205

4.87 @RiseTime

Function

Determines the **rise time** of a pulse in a waveform.

Syntax

@RiseTime(*Waveform*)

@RiseTime(*Waveform*; *Begin*)

@RiseTime(*Waveform*; *Begin*; *End*)

Parameters

<i>Waveform</i>	Input waveform containing the leading edge of a pulse.
<i>Begin</i>	Number: segment begin
<i>End</i>	Number: segment end

Output

The output is a numerical value.

This function works “while recording” when the *Begin* and *End* parameters are set. The result will be calculated when the data between *Begin* and *End* is available.

Description

The rise time is determined by taking the time difference between the proximal point (10% magnitude transition) and the distal point (90% magnitude transition) on the first leading edge of a pulse in the waveform (or waveform segment).

Magnitude percentage crossings are calculated using the base and top magnitudes. The base and top magnitudes are determined by searching two dominant populations in the waveform (or waveform segment) magnitude histogram and taking the mean values of these two populations.

The segment limits (*Begin* and *End*) are used to select a range of samples. If no segment limits are specified, the complete waveform is used. When only *Begin* is specified, the waveform segment from *Begin* to the end of the waveform is used.

Notice

The **Begin** and **End** parameters have to be specified in the units of the horizontal axis (for example time) and not in samples.

Example

The following example creates a pulse and calculates the 10%-90% rise time of the leading edge. The result is 40 ms:

```
Sig1    = @Ramp(1k; 100; 0; 0)
Sig2    = @Ramp(1k; 51; 0; 10)
Sig3    = @Ramp(1k; 100; 10; 10)
Sig4    = @Ramp(1k; 51; 10; 0)
Signal  = @Join(Formula.Sig1; Formula.Sig2;
                Formula.Sig3; Formula.Sig4; Formula.Sig1)
Rise    = @RiseTime(Formula.Signal)
```

See also

"@FallTime" on page 91 and "@PulseWidth" on page 195

4.88 @RMS

Function

Calculates the **Root Mean Square** of a waveform.

Syntax

@RMS(Waveform)

@RMS(Waveform; Begin)

@RMS(Waveform; Begin; End)

Parameters

<i>Waveform</i>	Input waveform for which the RMS is to be calculated.
<i>Begin</i>	Number: segment begin
<i>End</i>	Number: segment end

Output

The output is a numerical value.

Description

The RMS is calculated using the following formula:

$$\text{RMS} = \sqrt{\frac{1}{N} \sum_{n=n_1}^{n_2} y^2(n)} \text{ with } N = (n_2 - n_1 + 1)$$

The segment limits (Begin and End) are used to select a range of samples. If no segment limits are specified, the complete waveform is used. When only Begin is specified, the waveform segment from Begin to the end of the waveform is used.

Notice

The **Begin** and **End** parameters have to be specified in the units of the horizontal axis (for example time) and not in samples.

Example

The following example creates a 10 V amplitude sine wave of 50 Hz and calculates the RMS value using the complete waveform:

4 REFERENCE GUIDE

```
Signal    = 10.0 * @SineWave(20k; 1000; 50)
RMSAmpl   = @RMS(Formula.Signal)
```

See also

"@Energy" on page 85, "@Mean" on page 162 and "@ StdDev" on page 233

4.89 @SAEJ211Filter

Function

Filters the input signal in accordance to the **SAE J211** standard.

Syntax

@SAEJ211Filter(*Signal*; *CFC*)

Parameters

<i>Signal</i>	Input waveform
<i>CFC</i>	Number. Channel Frequency Class

Output

The output is the filtered waveform.

Description

The function performs a digital lowpass filtering to eliminate high frequency noise from raw data. The filter is a 4-pole phase-less lowpass Butterworth filter. The filter type complies with the SAE J211 standard that is widely used in the automotive industry.

The relation between the cut-off frequency (F_c) and the Channel Frequency Class (CFC) is defined as follows:

$$F_c = CFC * 1.67$$

The cut off frequency should be not less than one-quarter of the sampling frequency.

Example

The following example generates a sine wave with some high frequency noise. The signal is filtered with the SAE J211 filter function:

```
Signal      = 5 * @SineWave(80000; 1000; 400)
              + @Noise(80000; 1000)
Filtered    = @SAEJ211Filter(Formula.Signal; 300)
```

The cutoff frequency is: $F_c = 1.67 * 300 = 500$ Hz

4.90 @Sin

Function

Calculates the **sine** of the input parameter.

Syntax

@Sin(*Par*)

Parameters

Par Input waveform or numerical value.

Output

Waveform or numerical value containing the sine of the input.

Description

The trigonometric function sine is calculated assuming the input parameter is the angle in radians. If a waveform parameter is used, the sine is calculated for each individual sample.

Example

The following example calculates the sine of the variable "Angle" specified in degrees:

```
Angle      = 36
AngleRad   = System.Constants.Pi * Formula.Angle / 180
SinAngle   = @Sin(Formula.AngleRad)
```

See also

"@ATan" on page 62, "@Cos" on page 72 and "@Tan" on page 242

4.91 @SineWave

Function

Generates a waveform described by the **sine** function.

Syntax

`@SineWave(FSampling; NSamples; FSignal)`

Parameters

<i>FSampling</i>	Number: sampling frequency
<i>NSamples</i>	Number of samples
<i>FSignal</i>	Number: signal frequency

Output

Waveform containing a sine wave.

Description

This function creates a sine wave. The sampling frequency and number of samples that generate a sine wave frequency can be specified. There is no limit on the length of the waveform. The amplitude of the sine wave is 1 V.

You can use this function to synthesize a variety of waveforms. This data can be used as input for other analysis functions.

Example

The following example generates a 100 ms waveform segment containing a simulated 50 Hz sine wave of 10 V amplitude, sampled at 10 kHz with 4 V peak-to-peak noise added:

```
Signal = 10 * @SineWave(10k; 1000; 50) + 2 *  
         @Noise(10k; 1000)
```

Or you can add multiple sine waves for more complex waveforms:

```
Signal2 = 5 * @SineWave(10k; 1000; 50)
          + 7 * @SineWave(10k; 1000; 60)
          + 12 * @SineWave(10k; 1000; 90)
```

See also

"@Noise" on page 178, "@Pulse" on page 193, "@Ramp" on page 197 and
"@SquareWave" on page 232

4.92 @Smooth

Function

Smoothing a waveform using a moving average filter.

Syntax

@Smooth(*Waveform*; *N*)

Parameters

<i>Waveform</i>	Waveform to be smoothed.
<i>N</i>	Number of samples of the smoothing factor.

Output

Smoothed waveform

Description

The waveform is smoothed by taking a moving unweighted average over the specified number of samples according to the following formula:

$$\text{Output}(n) = \frac{1}{N} \sum_{k=n-N'}^{k=n+N'} \text{Input}(k)$$

With:

N = Smoothing factor

N' = (N-1)/2

n = Sample number in range 1 to 'end-of-waveform'

The smoothing factor N controls the number of samples used for building the average. If N is not an odd integer, it will be rounded to the nearest odd integer first. If the value is smaller than 3, it is set to 3. If the value is larger than 1001, it is set to 1001. A higher value of N produces a more pronounced smoothing effect.

Smoothing effectively filters out high frequency components. The best filtering is achieved if the number of samples is equal to some integer multiple of the period of the disturbances. Due to the symmetric nature of the smoothing the position of characteristic features of the waveform is not changed.

The length of the output waveform is the same as the length of the input waveform. For smoothing near the edges, the function assumes that the waveform is extended with the same data values as the edge.

If a triangular weighted smoothing is required, this can be achieved by smoothing the waveform twice. To suppress only high frequency disturbances, it is often more effective to use the @Smooth function several times using a smaller number of samples than once using a large number of samples.

Example

The following example extracts the high frequency noise from a 50 Hz sine wave by subtracting the smoothed signal from the original signal:

```
Mains  = 220 * @SineWave(200k; 4096; 50)
        + 10 * @Noise(200k; 4096)
Smo1   = @Smooth(Formula.Mains; 55)
Noise  = Formula.Mains - Formula.Smo1
```

See also

"@Integrate" on page 142

4.93 @SpaceVectorInverseTransformation

Function

Generates a Waveform of the required **Space Vector Inverse Transformation** component for either amplitude-invariant or power-invariant transformations.

Syntax

@SpaceVectorInverseTransformation(*Alpha*; *Beta*; *Zero(System)*;
Component; *Transformation*)

Parameters

<i>Alpha</i>	Input waveform of the alpha signal.
<i>Beta</i>	Input waveform of the beta signal.
<i>Zero(System)</i>	Input waveform or Value [rad] of the zero(system) signal.
<i>Component</i>	Required output Component (0="signal1", 1="signal2", 2="signal3", 3="magnitude", 4="angle[rad]")
<i>Transformation</i>	Invariant Inverse Transformation Rule (0="amplitude", 1="power")

Output

Waveform of the required component and transformation.

Description

The Space Vector Transformation (also known as the Clarke transformation) is a mathematical transformation that is used to simplify the analysis of three-phase power systems. After applying the Space Vector Transformation to the three-phases circuits to obtain the alpha, beta & zero(system) component, the Space Vector Inverse Transformation can be applied to these resulting signals to reobtain the original input signals of the thee-phases.

$$\begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix} = \frac{3}{2} \begin{bmatrix} 2/3 & 0 & 2/3 \\ -1/3 & \sqrt{3}/3 & 2/3 \\ -1/3 & -\sqrt{3}/3 & 2/3 \end{bmatrix} \begin{bmatrix} I_{A\alpha} \\ I_{A\beta} \\ I_{A0} \end{bmatrix}$$

Waveforms for the magnitude and direction are defined as:

$$I_{Am} = \sqrt{I_{A\alpha}^2 + I_{A\beta}^2}$$

$$I_{A\angle} = \text{atan2}(I_{A\beta}, I_{A\alpha})$$

And for the power-invariant transformation, the following formula is applied:

$$\begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix} = \sqrt{\frac{3}{2}} \begin{bmatrix} 2/3 & 0 & \sqrt{2}/3 \\ -1/3 & 1/\sqrt{3} & \sqrt{2}/3 \\ -1/3 & -1/\sqrt{3} & \sqrt{2}/3 \end{bmatrix} \begin{bmatrix} I_{P\alpha} \\ I_{P\beta} \\ I_{P\theta} \end{bmatrix}$$

Waveforms for the magnitude and direction are defined as:

$$I_{pm} = \sqrt{I_{P\alpha}^2 + I_{P\beta}^2}$$

$$I_{p\angle} = \text{atan2}(I_{P\beta}, I_{P\alpha})$$

Where	
I_1	Component of the phase1 signal
I_2	Component of the phase2 signal
I_3	Component of the phase3 signal
I_m	Magnitude
I_{\angle}	Direction
$I_{A\alpha}$	The amplitude-invariant "alpha" input value
$I_{A\beta}$	The amplitude-invariant "beta" input value
$I_{A\theta}$	The amplitude-invariant "zero(system)" input value
I_{Am}	The amplitude-invariant "magnitude"
$I_{A\angle}$	The amplitude-invariant "direction" (in radians)
$I_{P\alpha}$	The power-invariant "alpha" input value
$I_{P\beta}$	The power-invariant "beta" input value
$I_{P\theta}$	The power-invariant "zero(system)" input value
I_{pm}	The power-invariant "magnitude"
$I_{p\angle}$	The power-invariant "direction" (in radians)

Notes:

- The Transformation parameter is optional and has the default value 0="amplitude".
- The input waveforms (Alpha, Beta & Zero(System)) all required the same sampling frequency.
- This function is only available in the Formula database sheet (see Fig. 1.2 on page 12) if the eDrive option (1-PERC-OP-EDR) is part of your license.

Example

Given that the three-phase input signals are available as *Formula.SVATa*, *Formula.SVATb*, *Formula.SVATz*, the following Space Vector amplitude transformation waveforms can be defined:

Name	Formula
Phase1	= @SpaceVectorInverseTransformation(Formula.SVATa; Formula.SVATb; Formula.SVATz; 0; 0)
Phase2	= @SpaceVectorInverseTransformation(Formula.SVATa; Formula.SVATb; Formula.SVATz; 1; 0)
Phase3	= @SpaceVectorInverseTransformation(Formula.SVATa; Formula.SVATb; Formula.SVATz; 2; 0)
Mag	= @SpaceVectorInverseTransformation(Formula.SVATa; Formula.SVATb; Formula.SVATz; 3; 0)
Dir	= @SpaceVectorInverseTransformation(Formula.SVATa; Formula.SVATb; Formula.SVATz; 4; 0)

Likewise the power-invariant variations are:

Name	Formula
Phase1	= @SpaceVectorInverseTransformation(Formula.SVPTa; Formula.SVPTb; Formula.SVPTz; 0; 1)
Phase2	= @SpaceVectorInverseTransformation(Formula.SVPTa; Formula.SVPTb; Formula.SVPTz; 1; 1)
Phase3	= @SpaceVectorInverseTransformation(Formula.SVPTa; Formula.SVPTb; Formula.SVPTz; 2; 1)
Mag	= @SpaceVectorInverseTransformation(Formula.SVPTa; Formula.SVPTb; Formula.SVPTz; 3; 1)
Dir	= @SpaceVectorInverseTransformation(Formula.SVPTa; Formula.SVPTb; Formula.SVPTz; 4; 1)

The Zero(System) input parameter can also be specified as a value. Usually this waveform will have a constant 0 (zero) value. For example:

Name	Formula
Phase1	<code>= @SpaceVectorInverseTransformation(Formula. SVPTa; Formula.SVPTb; 0; 0; 1)</code>

See also
"@SpaceVectorTransformation" on page 228

4.94 @SpaceVectorTransformation

Function

Generates a Waveform for the required **Space Vector Transformation** component for either amplitude-invariant or power-invariant transformations.

Syntax

@SpaceVectorTransformation(*Signal1*; *Signal2*; *Signal3*; *Component*; *Transformation*)

Parameters

<i>Signal1</i>	Input waveform of phase1 signal.
<i>Signal2</i>	Input waveform of phase2 signal.
<i>Signal3</i>	Input waveform of phase3 signal.
<i>Component</i>	Required output Component (0="alpha", 1="beta", 2="zero(system)", 3="magnitude", 4="angle[rad]")
<i>Transformation</i>	Invariant Transformation Rule (0="amplitude", 1="power")

Output

The waveform of the required component and transformation.

Description

The Space Vector Transformation (also known as the Clarke transformation) is a mathematical transformation that is used to simplify the analysis of three-phase power systems.

The Vector Space Transformation is defined by the following matrix and can be applied for any three-phase quantities (e.g. voltage, currents, etc.)

The following formula is applied for amplitude-invariant transformation:

$$\begin{bmatrix} I_{A\alpha} \\ I_{A\beta} \\ I_{A0} \end{bmatrix} = \frac{2}{3} \begin{bmatrix} 1 & -1/2 & -1/2 \\ 0 & \sqrt{3}/2 & -\sqrt{3}/2 \\ 1/2 & 1/2 & 1/2 \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix}$$

Waveforms for the magnitude and direction are defined as:

$$I_{Am} = \sqrt{I_{A\alpha}^2 + I_{A\beta}^2}$$

$$I_{A\angle} = \text{atan2}(I_{A\beta}, I_{A\alpha})$$

And for the power-invariant transformation, the following formula is applied:

$$\begin{bmatrix} I_{P\alpha} \\ I_{P\beta} \\ I_{P\emptyset} \end{bmatrix} = \sqrt{\frac{2}{3}} \begin{bmatrix} 1 & -1/2 & -1/2 \\ 0 & \sqrt{3}/2 & -\sqrt{3}/2 \\ 1/\sqrt{2} & 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix}$$

Waveforms for the magnitude and direction are defined as:

$$I_{pm} = \sqrt{I_{P\alpha}^2 + I_{P\beta}^2}$$

$$I_{p\angle} = \text{atan2}(I_{P\beta}, I_{P\alpha})$$

Where	
I_1	Value of the phase1 signal
I_2	Value of the phase2 signal
I_3	Value of the phase3 signal
$I_{A\alpha}$	The amplitude-invariant "alpha" component
$I_{A\beta}$	The amplitude-invariant "beta" component
$I_{A\emptyset}$	The amplitude-invariant "zero(system)" component
I_{Am}	The amplitude-invariant "magnitude"
$I_{A\angle}$	The amplitude-invariant "direction" (in radians)
$I_{P\alpha}$	The power-invariant "alpha" component
$I_{P\beta}$	The power-invariant "beta" component
$I_{P\emptyset}$	The power-invariant "zero(system)" component
I_{Pm}	The power-invariant "magnitude"
$I_{P\angle}$	The power-invariant "direction" (in radians)

Notes:

- The Transformation parameter is optional and has the default value 0="amplitude".
- The input waveforms (phase1, phase2 & phase3) all required the same sampling frequency.
- This function is only available in the Formula database sheet (see Fig. 1.2 on page 12) if the eDrive option (1-PERC-OP-EDR) is part of your license.

Example

Given that the three-phase input signals are available as *Formula.Phase1*, *Formula.Phase2*, *Formula.Phase3*, the following Space Vector amplitude transformation waveforms can be defined:

Name	Formula
SVATa	= @SpaceVectorTransformation(Formula.Phase1; Formula.Phase2; Formula.Phase3; 0; 0)
SVATb	= @SpaceVectorTransformation(Formula.Phase1; Formula.Phase2; Formula.Phase3; 1; 0)
SVATz	= @SpaceVectorTransformation(Formula.Phase1; Formula.Phase2; Formula.Phase3; 2; 0)
SVATm	= @SpaceVectorTransformation(Formula.Phase1; Formula.Phase2; Formula.Phase3; 3; 0)
SVATd	= @SpaceVectorTransformation(Formula.Phase1; Formula.Phase2; Formula.Phase3; 4; 0)

Likewise the power-invariant variations are:

Name	Formula
SVPTa	= @SpaceVectorTransformation(Formula.Phase1; Formula.Phase2; Formula.Phase3; 0; 1)
SVPTb	= @SpaceVectorTransformation(Formula.Phase1; Formula.Phase2; Formula.Phase3; 1; 1)
SVPTz	= @SpaceVectorTransformation(Formula.Phase1; Formula.Phase2; Formula.Phase3; 2; 1)
SVPTm	= @SpaceVectorTransformation(Formula.Phase1; Formula.Phase2; Formula.Phase3; 3; 1)
SVPTd	= @SpaceVectorTransformation(Formula.Phase1; Formula.Phase2; Formula.Phase3; 4; 1)

See also

"@SpaceVectorInverseTransformation" on page 224 and "@DQ0Transformation" on page 81

4.95 @Sqrt

Function

Calculates the **square root** of the given parameter.

Syntax

`@Sqrt(Par)`

Parameter

Par Input waveform or numerical value.

Output

Waveform or numerical value containing the square root of the input.

Description

This function calculates the square root of the input parameter.

If a waveform parameter is used, the square root function is calculated for each individual sample. Negative values in a waveform will result in the negative square root of the absolute value. For numerical values the square root of a negative value does not exist.

Example

Examples of possible square root operations are:

Signal = 4 * @SineWave(10k; 1000; 50)

Result = @Sqrt(Formula.Signal)

Five = @Sqrt(25)

NoValue = @Sqrt(-1)

See also

"@Pow" on page 186

4.96 @SquareWave

Function

Generates a waveform containing a **square wave**.

Syntax

@SquareWave(*FSampling*; *NSamples*; *NFrequency*)

Parameters

<i>FSampling</i>	Number: sample frequency of the waveform.
<i>NSamples</i>	Number of samples in the waveform.
<i>NFrequency</i>	Number: frequency of the generated square wave.

Output

Waveform containing a square wave.

Description

This function creates a square wave. The sampling frequency, number of samples to generate and square wave frequency can be specified. There is no limit to the length of the waveform. The amplitude of the square wave is 1 V.

The possibility to generate a square wave function is used to synthesize a variety of waveforms. The simulated data can be used as input for other analysis functions.

Example

The following example generates a 100 ms waveform segment containing a simulated 50 Hz square wave of 10 V amplitude, sampled at 10 kHz with 2 V peak-to-peak noise added:

```
Signal = 10 * @SquareWave(10k;1000;50)
        + 2 * @Noise(10k;1000)
```

See also

"@Noise" on page 178, "@Pulse" on page 193, "@Ramp" on page 197 and "@SineWave" on page 220

4.97 @ StdDev

Function

Calculates the **standard deviation** of a waveform.

Syntax

@StdDev(*Waveform*)

@StdDev(*Waveform*; *Begin*)

@StdDev(*Waveform*; *Begin*; *End*)

Parameters

<i>Waveform</i>	Input waveform for which the standard deviation is to be calculated.
<i>Begin</i>	Number: segment begin
<i>End</i>	Number: segment end

Output

The output is a numerical value.

Description

The standard deviation is a statistical characteristic of a waveform. It is approximately the root means square deviation of the samples in the waveform from the mean value of the waveform: a measure of the spread of its values.

The standard deviation is calculated using the following formula:

$$\text{StdDev} = \sqrt{\frac{1}{N-1} \sum_{n=n_1}^{n_2} (y(n) - \bar{y})^2}$$

$$\text{in which: } \bar{y} = \frac{1}{N} \sum_{n=n_1}^{n_2} y(n)$$

$$\text{with } N = (n_2 - n_1 + 1)$$

The segment limits (*Begin* and *End*) are used to select a range of samples. If no segment limits are specified, the complete waveform is used.

When only **Begin** is specified, the waveform segment from **Begin** to the end of the waveform is used.

Notice

*The **Begin** and **End** parameters have to be specified in the units of the horizontal axis (for example time) and not in samples.*

Example

The following example creates a sine wave of 50 Hz and an RMS amplitude of 1 on top of a DC component of 100. From this signal the standard deviation is determined:

```
Signal = 100 + @Sqrt(2) * @SineWave(20k; 2000; 50)
StdDev = @StdDev(Formula.Signal)
```

See also

"@Energy" on page 85, "@Mean" on page 162 and "@RMS" on page 216

4.98 @Sweep

Function

Selects a specific **sweep** from a multi-sweep recording.

Syntax

@Sweep(*Signal*; *NSweep*)

Parameters

<i>Signal</i>	Waveform: multi-sweep recording
<i>NSweep</i>	Integer number of the sweep to be selected.

Output

Waveform containing only the selected sweep.

Description

Some recorder hardware and software settings make it possible to make multi-sweep recordings. Each valid trigger event will create a new sweep. The waveform of a channel from a multi-sweep recording contains multiple sweeps.

The @Sweep function makes it possible to select one specific sweep from the waveform. Nsweep is a numerical value in range 1 to number of sweeps.

Notice

The selected waveform contains only one sweep.

Example

This example calculates the maximum value of the pressure in the second sweep of the recording.

```
PressureSweep = @Sweep  
               (Active.Group1.Recorder_A.Ch_A2; 2)  
MaxPressure   = @Max(Formula.PressureSweep)
```

4.99 @SweepEndTime

Function

Returns the **end time** of a specific **sweep** from a multi-sweep signal.

Syntax

`@SweepEndTime(Signal; NSweep)`

Parameters

<i>Signal</i>	Waveform: multi-sweep signal
<i>NSweep</i>	Integer number of the sweep to be selected.

Output

Numerical value representing the end time of the selected sweep.

Description

This function returns the end time of the selected sweep on a multi-sweep waveform in seconds relative to the start of the recording.

Example

In the example below T1 is the end of the 3rd sweep and T2 is the end of the active sweep shown in the display. For the T2 formula the display has to be in ReviewSweep mode otherwise the parameter `Disply.Display.ActiveSweep.Index` is not defined.

```
T1 = @SweepEndTime(Formula.I; 3)
T2 = @SweepEndTime(Formula.I;
    Display.Display.ActiveSweep.Index)
```

See also

"@Sweep" on page 235, "@SweepNumberAtTime" on page 239,
"@SweepSource" on page 238, "@SweepStartTime" on page 239,
"@NumberOfSweeps" on page 180

4.100 @SweepNumberAtTime

Function

Returns the **sweep** index related to a **time** relative to the start of a multi-sweep recording.

Syntax

@SweepNumberAtTime(*Signal*; *Time*)

Parameters

<i>Signal</i>	Waveform: multi-sweep recording
<i>Time</i>	The time relative to the start of the recording.

Output

Numerical integer value representing the sweep index corresponding to the entered time.

Description

This function returns the sweep index corresponding to the entered time. The time is relative to the start of the recording. If no sweep is found at the specified time a zero value will be returned.

Example

In the formula below the returned value SweepN is the sweep index of the sweep which contains data samples at 27.5 seconds after the recording start.

```
SweepN = @SweepNumberAtTime(Formula.I; 27.5)
```

See also

"@Sweep" on page 235, "@SweepEndTime" on page 236, "@SweepSource" on page 238, "@SweepStartTime" on page 239, "@NumberOfSweeps" on page 180

4.101 @SweepSource

Function

Returns the source of the trigger of a specific sweep from a multi-sweep signal.

Syntax

`@SweepSource(Signal; NSweep)`

Parameters

<i>Signal</i>	Waveform: multi-sweep signal
<i>NSweep</i>	Integer number of the sweep to be selected.

Output

The output is a string representing the trigger source of the selected sweep.

Description

This function returns a string which is the trigger source of the selected sweep. The trigger source can be a channel, then the name of the channel is returned. If the trigger source is not a channel then it can also be a manual trigger, external or bus trigger.

Example

The example formula below returns the trigger source of the 3th sweep of the signal Formula.I. The value of the result (TriggerSource) can be e.g.: "Ch A1"

```
TriggerSource = @SweepSource(Formula.I; 3)
```

See also

"@Sweep" on page 235, "@SweepEndTime" on page 236,
"@SweepNumberAtTime" on page 237, "@SweepStartTime" on page 239,
"@NumberOfSweeps" on page 180

4.102 @SweepStartTime

Function

Returns the **start time** of a specific **sweep** from a multi-sweep signal.

Syntax

@SweepStartTime(*Signal*; *NSweep*)

Parameters

<i>Signal</i>	Waveform: multi-sweep signal
<i>NSweep</i>	Integer number of the sweep to be selected.

Output

Numerical value representing the start time of the selected sweep.

Description

This function returns the start time of the selected sweep on a multi-sweep waveform in seconds relative to the start of the recording.

Example

In the example below T1 is the start of the 3rd sweep and T2 is the start of the active sweep shown in the display. For the T2 formula the display has to be in ReviewSweep mode otherwise the parameter `Display.Display.ActiveSweep.Index` is not defined.

```
T1 = @SweepStartTime(Formula.I; 3)
T2 = @SweepStartTime(Formula.I;
    Display.Display.ActiveSweep.Index)
```

See also

"@Sweep" on page 235, "@SweepEndTime" on page 236, "
@SweepNumberAtTime" on page 237, "@SweepSource" on page 238,
"@NumberOfSweeps" on page 180

4.103 @SweptSineWave

Function

This function generates a waveform described by the sine function sweeping from a start frequency to an end frequency.

Syntax

@SweptSineWave(*FSampling*; *NSamples*; *FStart*; *FEnd*)

@SweptSineWave(*FSampling*; *NSamples*; *FStart*; *FEnd*; *Mode*)

@SweptSineWave(*FSampling*; *NSamples*; *FStart*; *FEnd*; *Mode*; *NLeadSamples*)

Parameters

<i>FSampling</i>	Number: sampling frequency
<i>NSamples</i>	Number: number of samples
<i>FStart</i>	Number: start frequency, must be larger than 0.
<i>FEnd</i>	Number: end frequency, must be larger than the start frequency
<i>Mode</i>	Number: the sweep mode, 0 = Logarithmic, 1 = Linear.
<i>NLeadSamples</i>	Number: number of lead samples

Output

Waveform containing a swept sine wave.

Description

This function creates a swept sine wave with an amplitude of 1 V. The sampling frequency and number of samples can be specified.

The generated signal sweeps from start frequency to end frequency. The lead samples come prior to this. Lead samples are useful to prevent windowing effects. The frequency sweep can be either linear or logarithmic. The transition from lead-in to sweep is always at a zero-crossing.

Example

The following example will generate a waveform sampled at 20 kHz with a length of 80 k samples, starting at 1 Hz, logarithmic sweeping to 100 Hz from sample 30 k onwards.


```
Signal = @SweptSineWave(20k; 80k; 1; 100; 0; 30k)
```

The output will look like (see Fig. 4.45):

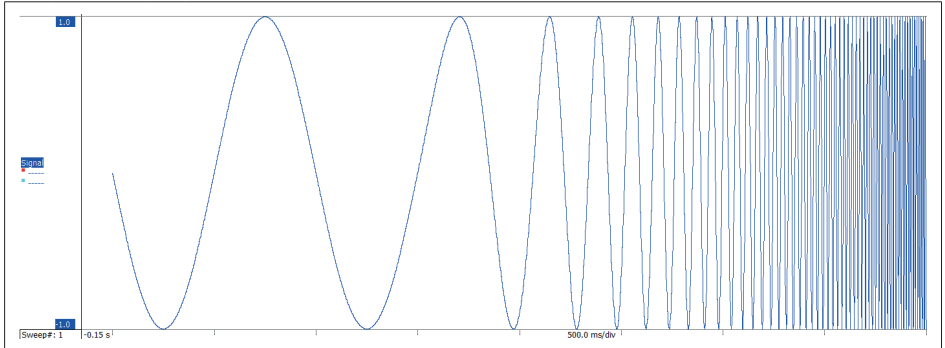


Fig. 4.45 @SweptSineWave - Example 1

The following example will generate a waveform, starting at 1 Hz, linear sweeping to 100 Hz.

```
Signal = @SweptSineWave(20k; 80k; 1; 100; 0)
```

The output will look like (see Fig. 4.46):

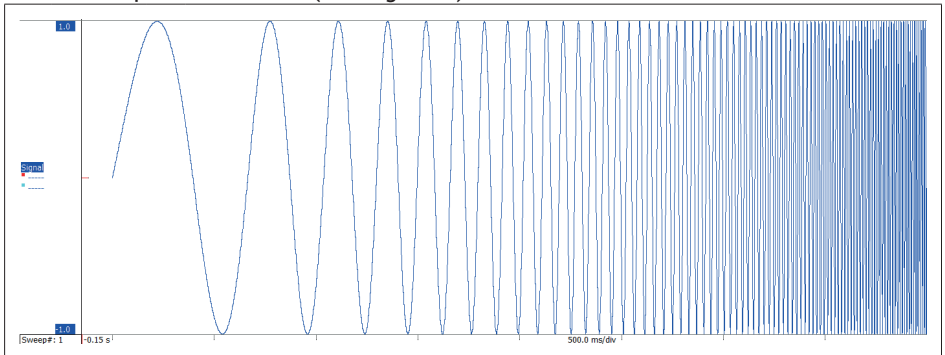


Fig. 4.46 @SweptSineWave - Example 2

See also

"@SineWave" on page 220

4.104 @Tan

Function

Calculates the **tangent** of the input parameter.

Syntax

@Tan(*Par*)

Parameters

Par Input waveform or numerical value.

Output

Waveform or numerical value containing the tangent of the input.

Description

The trigonometric function tangent is calculated assuming the input parameter is the angle in radians. When a waveform parameter is used, the tangent is calculated for each individual sample.

The function @ATan is the inverse trigonometric function of the function @Tan.

Example

The following example calculates the tangent of the variable "Angle" specified in degrees:

```
Angle      = 72
AngleRad   = System.Constants.Pi * Formula.Angle / 180
TanAngle   = @Tan(Formula.AngleRad)
```

See also

"@ATan" on page 62, "@Cos" on page 72 and "@Sin" on page 219

4.105 @TimeMaxAbove

Function

This function returns a number representing the time interval of the longest period that the signal is above a specified level.

Syntax

@TimeMaxAbove(Signal)
@TimeMaxAbove(Signal; Level)
@TimeMaxAbove(Signal; Level; Start)
@TimeMaxAbove(Signal; Level; Start; End)

Parameters

<i>Signal</i>	Input waveform
<i>Level</i>	Number. The used level default value is 0
<i>Start</i>	Number. Start of segment, default begin of signal
<i>End</i>	Number. End of segment, default end of signal

Output

The output is a numerical value representing the time interval of the longest period that the signal is above a specified level.

Description

This function is used when searching for the longest period that a signal is above a specified level. In Fig. 4.47 this is **P1**.

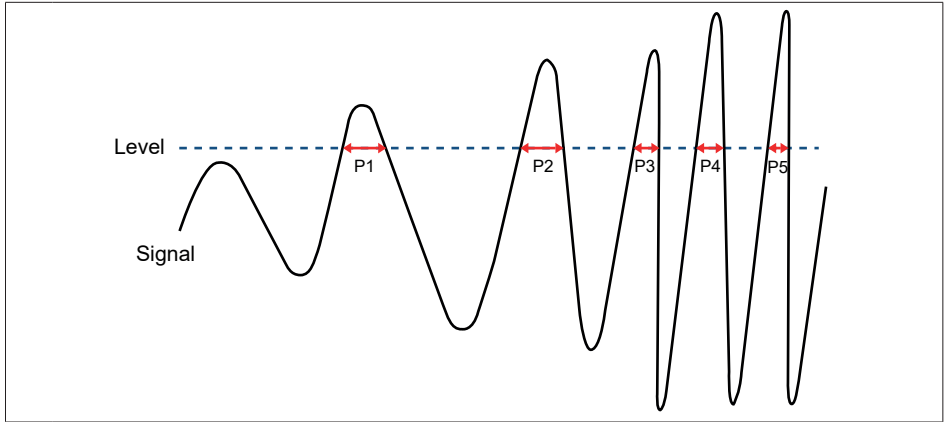


Fig. 4.47 TimeMaxAbove

See also

"@TimeMaxAboveBegin" on page 245, "@TimeMinAbove" on page 251, "@TimeTotalAbove" on page 259 and "@TimeMaxBelow" on page 247

4.106 @TimeMaxAboveBegin

Function

This function returns a number representing the start time of the longest period that the signal is above a specified level.

Syntax

`@TimeMaxAboveBegin(Signal)`
`@TimeMaxAboveBegin(Signal; Level)`
`@TimeMaxAboveBegin(Signal; Level; Start)`
`@TimeMaxAboveBegin(Signal; Level; Start; End)`

Parameters

<i>Signal</i>	Input waveform
<i>Level</i>	Number: The used level default value is 0
<i>Start</i>	Number: Start of segment, default begin of signal
<i>End</i>	Number: End of segment, default end of signal

Output

The output is a numerical value representing the start time of the longest period that the signal is above a specified level

Description

This function is used when you want to know where the longest period that a signal is above a specified level begins. In Fig. 4.48 the longest period is **P1**, the start of **P1** is **T1**, this number will be returned by the @TimeMaxAboveBegin function.

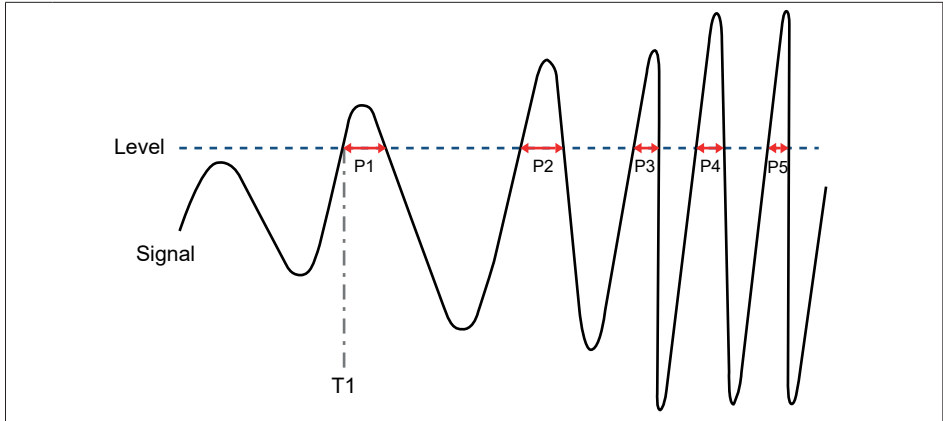


Fig. 4.48 *TimeMaxAboveBegin*

See also

"@TimeMaxAbove" on page 243, "@TimeMinAbove" on page 251, "@TimeTotalAbove" on page 259 and "@TimeMaxBelow" on page 247

4.107 @TimeMaxBelow

Function

This function returns a number representing the time interval of the longest period that the signal is below a specified level.

Syntax

@TimeMaxBelow(Signal)
@TimeMaxBelow(Signal; Level)
@TimeMaxBelow(Signal; Level; Start)
@TimeMaxBelow(Signal; Level; Start; End)

Parameters

<i>Signal</i>	Input waveform
<i>Level</i>	Number. The used level default value is 0
<i>Start</i>	Number. Start of segment, default begin of signal
<i>End</i>	Number. End of segment, default end of signal

Output

The output is a numerical value representing the time interval of the longest period that the signal is below a specified level.

Description

This function is used when searching for the longest period that a signal is below a specified level. In Fig. 4.49 this is **P1**.

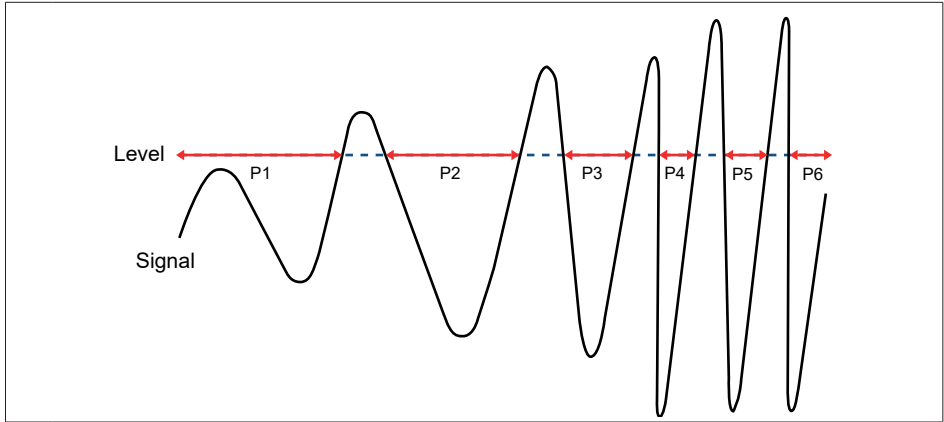


Fig. 4.49 TimeMaxBelow

See also

"@TimeMaxBelowBegin" on page 249, "@TimeMinBelow" on page 255, "@TimeTotalBelow" on page 263 and "@TimeMaxAbove" on page 243

4.108 @TimeMaxBelowBegin

Function

This function returns a number representing the start time of the longest period that the signal is Below a specified level.

Syntax

@TimeMaxBelowBegin(*Signal*)
@TimeMaxBelowBegin(*Signal*; *Level*)
@TimeMaxBelowBegin(*Signal*; *Level*; *Start*)
@TimeMaxBelowBegin(*Signal*; *Level*; *Start*; *End*)

Parameters

<i>Signal</i>	Input waveform
<i>Level</i>	Number: The used level default value is 0
<i>Start</i>	Number: Start of segment, default begin of signal
<i>End</i>	Number: End of segment, default end of signal

Output

The output is a numerical value representing the start time of the longest period that the signal is below a specified level.

Description

This function is used when you want to know where the longest period that a signal is below a specified level begins. In Fig. 4.50 the longest period is **P1**, the start of **P1** is **T1**, this number will be returned by the @TimeMaxBelowBegin function.

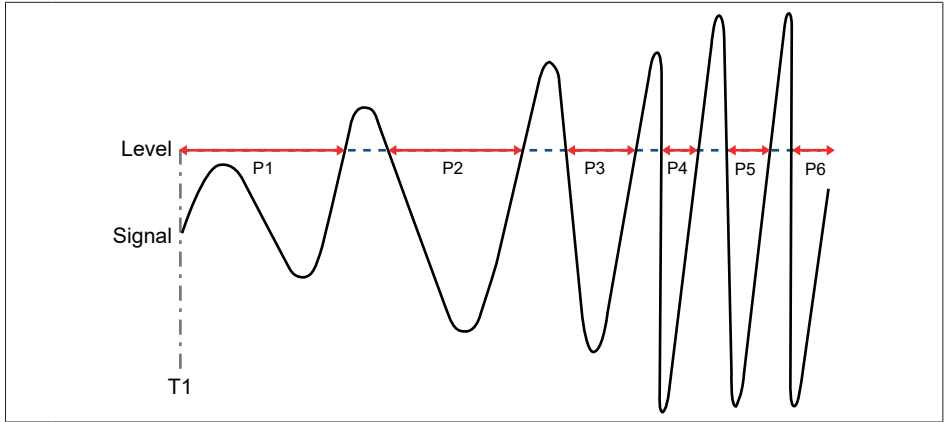


Fig. 4.50 TimeMaxBelowBegin

See also

"@TimeMaxBelow" on page 249, "@TimeMinBelow" on page 255, "@TimeTotal-Below" on page 263 and "@TimeMaxAbove" on page 243

4.109 @TimeMinAbove

Function

This function returns a number representing the time interval of the shortest period that the signal is above a specified level.

Syntax

@TimeMinAbove(*Signal*)
@TimeMinAbove(*Signal*; *Level*)
@TimeMinAbove(*Signal*; *Level*; *Start*)
@TimeMinAbove(*Signal*; *Level*; *Start*; *End*)

Parameters

<i>Signal</i>	Input waveform
<i>Level</i>	Number: The used level default value is 0
<i>Start</i>	Number: Start of segment, default begin of signal
<i>End</i>	Number: End of segment, default end of signal

Output

The output is a numerical value representing the time interval of the shortest period that the signal is above a specified level.

Description

This function is used when searching for the shortest period that a signal is above a specified level. In the picture below this is **P5**.

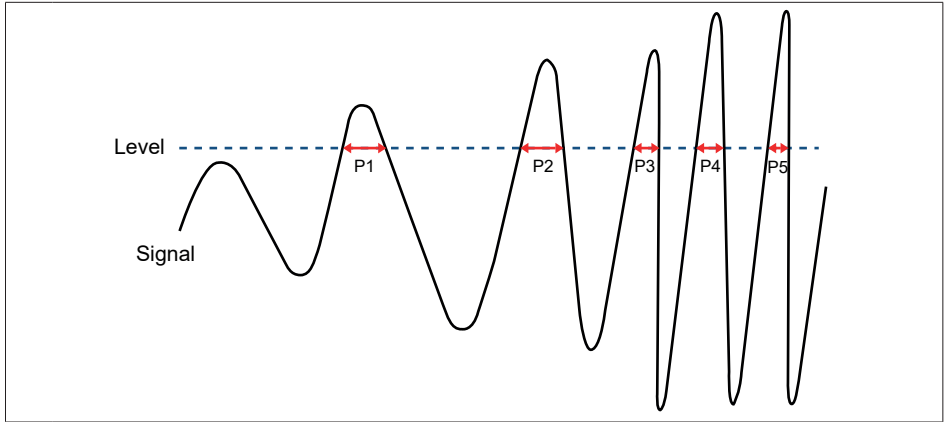


Fig. 4.51 *TimeMinAbove*

See also

"@TimeMinAboveBegin" on page 253, ", "@TimeTotalAbove" on page 259 and
"@TimeMinBelow" on page 255

4.110 @TimeMinAboveBegin

Function

This function returns a number representing the start time of the shortest period that the signal is above a specified level.

Syntax

`@TimeMinAboveBegin(Signal)`
`@TimeMinAboveBegin(Signal; Level)`
`@TimeMinAboveBegin(Signal; Level; Start)`
`@TimeMinAboveBegin(Signal; Level; Start; End)`

Parameters

<i>Signal</i>	Input waveform
<i>Level</i>	Number: The used level default value is 0
<i>Start</i>	Number: Start of segment, default begin of signal
<i>End</i>	Number: End of segment, default end of signal

Output

The output is a numerical value representing the start time of the shortest period that the signal is above a specified level.

Description

This function is used when you want to know where the longest period that a signal is above a specified level begins. In Fig. 4.52 the shortest period is **P5**, the start of **P5** is **T5**, this number will be returned by the @TimeMinAboveBegin function.

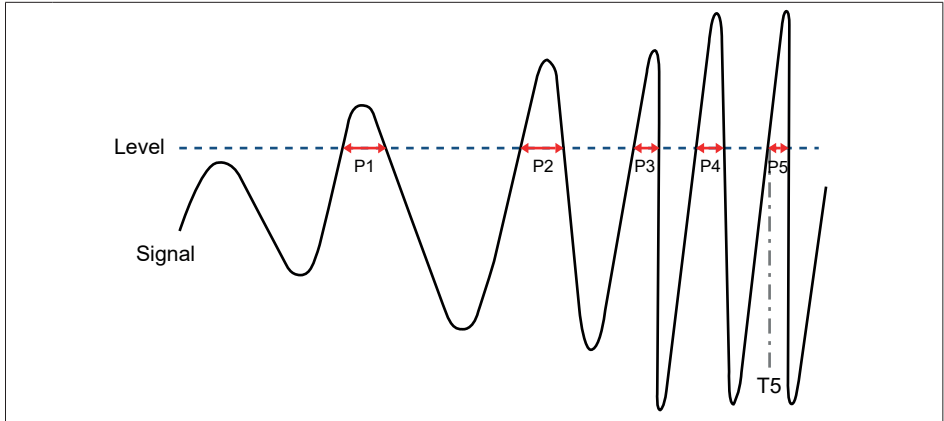


Fig. 4.52 *TimeMinAboveBegin*

See also

"@TimeMinAbove" on page 251, "@TimeTotalAbove" on page 259 and
"@TimeMinBelow" on page 255

4.111 @TimeMinBelow

Function

This function returns a number representing the time interval of the shortest period that the signal is below a specified level.

Syntax

`@TimeMinBelow(Signal)`
`@TimeMinBelow(Signal; Level)`
`@TimeMinBelow(Signal; Level; Start)`
`@TimeMinBelow(Signal; Level; Start; End)`

Parameters

<i>Signal</i>	Input waveform
<i>Level</i>	Number: The used level default value is 0
<i>Start</i>	Number: Start of segment, default begin of signal
<i>End</i>	Number: End of segment, default end of signal

Output

The output is a numerical value representing the time interval of the shortest period that the signal is below a specified level.

Description

This function is used when searching for the shortest period that a signal is below a specified level. In the picture below this is **P4**.

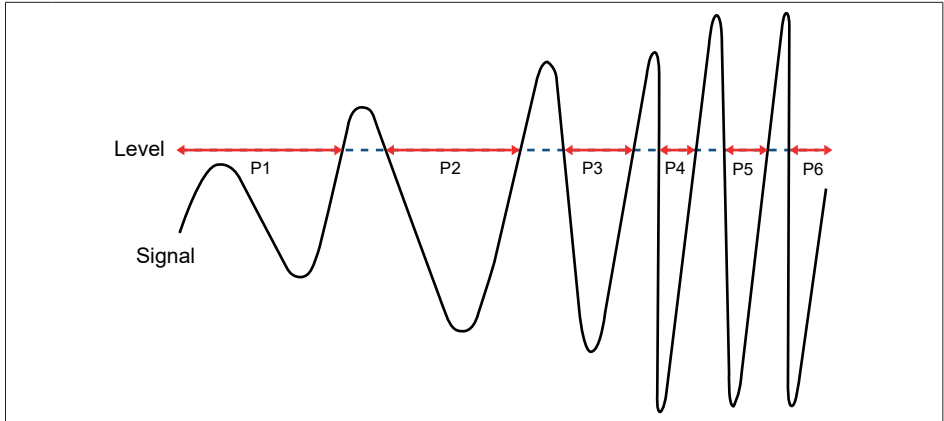


Fig. 4.53 TimeMinBelow

See also

"@TimeMinBelowBegin" on page 257, "@TimeTotalBelow" on page 263 and "@TimeMinBelow" on page 255

4.112 @TimeMinBelowBegin

Function

This function returns a number representing the start time of the shortest period that the signal is above a specified level.

Syntax

`@TimeMinBelowBegin(Signal)`
`@TimeMinBelowBegin(Signal; Level)`
`@TimeMinBelowBegin(Signal; Level; Start)`
`@TimeMinBelowBegin(Signal; Level; Start; End)`

Parameters

<i>Signal</i>	Input waveform
<i>Level</i>	Number: The used level default value is 0
<i>Start</i>	Number: Start of segment, default begin of signal
<i>End</i>	Number: End of segment, default end of signal

Output

The output is a numerical value representing the start time of the shortest period that the signal is above a specified level.

Description

This function is used when you want to know where the longest period that a signal is above a specified level begins. In Fig. 4.54 the shortest period is **P4**, the start of **P4** is **T4**, this number will be returned by the @TimeMinBelowBegin function.

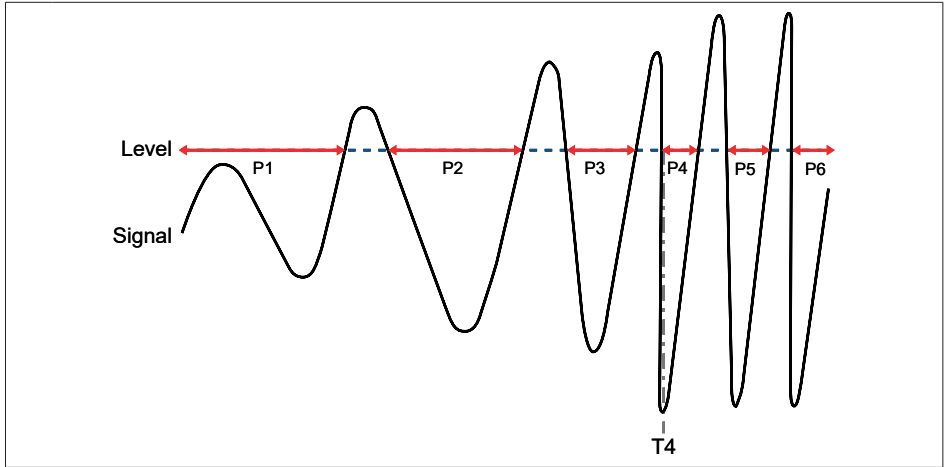


Fig. 4.54 *TimeMinBelowBegin*

See also

"@TimeMinBelow" on page 255 and "@TimeTotalBelow" on page 263

4.113 @TimeTotalAbove

Function

This function returns a number representing the total amount of time that the signal is above a specified level.

Syntax

@TimeTotalAbove(*Signal*)

@TimeTotalAbove(*Signal*; *Level*)

@TimeTotalAbove(*Signal*; *Level*; *Start*)

@TimeTotalAbove(*Signal*; *Level*; *Start*; *End*)

Parameters

Signal Input waveform

Level Number: The used level default value is 0

Start Number: Start of segment, default begin of signal

End Number: End of segment, default end of signal

Output

The output is a numerical value representing the total amount of time that the signal is above a specified level.

Description

This function is used when you want to know how long a signal has been above a specified level. In Figure 4.48 this is the sum of the interval lengths of **P1**, **P2**, **P3**, **P4** and **P5**.

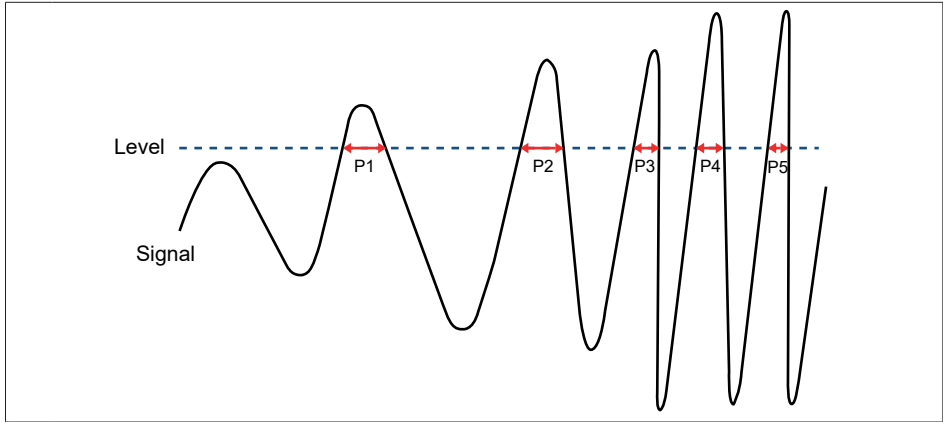


Fig. 4.55 *TimeTotalAbove*

See also

"@TimeTotalAboveBegin" on page 261, "@TimeMinAbove" on page 251, "@TimeTotalAbove" on page 259 and "@TimeMaxBelow" on page 247

4.114 @TimeTotalAboveBegin

Function

This function returns a number representing the start of the first period where the signal is above the specified level.

Syntax

`@TimeTotalAboveBegin(Signal)`
`@TimeTotalAboveBegin(Signal; Level)`
`@TimeTotalAboveBegin(Signal; Level; Start)`
`@TimeTotalAboveBegin(Signal; Level; Start; End)`

Parameters

<i>Signal</i>	Input waveform
<i>Level</i>	Number: The used level default value is 0
<i>Start</i>	Number: Start of segment, default begin of signal
<i>End</i>	Number: End of segment, default end of signal

Output

The output is a numerical value representing the start of the first period where the signal is above the specified level.

Description

This function is used when you want to know where the first period where the signal is above the specified level starts. In Fig. 4.56 this is **T1**, this number will be returned by the @TimeTotalAboveBegin function.

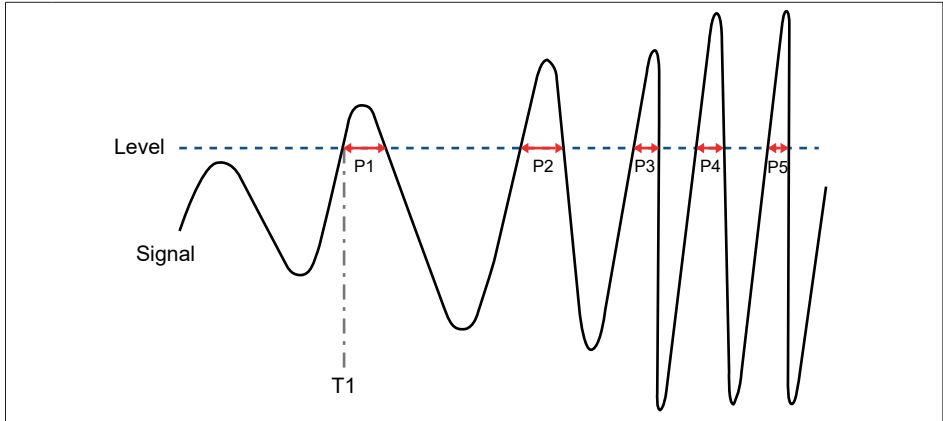


Fig. 4.56 *TimeTotalAboveBegin*

See also

"@TimeTotalAbove" on page 259, "@TimeMinAbove" on page 251 and "@TimeMaxBelow" on page 247

4.115 @TimeTotalBelow

Function

This function returns a number representing the total amount of time that the signal is below a specified level.

Syntax

@TimeTotalBelow(*Signal*)
@TimeTotalBelow(*Signal*; *Level*)
@TimeTotalBelow(*Signal*; *Level*; *Start*)
@TimeTotalBelow(*Signal*; *Level*; *Start*; *End*)

Parameters

<i>Signal</i>	Input waveform
<i>Level</i>	Number: The used level default value is 0
<i>Start</i>	Number: Start of segment, default begin of signal
<i>End</i>	Number: End of segment, default end of signal

Output

The output is a numerical value representing the total amount of time that the signal is below a specified level.

Description

This function is used when you want to know how long a signal has been below a specified level. In Fig. 4.57 this is the sum of the interval lengths of **P1**, **P2**, **P3**, **P4**, **P5** and **P6**.

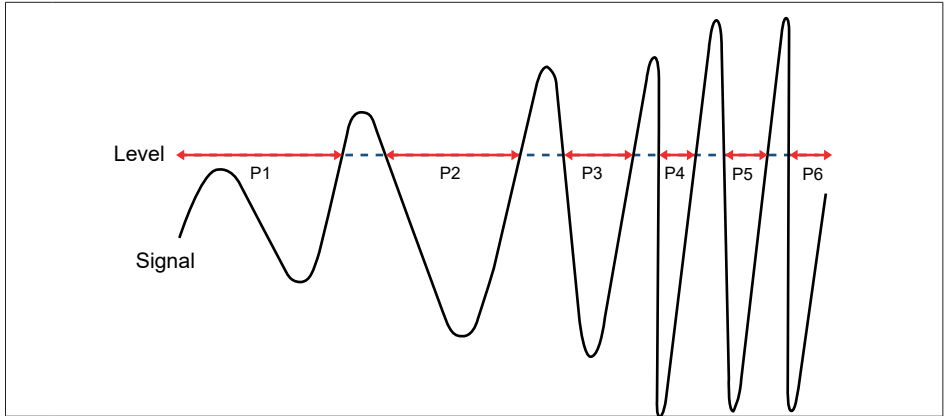


Fig. 4.57 *TimeTotalBelow*

See also

"@TimeTotalBelowBegin" on page 265, "@TimeMinAbove" on page 251, and "@TimeMaxBelow" on page 247

4.116 @TimeTotalBelowBegin

Function

This function returns a number representing the start of the first period where the signal is below the specified level.

Syntax

`@TimeTotalBelowBegin(Signal)`
`@TimeTotalBelowBegin(Signal; Level)`
`@TimeTotalBelowBegin(Signal; Level; Start)`
`@TimeTotalBelowBegin(Signal; Level; Start; End)`

Parameters

<i>Signal</i>	Input waveform
<i>Level</i>	Number: The used level default value is 0
<i>Start</i>	Number: Start of segment, default begin of signal
<i>End</i>	Number: End of segment, default end of signal

Output

The output is a numerical value representing the start of the first period where the signal is below the specified level.

Description

This function is used when you want to know where the first period where the signal is below the specified level starts. In Fig. 4.58 this is **T1**, this number will be returned by the @TimeTotalBelowBegin function.

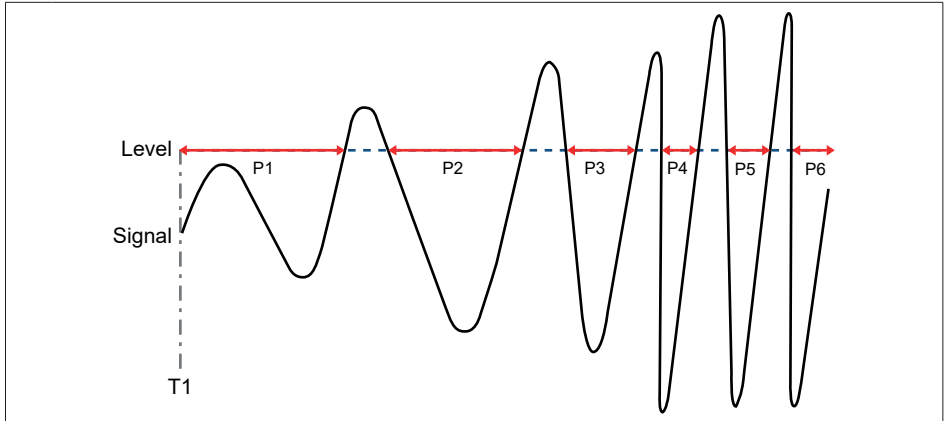


Fig. 4.58 *TimeTotalBelowBegin*

See also

"@TimeTotalBelow" on page 263, "@TimeMinAbove" on page 251 and "@TimeMaxBelow" on page 247

4.117 @TriggerTime

Function

Returns the **trigger** position of a selected sweep.

Syntax

@TriggerTime(*Waveform*)

@TriggerTime(*Waveform*; *NSweep*)

Parameters

Waveform Waveform containing at least one sweep.

NSweep Number: selected sweep, default = 1.

Output

A number containing the trigger position in X-Units.

Description

Some recorder hardware and software settings make it possible to make multi-sweep recordings. Each valid trigger event will create a new sweep. The waveform of a channel from a multi-sweep recording contains trigger time information for each individual sweep.

The @TriggerTime function makes it possible to get the trigger time in X-units for the specified sweep number.

This function can be used in combination with for example the "SweepReview" display. This kind of display shows only one single sweep. The display can move to each individual sweep. The trigger position of the shown sweep is always referenced to zero. Also the cursor values are referenced to the trigger position of the sweep. When a calculation is needed involving a cursor position, the actual location of the cursor in the original waveform needs to be calculated using the corresponding trigger time.

Example

This example shows the calculation of the maximum value of the voltage 20 ms around the trigger position of the second sweep from a multi-sweep recording.

```
Signal      = Active.Group1.Recorder_A.Ch_A2
TPos        = @TriggerTime(Formula.Signal; 2)
MaxVoltage  = @Max(Formula.Signal; Formula.TPos - 10m;
                  Formula.TPos + 10m)
```

See also

"@TriggerTimeToText" on page 269

4.118 @TriggerTimeToText

Function

Returns the **trigger** position of the selected sweep in a **time-date** formatted string.

Syntax

```
@TriggerTimeToText(Waveform)
@TriggerTimeToText(Waveform; NSweep)
@TriggerTimeToText(Waveform; NSweep; Decimals)
@TriggerTimeToText(Waveform; NSweep; Decimals; TimeFormat)
@TriggerTimeToText(Waveform; NSweep; Decimals; TimeFormat; DateFormat)
```

Parameters

<i>Waveform</i>	Waveform with at least one sweep	
<i>NSweep</i>	Number: selected sweep, default = 1	
<i>Decimals</i>	Number of decimals for the fractional part of the seconds, in the range 0 through 9. Default = 3	
<i>TimeFormat</i>	"Relative" (default)	Time since start of recording
	"Local"	Local time
	"UTC"	Coordinated Universal Time (UTC)
<i>DateFormat</i>	"None" (default)	No date information
	"Short"	Short date string representation
	"Long"	Long date string representation

Output

The output is a string that represents the trigger position.

Description

Various recorder hardware and software settings allow you to create multisweep recordings. Each valid trigger event will create a new sweep. The waveform of a channel from a multi-sweep recording contains trigger time information for each individual sweep.

The @TriggerTimeToText function makes it possible to get the trigger time of a specified sweep as a text string.

This function can be used in combination with for example the Perception SweepReview display. This display shows a single sweep. The display can move to each individual sweep. The trigger position of that sweep is always referenced to zero; however with the TriggerTimeToText function it is possible to present the trigger time in other relations and in different formats.

The date and time formats are associated with the regional settings of Windows® and therefore they can vary from machine to machine. Refer to your Windows® Help for more details.

Notice

This function works only for recordings recorded with internal time base, this means that the x-axis dimension is time. The Time and Date format strings are not case-sensitive. The return value is a string that can also be used and interpreted by other programs (like Excel) as a string.

Example

This example shows the trigger time of the second sweep in UTC time.

```
Signal          = Active.Group1.Recorder_A.Ch_A2
TriggerUTC      = @TriggerTimeToText
                  (Formula.Signal; 2; 6; "UTC"; "Short")
```

The output string could be: "3-3-2008 10:40:45.301455"

When only the time is required:

```
TriggerTime     = @TriggerTimeToText
                  (Formula.Signal; 2; 0; "Local"; "none")
```

This could result in: "11:40:45"

See also

"@TriggerTime" on page 267

4.119 @TrueFrequency

Function

Returns a waveform that represents the frequency as function of time.

Syntax

```
@TrueFrequency(Waveform)
@TrueFrequency(Waveform; Cycles)
@TrueFrequency(Waveform; Cycles; Fc)
@TrueFrequency(Waveform; Cycles; Fc; Hysteresis)
@TrueFrequency(Waveform; Cycles; Fc; Hysteresis; Begin)
@TrueFrequency(Waveform; Cycles; Fc; Hysteresis; Begin; End)
```

Parameters

<i>Waveform</i>	Waveform for the Frequency calculation.
<i>Cycles</i>	Number of cycles to be used for the calculating of the frequency, the number of cycles has to be an integer value (1, 2, 3 .. etc.). The default value is 1 cycle.
<i>Fc</i>	Cut-off frequency of 2 nd order phase less lowpass Bessel filter. This filter is used to filter the input signal before looking at the zero crossings. If not entered or 0 then no filtering is done.
<i>Hysteresis</i>	Number: Hysteresis in % of the difference between the maximum and minimum value of the input signal. This hysteresis is used to find the correct zero crossing values. The default value is 20%.
<i>Begin</i>	Number: Start of segment, default begin of signal.
<i>End</i>	Number: End of segment, default end of signal.

Output

Waveform containing the Frequency value as function of time started at **Begin** until **End**.

Description

This function will create a new waveform with the same sample rate as the original signal. The actual value of the created waveform is the value of the frequency at specific times of the supplied waveform. The parameter Cycles determines how many cycles have to be used for calculating the frequency in a given point, by default this is one cycle. The function uses the zero crossing to determine a cycle. The hysteresis value is used to determine the correct zero crossings and eliminate incorrect zero crossings caused by noise. An internal 2nd order phase less lowpass Bessel filter can be used to filter the signal before the zero crossings are detected. If the parameter Fc has not been entered or is equal to 0 then no filter is used.

Example

The following example creates a waveform showing the true Frequency value of the voltage over each cycle. The maximum frequency is also calculated.

```
Frequency = @TrueFrequency  
            (Active.Group1.Recorder_Voltage.Voltage; 5; 100)  
MaxFreq   = @Max(Formula.Frequency)
```


4.120 @TrueRMS

Function

Returns a waveform that represents the true **RMS (Root Mean Square)** per number of cycles.

Syntax

```
@TrueRMS(Waveform)
@TrueRMS(Waveform; Cycles; Begin; End)
```

Parameters

<i>Waveform</i>	Waveform for the RMS calculation.
<i>Cycles</i>	Number of cycles to be used for the RMS calculation.
<i>Begin</i>	Number: segment begin
<i>End</i>	Number: segment end

Output

Waveform containing the RMS value over the requested number of cycles.

Description

This function will create a new waveform with the same sample rate as the original signal. The actual value of the created waveform is the value of the true RMS of the supplied waveform.

The default value is 1 cycle. Other values will be converted to a multiple of 0.5 cycles. The function uses the zero crossing to determine a cycle. Use the Cycles parameter to set the number of cycles - in multiples of 0.5 - for the calculation.

Example

The following example creates a waveform showing the true RMS value of the voltage over each cycle. The average RMS over the complete signal is calculated.

```
RMS_Signal = @TrueRMS
              (Active.Group1.Recorder_Voltage.Voltage;1)
AverageRMS = @Mean(Formula.RMS_Signal)
```

See also

"@RMS" on page 216

4.121 @TrueRMSRef

Function

Returns a waveform that represents the true **RMS (Root Mean Square)** per number of cycles. A reference waveform can also be used to calculate the cycles by determining the zero crossing values of the reference waveform.

Syntax

```
@TrueRMSRef(Waveform)
@TrueRMSRef(Waveform; Reference)
@TrueRMSRef(Waveform; Reference; Cycles)
@TrueRMSRef(Waveform; Reference; Cycles; Begin)
@TrueRMSRef(Waveform; Reference; Cycles; Begin; End)
```

Parameters

<i>Waveform</i>	Waveform for the RMS calculation.
<i>Reference</i>	Reference waveform to be used to determine the zero crossings.
<i>Cycles</i>	Number of cycles to be used for the RMS calculation.
<i>Begin</i>	Number: segment begin
<i>End</i>	Number: segment end

Output

Waveform containing the RMS value over the requested number of cycles.

Description

This function will create a new waveform with the same sample rate as the original signal. The actual value of the created waveform is the value of the true RMS of the supplied waveform.

A reference waveform can be used to calculate the cycles by determining the zero crossing values from this reference waveform. If this reference waveform is not defined, the waveform itself will then be used to determine the cycles. The default number of cycles is 1. Other values will be converted to a multiple of 0.5 cycles. The function uses the zero crossing to calculate a cycle. Use the Cycles parameter to set the number of cycles - in multiples of 0.5 - for the calculation.

Example

The following example creates a waveform showing the true RMS value of the voltage over each cycle. The average RMS over the complete signal is calculated.

```
RMS_Signal = @TrueRMSRef
             (Active.Group1.Recorder_Voltage.Voltage;
             Active.Group1.Recorder_Voltage.RefSignal;1)
AverageRMS = @Mean(Formula.RMS_Signal)
```

See also

"@RMS" on page 216 and "@TrueRMS" on page 273

4.122 @Value

Function

Returns the amplitude **value** of a waveform at a specified x-position.

Syntax

`@Value(Waveform; XPos)`

Parameters

<i>Waveform</i>	Waveform whose value is to be determined at the specified x-position.
<i>XPos</i>	Number: x-position at which the value of the waveform is to be determined.

Output

Amplitude value of the waveform at the specified x-position.

Description

The value of a waveform at a specified x-position is determined. The x-position is to be specified as x-coordinate, not as a sample number. If the x-coordinate is between two sampling points, linear interpolation is used to determine the value at this x-position.

If the x-position is before the first sample or after the last sample of the waveform data set, the function returns no value.

Example

The following example searches a signal named Marker for a rising level crossing with level 2.5 and stores the time found in the variable TimeMarker. Then the value of another signal is determined at this position. Real-world signals are assumed.

```
Marker           = Active.Group1.Recorder_A.Ch_A1
OtherSignal      = Active.Group1.Recorder_A.Ch_A2
TimeMarker       = @NextLvlCross(Formula.Marker; 0; 2.5; 1)
ValueAtMark      = @Value(Formula.OtherSignal;
                          Formula.TimeMarker)
```

4.123 @XDelta

Function

Returns the sampling interval (the time between samples) of a waveform.

Syntax

`@XDelta(Waveform)`

Parameters

Waveform Input waveform

Output

The output is a numerical value.

Description

This function returns the sampling interval (period) of a waveform. The sampling interval is the reciprocal (inverse) value of the sampling frequency and represents the difference in x units between adjacent samples.

Example

The following example creates a sine wave with a sampling frequency of 1 kHz. The variable `dx` receives the sampling interval of this signal (1 ms). The variable `sf` receives the sampling frequency:

```
Signal = 25 * @SineWave(1k; 1000; 50)
dx      = @XDelta(Formula.Signal)
sf      = 1.0 / @XDelta(Formula.Signal)
```

See also

"@Length" on page 151, "@XFirst" on page 280 and "@XLast" on page 281

4.124 @XDeltaHigh

Function

Determines the **highest** sampling interval of a waveform.

Syntax

@XDeltaHigh(*Waveform*)

Parameters

Waveform Waveform for which the highest sampling interval is determined.

Output

The output is a numerical value: the difference in x units between two samples.

Description

When working with multi-time base waveforms, the highest sampling interval of the waveform is determined. The sampling interval is the reciprocal value of the sampling frequency and represents the difference in x units between adjacent samples.

Example

This example creates a waveform containing 2 different sample rates. The XDelta functions are used to determine the sampling frequency.

```
SignalLow   = @SineWave(10k;10k;50)
SignalHigh  = @SineWave(100k;10k;50)
Signal_LHL  = @Join(Formula.SignalLow;
                    Formula.SignalHigh; Formula.SignalLow)
XDelta      = @XDelta(Formula.Signal_LHL)
XHigh       = @XDeltaHigh(Formula.Signal_LHL)
XLow        = @XDeltaLow(Formula.Signal_LHL)
```

See also

"@XDelta" on page 277 and "@XDeltaLow" on page 279

4.125 @XDeltaLow

Function

Determines the **lowest** sampling interval of a waveform.

Syntax

@XDeltaLow(*Waveform*)

Parameters

Waveform Waveform for which the highest sampling interval is determined.

Output

The output is a numerical value: the difference in x units between two samples.

Description

When working with multi-time base waveforms, the lowest sampling interval of the waveform is determined. The sampling interval is the reciprocal value of the sampling frequency and represents the difference in x units between adjacent samples.

Example

This example creates a waveform containing 2 different sample rates. The XDelta functions are used to determine the sampling frequency.

```
SignalLow   = @SineWave(10k;10k;50)
SignalHigh  = @SineWave(100k;10k;50)
Signal_LHL  = @Join(Formula.SignalLow;
                    Formula.SignalHigh; Formula.SignalLow)
XDelta      = @XDelta(Formula.Signal_LHL)
XHigh       = @XDeltaHigh(Formula.Signal_LHL)
XLow        = @XDeltaLow(Formula.Signal_LHL)
```

See also

"@XDelta" on page 277 and "@XDeltaHigh" on page 278

4.126 @XFirst

Function

Returns the **x-coordinate** of the **first** sample in a waveform.

Syntax

`@XFirst(Waveform)`

Parameters

Waveform Input waveform

Output

The output is a numerical value.

Description

The x-coordinate of the first sample is returned. The x-coordinate is a value in terms of the horizontal units of the waveform (typically time).

Example

The following example creates a sine wave. Generated signals always start at time 0. The waveform is shifted horizontally by 100 ms. When the x-coordinate of the first sample of this shifted waveform is determined, it will be 100 ms.

```
Signal    = @SineWave(10k; 1000; 50)
Shifted   = @XShift(Formula.Signal; 100m)
Start     = @XFirst(Formula.Shifted)
```

See also

"@Length" on page 151, "@XDelta" on page 277, "@XLast" on page 263 and "@XShift" on page 282

4.127 @XLast

Function

Returns the **x-coordinate** of the **last** sample in a waveform.

Syntax

@XLast(*Waveform*)

Parameters

Waveform Input waveform

Output

The output is a numerical value.

Description

The x-coordinate of the last sample is returned. The x-coordinate is a value in terms of the horizontal units of the waveform (typically time).

Example

The following example creates a sine wave. Generated signals always start at time 0. The example shows two ways of determining the x-coordinate of the last sample in the waveform:

```
Signal = @SineWave(10k; 1000; 50)
XEnd1  = @XFirst(Formula.Signal) +
        (@Length(Formula.Signal) - 1)*
        @XDelta(Formula.Signal)
XEnd2  = XLast(Formula.Signal)
```

See also

"@Length" on page 151, "@XDelta" on page 277, "@XFirst" on page 280 and "@XShift" on page 282

4.128 @XShift

Function

Shifts a waveform horizontally by a specified amount of time.

Syntax

`@XShift(Waveform, Shift)`

Parameters

<i>Waveform</i>	Waveform to be shifted horizontally.
<i>Shift</i>	Number: amount of shift

Output

Original waveform shifted in time.

Description

This function does not alter the contents of a waveform but changes the horizontal scaling information in such a way that the waveform is shifted horizontally by the specified amount of time (or other units). Positive values shift the waveform to the right, negative values shift the waveform to the left.

Example

The following example makes sure the waveform's horizontal axis always starts at zero. We assume a real-world signal.

```
Signal      = Active.Group1.Recorder_A.Ch_A1
Shift       = -1 * @XFirst(Formula.Signal)
CorrSignal  = @XShift(Formula.Signal; Formula.Shift)
```

See also

"@XDelta" on page 278, "@XFirst" on page 280 and "@XLast" on page 281

4.129 @XYArray

Function

Creates a waveform using a two-dimensional array.

Syntax

@XYArray(X1; Y1; ...; ...; XN; YN)

Parameters

X1	X value of first sample.
Y1	Y value of first sample.
XN	X value of last sample; N>= 2.
YN	Y value of last sample; N>= 2.

Output

Waveform containing all the points of the array.

Description

This function creates a waveform containing all the points passed by the parameters X1, Y1, X2, Y2 etc. The entered points should be ordered in time.

The output waveform contains equidistant data points. This function will evaluate the input X values and define the best possible new sample rate. For the sample rate the 1, 2, 5 series is used.

Assume X1 = 0 ms, X2 = 30 ms, X3 = 50 ms. This yields a sampling interval of 20 ms.

Assume X1 = 25 ms, X2 = 30 ms, X3 = 50 ms. This yields a sampling interval of 5 ms.

Assume X1 = 2 s, X2 = 12 s, X3 = 15 s. This yields a sampling interval of 2 s.

Notice

The output samples are equidistant; the number of samples can be larger than the number of input samples.

Example

The following example creates a waveform containing 8 data points, while only 5 points were entered. The three new points are generated by the XYArray function.

```
NewWave = @XYArray(0; 0; 1; 2; 2; 1; 5; -3; 7; 0)
```

See also

"@YArray" on page 285

4.130 @YArray

Function

This function creates a waveform using a single dimensional **array** as data input.

Syntax

`@YArray(Fsampling; Y1; ...; YN)`

Parameters

<i>FSampling</i>	Number: Sampling frequency
<i>Y1</i>	Number: amplitude (Y-value) of first data point
<i>YN</i>	Number: amplitude (Y-value) of last data point. N >=2

Output

The output is a waveform containing all data points of the array.

Description

This function creates a waveform containing all the points passed by the parameters Y1 through YN. The first parameter defines the sampling frequency (= 1 / sampling period)

Example

The following example creates a waveform containing 9 data points with a sampling frequency of 100 Hz.

```
Signal = @YArray(100; 0; 1; 2; 2; 1; -1; -3; 2; 0)
```

See also

"@XYArray" on page 283

5 Reference Guide - HIC

5.1 Introduction - HIC

HIC functions are used for analyzing crash test data. HIC is an abbreviation of **H**ead **I**njury **C**riterion.

5.2 @Con3ms

Function

This function returns the highest acceleration level with a **continuous** duration of at least **3 milliseconds**.

Syntax

@Con3ms(Signal)
@Con3ms(Signal; Start)
@Con3ms(Signal; Start; End)

Parameters

<i>Signal</i>	Input waveform
<i>Start</i>	Number: Start of segment, default begin of signal
<i>End</i>	Number: End of segment, default end of signal

Output

The output is a numerical value containing highest acceleration level with a cumulative duration of at least 3 milliseconds.

Description

Con3ms injury criterion calculates the highest acceleration level with a continuous duration of at least 3 milliseconds. For an implementation we refer to the document: **Crash Analysis Criteria Description Version 2.1.1 from the Data Processing Vehicle Safety Workgroup, chapter 6 page 6-2**. In this document they call it **Xms**.

The Con3ms value can be calculated with one peak, as shown in Fig. 5.1, or with several peaks, as shown in Fig. 5.2

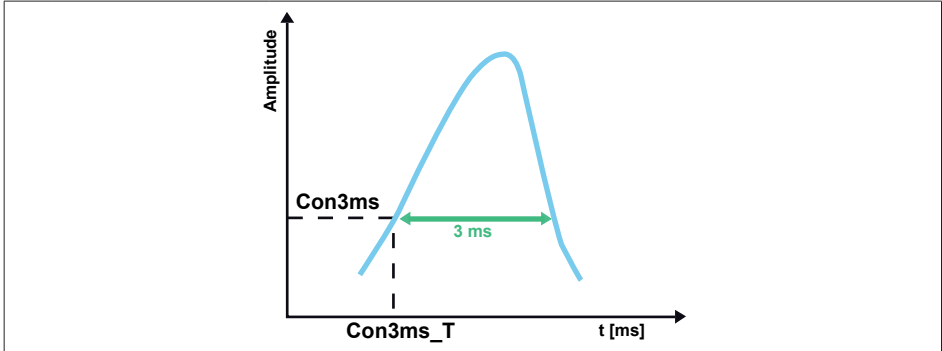


Fig. 5.1 Con3ms value (One peak)

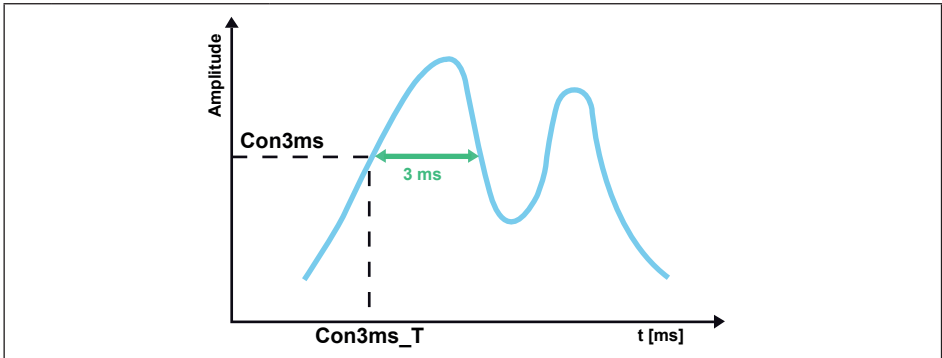


Fig. 5.2 Con3ms value (Several peaks)

See also

"@Cum3ms" on page 290, "@Cum3ms_T" on page 292 and "@Con3ms_T" on page 289

5.3 @Con3ms_T

Function

This function returns the start time of the highest acceleration level with a **continuous** duration of at least **3 milliseconds**.

Syntax

@Con3ms_T(*Signal*)
@Con3ms_T(*Signal*; *Start*)
@Con3ms_T(*Signal*; *Start*; *End*)

Parameters

<i>Signal</i>	Input waveform
<i>Start</i>	Number: Start of segment, default begin of signal
<i>End</i>	Number: End of segment, default end of signal

Output

The output is a numerical value containing start time of the highest acceleration level with a continuous duration of at least 3 milliseconds.

Description

Con3ms injury criterion calculates the highest acceleration level with a continuous duration of at least 3 milliseconds. For an implementation we refer to the document: **Crash Analysis Criteria Description Version 2.1.1 from the Data Processing Vehicle Safety Workgroup, chapter 6 page 6-2**. In this document they call it **Xms**.

See also

"@Con3ms" on page 287, "@Cum3ms" on page 290 and "@Con3ms_T" on page 289

5.4 @Cum3ms

Function

This function returns the highest acceleration level with a **cumulative** duration of at least **3 milliseconds**.

Syntax

@Cum3ms(*Signal*)
@Cum3ms(*Signal*; *Start*)
@Cum3ms(*Signal*; *Start*; *End*)

Parameters

<i>Signal</i>	Input waveform
<i>Start</i>	Number: Start of segment, default begin of signal
<i>End</i>	Number: End of segment, default end of signal

Output

The output is a numerical value containing highest acceleration level with a cumulative duration of at least 3 milliseconds.

Description

Cum3ms injury criterion calculates the highest acceleration level with a cumulative duration of at least 3 milliseconds. For an implementation we refer to the document: **Crash Analysis Criteria Description Version 2.1.1 from the Data Processing Vehicle Safety Workgroup, chapter 6 page 6-2**. In this document they call it **Xms**.

In the cumulative calculation, separate periods of the measurement signal are added, until 3 milliseconds are reached.

The Cum3ms value can be calculated with one peak, as shown in Fig. 5.3, or with several peaks, as shown in Fig. 5.4.

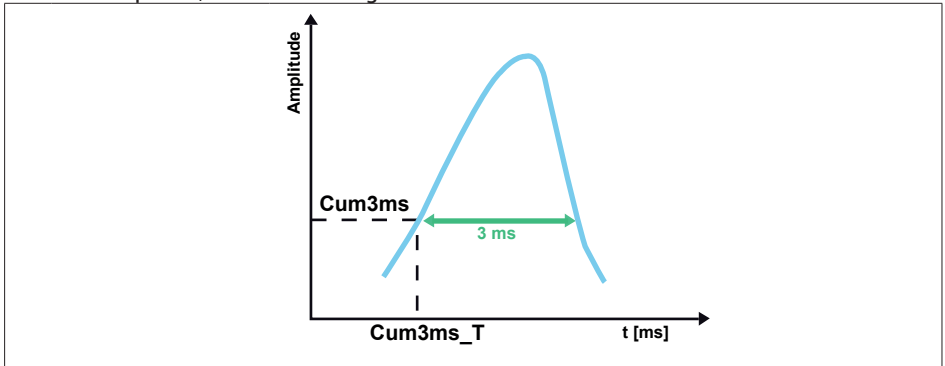


Fig. 5.3 Cum3ms value (One peak)

The special case shown in the second figure.

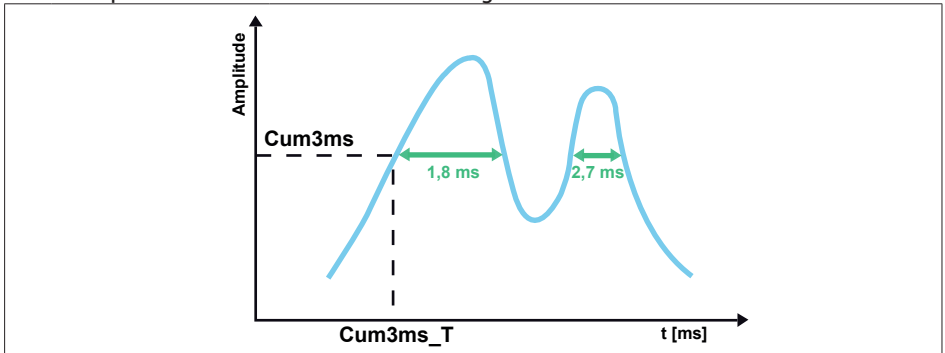


Fig. 5.4 Cum3ms value (Several peaks)

See also

"@Con3ms" on page 287, "@Cum3ms_T" on page 292 and "@Con3ms_T" on page 289

5.5 @Cum3ms_T

Function

This function returns the start time of the highest acceleration level with a **cumulative** duration of at least **3 milliseconds**.

Syntax

@Cum3ms_T(*Signal*)

@Cum3ms_T(*Signal*; *Start*)

@Cum3ms_T(*Signal*; *Start*; *End*)

Parameters

Signal Input waveform

Start Number: Start of segment, default begin of signal

End Number: End of segment, default end of signal

Output

The output is a numerical value containing start time of the highest acceleration level with a cumulative duration of at least 3 milliseconds.

Description

Cum3ms injury criterion calculates the highest acceleration level with a cumulative duration of at least 3 milliseconds. For an implementation we refer to the document: **Crash Analysis Criteria Description Version 2.1.1 from the Data Processing Vehicle Safety Workgroup, chapter 6 page 6-2**. In this document they call it **Xms**.

In the cumulative calculation, separate periods of the measurement signal are added, until 3 milliseconds are reached.

See also

"@Con3ms" on page 287, "@Cum3ms" on page 290 and "@Con3ms_T" on page 289

5.6 @HIC

Function

The HIC functions calculate the **Head Injury Criterion** of a waveform.

Syntax

```
@HIC(Signal)
@HIC(Signal; Start)
@HIC(Signal; Start; End)
```

Parameters

<i>Signal</i>	Input waveform
<i>Start</i>	Number: Start of segment, default begin of signal
<i>End</i>	Number: End of segment, default end of signal

Output

The output is a numerical value containing the calculated HIC value.

Description

HIC is the abbreviation for Head Injury Criterion. For an implementation we refer to the document: **Crash Analysis Criteria Description Version 2.1.1 from the Data Processing Vehicle Safety Workgroup**, chapter 2 page 2-2.

The Head Injury Criterion is a measure of the likelihood of head injury arising from an impact. The HIC can be used to assess safety related to vehicles, personal protective gear, and sport equipment.

Normally the variable is derived from the acceleration/time history of an accelerometer mounted at the centre of gravity of a dummy's head, when the dummy is exposed to crash forces.

The HIC functions calculate the Head Injury Criterion of a waveform. The parameters **Start** and **End** are optional and define the part of the input signal to be used for the HIC calculations. The HIC value is calculated with the following formula:

$$HIC = \sup_{t_1, t_2} \left\{ \left(\frac{1}{t_2 - t_1} \int_{t_1}^{t_2} a \, dt \right)^{2.5} (t_2 - t_1) \right\}$$

$$a = \sqrt{a_x^2 + a_y^2 + a_z^2}$$

The HIC function determines the interval defined by t_1 and t_2 where the HIC attains a maximum value. The variable a is the resultant acceleration of the center of gravity of the head in units of acceleration of gravity ($1g = 9.81 \text{ m/s}^2$).

See also

"@HICStartTime" on page 295 and "@HICEndTime" on page 296

5.7 @HICStartTime

Function

This function returns the **start** of the interval (t_1) as found during the **HIC** calculations.

Syntax

```
@HICStartTime(Signal)  
@HICStartTime(Signal; Start)  
@HICStartTime(Signal; Start; End)
```

Parameters

<i>Signal</i>	Input waveform
<i>Start</i>	Number: Start of segment, default begin of signal
<i>End</i>	Number: End of segment, default end of signal

Output

The output is a numerical value containing the start of the interval (t_1) as found during the HIC calculations.

Description

The HICStartTime function returns the start in time of the interval (t_1) where the HIC has its maximum value. See also the HIC description on page 293.

See also

"@HIC" on page 293 and "@HICEndTime" on page 296

5.8 @HICEndTime

Function

This function returns the **end** of the interval (t_2) as found during the **HIC** calculations.

Syntax

@HICEndTime(*Signal*)
@HICEndTime(*Signal*; *Start*)
@HICEndTime(*Signal*; *Start*; *End*)

Parameters

<i>Signal</i>	Input waveform
<i>Start</i>	Number. Start of segment, default begin of signal
<i>End</i>	Number. End of segment, default end of signal

Output

The output is a numerical value containing the end of the interval (t_2) as found during the HIC calculations.

Description

The HICEndTime function returns the end in time of the interval (t_2) where the HIC has its maximum value. See also the HIC description on page 293.

See also

"@HIC" on page 293 and "@HICStartTime" on page 295

5.9 @HIC15

Function

This function returns the maximum **HIC** value when working with a fixed **15 milli-second** interval.

Syntax

@HIC15(*Signal*)
@HIC15(*Signal*; *Start*)
@HIC15(*Signal*; *Start*; *End*)

Parameters

<i>Signal</i>	Input waveform
<i>Start</i>	Number: Start of segment, default begin of signal
<i>End</i>	Number: End of segment, default end of signal

Output

The output is a numerical value containing the HIC value when working with a fixed 15 millisecond interval.

Description

See also the HIC function on page 293.

See also

"@HIC36" on page 300, "@HIC15StartTime" on page 298 and "@HIC15EndTime" on page 299

5.10 @HIC15StartTime

Function

This function returns the **start** of the interval (t_1) as found during the **HIC15** calculations.

Syntax

@HIC15StartTime(*Signal*)
@HIC15StartTime(*Signal*; *Start*)
@HIC15StartTime(*Signal*; *Start*; *End*)

Parameters

<i>Signal</i>	Input waveform
<i>Start</i>	Number: Start of segment, default begin of signal
<i>End</i>	Number: End of segment, default end of signal

Output

The output is a numerical value containing the start of the interval (t_1) as found during the HIC15 calculations.

Description

The **HIC15StartTime** function returns the start in time of the interval (t_1) where the HIC has its maximum value and the interval has a maximum of 15 ms. See also the HIC description on page 293.

See also

"@HIC15" on page 297 and "@HIC15EndTime" on page 299

5.11 @HIC15EndTime

Function

This function returns the **end** of the interval (t_2) as found during the **HIC15** calculations.

Syntax

```
@HIC15EndTime(Signal)  
@HIC15EndTime(Signal; Start)  
@HIC15EndTime(Signal; Start; End)
```

Parameters

<i>Signal</i>	Input waveform
<i>Start</i>	Number: Start of segment, default begin of signal
<i>End</i>	Number: End of segment, default end of signal

Output

The output is a numerical value containing the end of the interval (t_2) as found during the HIC15 calculations.

Description

The HIC15EndTime function returns the end in time of the interval (t_2) where the HIC has its maximum value and the interval has a maximum of 15 ms. See also the HIC description on page 293.

See also

"@HIC15" on page 297 and "@HIC15StartTime" on page 298

5.12 @HIC36

Function

This function returns the maximum **HIC** value when working with a fixed **36 milli-second** interval.

Syntax

@HIC36(*Signal*)

@HIC36(*Signal*; *Start*)

@HIC36(*Signal*; *Start*; *End*)

Parameters

Signal Input waveform

Start Number: Start of segment, default begin of signal

End Number: End of segment, default end of signal

Output

The output is a numerical value containing the HIC value when working with a fixed 36 millisecond interval.

Description

See also the HIC function on page 293.

See also

"@HIC36StartTime" on page 301 and "@HIC36EndTime" on page 302

5.13 @HIC36StartTime

Function

This function returns the **start** of the interval (t_i) as found during the **HIC36** calculations.

Syntax

@HIC36StartTime(*Signal*)
@HIC36StartTime(*Signal*; *Start*)
@HIC36StartTime(*Signal*; *Start*; *End*)

Parameters

<i>Signal</i>	Input waveform
<i>Start</i>	Number: Start of segment, default begin of signal
<i>End</i>	Number: End of segment, default end of signal

Output

The output is a numerical value containing the start of the interval (t_i) as found during the HIC36 calculations.

Description

The HIC36StartTime function returns the start in time of the interval (t_i) where the HIC has its maximum value and the interval has a maximum of 36 ms. See also the HIC description on page 293.

See also

"@HIC36" on page 300 and "@HIC36EndTime" on page 302

5.14 @HIC36EndTime

Function

This function returns the **end** of the interval (t_2) as found during the **HIC36** calculations.

Syntax

@HIC36EndTime(*Signal*)
@HIC36EndTime(*Signal*; *Start*)
@HIC36EndTime(*Signal*; *Start*; *End*)

Parameters

<i>Signal</i>	Input waveform
<i>Start</i>	Number: Start of segment, default begin of signal
<i>End</i>	Number: End of segment, default end of signal

Output

The output is a numerical value containing the end of the interval (t_2) as found during the HIC36 calculations.

Description

The **HIC36EndTime** function returns the end in time of the interval (t_2) where the HIC has its maximum value and the interval has a maximum of 36 ms. See also the HIC description on page 293.

See also

"@HIC36" on page 300 and "@HIC36StartTime" on page 301

6 Reference Guide - Cycle Math

6.1 Introduction - Cycle Math

A characteristic of a sinusoid waveform is a cycle (or period). A full cycle is defined as the time segment where the waveform crosses its center amplitude in the same direction (rising or falling) twice.

The cycle math formulas calculate a characteristic value of a waveform on a cycle-per-cycle basis. In order to determine which input samples to use for the calculation of this characteristic value, the formulas employ a cycle detection algorithm, which determines and collects the times of center-amplitude crossings (or simply called zero crossings, if the center amplitude is zero).

The cycle detection algorithm principle works as follows (see Fig. 6.1):

- Assume that the input signal is $y(n)$, which is a sinusoid oscillating around a center amplitude.
- Let the Hysteresis Band be the amplitude range from the -threshold level to the +threshold level (user-defined).
- Find and store the sample at index k_1 where the signal is:
 - Entering the Hysteresis Band, or
 - Leaving the Hysteresis Band
- Find the next sample at index k_2 , where the signal is: (symmetrically to the previous step)
 - Leaving the Hysteresis Band, or
 - Entering the Hysteresis Band
- Calculate the center amplitude crossing time by linear interpolation between the samples at indices k_1 and k_2 . This crossing time detection can occur in two ways: either a rising edge crossing (point A) or a falling edge crossing (point B). See Fig. 6.1.
- Finally, store all center-amplitude crossings; these can be then used by specific formulas listed in this section to perform cycle-based calculations.

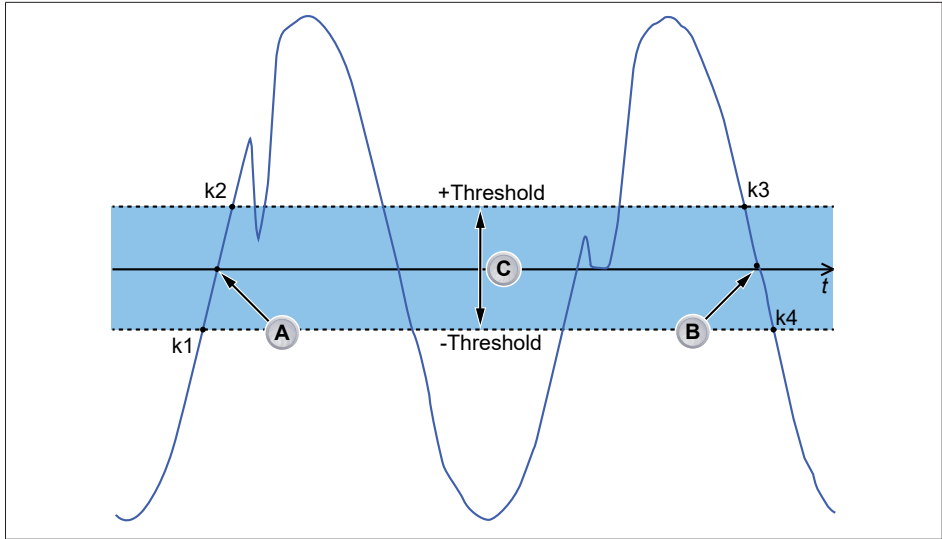


Fig. 6.1 Level crossings

- A Level crossing at rising edge
- B Level crossing at falling edge
- C Hysteresis band

A cycle formula works as follows:

- First, the cycle detection algorithm will determine the center amplitude crossing times.
- Each pair of crossing times, such as times **A** and **B** of Figure 6.1, defines a cycle time segment.
- The formula will gather the samples of the input waveform that belong to this time segment and apply a math function on them to calculate the cycle value.
- The formula's output waveform will assign this value to all samples that belong to the respective time segment.

The cycle math formulas have the versatility of applying a math function on input samples for a duration of a full cycle, for a half-cycle, or for a multiple number of cycles. For instance, a user might want to see the RMS value of the samples that correspond to two consecutive cycles of the input sine wave.

Linear Interpolation

In order to calculate the level crossing time precisely, the cycle detection algorithm employs linear interpolation in the following manner:

For a signal $y(n)$, assume that a - threshold crossing is detected at sample index k_1 (time t_1), and a +threshold crossing is detected at sample index k_2 (time t_2). The time of the center amplitude crossing t_c is then determined by the linear interpolation equation:

$$t_C = t_1 + (CA - y(k_1)) \cdot \frac{t_2 - t_1}{y(k_2) - y(k_1)}$$

CA = Center Amplitude level

If the input signal contains noise, this calculation may result in a t_c value that differs significantly from the actual time t_p where the signal crosses the center amplitude, as shown in Fig. 6.2. Therefore, use the t_c time in this case, not the t_p time.

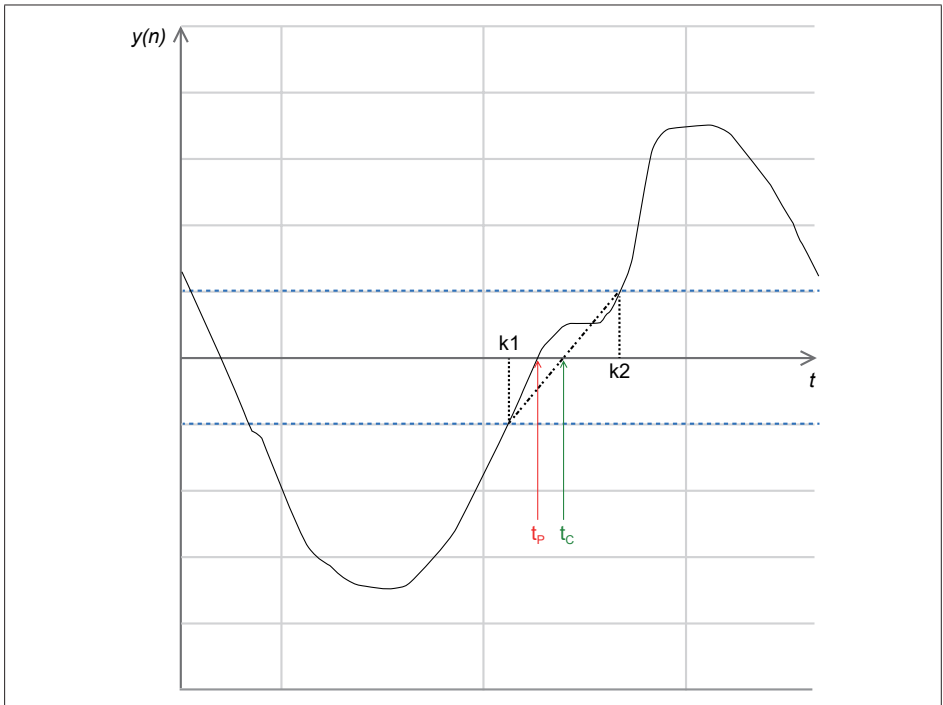


Fig. 6.2 Linear Interpolation

Performance

The cycle math formulas use a reference waveform, which is used to detect the level crossings. Depending on the defined reference waveform, cycle detection is performed either implicitly or explicitly. The reference waveform can be one of the following:

- A A `@CycleDetect` function on a waveform, which is defined in "`@CycleDetect`" on page 312. The current formula will use the level crossing times found in the `@CycleDetect` function directly to determine the output cycles and their values.
- B A recorded (sinusoid) signal. In this case, a `@CycleDetect` function will be created implicitly, with this signal as its input, in order to determine the level crossing times. The current formula will then use this `@CycleDetect` function, as in case (A). The disadvantage of this option is that the user cannot explicitly input a desired center level and threshold to the `@CycleDetect` function.
- C A `@CycleInterval` function, which is defined on page 306. The current formula will use the level crossing times calculated in the `@CycleInterval` function directly to determine the output cycles and their values.
- D No value. When the parameter is omitted, a `@CycleDetect` function applied on the input waveform will be created implicitly. The current formula will use this `@CycleDetect` function as in case (A).

Formula defaults

The cycle formulas may have optional parameters. The default behavior of the formulas is defined as the behavior when the optional parameters are not entered by the user.

Continuous vs. Sweep recordings

- If a signal contains a single sampling frequency (which is the simple behavior for continuous recordings), the cycle formulas process the whole length of the waveform. The user might notice a delay in the representation of a cycle formula for relatively long recordings.
- If a recorded signal is sampled at more than one frequency, the cycle formulas will operate only on each segment of the signal which has a separate sampling frequency.
- If a recorded signal contains sweeps (which are not continuous in time), the cycle formulas will operate on each sweep separately.

The output of a cycle formula has the same sampling frequency as its input, and contains a number of samples which is equal to the number of samples for the input waveform.

6.2 @CycleArea

Function

Calculates the **area** of every **cycle** detected on a reference signal.

Syntax

@CycleArea(Waveform; Cycles; ReferenceWaveform)

Parameters

<i>Waveform</i>	Input waveform to be used for calculation of math function.
<i>Cycles</i>	Number of cycles of the reference waveform used to calculate one output value.
<i>ReferenceWaveform</i>	Waveform to be used as a reference to detect cycles.

Output

A waveform which has a value of zero before the first level crossing, and after the end of the last cycle.

Description

For a range of input samples $y(k_1)$ to $y(k_2)$, where k_1 and k_2 are the indices defined by the detected cycle, the output samples $y'(k_1)$ to $y'(k_2)$ will be assigned the value:

$$\text{Area} = \left(\sum_{k=k_1}^{k_2} |y(k)| \right) \cdot \Delta x$$

Δx = x-difference between two consecutive samples

The start and end indices (k_1 and k_2) for these segments ("cycles") are calculated from the level crossing times determined by the CycleDetect algorithm, which is applied on the ReferenceWaveform.

The Cycles and ReferenceWaveform parameters are optional.

If the Cycles parameter is omitted, the default value is 1.

If the ReferenceWaveform parameter is omitted, a CycleDetect formula is created for the input Waveform.

See also

"@Area" on page 59

6.3 @CycleCount

Function

Counts the number of cycles detected on a reference signal.

Syntax

@CycleCount(*Waveform*; *Cycles*; *Level*; *Threshold*; *SuppressTime*)

Parameters

<i>Waveform</i>	Input waveform to be used for calculation of math function.
<i>Cycles</i>	Number of cycles of the reference waveform used to calculate one output value.
<i>Level</i>	Number: Y-value used as the center of oscillation of the periodic waveform.
<i>Threshold</i>	Number: Y-value used to indicate a level crossing.
<i>SuppressTime</i>	Minimum required duration (in seconds) of a half cycle, all half cycles with duration below this value are suppressed.

Output

At any given offset of time t in the recorded data for the input Waveform, the formula's output will be equal to the number of cycles detected on the Waveform signal from the start of the recorded signal until time t.

Description

If the Cycles parameter is N, the output is incremented by one for each N cycles of the Waveform.

Notice

For very long recordings, the output value of this formula may increase to a high number; to see the output on Perception's TimeDisplay correctly, it might be necessary to apply a different scale on the Y-axis.

The Cycles, Level, and Threshold parameters are optional.

If the Cycles parameter is omitted, the default value is 1.

See also

"@Cycles" on page 77

6.4 @CycleCrestFactor

Function

Calculates the **Crest Factor** of every cycle detected on a reference signal. The “Crest Factor” is the ratio between the absolute maximum and the RMS of that cycle.

Syntax

@CycleCrestFactor(*Waveform*; *Cycles*; *ReferenceWaveform*)

Parameters

- Waveform* Input waveform to be used for calculation of math function.
- Cycles* Number of cycles of the reference waveform used to calculate one output value.
- ReferenceWaveform* Waveform to be used as a reference to detect cycles.

Output

A waveform which has a value of zero before the first level crossing, and after the end of the last cycle.

Description

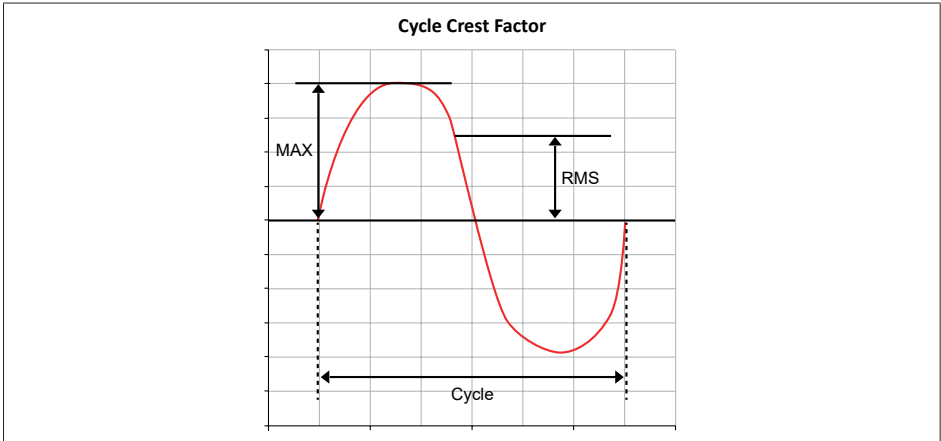


Fig. 6.3 Cycle crest factor

$$\text{CycleCrestFactor} = \frac{\text{Abs}(\text{CycleMax})}{\text{CycleRMS}}$$

$$N = k_2 - k_1 + 1$$

The start and end indices (k_1 and k_2) for these segments ("cycles") are calculated from the level crossing times determined by the CycleDetect algorithm, which is applied on the ReferenceWaveform.

The Cycles and ReferenceWaveform parameters are optional.

If the Cycles parameter is omitted, the default value is 1.

If the ReferenceWaveform parameter is omitted, a CycleDetect formula is created for the input Waveform.

See also

"@CycleDetect" on page 312

6.5 @CycleDetect

Function

Performs level crossing **detection** on the input waveform.

Syntax

@CycleDetect(*Waveform*; *Level*; *Threshold*; *HoldOffTime*)

Parameters

<i>Waveform</i>	Waveform to be used as a reference to detect cycles.
<i>Level</i>	Number: Y-value used as the center of oscillation of the periodic waveform.
<i>Threshold</i>	Number: Y-value used to indicate a level crossing.
<i>HoldOffTime</i>	Minimum required duration (in seconds) of a half cycle, all half cycles with duration below this value are suppressed.

Output

A square waveform which has a positive value between two consecutive level crossing times when the respective input samples value is above the threshold; it has a negative value between two consecutive level crossing times when the respective input samples are below the threshold.

Description

Put together, the Threshold and Level parameters indicate a hysteresis band, which is an amplitude range positioned symmetrically around the Level value. The hysteresis band's upper and lower levels are used to detect cycles. Each input Waveform sample is compared against the lower and upper levels of the hysteresis band. The respective output samples are calculated based on these comparisons.

If the Level and/or Threshold parameters are omitted, they are calculated based on the input Waveform. The Threshold value is calculated as 5% of the input waveform's Display Range; the Level value is calculated as the middle of the Display Range.

The output is set to 0 for all samples before the first level crossing. When it is known that the input waveform contains a DC offset, using this DC offset value as the detection level is recommended.

For better performance (faster calculation), it is also recommended to explicitly define the Level and Threshold parameters – so that the formula function does not need to calculate them.

Example

```
cycledetect = @CycleDetect
              (Active.Group1.Recorder_Vrms.Vrms;0;150)
```

This example creates the square waveform that shows the times of zero crossings in the Vrms signal, using a threshold of 150 V. The output is the red trace in Fig. 6.4. The hysteresis band is indicated by the dashed lines, which are located at +150 V and -150 V.

Fig. 6.4 shows a segment of a signal which is a sine wave with noise (blue trace). The red trace is the generated waveform of the @CycleDetect formula.

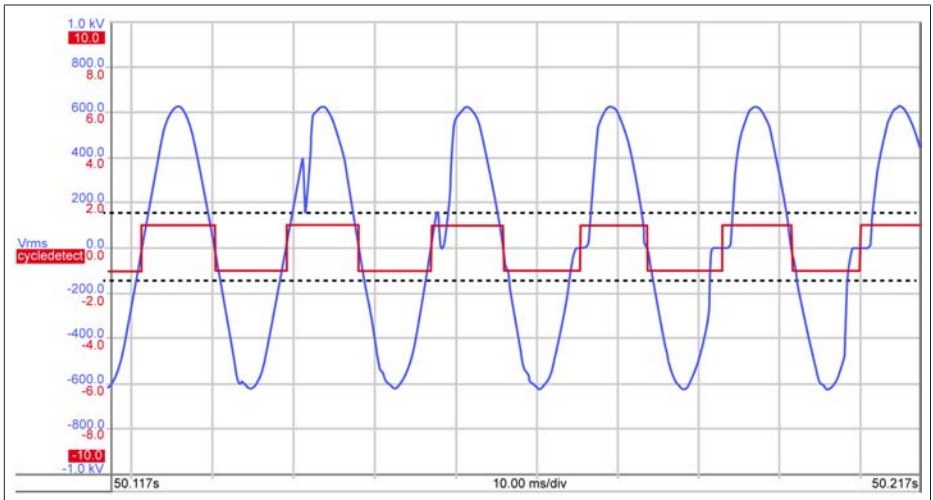


Fig. 6.4 Example of CycleInterval output

See also

"@CycleInterval" on page 324

6.6 @CycleEnergy

Function

Calculates the **energy** of every **cycle** detected on a reference signal.

Syntax

@CycleEnergy(*Waveform*; *Cycles*; *ReferenceWaveform*)

Parameters

<i>Waveform</i>	Input waveform to be used for calculation of math function.
<i>Cycles</i>	Number of cycles of the reference waveform used to calculate one output value.
<i>ReferenceWaveform</i>	Waveform to be used as a reference to detect cycles.

Output

A waveform which has a value of zero before the first level crossing, and after the end of the last cycle.

Description

For a range of input samples $y(k_1)$ to $y(k_2)$, where k_1 and k_2 are the indices defined by the detected cycle, the output samples $y'(k_1)$ to $y'(k_2)$ will be assigned the value:

$$\text{Energy} = \left(\sum_{k=k_1}^{k_2} (y(k))^2 \right) \cdot \Delta x$$

Δx = x-difference between two consecutive samples

The start and end indices (k_1 and k_2) for these segments ("cycles") are calculated from the level crossing times determined by the CycleDetect algorithm, which is applied on the ReferenceWaveform.

The Cycles and ReferenceWaveform parameters are optional.

If the Cycles parameter is omitted, the default value is 1.

If the ReferenceWaveform parameter is omitted, a CycleDetect formula is created for the input Waveform.

See also

"@Energy" on page 85

6.7 @CycleFrequency

Function

Calculates the **frequency** of all the **cycles** detected on the reference signal.

Syntax

@CycleFrequency(*Waveform*; *Cycles*; *ReferenceWaveform*)

Parameters

- Waveform* Input waveform to be used for calculation of math function.
- Cycles* Number of cycles of the reference waveform used to calculate one output value.
- ReferenceWaveform* Waveform to be used as a reference to detect cycles.

Output

The frequency in Hz that is calculated on the ReferenceWaveform. The frequency is the inverse of the time difference between two level crossings that determine the duration of a sinus cycle.

Description

If the ReferenceWaveform and the input Waveform are the same signal, and the Cycles parameter is 1, then the output is the actual frequency of the Waveform.

The frequency is calculated with the formula:

$$F = 1/\Delta T$$

where ΔT :

- Is the duration of half of a period of the Reference Waveform, when Cycles = 0.5.
- Is the duration of 1 period of the Reference Waveform, when Cycles = 1.
- Is the duration of 2 periods of the Reference Waveform, when Cycles = 2.
- etc.

The Cycles and ReferenceWaveform parameters are optional.

If the Cycles parameter is omitted, the default value is 1.

If the ReferenceWaveform parameter is omitted, a CycleDetect formula is created for the input Waveform.

Example

```
freq = @CycleFrequency(Group1.Recorder_A.Ch_A1; 1)
```

The formula above omits the ReferenceWaveform parameter, so it will detect zero crossings in the Ch_A1 waveform. Then, for each full cycle detected on Ch_A1, it will calculate the frequency and assign this value to all output samples that belong to the cycle.

See also

"@Frequency" on page 121 and "@TrueFrequency" on page 271

6.8 @CycleFundamental

Function

Generates the Waveform of the **Cycle Fundamental** of every cycle detected on a reference signal.

Syntax

@CycleFundamental(*Waveform*; *Cycles*; *ReferenceWaveform*)

Parameters

- Waveform*

Input waveform to be used for the calculation of the Fundamental waveform.
- Cycles*

Number of cycles of the reference waveform used to calculate one output value.
- ReferenceWaveform*

Waveform to be used as a reference to detect cycles.

Output

Waveform of the Fundamental Frequency.

Description

By performing harmonic analysis on an input signal it is possible to determine the basic waveforms, that is sines and cosines that compose the original input signal. As such any waveform $f(t)$ is represented by the Fourier series as:

$$f(t) = A_0 + \sum_{n=1}^{\infty} (A_n \cdot \cos(\frac{2 \cdot \pi \cdot n}{T} t) + B_n \cdot \sin(\frac{2 \cdot \pi \cdot n}{T} t))$$

The Fundamental frequency is the component of the above expression when n takes the value of 1; that is the term

$$A_1 \cdot \cos(\frac{2 \cdot \pi}{T} t) + B_1 \cdot \sin(\frac{2 \cdot \pi}{T} t)$$

Where

$$A_1 = \frac{2}{T} \int_t^{t+T} F(t) \cdot \cos\left(\frac{2\pi}{T} t\right) dt$$

$$B_1 = \frac{2}{T} \int_t^{t+T} F(t) \cdot \sin\left(\frac{2\pi}{T} t\right) dt$$

For each detected cycle, in the period t_1, t_2 , @CycleFundamental will determine the values A_1 and B_1 so that it can generate the waveform given by the expression:

$$f(t) = A_1 \cdot \cos\left(\frac{2\pi}{T} t\right) + B_1 \cdot \sin\left(\frac{2\pi}{T} t\right)$$

The Cycles and ReferenceWaveform parameters are optional.

If the Cycles parameter is omitted, the default value is 1.

If the ReferenceWaveform parameter is omitted, a CycleDetect formula is created for the input Waveform.

Example

Given that the input signal is available as Formula.input, the function @CycleDetect determines the zero level crossings which are used by the @CycleFundamental function to determine the Fundamental waveform for the detected cycle:

Name Formula

cd = @CycleDetect(Formula.input; 0.0, 0.1)

fund = @CycleFundamental(Formula.input; 1; Formula.cd)

See also

"@CycleDetect" on page 312

6.9 @CycleFundamentalPhase

Function

Returns a waveform representing the **phase** difference between the **cycle fundamentals** of two signals for each detected cycle in the cycle detect signal. The phase difference is in radians from 0 to 2π .

Syntax

@CycleFundamentalPhase(*Waveform1*; *Waveform2*; *CycleDetect*)

Parameters

<i>Waveform1</i>	Input waveform 1 to be used for the calculation of the cycle fundamental waveform 1.
<i>Waveform2</i>	Input waveform 2 to be used for the calculation of the cycle fundamental waveform 2.
<i>Cycle detect</i>	The waveform to be used as reference. This must be a pre-defined @CycleDetect or @CycleInterval formula.

Output

A waveform representing the phase difference between the cycle fundamentals of two signals for each detected cycle in the cycle detect signal. It has a value of zero before the first and after the last detected cycle.

Description

The CycleFundamentalPhase function determines the phase of the cycle fundamentals of two input signals with a cycle detect signal as reference. The phase difference θ is defined as:

$$\theta = \theta_1 - \theta_2 \text{ (rad)}$$

where:

- $\theta \text{ (rad)}$ = Phase difference in radians from 0 to 2π .
- $\theta_1 \text{ (rad)}$ = Phase of the cycle fundamental of input signal 1
- $\theta_2 \text{ (rad)}$ = Phase of the cycle fundamental of input signal 2

Example

The following example calculates the fundamental phase difference between two sine waves. The sine wave Signal_2 is 90 degrees behind sine wave Signal_1. The cycle detect of Signal_1 is used as reference.

Name	Formula
Signal_1	= @SineWave(1000;1000;5; 0)
Signal_2	= @SineWave(1000;1000;5; -90)
Reference	= @CycleDetect(Formula.Signal_1)
CycleFP	= @CycleFundamentalPhase(Formula.Signal_1; Formula.Signal_2; Formula.Reference)

Result:

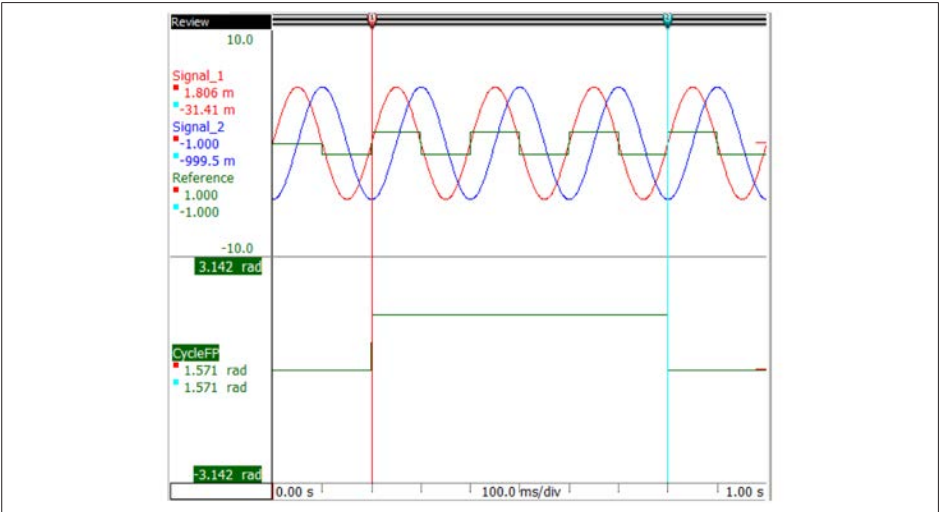


Fig. 6.5 Result of the CycleFundamentalPhase example

See also

"@CycleDetect" on page 312 and "@CycleFundamental" on page 318

6.10 @CycleFundamentalRMS

Function

Returns a waveform representing the **RMS** value of the fundamental of the signal for each detected **cycle** in the **cycle** detect signal.

Syntax

@CycleFundamentalRMS(*Waveform*; *CycleDetect*)

Parameters

<i>Waveform</i>	Input waveform to be used for the calculation of the cycle fundamental RMS.
<i>Cycle detect</i>	The waveform to be used as reference. This must be a pre-defined @CycleDetect or @CycleInterval formula.

Output

Returns a waveform representing the RMS value of the fundamental of the signal for each detected cycle in the cycle detect signal.

Description

The CycleFundamentalRMS function calculates the RMS value of the fundamental of the signal for each detected cycle in the cycle detect signal. It has a value of zero before the first and after the last detected cycle.

Example

The following example calculates the RMS value fundamental RMS value of sine wave. The cycle detect of Signal_1 is used as reference.

Name	Formula
Signal_1	= @SineSquare(1000;1000;5; 0)
Reference	= @CycleDetect (Formula.Signal_1)
CycleFR	= @CycleFundamentalRMS(Formula.Signal_1; Formula.Reference)

Result:

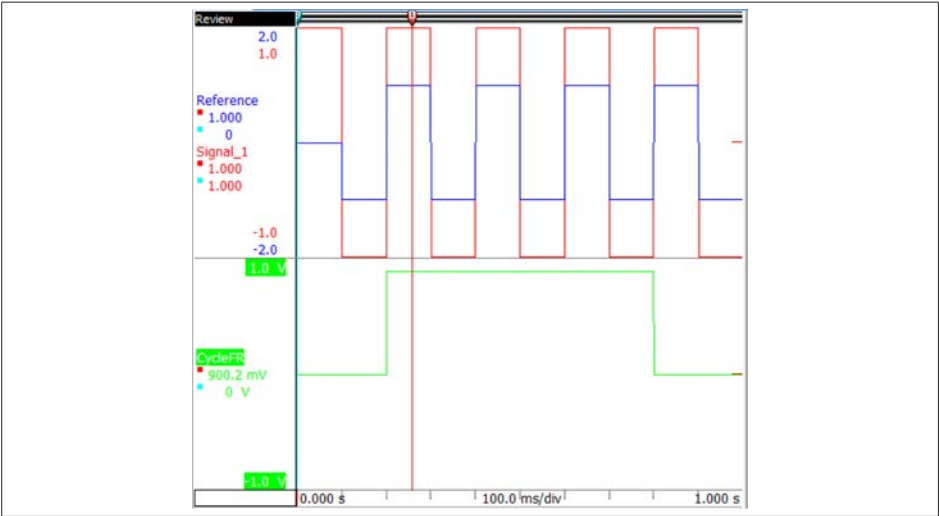


Fig. 6.6 Result of the CycleFundamentalRMS example

See also

"@CycleDetect" on page 312 and "@CycleFundamental" on page 318

6.11 @CycleInterval

Function

Generates a square waveform with a specified **interval** between the high and low states.

Syntax

@CycleInterval(*Waveform*; *Interval*; *StartOffset*)

Settings

<i>Waveform</i>	Waveform to be used as a reference.
<i>Interval</i>	Number: time in seconds representing the half-period of the output square waveform.
<i>StartOffset</i>	Number: time in seconds indicating a shift of the output signal on the X-axis. The default value is 0.

Output

A square waveform with 50% duty cycle. The duration between the start of a high pulse and the end of the following low pulse is equal to 2*Interval.

The output of this formula is equivalent to the output of the @SquareWave formula when the @CycleInterval formula has been defined with the appropriate parameters.

Description

The reference Waveform parameter is only used to determine the start time of the signal and to use it as the start time of the output waveform.

The StartOffset parameter is optional. If the StartOffset is defined, the start time of the output waveform is calculated by adding the StartOffset to the start time of the reference Waveform.

This formula can be used as the reference waveform parameter for the other Cycle formulas. The result is that all the cycles of the calculated math function have the same duration, i.e. they constantly operate on the same amount of samples.

Example

Assume that signal Ch_A1 is recorded at a sampling frequency of 1 MS/s, and it contains 1 MSamples. Then the formula

```
ci = @CycleInterval(Group1.Recorder_A.Ch_A1; 0.1)
```

is equivalent to the formula:

```
sw = @SquareWave(1000000; 1000000; 5)
```

Assuming that Ch_A1 starts at 0 s, this formula will indicate that the cycle start times are located at times 0.2 s, 0.4 s, 0.6 s, 0.8 s, etc.

The formula can be used to calculate a math function on the signal Ch_A1, using a constant cycle length of $2 \times 100 = 200$ ms from the start of the recording – which is equivalent to 200 kSamples ($= 1 \text{ MS/s} \times 0.2 \text{ s}$) of the input signal.

See also

"@CycleDetect" on page 312

6.12 @CycleLevel

Function

Returns a waveform that uses the value of the input signal at the start of a cycle of the reference signal and repeats that value (sample and hold) during half, one or multiple cycles of that reference signal.

Syntax

@CycleLevel(*Waveform*; *Cycles*; *ReferenceWaveform*)

Parameters

<i>Waveform</i>	Input waveform to be used for the input samples.
<i>Cycles</i>	Number of cycles of the reference waveform to repeat the same output value. Possible values: 0.5, 1, 2, 3, etc..
<i>ReferenceWaveform</i>	Waveform to be used as a reference to detect cycles.

Output

A waveform that uses the value of the input signal at the start of a cycle of the reference signal and repeats that value during half, one or multiple cycles of that reference signal.

The waveform value is zero before the first cycle begins and after the last one ends.

Description

For a range of input samples $y(k_1)$ to $y(k_2)$, where k_1 and k_2 are the indices defined by the detected cycle, the output samples $y'(k_1)$ to $y'(k_2)$ will be assigned the value of $y(k_1)$.

The Cycles and ReferenceWaveform parameters are optional.

If the Cycles parameter is omitted, the default value is 1. A value of 0.5 indicates half a cycle of the reference signal.

If the ReferenceWaveform parameter is omitted, a CycleDetect formula is created for the input Waveform.

In order to improve performance in case the same cycle reference signal is needed for multiple functions, you can generate the block cycle signal yourself using the CycleDetect function, and pass it as a reference waveform.

Example

The following example calculates an output signal where the 'y' value is set to a new 'y' value at each zero crossing of the second signal (sig2). The 'y' value is set to the 'y' value of the first signal (sig1) at the zero crossing point. The waveform value is zero before the first cycle begins and after the last one ends.

```
sig1 = @SineWave(1k;1k+1;3;90)
sig2 = @SineWave(1k;1k+1;12)
cd    = @CycleDetect(Formula.sig2;0;0.1)
cl    = @CycleLevel(Formula.sig1;1;Formula.cd)
```

Result:

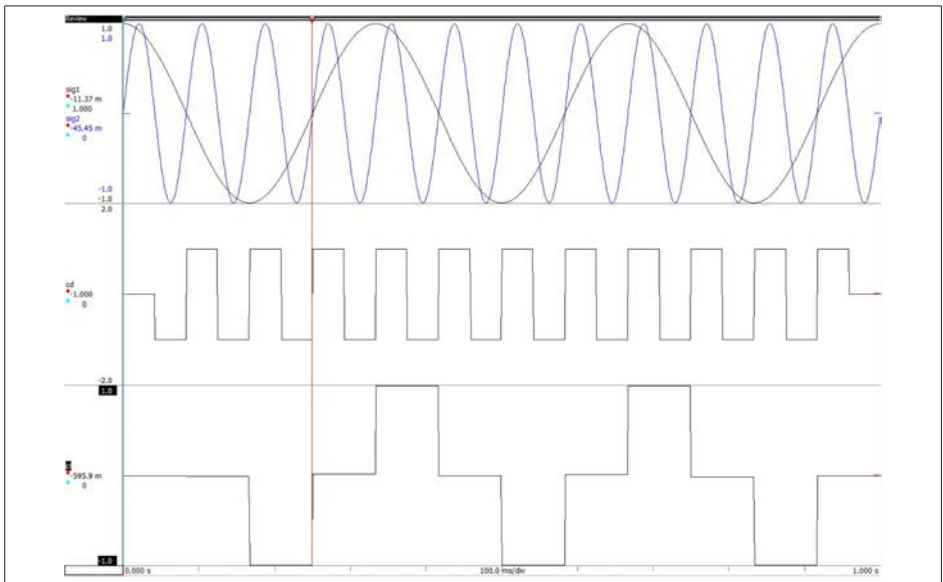


Fig. 6.7 Result of CycleLevel example

See also

"@CycleMax" on page 328, "@CycleMean" on page 329 and "@CycleMin" on page 331

6.13 @CycleMax

Function

Calculates the **maximum** value in every **cycle** detected on a reference signal.

Syntax

@CycleMax(*Waveform*; *Cycles*; *ReferenceWaveform*)

Parameters

<i>Waveform</i>	Input waveform to be used for calculation of math function.
<i>Cycles</i>	Number of cycles of the reference waveform used to calculate one output value.
<i>ReferenceWaveform</i>	Waveform to be used as a reference to detect cycles.

Output

A waveform which contains the maximum value of every detected cycle. The waveform value is zero before the first cycle begins and after the last one ends.

Description

For a range of input samples $y(k_1)$ to $y(k_2)$, where k_1 and k_2 are the indices defined by the detected cycle, if sample $y(k)$ is the maximum (where $k_1 \leq k \leq k_2$), the output samples $y'(k_1)$ to $y'(k_2)$ will be assigned the value of $y(k)$.

The Cycles and ReferenceWaveform parameters are optional.

If the Cycles parameter is omitted, the default value is 1.

If the ReferenceWaveform parameter is omitted, a CycleDetect formula is created for the input Waveform.

See also

"@Max" on page 157

6.14 @CycleMean

Function

Calculates the **mean** value in **every** cycle detected on a reference signal.

Syntax

@CycleMean(*Waveform*; *Cycles*; *ReferenceWaveform*)

Parameters

<i>Waveform</i>	Input waveform to be used for calculation of math function.
<i>Cycles</i>	Number of cycles of the reference waveform used to calculate one output value.
<i>ReferenceWaveform</i>	Waveform to be used as a reference to detect cycles.

Output

A waveform which contains the mean value of every detected cycle. The waveform value is zero before the first cycle begins and after the last one ends.

Description

The formula performs the following operation:

For a range of input samples $y(k_1)$ to $y(k_2)$, where k_1 and k_2 are the indices defined by the detected cycle, the output samples $y'(k_1)$ to $y'(k_2)$ will be assigned the value:

$$\text{Mean} = \frac{1}{N} \sum_{k=k_1}^{k_2} y(k)$$

$$N = k_2 - k_1 + 1$$

The start and end indices (k_1 and k_2) for these segments ("cycles") are calculated from the level crossing times determined by the CycleDetect algorithm, which is applied on the ReferenceWaveform.

The Cycles and ReferenceWaveform parameters are optional.

If the Cycles parameter is omitted, the default value is 1.

If the ReferenceWaveform parameter is omitted, a CycleDetect formula is created for the input Waveform.

See also

"@Mean" on page 162

6.15 @CycleMin

Function

Calculates the **minimum** value in every **cycle** detected on a reference signal.

Syntax

@CycleMin(*Waveform*; *Cycles*; *ReferenceWaveform*)

Parameters

<i>Waveform</i>	Input waveform to be used for calculation of math function.
<i>Cycles</i>	Number of cycles of the reference waveform used to calculate one output value.
<i>ReferenceWaveform</i>	Waveform to be used as a reference to detect cycles.

Output

A waveform which contains the minimum value of every detected cycle. The waveform value is zero before the first cycle begins and after the last one ends.

Description

For a range of input samples $y(k_1)$ to $y(k_2)$, where k_1 and k_2 are the indices defined by the detected cycle, if sample $y(k)$ is the minimum (where $k_1 \leq k \leq k_2$), the output samples $y'(k_1)$ to $y'(k_2)$ will be assigned the value of $y(k)$.

The Cycles and ReferenceWaveform parameters are optional.

If the Cycles parameter is omitted, the default value is 1.

If the ReferenceWaveform parameter is omitted, a CycleDetect formula is created for the input Waveform.

See also

"@Min" on page 167

6.16 @CyclePeriod

Function

Calculates the **period** of every **cycle** detected on a reference signal.

Syntax

@CyclePeriod(*Waveform*; *Cycles*; *ReferenceWaveform*)

Parameters

<i>Waveform</i>	Input waveform to be used for calculation of math function.
<i>Cycles</i>	Number of cycles of the reference waveform used to calculate one output value.
<i>ReferenceWaveform</i>	Waveform to be used as a reference to detect cycles.

Output

A waveform which contains the period of every detected cycle. The waveform value is zero before the first cycle begins and after the last one ends.

Description

This function is the inverse of the @CycleFrequency function. If the parameters given for the two functions are identical and the output of @CycleFrequency is F at time t , the output of @CyclePeriod at time t will then be $T = 1/F$.

The Cycles and ReferenceWaveform parameters are optional.

If the Cycles parameter is omitted, the default value is 1.

If the ReferenceWaveform parameter is omitted, a CycleDetect formula is created for the input Waveform.

See also

"@Period" on page 184, "@CycleFrequency" on page 316

6.17 @CyclePhase

Function

Calculates the **phase** difference θ between two waveforms based on the rising edge level crossings of the two waveforms.

Syntax

@CyclePhase(*InputWaveform*; *ReferenceWaveform*)

Parameters

<i>InputWaveform</i>	The waveform to be used to determine the time $t\theta$. This can be an analog or digital channel or a predefined @CycleDetect formula.
<i>ReferenceWaveform</i>	The waveform to be used as reference for the calculation of level crossing times and cycle period $T\theta$. This can be an analog or digital channel or a predefined @CycleDetect or @CycleInterval formula.

Output

A waveform containing the phase difference for each detected cycle of the reference waveform. It has a value of zero before the first and after the last rising edge level crossing of the reference waveform. If for a cycle no rising edge level crossing is found for the input waveform the phase difference is undefined for this cycle.

Description

The CyclePhase function calculates the phase difference θ between two waveforms based on the rising edge level crossings of the two waveforms.

The phase difference θ is defined as:

$$\theta = 2 \pi * t \theta / T \theta \text{ (rad)}$$

where:

- $T\theta$ (s) = Cycle period of the reference signal.
- $t\theta$ (s) = Time between a rising edge level crossing of the reference signal and the first following rising edge level crossing of the input signal which lies in $T\theta$.

For faster results, use manually created @CycleDetect or @CycleInterval formulas for the signal and the reference signal and use these as input signals for the CyclePhase function.

Example

The following example calculates the phase difference between two sine waves. The sine wave Signal is 90 degrees behind sine wave Reference.

Name	Formula
Signal	= @SineWave(1000;1000;5; -90)
Reference	= @SineWave(1000;1000;5)
CyclePhase	= @CyclePhase(Formula.Signal;Formula.Reference)

Result:

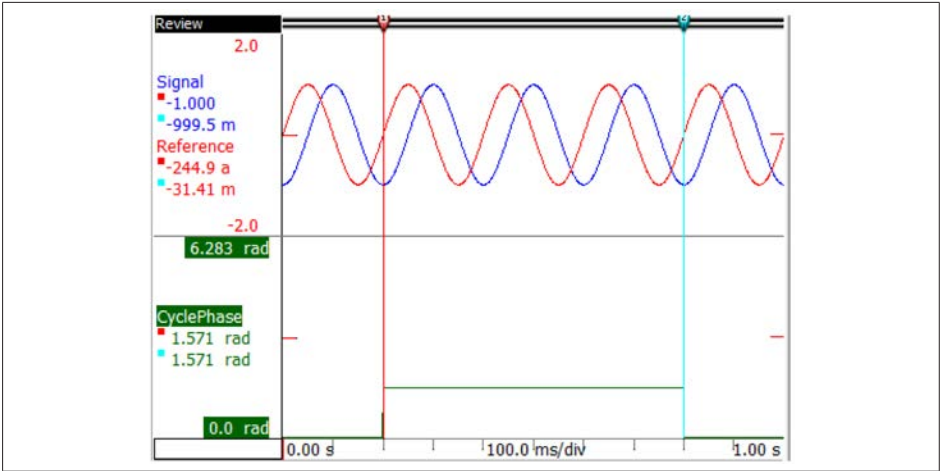


Fig. 6.8 Result of the CyclePhase example

See also

"@CycleDetect" on page 312

6.18 @CycleRPM

Function

Calculates the **revolutions per minute** (RPM) of every cycle detected on a reference signal.

Syntax

`@CycleRPM(Waveform; Cycles; ReferenceWaveform)`

Parameters

<i>Waveform</i>	Input waveform representing angular position in degrees.
<i>Cycles</i>	Number of cycles of the reference waveform used to calculate one output value.
<i>ReferenceWaveform</i>	Waveform to be used as a reference to detect cycles.

Output

Waveform representing the RPM value for each detected cycle of the source channel.

Description

The revolutions per minute (RPM) is calculated by determining the amount of angular rotation of the waveform that represents the angular position over the time periods that are defined by the detected cycles.

The Cycles and ReferenceWaveform parameters are optional.

If the Cycles parameter is omitted, the default value is 1.

If the ReferenceWaveform parameter is omitted, a CycleDetect formula is created for the input Waveform.

Example

Given that the input signal is available as Formula.input, the function @CycleDetect determines the zero level crossings which are used by the @CycleRPM function to determine the RPM value for each detected cycle:

Name	Formula
------	---------

cd	= @CycleDetect(Formula.input; 0.0, 0.1)
rpm	= @CycleRPM(Formula.input; 1; Formula.cd)

See also

"@CycleDetect" on page 312

6.19 @CycleRMS

Function

Calculates the **root mean square** value of every cycle detected on a reference signal.

Syntax

@CycleRMS(*Waveform*; *Cycles*; *ReferenceWaveform*)

Parameters

Waveform Input waveform to be used for calculation of math function.
Cycles Number of cycles of the reference waveform used to calculate one output value.
ReferenceWaveform Waveform to be used as a reference to detect cycles.

Output

A waveform which contains the RMS value of every detected cycle. The waveform value is zero before the first cycle begins and after the last one ends.

Description

For a range of input samples $y(k_1)$ to $y(k_2)$, where k_1 and k_2 are the indices defined by the detected cycle, the output samples $y'(k_1)$ to $y'(k_2)$ will be assigned the value:

$$\text{RMS} = \sqrt{\frac{1}{N} \sum_{k=k_1}^{k_2} (y(k))^2}$$

$$N = k_2 - k_1 + 1$$

The start and end indices (k_1 and k_2) for these segments ("cycles") are calculated from the level crossing times determined by the CycleDetect algorithm, which is applied on the ReferenceWaveform.

The Cycles and ReferenceWaveform parameters are optional.

If the Cycles parameter is omitted, the default value is 1.

If the ReferenceWaveform parameter is omitted, a CycleDetect formula is created for the input Waveform.

See also

"@RMS" on page 216

6.20 @CycleStdDev

Function

Calculates the **standard deviation** of every **cycle** detected on a reference signal.

Syntax

@CycleStdDev(*Waveform*; *Cycles*; *ReferenceWaveform*)

Parameters

Waveform Input waveform to be used for calculation of math function.
Cycles Number of cycles of the reference waveform used to calculate one output value.
ReferenceWaveform Waveform to be used as a reference to detect cycles.

Output

A waveform which contains the standard deviation from the mean value of every detected cycle. The waveform value is zero before the first cycle begins and after the last one ends.

Description

For a range of input samples $y(k_1)$ to $y(k_2)$, where k_1 and k_2 are the indices defined by the detected cycle, the output samples $y'(k_1)$ to $y'(k_2)$ will be assigned the value:

$$\text{StdDev} = \sqrt{\frac{1}{N} \sum_{k=k_1}^{k_2} (y(k) - \bar{y})^2}$$

$$\bar{y} = \frac{1}{N} \sum_{k=k_1}^{k_2} y(k)$$

$$N = k_2 - k_1 + 1$$

The start and end indices (k_1 and k_2) for these segments ("cycles") are calculated from the level crossing times determined by the CycleDetect algorithm, which is applied on the ReferenceWaveform.

The Cycles and ReferenceWaveform parameters are optional.

If the Cycles parameter is omitted, the default value is 1

If the ReferenceWaveform parameter is omitted, a CycleDetect formula is created for the input Waveform.

See also

"@ StdDev" on page 233

6.21 @CycleTHD

Function

Calculates the **Total Harmonic Distortion (THD)** of every cycle detected on a reference signal.

Syntax

@CycleTHD(*Waveform*; *Cycles*; *ReferenceWaveform*)

Parameters

- Waveform*Input waveform to be used for the calculation of the Cycle THD.
- Cycles*Number of cycles of the reference waveform used to calculate one output value.
- ReferenceWaveform*Waveform to be used as a reference to detect cycles.

Output

Waveform that contains the THD value of each detected cycle as a percentage. The waveform value is zero before the first and after the last detected cycles.

Description

The total harmonic distortion (THD) is a measurement of the harmonic distortion that is present in a signal. The measurement can be applied both signals that represent voltage as well as current levels.

The *THD* for voltage is defined as:

$$THD_v = \frac{\sqrt{V_{rms}^2 - V_1^2}}{V_1}$$

Where

V_{rms} = Root Mean Square Voltage

V_1 = Root Mean Square Voltage of the 1st Harmonic (Fundamental frequency)

The THD for Current is defined as:

$$THD_i = \frac{\sqrt{I_{rms}^2 - I_1^2}}{I_1}$$

Where

I_{rms} = Root Mean Square Current
 I_1 = Root Mean Square Current of the 1st Harmonic (Fundamental frequency)

The Cycles and ReferenceWaveform parameters are optional.

If the Cycles parameter is omitted, the default value is 1.

If the ReferenceWaveform parameter is omitted, a CycleDetect formula is created for the input Waveform.

Example

Given that the input signal is available as Formula.input, the function @CycleDetect determines the zero level crossings which are used by the @CycleTHD function to determine the THD value for each detected cycle:

Name	Formula
cd	= @CycleDetect(Formula.input; 0.0, 0.1)
thd	= @CycleTHD(Formula.input; 1; Formula.cd)

See also

"@CycleDetect" on page 312 and "@CycleFundamental" on page 318

7 Pulse Measurement and Analysis

7.1 General

IEEE Standard Pulse Terms and Definitions

The contents of this section are based on the IEEE Std 194-1977 and IEEE Std 181-1977.

For more information visit the website of the IEEE Standards Association:

www.standards.ieee.org/

Wave, Pulse and Transition

- **Wave:** A modification of the physical state of a medium which propagates in the medium as a function of time as a result of one or more disturbances.
- **Pulse:** A wave which departs from a first nominal state, attains a second nominal state, and ultimately returns to the first nominal state. Throughout the remainder of this document the term pulse is included in the term wave.
- **Transition:** A portion of a wave or pulse between a first nominal state. Throughout the remainder of this document the term transition is included in the terms pulse and wave.

Refer to the following diagram for an overview of the single pulse waveform.

The Single Pulse

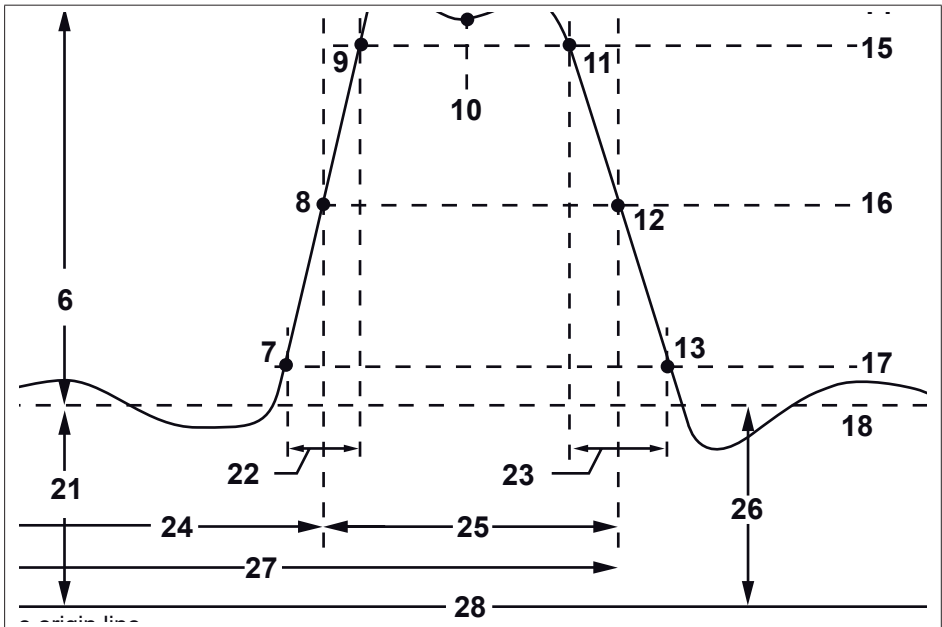


Fig. 7.1 Diagram of the single pulse waveform

1	Time origin line	A line of constant and specified time which, unless otherwise specified, has a time equal to zero and passes through the first datum time t_0 of a waveform epoch.
2	Pulse start line	Pulse Start Line. The time reference line at pulse start time.
3	Top center point	A specified time referenced point or magnitude referenced point on a pulse waveform top. If no point is specified, the top center point is time referenced point at the intersection of a pulse waveform and the top center line.
4	Pulse stop line	Pulse Stop Line. The time reference line at pulse stop time.
5	Top magnitude	The magnitude of the top as obtained by a specified procedure or algorithm.

7 Pulse Measurement and Analysis

6	Pulse amplitude	The algebraic difference between the top magnitude and the base magnitude.
7	Proximal (First transition point)	A magnitude referenced point at the intersection of a waveform and a proximal line.
8	Mesial (First transition point)	A magnitude referenced point at the intersection of a waveform and a mesial line.
9	Distal (First transition point)	A magnitude referenced point at the intersection of a waveform and a distal line.
10	Top center line	The time reference line at the average of pulse start and pulse stop time.
11	Distal (Last transition point)	A magnitude referenced point at the intersection of a waveform and a distal line.
12	Mesial (Last transition point)	A magnitude referenced point at the intersection of a waveform and a mesial line.
13	Proximal (Last transition point)	A magnitude referenced point at the intersection of a waveform and a proximal line.
14	Top line	The magnitude reference line at the top magnitude.
15	Distal line	A magnitude reference line at a specified magnitude in the distal region of a pulse waveform. Unless otherwise specified, the distal line is at the 90 percent reference magnitude.
16	Mesial line	A magnitude reference line at a specified magnitude in the mesial region of a pulse waveform. Unless otherwise specified, the mesial line is at the 50 percent reference magnitude.
17	Proximal line	A magnitude reference line at a specified magnitude in the proximal region of a pulse waveform. Unless otherwise specified, the proximal line is at the 10 percent reference magnitude.
18	Baseline	The magnitude reference line at the base magnitude.
19	First base point	Unless otherwise specified, the first datum point in a pulse epoch.
20	Last base point	Unless otherwise specified, the last datum point in a pulse epoch.
21	Base magnitude	The magnitude of the base as obtained by a specified procedure or algorithm. Unless otherwise specified, both portions of the base are included in the procedure or algorithm.

22	First transit duration	The transit duration of the first transition waveform in a pulse waveform.
23	Last transit duration	The transit duration of the last transition waveform in a pulse waveform.
24	Pulse start time	The instant specified by a magnitude referenced point on the first transition of a pulse waveform. Unless otherwise specified, the pulse start time is at the mesial point on the first transition.
25	Pulse duration	The duration between pulse start time and pulse stop time.
26	Offset	The algebraic difference between two specified magnitude reference lines. Unless otherwise specified, the two magnitude reference lines are the waveform baseline and the magnitude origin line.
27	Pulse stop time	The instant specified by a magnitude referenced point on the last transition of a pulse waveform. Unless otherwise specified, the pulse stop time is at the mesial point on the last transition.
28	Magnitude origin line	A line of specified magnitude which, unless otherwise specified, has a magnitude equal to zero and extends through the waveform epoch.
29	Pulse waveform epoch	The span of time for which waveform data are known or knowable. A waveform epoch manifested by equations may extend in time from $-\infty$ to $+\infty$ or, like all waveform data, may extend from a first datum time to t_0 to a second datum time t_1 .

8 IIR Filters

8.1 Introduction

The Perception formula database contains a series of IIR filters. These filters implement a class that filters the input sequence using the direct form IIR filter specified by the feedback and feedforward coefficients. IIR is an abbreviation of Infinite Impulse Response. The relation between the output and input signal looks like the following formula Fig. 8.1

$$y(n) = \sum_{k=0}^N b_k x(n-k) - \sum_{k=1}^N a_k y(n-k)$$

Fig. 8.1 Infinite Impulse Response – output, input

where:

N = Filter order

b_i = Feedforward filter coefficients, often also called the zeros

a_i = Feedback filter coefficients, often also called the poles

$x(n)$ = Input signal

$y(n)$ = Output signal

The block diagram looks like figure Fig. 8.2

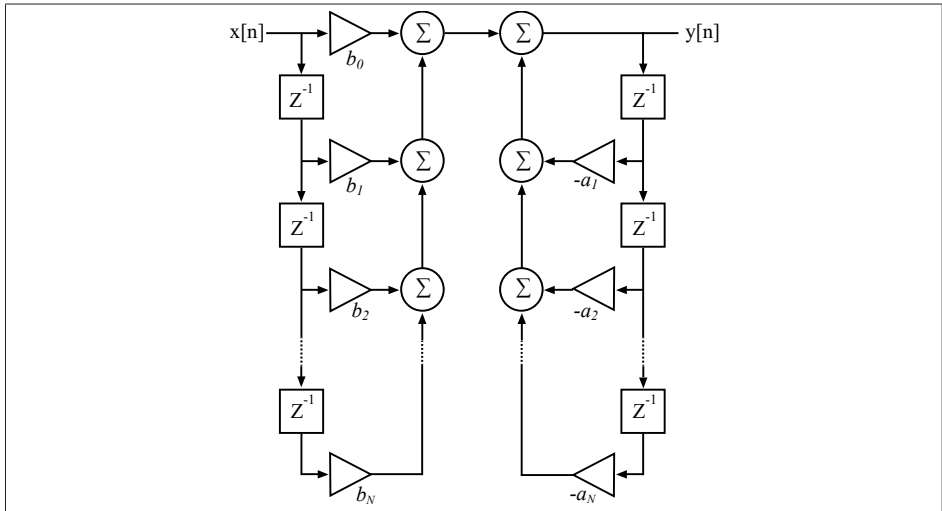


Fig. 8.2 Infinite Impulse Response – Block diagram

All the filter formulas start with “@Filter”. The complete list of filter functions can be found below:

@FilterButterworthLP
@FilterButterworthHP
@FilterButterworthBP
@FilterButterworthBS
@FilterBesselLP
@FilterBesselHP
@FilterBesselBP
@FilterBesselBS
@FilterChebyshevLP
@FilterChebyshevHP
@FilterChebyshevBP
@FilterChebyshevBS

Three different types of filters are available:

- **Bessel**
- **Butterworth**
- **Chebyshev**

For each type there is a **Lowpass** (LP), **Highpass** (HP), **Bandpass** (BP) and **Bandstop** (BS) implementation.

8.1.1 Bessel

A Bessel filter is a type of linear filter with a maximally flat group delay (maximally linear phase response). Bessel filter has a good step response with very little overshoot; however the frequency response is worse than that of Butterworth filter. Bessel filters have near linear phase response. The applications of Bessel are where the phase relations are critical.

This filter is best used in the time domain.

It is also recommended to use a Bessel filter when the signal to be filtered contains non sinusoidal signals like for example square waveforms or noise and you want to use the filtered signal for timing purposes. Bessel is then the best choice because all of the different frequency components of the original signal will have almost the same delay after they have been filtered.

Advantages:

- Best step response-very little overshoot or ringing.

Disadvantages:

- Slower initial rate of attenuation beyond the pass-band than Butterworth.

8.1.2 Butterworth

The Butterworth filter is a type of signal processing filter designed to have as flat a frequency response as possible in the passband so that it is also termed a maximally flat magnitude filter. It has a good frequency response with no ripple; however the phase response may be quite nonlinear especially for high order filters.

This filter is best used in the frequency domain.

If the signal to be filtered is a sinusoidal signal than Butterworth is often the best choice, the nonlinear phase response doesn't matter too much because we are dealing with only one dominant frequency in the input signal, therefore the deformation of the output signal will be minimal.

Advantages:

- Maximally flat magnitude response in the pass-band.
- Good all-around performance.
- Pulse response better than Chebyshev.
- Rate of attenuation better than Bessel.

Disadvantages:

- Some overshoot and ringing in step response.

8.1.3 Chebyshev (Type I)

Chebyshev filters have a steeper roll-off and more passband ripple (type I) than Butterworth filters. Chebyshev filters have the property that they minimize the error between the idealized and the actual filter characteristic over the range of the filter, but with ripples in the passband.

Compared to a Butterworth filter, a Chebyshev filter can achieve a sharper transition between the passband and the stopband with a lower order filter.

This filter is best used in the frequency domain.

Advantages:

- Better rate of attenuation beyond the pass-band than Butterworth.

Disadvantages:

- Ripple in pass-band.
- Considerably more ringing in step response than Butterworth.

8.1.4 Magnitude Spectrum

The following Fig. 8.3 shows the magnitude spectrums of the three different 2nd order lowpass IIR filters. The Bessel filter has the worsted frequency response in the stop-band. The Chebyshev has the steepest roll-off, but has a ripple in the passband.

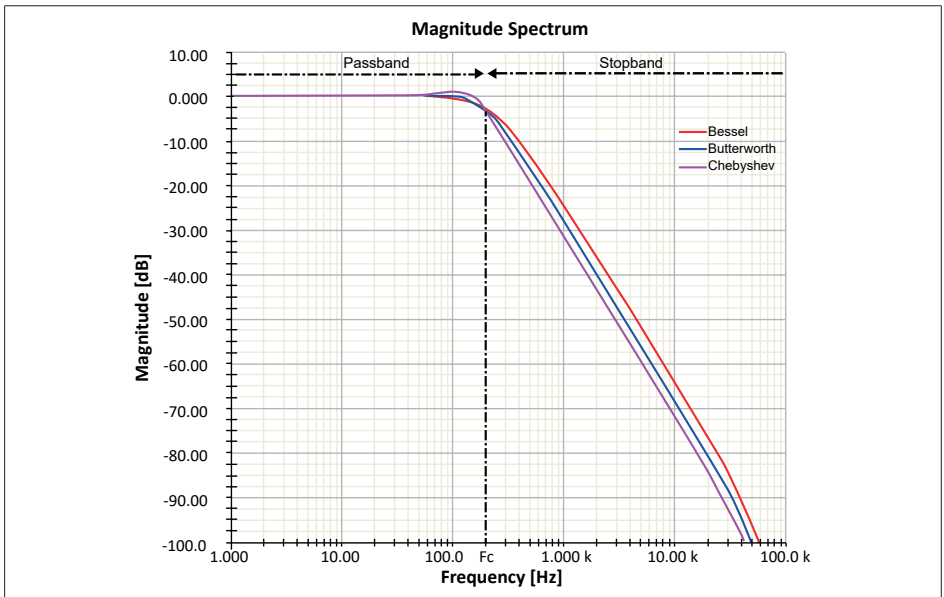


Fig. 8.3 Magnitude spectrum of lowpass filters

Fig. 8.4 shows the passband of the three different filters.

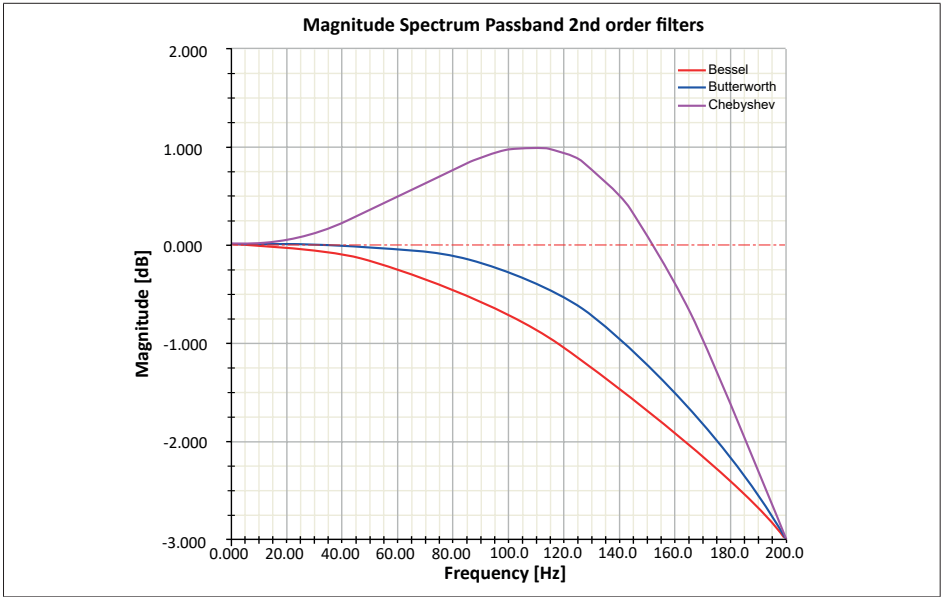


Fig. 8.4 Magnitude Spectrum Passband of filters

Fig. 8.5 shows the stopband of the three filters.

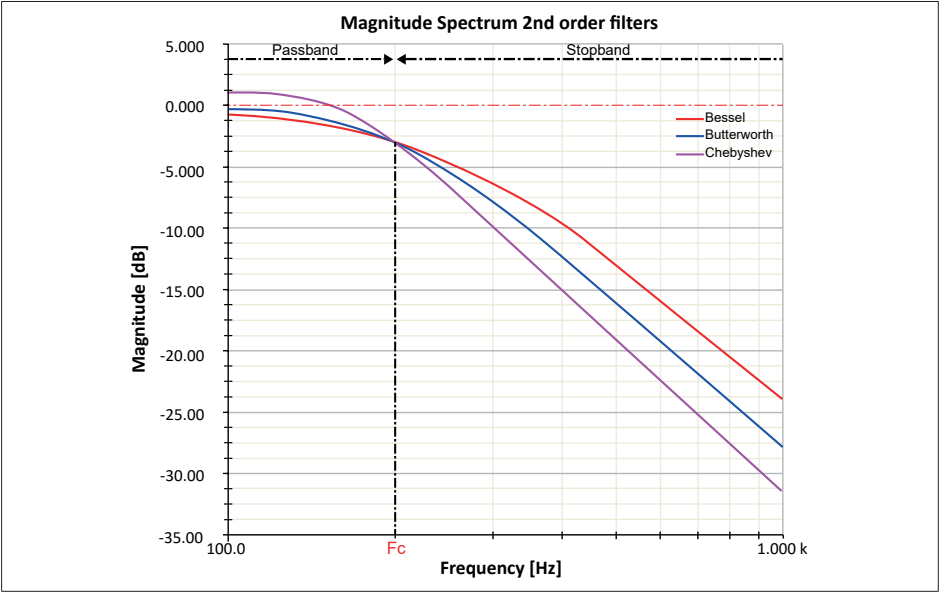


Fig. 8.5 Magnitude Spectrum – stopband

8.1.5 Impulse Response

The following Fig. 8.6 shows the Impulse response of the three different 2nd order low-pass IIR filters. The Bessel filter provides the best impulse response.

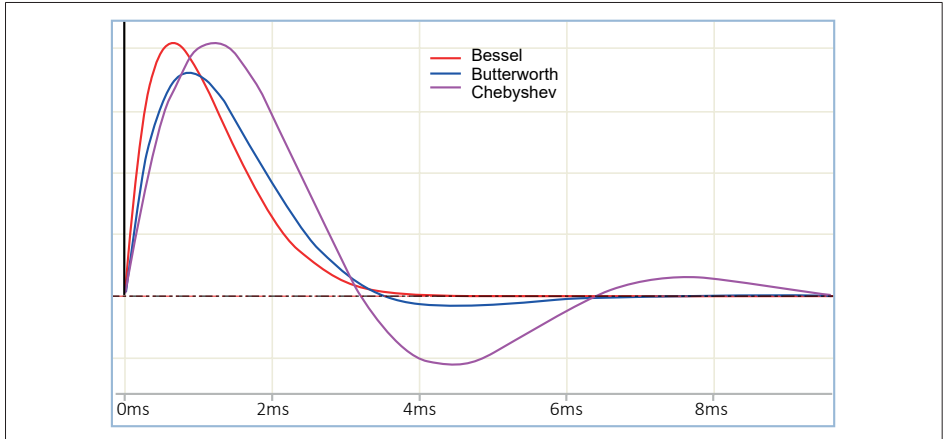


Fig. 8.6 Impulse response of lowpass filters

8.1.6 Step Response

Fig. 8.7 below shows the step response of the three different 2nd order lowpass IIR filters.

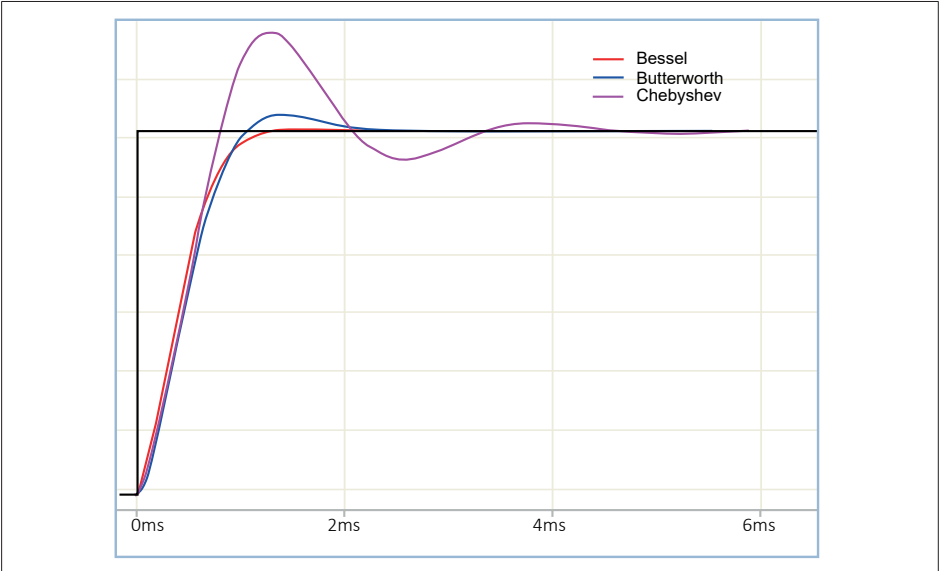


Fig. 8.7 Step response of lowpass filters

Again the Bessel filter gives the best step response. The above impulse and step responses can be reconstructed by using the following formulas:

Num	Name	Formula	Units
1	Pulse	@Pulse(80k; 80k; 20k)	
2	ButherworthPR	@FilterButterworthLP[Formula.Pulse; 2; 200]	
3	BesselPR	@FilterBesselLP[Formula.Pulse; 2; 200]	
4	ChebyshevPR	@FilterChebyshevLP[Formula.Pulse; 2; 200; 3]	
5			
6	Step	@Pulse(80k; 80k; 20k; 10k)	
7	ButherworthSR	@FilterButterworthLP[Formula.Step; 2; 200]	
8	BesselSR	@FilterBesselLP[Formula.Step; 2; 200]	
9	ChebyshevSR	@FilterChebyshevLP[Formula.Step; 2; 200; 3]	
10			

Fig. 8.8 Filter response

The above formulas can also be used to examine the step and impulse responses of any filter you can create with the current filter formulas of Perception.

8.1.7 Phaseless Filtering

Phaseless filtering is done by using the reverse order technique. This technique filters the signal twice. The order of the used filter is half of the entered order value ($Order/2$). During the first step the signal will be filtered as usual, during the second step the filtered output will be filtered again using the same filter but the data points will go through the filter in reverse order. Because we use the filter twice the overall filter order will be equal to the entered order value ($Order$). If the entered order is an odd value than the filter order will be the first even value below this odd value. If for example the entered order value is 7 then the filter will have an order value of 6. The phase shift of the filtered signal will be zero, see Fig. 8.9.

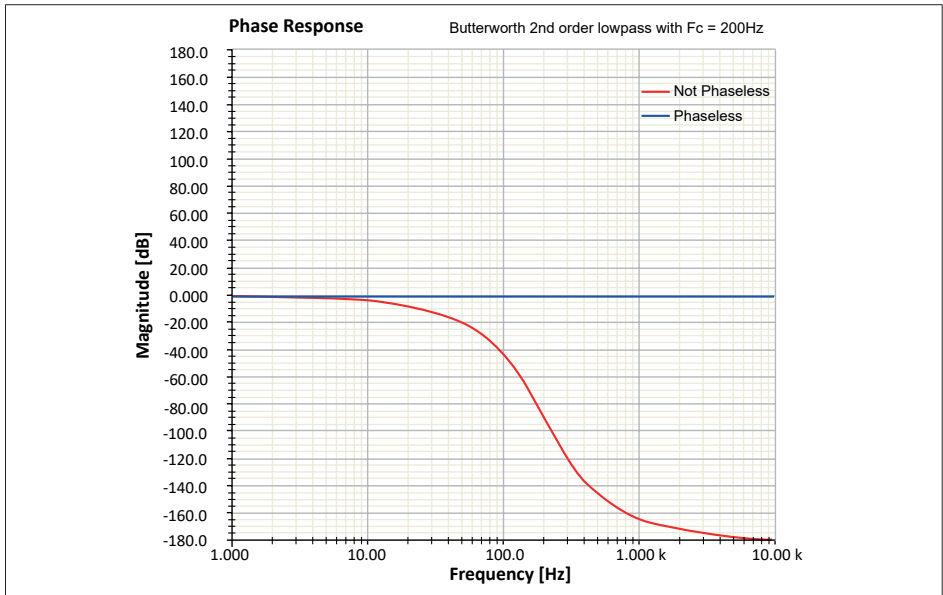


Fig. 8.9 Phase response of filters

There will be a difference between the magnitude spectrums of the phaseless and not phaseless filters. This difference is because the phaseless filter is the result of filtering with a filter with half the order twice, therefore the attenuation at the cutoff frequency is not -3 dB but -6 dB, see Fig. 8.10.

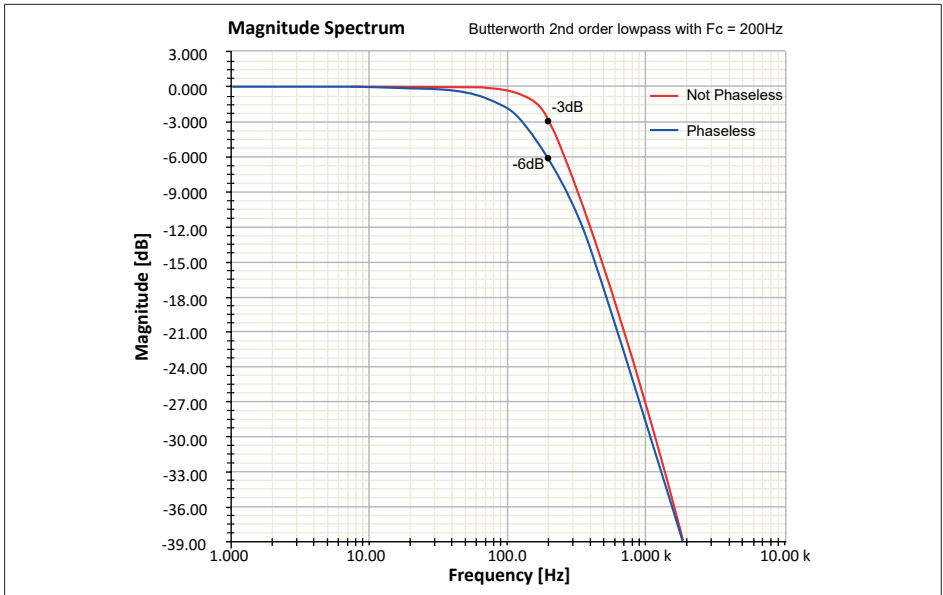


Fig. 8.10 Magnitude Spectrum - Attenuation

8.1.8 Importance of sampling rate and cutoff frequency

It is important to know the influence off the sampling rate of a discrete signal to be filtered and the filter cutoff frequency. The sampling rate must be at least twice the cutoff frequency, this is also known as the Nyquist frequency.

The filter magnitude curve will also be influenced by the sampling frequency. A second order filter has a roll-off of 12 dB/Octave or 40 dB/Decade, however for frequencies near half the sampling rate the filter behaves different, see Fig. 8.11. You should be aware of this difference if you want to compare digital filtering with analog filtering. Analog filters roll off with a constant slope for the complete stopband.

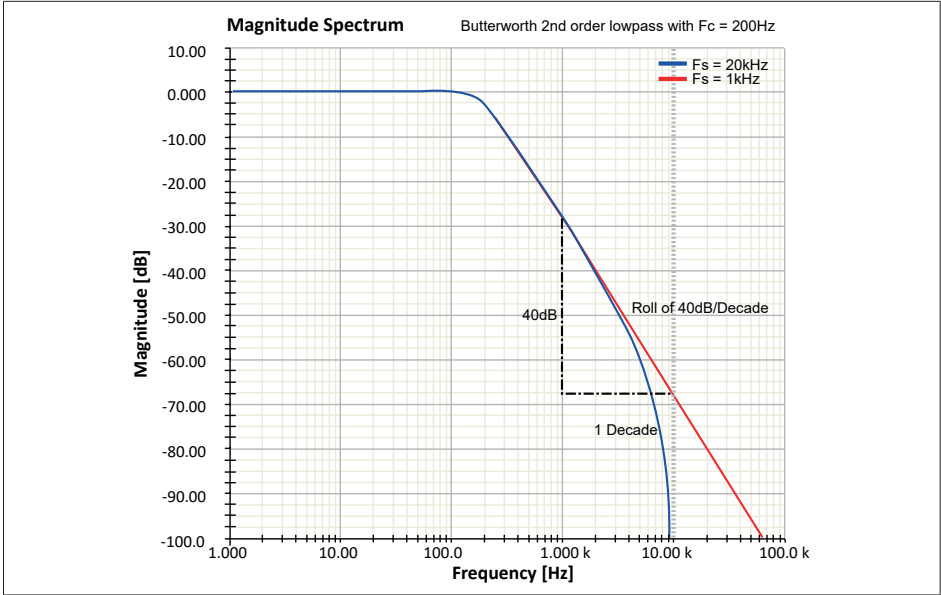


Fig. 8.11 Frequencies at half sample rate

Index

A

ABS	55
ACosine.....	56
Analysis	
Constants	15
Formula sheet	12
Functions.....	16
Operators.....	12
Tools	12
Variables	15
And.....	57
Area.....	59
Arithmetic Operations.....	45
Addition	45
Division	51
Modulo.....	54
Multiplication	49
Subtraction.....	47
Unary minus	53
ASine.....	61
ATan.....	62
ATan2.....	63

B

BlockFFT.....	66
---------------	----

C

Clip	68
Comparator.....	70
Cos	72
Creating formulas	19
Entering comment	25
Entering formulas	25
Function creator.....	19
Problems in formulas	26
CurveFitting.....	73
Cut.....	75
Cycle Math Formulas.....	307
CycleArea.....	307
CycleCount	309
CycleCrestFactor.....	310
CycleDetect	312

CycleEnergy.....	314
CycleFrequency.....	316
CycleFundamental	318
CycleFundamentalPhase.....	320
CycleFundamentalRMS	322
CycleInterval.....	324
CycleLevel.....	326
CycleMax.....	328
CycleMean.....	329
CycleMin	331
CyclePeriod	332
CyclePhase.....	333
CycleRMS	336
CycleRPM	335
CycleStdDev	338
CycleTHD	340
Cycles	77

D

Definitions.....	15
Diff.....	79
DQ0Transformation	81

E

Energy	85
EqualTo.....	87
Exp	88
ExpWave	89

F

FallTime	91
FilterBesselBP	108
FilterBesselBS	110
FilterBesselHP.....	106
FilterBesselLP	104
FilterBiquad	93
FilterButterworthBP	100
FilterButterworthBS	102
FilterButterworthHP	98
FilterButterworthLP	96
FilterChebyshevBP	117
FilterChebyshevBS.....	119
FilterChebyshevHP.....	115
FilterChebyshevLP	112

FormatDate.....	123	IIF.....	140
FormatTime.....	125	IIR Filters.....	346
Formula database functions.....	33	Bessel.....	347
Exception.....	33	Butterworth.....	348
Function Overview.....	35	Chebyshev (Type I).....	348
Function Overview = Cycle Math section.....	43	Cutoff frequency.....	355
Function Overview = HIC section.....	42	Impulse Response.....	352
Formula database sheet.....	12	Magnitude Spectrum.....	349
Formula menu.....	30	Phaseless Filtering.....	354
Load Formulas.....	30	Sampling rate.....	355
Move Sheet.....	32	Step Response.....	353
Print Formulas.....	31	Installation.....	10
Save Formulas.....	31	Install the Analysis option.....	10
Frequency.....	121	Key information.....	10
G		Integrate.....	142
GetNumberFromString.....	127	IntLookUp.....	143
GreaterEqualThan.....	128	IntLookUp12.....	146
GreaterThan.....	129	IsNaN.....	148
H		L	
HIC Formulas.....	286	Layout.....	17
Con3ms.....	287	Modifying the layout.....	17
Con3ms_T.....	289	Length.....	151
Cum3ms.....	290	LessEqualThan.....	152
Cum3ms_T.....	292	LessThan.....	153
HIC.....	293	LICENSE AGREEMENT and WARRANTY 3	
HIC15.....	297	Ln.....	154
HIC15EndTime.....	299	Log.....	156
HIC15StartTime.....	298	M	
HIC36.....	300	Max.....	157
HIC36EndTime.....	302	MaxNum.....	159
HIC36StartTime.....	301	MaxPos.....	160
HICEndTime.....	296	Mean.....	162
HICStartTime.....	295	MedianFilter.....	164
Histogram.....	130	Min.....	167
HSFundamentalFrequency.....	132	MinNum.....	169
HSOneHarmonicAmplitude.....	133	MinPos.....	170
HSOneHarmonicPhase.....	138	N	
I		NextHillPos.....	172
IEEE Standard Pulse Terms and Definitions.....	342	NextLvlCross.....	174
		NextValleyPos.....	176
		Noise.....	178

Not	179	SweepEndTime	236
NumberOfSweeps	180	SweepNumberAtTime.....	237
O		SweepSource.....	238
Or.....	182	SweepStartTime.....	239
P		SweptSineWave.....	240
paceVectorTransformation	228	T	
Perception		Tan	242
Introduction.....	10	TimeMaxAbove	243
Period.....	184	TimeMaxAboveBegin.....	245
Pow	186	TimeMaxBelow.....	247
PrevHillPos	187	TimeMaxBelowBegin	249
PrevLvlCross.....	189	TimeMinAbove	251
PrevValleyPos.....	191	TimeMinAboveBegin.....	253
Pulse	193	TimeMinBelow.....	255
PulseWidth	195	TimeMinBelowBegin.....	257
R		TimeTotalAbove	259
Ramp.....	197	TimeTotalAboveBegin.....	261
ReadAsciiFile.....	199	TimeTotalBelow	263
ReadLogFile.....	201	TimeTotalBelowBegin.....	265
Reduce.....	205	TriggerTime	267
RefCheck	206	TriggerTimeToText	269
Reference guide	55	TrueFrequency.....	271
Cycle Math.....	303	TrueRMS	273
HIC	286	TrueRMSRef	274
RelativeTime2Local	208	V	
RelativeTime2UTC.....	210	Value	276
RemoveGlitch	212	W	
Res2.....	213	Warranty	3
RiseTime.....	214	Wave, Pulse and Transition	342
RMS	216	X	
S		XDelta	277
SAEJ211Filter.....	218	XDeltaHigh.....	278
Sin	219	XDeltaLow	279
SineWave	220	XFirst.....	280
Single Pulse.....	343	XLast.....	281
Smooth	222	XShift	282
SpaceVectorInverseTransformation ...	224	XYArray	283
Sqrt	231	Y	
SquareWave.....	232	YArray.....	285
StdDev.....	233		
Sweep	235		

