RTLvision PRO Reference Manual

Table of Contents

Copyright Notice and Proprietary Information

This software and documentation contain confidential and proprietary information that is the property of Altair Engineering, Inc.. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the license agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Altair Engineering, Inc., or as expressly provided by the license agreement.

Associated third party license terms can be found in the Appendix.

Right to Copy Documentation

This publication is protected by copyright law and international treaties and contains trade secrets and proprietary information owned by Altair Engineering, Inc.. This publication may not be copied, reproduced, modified, published, uploaded, posted, transmitted, or distributed in any way, without prior written permission from Altair Engineering, Inc.. The license agreement with Altair Engineering, Inc. permits licensee to make copies of the documentation for its internal use only. Each copy shall include all copyrights, trademarks and proprietary rights notices, if any.

Trademarks

RTLvision® is a registered trademark of Altair Engineering, Inc.. All other product or company names may be trademarks of their respective owners.

Disclaimer

Information in this publication is subject to change without notice and does not represent a commitment on the part of Altair Engineering, Inc.. Except as may be explicitly set forth in such agreement, Altair Engineering, Inc. does not, and expressly disclaims, any representations or warranties as to the completeness, accuracy or usefulness of the information contained in this document. Altair Engineering, Inc. does not warrant that use of such information will not infringe any third party rights, nor does Altair Engineering, Inc. assume any liability for damages or costs of any kind that may result from use of such information.

Introduction

This documentation is for Altair Engineering, Inc.'s visualization tool RTLvision PRO. It is a netlist visualization tool for debugging, verification and documentation. A list of key features is below. Please start reading the Quickstart Guide.

Document Index

Document	Description
Quickstart	This document describes the basic features of RTLvision PRO and provides an overview of the most important GUI functions.
Installation Notes	These notes describe how to unpack and install the software (and the FlexNet based license mechanism) on UNIX or Windows platforms.
Changelog	This list documents the most important of changes and bug fixes between the releases.
Reference Manual	This manual describes the features and usage of the RTLvision PRO GUI. Please note: all dialog windows provide a help button ("?") to display the appropriate section of this manual in a separate help window. The File Open chapter describes how to read digital design files. The Export Schematic chapter describes the schematic export features to schematic entry tools.
API Doc	The API documentation describes the RTLvision PRO Tcl/Tk-based Application Programming Interface.
Tutorials	The Command Line Tools Tutorial demonstrates the usage of the command line tools. The Symlib Tutorial explains how to use custom symbol shapes. If you plan to use the Clock Domain Analyzer then see the Clock Domain Analyzer Tutorial for more information. For using the Tcl API please read the API Tutorial. The GUI Customization Tutorial describes how to perform common GUI customization tasks such as extending the main menu, extending the popup menu, and adding your own custom widgets.
Symutils	The Symbol Library Utilities (Symutils) documentation.
Platforms	This is a list of supported Operating Systems and Hardware Platforms.
Glossary	This glossary defines several terms (words and phrases) that are used within this documentation and on the GUI (graphical user interface) of RTLvision PRO.
Third-Party Licenses	Licensing information about third-party software.

Key Features

- Read RTL-level System-Verilog, Verilog and VHDL files
- Read gate-level Verilog and EDIF files

- Automatic cone extraction
- Generate easy-to-read schematics from circuit netlists
- Incremental schematic navigation for big designs
- Save/restore/exchange schematic fragments (Snapshot feature of the "Cone" window)
- Binfile support to "compile" big designs
- Drag & Drop between different views
- APIs: Full database API - and GUI API - both to extend RTLvision PRO

Overview of the File Names

The main graphical user interface and main tool is **RTLvision PRO**. It can be invoked using the executable name `rtlvisionpro` (`rtlvisionpro.exe` on Windows) and requires at least the following FlexNet features:

- `rtlvision2pro`
- `gv-veri2zdb`
- `gv-gate2zdb`
- `gv-wave`

Batch Tools and Tcl Commands

The RTLvision PRO download package also contains batch tools for command line compilation of design data. The FlexNet features required to run these tools are listed in the table below.

Another option to read design files are the Tcl commands either available on the Console window of RTLvision PRO or from within a Userware script. Using the Tcl command in the Console window of RTLvision PRO does not require a separate FlexNet feature. The parser is already licensed by the RTLvision PRO master feature.

The RTLvision PRO package also contains an interactive shell named `zdbsh` (= the 'ZDB Shell') that provides a method of running scripts in a GUI-less mode.

Batch Tool Name	Required License Feature	Tcl Command	Brief Description
rtl2zdb	`gv-veri2zdb`	zrtl	Read RTL files (Verilog, SystemVerilog, VHDL) and create a ZDB binfile.
vhdl2vdb	`gv-veri2zdb`	zvdb	Read VHDL files and compile them into a binary format that is required by `rtl2zdb` to access VHDL libraries.
verilog2zdb	`gv-gate2zdb`	zverilog	Read structural Verilog files (netlist) and create a ZDB binfile.
edif2zdb	`gv-gate2zdb`	zedif	Read EDIF files and create a ZDB binfile.

Batch Tool Name	Required License Feature	Tcl Command	Brief Description
lef2zdb	gv-gate2zdb	zlef	Read LEF files and create a ZDB binfile.
def2zdb	gv-gate2zdb	zdef	Read DEF files and create a ZDB binfile.
vcdcompile	gv-wave	vcd read	Read a VCD file and compile it into Altair Engineering, Inc.'s compressed format.
liberty2zdb	gv-gate2zdb	zliberty	Read Liberty files and create either a ZDB binlib or a symbol library.
sym2zdb	-	-	Read Altair Engineering, Inc.'s symbol library format (`.sym`) and create a ZDB binlib.
zdb2sym	-	-	Open a ZDB binfile and write out a symbol library.

Supported Platforms

This document lists the operating systems and processor hardware platforms that are supported by Altair Engineering, Inc.'s visualization tool RTLvision PRO.

Operating System	**Processor Hardware**		**Release Directory**	**Download Package**
Linux (glibc > 2.12)	x86_64 (AMD64, EM64T)	ILP64	linux64	rtlvision-2024-linux.tgz
Linux (glibc 2.5-2.12)	x86_64 (AMD64, EM64T)	ILP64	linux-legacy	rtlvision-2024-linux-legacy.tgz
Windows	Intel x64 (AMD64, EM64T)	LLP64 (or P64)	win64	rtlvision-2024-win.zip rtlvision-2024-setup64.exe

Installation Notes

In this Installation Notes Guide you will find download and setup instructions for the RTLvision PRO software.
These notes also describe how you can setup the different license key mechanisms based on the FlexNet technology.

Overview

Installing the Altair Engineering, Inc. Software is very simple. Follow these steps to download, unpack and setup RTLvision PRO.

Download the Software

For downloading the software package you need an account provided to you by one of Altair Engineering, Inc.'s distributors or directly by us.

Please go to https://download.concept.de/rtlvision to download the RTLvision PRO software package.

If your license file is a floating license then you need to download the `flexnet-11.16.3.tar.gz` package that contains the Flexera license server software and the `dconcept` vendor daemon.

Unpack and Install RTLvision PRO

After downloading the package `rtlvision-2024-<PLATFORM>` for your platform, just unpack it in an empty directory of your choice, e.g. in `/usr/local/concept`:

```
mkdir /usr/local/concept
cd /usr/local/concept
gunzip -c rtlvision-2024-<PLATFORM>.tgz | tar xf -
```

This creates the directory `rtlvision-2024`. It contains the Altair Engineering, Inc. RTLvision PRO software.

For the Windows platform a zip package and a `setup.exe` is provided.

No further installation is required except setting up the environment for all users.

Node-Locked vs. Floating Licenses

Node-Locked License

A 'node-locked license' is intended for a single user and is tied to one machine or dongle. The advantage is that you do not need to run a license server.

An example of a node-locked license file is given below:

```
FEATURE <FEATURE_NAME> dconcept 99.9 31-dec-2018 uncounted \
       VENDOR_STRING=0,uRVPSI HOSTID=12345678 NOTICE="Concept \
       Engineering" START=01-Jan-2014 TS_OK SIGN=043831664CDC
```

Floating License

A 'floating license' is intended to be shared in a network environment and allows an user to use a license from any machine that can access the license server. The license server and vendor daemon must be running on the machine whose hostname and Hostid are specified in the license key.

An example of a floating license file is given below:

```
SERVER <HOSTNAME> <HOSTID> 1700
VENDOR dconcept <install_dir>/<platform>/dconcept
USE_SERVER
FEATURE <FEATURE_NAME> dconcept 99.9 31-dec-2018 1 \
       VENDOR_STRING=0,FSYBVI NOTICE="{Company}" \
       START=01-Jan-2014 SIGN=7BB1C0C4AA3A
```

Get the Hostid

We use Flexera's FlexNet software for license protection. To provide you with a license file, we need the "hostid" information of either your computer (node-locked license) or of your license server (floating license).

Using FlexNet

To get the FlexNet hostid, you must run the FlexNet tool `lmutil` with the argument `lmhostid` inside a terminal window (`xterm` on UNIX or "Command Prompt" on Windows). E.g.:

```
$ lmutil lmhostid
lmutil - Copyright (c) 1989-2013 Flexera Software LLC. All Rights Reserved.
The FlexNet host ID of this machine is "0018f322ceb6"
```

In this example the hostid is `0018f322ceb6`.

You can either use the `lmutil` tool that is part of the flexnet package you get on the RTLvision PRO download site or you can download `lmutil` directly from the Flexera download page (registration is requested).

NOTE

On Windows you can use `lmtools.exe`, a graphical frontend for the licensing tools, as an alternative to `lmutil`.

This tool is also part of the flexnet package you get on the RTLvision PRO download site or it can also be found at the Flexera download page.

If you prefer `lmtools.exe`, here is some help:

Start `lmtools.exe` and press the "hostid" button. The 2nd last line is a single hex number - that's the hostid. However, if that field is `0` or `ffffffffffff`, or if you are unsure, please copy & paste all the output into the field "General comments" above. The best way to do copy & paste probably is: Use the "Save Text" button to save the output into a file and then use a text editor (e.g. Notepad) to open that file and then use the text editor's copy & paste.

Alternative Method

You can start the RTLvision PRO main binary (`rtlvisionpro`), if you do not have a license file installed, then an error window pops up and you can find the "hostid" for your machine at the end of the error text.

Please find below alternative methods to obtain the hostid for your system without running any third party software.

On Linux and Windows the MAC address ("Physical Address") of the network adapter is used as the hostid.

Solaris also provides an alternative method to get the hostid.

Linux

Start a terminal and type in the following to extract this information without running the `lmutil` binary:

```
/sbin/ifconfig -a | grep 'Hwaddr\|ether'
eth0 ... HWaddr 00:12:34:56:78:9A
```

Remove the colons (':') to get the hostid. In this example, the hostid is `00123456789A` (the result is always 12 characters).

NOTE

The Linux `hostid` command reports a 6- or 8-character result that is not a valid hostid for the FlexNet license software.

Windows

The MAC address ("Physical Address") of the network adapter is used as the hostid.

Start a command prompt (Click **Start › Run** and type `cmd`) and type the following:

```
C:\> ipconfig -all
Ethernet adapter Local Area Connection:
   ...
   Physical Address. . . . . . . . : 00-50-56-B3-68-A5
```

Remove the hyphens ('-') to get the hostid. In this example, the hostid is `005056B368A5` (the result is always 12 characters).

Solaris

On Solaris, the hostid is similar to the hostname. To determine the hostid of a Solaris machine, the `sysdef` utility can be used to get this information.

Start a terminal and type in the following:

```
$ sysdef -h
*
* Hostid
*
  84c3c53a
```

The `hostid` utility is a shortcut command to quickly get the hostid.

Start a terminal and type in the following:

```
$ /usr/bin/hostid
84c3c53a
```

The result is the hostid and is always eight characters.

Install Licensing Software for a Floating License

The license server is usually managed by your system or license administrator. These Installation Notes provide some information about setting up a FlexNet floating license server for the Altair Engineering, Inc. tools.

These notes only cover UNIX platforms, however you can also setup a FlexNet floating license server running as a service on a Windows machine using the graphical frontend `lmtools.exe`.

License Server on UNIX

Download the FlexNet software package from the same location as the RTLvision PRO software and unpack it on your license server machine.

If your license server machine already has the FlexNet server software (`lmgrd` and other FlexNet tools) installed and running, then you only need to add the "Altair Engineering, Inc." vendor daemon (`dconcept`). Go to the directory where you store FlexNet vendor daemons and add `dconcept`.

The following command extracts only the vendor daemon:

```
gunzip -c flexnet-11.16.3.tgz | \
     tar xf - "flexnet-11.16.3/*/dconcept"
```

If your license server machine does not have any FlexNet software installed, then you need to install the FlexNet software and the "Altair Engineering, Inc." vendor daemon. The following command extracts that software:

```
gunzip -c flexnet-11.16.3.tgz | \
    tar xf - flexnet-11.16.3
```

Store your license file `license.dat` on your license server machine, or merge its contents into your existing `license.dat`. Modify the `VENDOR` line for `dconcept` and enter the correct path to the Altair Engineering, Inc. vendor daemon executable (called `dconcept`).

NOTE	The vendor daemon TCP/IP port number, if not specified, is chosen by the operating system at run-time. On windows this procedure can result in a longer startup time for the application. So we recommend to specify a TCP/IP port number for `dconcept`, e.g. `port=1701`.

If `lmgrd` is already running, then you only need to read the license file again:

```
lmreread -c license.dat
```

If `lmgrd` is not running, then you should start it by:

```
lmgrd -c license.dat > lmgrd.log &
```

In both cases please check the log file `lmgrd.log` and verify that the license works:

```
lmdiag -c license.dat <FEATURE>
```

Replace `<FEATURE>` with e.g. `rtlvision2pro`.

NOTE	Check the IP port number in the `SERVER` line and make sure that all users need to specify that number in the `LM_LICENSE_FILE` environment variable.

Usually, you may want `lmgrd` to automatically start at boot time. For this, you must add a start script (or just the one line above) into the boot rc script of you system. Different UNIX systems are doing that differently - please ask your system administrator.

Setting up the User Environment

To start the RTLvision PRO GUI and batch tools you need to set or extend the `PATH` variable and point to the platform dependent directory in the installation path.

```
export PATH
PATH=/usr/local/concept/rtlvision-2024/linux64:$PATH
```

The RTLvision PRO software needs to checkout a license at startup time. You can either copy the `license.dat` file to the installation directory as `/usr/local/concept/rtlvision-2024/license.dat` or set/extend the `LM_LICENSE_FILE` environment variable and point to the license file:

```
export LM_LICENSE_FILE
LM_LICENSE_FILE=/path/to/license.dat
```

NOTE	Alternatively you can use the vendor daemon specific variable `DCONCEPT_LICENSE_FILE` to avoid conflicts with any other software also using `LM_LICENSE_FILE`.

You may want to setup the environment permanently. If you run the Bourne Shell (sh), bash, ksh, zsh or compatible, then the commands above are used to setup the environment to your `$HOME/.profile` or `$HOME/.login` file. Please refer to your operating system's documentation for detailed instructions on how to permanently set environment variables.

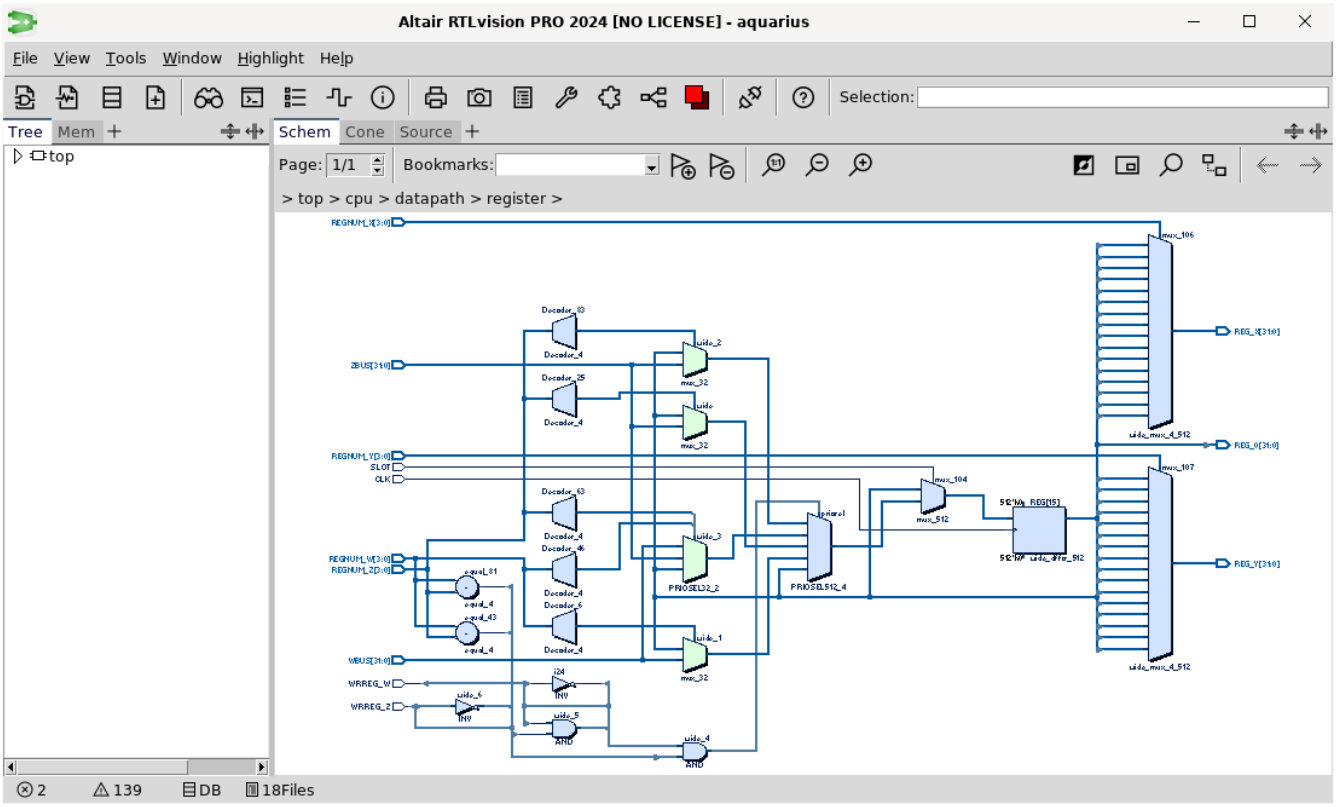
On Windows you can set these environment variables either for the system or for the current user in the "System Properties" dialog accessible from the Control Panel (**System › Advanced › Environment Variables**).

The RTLvision PRO GUI

This manual describes the features and usage of the RTLvision PRO GUI.

Overview

RTLvision PRO provides fast visualization of RTL, so that an engineer can easily understand, and implement existing code elements, whether in VHDL, Verilog or SystemVerilog.



Tooltips

If you place the mouse over an object, then after a small delay a tooltip label pops up. This label gives you additional context sensitive information or hints about the object that is displayed under the mouse cursor. In the Preferences dialog you can toggle the display of tooltips and the display of attributes in the tooltips.

Drag & Drop

The tool supports Drag & Drop with either the **left** or **right** mouse button. The left mouse button is the default for Drag & Drop, but this can be configured in the Preferences dialog or using the command line option `-dndButton`.

You can start dragging one or more objects depending on how many objects are currently selected or where you start dragging from. While dragging is active, the mouse cursor changes. The "forbidden" cursor indicates an invalid drop zone; you may not drop objects here. The "dropped" objects will temporarily be highlighted with the Goto color (similar to the Goto function).

Dropping Objects to Inactive Tabs

If you want to drop objects on one of the Tab group's tab that is currently not visible (not active) you can do the following: Drag the object(s) on one of Tab group's tab name (the cursor changes) and hold this position for about half a second. The tab will be activated and you may drop the object(s) into the window, if possible. You can also drop directly to the tab name.

Highlight

The tool supports different global highlight lists, each represented by a different color. Any object of the design loaded in RTLvision PRO can be highlighted. Highlighting is global, i.e. all Pane windows and their tabs like Schem, Cone, Source, Mem and Search realize a change in the highlight list.

To **highlight** the selected database object(s), use the context menu entry "Highlight" or press `Ctrl` + `H`. The selected object(s) will be highlighted in the current highlight color.

To unhighlight the selected database object(s), use the context menu entry "Unhighlight" or press `Ctrl` + `Shift` + `H`.

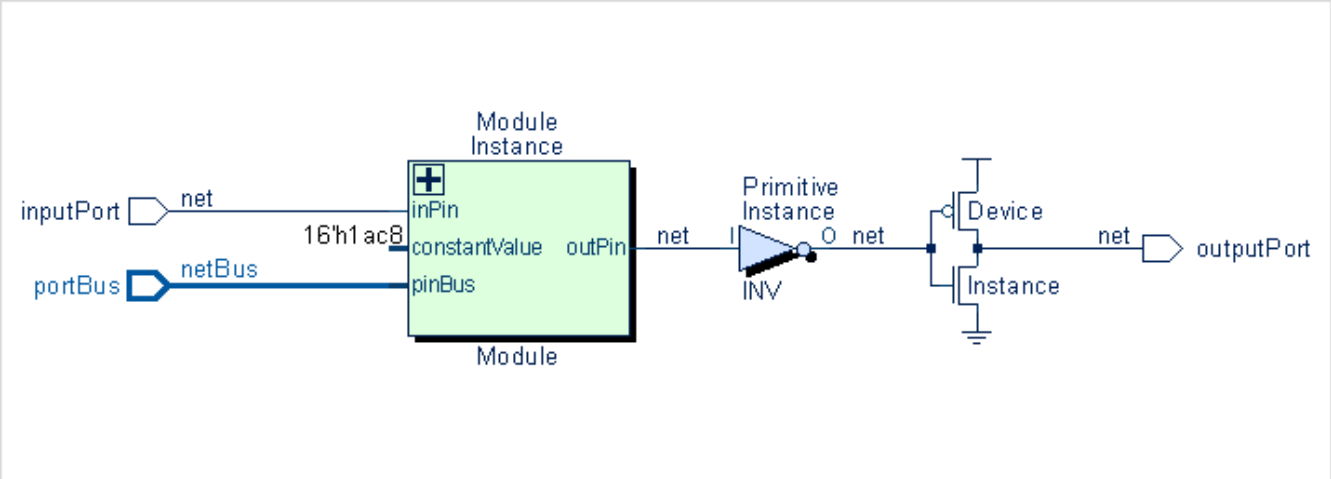
The main menu's Highlight entries are:

- The **Highlight › Current › 0** to **Highlight › Current › 15** sub-menu entries let the user choose the current highlighting color from a selection 16 highlight colors.
- Enabling **Highlight › Auto Increment** causes RTLvision PRO to increment/change the current highlighting color after each highlight command.
- The **Highlight › Unhighlight › 0** to **Highlight › Unhighlight › 15** sub-menu entries remove the highlight information from all database objects of a certain color.
- **Highlight › Unhighlight All** removes the highlight information from all database objects.

Goto

If you invoke the "**Goto**" function using the context menu on the selected object in a Pane window, then the Pane window's sub windows Schem, Cone, Source, Mem and Search will also show the selected object. To indicate the object it is colored with the "goto color". This coloring will disappear after the next mouse click (the Drag & Drop function works similar and also uses the "goto color").

Object Identification



Individual "Objects" can be selected by mouse or highlighted or addressed through the Database API (for more information, please check out the Object Identification (OID) API and this UML diagram). All the GUI's communication features like Drag & Drop, the Goto function, all kind of selection and highlighting, and especially the Memory and Search windows internally base on these Object IDs.

The image above shows an Instance of a Module, one Primitive Instance, two Device Instances, an input and an output Port, an input PortBus, Pins and a PinBus and some Nets and a NetBus.

The basic Object types are:

- Instance - the instance of a Module or a Primitive.
- Pin - an instance pin (refers to a Module/Primitive Port).
- PinBus - an instance bus-pin (refers to a Module/Primitive PortBus).
- Port - a Module/Primitive interface port.
- PortBus - a Module/Primitive interface bus-port.
- Net - a single-bit net connecting pins (and eventually a port).
- NetBus - an array of nets.
- Primitive - defines the Cell's primitive function and the interface (not selectable in Schem or Cone windows).
- Module - defines the interface and contents of a hierarchical block (not selectable in Schem or Cone windows).

Save Settings

There are two types of settings that can be saved:

1. GUI related user preferences can be saved as a workspace.
2. Design related settings can be saved as a project.

Workspace

The Workspace includes all user preferences (including color schemes).

Workspaces are managed via the **File › Workspace** menu.

You can start RTLvision PRO with a certain workspace pre-loaded with the command line option `-workspace`. Workspaces are plain text files with the default extension `.ws`.

Project

A project file includes all design related settings used to load the input files.

Projects are managed via the **File › Project** menu or in the Open Input Files dialog.

You can start RTLvision PRO with a certain project pre-loaded with the command line option `-project`. Projects are plain text files with the default extension `.vpj`.

Display Documentation

The Help function starts the "native" browser of the system to display the tool documentation. In the Preferences dialog you can change the command to invoke a browser.

The Main Menu

The tool's main menu looks like the following image:

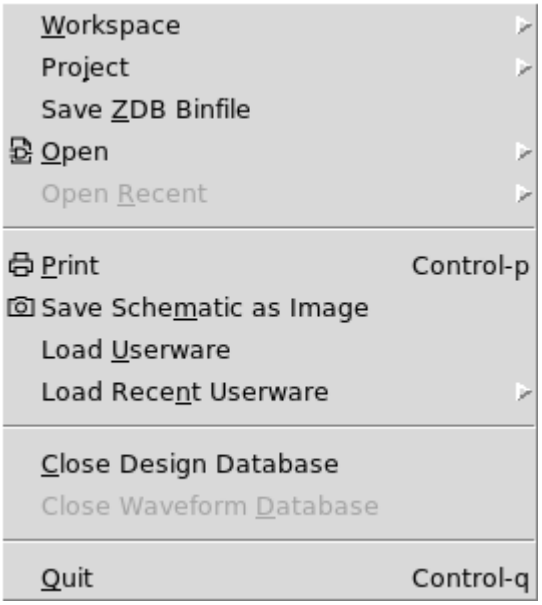
You can activate the menu by mouse or by keyboard (if you press `Alt` and the underlined letter, e.g. `Alt` + `V` followed by `P` will show the Preferences dialog window from the View menu).

The GUI API allows the user to extend the main menu.

The following table shows the entries from the main menu:

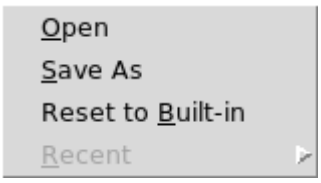
Menu Item	Description
File	Open the File menu.
View	Open the View menu.
Tools	Open the Tools menu.
Window	Open the Window menu.
Highlight	Open the Highlight menu.
Help	Open the Help menu.

The File Menu



Menu Item	Description	Keyboard
Workspace	Start sub-menu to read/write workspace settings from/to file.	
Project	Start sub-menu to read/write project settings from/to file.	
Save ZDB Binfile	Save the current design as a precompiled binary database file (ZDB Binfile).	
Open	Open the Open sub-menu.	
Open Recent	Open recent files.	
Print	Open the Print dialog window.	`Ctrl` + `P`
Save Schematic as Image	Open the Save Schematic as Image dialog window.	
Load Userware	Load and execute a Tcl/Tk Userware script.	
Load Recent Userware	Reload a previously opened Tcl/Tk Userware script.	
Close Design Database	Close the opened design database.	
Close Waveform Database	Close the opened waveform database.	
Quit	Close RTLvision PRO. Before closing the application, a window will pop up to ask for confirmation. This can be omitted with a settings variable. [`showExitPrompt`]	`Ctrl` + `Q`

The File/Workspace Menu

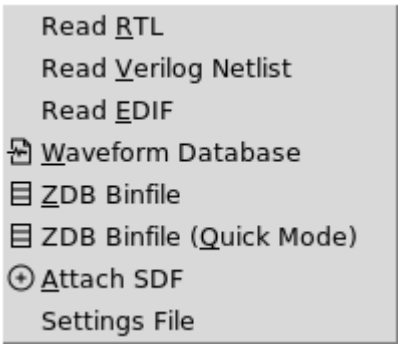


Menu Item	Description	Keyboard
Open	Load a workspace by choosing the file in a dialog.	
Save As	Save current settings to a workspace file.	
Reset to Built-in	Reset all global settings to the built-in defaults.	
Recent	List of recent workspace files.	

The File/Project Menu

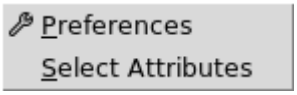
Menu Item	Description	Keyboard
Open	Load a Project by choosing the file in a dialog.	
Save As	Save current design related settings to a Project file.	
New	Create a new Project.	

The File/Open Menu



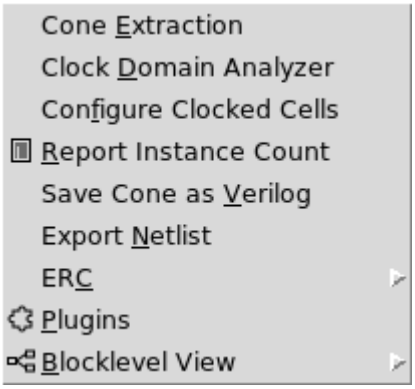
Menu Item	Description	Keyboard
Read RTL	Open the Read RTL dialog window.	
Read Verilog Netlist	Open the Read Verilog Netlist dialog window.	
Read EDIF	Open the Read EDIF dialog window.	
Waveform Database	Open a Waveform database.	
ZDB Binfile	Open a precompiled binary database file (ZDB Binfile).	
ZDB Binfile (Quick Mode)	Open a precompiled binary database file (ZDB Binfile) in Quick Mode.	
Attach SDF	Open the Attach SDF dialog window.	
Settings File	Open a parser settings file.	

The View Menu



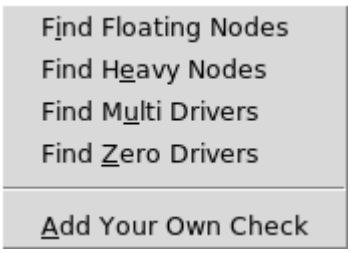
Menu Item	Description	Keyboard
Preferences	Open the Preferences dialog window.	
Select Attributes	Open the Select Display Attributes dialog window.	

The Tools Menu



Menu Item	Description	Keyboard
Cone Extraction	Show the Cone Extraction dialog.	
Clock Domain Analyzer	Run automatic Clock Domain Extraction and Cross Domain Check.	
Configure Clocked Cells	Manually configure Clocked Cells.	
Report Instance Count	Count and display the number of module/primitive instances.	
Save Cone as Verilog	Save the contents of the Cone window As Verilog.	
Export Netlist	Export the loaded design either as a Spice or Verilog netlist.	
ERC	Open the ERC sub-menu.	
Plugins	Show the Plugins dialog window.	
Blocklevel View	Open the Blocklevel View sub-menu.	

The Tools/ERC Menu

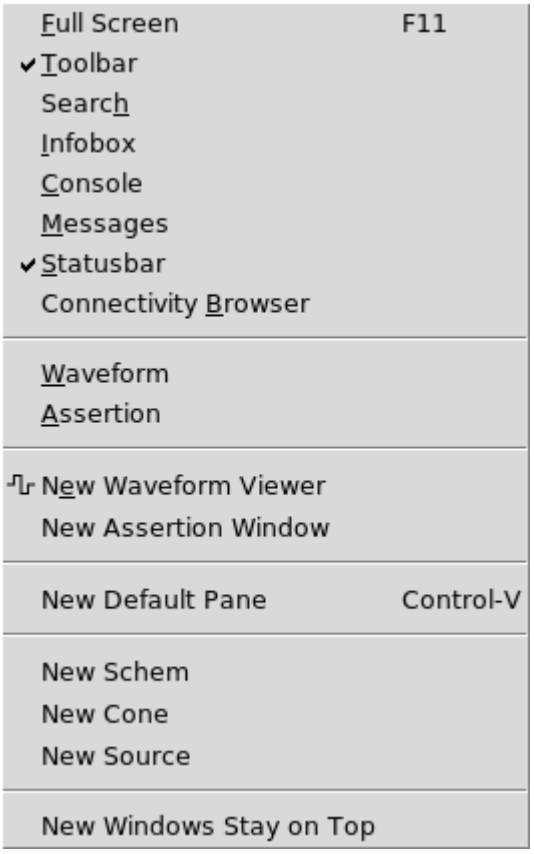


Menu Item	Description	Keyboard
Find Floating Nodes	Search for floating nodes.	
Find Heavy Nodes	Search for heavily connected nodes.	
Find Multi Drivers	Search for signals with multiple drivers.	
Find Zero Drivers	Search for signals without any driver.	

The Tools/Blocklevel View Menu

Menu Item	Description	Keyboard
Enable	Toggle the Blocklevel View mode.	
Neighborhood View	Show the selected instance’s neighbors in a neighborhood view.	
Show Legend	Display a legend explaining neighborhood colors.	
Reset Highlighting	Reset the neighborhood highlighting.	
Toggle Clouds	Merge logic gates into 'logic clouds' and back.	

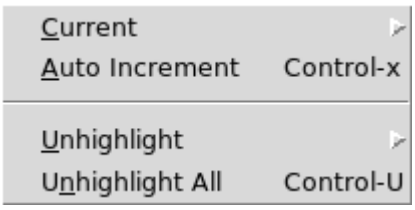
The Window Menu



Menu Item	Description	Keyboard
Full Screen	Toggle fullscreen mode of the main window.	`F11`
Toolbar	Toggle the display of the Toolbar.	
Search	Toggle the display of the Search window.	
Infobox	Toggle the display of the Infobox window.	
Console	Toggle the display of the Console window.	
Messages	Toggle the display of the Messages window.	
Statusbar	Toggle the display of the Statusbar.	
Connectivity Browser	Toggle the display of the Connectivity Browser.	
Waveform	Toggle the display of the Waveform window.	
Assertion	Toggle the display of the Assertion window.	
New Waveform Viewer	Open a new toplevel Waveform window.	
New Assertion Window	Open a new toplevel Assertion window.	
New Default Pane	Create a new Pane window with a set of default tabs.	`Ctrl` + `Shift` + `V`
New Schem	Create a new toplevel Schematic window.	
New Cone	Create a new toplevel Cone window.	
New Source	Create a new toplevel Source window.	

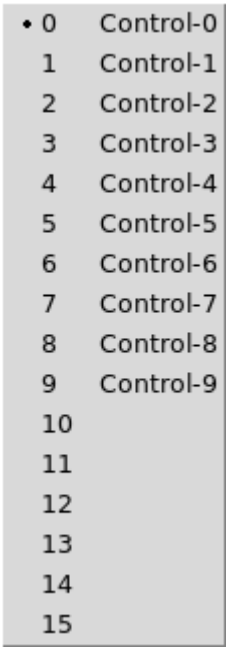
Menu Item	Description	Keyboard
New Windows Stay on Top	If activated, new Pane windows will stay on top of the main GUI. Technically spoken: "on" means "transient to" the main window and "off" means "independent from" the main window.	

The Highlight Menu



Menu Item	Description	Keyboard
Current	Choose the current highlighting color.	
Auto Increment	If activated, the current highlighting color is 'incremented' after each highlighting command.	`Ctrl` + `X`
Unhighlight	Remove the highlight information from all objects of a certain color.	
Unhighlight All	Remove the highlight information from all objects.	`Ctrl` + `Shift` + `U`

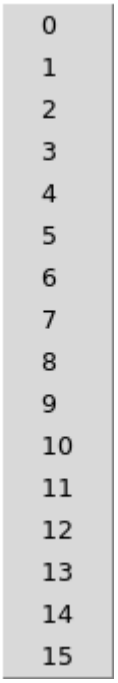
The Highlight/Current Menu



Menu Item	Description	Keyboard
0	Activate the highlight color associated with the number 0.	`Ctrl` + `0`
1	Activate the highlight color associated with the number 1.	`Ctrl` + `1`

Menu Item	Description	Keyboard
2	Activate the highlight color associated with the number 2.	`Ctrl` + `2`
3	Activate the highlight color associated with the number 3.	`Ctrl` + `3`
4	Activate the highlight color associated with the number 4.	`Ctrl` + `4`
5	Activate the highlight color associated with the number 5.	`Ctrl` + `5`
6	Activate the highlight color associated with the number 6.	`Ctrl` + `6`
7	Activate the highlight color associated with the number 7.	`Ctrl` + `7`
8	Activate the highlight color associated with the number 8.	`Ctrl` + `8`
9	Activate the highlight color associated with the number 9.	`Ctrl` + `9`
10	Activate the highlight color associated with the number 10.	
11	Activate the highlight color associated with the number 11.	
12	Activate the highlight color associated with the number 12.	
13	Activate the highlight color associated with the number 13.	
14	Activate the highlight color associated with the number 14.	
15	Activate the highlight color associated with the number 15.	

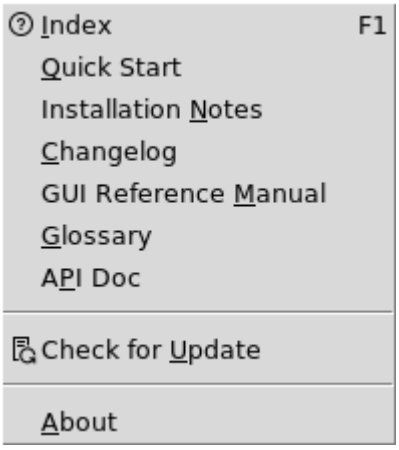
The Highlight/Unhighlight Menu



Menu Item	Description	Keyboard
0	Unhighlight all highlights for the highlight color associated with the number 0.	
1	Unhighlight all highlights for the highlight color associated with the number 1.	

Menu Item	Description	Keyboard
2	Unhighlight all highlights for the highlight color associated with the number 2.	
3	Unhighlight all highlights for the highlight color associated with the number 3.	
4	Unhighlight all highlights for the highlight color associated with the number 4.	
5	Unhighlight all highlights for the highlight color associated with the number 5.	
6	Unhighlight all highlights for the highlight color associated with the number 6.	
7	Unhighlight all highlights for the highlight color associated with the number 7.	
8	Unhighlight all highlights for the highlight color associated with the number 8.	
9	Unhighlight all highlights for the highlight color associated with the number 9.	
10	Unhighlight all highlights for the highlight color associated with the number 10.	
11	Unhighlight all highlights for the highlight color associated with the number 11.	
12	Unhighlight all highlights for the highlight color associated with the number 12.	
13	Unhighlight all highlights for the highlight color associated with the number 13.	
14	Unhighlight all highlights for the highlight color associated with the number 14.	
15	Unhighlight all highlights for the highlight color associated with the number 15.	

The Help Menu



Menu Item	Description	Keyboard
Index	Open the start page of the documentation.	F1
Quick Start	Open RTLvision PRO's Quick Start Guide.	
Installation Notes	Show the Installation Notes.	
Changelog	Display the list of changes in RTLvision PRO.	
GUI Reference Manual	Open the GUI Reference Manual.	
Glossary	Open the Glossary, explaining several terms (abbreviations/phrases/ words) used within RTLvision PRO's GUI and its documentation.	
API Doc	Open the API Documentation.	
Check for Update	Check if a new version of RTLvision PRO is available for download.	
About	Open the About dialog: display version numbers and license details.	

The Context Menus

If you click on certain objects with the **right** mouse button, a context menu will pop up.

Most of the menu entries apply to the selected object(s).

Multiple selection with `Ctrl` and left mouse button is possible before pressing the right mouse button.

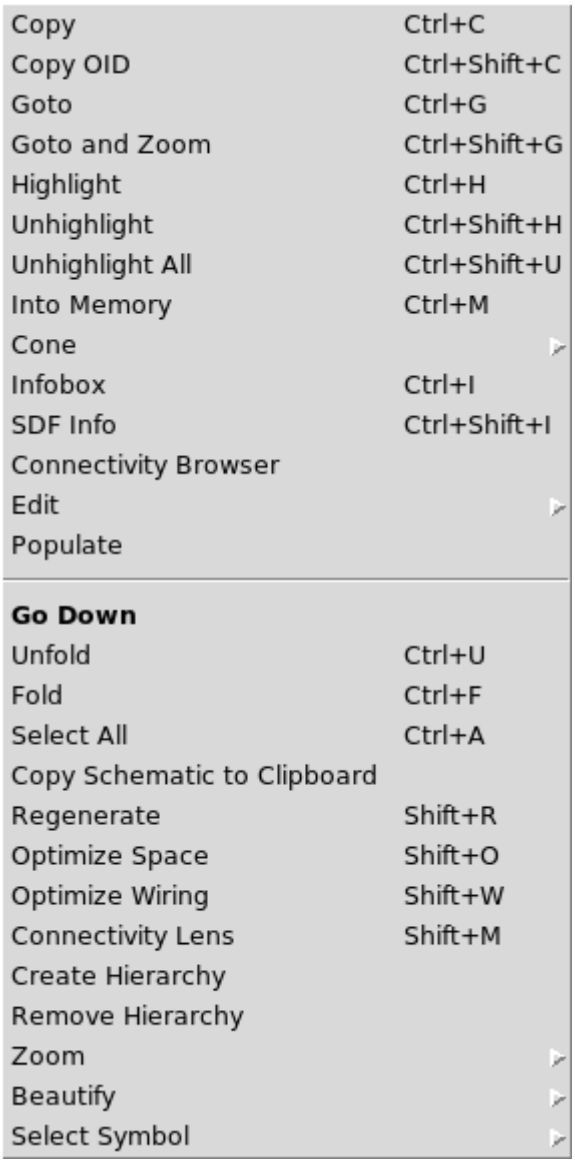
The upper part of the menu (above the separator line) stays the same for all windows whereas the lower part is different in each window (Schem, Cone, Source, Mem, etc.). Depending on the selected object(s) some of the functions are grayed out.

One of the context menu entries can be in **bold** font indicating the default action; he default action is performed if you double-click on the object with the left mouse button.

NOTE	If Drag & Drop is configured to the right mouse button it can also start Drag & Drop for the selected objects: press-move-release means Drag & Drop, while press-release means Popup context menu.

The Schem Popup Menu

The picture shows the context menu of the Schem window.



The following table describes all commands accessible from the Context Menu of the Schem window. The invoked command runs on the selected object(s). Multiple selection with `Ctrl` and left mouse button is possible before pressing the right mouse button.

Popup Item (Schem)	Description	Keyboard
Copy	Copy the name(s) of the selected object(s) to the clipboard.	`Ctrl` + `C`
Copy OID	Copy the Object ID(s) of the selected object(s) to the clipboard.	`Ctrl` + `Shift` + `C`

Popup Item (Schem)	Description	Keyboard
Goto	Goto the selected object(s) in each window. Each window will temporarily highlight the object(s) with the Goto color (the next mouse selection will remove the "Goto color").	`Ctrl` + `G`
Goto and Zoom	Goto and zoom to the selected object(s) in each window.	`Ctrl` + `Shift` + `G`
Highlight	Highlight the selected object(s) with the current highlight color (0 to 15). The current highlight color is selected from the main menu or the toolbar.	`Ctrl` + `H`
Unhighlight	Unhighlight the selected object(s).	`Ctrl` + `Shift` + `H`
Unhighlight All	Unhighlight all highlighted object(s) and remove all highlight information.	`Ctrl` + `Shift` + `U`
Into Memory	Add the selected object(s) into the Mem window.	`Ctrl` + `M`
Cone	The "Cone" sub-menu provides functions to add objects to the Cone window.	
Cone › Load	Load the selected object(s) to the Cone window.	`Ctrl` + `L`
Cone › Load Net	Load the selected net and its driver instance to the Cone window.	`Ctrl` + `N`
Cone › Append	Append the selected object(s) to the Cone window.	`Ctrl` + `Shift` + `L`
Cone › Append Net	Append the selected net and its driver instance to the Cone window.	`Ctrl` + `Shift` + `N`
Cone › Load Module	Load a flat view of the selected hierarchical instance into the Cone window (similar to the Complete Hierarchy function).	
Cone › Extract to Driver/Load	This is a shortcut to the often used features of the Cone Extraction dialog to append the paths to clocked cells respectively top-level I/O ports.	
Cone › Cone Extraction Dialog	Show the Cone Extraction dialog window.	
Infobox	Show a dialog with additional information for the current selection. This includes also information about bus members of a bus.	`Ctrl` + `I`
SDF Info	Show timing information for the current selection.	`Ctrl` + `Shift` + `I`

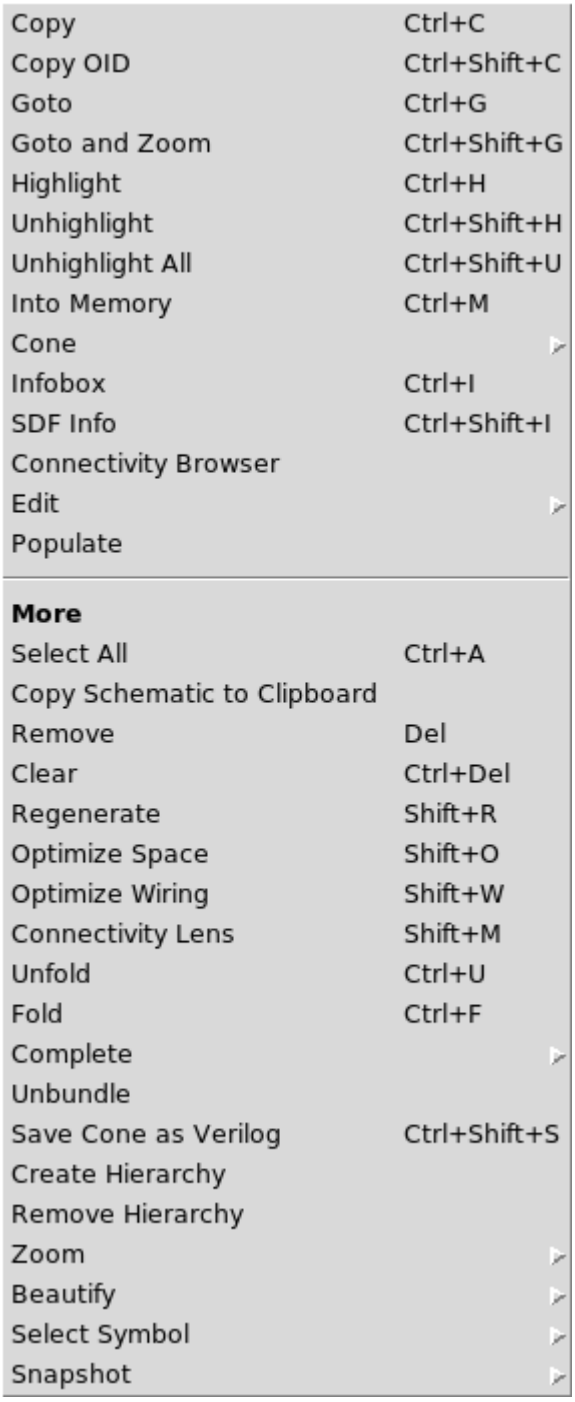
Popup Item (Schem)	Description	Keyboard
Connectivity Browser	The Connectivity Browser helps to examine a Net on the module level or a Signal (all interconnected nets as they pass hierarchy borders). There are functions to browse the connected pins and to display some or all connections in the Cone window.	
Edit	The Edit submenu provides functions to edit or add attributes to an object.	
Edit › Add Comment	Add comments to any object.	`Ctrl` + `O`
Populate	Populate the contents of this module from the binfile.	
Go Down	Dive down and display the module below the selected object (like a double-click in the Schem window).	
Unfold	Unfold the hierarchy box to see the contents.	`Ctrl` + `U`
Fold	Fold the hierarchy box to hide the contents.	`Ctrl` + `F`
Select All	Select all objects.	`Ctrl` + `A`
Copy Schematic to Clipboard	Copy the current schematic view to the clipboard.	
Regenerate	Regenerate the displayed schematic.	`Shift` + `R`
Optimize Space	Optimize the displayed module: keep the existing placement of ports and instances and also the net routing. Optimize the net length (in y direction) and compact the y-space between components.	`Shift` + `O`
Optimize Wiring	Optimize the wiring of the displayed module: keep the placement of ports and instances but optimizes the net routing.	`Shift` + `W`
Connectivity Lens	Show an internal window with detailed connectivity information for the selected pinBus or portBus object.	`Shift` + `M`
Create Hierarchy	Create an artificial level of hierarchy around the selected instances.	
Remove Hierarchy	Remove the selected hierarchical instance. if signal mode is enabled all modules in the path get singlized first.	
Zoom	Provide schematic zoom operations.	
Zoom › Zoom 1:1	Set zoom level to 1.	`1`

Popup Item (Schem)	Description	Keyboard
Zoom › Zoom in	Zoom in.	`I`
Zoom › Zoom out	Zoom out.	`O`
Zoom › Fullfit	Make the full schematic fit the window (not including the page frame and not any space around the schematic).	`F`
Zoom › Fullpage	Make the full page size fit the window - including the page frame.	`Shift` + `F`
Zoom › To Object	Zoom to the selected object.	`Shift` + `G`
Beautify	The functions in this sub-menu allow you to beautify the schematic: hiding ports, rotate and mirror instances, place several instances in the same column, set the location of a port to top, bottom, left, or right.	
Beautify › Rotate	The Rotate submenu provides functions to rotate instances.	
Beautify › Rotate › R90	Rotate the selected instances by 90 degrees.	
Beautify › Rotate › R180	Rotate the selected instances by 180 degrees.	
Beautify › Rotate › R270	Rotate the selected instances by 270 degrees.	
Beautify › Rotate › X-Mirror	Mirror the selected instances along the X axis.	
Beautify › Rotate › Y-Mirror	Mirror the selected instances along the Y axis.	
Beautify › Location	The Location submenu provides functions to set the location for pins or ports.	
Beautify › Location › Left	Set the location of the selected pin or port to 'Left'.	
Beautify › Location › Right	Set the location of the selected pin or port to 'Right'.	
Beautify › Location › Top	Set the location of the selected pin or port to 'Top'.	
Beautify › Location › Bottom	Set the location of the selected pin or port to 'Bottom'.	
Beautify › Column Group	All selected instances will be displayed in one column.	

Popup Item (Schem)	Description	Keyboard
Select Symbol	The Select Symbol submenu provides functions to set the symbol shape for the cell (with no primitive function) of the selected instance (with exactly one output port).	
Select Symbol › BOX	Set the symbol shape to 'BOX' at the selected instance.	
Select Symbol › BUF	Set the symbol shape to 'BUF' at the selected instance.	
Select Symbol › INV	Set the symbol shape to 'INV' at the selected instance.	
Select Symbol › AND	Set the symbol shape to 'AND' at the selected instance.	
Select Symbol › NAND	Set the symbol shape to 'NAND' at the selected instance.	
Select Symbol › OR	Set the symbol shape to 'OR' at the selected instance.	
Select Symbol › NOR	Set the symbol shape to 'NOR' at the selected instance.	
Select Symbol › XOR	Set the symbol shape to 'XOR' at the selected instance.	
Select Symbol › XNOR	Set the symbol shape to 'XNOR' at the selected instance.	
Select Symbol › MUX	Set the symbol shape to 'MUX' at the selected instance.	
Select Symbol › Dialog...	Open a dialog to select a symbol shape and define a port mapping.	

The Cone Popup Menu

The picture shows the context menu of the Cone window.



The following table describes all commands accessible from the Context Menu of the Cone window. The invoked command runs on the selected object(s). Multiple selection with `Ctrl` and left mouse button is possible before pressing the right mouse button.

Popup Item (Cone)	Description	Keyboard
Copy	Copy the name(s) of the selected object(s) to the clipboard.	`Ctrl` + `C`
Copy OID	Copy the Object ID(s) of the selected object(s) to the clipboard.	`Ctrl` + `Shift` + `C`

Popup Item (Cone)	Description	Keyboard
Goto	Goto the selected object(s) in each window. Each window will temporarily highlight the object(s) with the Goto color (the next mouse selection will remove the "Goto color").	`Ctrl` + `G`
Goto and Zoom	Goto and zoom to the selected object(s) in each window.	`Ctrl` + `Shift` + `G`
Highlight	Highlight the selected object(s) with the current highlight color (0 to 15). The current highlight color is selected from the main menu or the toolbar.	`Ctrl` + `H`
Unhighlight	Unhighlight the selected object(s).	`Ctrl` + `Shift` + `H`
Unhighlight All	Unhighlight all highlighted object(s) and remove all highlight information.	`Ctrl` + `Shift` + `U`
Into Memory	Add the selected object(s) into the Mem window.	`Ctrl` + `M`
Cone	The "Cone" sub-menu provides functions to add objects to the Cone window.	
Cone › Load	Load the selected object(s) to the Cone window.	`Ctrl` + `L`
Cone › Load Net	Load the selected net and its driver instance to the Cone window.	`Ctrl` + `N`
Cone › Append	Append the selected object(s) to the Cone window.	`Ctrl` + `Shift` + `L`
Cone › Append Net	Append the selected net and its driver instance to the Cone window.	`Ctrl` + `Shift` + `N`
Cone › Load Module	Load a flat view of the selected hierarchical instance into the Cone window (similar to the Complete Hierarchy function).	
Cone › Extract to Driver/Load	This is a shortcut to the often used features of the Cone Extraction dialog to append the paths to clocked cells respectively top-level I/O ports.	
Cone › Cone Extraction Dialog	Show the Cone Extraction dialog window.	
Infobox	Show a dialog with additional information for the current selection. This includes also information about bus members of a bus.	`Ctrl` + `I`
SDF Info	Show timing information for the current selection.	`Ctrl` + `Shift` + `I`

Popup Item (Cone)	Description	Keyboard
Connectivity Browser	The Connectivity Browser helps to examine a Net on the module level or a Signal (all interconnected nets as they pass hierarchy borders). There are functions to browse the connected pins and to display some or all connections in the Cone window.	
Edit	The Edit submenu provides functions to edit or add attributes to an object.	
Edit › Add Comment	Add comments to any object.	`Ctrl` + `O`
Populate	Populate the contents of this module from the binfile.	
More	Incrementally extend the schematic excerpt (like a double-click in the Cone window).	
Select All	Select all objects.	`Ctrl` + `A`
Copy Schematic to Clipboard	Copy the current Cone schematic view to the clipboard.	
Remove	Remove the selected object(s) from the window.	`Del`
Clear	Clear the window (remove all objects from the window).	`Ctrl` + `Del`
Regenerate	Regenerate the displayed schematic.	`Shift` + `R`
Optimize Space	Optimize the displayed module: keep the existing placement of ports and instances and also the net routing. Optimize the net length (in y direction) and compact the y-space between components.	`Shift` + `O`
Optimize Wiring	Optimize the wiring of the displayed module: keep the placement of ports and instances but optimize the net routing.	`Shift` + `W`
Connectivity Lens	Show an internal window with detailed connectivity information for the selected pinBus or portBus object.	`Shift` + `M`
Unfold	Unfold the hierarchy box to see the contents.	`Ctrl` + `U`
Fold	Fold the hierarchy box to hide the contents.	`Ctrl` + `F`
Complete	The "Complete" sub-menu provides some functions to add contents to the selected hierarchical instance.	

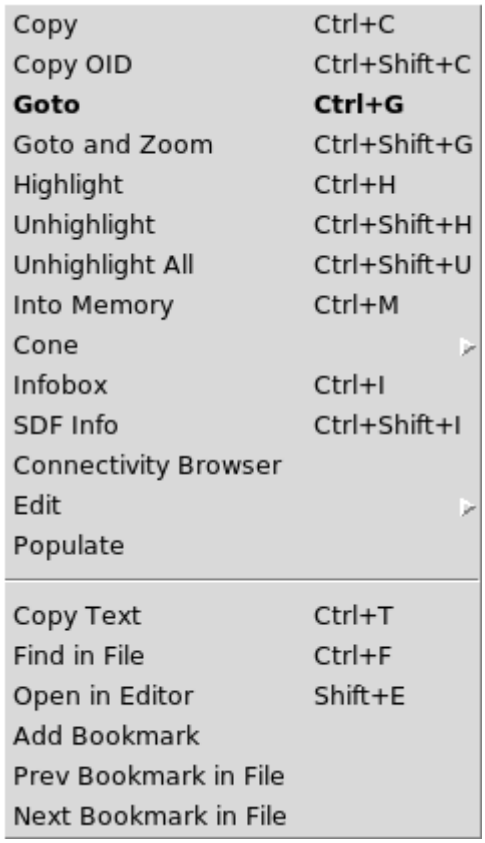
Popup Item (Cone)	Description	Keyboard
Complete › Complete One Level	Add all objects of the down-module	`Ctrl` + `T`
Complete › Complete Hierarchy	Add all objects on all hierarchy levels below.	`Ctrl` + `Shift` + `T`
Complete › Complete Hierarchy and Fold	Add all objects on all hierarchy levels below and fold the selected hierarchical instance.	
Unbundle	The selected netBus is split into individual nets.	
Save Cone as Verilog	The contents of the Cone window is saved as a Verilog netlist.	`Ctrl` + `Shift` + `S`
Create Hierarchy	Create an artificial level of hierarchy around the selected instances.	
Remove Hierarchy	Remove the selected hierarchical instance.	
Zoom	Provide schematic zoom operations.	
Zoom › Zoom 1:1	Set zoom level to 1.	`1`
Zoom › Zoom in	Zoom in.	`I`
Zoom › Zoom out	Zoom out.	`O`
Zoom › Fullfit	Make the full schematic fit the window (not including the page frame and not any space around the schematic).	`F`
Zoom › Fullpage	Make the full page size fit the window - including the page frame.	`Shift` + `F`
Zoom › To Object	Zoom to the selected object.	`Shift` + `G`
Beautify	The functions in this sub-menu allow you to beautify the schematic: hiding ports, rotate and mirror instances, place several instances in the same column, set the location of a port to top, bottom, left, or right.	
Beautify › Rotate	The Rotate submenu provides functions to rotate instances.	
Beautify › Rotate › R90	Rotate the selected instances by 90 degrees.	
Beautify › Rotate › R180	Rotate the selected instances by 180 degrees.	
Beautify › Rotate › R270	Rotate the selected instances by 270 degrees.	

Popup Item (Cone)	Description	Keyboard
Beautify › Rotate › X-Mirror	Mirror the selected instances along the X axis.	
Beautify › Rotate › Y-Mirror	Mirror the selected instances along the Y axis.	
Beautify › Location	The Location submenu provides functions to set the location for pins or ports.	
Beautify › Location › Left	Set the location of the selected pin or port to 'Left'.	
Beautify › Location › Right	Set the location of the selected pin or port to 'Right'.	
Beautify › Location › Top	Set the location of the selected pin or port to 'Top'.	
Beautify › Location › Bottom	Set the location of the selected pin or port to 'Bottom'.	
Beautify › Column Group	All selected instances will be displayed in one column.	
Select Symbol	The Select Symbol submenu provides functions to set the symbol shape for the cell (with no primitive function) of the selected instance (with exactly one output port).	
Select Symbol › BOX	Set the symbol shape to 'BOX' at the selected instance.	
Select Symbol › BUF	Set the symbol shape to 'BUF' at the selected instance.	
Select Symbol › INV	Set the symbol shape to 'INV' at the selected instance.	
Select Symbol › AND	Set the symbol shape to 'AND' at the selected instance.	
Select Symbol › NAND	Set the symbol shape to 'NAND' at the selected instance.	
Select Symbol › OR	Set the symbol shape to 'OR' at the selected instance.	
Select Symbol › NOR	Set the symbol shape to 'NOR' at the selected instance.	
Select Symbol › XOR	Set the symbol shape to 'XOR' at the selected instance.	
Select Symbol › XNOR	Set the symbol shape to 'XNOR' at the selected instance.	

Popup Item (Cone)	Description	Keyboard
Select Symbol › MUX	Set the symbol shape to 'MUX' at the selected instance.	
Select Symbol › Dialog...	Open a dialog to select a symbol shape and define a port mapping.	
Snapshot	The "Snapshot" sub-menu supports Save and Restore the contents of the Cone window.	
Snapshot › Save As	Save the contents of the Cone window as a Snapshot file.	`Ctrl` + `S`
Snapshot › Open	Restore the contents of the Cone window from a Snapshot file.	`Ctrl` + `Shift` + `O`
Snapshot › Open and Validate (remove)	Restore the contents of the Cone window from a Snapshot file and validate it against the loaded database. All invalid objects and connections are removed.	
Snapshot › Open and Validate (grey out)	Restore the contents of the Cone window from a Snapshot file and validate it against the loaded database. All invalid objects and connections are grayed out.	

The Source Popup Menu

The picture shows the context menu of the Source window.



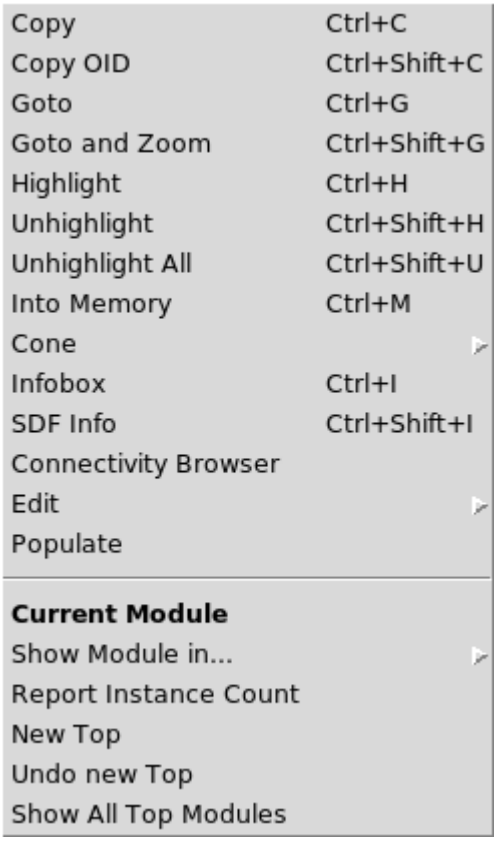
The following table describes all commands accessible from the Context Menu of the Source window. The invoked command runs on the selected object(s). Multiple selection with `Ctrl` and left mouse button is possible before pressing the right mouse button.

Popup Item (Source)	Description	Keyboard
Copy	Copy the name(s) of the selected object(s) to the clipboard.	`Ctrl` + `C`
Copy OID	Copy the Object ID(s) of the selected object(s) to the clipboard.	`Ctrl` + `Shift` + `C`
Goto	Goto the selected object(s) in each window. Each window will temporarily highlight the object(s) with the Goto color (the next mouse selection will remove the "Goto color").	`Ctrl` + `G`
Goto and Zoom	Goto and zoom to the selected object(s) in each window.	`Ctrl` + `Shift` + `G`
Highlight	Highlight the selected object(s) with the current highlight color (0 to 15). The current highlight color is selected from the main menu or the toolbar.	`Ctrl` + `H`
Unhighlight	Unhighlight the selected object(s).	`Ctrl` + `Shift` + `H`
Unhighlight All	Unhighlight all highlighted object(s) and remove all highlight information.	`Ctrl` + `Shift` + `U`
Into Memory	Add the selected object(s) into the Mem window.	`Ctrl` + `M`
Cone	The "Cone" sub-menu provides functions to add objects to the Cone window.	
Cone › Load	Load the selected object(s) to the Cone window.	`Ctrl` + `L`
Cone › Load Net	Load the selected net and its driver instance to the Cone window.	`Ctrl` + `N`
Cone › Append	Append the selected object(s) to the Cone window.	`Ctrl` + `Shift` + `L`
Cone › Append Net	Append the selected net and its driver instance to the Cone window.	`Ctrl` + `Shift` + `N`
Cone › Load Module	Load a flat view of the selected hierarchical instance into the Cone window (similar to the Complete Hierarchy function).	
Cone › Extract to Driver/Load	This is a shortcut to the often used features of the Cone Extraction dialog to append the paths to clocked cells respectively top-level I/O ports.	

Popup Item (Source)	Description	Keyboard
Cone › Cone Extraction Dialog	Show the Cone Extraction dialog window.	
Infobox	Show a dialog with additional information for the current selection. This includes also information about bus members of a bus.	`Ctrl` + `I`
SDF Info	Show timing information for the current selection.	`Ctrl` + `Shift` + `I`
Connectivity Browser	The Connectivity Browser helps to examine a Net on the module level or a Signal (all interconnected nets as they pass hierarchy borders). There are functions to browse the connected pins and to display some or all connections in the Cone window.	
Edit	The Edit submenu provides functions to edit or add attributes to an object.	
Edit › Add Comment	Add comments to any object.	`Ctrl` + `O`
Populate	Populate the contents of this module from the binfile.	
Copy Text	Copy the selected text to the clipboard.	`Ctrl` + `T`
Find in File	Search for a pattern in the currently displayed file.	`Ctrl` + `F`
Open in Editor	Open the file in an external editor.	`Shift` + `E`
Add Bookmark	Add a bookmark in the Source window.	
Prev Bookmark in File	Jump to the previous bookmark in the current source file.	
Next Bookmark in File	Jump to the next bookmark in the current source file.	

The Tree Popup Menu

The picture shows the context menu of the Tree window.



The following table describes all commands accessible from the Context Menu of the Tree window. The invoked command runs on the selected object.

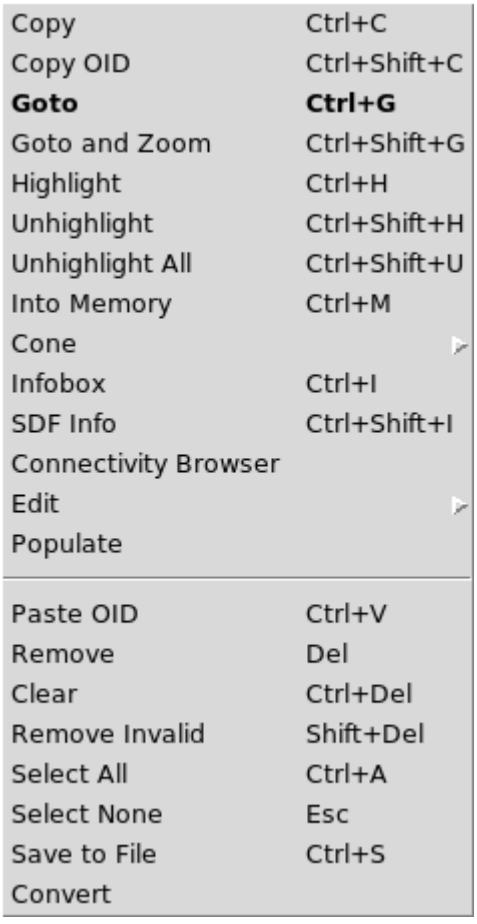
Popup Item (Tree)	Description	Keyboard
Copy	Copy the name(s) of the selected object(s) to the clipboard.	`Ctrl` + `C`
Copy OID	Copy the Object ID(s) of the selected object(s) to the clipboard.	`Ctrl` + `Shift` + `C`
Goto	Goto the selected object(s) in each window. Each window will temporarily highlight the object(s) with the Goto color (the next mouse selection will remove the "Goto color").	`Ctrl` + `G`
Goto and Zoom	Goto and zoom to the selected object(s) in each window.	`Ctrl` + `Shift` + `G`
Highlight	Highlight the selected object(s) with the current highlight color (0 to 15). The current highlight color is selected from the main menu or the toolbar.	`Ctrl` + `H`
Unhighlight	Unhighlight the selected object(s).	`Ctrl` + `Shift` + `H`
Unhighlight All	Unhighlight all highlighted object(s) and remove all highlight information.	`Ctrl` + `Shift` + `U`
Into Memory	Add the selected object(s) into the Mem window.	`Ctrl` + `M`

Popup Item (Tree)	Description	Keyboard
Cone	The "Cone" sub-menu provides functions to add objects to the Cone window.	
Cone › Load	Load the selected object(s) to the Cone window.	`Ctrl` + `L`
Cone › Load Net	Load the selected net and its driver instance to the Cone window.	`Ctrl` + `N`
Cone › Append	Append the selected object(s) to the Cone window.	`Ctrl` + `Shift` + `L`
Cone › Append Net	Append the selected net and its driver instance to the Cone window.	`Ctrl` + `Shift` + `N`
Cone › Load Module	Load a flat view of the selected hierarchical instance into the Cone window (similar to the Complete Hierarchy function).	
Cone › Extract to Driver/Load	This is a shortcut to the often used features of the Cone Extraction dialog to append the paths to clocked cells respectively top-level I/O ports.	
Cone › Cone Extraction Dialog	Show the Cone Extraction dialog window.	
Infobox	Show a dialog with additional information for the current selection. This includes also information about bus members of a bus.	`Ctrl` + `I`
SDF Info	Show timing information for the current selection.	`Ctrl` + `Shift` + `I`
Connectivity Browser	The Connectivity Browser helps to examine a Net on the module level or a Signal (all interconnected nets as they pass hierarchy borders). There are functions to browse the connected pins and to display some or all connections in the Cone window.	
Edit	The Edit submenu provides functions to edit or add attributes to an object.	
Edit › Add Comment	Add comments to any object.	`Ctrl` + `O`
Populate	Populate the contents of this module from the binfile.	
Current Module	The selected module will become the current module and it will be loaded and displayed in the Schem window (like a double-click in the Tree).	

Popup Item (Tree)	Description	Keyboard
Show Module in...	Show the selected module in an existing Schem window, a new Schem window in an existing Tab, or in a new top-level Schem window.	
Report Instance Count	Count the number of module/primitive instances starting at the selected module.	
New Top	Make this module the new top module.	
Undo new Top	Undo the top module.	
Show All Top Modules	Show all available top modules. This option is available if top module is not defined.	

The Mem Popup Menu

The picture shows the context menu of the Mem window.



The following table describes all commands accessible from the Context Menu of the Mem window. The invoked command runs on the selected object(s). Multiple selection with `Ctrl` and left mouse button is possible before pressing the right mouse button.

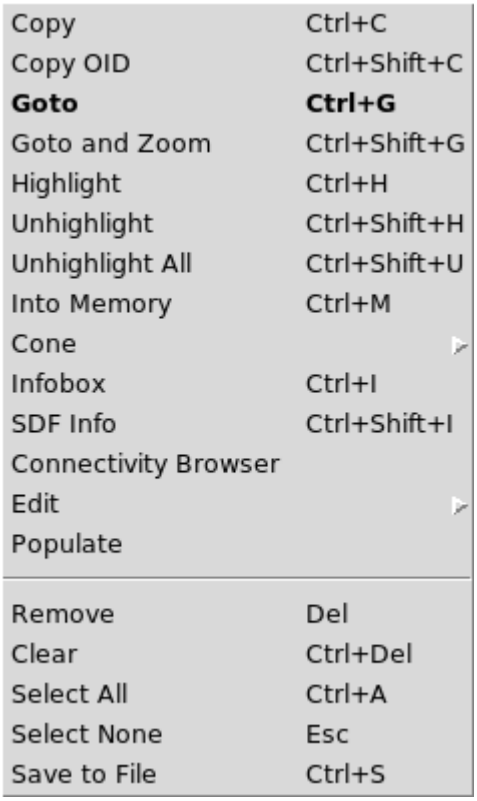
Popup Item (Mem)	Description	Keyboard
Copy	Copy the name(s) of the selected object(s) to the clipboard.	`Ctrl` + `C`

Popup Item (Mem)	Description	Keyboard
Copy OID	Copy the Object ID(s) of the selected object(s) to the clipboard.	`Ctrl` + `Shift` + `C`
Goto	Goto the selected object(s) in each window. Each window will temporarily highlight the object(s) with the Goto color (the next mouse selection will remove the "Goto color").	`Ctrl` + `G`
Goto and Zoom	Goto and zoom to the selected object(s) in each window.	`Ctrl` + `Shift` + `G`
Highlight	Highlight the selected object(s) with the current highlight color (0 to 15). The current highlight color is selected from the main menu or the toolbar.	`Ctrl` + `H`
Unhighlight	Unhighlight the selected object(s).	`Ctrl` + `Shift` + `H`
Unhighlight All	Unhighlight all highlighted object(s) and remove all highlight information.	`Ctrl` + `Shift` + `U`
Into Memory	Add the selected object(s) into the Mem window.	`Ctrl` + `M`
Cone	The "Cone" sub-menu provides functions to add objects to the Cone window.	
Cone › Load	Load the selected object(s) to the Cone window.	`Ctrl` + `L`
Cone › Load Net	Load the selected net and its driver instance to the Cone window.	`Ctrl` + `N`
Cone › Append	Append the selected object(s) to the Cone window.	`Ctrl` + `Shift` + `L`
Cone › Append Net	Append the selected net and its driver instance to the Cone window.	`Ctrl` + `Shift` + `N`
Cone › Load Module	Load a flat view of the selected hierarchical instance into the Cone window (similar to the Complete Hierarchy function).	
Cone › Extract to Driver/Load	This is a shortcut to the often used features of the Cone Extraction dialog to append the paths to clocked cells respectively top-level I/O ports.	
Cone › Cone Extraction Dialog	Show the Cone Extraction dialog window.	
Infobox	Show a dialog with additional information for the current selection. This includes also information about bus members of a bus.	`Ctrl` + `I`

Popup Item (Mem)	Description	Keyboard
SDF Info	Show timing information for the current selection.	`Ctrl` + `Shift` + `I`
Connectivity Browser	The Connectivity Browser helps to examine a Net on the module level or a Signal (all interconnected nets as they pass hierarchy borders). There are functions to browse the connected pins and to display some or all connections in the Cone window.	
Edit	The Edit submenu provides functions to edit or add attributes to an object.	
Edit › Add Comment	Add comments to any object.	`Ctrl` + `O`
Populate	Populate the contents of this module from the binfile.	
Paste OID	Insert objects from the clipboard into the Mem window.	`Ctrl` + `V`
Remove	Remove the selected object(s) from the window.	`Del`
Clear	Clear the window (remove all objects from the window).	`Ctrl` + `Del`
Remove Invalid	Remove objects ids that are now invalid (e.g. after the database has changed).	`Shift` + `Del`
Select All	Select all objects.	`Ctrl` + `A`
Select None	Deselect all objects.	`Esc`
Save to File	Save the contents to an ASCII file.	`Ctrl` + `S`
Convert	Depending on the selected object type this entry converts net to signal or instance to module/primitive and vice versa.	

The Search Popup Menu

The picture shows the context menu of the Search window.



The following table describes all commands accessible from the Context Menu of the Search window. The invoked command runs on the selected object(s). Multiple selection with `Ctrl` and left mouse button is possible before pressing the right mouse button.

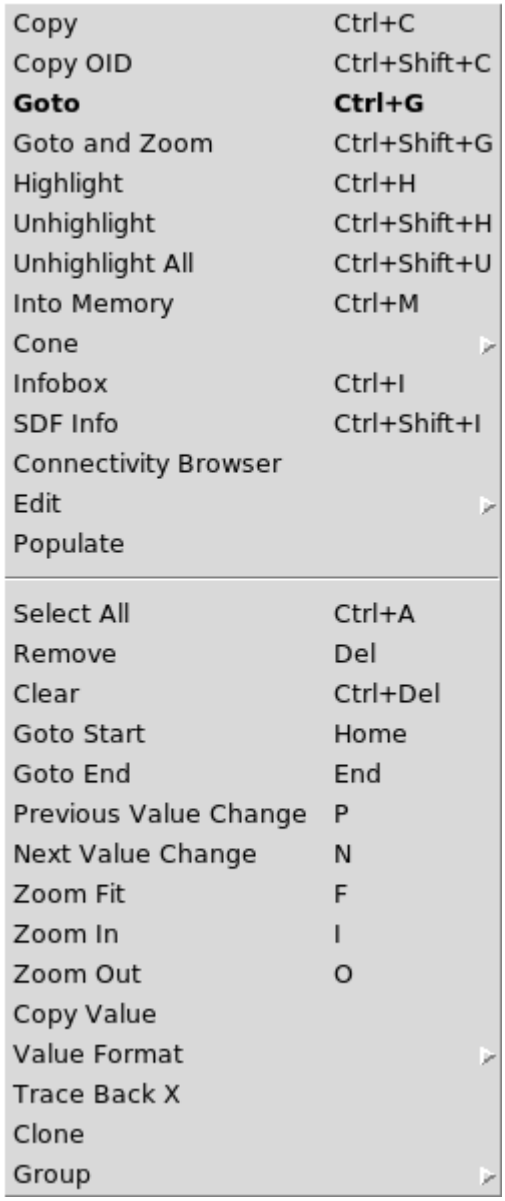
Popup Item (Search)	Description	Keyboard
Copy	Copy the name(s) of the selected object(s) to the clipboard.	`Ctrl` + `C`
Copy OID	Copy the Object ID(s) of the selected object(s) to the clipboard.	`Ctrl` + `Shift` + `C`
Goto	Goto the selected object(s) in each window. Each window will temporarily highlight the object(s) with the Goto color (the next mouse selection will remove the "Goto color").	`Ctrl` + `G`
Goto and Zoom	Goto and zoom to the selected object(s) in each window.	`Ctrl` + `Shift` + `G`
Highlight	Highlight the selected object(s) with the current highlight color (0 to 15). The current highlight color is selected from the main menu or the toolbar.	`Ctrl` + `H`
Unhighlight	Unhighlight the selected object(s).	`Ctrl` + `Shift` + `H`
Unhighlight All	Unhighlight all highlighted object(s) and remove all highlight information.	`Ctrl` + `Shift` + `U`
Into Memory	Add the selected object(s) into the Mem window.	`Ctrl` + `M`

Popup Item (Search)	Description	Keyboard
Cone	The "Cone" sub-menu provides functions to add objects to the Cone window.	
Cone › Load	Load the selected object(s) to the Cone window.	`Ctrl` + `L`
Cone › Load Net	Load the selected net and its driver instance to the Cone window.	`Ctrl` + `N`
Cone › Append	Append the selected object(s) to the Cone window.	`Ctrl` + `Shift` + `L`
Cone › Append Net	Append the selected net and its driver instance to the Cone window.	`Ctrl` + `Shift` + `N`
Cone › Load Module	Load a flat view of the selected hierarchical instance into the Cone window (similar to the Complete Hierarchy function).	
Cone › Extract to Driver/Load	This is a shortcut to the often used features of the Cone Extraction dialog to append the paths to clocked cells respectively top-level I/O ports.	
Cone › Cone Extraction Dialog	Show the Cone Extraction dialog window.	
Infobox	Show a dialog with additional information for the current selection. This includes also information about bus members of a bus.	`Ctrl` + `I`
SDF Info	Show timing information for the current selection.	`Ctrl` + `Shift` + `I`
Connectivity Browser	The Connectivity Browser helps to examine a Net on the module level or a Signal (all interconnected nets as they pass hierarchy borders). There are functions to browse the connected pins and to display some or all connections in the Cone window.	
Edit	The Edit submenu provides functions to edit or add attributes to an object.	
Edit › Add Comment	Add comments to any object.	`Ctrl` + `O`
Populate	Populate the contents of this module from the binfile.	
Remove	Remove the selected object(s) from the window.	`Del`
Clear	Clear the window (remove all objects from the window).	`Ctrl` + `Del`

Popup Item (Search)	Description	Keyboard
Select All	Select all objects.	`Ctrl` + `A`
Select None	Deselect all objects.	`Esc`
Save to File	Save the contents to an ASCII file.	`Ctrl` + `S`

The Wave Popup Menu

The picture shows the context menu of the Wave window.



The following table describes all commands accessible from the Context Menu of the Wave window. The invoked command runs on the selected object(s). Multiple selection with `Ctrl` and left mouse button is possible before pressing the right mouse button.

Popup Item (Wave)	Description	Keyboard
Copy	Copy the name(s) of the selected object(s) to the clipboard.	`Ctrl` + `C`

Popup Item (Wave)	Description	Keyboard
Copy OID	Copy the Object ID(s) of the selected object(s) to the clipboard.	`Ctrl` + `Shift` + `C`
Goto	Goto the selected object(s) in each window. Each window will temporarily highlight the object(s) with the Goto color (the next mouse selection will remove the "Goto color").	`Ctrl` + `G`
Goto and Zoom	Goto and zoom to the selected object(s) in each window.	`Ctrl` + `Shift` + `G`
Highlight	Highlight the selected object(s) with the current highlight color (0 to 15). The current highlight color is selected from the main menu or the toolbar.	`Ctrl` + `H`
Unhighlight	Unhighlight the selected object(s).	`Ctrl` + `Shift` + `H`
Unhighlight All	Unhighlight all highlighted object(s) and remove all highlight information.	`Ctrl` + `Shift` + `U`
Into Memory	Add the selected object(s) into the Mem window.	`Ctrl` + `M`
Cone	The "Cone" sub-menu provides functions to add objects to the Cone window.	
Cone › Load	Load the selected object(s) to the Cone window.	`Ctrl` + `L`
Cone › Load Net	Load the selected net and its driver instance to the Cone window.	`Ctrl` + `N`
Cone › Append	Append the selected object(s) to the Cone window.	`Ctrl` + `Shift` + `L`
Cone › Append Net	Append the selected net and its driver instance to the Cone window.	`Ctrl` + `Shift` + `N`
Cone › Load Module	Load a flat view of the selected hierarchical instance into the Cone window (similar to the Complete Hierarchy function).	
Cone › Extract to Driver/Load	This is a shortcut to the often used features of the Cone Extraction dialog to append the paths to clocked cells respectively top-level I/O ports.	
Cone › Cone Extraction Dialog	Show the Cone Extraction dialog window.	
Infobox	Show a dialog with additional information for the current selection. This includes also information about bus members of a bus.	`Ctrl` + `I`

Popup Item (Wave)	Description	Keyboard
SDF Info	Show timing information for the current selection.	`Ctrl` + `Shift` + `I`
Connectivity Browser	The Connectivity Browser helps to examine a Net on the module level or a Signal (all interconnected nets as they pass hierarchy borders). There are functions to browse the connected pins and to display some or all connections in the Cone window.	
Edit	The Edit submenu provides functions to edit or add attributes to an object.	
Edit › Add Comment	Add comments to any object.	`Ctrl` + `O`
Populate	Populate the contents of this module from the binfile.	
Select All	Select all signals.	`Ctrl` + `A`
Remove	Remove the selected signal(s) from the Waveview window.	`Del`
Clear	Clear the Waveview window (remove all signals).	`Ctrl` + `Del`
Goto Start	Jump to the start time.	`Home`
Goto End	Jump to the end time.	`End`
Previous Value Change	Jump to the previous value change of the selected signal(s).	`P`
Next Value Change	Jump to the next value change of the selected signal(s).	`N`
Zoom Fit	Zoom fit to show all value changes.	`F`
Zoom In	Zoom in.	`I`
Zoom Out	Zoom out.	`O`
Copy Value	Copy the signal value at the time marker to the clipboard.	
Value Format	The "Value Format" sub-menu provides commands to change the displayed format of the bus value only for the selected signal(s).	
Value Format › Hexadecimal	Display bus values in hexadecimal format.	
Value Format › Decimal	Display bus values in decimal format.	
Value Format › Binary	Display bus values in binary format.	

Popup Item (Wave)	Description	Keyboard
Value Format › Octal	Display bus values in octal format.	
Trace Back X	Find the source of the selected X value.	
Clone	Clone selected variables.	
Group	The "Group" sub-menu provides grouping related functions.	
Group › Create Group	Create a new group containing the selected signals (and keep the signals loaded).	
Group › Move to Group	Create a new group containing the selected signals (and remove the signals).	
Group › Rename Group	Change the name of a group.	

Toolbar

The toolbar shows several icons as shortcuts to commonly used commands (most of them are also available in the main menu).

- Show the Read File(s) dialog.
- Open a Waveform Database.
- Open a ZDB Binfile.
- Attach additional information (e.g. SDF).
- Show/Hide the Search window.
- Show/Hide the Console window.
- Show/Hide the Messages window.
- Show/Hide the Waveform Viewer.
- Show/Hide the Infobox window.
- Show the Print dialog.
- Save a schematic as an image.
- Count the number of instantiated modules and primitives.
- Show the Preferences dialog.
- Show the Plugins dialog.
- Toggle the Blocklevel mode.
- Select a highlight color.
- Toggle the Signal Mode. If Signal Mode is enabled then all selected net objects will be converted to signal objects (list of interconnected nets through the hierarchy).

- Open the documentation in your browser.
- Selection: The Last Selection label always displays the last recently selected object(s). It displays the type and the name of the object. If more than one object is selected, then the number of additional selected objects is displayed in parenthesis.

Open Input Files Dialog

This chapter describes the Open Input Files dialog window which is the GUI to all parsers shipped with RTLvision PRO. It also explains how to add design files and configure the parsers. The "Open Files" dialog can be opened from the File > Open menu or by the corresponding icon in the toolbar.

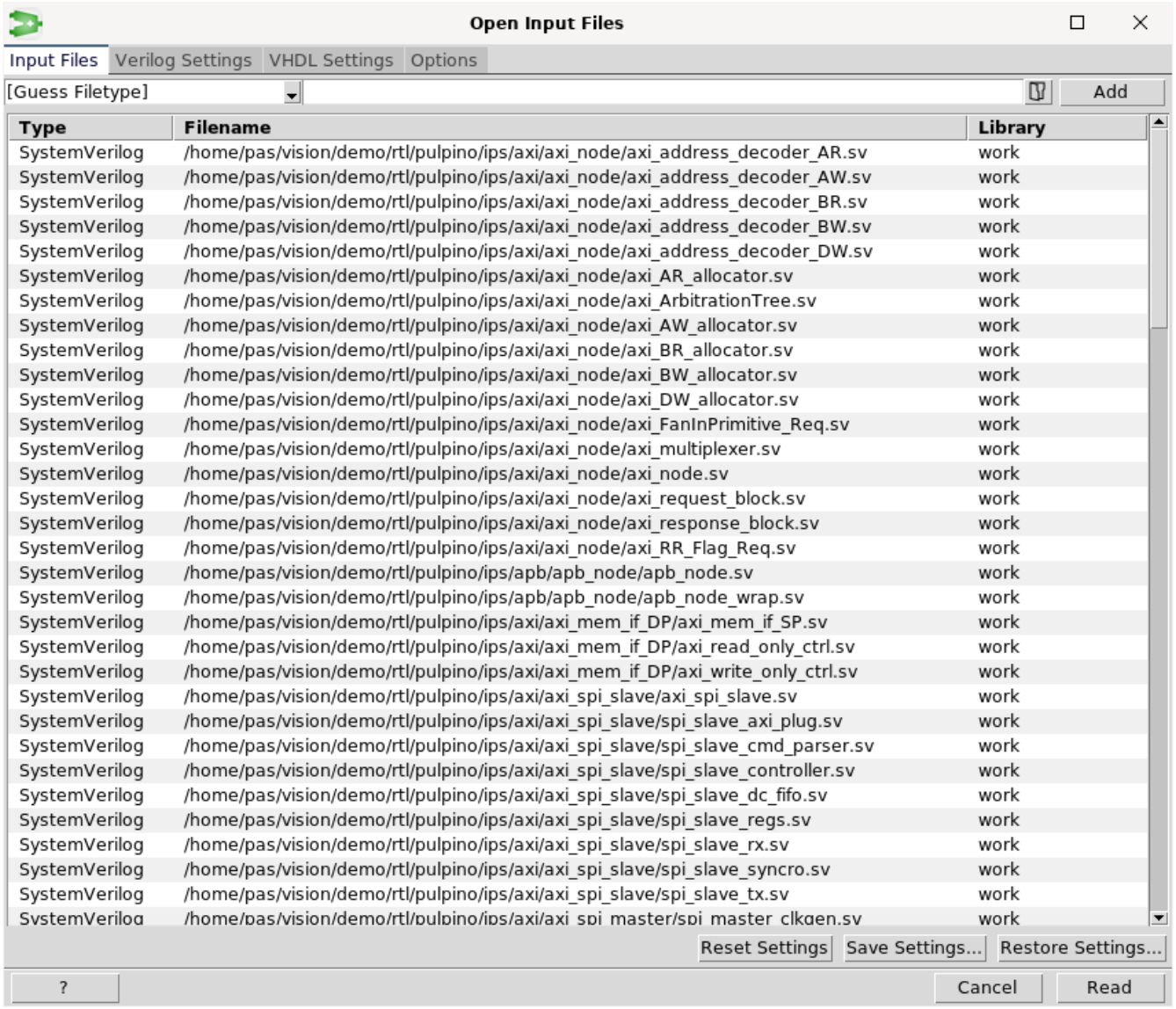
Supported Input File Types

RTLvision PRO is shipped with parsers to read the following file types:

- Behavioral Verilog (RTL)
- SystemVerilog
- VHDL
- Structural Verilog (Netlist)
- EDIF 2.0.0
- LEF/DEF
- Liberty

Add Design Files

The Input Files tab provides an interface to select various design files for compilation. This can be a mix of RTL Files, Netlist Files and Library Files.



To add a new file, press the button and choose an input file. The file type is guessed based on the extension. The **[Add]** button can be used to add a manually entered file to the input file table.

The table on the Input Files tab shows all added design files, their types and the names of their compilation libraries (for netlist files the library name is always "work").

To delete a file, select the row in the input file table and press the `Delete` key on your keyboard. To delete all files at a time, the `Ctrl` + `Delete` key can be used.

To change a file's type or compilation library, you can double click on the corresponding column of the row in the input file table to enter the edit mode.

The **[Save Settings]** and **[Restore Settings]** buttons at the bottom of the dialog can be used to save and restore all settings from the Open Input Files dialog.

The **[Reset Settings]** button sets all values back to their built-in defaults and also clears the input file table.

The **[Read]** button will start the parser on all selected input files.

RTL Files

The RTL parser of RTLvision PRO can read different versions of Verilog and VHDL.

Verilog

For Verilog the versions "Verilog 95", "Verilog 2001" and "SystemVerilog" are supported. The default version, if the input file type is guessed, is "Verilog 2001".

VHDL

For VHDL the versions "VHDL 87", "VHDL 93", "VHDL 2000", "VHDL 2008", and "VHDL 2019" are supported. The default version, if the input file type is guessed, is "VHDL 93".

Netlist Files

RTLvision PRO provides netlist parsers for structural Verilog, EDIF 2.0.0 and DEF files.

Verilog

The Verilog netlist parser of RTLvision PRO is optimized to read huge netlists fast and memory efficient. Therefore only the structural subset of "Verilog 95" is supported. For reading behavioral Verilog code please use the RTL Verilog parser.

All instantiated cells must be defined either as Verilog "modules" or "primitives", however, a declare-before-use restriction does not apply. This means, it's ok for the Verilog Parser to find the cell definitions just anywhere in the Verilog design or Library files. But all library cells must be defined in Verilog. Specifying symbol shapes (e.g. with Symlib) is not sufficient.

If Verilog libraries are missing, then the Verilog parser gets "undefined instances" and tries to guess the footprints of the missing cells. However, for Verilog instantiation by order, the library cell's port names cannot be guessed. They are created with "undefined" direction. In general, missing Verilog library cells result in bad looking schematics - even if symbol shapes are specified (as mentioned above).

EDIF

The EDIF parser of RTLvision PRO can read EDIF 2.0.0 files and extract the netlist information. All EDIF schematic elements will be silently ignored.

DEF

A DEF file is usually read in combination with the corresponding LEF file. A DEF file contains the structure and the layout of a design. The DEF parser of RTLvision PRO extracts only the netlist information to display a schematic of the design data contained in the DEF file.

Library Files

Liberty

A Liberty file can be used to provide library information for e.g. the Verilog netlist parser. The interface port directions are extracted from the Liberty file as well as the function of the output

port(s). Based on the Boolean equation of the function a gate symbol shape is generated.

LEF

A LEF file is usually used in combination with the corresponding DEF file. A LEF file contains the interface definition and the layout of a cell. The LEF parser of RTLvision PRO extracts only the interface information and ignores all layout information.

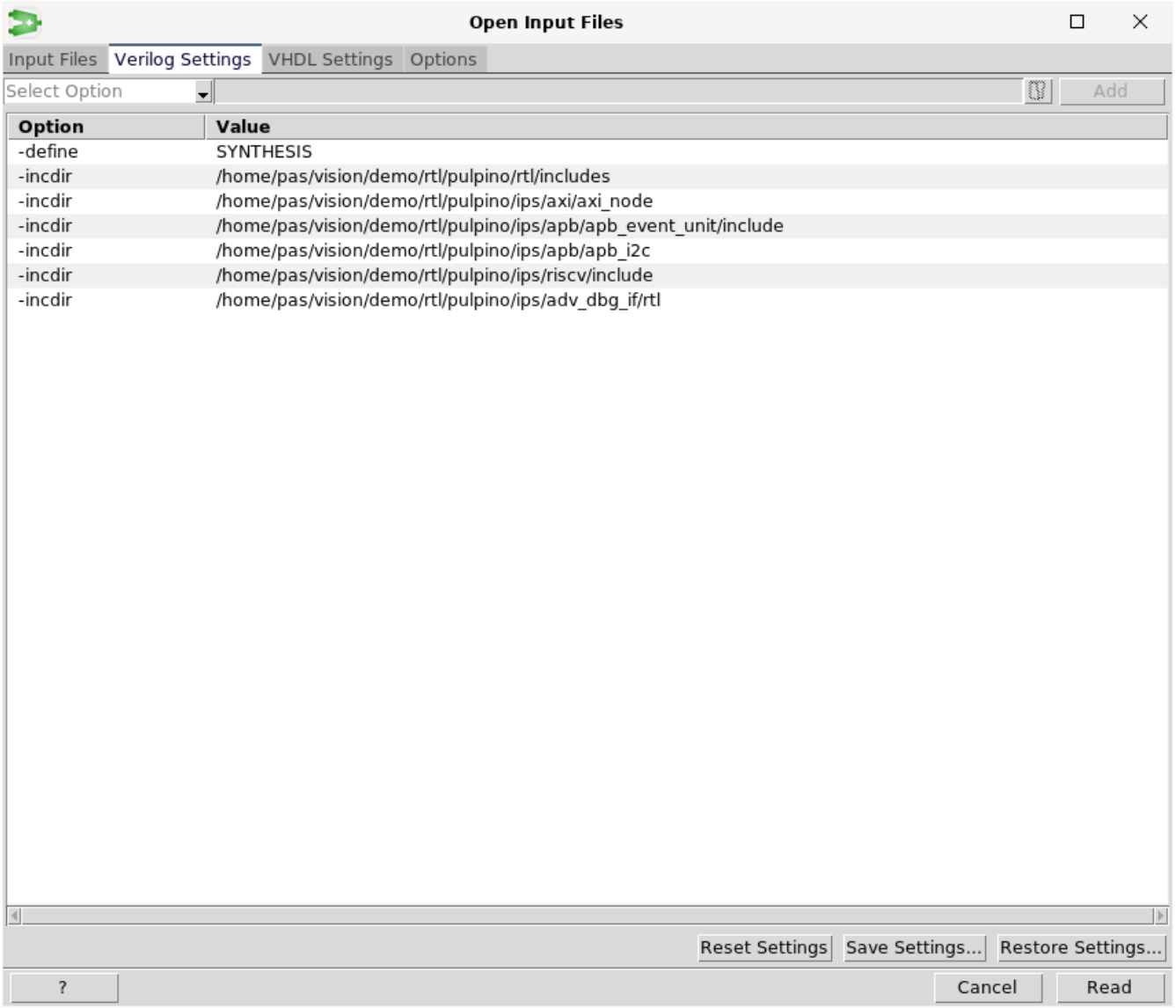
Symlib

A "Symbol Library" file only defines the symbol shapes in the schematic. The interface definition is not extracted and needs to be defined either in the input file or any other library format. Enable the "Preload a symbol library file" option to read the symbol library files before the netlist.

Import Verilog Fileset

For your convenience, a Verilog XL fileset can be imported into RTLvision PRO. The input files, include directories, library settings and macro definitions are distributed into the corresponding fields of this dialog.

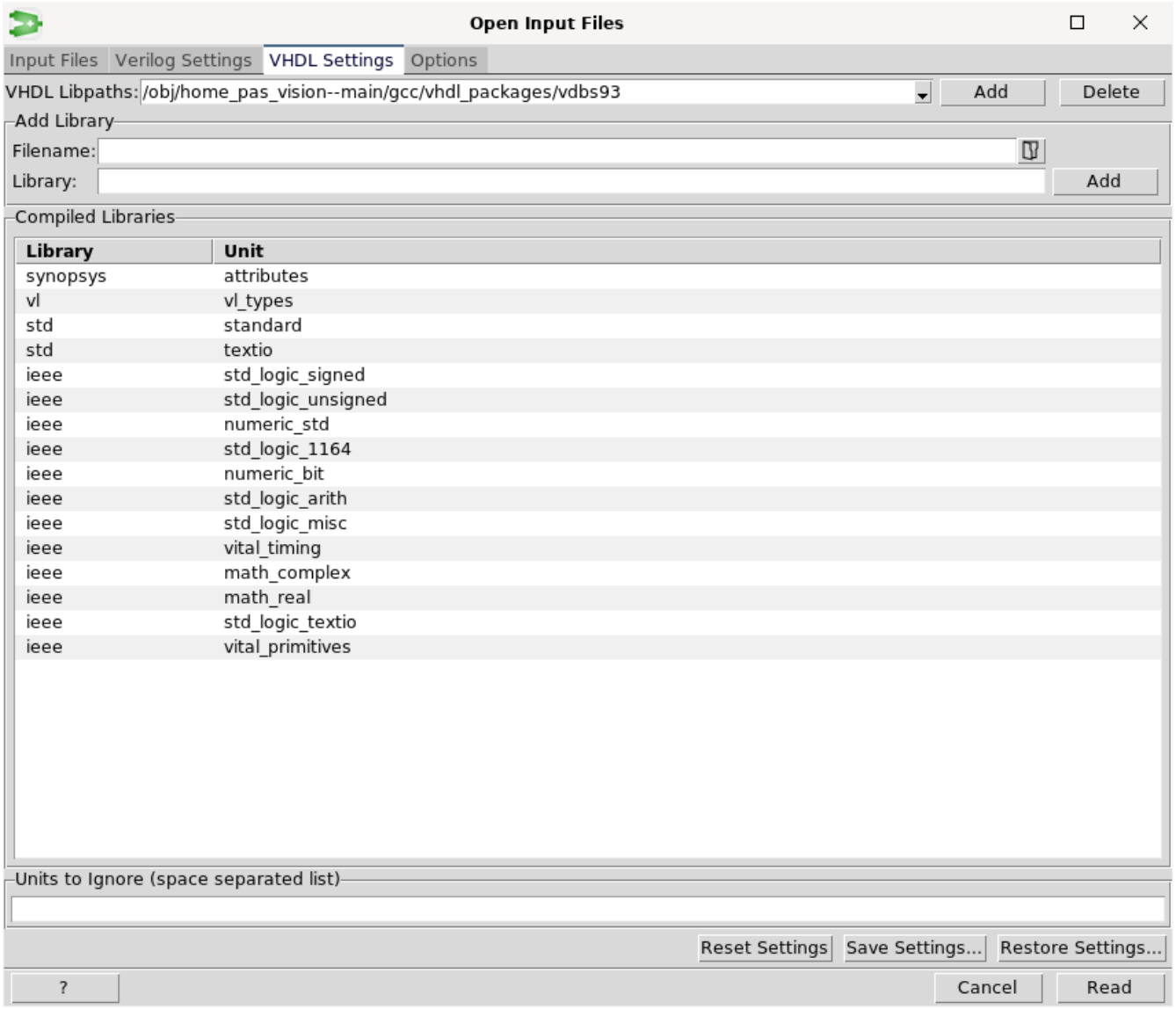
Verilog Specific Settings



The Verilog Settings tab allows you to add include directories, add Verilog library files, specify a library search path and a library file extension. Also Verilog macros can be defined in this tab.

To add an option select the corresponding entry in the Select Option combobox and browse for a file or add a value.

VHDL Specific Settings



The VHDL Settings tab allows you to specify VHDL files to be compiled into binary VHDL libraries (VDB).

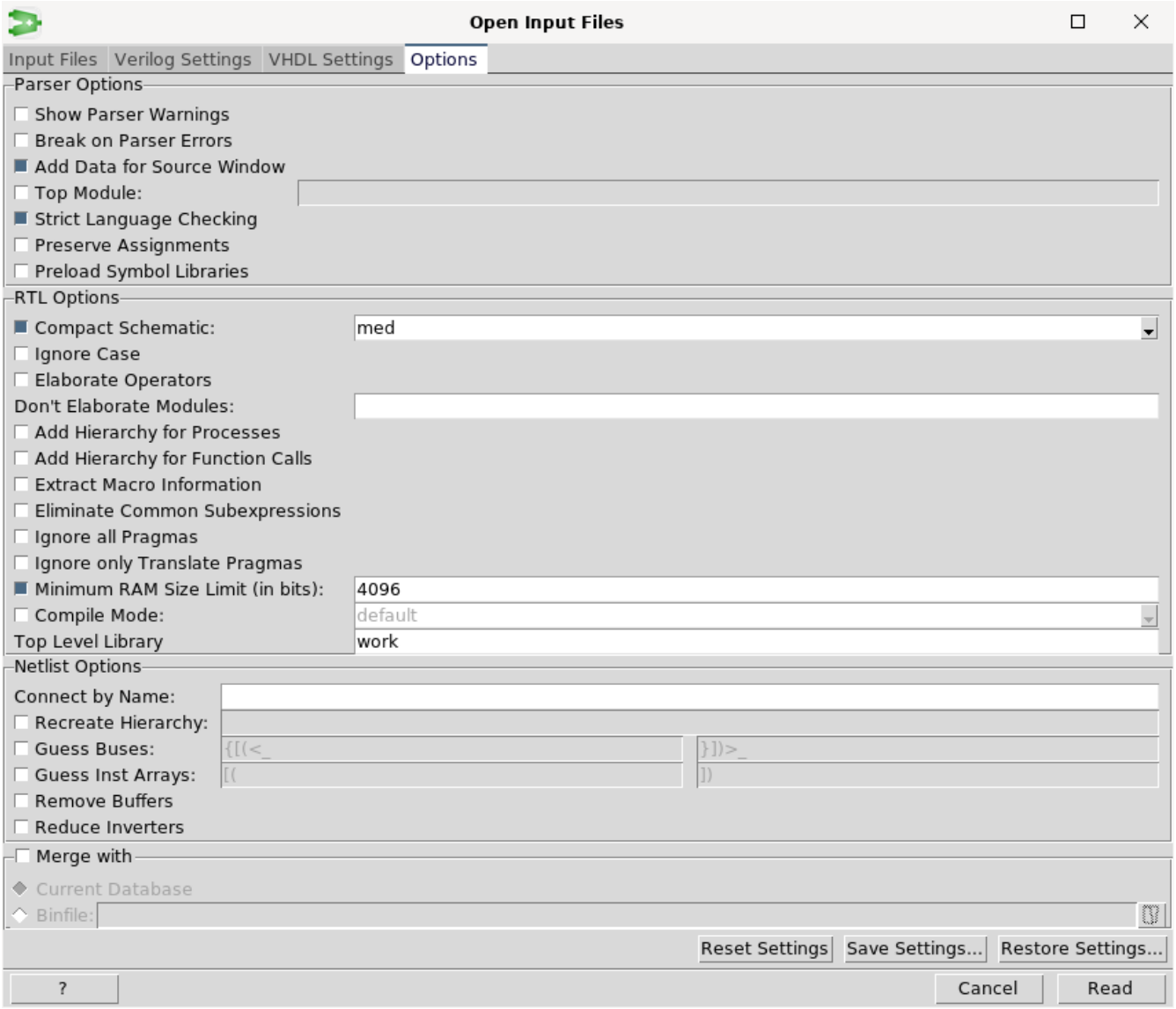
RTLvision PRO comes with a set of precompiled VDBs for standard libraries. You can add your own VHDL Libpaths. Adding a new path will first check if VDB files exist in this path. If this is not the case then a set of initial libraries can be created.

The Add Library section allows you to add your own VHDL files and the corresponding library name.

The Compiled Libraries table will show all VDB libraries and their units found in the selected VHDL Libpath.

Options to Configure the Parsers

The Options tab allows you to define various parser settings. All options can also be specified on the command line.



General Parser Options

- Show Parser Warnings - Displays parser warnings in the Console window.
- Break on Parser Errors - Stop the parser if an error occurs.
- Add Data for Source window - Controls if additional source file cross-reference information is created (switching off saves some memory but disables the Source window).
- Top Module - Enter a new top-level module name. The parser removes all modules not inside the instantiation tree below the given design top module.
- Strict Language Checking - Degrade selected errors into warnings and warnings into info messages.
- Preserve Assignments - Preserve every assignment in the schematic.
- Preload Symbol Libraries - Load symbol library files before the netlist.

RTL Options

- Compact Schematic - Create a more compact schematic view.
- Ignore Case - Ignore character-case during module-name and port-name lookup.
- Elaborate Operators - Control creation of operator implementation.
- Don't Elaborate Modules - Space separated list of module name patterns. Matching modules are **not** elaborated, i.e. no content is created for them - just the interface.
- Add Hierarchy for Processes - Add artificial hierarchy for all procedural statements.
- Add Hierarchy for Function Calls - Add artificial hierarchy for function calls.
- Extract Macro Information - extract macro definitions and references from the source files and display them in the Source window.
- Eliminate Common Subexpressions - Merges logic that creates the same functional behavior.
- Ignore all Pragmas - Ignore all Pragmas defined in the input file(s).
- Ignore only Translate Pragmas - Ignore only Translate and Synthesis Pragmas.
- Minimum RAM Size Limit (in bits) - Limit for multiport RAM generation.
- Compile Mode - Select the multi file (mfcu) or single file (sfcu) compile mode for the input files.
- Top Level Library - The name of the library containing the top-level design.

Netlist Options

- Connect by Name - Space separated list of net name patterns. Matching nets are connected by name, i.e. they are not routed in the schematic view.
- Recreate Hierarchy - Split instance names of a flat design at the given hierarchy separator character and re-create the design hierarchy.
- Guess Buses - Guesses module port and net buses. The two entries contain open/close bit subscript delimiter characters. Close may be empty. For the special case that open contains `0`, no character between base name and bit subscript is needed.
- Guess Inst Arrays - Guesses instance arrays. The two entries contain open/close subscript delimiter characters. The close field can be empty to support only one separator character between the name and the bit subscript. For the special case that open contains `0`, no character between base name and the subscript is needed.
- Remove Buffers - Remove all BUF and WIDE_BUF instances and merge the connected nets.
- Reduce Inverters - Replace odd number of INVs in a chain by one INV and remove an even number of INVs in a chain.

Merge Design Data

The 'Merge with' option allows you to merge the selected design files either with the Current Database or with a previously saved Binfile.

Command Line Options

The command line options of RTLvision PRO are:

Option	Parameters	Description
-argsFromFile	<file>	Read cmdline arguments from file.
-binfile	<file>	Open a zdb binfile.
-binlib	<file>	Open this binfile as a precompiled library.
-breakOnError | -breakOnErr	on|off	Stop on errors during parsing. (The default value for this option is off.)
-builtinWorkspace		Don't read workspace file from home directory.
-check_license		Check the availability of the license.
-compact	off|low|med|full	Adjust the level of compaction for a RTL schematic. (The default value for this option is med.)
-compileMode	default|mfcu|sfcu	Set the RTL compile mode. Mode can either be the default of the specified input language, multi file compilation unit (mfcu) or single file compilation unit (sfcu). (The default value for this option is default.)
-connectByName	<netnamepattern>	Connect matching net by name, i.e. don't route it.
-createBus	on|off	Create buses for ports with Verilog conform, consecutive numbering. (The default value for this option is off.)
-createHier	<sep>	Create hierarchy from flat instance names. Split the instance names at the given hierarchy separator. If the given hierarchy separator character is an empty string then the hierarchy separator character is guessed.
-createMacroObjects	on|off	Store macro definition and reference info in virtual objects. (The default value for this option is off.)
-cse	on|off	Perform common subexpression elimination and merge logic that creates the same functional behavior. (The default value for this option is off.)
-debugFlag	<flag>	Enable a specific debug flag.
-def		Read the input file as a DEF file.
-define	<macro>	Define a Verilog macro on the command line.
+define+	<m>=<v>	Define Verilog macros on the command line.
-defParam	<param>=<value>	Define generic value (VHDL only).

Option	Parameters	Description
-disableHistory		Don't read history file from home directory.
-display	<dsp>	The name of the X server to use.
-dndButton	left|right	Set drag & drop mouse button. (The default value for this option is right.)
-dontElaborate	<pattern>	Don't elaborate modules matching the name pattern.
-edif		Read the input file as an EDIF netlist.
-endLibs		End of library files, design follows.
-F	<fileset>	Read Verilog fileset file. The files in the fileset are relative to the fileset file.
-f	<fileset>	Read Verilog fileset file. Files in the fileset are relative to the current working directory.
-fullscreen		Display main window in fullscreen mode.
-funcHier	on|off	Create hierarchy for function calls. (The default value for this option is off.)
-geometry	<geometry>	Specify the initial geometry for main window using the format WIDTHxHEIGHT+XPOS+YPOS.
-globalInclude	<file>	Define global Verilog include files. Global includes are processed before any other source files (this option can be repeated multiple times).
-guessBus	<open> <close>	Guess buses based on net and port names with a bit subscript enclosed in the given 'open' and 'close' characters.
-guessInstArray	<open> <close>	Guess instance arrays based on instance names with a bit subscript enclosed in the given 'open' and 'close' characters.
-help | -h		Print a help text with a short description of each option.
-hierSep	<hiersepchar>	Set the desired hierarchy separator 'hiersepchar' of your choice. Any character can be used. To be able to identify the hierarchy separator, a character that is not already used in an identifier should be used.
-iconify		Start the main window iconified.
-ignoreCase	on|off	Case-insensitive parser. (The default value for this option is on.)
-ignorePragmas	on|off	Ignore all Pragmas. (The default value for this option is off.)
-ignoreTranslate	on|off	Ignore only Translate and Synthesis Pragmas. (The default value for this option is off.)

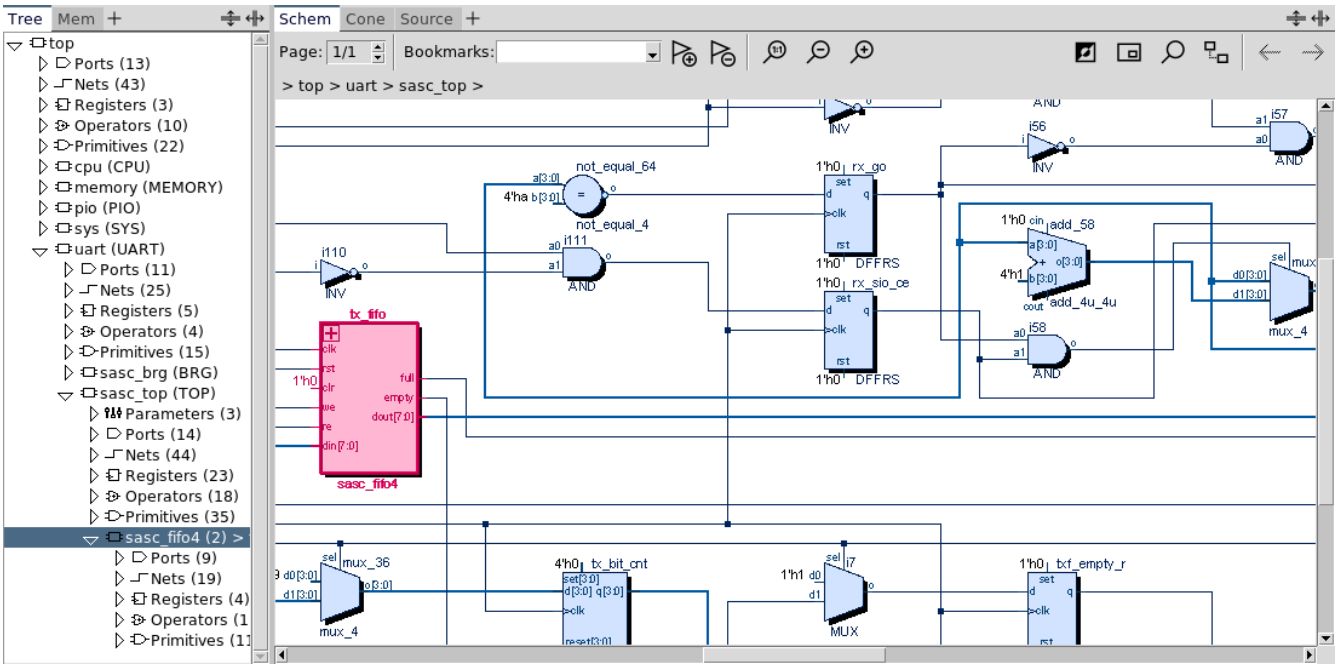
Option	Parameters	Description
-ignoreUnit	<unit>	Do not elaborate the VHDL unit <unit>; any VHDL unit whose name matches <unit> case-insensitively will not be elaborated.
-incdir	<dir>	Define an include directory (this option can be repeated multiple times).
+incdir+	<d1>+<d2>…	Define include directories.
-info	None|Error|Warning|Verbose|Debug	Level of verbosity for issued messages. (The default value for this option is Error.)
-initialDirectory	<directory>	Use this initial directory for the file dialog.
-L	<library>	Search for Verilog modules and packages in <library>; multiple libraries can be specified using multiple -L options; libraries are searched in the order of the -L options.
-lef		Read the input file as a LEF file.
-liberty		Read the input files as a Liberty library.
+libext+	<e1>+<e2>…	Define the file name extensions for the -y option.
-library	<name>	Parse all files following this option into a library with the given name.
-logfile	<file>	Generate log file.
-maxErrCnt	<num>	Set maximum number of errors that can occur before the parser stops reading the input file(s). (The default value for this option is 0.)
-minRamSize	<min>	Minimum size (in bits) a RAM needs to be before RTL elaboration extracts it. Value 0 means no lower limit. (The default value for this option is 4096.)
-operContents	on|off	Create operator implementation. (The default value for this option is on.)
-pedantic	on|off	Toggle pedantic language checking mode (in relaxed mode some errors are just warnings and some warnings are suppressed). (The default value for this option is on.)
-pluginDir	<dir>	Specify plugin directory.
-preserveAssign	on|off	Preserve assignments in the netlist. (The default value for this option is off.)
-procHier	on|off	Create hierarchy for always and process blocks. (The default value for this option is off.)
-profile		Enable profiling of the GUI and ZDB API.
-project	<file.vpj>	Use options from the given project.

Option	Parameters	Description
-quickMode		Open a zdb binfile in quick mode.
-reduceInvChain	on|off	Replace odd number of INVs in a chain by one INV and remove an even number of INVs in a chain. (The default value for this option is off.)
-removeBuffer	on|off	Remove all BUF and WIDE_BUF instances and merge the connected nets. (The default value for this option is off.)
-resolveDuplicates	on|off	Resolve duplicate cells. If "off" all duplicate cells are renamed. (The default value for this option is on.)
-sdf	<file.sdf>	Specify a SDF file to attach.
-sdfTop	<topname>	Specify the name of the design under test.
-spos	on|off	Create source code references. (The default value for this option is on.)
-sym2zdb		Preload symbol library file(s) given with the -symlib option.
-symlib	<symlib>	Specify a symbol library file.
-systemVerilog | -sysverilog | -sverilog | -sv		Read RTL SystemVerilog 2009.
+systemverilogext+	<e1>+<e2>…	Define SystemVerilog file name extensions.
-sysVerilog2005		Read RTL SystemVerilog 2005.
-tempDir	<dir>	Specify a directory for temporary files.
-time		Print CPU time consumption (requires enabled progress updates).
-title	<toolname>	Display the given toolname in the titlebar.
-top	<name>	Define this module as the top module. If * is set, then all unreferenced cells are used as top.
-topLibrary	<name>	The library containing the top-level design.
-userware	<file>	Load Userware (source a Tcl file).
-userware2	<file> <arg>	Load Userware with one argument in argv.
-userware3	<file> <arg1> <arg2>	Load Userware with 2 arguments in argv.
-userwareArgs	<file> <argList>	Load Userware with a list of arguments in argv. On the command line the list needs to be one quoted argument which will be expanded before it is passed to the specified Userware script.

Option	Parameters	Description
-userwareEval	<script>	Evaluate Userware following in next argument. On the command line the list needs to be one quoted argument which will be evaluated as a Userware script.
-v	<libfile>	Read <libfile> as Verilog library file.
-verilog		Read Verilog netlist.
+verilog1995ext+	<e1>+<e2>…	Define Verilog 1995 file name extensions.
-verilog2001 | -v2k | +v2k		Read RTL Verilog 2001.
+verilog2001ext+	<e1>+<e2>…	Define Verilog 2001 file name extensions.
-verilog95		Read RTL Verilog 95.
-verilogAMS		Read RTL Verilog AMS.
-version		Print the tool version.
-vhdl2000		Read RTL VHDL 2000.
-vhdl2008		Read RTL VHDL 2008.
-vhdl2019		Read RTL VHDL 2019.
-vhdl87		Read RTL VHDL 87.
-vhdl93		Read RTL VHDL 93.
-vhdlLibPath	<dir>	Look for and store precompiled VHDL libraries in <dir>.
-wait_for_license | -waitForLicense	<sec>	Wait 'sec' seconds for a license. If the value is -1 then the started tool will not wait for the next free license. Use a value of 0 to wait forever. (The default value for this option is -1.)
-wdb | -vcd	<file.vcd>	Specify a VCD or WDB file.
-wdbCreateVectors | -vcdCreateVectors	on|off	Create vectors based on scalar names. (The default value for this option is off.)
-wdbNameOfDUT | -vcdTop	<path>	The scope name of the design under test.
-wdbPathToDUT | -vcdTop	<path>	The scope path which needs to be removed before before the remaining path will be matched against the design. The first name of the remaining scope path will be matched against top module names of the design. If there is only one top module the scope name need not to match
-workspace	<file.ws>	Use options from the given workspace.
-y	<libdir>	Read files matching the extension given with +libext+ from the specified directory as Verilog library files.

Pane Window

The Pane window is a container for a Tab group. In the built-in default GUI layout there are only two Pane windows. Additional Pane windows can be created on demand using the split icons in the tab pane. From the Window menu new toplevel Pane windows can be created.



Tab Group

The tab group on the left side provides fast access to the Tree and Mem.

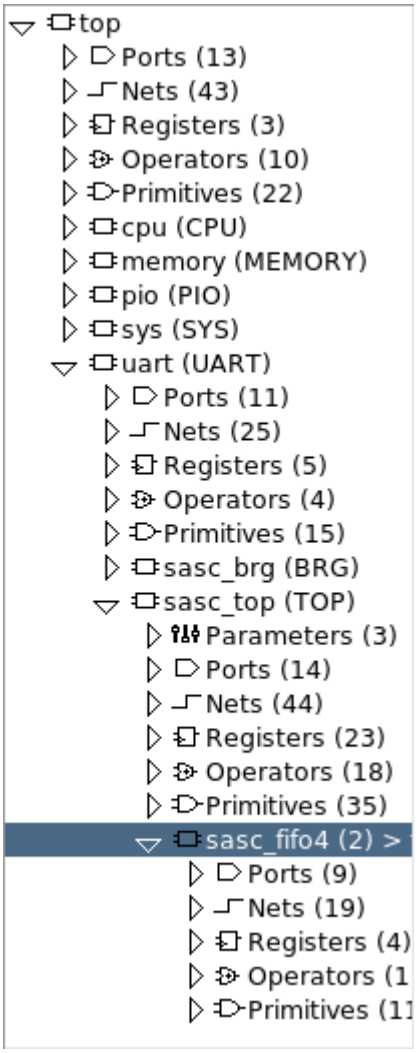
The tab group on the right side provides fast access to Schem, Cone and Source.

Click on the tab's name to bring its window to the front (activate it).

Additional tabs can be added using the special "+" tab.

The Tree Window

The Tree window displays the design hierarchy.



The Tree widget supports two different view types.

The module based view show a folded tree sorted and grouped by modules. Modules that are instantiated several times display only one instance, i.e. the hierarchy tree is "folded". Such modules have a small left to right arrow (▯) to indicate that it is instantiated multiple times.

To open the module's instantiation list click on the down-arrow at the end of the line. To select one specific instance, click on the instance name in the drop down list.

The instance based view show each module instance and sort the tree by instance names.

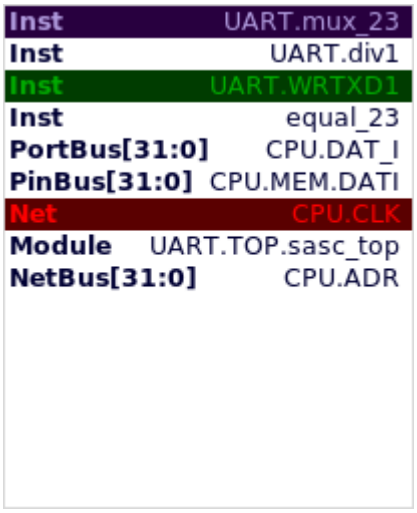
Each tree node of a hierarchical module instance in the tree can contain up to six sub-entries to group the module contents. The "Ports" and "Nets" entries allow for quick access to all ports, portBuses, nets and netBuses. The "Primitives", "Registers" and "Operators" entries group primitive instances by its function and type. Instances can be grouped in a user defined custom group by flagging instances with the `group` flag.

Modules with parameters have an additional entry "Parameters" which lists the parameters and their respective values.

The default action in the context menu of the tree is **Current Module**, i.e. if you double-click on a module in the tree, it will become the 'current module' and it will be loaded and displayed in the Schem window. The Schem window's tab will be activated, if not already open.

The Memory Window

The Mem window can be used as a notepad.



You can Drag & Drop any object to and from the Mem window.

Objects that are currently highlighted will be displayed in the corresponding highlight color.

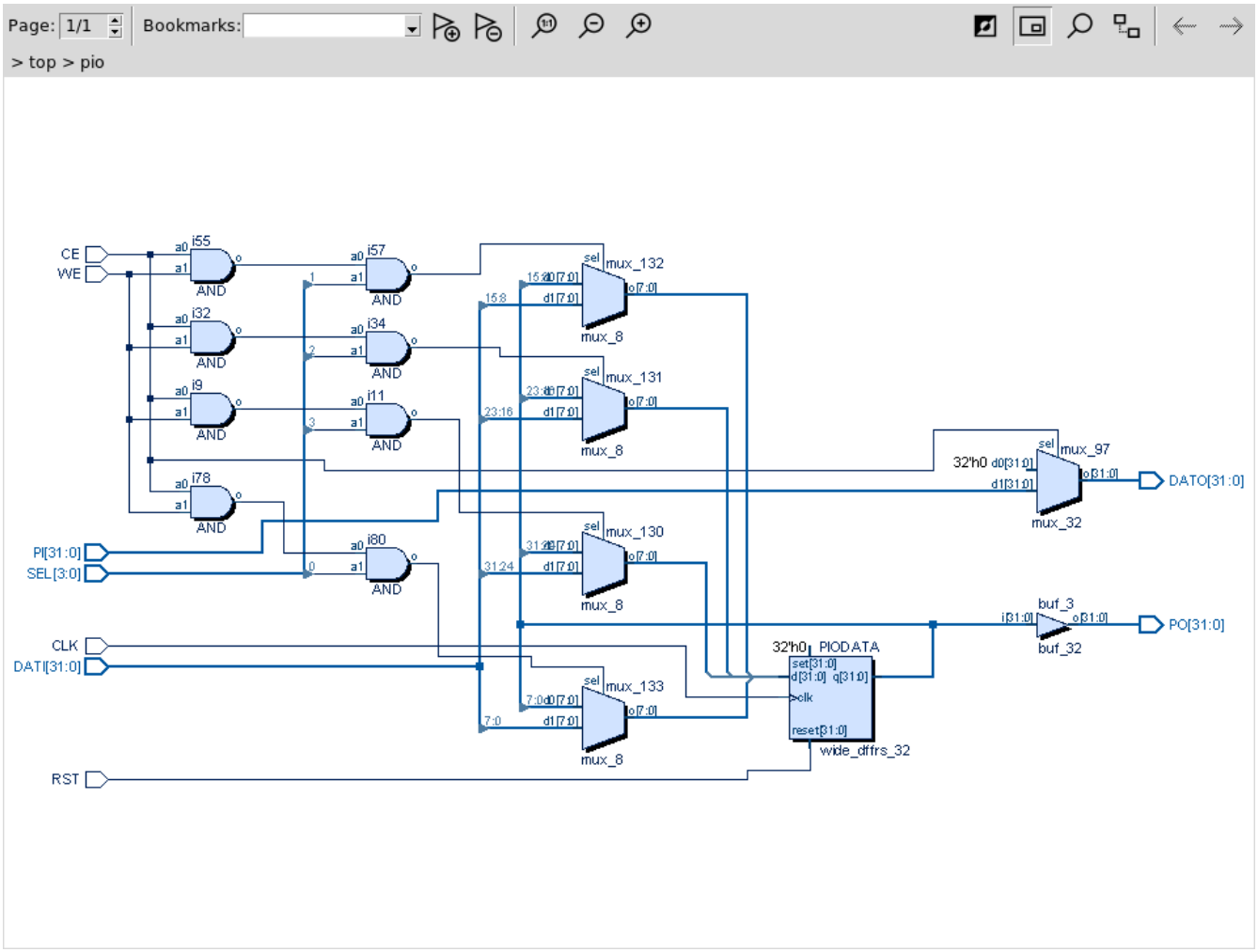
Invalid entries, i.e. those who cannot be matched to a valid OID in the current database are grey and disabled.

The picture above shows a Mem window that contains 9 valid objects, 3 of them are highlighted in different colors.

The default action of the context menu in this window is **Goto**, i.e. the selected object is temporarily highlighted in each window (the 'current module' as displayed in the Tree window may change).

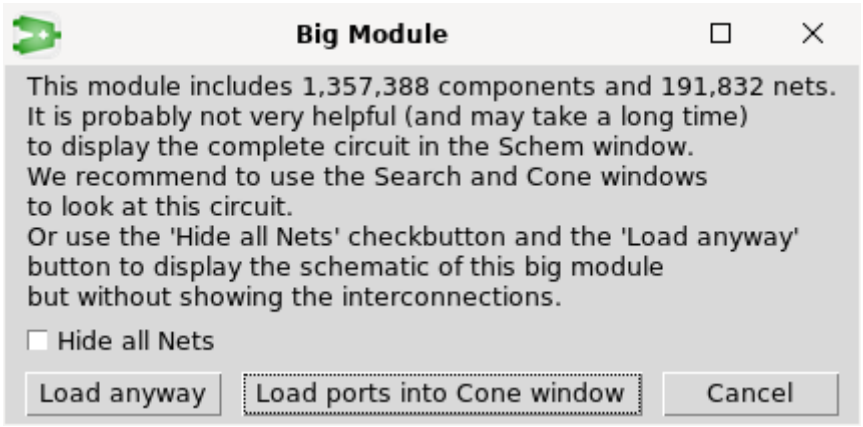
The Schematic Window

The Schem window displays a complete module (split into pages).



The **Bookmark** feature in the toolbar can be used to save/restore the current schematic view. The default name of a bookmark (created using an unique number and the module name) can be changed.

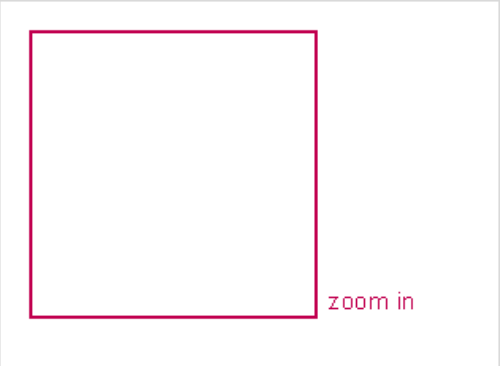
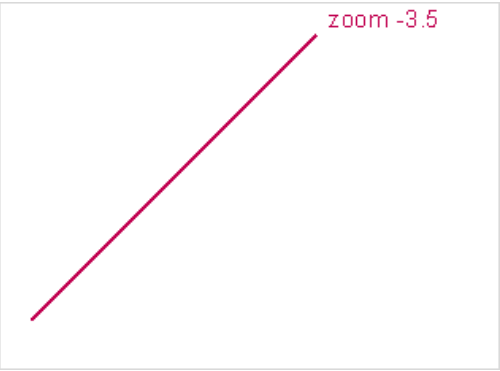
If the module is larger than the page size you selected in Preferences it must be split into different pages. Splitting is done automatically. See below, how to change between the different pages. If the module to be displayed exceeds the limit set in Preferences, a dialog box is presented.

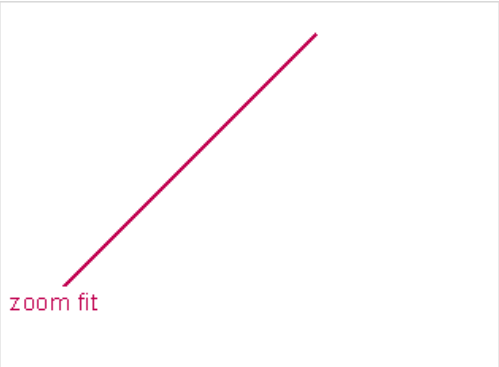
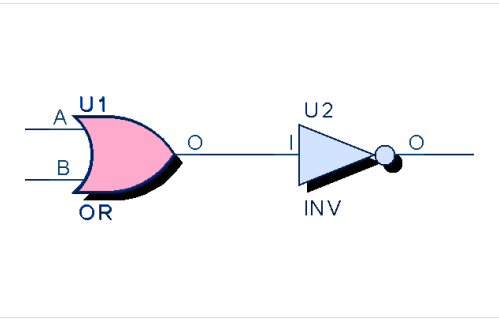
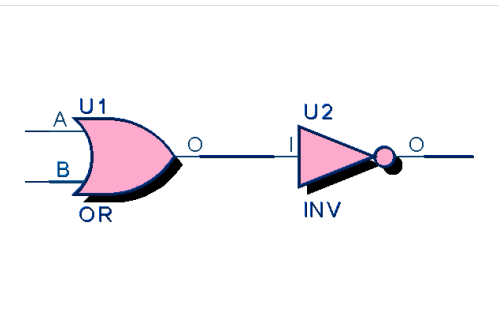


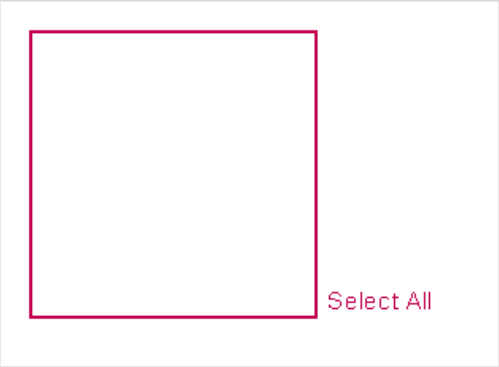
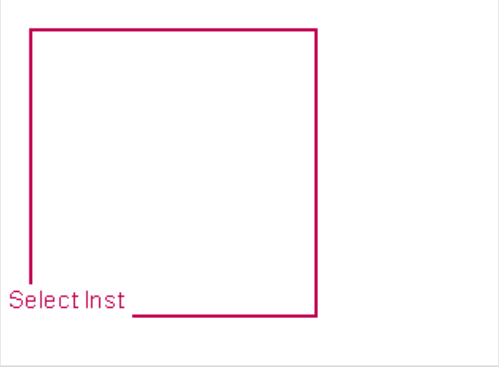
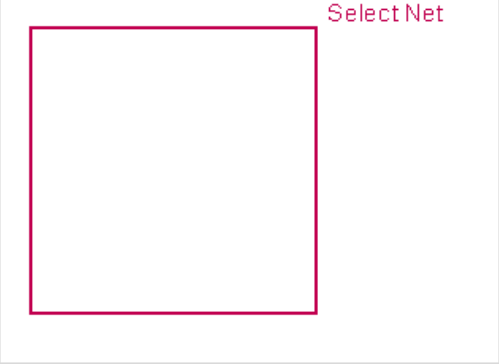
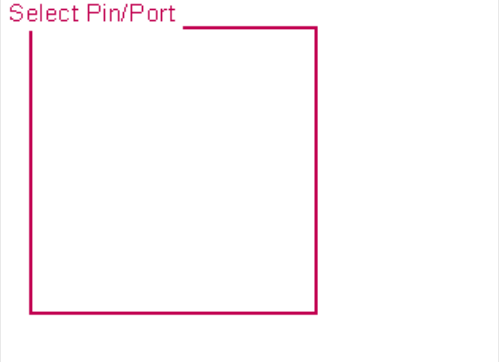
The Schem window and Cone window have many functions in common. The table below explains some of the common features and later the Schem window specific part.

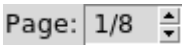
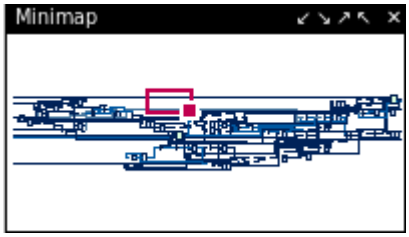
The stroke mouse button depends on the setting of the Drag & Drop mouse button. It is always the opposite, e.g. if Drag & Drop is bound to the right mouse button, then the strokes are bound to the

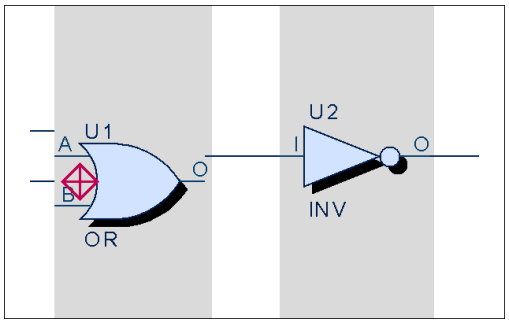
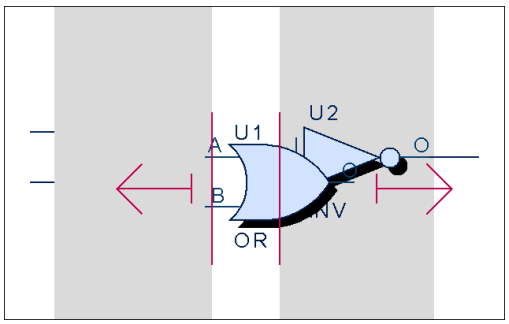
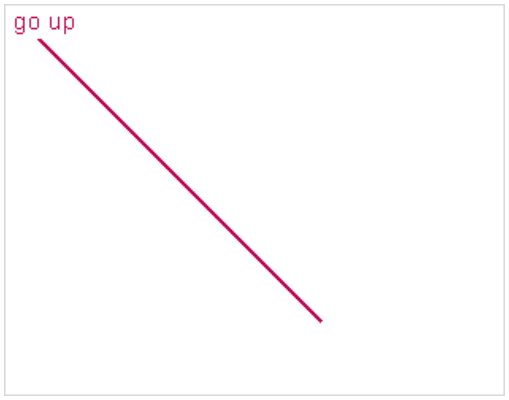
left mouse button and vice versa.

Action	Info	Keyboard(*)
	Schem- & Cone window common functions	
zoom in	 Use the stroke mouse button to drag a rectangular (zoom) area. Start dragging from top left corner and stop dragging in the bottom right corner of the area to be zoomed in. You can also use the mouse wheel while the `Ctrl` key is pressed to zoom in.	`I`
zoom out	 Use the stroke mouse button to drag a line in top right direction. The longer the line the higher the zoom-out factor will be. You can see the zoom factor while dragging; minimum zoom factor is 0.5, maximum is 3. You can also use the mouse wheel while the `Ctrl` key is pressed to zoom out.	`O`

Action	Info	Keyboard(*)
zoom fit	 Use the stroke mouse button to drag a line in bottom left direction. The schematic will be zoomed to fit completely into the current window. The zoom factor is automatically adjusted whenever the window size changes.	`F`
regenerate	accessible via context menu - it will regenerate the displayed module or schematic excerpt.	`Shift` + `R`
optimize	accessible via context menu - optimize the displayed module: keep the placement of ports and instances, but optimizes the net routing.	`Shift` + `O`
select single objects	 Single-click any visible object in the window with the left mouse button. Selected objects have a thicker borderline.	none
select multiple objects	 Keep the `Ctrl`-key pressed while single-clicking any visible object in the window with the left mouse button. If the object was already selected, it will become deselected.	none

Action	Info	Keyboard(*)
select multiple objects	Keep the `Shift`-key pressed while dragging the mouse with the stroke mouse button pressed down to open a rectangular area. Depending on the direction different object types within the area will be selected when releasing the mouse button.  A rectangle to the bottom right will select objects of all types.  A rectangle to the bottom left will only select instance objects.  A rectangle to the top right will only select net objects.  A rectangle to the top left will only select pin or port objects.	none
highlight	accessible via context menu - it will highlight the current selection.	`Ctrl` + `H`

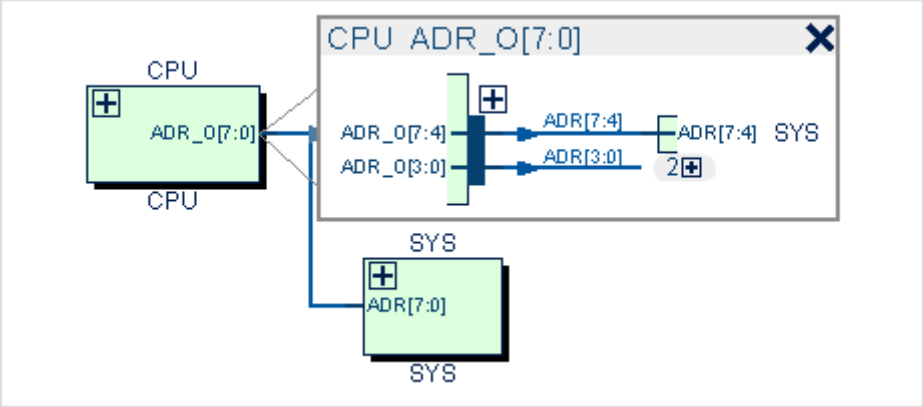
Action	Info	Keyboard(*)
unhighlight	accessible via context menu - it will unhighlight the current selection.	`Ctrl` + `Shift` + `H`
change page	 If there are multiple schematic pages then a spinbox appears in the top left corner of the window. The up and down buttons allow you to navigate to the previous and next schematic page. A label left to the buttons shows the current page and the total number of pages. A click into the page field opens a listbox to provide fast access to each page.	`PageUp` `PageDown`
	The **Greymode** icon will toggle "greymode". The complete displayed schematic will be grayed out, except for the highlighted components.	none
	 The **Minimap** icon will toggle the visibility of a Minimap window in the bottom right corner. Once the Minimap window is visible it can be moved within the window.	none
	The **Magnify** icon will toggle the visibility of a Magnify window in the bottom left corner. Once the Magnify window is visible it can be moved within the window.	none

Action	Info	Keyboard(*)
inst drag	  Keep the `Ctrl`-key pressed while dragging the mouse in any direction with stroke mouse button (opposite of the Drag & Drop button) pressed down to move one or more components (inst, port, portBus or page connectors) or a net object (net or netBundle) to a new location and update the net routing. For net objects only the y position inside the component column (level) can be changed. Multiple components can be dragged by selecting them first and then start dragging at one of the selected components. A draft version of the moved component(s) is drawn while being moved. The echo of the component's symbol shape identifies that the target location is valid. Dragging of device instances is ignored.	none
Schem window specific functions		
hierarchy up	 Use the stroke mouse button to drag a line in top left direction. The instance we are coming from will be selected in Schem window.	none
hierarchy down (default)	A double-click on a hierarchical (non-primitive) instance will load its down-module in the Schem window.	none

Action	Info	Keyboard(*)
	The **Nethide** icon will toggle the hide flag at all displayed nets. Hidden nets are not routed through the schematic page but only displayed as stubs (with a dot at the end) at all connected pins, which allows to select the net object. Each selected hidden net is drawn with a zigzag path to all connected pins. A double click on a stub displays and routes the net. A double click on a displayed net will set the hide flag and the net disappears from the schematic view.	none

NOTE For the keyboard shortcuts to work properly, the Schem window must have the focus. Use the `Tab`-key to change the focus or click into the Schem window to transfer the focus.

Connectivity Lens



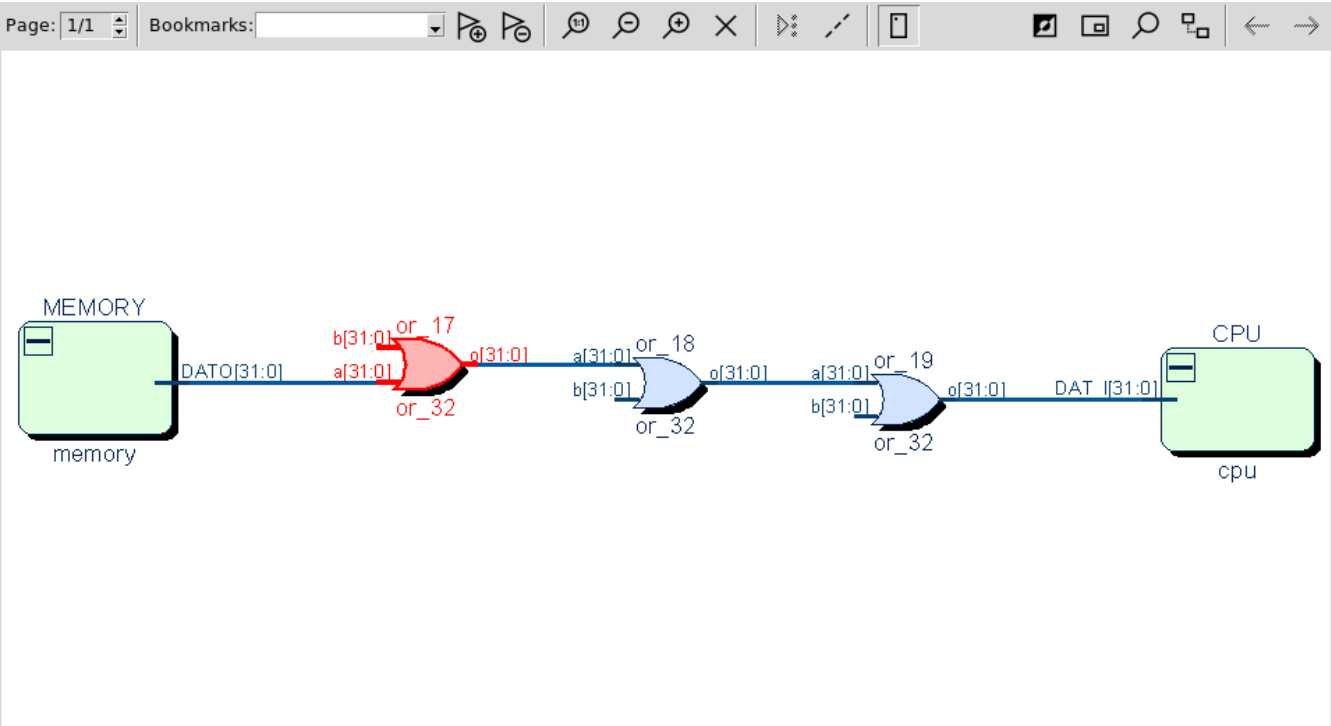
The Connectivity Lens is an inline window accessible from the Popup menu of the Schem or Cone window to display detailed connectivity information for the selected pin or port. There can only be one Connectivity Lens window on a schematic page.

If the selected object is a bus then all single-bit connections (from the msb to lsb) are listed, while grouping adjacent pins into sub-buses if that is possible without any ambiguity. One condition for this grouping is that the adjacent pins can form a regular bus name. For example, `A[3]`, `A[2]` and `A[1]` qualify to form the regular bus name `A[3:1]`.

Small triangles indicate the logical signal flow direction from the driver (e.g. output pin) to its receiver (e.g. input pin).

The Cone Window

The Cone window displays schematic excerpts (paths, cones, etc.) and is used for **Incremental Schematic Navigation**.

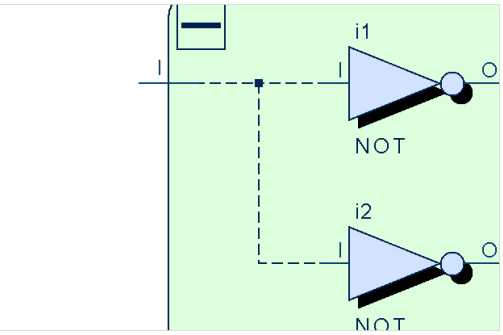
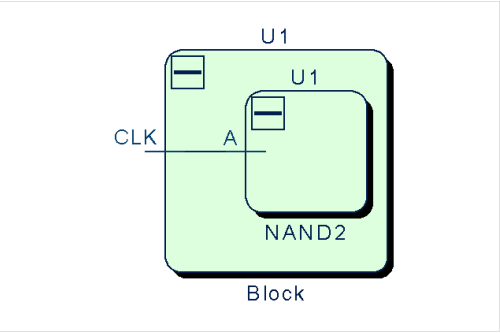


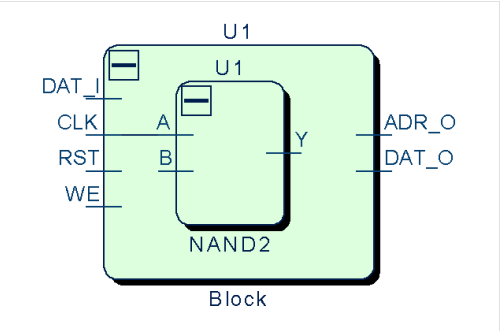
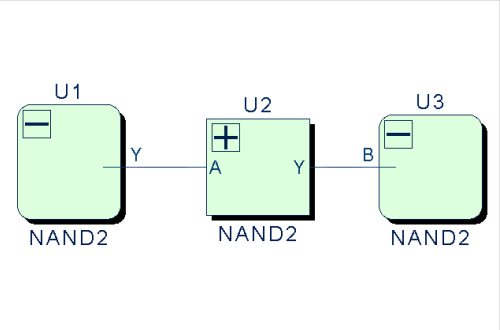
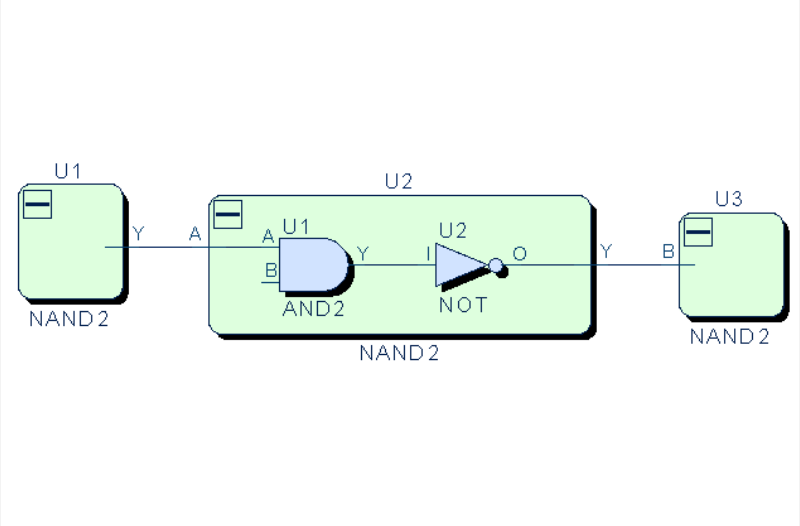
Its behavior is pretty much the same as the Schem window but it allows for interactive incremental disclosure of the schematic, even across hierarchy borders (giving a kind of "flat-view" on the database).

The **Bookmark** feature in the toolbar can be used to save/restore the current view of the Cone window. The default name of a bookmark (created using the term Bookmark followed by an unique number) can be changed. In addition the Cone window provides a second, file based, snapshot mechanism accessible through the Popup menu.

The following table explains the functions that only apply to the Cone window. For general functions like, e.g. zoom, selection or highlighting, see the table in the Schem window section.

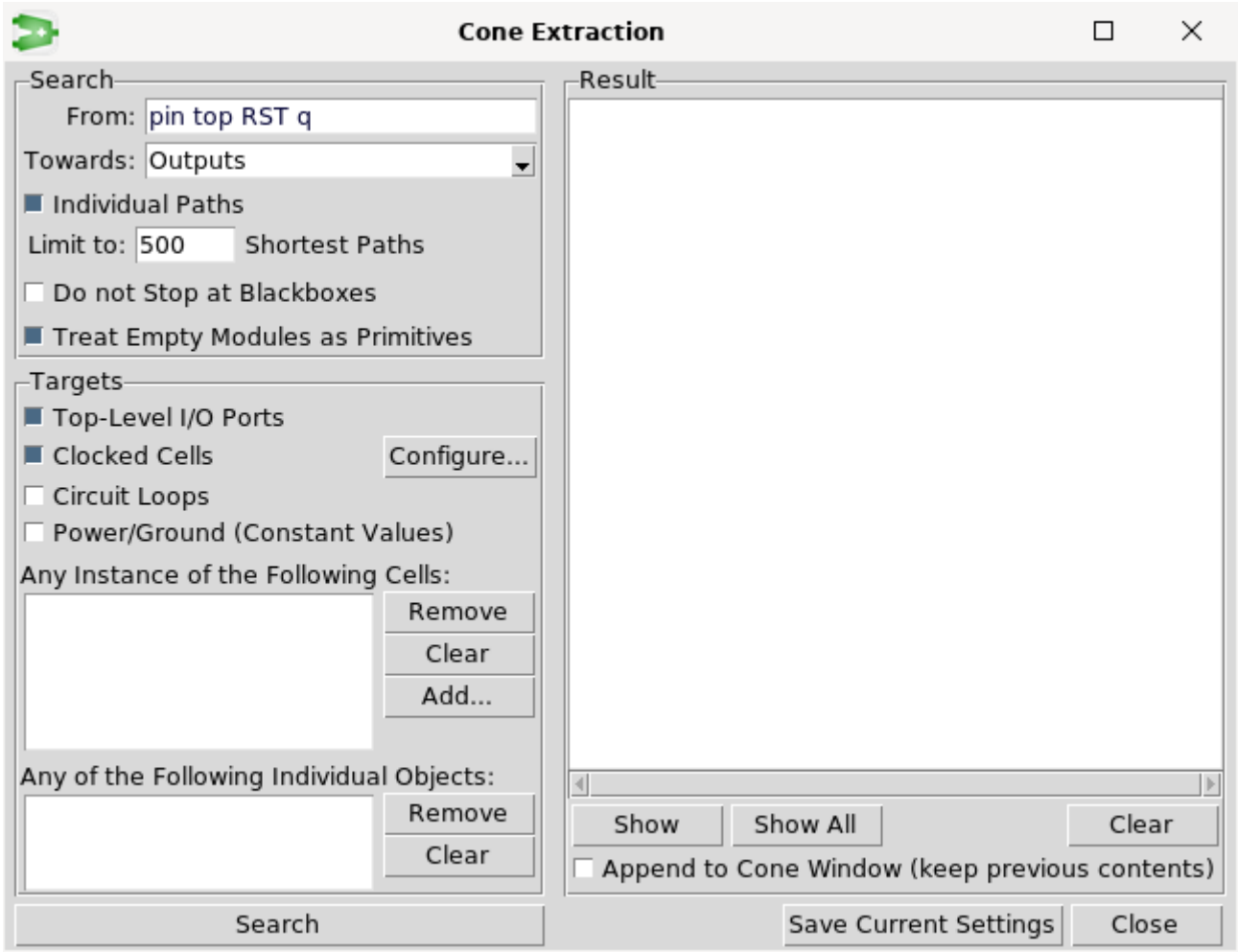
Action	Info	Keyboard(*)
Cone window specific functions		
add objects	To add a component into the Cone window, use the Drag & Drop feature. You can drop any valid database object into the Cone window. Dropping one or more database objects, will cause it to be loaded (if not already visible) and centered under the mouse cursor. Dropping pins will cause the corresponding instance to be loaded.	none
remove objects	To delete one or more components from the Cone window, select it and choose 'Remove' from the context menu. Deleting instances that contain other objects cause them to be removed, too.	`Del`
remove all objects	To delete all components from the Cone window, choose 'Clear' from the context menu.	`Ctrl` + `Del`

Action	Info	Keyboard(*)
more (default action)	 To incrementally extend the schematic excerpt, double-click with the left mouse button on any component you see in Cone window: Nets & NetBuses are displayed in dashed style to indicate that they are not completely loaded; "more" will load more components connecting to that Net/NetBus until the Net is complete. It then turns into solid-style. Ports & PortBuses as well as Pins & PinBuses can also be double-clicked on. It causes the corresponding net(s) to be loaded or extended (see item above), if it is already loaded. PinBuses & PortBuses can be double-clicked on with `Ctrl`-key pressed. A small dialog pops up that allows you to select the subpin or subport. Pins & Nets can also be double-clicked on with `Ctrl`-key pressed. A small dialog pops up that allows you to select the component to load. Also all connected components can be loaded.	none
autohide on	 Double-click with the left mouse button on the border of an instance to turn the autohide feature 'on'. Autohide causes (temporarily and permanently) unconnected instance pins to be hidden automatically (until they are connected to a 'loaded', i.e. visible, net). Alternatively, you may also set the 'Hide Unconnected Pins' option either in the toolbar of the Cone window or in the Cone tab of the Preferences dialog.	none

Action	Info	Keyboard(*)
autohide off	 Double-click with the left mouse button on the border of an instance to turn the autohide feature 'off'.	none
fold	 The content of the selected hierarchy boxes is hidden thus the boxes look empty. This can be quite useful after a cone extraction to reduce the amount of details displayed in cone. A folded hierarchy box is locked, the hierpins are not displayed.	`Ctrl` + `F`
unfold	 The content of the selected folded hierarchy boxes is shown again.	`Ctrl` + `U`

Action	Info	Keyboard(*)
Extract to Driver/Load	 If a pin or port is selected in the Cone window, the Extract to Driver/Load button in the toolbar lets you quickly extract the cone towards the driver/towards the loads. Note that the toolbar button changes its icon depending on whether an input or an output port/pin is selected.	none
Expand Nets	 Expand all partially loaded nets and netBuses (which are shown with dashed lines).	none
	The **Nethide** icon will toggle the hide flag at all displayed nets. Hidden nets are not routed through the schematic page but only displayed as stubs (with a dot at the end) at all connected pins, which allows to select the net object. Each selected hidden net is drawn with a zigzag path to all connected pins. A double click on a stub displays and routes the net.	none
History	 If the "Remember History" check-button is enabled then the two buttons in the top right corner of the Cone window allow you to move back and forward in the history of the Cone window.	none
Bookmark	The Bookmark feature allows to save the current Cone window content as a Cone bookmark and restore it afterwards.	`Ctrl` + `D`
Snapshot	The Snapshot feature allows to save the current Cone window content as a Cone snapshot and restore it afterwards. Snapshots are saved to files that can later be loaded or manually copied to another machine for review (even without the design database loaded). The context menu has a separate sub-menu (called 'Snapshot') for the snapshot functions save and open. Snapshot are plain text files; they have the extension `.bm5`.	`Ctrl` + `S` `Ctrl` + `Shift` + `O`
Save Cone as Verilog	The contents of the Cone window is saved as a Verilog netlist. The Save Cone as Verilog feature is available through the context menu of the Cone window.	`Ctrl` + `Shift` + `S`

Cone Extraction Dialog



The Cone Extraction dialog is accessible either through the Tools main menu (**Tools › Extract Cone**) or the context menu of the Cone window. If invoked from the Popup menu then the selected object is used as the start point. Drag & Drop can also be used to define the start object.

Using the Cone Extraction dialog, you can search for all paths from the given start point to all reachable targets either towards the inputs or towards the outputs.
[extract:dir]

If the option Individual Paths is enabled, the search result is returned as a list of individual paths, if the option is disabled, the complete cone is returned, which is a summary of all paths.
[extract:path]
[extract:pathLimit]

Usually a blackbox instance is a stop cell. If the option "Do not Stop at Blackbox" is enabled then all unknown pins are treated as bidirectional and the extraction goes through the cell in all directions.
[extract:unknown2IO]

The cone extraction would stop at empty modules, because there is no path through the module. With the option "Treat Empty Modules as Primitives", an empty module can be treated as a primitive and the extraction will not dive into it, but continue at all pins of the module instance.
[extract:emptyModAsPrim]

The cone extraction can stop at the following targets:

- Top-Level I/O Ports - run the extraction to all top-level I/O ports.

[extract:targetIO]

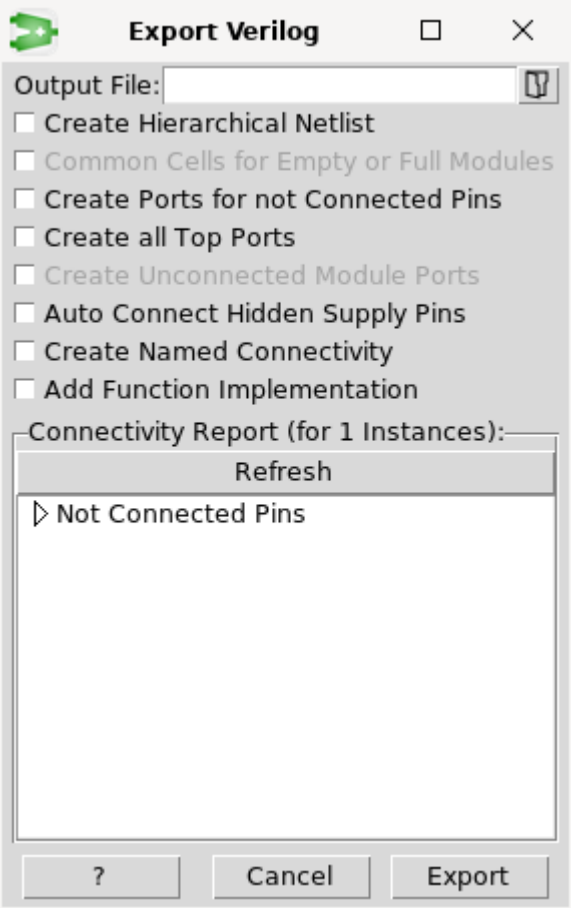
- Clocked Cells - stop at all clocked cells.
  [extract:clkCell]
- Circuit Loops - stop at a loop in the circuit.
  [extract:targetLoop]
- Power/Ground (Constant Values) - stop at supply nets or nets with a constant value.
  [extract:targetPGNet]
- Any Instance of the Following Cells - all instances of the specified cells are targets. Use Drag & Drop to specify a list of cell names.
  [extract:targetCells]
- Any of the Following Individual Object - a specific object is the target. Use Drag & Drop to specify a list of possible target objects.
  [extract:targetOIDs]

The search result is shown in the Result list of this dialog. The list of paths is sorted by path depth (the shortest path is the first). A double click in the result list loads the path or cone to the Cone window. Use the "Append to Cone Window" option to keep the previous contents in the Cone window loaded.
[extract:appendResult]

Using Drag & Drop always appends a path to the Cone window.

Save Cone

Save Cone as Verilog



The contents of the Cone window can be saved as a Verilog file. The Save Cone as Verilog feature is available either through the Tools main menu (**Tools › Save Cone as Verilog**) or the context menu of the Cone window.

In a dialog window similar to the image above, you get a detailed Connectivity Report of the Cone fragment you want to export. You can drag objects from the report window to the Cone window. In the Cone window the object is highlighted using the goto color.

If you are satisfied with the report, then enter a filename and press the **[Export]** button to save the contents of the Cone window as a Verilog netlist.

If the option "Create Hierarchical Netlist" is enabled, a hierarchical Verilog netlist will be created.

The option "Common Cells for Empty or Full Modules" controls whether modules are instantiated multiple times or not.

The option "Create Ports for not Connected Pins" will add an extra level of hierarchy with interface ports for all not connected pins in the top hierarchy. This extra module is instantiated in the design top.

The option "Create all Top Ports" will create all top-level I/O ports of the design top regardless of the visibility in the Cone window.

The option "Create Unconnected Module Ports" will create all module ports visible in the Cone window even if they are not connected.

The option "Auto Connect Hidden Supply Pins" will automatically connect all supply pins that are

hidden and thus not displayed in the Cone.

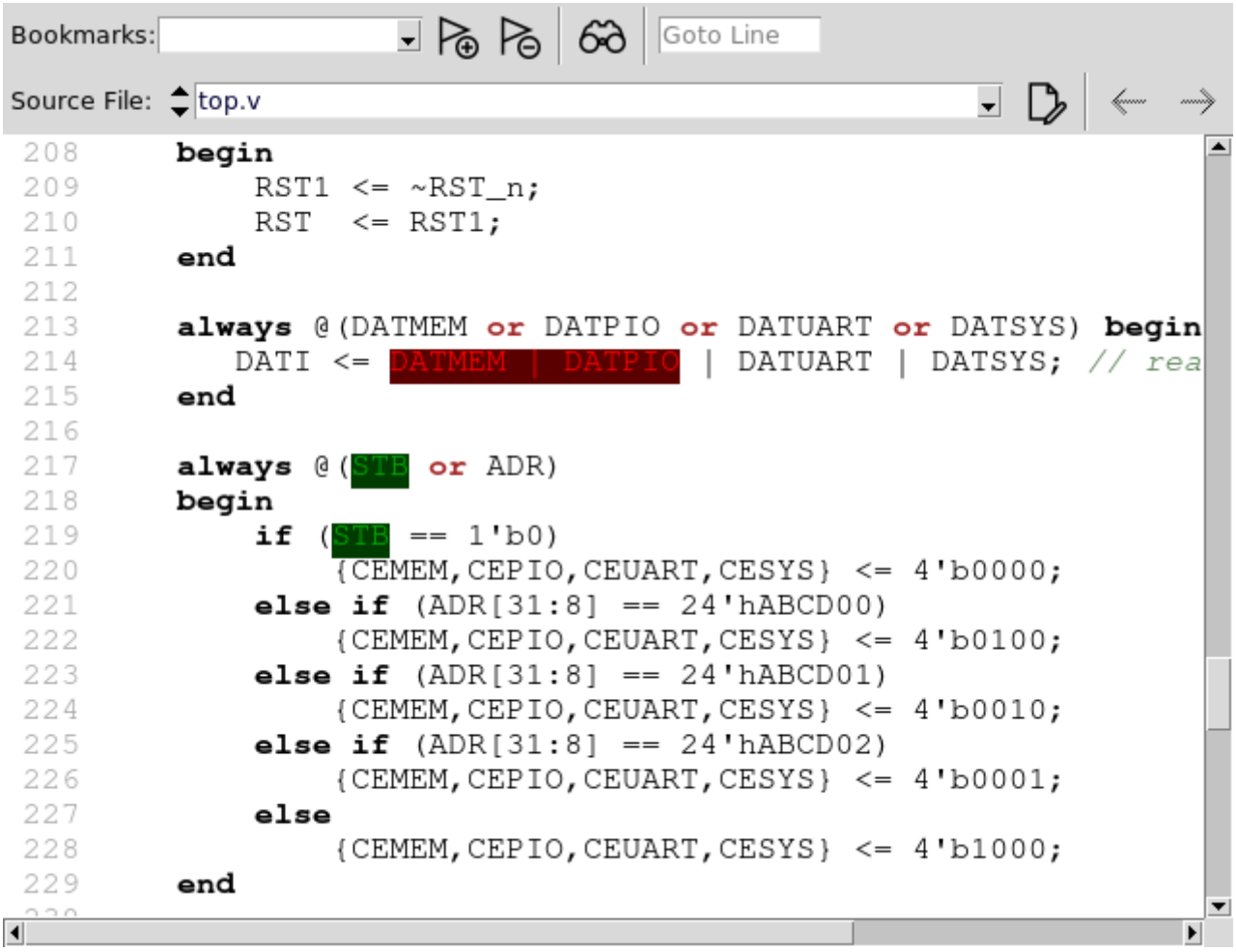
If the option "Create Named Connectivity" is enabled then named connectivity will be created in the generated Verilog netlist. Otherwise implicit connectivity will be created.

If the option "Add Function Implementation" is enabled then the primitive function of cells with a known function is implemented (this can be used by a synthesis tool or to run a simulation).

Source Window

The Source window displays the source code of the input files used to generate the schematic.

The screenshot shows an example source file loaded in the Source window; several objects have been highlighted.



Highlighting

All highlighted objects will be displayed in their corresponding highlight color for easy cross probing.

The Toolbar

The Source window's toolbar offers several actions:

- The **Bookmark** feature can be used to save/restore line positions in the displayed source file.

Bookmarked lines are marked with a black flag (⚑) icon before the line number. The default name of a bookmark (created using the filename and line number) can be changed.

- The **Find in file** button opens a panel for incremental text search within the displayed source file and across all files of the current project.
- **Goto line** jumps to the entered line number in the current file.
- The **Sourcefile** drop-down list shows the currently displayed file and allows for selecting other files to display.
- The **Open Editor** button executes an external editor, loads the source file and scrolls to the first line displayed in the Source window. If there is either an object marked by the Goto function or a line marked by "Goto Line" then this information is used as the start line passed to the editor. You may specify the editor in Source tab of Preferences dialog.
- The **history** buttons allow for quick navigation to previously visited source file positions.

The Action Bar

If an object is selected in the Source window, the **Action Bar** opens below the object, offering object specific information and navigation actions.

Depending on the type of selected object, the following information/actions are available:

Modules

- △ goes 'up' to the current instance of the module.
- A list of instances.
- **[Show]** goes to the selected instance.

Instances

- ▽ goes 'down' to the corresponding module.

Ports/Port Buses

- △ goes 'up' to the corresponding pin/pin bus.

Pins/Pin Buses

- ▽ goes 'down' to the corresponding port/port bus.

Nets/Net Buses

- ◁ and ▷ go to the previous/next occurrence of the selected net.
- •◁ goes to the net's 'driver'; if multiple drivers exist, ⁝◁ is shown.
- A drop-down list of connected pins/ports; if the object is a 1-to-1 connected net bus, the connected pin buses/port buses are shown; if the object is a ripping net bus, pins/ports connected to the individual bus members are shown.
- **[Show]** goes to the selected pin/port.

Drag & Drop

The Source window supports Drag & Drop: if one or more objects are dropped into the Source window, then the tool will scroll to the closest line number (relative to the currently displayed source file) of any dropped object and selects it. You can also drag objects from the Source window to the other windows (e.g. Schem or Cone window).

The Search Window

The Search window can be used to search the current database for certain components.

The Search window can be opened by clicking the icon in the toolbar or by selecting **Window › Search** from the main menu.

Simple Search Mode

The simple mode of the Search window is composed of a combobox for selecting the Object Type to search for, a text entry field for entering a glob-style search pattern, and a **[Search]** button.
[Search:type], [Search:name]

The **[Search]** button (or the Enter key in the search pattern field) starts the search and searches for all database objects matching the entered pattern in all levels of hierarchy.

A long running search may be terminated by clicking the **[Stop]** button of the progress dialog.

The result list displays matching objects along with their type.

The default action in the context menu of the result list is **Goto**, i.e. if you double-click on a result entry and you have the Schem window open, its parent module will become the 'current module' and the result object is selected.

Advanced Search Mode

The Search window also provides an advanced mode that can be activated by clicking on the icon. The advanced search allows for a more precise specification of the search parameters and usually results in a better search performance than the "simple mode".
[Search:advanced]

Object Type

As in the "simple search mode", the object type option can be used to restrict the search to specific types of objects.
[`Search:type`]

Name Pattern

A search pattern for the object's name. This does not include the object's path.
[`Search:name`]

Search Mode

- Entire Design - search in the whole data base, glob-style characters in the path pattern also match hierarchy separators.
- Specific Hierarchy - search in the whole data base, glob-style characters in the path pattern **don't match** hierarchy separators. [`Search:mode`]

Path Pattern

A search pattern for the object's path.

You may drop objects of type "module", "primitive", and "inst" into the entry field to populate it with the path of the dropped object.
[`Search:path`]

Top Pattern

A search pattern for the object's top module.
[`Search:top`]

Hiersep

Specify the hierarchy separation character used in the path pattern, e.g. if the path pattern is `top/cpu/adder`, use `/` as the hiersep character.
[`Search:hiersep`]

Cell Pattern

When searching for instances (⇒ `Object Type` is `Inst` or `Any`), this option restricts the list of reported instances to instances of the specified cell type, e.g. if `Cell Pattern` is `cpu`, only instances of the `cpu` cell are reported. If you use glob-style patterns (e.g. `and*`) only instances of cells with a cell name matching this pattern are reported.

When searching for pins/pinbuses (⇒ `Object Type` is `Pin`/`PinBus` or `Any`), only pins/pinbuses of instances instantiating a matching cell are reported.

When searching for ports/portbuses or nets/netbuses (⇒ `Object Type` is `Port`/`PortBus`/`Net`/`NetBus` or `Any`), only ports/portbuses/nets/netbuses of a matching cell are reported.

You may drop objects of type "module", "primitive", and "inst" into the entry field to populate it with the cell name of the dropped object.
[`Search:cname`]

Case

- smart - automatically select case insensitivity if the search pattern contains no uppercase characters. Otherwise search case sensitive.
- insensitive - ignore upper/lower case of characters in search patterns.
- sensitive - search case sensitive.
  [`Search:case`]

Exact Search

If this option is enabled, special characters in patterns (`*`, `?`, `[`, `]`, ...) are interpreted **literally** instead of being used for glob-style wildcard matching.
[`Search:exact`]

Wildcards / Glob-Style Patterns

If the `Exact Search` option is disabled, search patterns are interpreted as glob-style patterns (similar to file name patterns in UNIX shells):

- `*` - Matches any sequence of characters including an empty string.
- `?` - Matches a single character.
- `[ABC]` - Matches any character in the given set.
- `[a-z]` matches any character in the range `a`-`z`.

Examples

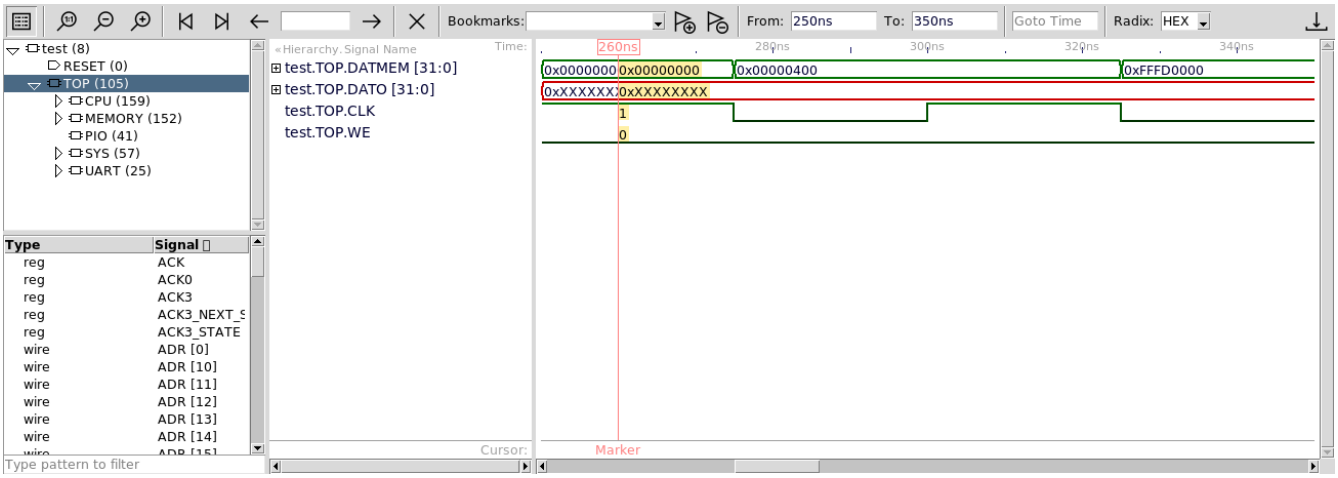
- `d*` matches any string starting with `d`: `d`, `d1`, `diode`, ...
- `md[0-9]` matches `md0`, `md1`, ..., `md9`.
- `m[A-Z]5` matches `mA5`, `mB5`, ..., `mZ5` in case sensitive search mode, it will also match `ma5`, `mb5`, ..., `Ma5`, `MA5`, ... in case insensitive search mode.

General Hints

- The tooltips provide you with additional valuable usage information.
- You can Drag & Drop objects from the result window into the Schem, Cone or Source windows.

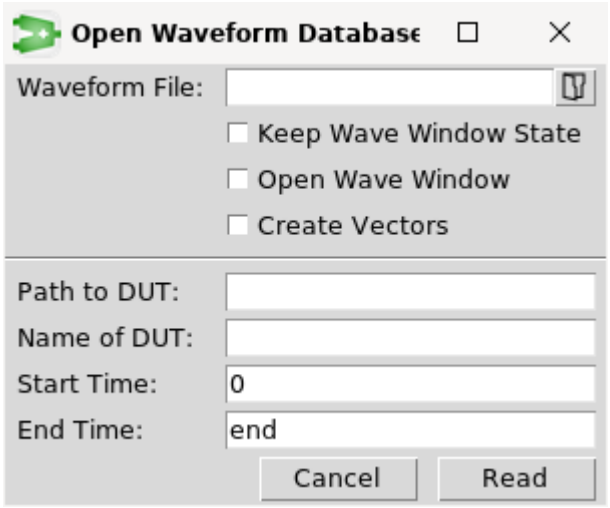
The Wave Window

The Wave window displays the waveform from a VCD file. The Waveform viewer can be opened with the icon in the toolbar or with **Window › Waveform** from the main menu.



Open Wave Files

Within the Wave window, the open a waveform file icon can be used to open a VCD or WDB file.



The Open Wave Database dialog has the following options:

- Waveform File: The waveform database file to open (`.vcd` or `.wdb`).
- Keep Wave Window State: When reloading the previous waveform database file, try to keep the current state of each Wave window (displayed signals, time range, etc.)
- Open Wave Window: Open the Wave window after reading the file.
- Create Vectors: Build groups based on scalar names with the same basename and a consecutive bit subscript.
- Top: Instantiation path to the Design Under Test (DUT).
- Create Hier: Convert a flat signal name containing the given hierarchy separator into a list of instance names and the signal name.

- Start Time: Ignore value changes before this time.
- End Time: Ignore value changes after this time.

NOTE	When loading a RTL design it is recommended to disable the Compact Schematic Option in the Read File dialog to get a better match between the design signals and the VCD signals.

Load Signals

The waveforms can be shown either by using Drag&Drop or by double clicking on the signal name in the signal selection window. Signals that could not be matched to the loaded design appear grayed out in the signal selection window. Below the signal selection window there is a search field. Entering a search term here filter all available signal names to the ones containing the search term (case insensitive). The waveform of the grayed out signals can be displayed but these signals cannot be dropped to other views in the GUI.

With this  icon the Waveform window can be cleared. Signals from an entire module can be displayed by dropping the module name to the Waveform window. The order of appearance of the signals can be changed with Drag&Drop.

Zoom Operation

To get an overview of the value changes of the loaded wave signals the zoom fit  icon can be used. To adjust the zoom factor use either the zoom out  or the zoom in  icon. Zooming can also be done with the mouse wheel plus the pressed `Ctrl` key. Using the stroke mouse button a time region of interest can also be selected.

Time Marker

Moving the mouse within the wave area shows the time of the cursor location in grey color. By clicking within the wave area the time marker (in light red color) is set. The values of all signals are written with yellow background at the Marker location. In the Schem, Cone and Source window these values are also shown. The displayed values are always updated when setting a new marker position. The radix of the shown values can be changed between hexadecimal, decimal, binary or octal. If the time marker is set to a transition, then either a rising or falling edge icon is displayed as the value to indicate the transition type.

The time marker and with it the value display can be removed by clicking into the red field in the timeline.

Jump to Previous and Next Value Changes

With the first time position  and  icons the cursor can be moved to the start or end of the timeline. The previous value change  and next value change  icons jump to the next or previous value change of the selected signal(s). Alternatively use the keys `p` or `n`. If no signal is selected, the cursor jumps to the previous/next value change of all loaded signals.

Time Cursor

Clicking into the narrow line at the bottom of the wave area, a cursor can be set. The cursor labels can be edited to represent their meaning in your design. These cursor marks can be removed again by clicking their label in the timeline.

Signal Grouping

Clicking on signal names with `Shift` or `Ctrl` keys pressed will select some signals. With the popup **Group › Create/Move Group** they can be copied/moved to a group. The Group name can be edited. The value of this new group is shown as if the group members would form a bus. The group members can be dragged to a custom order.

Groups can also be renamed or removed with the popup menu.

Time Fields

The **[From]** and **[To]** fields reflect the displayed time range. They can be edited with the suffixes `p`, `n`, `u`, `m`, `s` for pico-, nano-, micro-, milliseconds or seconds. If the time unit is omitted, the number is interpreted as the number of time steps.

To go to the beginning of the time range, just enter a zero.

To go to the end of the time range, just enter `end`.

In the field named **[Goto Time]** a time with the same units as before can be entered.

To use calculated time spans, click to a reference point to set the red marker. From here one can go a calculated time span forward or backward by pressing the `Ctrl` key while dragging the mouse in negative or positive direction.

Bookmarks

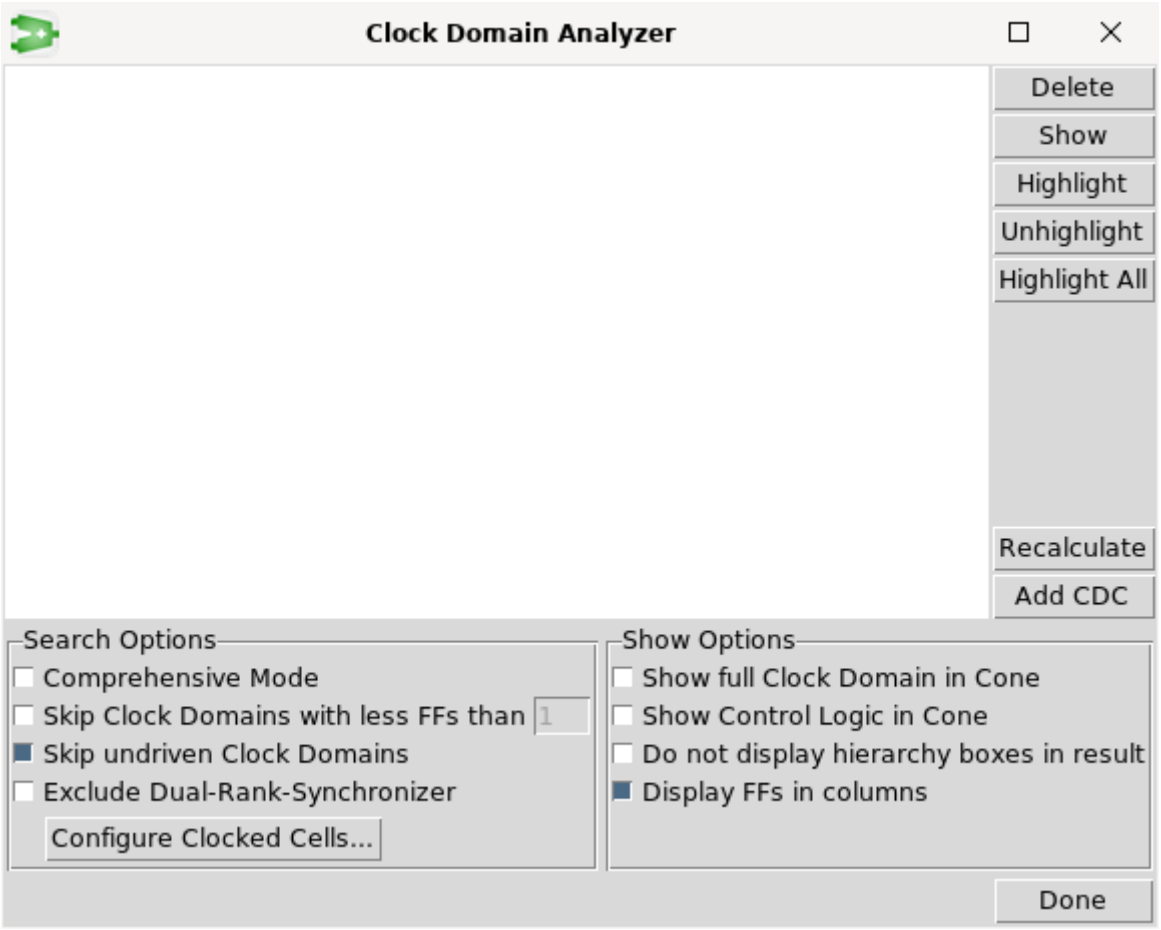
From any view a bookmark can be added with the add bookmark  icon. The proposed name can be changed by simply overwriting it. A bookmarked view can be recalled anytime or deleted with the delete bookmark  icon.

Working with the Cone Window

To debug a path that is loaded to the Cone window, the  button ("get all signals from the path displayed in the Cone window") can be used to transfer all the loaded signals to the Wave window. As before, the value representations can be seen in the Cone window when setting the time marker.

Clock Domain Analyzer

The `Clock Domain Analyzer` can extract clock domain structures and search for circuits between different clock domains (Clock Domains Crossings, CDC). This function can be invoked from the **Tools › Clock Domain Analyzer** menu.



Prerequisites

For usable results it is necessary to provide as much information as possible. Especially the 'pin directions' of all cells should be known to the database.

If the RTL parser was used to read the design data then no additional steps are needed as the pin directions are committed by the parser.

If a netlist parser was used to read the design data then the pin directions may be unknown. The following suggestions show how the pin directions can be created or restored:

- Use VHDL for netlists: As all used components must be declared before use, all pin directions are automatically known.
- Provide an SDF file: The pin directions can be restored by reading the associated SDF file.
- Provide 'Technology-Libraries' in Liberty format.

Additionally it's necessary that you specify all clocked cells and the names of the clock ports in the Configure Clocked Cells dialog.

Clock Domain Browser

Configuration options for the clock domain and domain crossing search are:

- Comprehensive Mode: Collects all possible clock domain sources for all FFs rather than trying to estimate the clock domain sources. As this will slow down the process and may create complex

results it should be only enabled if clock domain violations are expected.
[ClockTree:Comprehensive:ok]

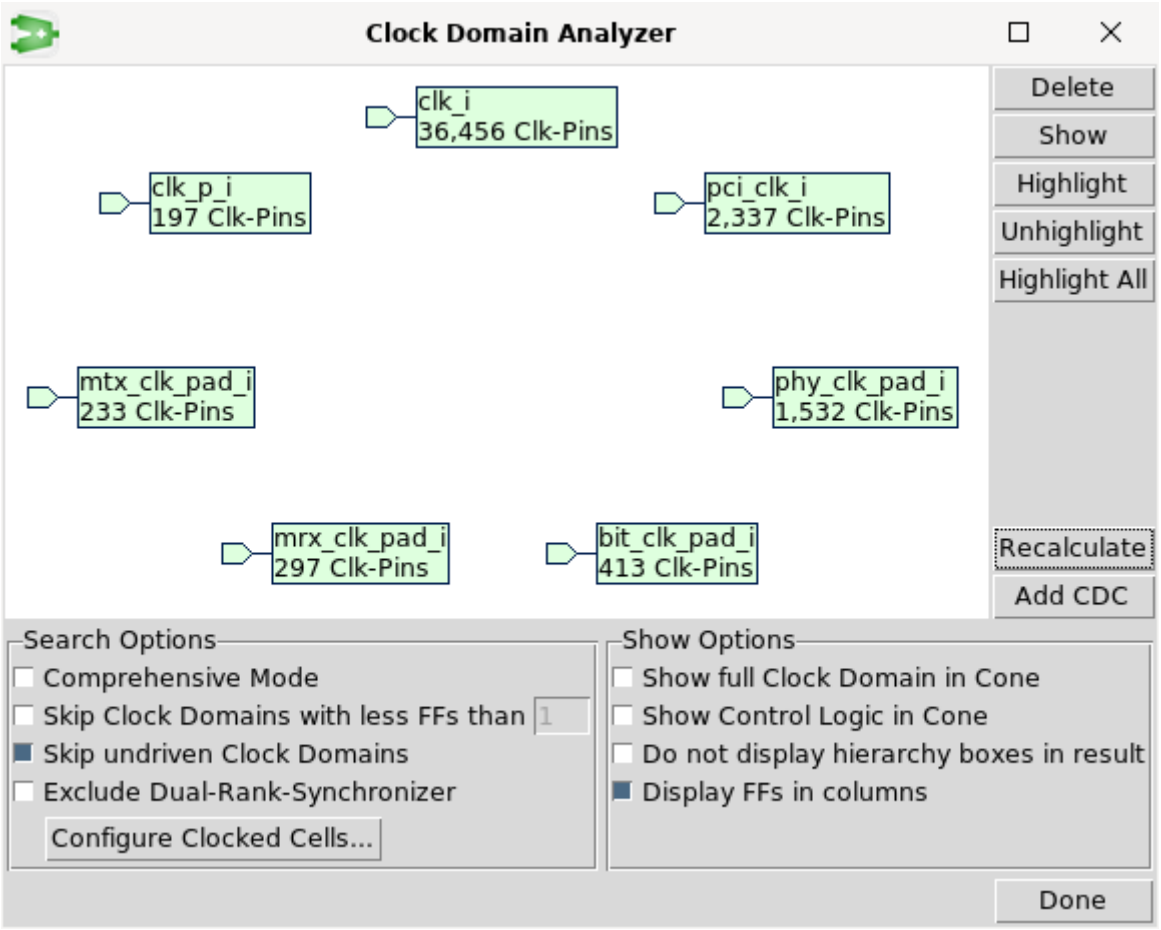
- Skip Clock Domains with less FFs than...: This option can be used to ignore small clock domains (e.g. asynchronous Flip-Flops).
[ClockTree:SkipLess:ok], [ClockTree:TargetCnt]
- Skip undriven Clock Domains: Normally, the clock domain search will stop with an error message if it finds any undriven clock signal. Use this option to ignore such errors.
[ClockTree:SkipUndriven:ok]
- Exclude Dual-Rank-Synchronizer: Exclude crossings which are synchronized by a Dual-Rank synchronizer.
[ClockTree:exclDrs:ok]

Configuration options for displaying the results in the Cone window are:

- Show full Clock Domain in Cone: Displays the full clock domain rather than creating the default (strongly recommended) reduced view.
[ClockTree:ShowFullTree:ok]
- Show Control Logic in Cone: Also load control logic into the cone, e.g. logic connected to the 'select', 'set', 'reset', 'enable' input ports of cells.
[ClockTree:ShowControlLogic:ok]
- Do not display hierarchy boxes in result: The result is displayed in the Cone window without hierarchy boxes. This option is identically equal to the "turn off the visibility of hierarchy boxes" option in the Cone window.
- Display FFs in columns: All Flip-Flops of the same clock domain will be shown in one column.
[ClockTree:colgroup:ok]

The clock domain search is automatically started when the dialog is opened the first time. It can be invoked again at any time by pressing the **[Recalculate]** button.

The result will be shown in a graphical way. Each clock domain is displayed as a node that is labeled with the domain's clock source and the number of clock pins driven by this source. A (+ <NUMBER>) extension to the clock source label denotes that the clock domain has additional <NUMBER> clock sources. The following screenshot shows a clock domain browser with a finished clock domain extraction.



The following options are available for found clock domains:

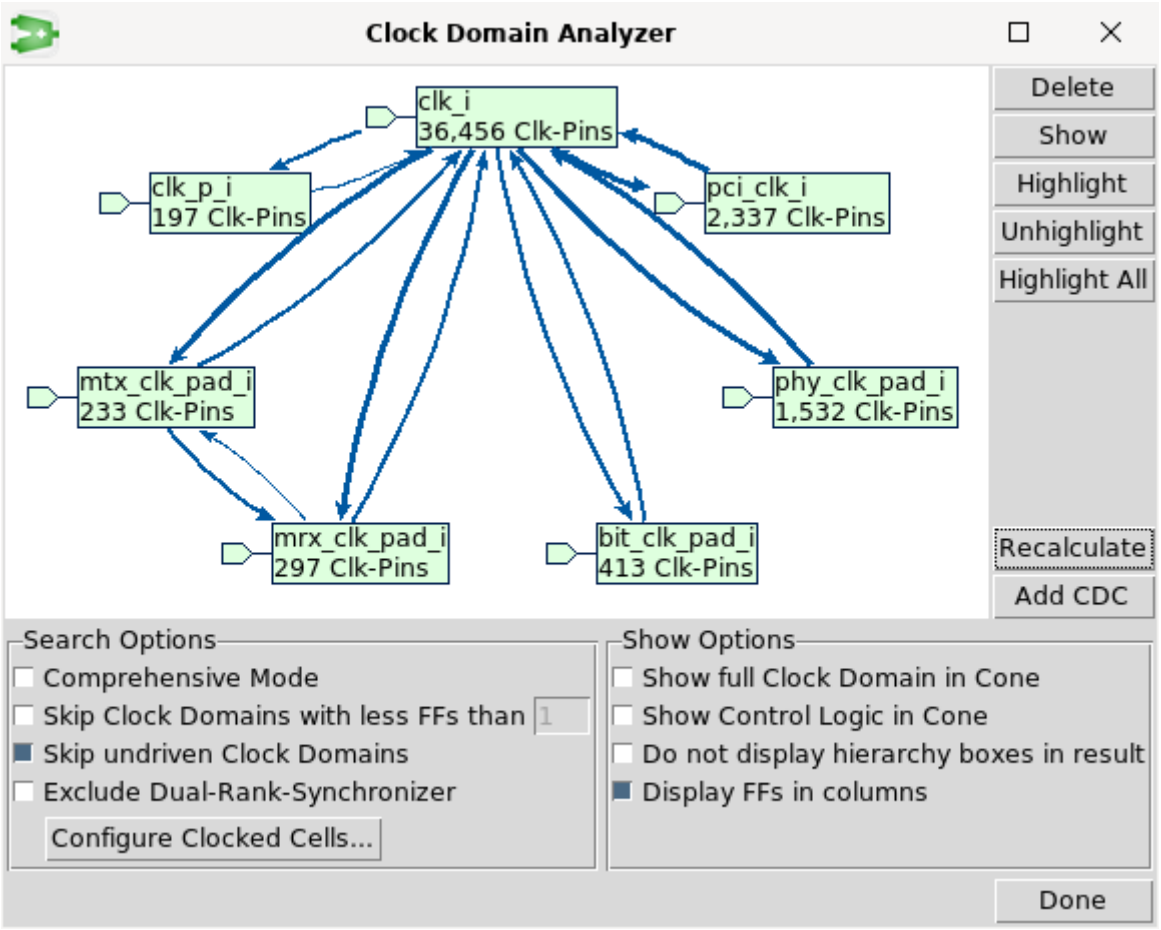
- **[Delete]**: Removes the selected clock domain or crossing from the result view.
- **[Show]**: Shows the selected clock domain or crossing in the Cone window. This action can also be performed by double-clicking on the corresponding item in the graphical view.
- **[Highlight]**: Highlights the selected clock domain or crossing with the actual highlight color.
- **[Unhighlight]**: Remove the highlight information from the selected clock domain or crossing.
- **[Highlight All]**: Highlight all clock domains with a different highlight color.

Clock Domain Crossing Browser

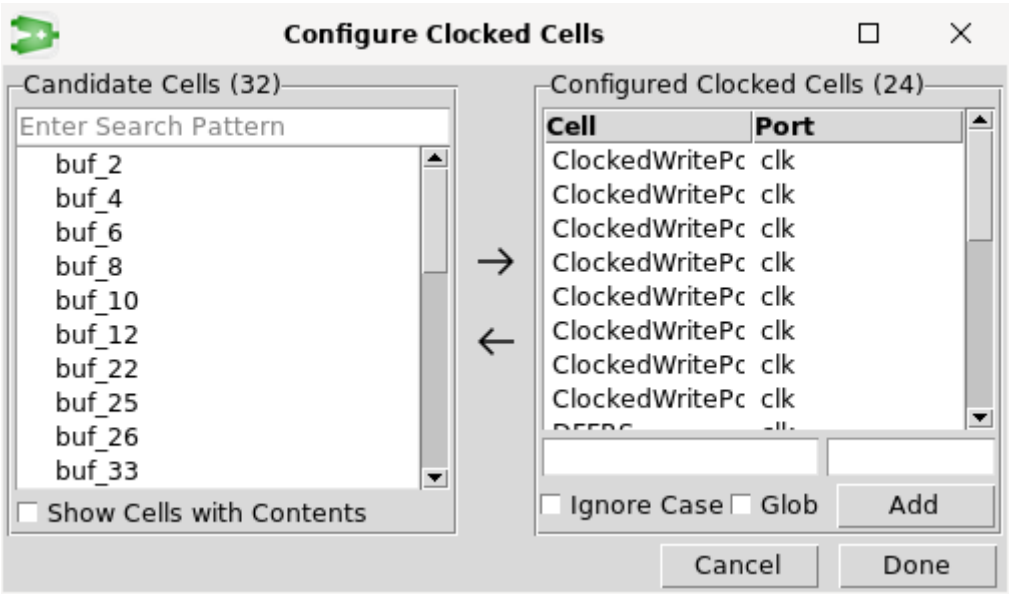
All selected clock domains are used as source and targets. If no clock domain is selected, then all domains are source and target.

The clock domain crossing search can be invoked by pressing the **[Add CDC]** button. The result will be shown as splines in the graphical view.

The following screenshot shows a CDC browser with a finished CDC search.



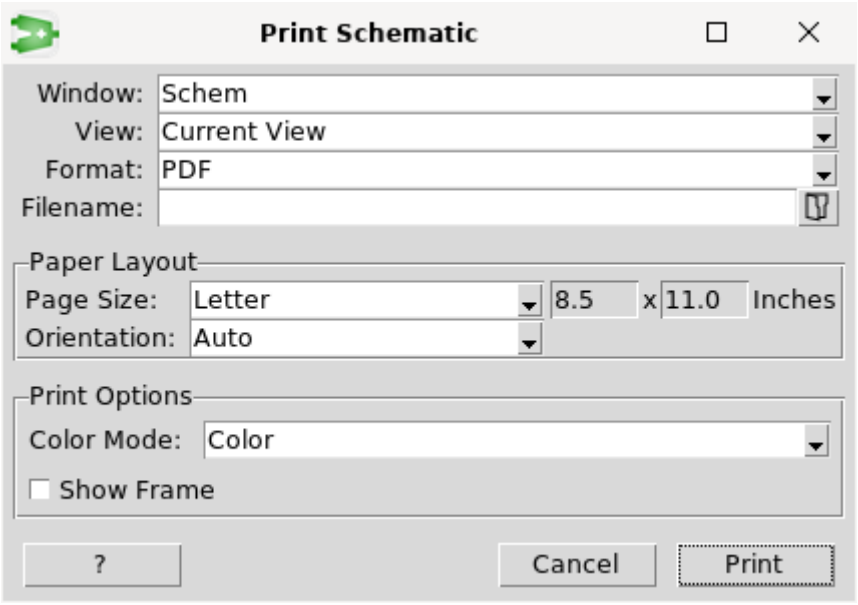
Configure Clocked Cells



This dialog allows you to specify clocked cells and one or more clock port names for each cell.

The cell names are used for the Cone Extraction target Clocked Cells. The port names are used in the Clock Domain Analyzer.

The Print Schematic Dialog



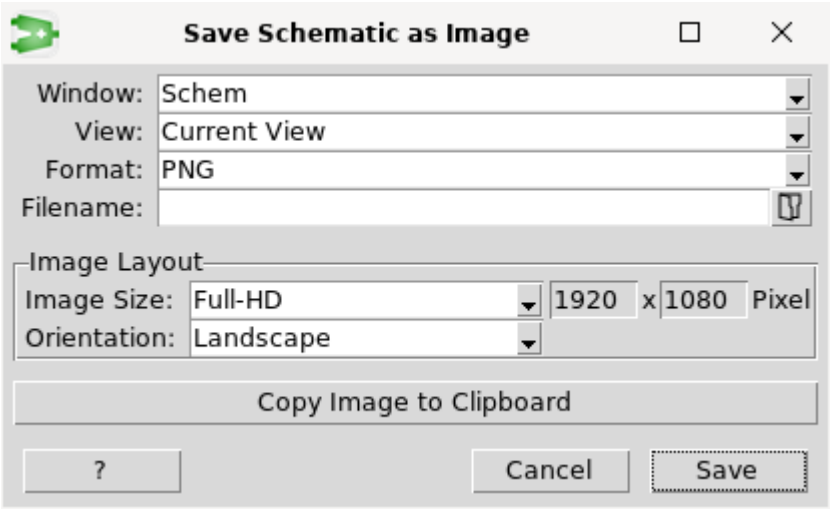
The Print dialog is available through the icon, **File › Print** menu or `Ctrl` + `P` and provides the possibility to change certain print parameters:

- Select one of the following supported window types to print (in addition to the default windows of each class, all additionally created windows will be listed as well):
  [`print:what`]
  - Schem - Prints the content of the Schem window.
  - Cone - Prints the content of the Cone window.
  - Design Hierarchy - Prints all modules in the database; when output is set to Postscript File or PDF File then you specify a directory name below; for each module, a Postscript/PDF file will be created in the selected directory.
- Select the view to print:
  [`print:view`]
  - Current View - Print the current view of the page. The area to print is limited by the current visible area (window borders).
  - All Pages - Print all schematic pages. If you redirect printing to a file, only one file will be created.
  - Current Page - Print the schematic as displayed on the screen after a zoom fullfit. The area to print is defined by the size of the schematic drawing, extra space around the drawing is not included.
- Output format:
  [`print:dest`]
  - Default Printer - Uses the default printer. The print command (`lpr` by default) might need to be adjusted to send the Postscript data to a printer.
    On Windows the installed default printer will be used. [`print:command`]
  - Postscript - Creates a file in Postscript format; enter the file name below.

    - PDF - Creates a PDF file; enter the file name below.
- Filename - Specify the output filename, directory or command.
- Page Size: Select the paper size.
  [print:paper]
- Orientation: Select the orientation: Landscape, Portrait or Auto (rotate whenever the schematic's width/height ratio is < 1).
  [print:orient]
- Color Mode:
  [print:colormode]
    - Mono prints in black & white
    - Color prints in original colors
    - Inverted color prints in inverted colors (white gets black and vice versa).
- Show Frame: If turned on, a page frame will be drawn around the schematic. [print:showFrame]

NOTE On Windows the Dialog button can be used to open the native Print dialog.

The Save Schematic as Image Dialog



The Save Schematic as Image dialog is available through the icon, **File › Save Schematic as Image** menu and provides the possibility to change certain parameters:

- Select one of the following supported window types to save the schematic as an image (in addition to the default windows, all additionally created windows will be listed as well):
  [photo:what]
    - Schem - Save the content of the Schem window.
    - Cone - Save the content of the Cone window.
    - Design Hierarchy - Save all modules in the database. Please specify a directory name below instead of a filename. For each module, an image will be created with a filename matching the module's name.
- Select the view to save:

[photo:view]

- Current View - Save only the visible part of the schematic.
- All Pages - Save all schematic pages. Each schematic page will result in an image file. The page number is appended to the filename.
- Current Page - Save only the current schematic page.

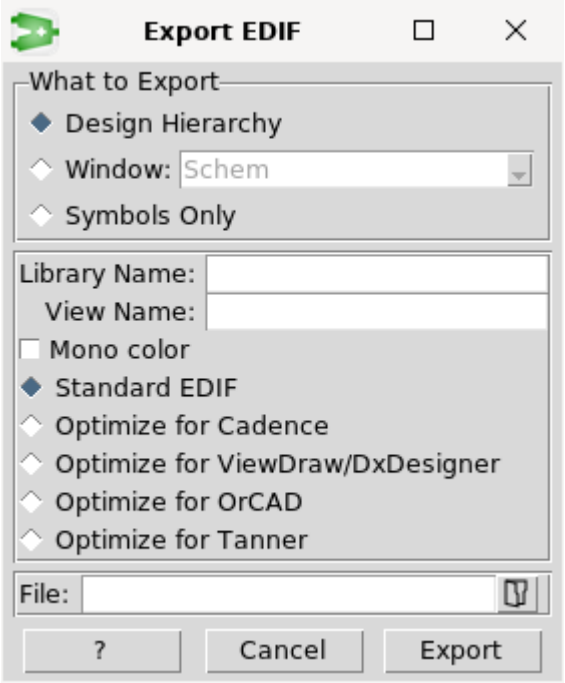
- Image Type: Select the image type of the generated file.
  [photo:dest]
- Filename: Enter the file name of the generated file.
- Image Size: Select the image size. If set to custom then you can specify the width and height of the image next to the option menu.
  [photo:paper]
- Orientation: Select the orientation of the image.
  [photo:orient]

Export Schematic

This chapter describes the export schematic dialogs and how to use them to export schematics to other tools.

Export EDIF 2.0.0 Schematics

This feature is no longer maintained, but is still available as is. To activate the **File › Export EDIF** menu entry, the RTLvision PRO tool needs to be invoked with the `-enableExportEDIF` command line option.



The Export EDIF dialog window is available through the **File › Export EDIF** menu. The EDIF export can be done in two different ways:

- Design Hierarchy - exports the whole design hierarchy.
- Window - exports either all pages of the current module in the Schem window or the contents of the Cone window.

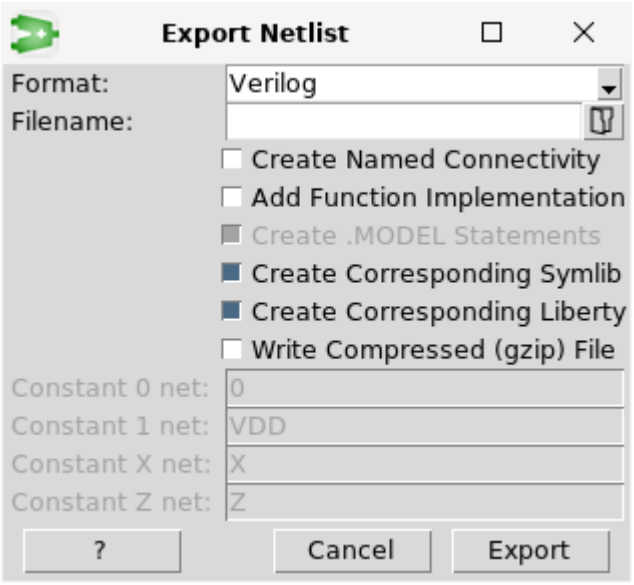
Optionally, the EDIF Library Name (defaults to the filename) and the EDIF Cell View Name (defaults to "") can be changed.

The following switches can additionally be turned on or off:

- Mono color - use black on white background
- Standard EDIF - follow the EDIF 2.0.0 standard.
- Optimize for Cadence - tweak EDIF to fit Cadence Composer.
- Optimize for ViewDraw/DxDesigner - tweak EDIF to fit Mentor DxDesigner.
- Optimize for OrCAD - tweak EDIF to fit OrCAD.
- Optimize for Tanner - tweak EDIF to fit Tanner (S-Edit).

Export Netlist

This chapter describes the Export Netlist dialog and how to use it to generate netlists in different formats.



The Export Netlist dialog window is available through the **Tools › Export Netlist** menu.

The following output formats are supported:

- Spice - Generic Spice.
- Verilog - Structural Verilog.
- Tcl - RTLvision PRO specific Tcl netlist.
- VHDL - VHDL netlist.

The following options are available:

- Create Named Connectivity (Verilog format only) - Write the Verilog file using named connectivity.
- Add Function Implementation (Verilog and VHDL) - Add a function implementation of selected primitives.
- Create .MODEL Statements (Spice format only) - Create the corresponding `.MODEL` or `.subckt` statements for primitive devices or macro models.
- Create Corresponding Symlib - Create a symbol library file containing symbol shape information.
- Create Corresponding Liberty - Create a Liberty file containing cell and clock information.
- Write Compressed (gzip) File - Create a gzip compressed output file.

If the output format is Spice then constant values are modeled as nodes using the names listed in this dialog.

Open/Save a Binfile

The current database contents can be stored into and retrieved from a file in binary format (the so called 'Binfile'), sometimes referred to as ZDB-file (it uses the **.zdb** filename extension by default). The binary file is a way to load big circuits very fast.

The batch-tool **rtl2zdb** can be used to create a Binfile from Verilog and VHDL files as a background task (with no GUI).

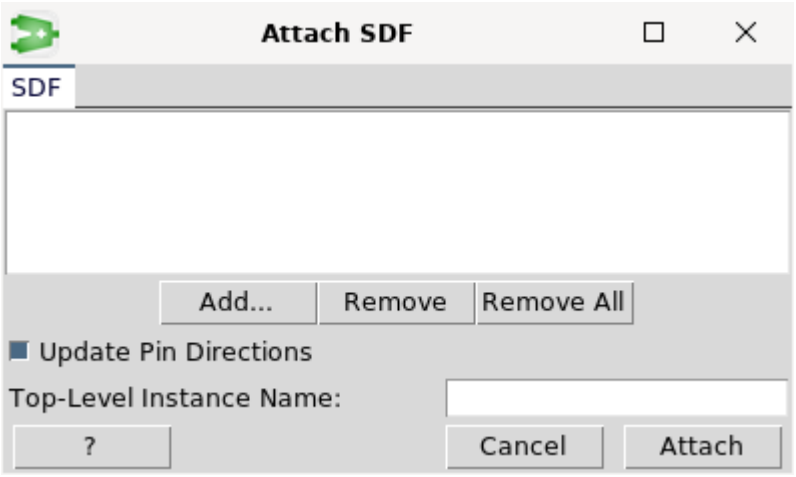
The batch-tools **verilog2zdb** and **edif2zdb** can be used to create a Binfile from Verilog or EDIF netlists as a background task (with no GUI).

Use the icon or the **File › Open › ZDB Binfile** menu to open a binfile in RTLvision PRO.

The **File › Save ZDB Binfile** menu can be used to save a binfile from RTLvision PRO.

SDF Files

The Standard Delay Format (SDF) files can be provided by selecting them in the Attach SDF dialog, which is available through the **File › Open › Attach SDF** menu:

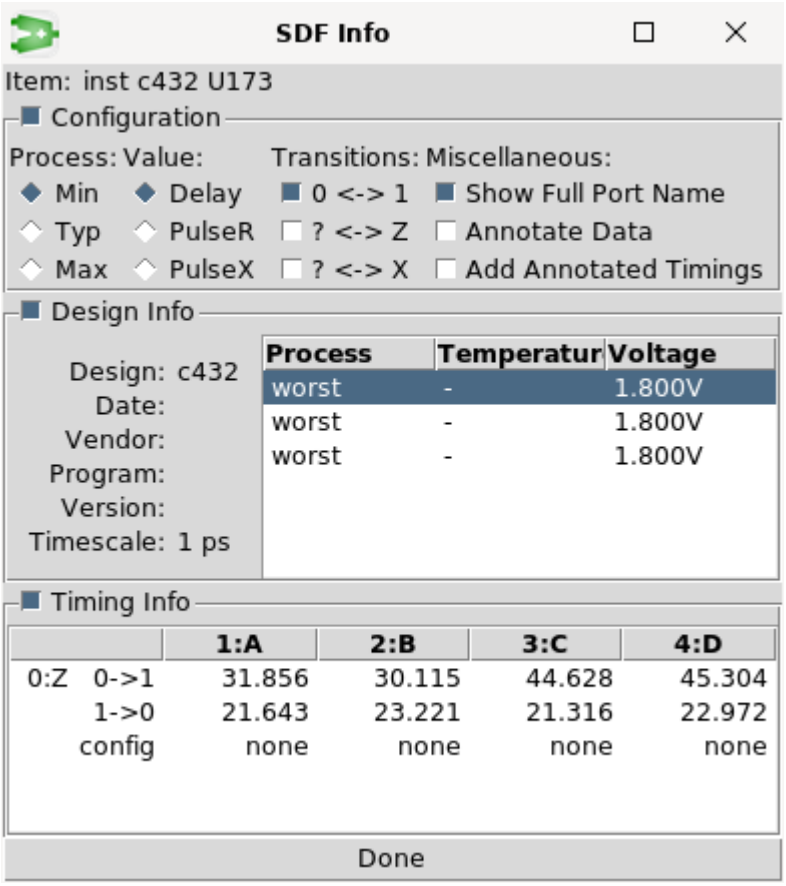


- Use the **[Add]**, **[Remove]**, **[Remove All]** buttons to edit the list of SDF files to be linked to the Verilog design.
  [SDF:FL].
- If you enable the `Update Pin Directions` option, pin directions are reconstructed from the SDF data. This can be used e.g. to create valid pin directions if no cell-library is given. All cells must have valid pin directions to show proper timing.
  [Sdf:pindirupd]
- `Top-Level Instance Name` lets you choose the name of the top-level instance.
  [Sdf:topInstName]

SDF Info

The SDF Info dialog can be opened with the Popup invoked on an object that contains SDF information. Therefore one or more SDF files need to be provided. This dialog shows timing information for selected items in the Schematic or Cone window.

The SDF Info dialog is divided in three sections: Configuration, Design Info, and Timing Info. Each section can be disabled or enabled by setting the checkbox next to the label of each part.



Configuration

- Process, Value, and Transition - Select which process, value and transitions shall be shown at Timing Info section.
- Show Full Port Name - The full port name will be shown in the Timing Info section. Disable this option to shorten long names e.g. to make configuration expressions more readable.

- Annotate Data - If enabled, each timing value selected in Timing Info will be immediately annotated to the according pin/port in the schematic. Three timing values can be stored at each pin/port which can be switched by selecting a different process.
- Add Annotated Timings - If the source of the path of the actual timing already has an annotated timing, the sum of both timings is shown. To tell the difference between real timing and accumulated timing, a + is shown for accumulated timings.

Design Info

Shows design information provided by the SDF file(s) linked to the actual design.

Timing Info

Shows the timings for the currently selected net, pin, port or inst.

If a pin or inst is selected, the upper legend of the table shows the input-pins of the current instance and on the left side the output-pins. Bidirectional pins may be found in both legends.

If a net or port is selected, the upper legend shows the output-pins, the left legend the input-pins.

For each input/output pair, all transitions selected in Configuration are shown as a list. Values shown in 'green' are explicitly given values from the SDF file. Values shown in 'grey' are implicitly given. If no value is given, a - is shown. An optional configuration for each transition may be shown at the bottom of each transition list. The configuration can be changed by clicking on the configuration entry.

Report Instance Count

This function can be invoked with the  icon, from the **Tools › Report Instance Count** menu entry (for the top module only) or from the context menu of the Tree window (for the selected module).

The Report Instance Count feature counts the number of instantiated modules and primitives in the hierarchy tree below the current module and displays it in a separate dialog window.

Preferences

The Preferences dialog can be opened with the  icon or from the **View › Preferences** menu entry.

The Preferences dialog can be used to configure the GUI. All settings can be saved to (and read from) a Workspace file.

You can use the environment variable `GV_GLOBAL_WORKSPACE` to define a global workspace which is read at startup.

The Preferences Dialog

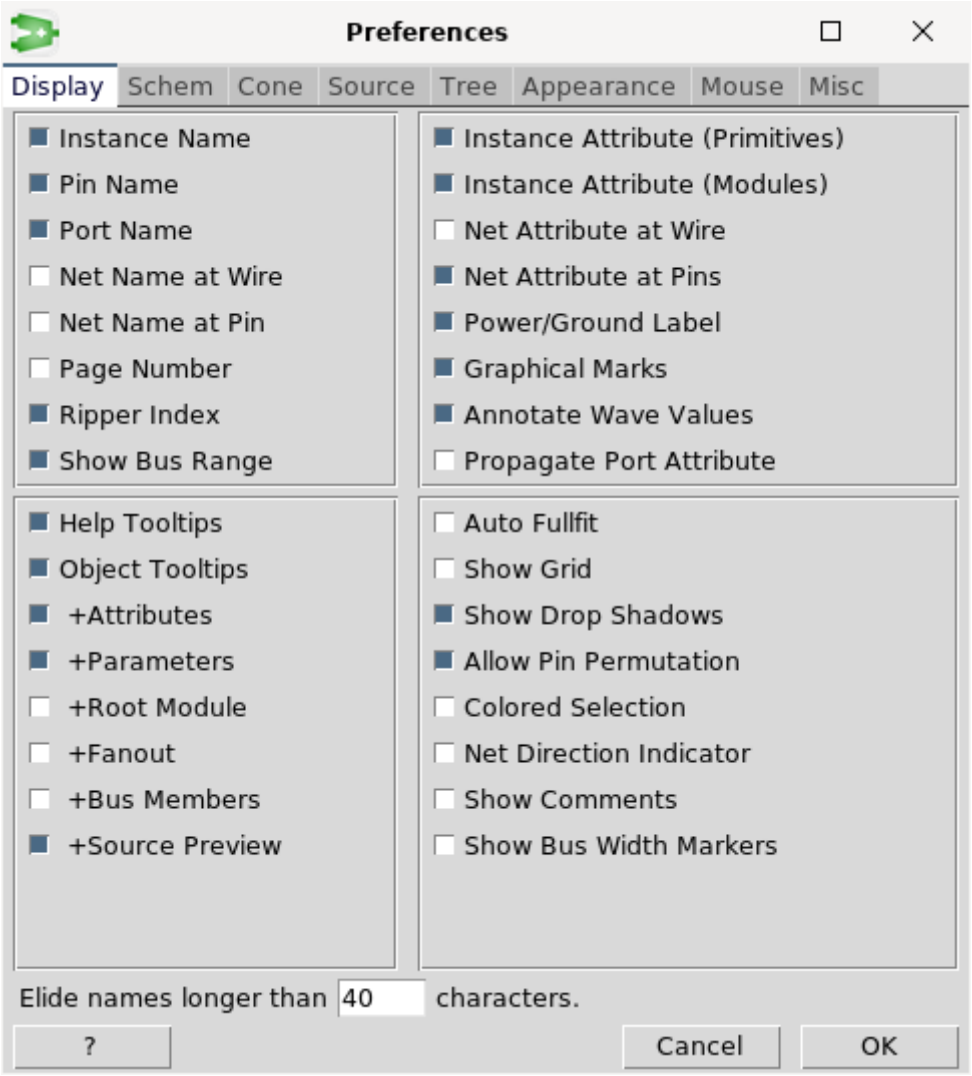
The buttons in the lower part of the Preferences dialog have the usual meaning:

- **[OK]** - Close dialog and commit changes.
- **[Cancel]** - Close dialog and discard changes.
- **[?]** - Open the documentation.

The dialog is split into the following sections (tabs): Display, Schem, Cone, Source, Tree, Appearance, Mouse, and Misc.

The pictures below show the 'default' settings in the different dialog tabs. The tooltips might give you useful information about specific items when running the tool. To change one of the settings using the GUI API you can use the `gui settings set` command with the name shown in brackets after each option.

Display Properties



All preferences in this Display tab apply to the Schematic and Cone windows of all Pane windows simultaneously.

- Instance Name - Toggle display of instance names in schematic.

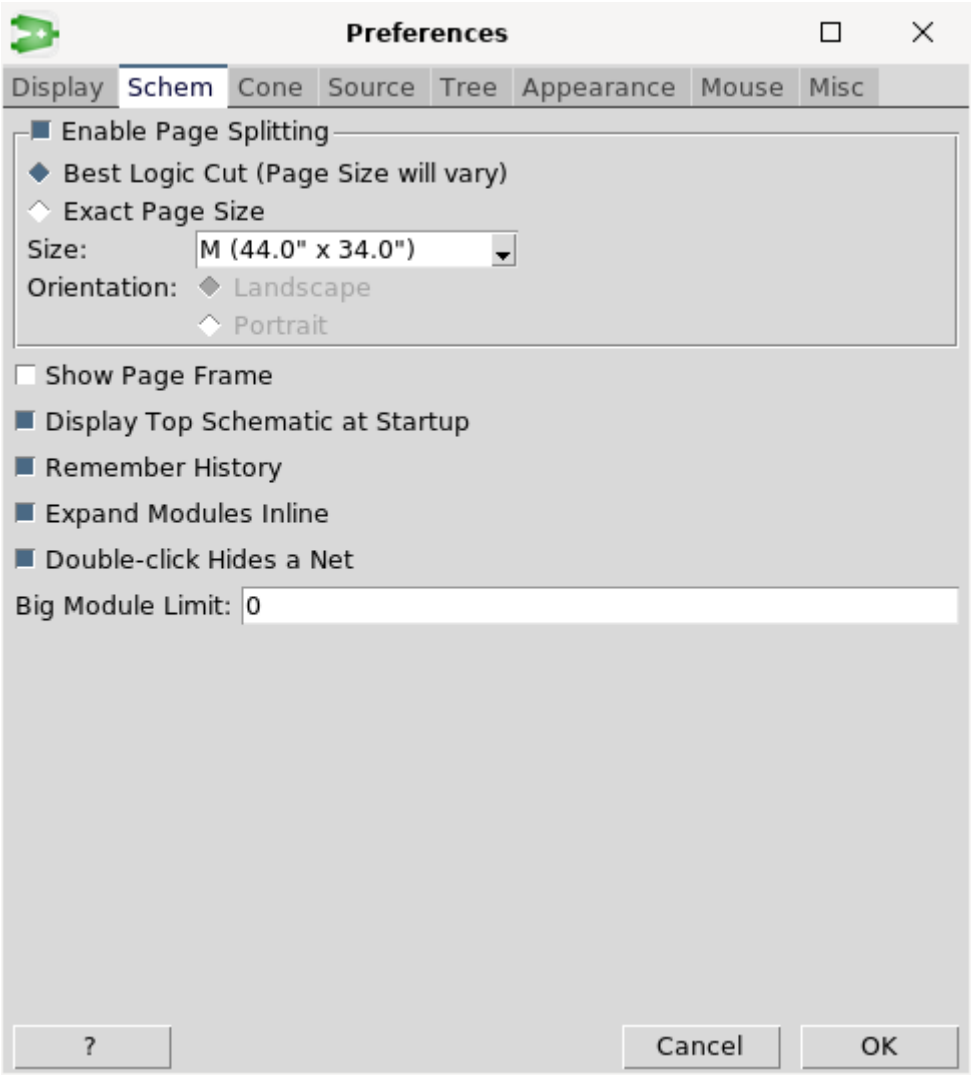
[nlv:showinstname]

- Pin Name - Toggle display of pin names in schematic.
  [nlv:showpinname]
- Port Name - Toggle display of port names in schematic.
  [nlv:showportname]
- Net Name at Wire - Toggle display of net names at wires in the schematic.
  [nlv:shownetname]
- Net Name at Pin - Toggle display of net names at pins in the schematic.
  [nlv:shownetnamepin]
- Page Number - Toggle display of page numbers at off page connectors.
  [nlv:showpagenumbers]
- Ripper Index - Toggle display of ripper index at netBus ripper.
  [nlv:showripindex]
- Show Bus Range - Toggle display of the bus range at pinBus and portBus objects.
  [showBusRange]
- Instance Attribute (Primitives) - Toggle display of primitive instance attributes in schematic.
  [nlv:showattribute]
- Instance Attribute (Modules) - Toggle display of module instance attributes in schematic.
  [nlv:showcellname]
- Net Attribute at Wire - Toggle display of net attributes at the wire.
  [nlv:netattrwire]
- Net Attribute at Pins - Toggle display of net attributes at the pins.
  [nlv:netattrpin]
- Power/Ground Label - Toggle display of labels of power and ground nets in schematic.
  [nlv:showpgtype]
- Graphical Marks - Toggle display of graphical marks.
  [nlv:showmarks]
- Annotate Wave Values - Toggle annotation of logic values from the Wave window in the Schematic and Cone window.
  [nlv:showval]
- Propagate Port Attribute - Propagate port attribute to the corresponding pin.
  [nlv:portAttrAtPin]
- Help Tooltips - Toggle display of help tooltips in the GUI.
  [helpTooltips]
- Object Tooltips - Toggle display of database object tooltips in the GUI.
  [tooltips]
- +Attributes - Toggle display of attributes in the tooltips in schematic; this only applies to database objects that have attributes set.
  [tooltipsWithAttrs]
- +Parameters - Toggle display of instance/cell parameters in the tooltips.

[tooltipsWithParameters]

- +Root Module - Toggle display of the root module in the tooltips in schematic.
  [tooltipsWithRoot]
- +Fanout - Toggle display of fanout in the tooltips in schematic.
  [tooltipsWithFanout]
- +Bus Members - Display bus members in the tooltips for pinBus and netBus objects.
  [tooltipsWithMembers]
- +Source Preview - Display source code preview in tooltip for files smaller than 256MB.
  [tooltipsWithSource], [tooltipsWithSourceFilesizeLimit], [tooltipsWithSourceLineWidthLimit]
- Auto Fullfit - Automatically zoom new modules to fullfit.
  [nlv:autoFullfit]
- Show Grid - Display a dotted grid (visible only with an appropriate zoom factor).
  [nlv:showGrid]
- Show Drop Shadows - Toggle drop shadow effect on instances.
  [nlv:dropshadow]
- Allow Pin Permutation - Allow the permutation of pins to reduce wire crossings. If a custom symbol shape is used then the symbol needs to allow pin permutation.
  [nlv:pinpermute]
- Colored Selection - Use the goto color for additional selection feedback.
  [nlv:coloredSelection]
- Net Direction Indicator - Decorate net objects with additional logical direction indicator icons.
  [nlv:netDirIndicator]
- Show Comments - Show graphical object comments.
  [nlv:showComments]
- Show Bus Width Markers - Add graphical marks to netBuses that show the bus widths.
  [nlv:showBusMarkers]
- Elide names longer than X characters - Labels longer than a given limit are displayed clipped with three leading dots. A value of 0 disables text eliding.
  [nlv:elidetext]

Schematic Properties

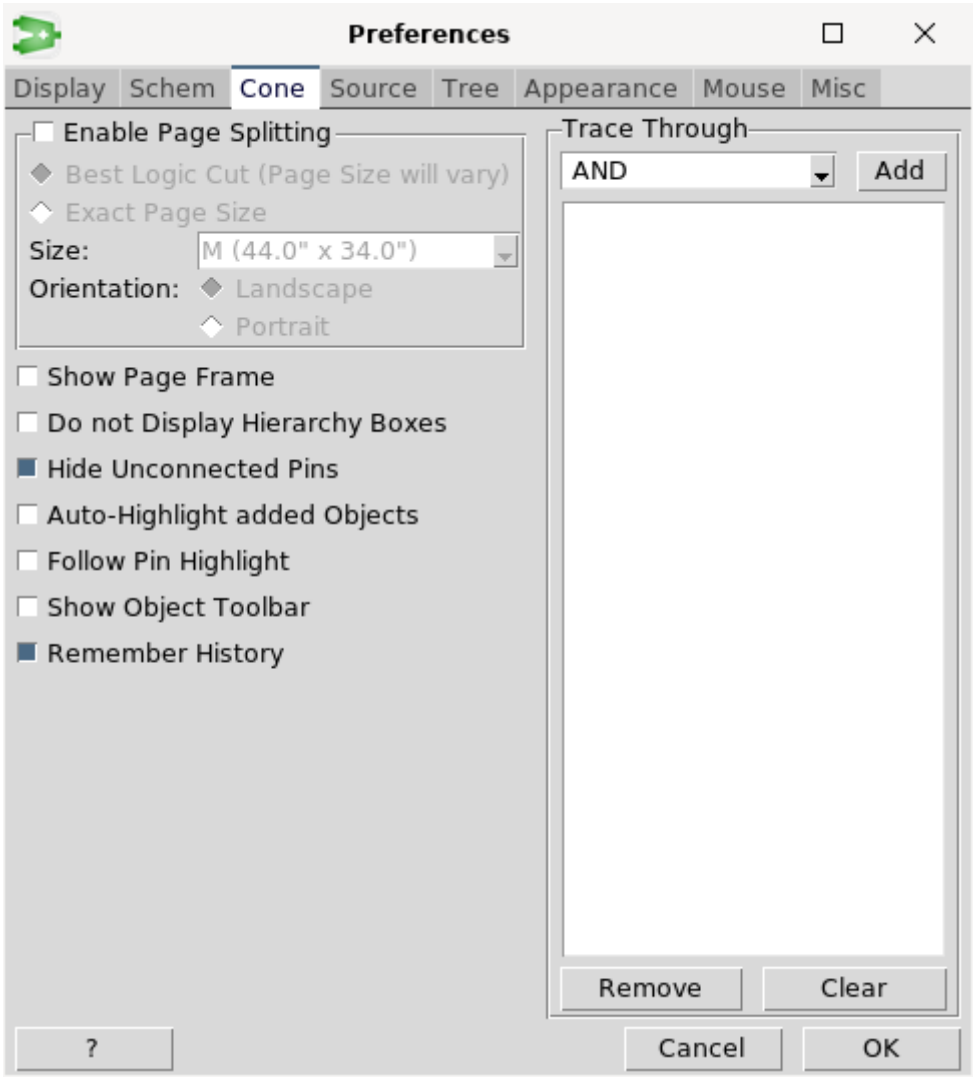


All preferences in this Schem tab apply to the Schem window of all Pane windows simultaneously.

- Enable Page Splitting - Activate page splitting for the Schem window.
  [schem:splitpage]
- Best Logic Cut (Page Size will vary) - Use a built-in automatism to determine the actual page size of the schematic; use the Size listbox to change the preferred size factor.
  [schem:fitpage]
- Exact Page Size - Use a fixed page size for schematic generation; use the Size listbox to change the preferred sheet size; there will be a visible frame surrounding the schematic in this mode.
  [schem:fitpage]
- Size - Select the page size.
  [schem:logicSize], [schem:exactSize],
  [schem:sheetwidth], [schem:sheetheight]
- Orientation - Toggle the orientation of the schematic; choose between Landscape (width > height) and Portrait (height > width) mode.
  [schem:orientation]
- Display Top Schematic at Startup - This option controls whether the top-level schematic is automatically displayed in the default Schem window.
  [startTopSchematic]

- Remember History - Enable the forward and backward history buttons.
  [schem:history]
- Expand Modules Inline - Expand the contents of modules inline in the current module using the (+) button in the top left corner.
  [schem:expandInline]
- Double-click hides a Net - A double-click on a net toggles the hide state of the net.
  [schem:doubleNetHide]
- Big Module Limit - Before the schematic of a module is generated the number of objects the module contains is compared with the number specified here. If the number of components in the module is less than this limit, the schematic is generated without user interaction.
  Increase this number for fast computers and decrease it for slower machines.
  This value is also used as a limit if you load heavy connected Nets/Signals to the Cone window using **Connectivity Browser › Signal** from the context menu.
  A value of `0` disables the limit. The default value is 60,000.
  [bigModuleLimit]

Cone Properties



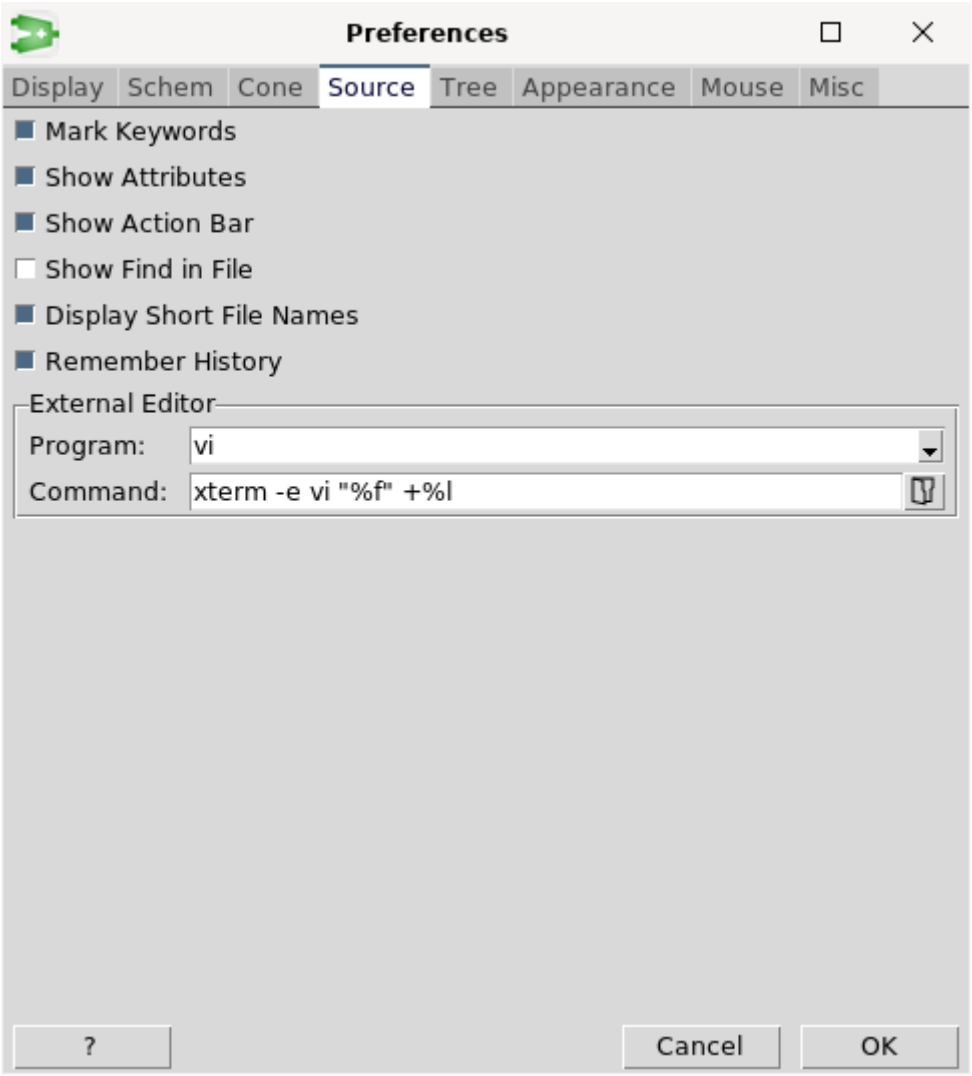
All preferences in this Cone tab apply to the Cone window of all Pane windows simultaneously.

- Enable Page Splitting - Activate page splitting for the Cone window.

[cone:splitpage]

- Best Logic Cut (Page Size will vary) - Use a built-in automatism to determine the actual page size of the schematic; use the Size listbox to change the preferred size factor.
  [cone:fitpage]
- Exact Page Size - Use a fixed page size for schematic generation; use the Size listbox to change the preferred sheet size; there will be a visible frame surrounding the schematic in this mode.
  [cone:fitpage]
- Size - Select the page size.
  [cone:logicSize], [cone:exactSize],
  [cone:sheetwidth], [cone:sheetheight]
- Orientation - Toggle the orientation of the schematic; choose between Landscape (width > height) and Portrait (height > width) mode.
  [cone:orientation]
- Do not Display Hierarchy Boxes - If this option is enabled then the hierarchy boxes are not displayed in the Cone window. You get a flat view of your design. But the path is still visible in the instance name.
  [cone:hierfilter]
- Hide Unconnected Pins - Unconnected instance pins are hidden automatically in the Cone window (until they are connected to a loaded, i.e. visible, net). You may also use a double-click with the left mouse button on the border of an instance in Cone window to quickly toggle the hide mode.
  [cone:autohide]
- Auto-Highlight added Objects - Automatically highlight added objects.
  [cone:autohighlight]
- Follow Pin Highlight - A double click on a highlighted pin or port object will propagate the highlight color to the added object(s).
  [cone:followPinHighlight]
- Show Object Toolbar - Toggle the visibility of the object toolbar to access the delete function at selected instance and port objects.
  [cone:objtoolbar]
- Remember History - Enable the forward and backward history buttons.
  [cone:history]
- Max Path Length to Power/Ground - When double-click extends the schematic by a device, then paths to power and ground are searched and also added automatically. This search can be limited to the given number of added instances.
  [cone:devicePathMax]
- Trace Through - A double click in the Cone window will go through all cells specified here as trace through cells. To enter trace through cells either select them from the drop down menu or enter a glob style pattern to add all cells matching the given pattern.

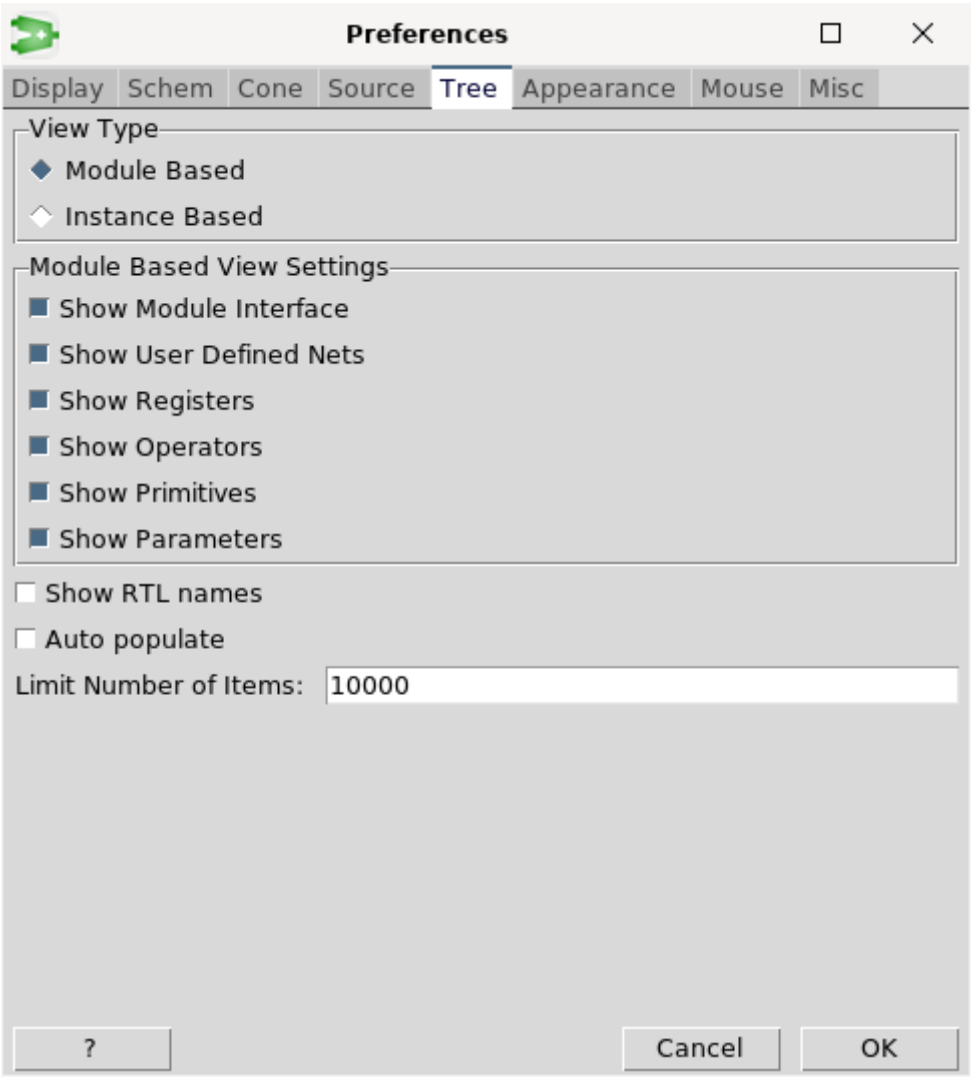
Source Properties



- Mark Keywords - Syntax highlights for known keywords.
  [`Syntaxhighlight`]
- Show Attributes - Display attribute values of net and pin objects.
  [`ShowAttributes`]
- Show Action Bar - Display action bar when clicking on objects in the Source window.
  [`ShowActionBar`]
- Show Find in File - Always show the Search for toolbar.
  [`source:showFindInFile`]
- Display Short File Names - Display short file names in the Source window's file selection list instead of full paths
  [`source:displayShortFnames`]
- Remember History - Enable navigation history in the Source window.
  [`source:history`]
- External Editor - Choose your favorite external editor (must be installed on the machine) for viewing and editing source files from Source window.
  The following formatting arguments are supported:
  - `%f` - current file name in Source window
  - `%l` - current line number in file displayed in Source window

[source:editor], [source:editorCmd]

Tree Properties



The Tree widget supports two different view types:

- Module Based - Show a folded tree sorted and grouped by modules.
  [Tree:ModuleView]
- Instance Based - Sort the tree by instances.
  [Tree:ModuleView]

In the module based view additional information can be displayed in the Tree:

- Show Module Interface - Add a node to the tree to show the I/O ports of a hierarchical module.
  [Tree:Show:PORT]
- Show User Defined Nets - Add a node to the tree to show all nets and buses not automatically generated by a parser.
  [Tree:Show:NET]
- Show Registers - Add a node to the tree to show all clocked elements of a module.
  [Tree:Show:REG]
- Show Operators - Add a node to the tree to show hierarchical blocks with a known function.

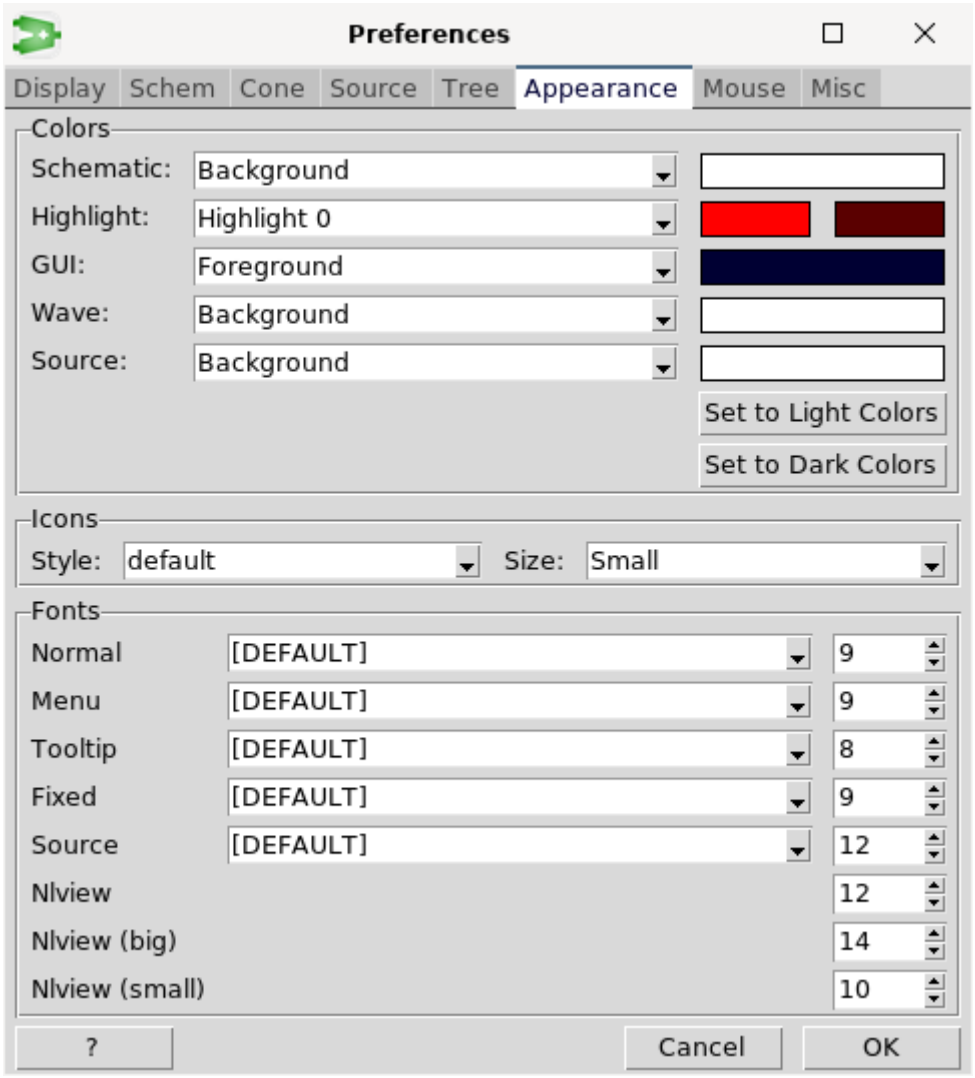
[Tree:Show:OPER]

- Show Primitives - Add a node to the tree to show all primitives of a module.
  [Tree:Show:PRIM]
- Show Parameters - Add a node to the tree to show all parameters of a module.
  [Tree:Show:PARAMS]

General Tree preferences:

- Show the original RTL name without parameters in the tree.
  [Tree:UseRtlName]
- Auto populate - If the database was opened from a binfile using quick mode, then the auto populate option can be used to automatically populate modules to see the entire contents.
  [Tree:AutoPopulate]
- Limit Number of Items - Limit the number of children in each sub-tree.
  [Tree:Limit]

Appearance Properties



In this tab you can customize the appearance of the tool. There are several predefined color schemes to choose from. You can customize one of these color schemes by clicking with the mouse on the preview label of the color. The highlight color is subdivided into a foreground and a

background color. The selection background color has a higher priority as the highlight background color.

NOTE	All changes to Schematic apply to the Schem and Cone window of all Pane windows simultaneously.

- Schematic - A listbox for choosing the schematic element; the current color for the currently selected schematic element is displayed on the right; click on that color to change it.
- Highlight - A listbox for choosing the highlight color; the current colors for the currently selected highlight list are displayed on the right; the left color is the foreground highlight color and the right color is the background highlight color; click on the colors to change them.
  Please note that the highlight background color of the Schem and Cone window is automatically derived from the highlight foreground color.
  You can choose between 16 different colors for highlighting.
- GUI - A listbox for choosing a GUI element; the current color for the currently selected GUI element is displayed on the right; click on that color to change it.
- Wave - A listbox for choosing an element of the Wave window; the current color for the currently selected element is displayed on the right; click on that color to change it.
- Source - A listbox for choosing an element of the Source window; the current color for the currently selected element is displayed on the right; click on that color to change it.
- Set to Light Colors - Reset the colors to the built-in 'light' color set (this is the default color scheme).
- Set to Dark Colors - Reset the colors to the built-in 'dark' color set.
- Icon Size - Select the size of icons in the toolbars.

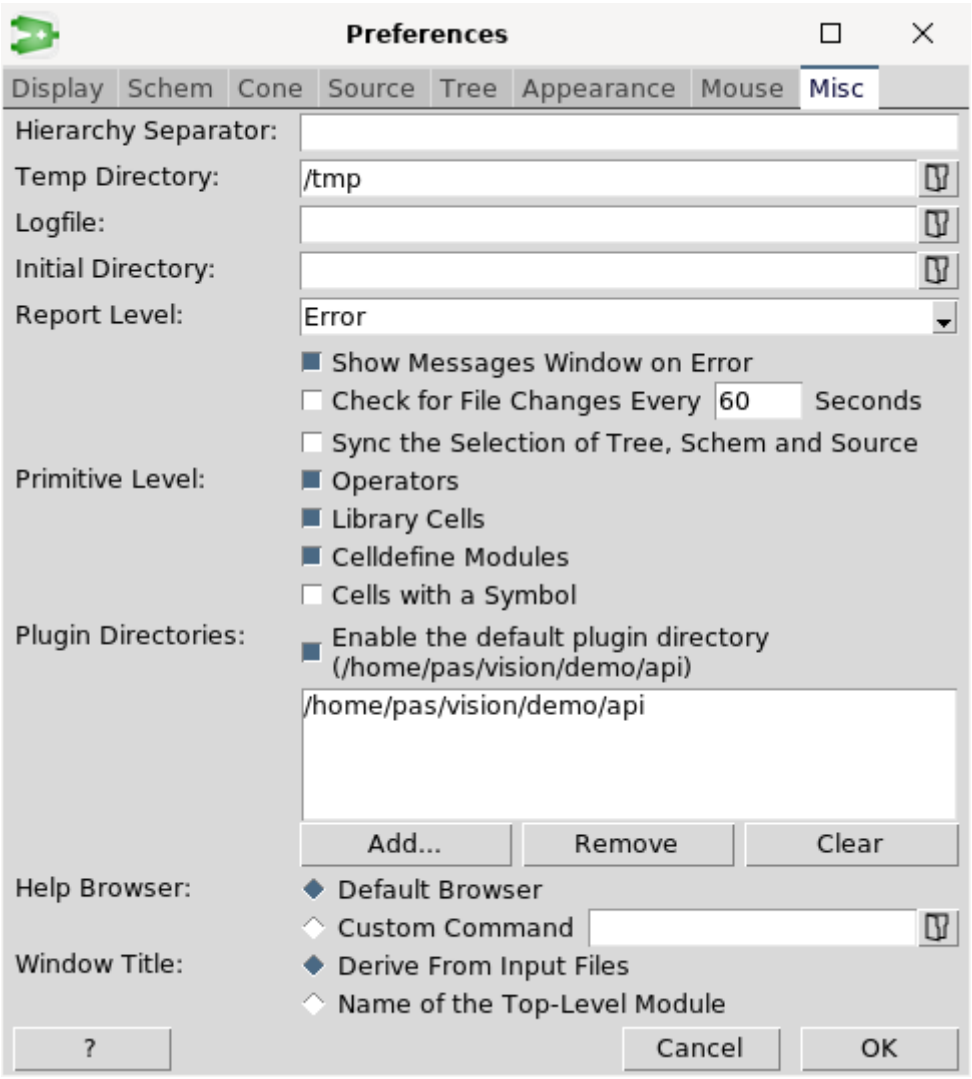
You can also customize the fonts used in {RTLvision PRO}:

- Normal - the generic GUI font used for buttons, labels, lists.
  [`fontFamily:normal`], [`fontSize:normal`]
- Menu - the font used for menus.
  [`fontFamily:menu`], [`fontSize:menu`]
- Tooltip - the font used in tooltips.
  [`fontFamily:tooltip`], [`fontSize:tooltip`]
- Fixed - the fixed-width font used in the Console and Messages windows.
  [`fontFamily:fixed`], [`fontSize:fixed`]
- Source - the font used in the Source windows.
  [`source:fontFamily`], [`source:fontSize`]
- Nlview - the font sizes used in the schematic displays (Nlview widget) of Schem, Cone, and Infobox windows. Please note that the configured size corresponds to zoom level 1, and that the font family cannot be configured.
  [`fontSize:nlview`], [`fontSize:nlview_big`], [`fontSize:nlview_small`]

Mouse Properties

- Drag & Drop Button - Select the mouse button used for Drag & Drop.
  [dndbutton]
- Gestures/Strokes Button - Select the mouse button used for drawing gestures/strokes in Nlview windows (Schem, Cone, etc.).
  [button_gestures]
- Panning Button - Select the mouse button used to pan Nlview windows (Schem, Cone, etc.).
  [button_panning]
- Zoom Wheel - Select the mouse wheel + modifiers to zoom in Nlview windows (Schem, Cone, etc.).
  [wheel_zoom]
- Horizontal Scrolling Wheel - Select the mouse wheel + modifiers to scroll horizontally in Nlview windows (Schem, Cone, etc.).
  [wheel_scroll_x]
- Vertical Scrolling Wheel - Select the mouse wheel + modifiers to scroll vertically in Nlview windows (Schem, Cone, etc.).
  [wheel_scroll_y]

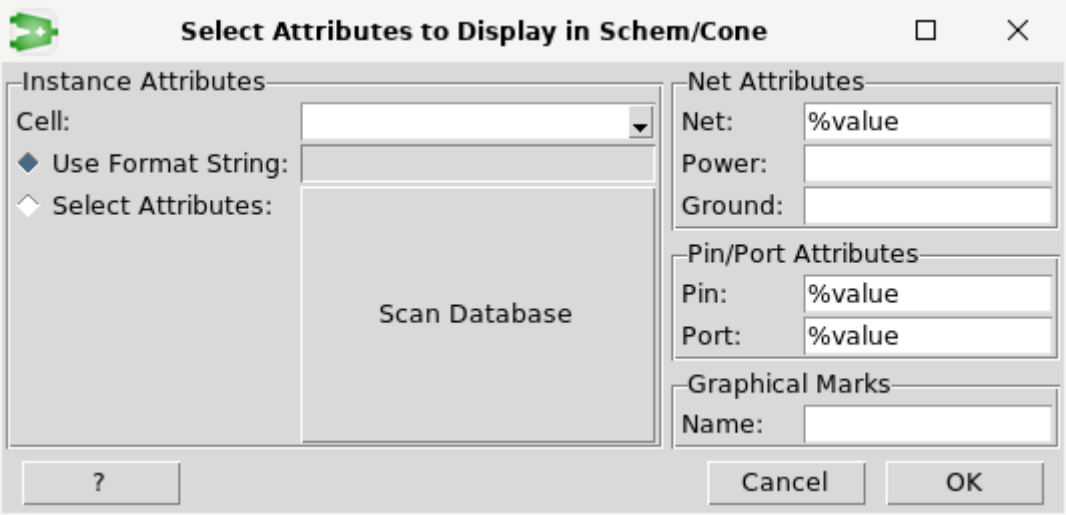
Misc Properties



- Hierarchy Separator - If the specified character is not used in any name then it will be used as a separator character for path names.
  [hiersep]
- Temp directory - Choose the directory, where RTLvision PRO can store temporary files (e.g. for parsing Verilog netlist files on Windows).
  [tempdir]
- Logfile - Write all warning and error messages shown in the Console window to this file. If Report Level is Debug then additional information (not shown in the Console window) is written to this file.
  [logfile]
- Initial Directory - Use this initial directory for all file dialogs.
  [initialDirectory]
- Report Level - Select the level of created messages.
  [info]
- Show Messages Window on Error - In case of an error message the Messages window becomes visible.
  [showConsoleOnError]

- Check for File Changes Every - If enabled then the current design's files will be checked for changes at regular intervals. In case a change is detected, an appropriate message will be displayed in the Console window, asking the user to reload the design.
  [filewatcher:enabled], [filewatcher:interval]
- Sync the Selection of Tree, Schem and Source - If enabled each selection in Tree and Source and each module change in Schem is synchronized to all Tree, Schem and Source windows.
  [syncToObject]
- Primitive Level - Specify hierarchical cells as primitives.
  - Operators are all cells with a known function.
  - Library Cells are cells flagged as coming from a library (e.g. a binlib).
  - Celldefine Modules are modules surrounded by the Verilog macros `celldefine and `endcelldefine.
  - Cells with a Symbol are all cells with a @symbol attribute (from a symlib).
    [primLevelOper],
    [primLevelLibCell],
    [primLevelCelldefine],
    [primLevelSymbol]
- Enable the default plugin directory - If checked, the installation's default plugin directory is scanned for plugin scripts.
  [plugins:usedefaultdir]
- Plugin Directories - Edit the list of directories that are scanned for plugin scripts.
  [plugindirs]
- Help Browser - Select the type of help browser to use:
  - Default Browser - Use the operating system's default web browser and use it as the help browser.
  - Custom command lets the user specify the path to an external web browser (e.g. the path to the system's Firefox executable).
    [help:browser_type], [help:browser_command]
- Window Title - Select if the displayed window title is derived from one of the input filenames or from the top module name.
  [deriveTitle]

Select Display Attributes



This dialog is accessible from the **View › Select attributes** menu. With this dialog the attributes to display in the Schematic and Cone window can be selected.

In this dialog you can define display rules for different kinds of attributes. You must enter a format string that defines the attributes to display.

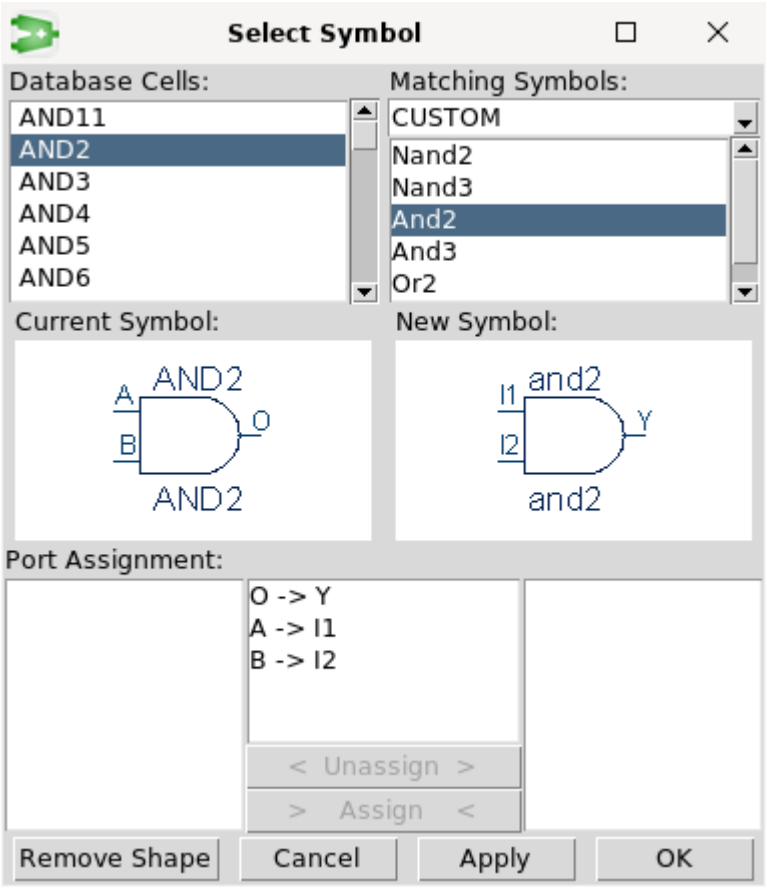
The entry field for the various Instance attributes define format strings for instances of each cell.

After pressing the "Scan Database" button all available attributes are shown as check buttons and can be selected to make them visible.

The entry fields Net, Power, Ground, Pin and Port define global format strings for attributes at each of those objects.

In the Graphical Marks field an attribute name containing graphical marks information can be specified.

Select Symbol



This dialog is accessible from the Select Symbol entry in the context menu and allows you to assign symbol shapes to modules displayed as boxes in the Schem and Cone window.

The list on the left side shows all available database cells. A preview of the selected cell is displayed below the list. This preview shows how the cell will be drawn in the Schem and Cone window.

On the right side a symbol can be selected. This could either be one of the built-in symbol shapes or a custom symbol. A list of all available custom symbols coming from a symbol library is displayed. The preview window shows how the selected cell would be drawn in the Schem and Cone window if this symbol shape is assigned by pressing the **[Apply]** or **[OK]** button.

If the order of ports at the cell loaded to the database does not match to the order of ports in the symbol shape definition then the port order can be changed using the "Port Assignment" dialog at the bottom of the Select Symbol dialog.

If a selected database cell already has an assigned symbol shape, the **[Remove Shape]** button can be used to remove the assigned symbol shape.

The Blocklevel View

The Blocklevel View mode is an abstract view of the original design that displays data flow between instances instead of explicit connectivity.

The Blocklevel View mode can be activated by clicking the icon in the toolbar or by selecting **Tools › Blocklevel View › Enable** from the main menu.

The Blocklevel View replaces all hierarchical cells by special blocklevel cells (indicating signal flow) and merges all nets/buses connecting two instances by exactly one artificial net.

Selecting an instance in the Blocklevel View mode colors the neighboring instances such that it becomes obvious which neighboring instance is driving the selected instance, is driven by the selected instance, or is both driving and driven (feedback) by the selected instance.

The behaviour of the Blocklevel view can be configured by activating features accessible from the **Tools › Blocklevel View** menu.

Console Window

The Console window can be opened with the icon in the toolbar or with **Window › Console** from the main menu.

The tool is based on Tcl/Tk and the Console window displays error or warning messages and provides direct access to the Tcl interpreter. The user can type in any Tcl command to execute it.

```
% set version [zdb version]
2024 (git=e5cecb3), License=DUMMY 2.0 (license version: 2.0), Binfile=16.0 (2024-A
pr-09)
%
```

The example above shows a **Console** window with one user command.

The **Console**'s main purposes are:

- **Evaluate Tcl commands**. It accepts the complete Tcl/Tk language and some tool specific extensions, like the database API and GUI API. These extensions make the tool very flexible, e.g. through the usage of user scripts called 'Userware'.
- **Present Warnings and Errors**. In some cases the GUI emits warnings, errors or information that are not displayed in a message box; they're rather displayed in the Console in a different color (blue for warnings, green for return values of executed commands and red for errors). Parser errors which contain file and line information can be clicked with the mouse to display the file in the Source window.

The **Console** also features a small context menu with text-based copy & paste functionality, the option to clear the Console window and the possibility to save the contents as a text file.

Statusbar

The appearance of the Statusbar (appears at the very bottom) can be toggled by selecting **Window › Statusbar** from the main menu.

The statusbar displays statistics of the loaded design:

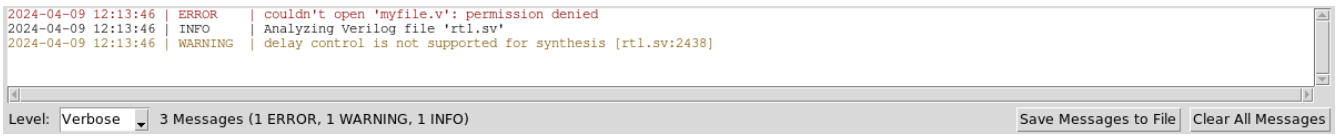
- the number of errors and warnings (by clicking on this field, the Messages window will be displayed).
- the memory consumption (by clicking on this field, a detailed report will be displayed).
- the number of loaded design files.

Messages Window

The Messages window can be opened with the icon in the toolbar or with **Window › Messages** from the main menu.

The Messages window shows all messages issued to the internal messaging system.

By default only error messages are displayed. The type of messages to be shown can be selected from the "Level" combobox.



Connectivity Browser

The Connectivity Browser can be opened with **Window › Connectivity Browser** from the main menu or from the submenu when right-clicking a net or a net bus.

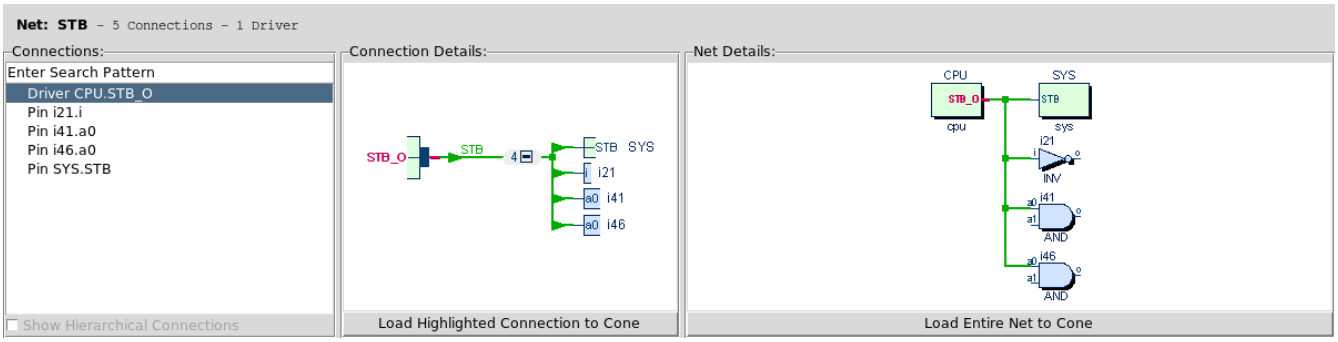
The Connectivity Browser shows the connectivity of the selected net or netBus.

It provides a list of the connected pins and ports. Connection Details displays a port/pin centric view for the port/pin selected in Connections.

In the Net Details and NetBus Details an overview of the entire connectivity of the selected net or netBus is shown.

The net or netBus can be selected via the popup menu or can be dragged and dropped in the Browser's tab.

In the signal mode all connected pins and ports of the selected signal are listed. The option "Show Hierarchical Connections" adds all directly connected pins and ports of the traversed modules. The Signal Details shows a signal overview at the top level of the signal.

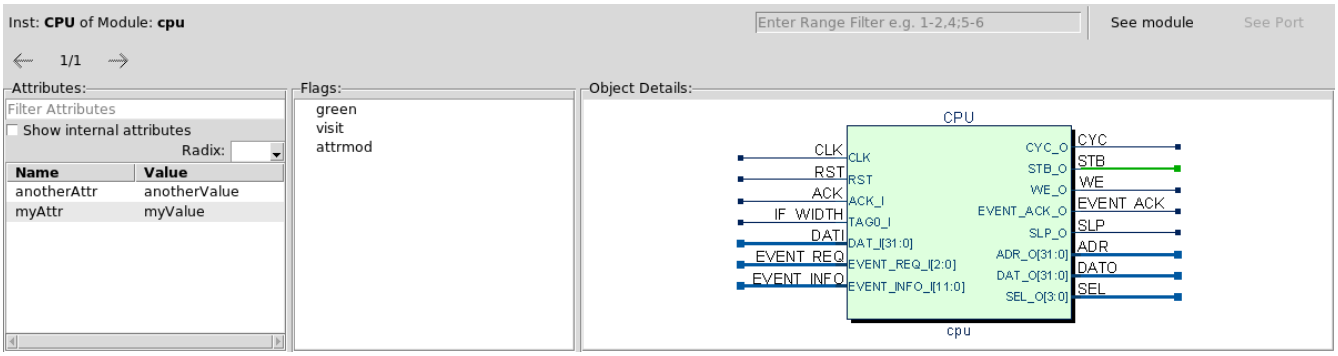


Infobox

The Infobox window can be opened with the ⓘ icon in the toolbar or with **Window › Infobox** from the main menu.

The Infobox shows additional information of the selected object. In the first column the appended attributes are listed. Show internal attributes in the list by toggling the "Show internal attributes" checkbox (this also includes calculated attributes). [`showInternalAttributes`] In the second column the active flags are shown. The third area contains a detailed view of the object. It show e.g. hidden ports, highlights and connection details. For large elements this view is optimized for performance reasons and can be adjusted by increasing the global Big Module Limit.

If signal mode is enabled an overview of the signal is shown.



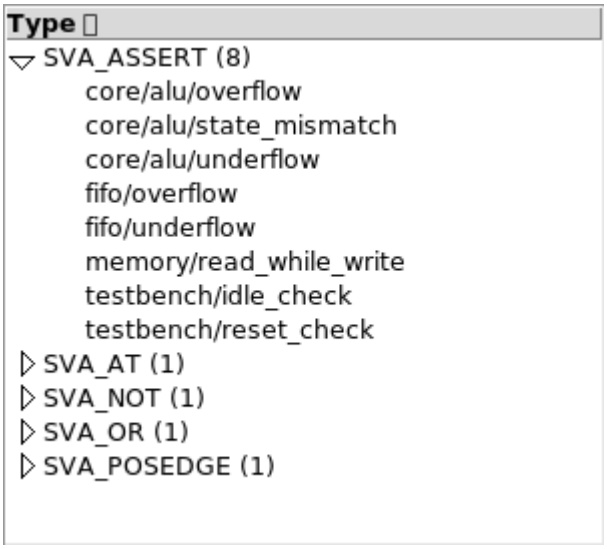
If a bus is selected a filter can be applied to display chosen parts. Valid strings are single bits like `1,2,3` or ranges e.g. `1-3`, or mixed `1-3,4`.

Attributes

The Attributes shows attributes of the selected object.

The Assertion Window

The Assertion window lists all assertions in the design. The assertions are presented as a tree, where the different types of assertions are the parents and the corresponding instances the children.



Userware

RTLvision PRO can execute Tcl scripts (we call them **Userware**). These scripts can access the GUI API and the database API and Tk (e.g. to extend the GUI). To execute an Userware script, you have several possibilities:

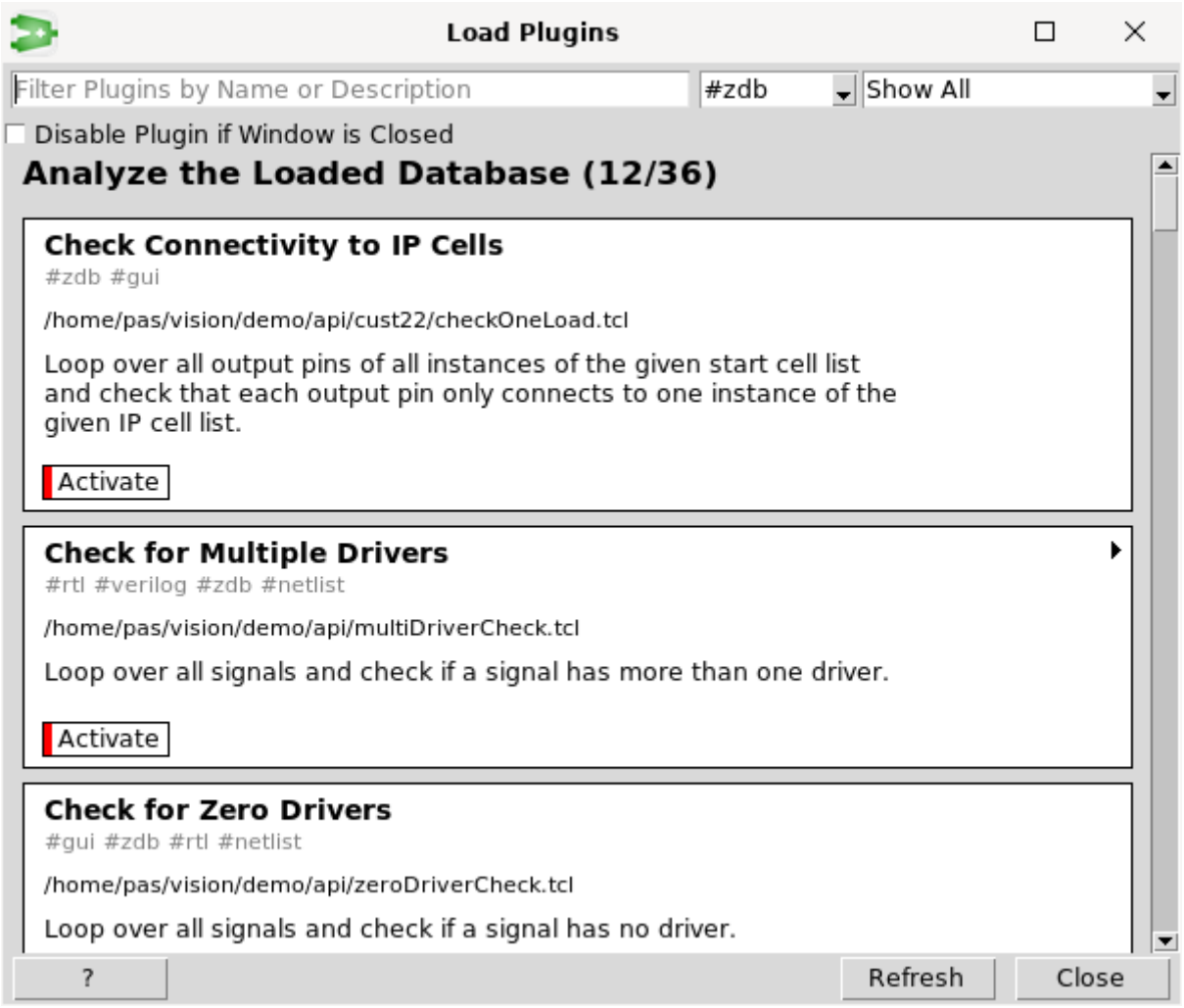
- Choose the menu entry **File › Load Userware**
- Specify the command line option `-userware <scriptname.tcl>` at startup
- Type `source "scriptname.tcl"` in the Console window

Learn more about Userware scripts here:

- API Tutorial - Learn how to write Userware scripts
- GUI API - Learn how to access the GUI components
- Database API - Learn how to access the database

The Plugins Dialog

The Plugins dialog can be opened with the icon or from the **Tools › Plugins** menu entry.



The Plugins dialog lets the user explore and load plugins, which are Userware Tcl scripts with 'plugin character' in contrast to just being mere 'API usage examples'.

The dialog lists all plugin scripts found in the configured 'Plugin Directories' - you can change the list of directories by the Plugin Directories section in the Misc tab of the Preferences dialog, or by using the command line option -pluginDir.

A specific plugin can be loaded by clicking on the plugin's **[Activate]** button. The behavior of some plugins can be configured via the **[Configure]** button.

The Filter input field lets you perform a keyword search on the list of plugins.

Each Plugin is tagged with one or more keywords to additionally categorize it. The list of Plugins can also be filtered based on these tags.

Some Plugins create a tabbed window. If the checkbutton "Disable Plugin if Window is Closed" is enabled, then closing a tab (e.g. by clicking the 'x' in the tab) will unload and therewith disable the corresponding plugin.
[plugins:unloadondestroy]

Symlib Utilities

The Symlib Utilities is a package that includes small programs to convert and maintain Altair Engineering, Inc.'s **Symlib files**. Each Symlib file defines schematic symbol shapes for a set of netlist cells. Below is an index of the related documents.

Document	Description
Symlib Format	This chapter describes the (ASCII based) format that is used to define custom schematic symbol shapes. The symlib data is stored in a symlib file, that is scanned by RTLvision PRO.
Symlib Editor/Viewer	This chapter describes Symedit, a simple symbol editor program that displays the content of a Symlib file - and allows the user to modify and/or create graphical symbol shapes interactively.
cadence2symlib	This chapter describes cadence2symlib.il, a Skill script that exports symbol shapes from a Cadence library to the Symlib format.
slibconv	This chapter describes the slibconv utility to convert symbol libraries from the Synopsys slib format to the Symlib format.

Symedit - Symlib Editor/Viewer

Symedit is a GUI tool to create, modify or maintain symlib files. A symlib file is a collection of schematic symbol shapes stored in ASCII text format.

Symedit reads the symlib files and displays the symbol shapes. The symedit program also creates or updates the file index header if necessary.

Overview

The symbol editor / viewer is an executable file called `symedit` that is part of the RTLvision PRO package.

Invoke the Symlib Editor/Viewer

Just call the executable file: `symedit.exe` on Windows, or `symedit` on Linux. If you want, you can specify the symlib file as additional argument, in this case the specified file is read at startup time:

```
symedit file.sym
```

If the symlib file does not exist, then an empty file with the specified name is created.

The optional argument -view starts symedit with the View tab opened.

Creating a New Symlib File

If you already have an empty (untitled) window displayed, just begin creating your symbols as

described below. To create a new untitled symlib file, select **File › New Symlib File**. An opened symlib file is closed and the lists of symbols and symbol elements are cleared. To permanently commit your changes, choose **File › Save** or **File › Save As**.

Open Symlib File

When opening a symlib file (**File › Open Symlib File**) or when symedit is invoked with a filename argument, then the file is parsed and checked. If the checker finds an invalid index header (invalid `symref` entries) then the user is asked to create a new (or overwrite the existing) symlib file - in order to update the index header. After your existing symlib file has been opened successfully, the "Symbols" listbox is populated with the names and view names of the symbols just parsed. In the View tab the symbols shapes are displayed - possibly split over a couple of pages.

Save Symlib File

Whenever a symlib file is saved (**File › Save** or **File › Save As**) a **valid index header** is created automatically.

The symlib file's `symref` entries store character positions that usually get invalid if the file is hand-modified (by a text editor). This means, after hand-editing a symlib file, you must use "symedit" to read the symlib file in order to update the *index header*.

Revert Symlib File

To discard all changes and to revert back to the last saved state, you may use **File › Revert**. This function is only available if you saved your symlib file before.

Import

The **File › Import** sub-menu offers the possibility to:

- import symbols from a **Synopsys Slib** library
- import symbols from a **EDIF 2.0.0** schematic file

Export

The **File › Export** sub-menu offers the possibility to:

- write out a (schematic) **EDIF 2.0.0** compliant file
- write out a **Postscript** picture of all symbols
- write out a **Verilog module** with the symbols' footprints

Exporting EDIF and Postscript will pop up a small dialog first with some user options for tweaking the output.

Symlib Properties

The menu entry **File › Properties** pops up a small dialog window to enter some global settings and comments for your symlib file.

Controlling the View

Use the **View** menu or the built-in mouse strokes (left mouse button press-drag-release) to zoom in/out and the scroll-bars or middle mouse button press-drag-release to change the scroll position. This works in both the Edit and the View window. If the View window is active you can use the **Pages** menu to switch to different pages.

The **Sheetsize** menu defines the sheet size that is used when reading a new symlib file. The sheet size is one of the US formats A through D or one of the European A4, A3 or A2.

Running Tcl Scripts

The menu entry **Utilities › Run Tcl Script** pops up a file selection dialog asking the user to specify a Tcl script. This enables the user to modify the symbol shapes by self-made Tcl scripts.

The specified Tcl file is evaluated by a slave interpreter that only knows about one Tcl variable called `symbolList`. That variable is a Tcl-list that stores all the symbols, one in each list element. If the `symbolList` is modified by executing the script, then the changed symbols are displayed in the Edit window. After that you must save (**File › Save As**) the modified symbol data into a file, if you don't want to lose it. Here is a small example that deletes all symbols that have name prefixes other than "and":

```
set resultList {}

foreach sym $symbolList {
    set sym_name [lindex $sym 1]
    if [string match "and*" $sym_name] {
        lappend resultList $sym
    }
}
set symbolList $resultList
```

Example Tcl Scripts

nullmap.tcl

The example script `symutils/scripts/nullmap.tcl` defines a set of 1:1 mapping rules and calls `apply_rules` to apply that rules for all loaded symbols (as stored in `$symbolList`).

The Tcl procedure `apply_rules` is implemented in the file `symutils/scripts/mapping.tcl`, that is "sourced" from `nullmap.tcl`.

This example can easily be modified to implement a real symbol-name and port-name mapping, based on `symlib/generic.sym`

move.tcl

There is a more complex example script in `symutils/scripts/move.tcl` that changes all x coordinate values to move the left edge of all symbol shapes to 0.

Log Console

All Events are logged, you can see these events using the Log Console (**Utilities › Show Log Console**). A new window appears where you can save the logs to a file or clear all logged events.

Manipulating Symbols

Deleting a Symbol

If a symbol is selected from the list of symbols, then it can be deleted by the **Symbol › Delete** menu button. A confirmation dialog will pop up first.

Renaming a Symbol

If a symbol is selected from the list of symbols, then it can be renamed by the **Symbol › Rename** menu button. Now the symbol name and the view name can be changed in a new dialog window.

Copying a Symbol

If a symbol is selected from the list of symbols, then it can be copied by the **Symbol › Copy** menu button. "copy_of_SYMBOLNAME" is appended to the list of symbols.

Editing the Symbol Shape

If a symbol is selected from the list of symbols, the symbol shape can be modified interactively using the mouse. Open the Edit window to view the shape.

The radio buttons above the editing canvas define one of the following operations:

- Select:
  An element can be marked by clicking on it with the mouse. The color changes to red and the definition is shown in the text field above the canvas.
  Another way to select one or more elements is to mark them in the listbox on the lower left side which contains all elements of a symbol. In this case, the definition of the last selected item in the listbox will be displayed.
  In this mode you can use the left mouse button to drag a rectangular (zoom) area. Start dragging from top-left corner and stop dragging in the lower right corner of the area to be zoomed in.
  To zoom out: use the left mouse button to drag a line in top right direction.
  Use the left mouse button to drag a line in bottom left direction. The zoom factor is adjusted in such that the shape will fit best into the editing canvas.
- Move:
  Click on an element and hold down the mouse button. Now the element can be dragged to the new position. After the mouse is released, the new position is stored. Hint: You can always use the right mouse button to perform a temporary move action.

- Copy:
  Click on an item and hold down the mouse button. Now a copy of the element can be dragged to a new position. The new element is appended to the list of elements.
- Rotate cw/ccw:
  Click on an element to rotate the element either clockwise (cw) or counterclockwise (ccw) by 90 degrees. The rotation center point is the text justification point or the graphic element's bounding box center.
- Delete:
  Click on an element to delete the element. Alternatively, you may select "Delete" from a popup dialog which opens on right click in the Symbol elements listbox on the lower left.
- Create:
  There is a drop down menu next to the Create button with some graphical symbol elements. After selecting an element from this list the new item can be drawn on the canvas using the mouse. A create operation can be canceled by pressing the ESC key or by switching to another action.
  Possible elements are:
  - port/portBus:
    A port or portBus shape is usually defined by two points, the start point and the end point (polygon shaped ports are supported while parsing and manipulating, but cannot be created by symedit). The port direction is determined by the location of the start and end point. The port name is assigned automatically (starting with A, B, ..., Z, AA, AB, ... etc.) and can be modified later. The default width of a portBus is 4 and can be modified later.
  - pinattrdsp:
    A pinattrdsp applies to the most recently preceding port or portBus definition; therefore a pinattrdsp needs to be created after a port or portBus.
  - path:
    While clicking path points in the canvas these points will form a polygon. Path creation ends by double clicking the path end point.
  - arc:
    An arc is defined by three points. First the arc start and the arc end points are defined. Now the mouse can be moved to define the arc middle point.
    In order to create a circle, double click at the circle start point and move the mouse to the circle end point.
  - attrdsp:
    After selecting the position of the lower left justification point, a text window appears to enter the attribute name.
  - text:
    After selecting the position of the lower left justification point, a text window appears to enter the text.

After creating an element you can switch to the View window to see a preview of the symbol (you might want to perform a "Zoom Fit" mouse stroke first). With the **Edit › Undo** menu button you can undo the last symbol manipulations.

Hint: You can use the `Shift` and/or `Ctrl` keyboard modifiers in combination with move or create actions to grid snap your mouse position; this will give you more precise control over coordinates.

Create a New Symbol

The **Symbol › New** menu button creates an empty symbol. Now, new symbol elements can be added as described above.

Search a Symbol

The menu entry **Symbol › Search** pops up a small dialog window to search for symbols by name. If "wildcard" is active (default), then the characters *****, **?** and **[...]** have the special meanings: "*any string*", "*any character*" or "*any of the enclosed characters*" respectively.

The result of the search is displayed in a list of symbol names including each symbol's view-name. Commonly, the view-names are * to indicate that the view-name matches any view-name requested by e.g. the RTLvision PRO tool. If you choose one of the symbols in the listbox, it will be selected and centered in the View or Edit window.

Cadence Symlib Export

Using the `cadence2symlib.il` script symbol libraries needed for RTLvision PRO can be extracted from Cadence libraries (e.g. analogLib). The contained mappings from symbol names to model names may need some manual modifications.

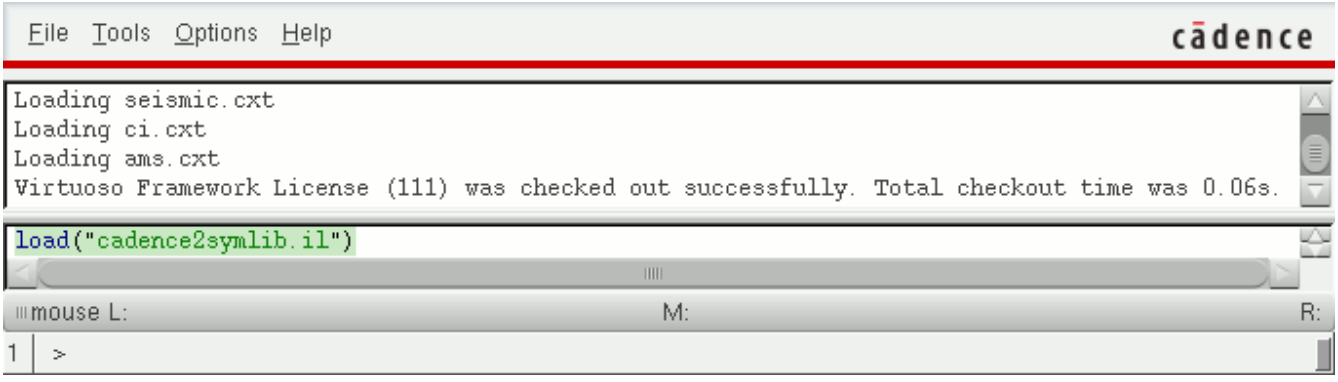
To extract an symlib file the following steps need to be done.

Start your Cadence environment, e.g. by typing `virtuoso`:

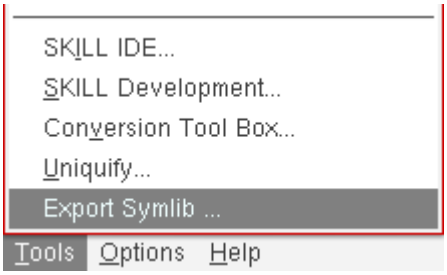
```
% virtuoso &
```

Type the following command into the Input Line of the Command Interpreter Window (CIW):

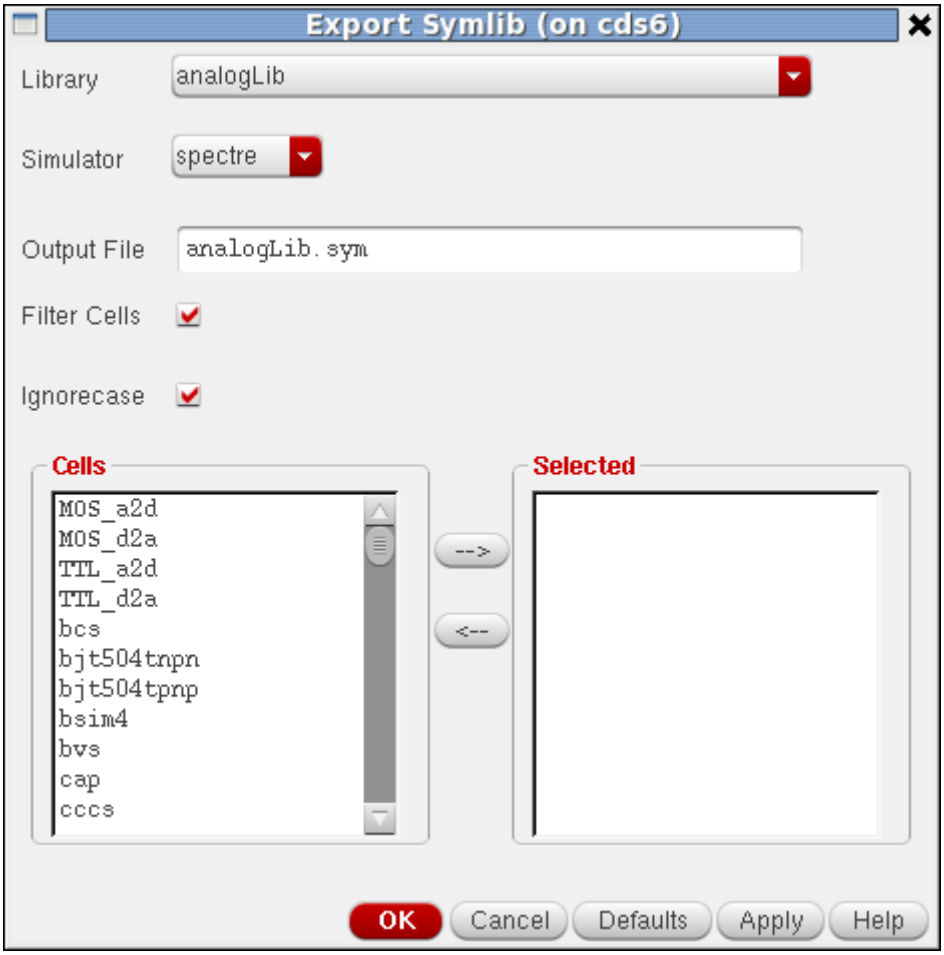
```
load("cadence2symlib.il")
```



The `cadence2symlib.il` script adds a new entry "Export Symlib ..." to the "Tools" menu.



If you invoke this entry then the following dialog window will show up.



Please select your library in this form, e.g. analogLib and the Simulator to obtain CDF information. You can also change the name of the output file. Press the OK button to run the export procedure.

Please examine the output of the script in the CIW output area to see if any symbols you intend to use have problems, e.g. no CDF info or no TermOrder info.

Slibconv - Symbol Format Converter

This is a converter that converts symbol libraries from Synopsys slib format to the Symlib Format used by the RTLvision PRO tools.

Invoke the Converter

The converter is invoked from the command line (from a Linux-shell or from the Window's Command Window). You must specify the Synopsys slib file (input file) as a command argument:

```
slibconv file [-o output.sym]
```

The converter expects a configuration file called `slibconv.conf` to be found in the current directory or specified using the `-slibConfig` command line option. If no output file using the `-o` command line option is specified then the converted output file is called `output.sym` and is placed in the current directory.

Example: If the file to be converted is called `generic.slib`, conversion can be performed by issuing the following command at the prompt:

```
slibconv -o generic.sym generic.slib
```

The Configuration File

This is the configuration file for the converter. It should be present in the same directory as the converter (or specified using the `-slibConfig` command line option). If the converter is unable to find the configuration file, it will use the default option values.

The configuration file allows the configuration of two types of options:

1. Symbol Attribute Option (such as: the font size, size of the symbols, etc...).
2. Advanced Program Options (such as: whether zero length lines should be deleted or not, etc...)

The Configuration File Format

The file is in ASCII format and except for comments must be written in lower case characters. The file contains two types of entries:

Configuration Options

A configuration option instructs the program to perform a task. A configuration option takes the format:

```
option value
```

Where **option** is a name of a valid option and **value** is a valid setting for that option. Depending on the **option**, an integer, a real number, keyword **yes**, keyword **no** are the only valid settings for the **value** field.

Comment

A comment is ignored by the program. A comment takes the format:

```
# This is a comment
```

A comment must start at the beginning of a new line with the character `#`, and the text following the character are ignored to the end of line. There must be at least one empty space between the `#` character and the first character of the comment.

The Configuration File: Valid Options

Symbol Attribute Options	Short Description	Default Value	Type
display_cell_name	If switched on displays the name of the symbol below the symbol.	yes	yes/no
display_instance_name	If switched on displays the symbol instance name at the top of the symbol.	yes	yes/no
font_size	This is the font size used to display pin names, symbol names and symbol instance names.	10	integer
display_pin_name	If switched on displays the names of symbol pins.	yes	yes/no
scale	All the symbols in the input file are multiplied by the `scale` value to arrive at the converted symbol.	10.0	real
ignore_templates	If switched on templates are ignored by the converter.	yes	yes/no
default_pin_length	If a pin is not affected by the Advanced Option `convert_pins` the displayed pin will have the length defined by this variable	5	integer
ignore_case	If on, then the resulting symlib file is case-insensitive (symbol and port names).	no	yes/no

Advanced Options	Short Description	Default Value	Type
convert_pins	If there is a pin defined at a place where there is also a line that can function as the pin, switching on this option will convert this line into a pin.	yes	yes/no
continuous_pins	Works together with the `convert_pins` option. If switched on, the converter will try to provide the longest possible pins.	yes	yes/no
delete_zero_length_lines	If switched on deletes the lines that do not have a length.	yes	yes/no
delete_duplicate_lines	If switched on deletes lines that are redundant.	yes	yes/no
divide_lines	If switched on line division is performed (i.e.: if the endpoints of a line or an arc fall on another line, the latter line is divided at that point.)	yes	yes/no

Advanced Options	Short Description	Default Value	Type
delta	The value ignored by the converter when performing various internal calculation. This is necessary because the converter is converting from the Synopsys format that uses real numbers to the Symlib Format that uses integer numbers.	0.051	real
unknown_pin_direction	If switched on ANY_ROTATION if converted LEFT/UP/DOWN/RIGHT.	guess_sym_dir	If switched on in the sym file INPUT is added to all pins with LEFT/UP/DOWN direction and OUTPUT to all pins with RIGHT direction

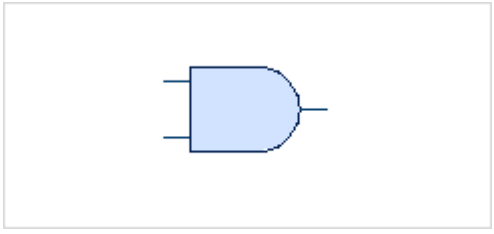
The Configuration File Option: `display_cell_name`

This option enables symbol name displaying. The symbol names are displayed directly below the symbols.

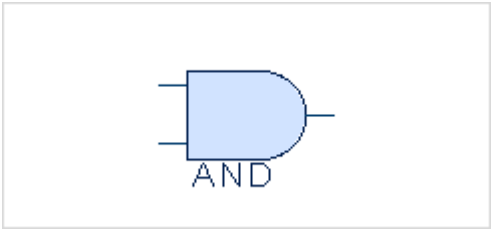
attribute type: `yes/no`

default value: `yes`

`display_cell_name no`



`display_cell_name yes`



The Configuration File Option: `display_instance_name`

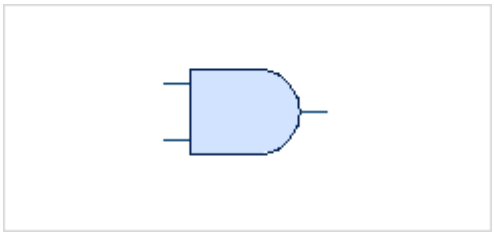
This option enables symbol instance name displaying. The symbol instance names are displayed

directly above the symbols.

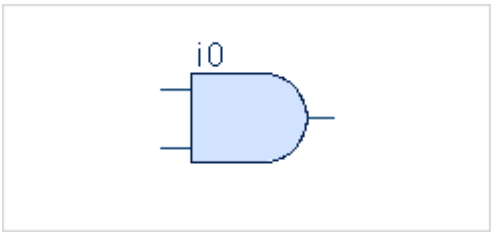
attribute type: `yes/no`

default value: `yes`

`display_instance_name no`



`display_instance_name yes`



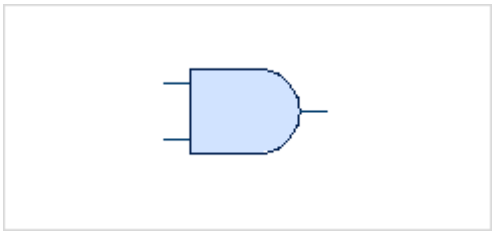
The Configuration File Option: `display_pin_name`

This option enables symbol pin name displaying.

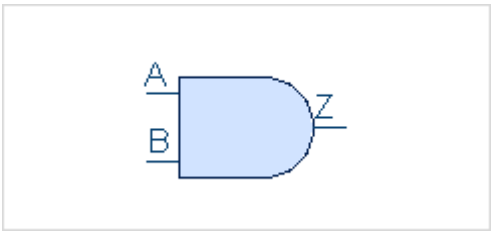
attribute type: `yes/no`

default value: `yes`

`display_pin_name no`



`display_pin_name yes`



The Configuration File Option: `font_size`

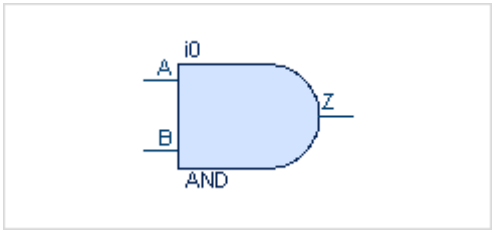
Defines the font size used by the following symbol attribute options.

1. display_cell_name
2. display_pin_name
3. display_instance_name

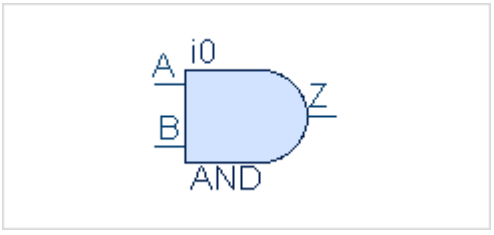
attribute type: `integer`

default value: `10`

`font_size 5`



`font_size 10`



The Configuration File Option: `scale`

The Symlib Format symbols are defined in 1/100 of an inch integer units. If your Synopsys Format input file is using a different unit it may be necessary to scale the symbols to their correct size.

```
scale = input file scale (in inches) / Symlib scale (in inches)
```

For example: If the Synopsys symbols are defined in 1/10 of an inch units, the scale can be calculated as follows.

```
scale = 0.1 / 0.01 = 10.0
```

Every input file symbol definition is multiplied by the `scale` value.

attribute type:	`real`
default value:	`10.0`

The Configuration File Option: `ignore_templates`

The Symlib Format does not support sheet frame templates. So there is no need to convert the templates in the input file, and this option should always be switched on. If this option is switched off Templates are processed as normal symbols.

attribute type:	`yes/no`
default value:	`yes`

The Configuration File Option: `ignore_case`

If switched on, then the resulting Symlib file is marked case-insensitive and slibconv internally maintains the symbol names and variable names case insensitive.

attribute type:	`yes/no`
default value:	`no`

The Configuration File Option: `default_pin_length`

This option controls the length of symbol pins and is given in 1/100 of an inch units. All converted symbols will have pins with this length.

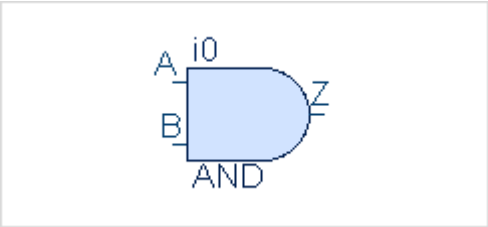
Note: In the following cases symbol pin length is not controlled by the `default_pin_length` option.

1. If the `convert_pins` option is switched on and the symbols were affected by this option.
2. Pins with the direction **ANY_ROTATION** will always be of zero length (except when affected by the `convert_pins` option).

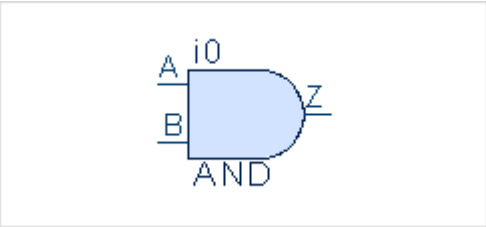
attribute type: `integer`

default value: `5` (i.e.: 5/100 of an inch)

`default_pin_length 5`



`default_pin_length 10`



The Configuration File Option: `convert_pins`

Sometimes a converted symbol may have pins as illustrated by the picture at below left. In the illustration the selected pin A has the length as defined by `default_pin_length`, but it is evident that the line between the pin and the triangle is what was actually meant to be the pin.

In situations like this switching `convert_pins` on would convert such lines into pins. This situation is demonstrated in the below right illustration.

See also `continuous_pins`.

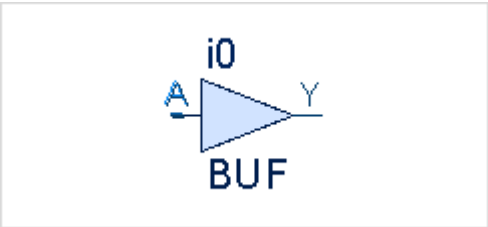
Note: If the `convert_pins` option is switched on, it is highly recommended that the following options 1, 2 and 3 are also switched on with the 4th option configured to an appropriate value.

1. `delete_zero_length_lines`
2. `delete_duplicate_lines`
3. `divide_lines`
4. `delta`

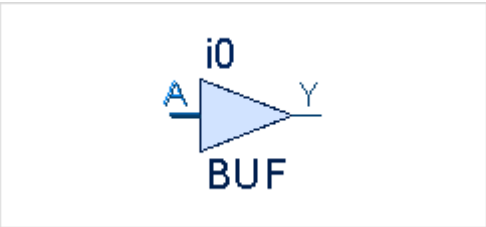
attribute type: `yes/no`

default value: `yes`

`convert_pins no`



`convert_pins yes`



The Configuration File Option: `continuous_pins`

Note: This option works together with `convert_pins` option and has no effect if `convert_pins is switched off`.

Sometimes switching `convert_pins` on alone would not ensure that the pins have the correct length.

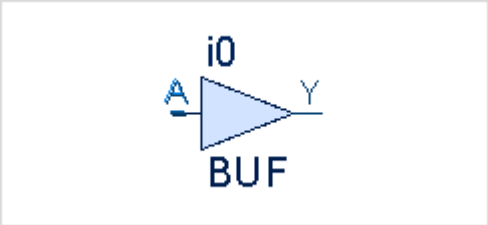
This is because sometimes a line from point X to point Y is not defined as a continuous line, but as two lines (i.e. one from point X to an intermediate point Z, and the other from the intermediate point Z to point Y). In situations like this only the line up to the intermediate point is converted into a pin. This situation is demonstrated in the below left illustration with the pin A selected.

Switching `continuous_pins` on would solve this problem by instructing the converter to provide longest possible pins (illustration below right).

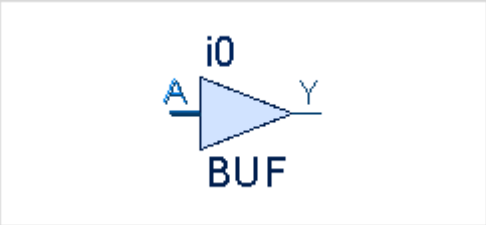
attribute type: `yes/no`

default value: `yes`

`continuous_pins no`



`continuous_pins yes`



The Configuration File Option: `delete_zero_length_lines`

Some symbol definitions may include lines that do not have a length.

Example: `line (0.0,0.0,0.0,0.0);`

Having zero length lines could interfere with the appearance of symbols. Sometimes the user may not be able to select a pin due to the presence of the zero length lines. Zero length lines may also, in some cases, cause `convert_pins` and `continuous_pins` options to function incorrectly. It is highly recommended that this option is always switched on.

attribute type: `yes/no`

default value: `yes`

The Configuration File Option: `delete_duplicate_lines`

Some symbol definitions may include redundant lines. Example:

```
line(0.0,0.0,10.0,0.0);

line (0.0,0.0,10.0,0.0); (redundant line 1)
line (0.0,0.0, 5.0,0.0); (redundant line 2)
```

Having redundant lines may interfere with the appearance of symbols. Sometimes the user may not be able to select a pin due to the presence of redundant lines. Redundant lines may also, in some cases, cause `convert_pins` and `continuous_pins` options to function incorrectly. It is highly recommended that this option is always switched on.

attribute type: `yes/no`

default value: `yes`

The Configuration File Option: `divide_lines`

If this option is switched on line division is performed (i.e.: if endpoints of a line or an arc fall on another line, the latter line is divided at that point.

Example:

before division:

```
line(0.0,0.0,10.0,10.0);
line(4.0,4.0,7.0,4.0);
```

after division:

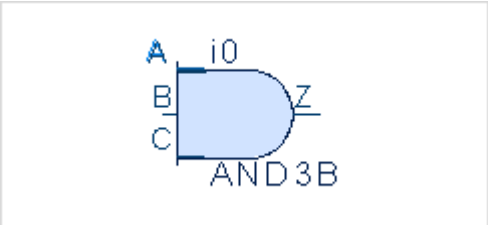
```
line(0.0,0.0,4.0,4.0);
line(4.0,4.0,10.0,10.0);
line(4.0,4.0,7.0,4.0);
```

This option provides a supporting function to `convert_pins`. Sometimes `convert_pins` may produce pins at unexpected places as demonstrated by pins A and C in the illustration below left (only pin A is selected). Switching `divide_lines` on would correct the error (illustration below right).
It is highly recommended that this option is always switched on.

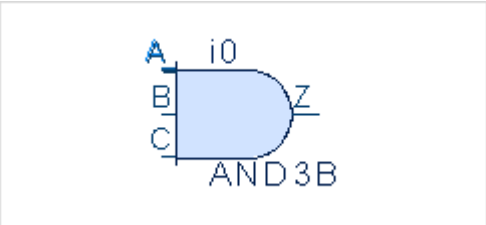
attribute type: `yes/no`

default value: `yes`

`divide_lines no`



`divide_lines yes`



The Configuration File Option: `unknown_pin_direction`

If this option is switched off ANY_ROTATION is converted to direction LEFT/UP/DOWN/RIGHT.

attribute type: `yes/no`

default value: `no`

The Configuration File Option: guess_sym_dir

If this option is switched on INPUT is added to the ports in the sym file for the directions LEFT/UP/DOWN. OUTPUT is added to ports with direction RIGHT.

attribute type: yes/no

default value: yes

The Configuration File Option: delta

The Synopsys format is based on real numbers and this is the value ignored by the converter when dealing with real numbers.

For example: if the difference between two real numbers is less than delta they are taken to be equal.

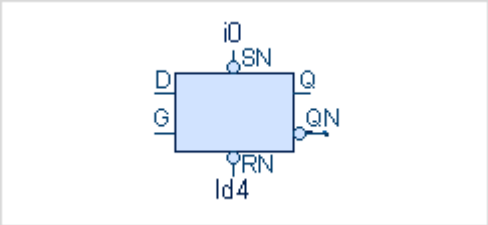
For some Synopsys input files it is necessary to have a high delta value so that convert_pins option can function properly. The default value provided will suffice in most cases.

The illustration below left demonstrates the effect of having a too low delta value when convert_pins is switched on. Even though convert_pins was switched on, pin QN was not converted because pin QN is defined more than the delta value away from the line that should have been converted into a pin. The illustration below right demonstrates the situation fixed, when the delta value was increased to 0.15.

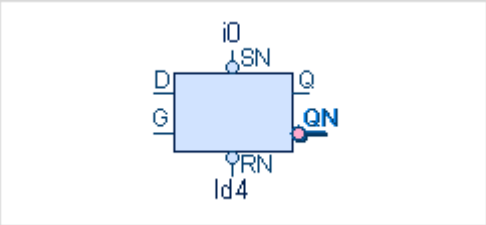
attribute type: real

default value: 0.15

delta 0.051



delta 0.15



A Sample Configuration File

```
# Configuration file for the slibconv converter.
#

# The font size of instance names, symbol names and pin names
# font_size is declared as integer
#
font_size               10

# This variable controls the value neglected by the program in order to
# deal with errors associated with dealing with real numbers. The program
```

```
# uses values less than delta as irrelevant in some functions
# delta is declared as double
#
delta  0.15

# The size of the symbols in the input file are multiplied by the scale value
# scale is declared as double
#
scale 10.0

# The default length of pins. This applies only if the program could not find
# a suitable line that could be used as a pin (finding suitable lines for pins
# is controlled by "convert_pins" variable).
# default_pin_length is declared as an integer
#
default_pin_length 5

# If switched on deletes lines with length less than delta
#
delete_zero_length_lines    yes

# If switched on deletes duplicate lines
#
delete_duplicate_lines  yes


# If switched on line division is performed (i.e.: if an endpoint of a line or an
# arc falls on another line, the latter line is divided at that point.
# example :
# before division :
#               line(0,0,10,10);
#               line(4,4,7,4);
#
# after division :
#               line(0,0,4,4);
#               line(4,4,10,10);
#               line(4,4,7,4);
#
divide_lines yes

# if switched on find suitable lines that could be viewed as pins. That is if a
# pin is declared at an endpoint of a line that can itself act as a pin, this
# line is taken as the pin.
#
convert_pins            yes

# works together with convert_pins option. (i.e.: has no effect if convert_pins
# option is switched off.
# if switched on, tries to make the converted pin longer if it is possible.
#
continuous_pins        yes
```

```
# If switched on displays the symbol names
#
display_cell_name       yes

# If switched on displays pin names
#
display_pin_name        yes

# If switched on displays the symbol instance names
#
display_instance_name   yes

# If switched on, templates are ignored
#
ignore_templates        yes
```

Error Messages

All error messages are written to standard error. There are two types of error messages printed by the converter.

syntax errors (errors in the syntax of the input file)
program errors (errors encountered while processing the correctly read input)

After an error has occurred the converter would pursue to recover and continue processing. But it should be noted that errors may affect the subsequent processing of the input and could provide unexpected output.

syntax errors

- `line no: syntax error`
- `line no: unknown function name`
- `line no: unknown token after name`
- `line no: ignoring symbol name`
- `line no: ignoring layer name`
- `line no: unknown expression name`

program errors

- `line no: no symbol name`
- `line no: duplicate symbol name`
- `line no: duplicate layer name`
- `symbol name has no elements`
- `missing slibconv.conf: reverting to defaults`
- `unknown parameter in slibconv.conf : option name`

Error Messages: Syntax Errors

Syntax errors inform the user of errors encountered while reading the input file. Syntax errors would normally produce two error messages. The first error message prints the line number and informs that an error has occurred.

```
line number: syntax error
```

The second prints the line number and the cause of the error.

```
line number: cause of the error
```

After an occurrence of a syntax error, the converter would try to proceed with the reading of the input file as long as it is possible. For example, if an error occurs while reading the contents of a symbol definition, the converter would first try to ignore the erroneous expression and proceed. If this is not possible it would try to ignore the whole symbol definition and proceed. If there are references to an ignored symbol later in the input file, the converter would generate a program error message.

The Syntax Errors

- `line no: syntax error`
- `line no: unknown function name`
- `line no: unknown token after name`
- `line no: ignoring symbol name`
- `line no: ignoring layer name`
- `line no: unknown expression name`

Error Messages: Program Errors

Program errors mainly inform the user of errors encountered while processing the correctly read data. Some syntax errors in the input file could also cause program errors at later stages. Program errors are not fatal errors.

Example program error:

```
duplicate symbol BUF
```

The Program Errors

- `line no: no symbol name`
- `line no: duplicate symbol name`
- `line no: duplicate layer name`
- `symbol name has no elements`
- `missing slibconv.conf: reverting to defaults`
- `unknown parameter in slibconv.conf : option name`

Syntax Error: *line no*: syntax error

This error message informs the user that an error has been encountered while reading the input file. The message also points to the line no in the input file where the error occurred. After a *line no*: `syntax error` message the converter may print another error message pointing to the cause of the error.

Example:

```
1137: syntax error
1137: unknown function "my_error_test_function"
```

Syntax Error: *line no*: unknown function *name*

A function is defined in the input file as follows

```
function_name  ( parameters ...)
```

This error message is produced when an expression in the input file contains a function that is not allowed by the Synopsys Format. The message prints the line number in the input file where the error was encountered and the name of the function in error.

Example:

```
1137: unknown function "my_error_test_function"
```

Syntax Error: *line no*: unknown token after *name*

This error message informs of errors that occurred in the input file before encountering the keyword `library(library_name)`. The message prints the line number in the input file where the error was encountered and the word immediately preceding the error.

Example:

```
928: unknown token after "loc"
```

Syntax Error: *line no*: ignoring symbol *name*

If an input file symbol definition contained one or more errors, the converter would first try to continue ignoring only the erroneous expression. If this is not possible it will continue ignoring the erroneous symbol. The error message prints both the line number of the last line of the ignored symbol definition and the name of the ignored symbol.

Example:

```
101: ignoring symbol "upward_port"
```

Syntax Error: *line no*: ignoring layer *name*

If one or more syntax errors occurred within a layer definition while reading the input file, the converter would first pursue to continue despite the error, but if that is not possible it will pursue to ignore the erroneous layer and to proceed again at the end of the layer. This error message prints the line number that contains the last line of the ignored layer and the name of the layer ignored.

Example:

```
1221: ignoring layer "test_error_layer"
```

Syntax Error: *line no*: unknown expression *name*

If an error occurs while reading the input file, within the `library(library_name)` keyword but outside the symbol and layer definitions, it is reported as an `unknown expression`. The line number of the error and the name of the unknown expression is printed.

Example:

```
4: unknown expression sdf
```

Program Error: *line no*: no symbol *name*

This error message is generated when a reference is made to a non existing symbol. The most common way that this can happen is when a symbol is ignored by the converter due to syntax errors, but is included as a sub symbol later in the input file. The line number of the error is also printed.

Example:

```
24: no symbol BUF
```

Program Error: *line no*: duplicate symbol *name*

This error message is generated when there exist multiple definitions of the same symbol in the input file. When a second definition of an existing symbol is encountered, the converter deletes the previous definition and inserts the new definition. The line number points to the last line of the duplicate symbol definition.

Example:

```
193: duplicate symbol BUF
```

Program Error: *line no*: duplicate layer *name*

This error message is generated when there exist multiple definitions of the same layer in the input file. When a second definition of an existing layer is encountered, the converter deletes the previous definition and inserts the new definition. The line number points to the last line of the duplicate layer definition.

Example:

```
1275: duplicate layer cell_name_layer
```

Program Error: symbol *name* has no elements

This error message informs the user that the converted symbol `name` contains no graphical information. This may be due to either

1. The original symbol definition in the input file did not have any graphical information in it.
2. The graphical information in the symbol was deleted during processing, e.g. the symbol only had zero length lines and the delete_zero_length_lines option was switched on.

Even though the symbol does not have graphical information in it, it will have an entry in the output file `output.sym`.

Example:

```
symbol BUF has no elements
```

Program Error missing slibconv.conf: reverting to defaults

This error message means that the converter was unable to find the configuration file slibconv.conf in the same directory as the converter or in the search path. If the configuration file is not available the converter would proceed with processing the input file using the default values of the configurable options.

Example:

```
missing slibconv.conf: reverting to defaults
```

Program Error: unknown parameter in slibconv.conf: *option name*

This error message informs that there was an unknown option in the configuration file slibconv.conf. Please make sure that only the valid options are included in the configuration file,

and that each line confirms to the format of the configuration file.

Example:

```
unknown parameter in slibconv.conf : test_error
```

Trouble Shooting (Symbols do not Look the Way they Should)

If the symbols do not look the way they should, please check whether the configuration file options are properly configured and that there are no syntax errors or program errors produced during the conversion process. If the problem has something to do with the graphical attributes of symbols (such as the absence of symbol names) these can be configured through Symbol Attribute Options.

If this is not the case read the description of the Configuration File Options above carefully. Pay particular attention to the option `delta`. If the pins are not the way they should be, or are not selectable by a mouse click it is most probably due to a too small or too a large `delta` value.

If you still could not find the source of the problem please do the following:

1. Turn off all of the following options in the configuration file.
    - `convert_pins`
    - `continuous_pins`
    - `delete_zero_length_lines`
    - `delete_duplicate_lines`
    - `divide_lines`
2. Run the converter and check the output. This output is produced with minimum processing by the converter. If this output is still not according to your expectations, proceed to the next step.
3. Turn on the options one by one in the following order, run the converter each time an option is switched on, and check the output. This would give you an idea as to where the things go wrong. You can stop at the point where you get a satisfactory output.
    - `delete_zero_length_lines`
    - `delete_duplicate_lines`
    - `divide_lines`
    - `convert_pins`
    - `continuous_pins`
4. If the problem occurs after `convert_pins` is switched on, then go back to the previous step, and switch on each of the option one by one together with the `convert_pins` option as a pair (i.e.: both options together at the same time).

Symlib Data Format

The Symlib format is a text file format that defines schematic symbol shapes. Symlib files are used

by the RTLvision PRO tool (see the Read Dialog or use cmdline option “-symlib”). RTLvision PRO scans the symlib files in order to get symbol shape information.

Symlib files can either be created manually or by a converter program such as cadence2symlib (see Symlib Utilities for an overview).

Overview

Each Symlib file stores a set of symbol shapes, usually customer shapes (but built-in shapes and Boolean equations are also supported). This document describes:

- The Symlib File Structure
  - The File Header Line
  - The File Index Section
  - The DEF Symbol Definition
    - The Pin and PinBus Definition
    - SubPin options
    - Pin Permutation
    - Display location for attributes
    - Display location for pin attributes
    - Constant text
    - Symbol Properties
    - Graphic Definitions: Arc, Path, Fpath
    - Placement Hints
    - Color Settings (boxcolor, fillcolor)
    - Symbol Adjustments: autoalign, scalenow, bboxnow
    - Define Function
- Symbol Definition Details
  - Symbol Pin Directions.
  - The Polygon Shaped Pins/PinBuses
  - Builtin Shapes and Boolean Equations - alternatives to DEF custom shapes.

The Symlib File Structure

A symlib file consists of: (a) one `symlib` header line, (b) a set of `symref` index lines, and (c) a set of `symbol` lines, defining DEF customer shapes (please check out built-in shapes or Boolean equations as alternatives).

The Symlib File is ASCII line-oriented; each entry ends with a newline character, but the lines can be concatenated with a "\" (backslash) character. Comment lines start with a "#" character.

Here is a small example:

header line	`symlib 1.5 noname 4 16 25`

file index	`symref 219 and2`
	`symref 626 xor3`
	`...`

port symbols	`symio ipin in * *`
	`symio gnd pg0 * *`
	`...`

symbol entries	`symbol and2 * DEF \`
	`pin Y out -loc 50 0 40 0 \`
	`pinattrdsp @name -cr 50 -5 8i \`
	`pin A in -loc -10 -10 0 -10 \`
	`pinattrdsp @name -cl -10 -15 8i \`
	`pin B in -loc -10 10 0 10 \`
	`pinattrdsp @name -cl -10 5 8i \`
	`permute all_inputs \`
	`path 25 15 40a 0a 25 -15 0 -15 0 15 25 15 \`
	`attrdsp @name -ll 5 -22 12 \`
	`attrdsp @cell -cl 5 22 10i`
	`symbol xor3 * DEF \`
	`pin C in -loc -10 20 0 20 \`
	`...`

The File Header Line

The header line starts with “symlib”, followed by (1) the version number, (2) the symlib name (unused), (3) the number of symref entries in the index (that is also the number of symbol entries), (4) the character offset to the symbol name in each symref line, (5) the length of each symref line, and (6) the options. For example:

```
symlib 1.5 noname 4 16 25 i
```

The option "i" (6th argument) makes the name matching case-insensitive (for symbol-name and view-name and symbol pin names); this option is only available in symlib version >= 1.2.
The SpiceVision, GateVision, RTLvision, and StarVision tools additionally support the option 'g' to define that glob style pattern can be used for symbol-name matching; this option is only available in symlib version >= 1.4.
The header line as well as the file index is usually automatically created/updated by a converter program or by Symbol Viewer/Editor.

The File Index Section

The file index consists of a set of `symref` lines, one per symbol entry. It is used to get fast access to the required symbols. Each `symref` line defines the start position of a symbol definition and the symbol name. This *start position* is the lseek position, that is the byte count within the file.

All `symref` lines have a fixed common line width - they are padded with trailing blanks. The line width is defined by the header line. For example:

```
symref      219 and2
symref      626 xor3
```

The Port Symbols

The port symbol part consists of a set of `symio` lines. They define rules for user defined port and power/ground symbols. Each line references a symbol shape, which will replace the builtin symbol if the following three pattern match. The first pattern is one of the types

- * matches for all types
- in matches input port symbols
- out matches output port symbols
- inout matches inout port symbols
- pg+ matches power symbols
- pg- matches negpower symbols
- pg0 matches ground symbols

The second pattern matches the port (for in,out,inout) or net (for pg+, pg-, pg0) name. The third pattern matches the module name. Both may contain glob style pattern.

The DEF Symbol Definition

The last section in the Symlib File consists of a set of "symbol" lines, each defines one symbol shape, either a `DEF` customer shape (as described in this chapter) or a Builtin Shapes or a Boolean Equations; the syntax for them is similar, but described in an extra chapter below.

Each symbol shape is defined in one line (with some optional continuous lines). The symbol definition list is actually a tcl-style list of words. Special character in the words (like spaces and some punctuation characters) must be protected according to the tcl rules by surrounding braces **{** ... **}**.

In each symbol line, the first words identify the symbol and the word #3 identifies the symbol type, here a DEF. Each sub-keyword (the first sub-keyword is at word #4) defines a symbol item with a fixed number of subsequent words that belong to that item (the documentation of the symbol items follows). The Symbol Definition for a DEF customer shape has this format:

word count	meaning	example
0	keyword: `symbol`	symbol
1	symbol name	and2
2	symbol view name (or wildcard=*)	*
3	keyword: `DEF`	DEF
4	a new sub-keyword, one of `pin`, `pinBus`, `permute`, `attrdsp`, `pinattrdsp`, `text`, `pintext`, `fillcolor`, `pinfillcolor`, `arc`, `path`, `fpath`, `prop`, `place`, `boxcolor`, `autoalign`, `scalenow`, `bboxnow`, `func`	pin
5	... (depends on previous sub-keyword)	
...	...	

Pin Definition: `pin`

A symbol Pin Entry requires 7 more arguments:

word count	meaning	example
0	keyword: `pin`	pin
1	pin name	O
2	pin direction, (see below for supported values)	out
3	pin option-keyword: `-loc` (see also: polygons)	-loc
4	pin connect-location x-coord	50
5	pin connect-location y-coord	0
6	pin stub end location (inside) x-coord	40
7	pin stub end location (inside) y-coord	0

The order of the pins within a symbol does not matter, the mapping to the EXT symbol is done by pin name (usually case-sensitive, unless the option "i" is specified). Additional pins do not cause any problem - they are always unconnected, of course. Missing pins cause a warning message and the missing pins are marked by the hidden flag, to disable the routing of the connected net to this symbol.

Note: The 4th and 5th argument (pin connect-location) should be a multiple of the schematic grid; the default is 10.

Pin Bus Definition: `pinBus`

A symbol PinBus Entry requires 8+width more arguments (except for the gwidth syntax - see below). Although this entry defines the pin width and the subpin names, that information is ignored when the symbol shape is mapped to a netlist-level symbol statement (symbol type = EXT); this means, the netlist-level symbol object overwrite bus details. The order of the pinBuses within a symbol does not matter, the mapping is done by pinBus name - analog to the pin.

word count	meaning	example
0	keyword: `pinBus`	pinBus
1	pinBus name	DATA(3:0)
2	pinBus direction (see below for supported values)	in
3	pinBus width	4
4	pinBus's subpin #0 (MSB)	DATA(3)
5	pinBus's subpin #1	DATA(2)
6	pinBus's subpin #2	DATA(1)
7	pinBus's subpin #3 (LSB)	DATA(0)
8	pinBus option-keyword: `-loc (see also: polygons)	-loc
9	pinBus connect-location x-coord	50
10	pinBus connect-location y-coord	0
11	pinBus stub end location (inside) x-coord	40
12	pinBus stub end location (inside) y-coord	0

Below are two alternative ways to define a pinBus using the gwidth and range syntax, respectively. This is the preferred syntax for regular bus names, where sub-names can be derived from the bus name: The column "example: gwidth" is equivalent to the example above. The column "example: range" differs from above example in such as the sub-script range is not part of the pinBus name; instead, the range is given as an extra argument defining the bus width and the sub-names' sub-scripts.

word count	meaning	example: gwidth	example: range
0	keyword: `pinBus`	pinBus	pinBus
1	pinBus name	DATA(3:0)	DATA
2	pinBus direction (see below for supported values)	in	in
3	pinBus g_width_ or range (auto-generate sub-names)	g4	(3:0)
8	pinBus option-keyword: `-loc` (see also: polygons)	`-loc`	`-loc`
9	pinBus connect-location x-coord	50	50
10	pinBus connect-location y-coord	0	0
11	pinBus stub end location (inside) x-coord	40	40
12	pinBus stub end location (inside) y-coord	0	0

SubPin Options: `pinsopt`

A symbol SubPin Option Entry defines deviating options for one subpin of the previous pinBus Entry. Usually, the subpins inherit the options from their pinBus Entry, but can be overwritten on a

per-subpin basis. The subpin is addressed by order number, starting with index 0 for the first subpin (MSB). It requires 2 more arguments:

word count	meaning	example
0	keyword: `pinsopt`	pinsopt
1	the zero-based subpin order number	0
2	the subpin option(s)	in.neg.clk

Notes: * Directional options like **`top`** or **`right`** are ignored, the pinBus direction always applies to all its subpins. * This entry must be repeated for every subpin that differs in options from its pinBus entry.

Pin Permutation: `permute`

A symbol Permute Entry requires 1 more argument:

word count	meaning	example
0	keyword: `permute`	permute
1	the permutation definition	all_inputs

The permutation definition is either a keyword, one of “all_inputs”, “all_outputs” or “all” - or a pin permutation definition list by name like “(A,B,D)” or by index numbers like “#(0,1,3)”.

Permutation Syntax

Each permutation group is defined by a list of symbol pin names (or index numbers) or nested sub-groups, the elements are separated by commas. A permutable group is enclosed in parentheses `()`, a non-permutable group is enclosed in brackets `[]`. The top-most group is considered non-permutable, unless it is explicitly enclosed in parentheses `()`.

Please note: the keywords "all_inputs" and "all_outputs" only work, if all input pins (or all output pins) are located on the same side (left, right, top or bottom) - see also symbol pin directions.

`a,b,c`	a,b and c are **not** permutable.
`[a,b,c]`	same as above; the braces at the top group are optional
`(a,b,c)`	a,b and c are permutable.
`((a,b),(c,d))`	a and b can be permuted, c and d can be permuted, the two groups can be permuted
`([a,b],[c,d])`	only the two groups "a,b" and "c,d" can be permuted.
`([a,(b,c)],[d,(e,f)])`	defines two permutable sets, abc and def, where bc and ef may also be permuted
`([a,(b,c)][d,(e,f)])`	same as above; the commas preceding a `(` or `[` are optional.

Permutation Syntax (by Pin Number)

If the permute string starts with a # character, then pin index numbers (instead of pin names) are expected. The parentheses, brackets and commas "()[]," are used as above with identical meaning. The pin index numbers start at zero. For example, "#(0,1,2,3)" defines a permutation group of 4 pins (the index refers to the definition order of the pins).

Display Location for Attributes: `attrdsp` and `pinattrdsp`

A symbol (or symbol-pin) Attribute Display Entry defines a display location for instance (or instance-pin) attributes. It requires 5 more arguments:

word count	meaning	example
0	keyword: `attrdsp` or `pinattrdsp`	attrdsp
1	attribute name	@name
2	text justification, one of `-ll`, `-lc`, `-lr`, `-cl`, `-cc`, `-cr`, `-ul`, `-uc`, `-ur` or `-ll.v`, `-lc.v`, `-lr.v`, `-cl.v`, `-cc.v`, `-cr.v`, `-ul.v`, `-uc.v`, `-ur.v`	-ll
3	display-location x-coord	5
4	display-location y-coord	-22
5	font size - optionally followed with suffix flag characters "i", "c" or "r" (no space between number and flag characters).	12

Word #2 (the justification) defines:

the origin of the displayed text - to be at lower-left, lower-center, lower-right, ..., etc. The display orientation is either horizontal (default) or vertical if `.v` is added. Horizontal text means: rotated 90 degrees counterclockwise. Justification will be applied before rotation, e.g. `-ll.v` will display the attribute vertically, growing from the display-location (words #3 and #4) towards top; the attribute's bounding box's lower-right corner will meet the display-location.

Word #5 (the font size) defines optional flags:

"**i**" means, this is an *invisible* attribute.
"**c**" defines an *case-insensitive* name lookup for finding the matching instance attributes. Special attributes like @name, @cell, @attr, etc cannot be flagged to be *case-insensitive*.
"**r**" defines a *non-rotating* attribute (or text) that means it is not updated when an instance is rotated.

Each `pinattrdsp` entry refers to the previous `pin` or `pinBus`.

Display Location for Constant Text: `text` and `pintext`

A symbol (or symbol-pin) Text Entry defines a text label and a display location for it. It requires the same 5 arguments as attrdsp above, except argument #1 is the text label string instead of the attribute name.

The text can also span several lines (**multi-line** text). Use **\n** in the text string to force the beginning of a new line. **\n** itself will not be displayed. For example:

```
text "This component\ndemonstrates\nmultiline attributes." -ul 40 50 12
```

Symbol Properties: `prop` **and** `pinprop`

A symbol (or symbol-pin) Property Entry defines an arbitrary name-value-pair like an attribute, but at a Symbol or Symbol-Pin instead of Instance or Pin. These properties are never displayed and they do not derive their values to instances/pins of the symbol. It requires 2 more arguments:

word count	meaning	example
0	keyword: `prop` or `pinprop`	prop
1	name	ident
2	value	5812

Arc Definition: `arc` **(Deprecated)**

A symbol Arc Entry defines a (unfilled) circle or a circular arc. If the start and end locations are identical, then it is a circle, else it is an arc. As of Symlib format 1.5 arc bows can also occur within a `path` element - effectively rendering the `arc` keyword obsolete; but the Arc entry is still accepted for backward compatibility. The Arc entry requires 6 more arguments:

word count	meaning	example
0	keyword: `arc`	arc
1	arc start location x-coord	25
2	arc start location y-coord	15
3	middle location on arc x-coord	40
4	middle location on arc y-coord	0
5	arc end location x-coord	25
6	arc end location y-coord	-15

Polygon Path Definition: `path`

A symbol Path Entry defines a non-closed polygon path. It consists of vertices (x/y pairs) that are automatically terminated by the first non-number argument. The intermediate vertices define corner points. There is one exception: If the vertex coordinates are postfixed by an "**a**", then they define the middle point of a circular arc (from previous vertex to next vertex) - see also `fpath`.

word count	meaning	example
0	keyword: `path`	path
1	1st vertex (start location) x-coord	25
2	1st vertex (start location) y-coord	15
3	2nd vertex x-coord - middle point of intermittent arc	40a
4	2nd vertex y-coord - middle point of intermittent arc	0a
5	3rd vertex (start location) x-coord	25
6	3rd vertex (start location) y-coord	-15
7	4th vertex x-coord	0
8	4th vertex y-coord	-15
9	5th vertex x-coord	0
10	5th vertex y-coord	15
11	6th vertex x-coord	25
12	6th vertex y-coord	15
13	not a number - beyond end of path	attrdsp

Polygon Fill-Path Definition: `fpath`

A symbol Fpath Entry defines a closed polygon path as required for the fillcolor feature. It consists of vertices (x/y pairs) that are automatically terminated by the first non-number argument. The first and last vertex should be the same. The intermediate vertices define corner points. There is one exception: If the vertex coordinates are postfixed by an "**a**", then they define the middle point of a circular arc (from previous vertex to next vertex) - see also `path`. The example below defines two fill-paths for a NAND gate. The second path is actually a filled circle (a filled closed arc):

word count	meaning	example
0	keyword: `fpath`	fpath
1	1st vertex (start location) x-coord	0
2	1st vertex (start location) y-coord	0
3	2nd vertex x-coord	0
4	2nd vertex y-coord	15
5	3rd vertex x-coord	25
6	3rd vertex y-coord	15
7	4th vertex x-coord - middle point of intermittent arc	40a
8	4th vertex y-coord - middle point of intermittent arc	0a
9	5th vertex x-coord	25

word count	meaning	example
10	5th vertex y-coord	-15
11	6th vertex x-coord	0
12	6th vertex y-coord	-15
13	7th vertex x-coord	0
14	7th vertex y-coord	0
0	end of fpath, start of another fpath	fpath
1	1st vertex (start location) x-coord	40
2	1st vertex (start location) y-coord	0
3	2nd vertex x-coord	48a
4	2nd vertex y-coord	0a
5	3rd vertex x-coord	40
6	3rd vertex y-coord	0
15	another keyword - beyond end of fpath	???

Placement Hint: `place`

A symbol Place Entry can be used to give hints for the placement of special symbols like latches or IO buffers or to define a port symbol. The Place entry requires 1 more argument:

word count	meaning	example
0	keyword: `place`	place
1	one of “latch”, “in_buffer”, “out_buffer”, “io_buffer” or, to specify a port symbol: “port” or, to specify a power/ground symbol: “power_stub”, “ground_stub”	latch

Color Setting #1: `boxcolor`

A symbol Boxcolor Entry can be used to define an alternate color for the symbol body. The Boxcolor entry requires 1 more argument:

word count	meaning	example
0	keyword: `boxcolor`	boxcolor
1	a number between 0 and 15	2

Color Setting #2: `fillcolor` **and** `pinfillcolor`

A symbol (or symbol-pin) Fillcolor Entry can be used to define an alternate background color (fillcolor) for the symbol body (or polygon shaped symbol-pin). In order to see a background color for a particular area an appropriate closed shape using fpath and/or arc (-locfpath and/or -locarc) must be defined. The Fillcolor Entry requires 1 more argument:

word count	meaning	example
0	keyword: `fillcolor`	fillcolor
1	a number between 1 and 15	1

Automatic Symbol Alignment: `autoalign`

A symbol Autoalign entry can be used to immediately transform all coordinates of the symbol shape, so that the symbol origin (0,0) is placed at the upper left corner of the symbol shape. To keep the symbol on grid the y origin is not always at the top edge but snapped to grid.

As a result all components of these symbols will be lined up (left side aligned) in each component column of the schematic page. Autoalignment is only supported for DEF and EXT symbols. All builtin symbols are already defined with a left side alignment.

word count	meaning	example
0	keyword: `autoalign`	autoalign

Automatic Symbol Scaling: `scalenow`

A symbol Scalenow entry can be used to immediately scale all coordinates of the symbol shape by the given factor. Scalenow is only supported for DEF symbols. Since the symbol is scaled immediately after reading this keyword, it is required that this keyword is placed after the last declaration containing coordinates.

word count	meaning	example
0	keyword: `scalenow`	scalenow
1	the factor	1.5

Enlarge Symbol Bounding-Box: `bboxnow`

A symbol Bboxnow entry can be used to immediately enlarge the symbol's bounding-box by the given coordinates. Bboxnow is only supported for DEF symbols.

word count	meaning	example
0	keyword: `bboxnow`	bboxnow

word count	meaning	example
1	left coord	-20
2	top coord	-80
3	right coord	200
4	bottom coord	80

Define Symbol Function: `func`

A symbol function can be defined (only needed if different from the symbol shape type). If the symbol type is DEF or EXT (user-defined shape), then a function for that user-defined symbol may be specified to control some internal functions. Not defining a function usually has very little impact. Non-simple shape types (like those for combi gates with a number sequence as part of the shape type) are not supported and silently handled as UNDEF.

word count	meaning	example
0	keyword: `func`	func
1	One of the builtin device symbol shapes, or simple builtin symbol shapes, or UNDEF.	NMOS

Symbol Pin Directions

The pin direction (word #2 of the pin or pinBus definition) must specify the graphical direction. It is a dot-separated list of keywords; this list must not be empty, and may contain a combination of electrical direction and graphical direction, e.g. `input.top`.

For symbol shape definitions in Symlib Files the electrical direction can be omitted.

Direction	Description
`left`	The pin will be connected from the left side.
`right`	The pin will be connected from the right side.
`top`	The pin will be connected from the top.
`bottom` or `bot`	The pin will be connected from the bottom.
`unknown`	The pin connect direction is unknown and will be determined by (a) the graphical orientation of the pin stub or if not possible by (b) the distance between the gravity center and the pin location.

Builtin Shapes in the Symlib File

In addition to the DEF custom shapes (described above), the Symlib Files also supports Builtin Shapes. When Builtin Shapes are defined in the symlib files then most builtin types do not require to have the pins listed, however, they can - to specify additional pin options like **neg**, **clk**, **top** or **bot**, etc - if needed. However, the builtin combi-gates and PLA need all (input) pins to be listed (to match the pin order).

A mapping of symbol names to Builtin Shapes can be defined as simple as in this Symlib Files example:

header line	`symlib 1.5 example ...`
file index	...

symbol entries	`symbol invx1 * INV`
	`symbol invx4 * INV`
	`symbol nand2x1 * NAND`
	`symbol nor2x1 * NOR`
	`symbol xnor2 * XNOR`
	`symbol xor2 * XOR`
	`symbol mux2 * MUX pin S in.top`
	`symbol lat * GEN fillcolor 2 pin G in.neg.clk \`
	`pin Set in.top pin Res in.bot`

Builtin Combi-Gate Shapes and PLA

Builtin shapes for combi-gates (AO, OA, AX, etc) as well as for PLA need to have all pins defined, because the pin order matters. The pin order from the symlib file entry overwrite the original symbol.

If the pin names as defined in the symlib file entry don't match to the original symbol, then the symlib file entry is dropped and a BOX is used instead.

header line	`symlib 1.5 example ...`
file index	...
symbol entries	`symbol ao1 * AO(21) pin A in pin B in pin C in.neg`

Boolean Equations in the Symlib File

A special way for defining a Builtin Shapes is using a Boolean Equations, as explained in the Builtin Symbol Shapes document.

A mapping from a symbol name to Boolean Equation (resulting in a Builtin Shapes) can be defined as simple as in this Symlib File example:

header line	`symlib 1.5 example ...`
file index	...
symbol entries	`symbol xor * BOOL Y a^b^c  fillcolor 1` `symbol xa1 * BOOL Y (a^b)*c fillcolor 1`

Polygon Shaped Pins and PinBuses

Starting with Symlib format version 1.4, the `pin` and `pinBus` keywords alternatively support defining arbitrary polygon shapes. Depending on the desired polygon(s) you have to replace the pin option `-loc` by one of: `-locarc` (deprecated), `-locpath` or `-locfpath`. To define a **multi**-part polygon, you can repeat these keywords multiple times (including the mandatory connect location - see below for details). These polygon shaped pins are implicitly excluded from Pin Permutation.

Please note: The first point in all three polygon pin options below is the connect-location. This defines the start/end point of a connecting net wire for that pin. The connect-location is **not** part of the polygon; it can be placed arbitrarily. This, however, may confuse the internal guessing for the net connect direction. That's why you probably want to add the desired pin direction (`left, right, top, bot`) explicitly.

Pin Option: `-locarc` (Deprecated)

Similar to the `arc` keyword, this polygon pin option defines a single (unfilled) circle or circular arc. If the start and end locations are identical, then it is a circle, else it is an arc. Note, as of Symlib format 1.5 arcs can also occur within a `-locpath` element.

word count	meaning	example
0	polygon pin/pinBus option keyword: `-locarc`	-locarc
1	pin/pinBus connect-location x-coord	-15
2	pin/pinBus connect-location y-coord	45
3	arc start location x-coord	0
4	arc start location y-coord	45

word count	meaning	example
5	middle location on arc x-coord	-30
6	middle location on arc y-coord	45
7	arc end location x-coord	0
8	arc end location y-coord	45

Pin Option: `-locpath`

Similar to the `path` keyword, this polygon pin option defines a (unfilled) non-closed polygon path. It consists of vertices (x/y pairs) that are automatically terminated by the first non-number argument. The intermediate vertices define corner points. There is one exception: If the vertex coordinates are postfixed by an "**a**", then they define the middle point of a circular arc (from previous vertex to next vertex) - see also `-locfpath`.

word count	meaning	example
0	polygon pin/pinBus option keyword: `-locpath`	-locpath
1	pin/pinBus connect-location x-coord	27
2	pin/pinBus connect-location y-coord	-10
3	1st vertex (start location) x-coord	20
4	1st vertex (start location) y-coord	-10
5	2nd vertex x-coord	25
6	2nd vertex y-coord	-10
7	3rd vertex x-coord - middle point of intermittent arc	30a
8	3rd vertex y-coord - middle point of intermittent arc	-10a
9	4th vertex x-coord	25
10	4th vertex y-coord	-10
...	...	...
...	not a number - beyond end of polygon path	pinattrdsp

Pin Option: `-locfpath`

Similar to the `fpath` keyword, this polygon pin option defines a closed polygon path as required for the fillcolor or @fillcolor features. It consists of vertices (x/y pairs) that are automatically terminated by the first non-number argument. The first and last vertex should be the same. The intermediate vertices define corner points. There is one exception: If the vertex coordinates are postfixed by an "**a**", then they define the middle point of an intermittent circular arc (from previous vertex to next vertex) - see also `-locpath`.

word count	meaning	example
0	polygon pin/pinBus option keyword: `-locfpath`	-locfpath
1	pin/pinBus connect-location x-coord	-15
2	pin/pinBus connect-location y-coord	15
3	1st vertex (start location) x-coord	10
4	1st vertex (start location) y-coord	56
5	2nd vertex x-coord	10
6	2nd vertex y-coord	50
7	3rd vertex x-coord	20
8	3rd vertex y-coord	50
9	4th vertex x-coord	20
10	4th vertex y-coord	62
11	5th vertex x-coord	10
12	5th vertex y-coord	62
13	6th vertex x-coord - middle point of intermittent arc	4a
14	6th vertex y-coord - middle point of intermittent arc	56a
15	7th vertex x-coord	10
16	7th vertex y-coord	50
…	…	…
…	another keyword - beyond end of polygon fpath	pinattrdsp

The Graphical User Interface API

This Document describes the API (Application Programming Interface) to control the GUI (Graphical User Interface). In combination with the Database API the user can write its own Tcl scripts as extension to RTLvision PRO.

gui analogWave

APIs to interact with the AnalogWave window.

gui analogWave closeAllFiles

Close all files.

Usage:

```
gui analogWave closeAllFiles ?-window windowValue?
```

Parameters:

`-window windowValue (optional default is NULL)`

Path or name of a AnalogWave window, "DEFAULT" or "" for the default AnalogWave window.

Example:

```
gui analogWave closeAllFiles
```

gui analogWave closeAllTabs

Close all tabs inside the Analog Waveform window.

Usage:

```
gui analogWave closeAllTabs ?-window windowValue?
```

Parameters:

`-window windowValue (optional default is NULL)`

Path or name of a AnalogWave window, "DEFAULT" or "" for the default AnalogWave window.

Example:

```
gui analogWave closeAllTabs
```

gui analogWave closeFile

Close the specified file.

Usage:

```
gui analogWave closeFile ?-window windowValue? fileName
```

Parameters:

`-window windowValue (optional default is NULL)`
: Path or name of a AnalogWave window, "DEFAULT" or "" for the default AnalogWave window.

`fileName`
: The name of the file to be closed.

Example:

```
gui analogWave closeFile $fileName
```

gui analogWave getCurrentPlotName

Return the name of the plot currently visible.

Usage:

```
gui analogWave getCurrentPlotName ?-window windowValue?
```

Parameters:

`-window windowValue (optional default is NULL)`
: Path or name of a AnalogWave window, "DEFAULT" or "" for the default AnalogWave window.

Example:

```
gui analogWave getCurrentPlotName
```

gui analogWave getCurveNames

Return the curve names for the given section. The curves are the signals which can be loaded into a plot.

Usage:

```
gui analogWave getCurveNames ?-sectionName sectionNameValue? ?-window windowValue?
```

Parameters:

`-sectionName sectionNameValue (optional default is "")`

Specify the section name which corresponds to the file name. In case no section name is specified the first section is used.

`-window windowValue (optional default is NULL)`

Path or name of a AnalogWave window, "DEFAULT" or "" for the default AnalogWave window.

Example:

```
gui analogWave getCurveNames
```

gui analogWave getPlotNames

Return a list of all plot names currently existing in the AnalogWave window.

Usage:

```
gui analogWave getPlotNames ?-window windowValue?
```

Parameters:

`-window windowValue (optional default is NULL)`

Path or name of a AnalogWave window, "DEFAULT" or "" for the default AnalogWave window.

Example:

```
gui analogWave getPlotNames
```

gui analogWave getSectionNames

Return the section names of the Analog Waveform window. Each section corresponds to one specific file.

Usage:

```
gui analogWave getSectionNames ?-window windowValue?
```

Parameters:

`-window windowValue (optional default is NULL)`

Path or name of a AnalogWave window, "DEFAULT" or "" for the default AnalogWave window.

Example:

```
gui analogWave getSectionNames
```

gui analogWave getWidget

Get the path of the BeSpice widget.

Usage:

```
gui analogWave getWidget ?-window windowValue?
```

Parameters:

`-window windowValue (optional default is NULL)`

Path or name of a AnalogWave window, "DEFAULT" or "" for the default AnalogWave window.

Example:

```
gui analogWave getWidget
```

gui analogWave hideWindow

Hide the default AnalogWave window.

Usage:

```
gui analogWave hideWindow
```

Parameters:

Example:

```
gui analogWave hideWindow
```

gui analogWave loadCurve

Show a curve in a plot.

Usage:

```
gui analogWave loadCurve ?-clear? ?-plotName plotNameValue? ?-sectionName sectionNameValue? ?-window windowValue? curveName
```

Parameters:

`-clear (optional)`

Remove all other curves in the target plot.

`-plotName plotNameValue (optional default is "")`

If no plot name is given, the curve is added to the last plot. In case a plot name is defined, the plot is created if necessary and the curve added.

`-sectionName sectionNameValue (optional default is "")`

Specify the section name which corresponds to the file name. In case no section name is specified the first section is used.

`-window windowValue (optional default is NULL)`

Path or name of a AnalogWave window, "DEFAULT" or "" for the default AnalogWave window.

`curveName`

Specify the curve name to be loaded. Curves in sub-sections are separated by a dot, e.g. 'xopamp.v2'.

Example:

```
gui analogWave loadCurve $curveName
```

gui analogWave loadFile

Load a Spice simulation output file.

Usage:

```
gui analogWave loadFile ?-window windowValue? fileName
```

Parameters:

`-window windowValue (optional default is NULL)`
: Path or name of a AnalogWave window, "DEFAULT" or "" for the default AnalogWave window.

`fileName`
: The name of the file to be loaded.

Example:

```
gui analogWave loadFile $fileName
```

gui analogWave setCurrentPlot

Make the given existing plot visible.

Usage:

```
gui analogWave setCurrentPlot ?-window windowValue? plotName
```

Parameters:

`-window windowValue (optional default is NULL)`
: Path or name of a AnalogWave window, "DEFAULT" or "" for the default AnalogWave window.

`plotName`
: The name of the plot.

Example:

```
gui analogWave setCurrentPlot $plotName
```

gui analogWave showOids

Show the curves corresponding to the oids. Multiple oids are shown in the same plot.

Usage:

```
gui analogWave showOids ?-window windowValue? oidList
```

Parameters:

`-window windowValue (optional default is NULL)`
: Path or name of a AnalogWave window, "DEFAULT" or "" for the default AnalogWave window.

`oidList`
: List of oids.

Example:

```
gui analogWave showOids $oidList
```

gui analogWave showWindow

Show the default AnalogWave window.

Usage:

```
gui analogWave showWindow
```

Parameters:

Example:

```
gui analogWave showWindow
```

gui analogWave zoomFit

Reset the zoom.

Usage:

```
gui analogWave zoomFit ?-plotName plotNameValue? ?-window windowValue?
```

Parameters:

`-plotName plotNameValue (optional default is "")`

Set the target plot. If not defined, the current plot will be the target.

`-window windowValue (optional default is NULL)`

Path or name of a AnalogWave window, "DEFAULT" or "" for the default AnalogWave window.

Example:

```
gui analogWave zoomFit
```

gui analogWave zoomIn

Zoom in one step both in the x and y direction.

Usage:

```
gui analogWave zoomIn ?-plotName plotNameValue? ?-window windowValue?
```

Parameters:

`-plotName plotNameValue (optional default is "")`

Set the target plot. If not defined, the current plot will be the target.

`-window windowValue (optional default is NULL)`

Path or name of a AnalogWave window, "DEFAULT" or "" for the default AnalogWave window.

Example:

```
gui analogWave zoomY
```

---

gui analogWave zoomOut

Zoom out one step both in the x and y direction.

Usage:

```
gui analogWave zoomOut ?-plotName plotNameValue? ?-window windowValue?
```

Parameters:

`-plotName plotNameValue (optional default is "")`
: Set the target plot. If not defined, the current plot will be the target.

`-window windowValue (optional default is NULL)`
: Path or name of a AnalogWave window, "DEFAULT" or "" for the default AnalogWave window.

Example:

```
gui analogWave zoomY
```

gui analogWave zoomX

Zoom in one step in the x direction.

Usage:

```
gui analogWave zoomX ?-plotName plotNameValue? ?-window windowValue?
```

Parameters:

`-plotName plotNameValue (optional default is "")`
: Set the target plot. If not defined, the current plot will be the target.

`-window windowValue (optional default is NULL)`
: Path or name of a AnalogWave window, "DEFAULT" or "" for the default AnalogWave window.

Example:

```
gui analogWave zoomX
```

gui analogWave zoomY

Zoom in one step in the y direction.

Usage:

```
gui analogWave zoomY ?-plotName plotNameValue? ?-window windowValue?
```

Parameters:

`-plotName plotNameValue (optional default is "")`
: Set the target plot. If not defined, the current plot will be the target.

`-window windowValue (optional default is NULL)`
: Path or name of a AnalogWave window, "DEFAULT" or "" for the default AnalogWave window.

Example:

```
gui analogWave zoomY
```

gui assertion

APIs to interact with the Assertion window.

gui assertion hideWindow

Hide the Assertion window.

Usage:

```
gui assertion hideWindow
```

Parameters:

Example:

```
gui assertion hideWindow
```

gui assertion showWindow

Show the Assertion window.

Usage:

```
gui assertion showWindow
```

Parameters:

Example:

```
gui assertion showWindow
```

gui attribute

The tool maintains module-based and tree-based (flat) attributes.

gui attribute changed

Inform the GUI that attributes in the database have changed.

Usage:

```
gui attribute changed ?-enableSaveSpice?
```

Parameters:

`-enableSaveSpice` `(optional)`

If this switch is set then the Save Spice button in the toolbar and in the main menu will be enabled.

Example:

```
set oidList {
    {inst TOP U2 m1}
    {inst TOP U2 m2}
}
set cnt 0
foreach oid $oidList {
    $db flatattr $oid set cnt=[incr cnt]
}

##
# Create display attributes to make the $cnt attribute
# visible in Schem and Cone.
#
$db attr -db set {@nlv:port=c=$cnt}
$db attr {primitive nch nch} set {@nlv=c=$cnt}
$db attr {primitive pch pch} set {@nlv=c=$cnt}

gui attribute changed
```

gui attribute registerGetCallback

Register a callback to get attribute values from an external source. This callback is used in OID tooltips and in the attributes list of the Infobox.

Usage:

```
gui attribute registerGetCallback callback
```

Parameters:

`callback`

The callback script.

Example:

```
proc getMyAttributes {oid} {
    return {A=1 B=2}
}
gui attribute registerGetCallback "getMyAttributes"
```

gui attribute removeGetCallback

Remove a previously registered callback to get attribute values from an external source.

Usage:

```
gui attribute removeGetCallback callback
```

Parameters:

`callback`
: The callback script.

Example:

```
gui attribute removeGetCallback "getMyAttributes"
```

gui bookmark

APIs to manage bookmarks.

gui bookmark add

Add a new bookmark for the current contents of the given window. Return the new bookmark's name.

Usage:

```
gui bookmark add window ?name?
```

Parameters:

`name (optional)`
: Name of the new bookmark. If no name is given, a unique name is automatically created.

`window`
: Path or name of a window of class Cone, Schem, Source, or Wave.

Example:

```
##
# Add a named bookmark for the default Cone window.
#
set cone [::WindowManager::DefaultWindow Cone]
gui bookmark add $cone "myBookmark"

##
# Add an automatically named bookmark.
#
set bookmark_name [gui bookmark add $cone]
```

gui bookmark changed

Inform the GUI about changed bookmarks.

Usage:

```
gui bookmark changed ?-class classValue?
```

Parameters:

`-class classValue (optional default is NULL)`

Bookmark class that has been changed. Supported classes are "Schem", "Cone", "Source" and "Wave". Omitting the "-class" option means that all bookmark classes have changed.

Example:

```
##
# Schem bookmarks have changed.
#
gui bookmark changed -class "Schem"

##
# All bookmark classes have changed.
#
gui bookmark changed
```

gui bookmark delete

Delete the given class bookmark.

Usage:

```
gui bookmark delete classOrWindow name
```

Parameters:

`classOrWindow`

Bookmark class (must be Schem, Cone, Source, or Wave) or window path.

`name`

Name of the bookmark to delete.

Example:

```
##
# Delete a Cone bookmark.
#
gui bookmark delete Cone "myNewBookmark"
```

gui bookmark get

Retrieve all class bookmarks.

Usage:

```
gui bookmark get classOrWindow
```

Parameters:

`classOrWindow`

Bookmark class (must be Schem, Cone, Source, or Wave) or window path.

Example:

```
##
# Get all Cone bookmarks.
#
set bookmarks [gui bookmark get Cone]
```

gui bookmark rename

Rename the given class bookmark.

Usage:

```
gui bookmark rename classOrWindow name new_name
```

Parameters:

`classOrWindow`
: Bookmark class (must be Schem, Cone, Source, or Wave) or window path.

`name`
: Name of the bookmark to rename.

`new_name`
: New name.

Example:

```
##
# Rename a Cone bookmark.
#
gui bookmark rename Cone "myBookmark" "myNewBookmark"
```

gui bookmark select

Select/restore the given bookmark in the given window.

Usage:

```
gui bookmark select window name
```

Parameters:

`name`
: Name of the bookmark to select.

`window`
: Path or name of a window of class Cone, Schem, Source, or Wave.

Example:

```
##
# Select/restore a bookmark for the default Cone window.
#
set cone [::WindowManager::DefaultWindow Cone]
gui bookmark select $cone "myBookmark"
```

gui bookmark set

Set all class bookmarks.

Usage:

`gui bookmark set classOrWindow bookmark_list`

Parameters:

`bookmark_list`

List of bookmarks (as returned by a previous call to 'gui bookmark get ...').

`classOrWindow`

Bookmark class (must be Schem, Cone, Source, or Wave) or window path.

Example:

```
##
# Get all Cone bookmarks.
#
set saved_bookmarks [gui bookmark get Cone]

##
# Restore the previously saved Cone bookmarks.
#
gui bookmark set Cone $saved_bookmarks
```

gui clipboard

APIs to interact with the clipboard.

gui clipboard get

Returns Windows/X11 primary selection or clipboard.

Usage:

```
gui clipboard get ?-path pathValue?
```

Parameters:

`-path pathValue (optional default is ".")`
: Specify the window path.

Example:

```
gui clipboard get
```

gui clipboard set

Set the Windows/X11 primary selection and store the text in the clipboard.

Usage:

```
gui clipboard set ?-path pathValue? text
```

Parameters:

`-path pathValue (optional default is ".")`
: Specify the object path.

`text`
: The text which should be stored.

Example:

```
gui clipboard set "test"
```

gui clockdomain

APIs to interact with the clock domain analyzer.

gui clockdomain hideDialog

Hide the Clock Domain Analyzer window.

Usage:

```
gui clockdomain hideDialog
```

Parameters:

No parameters.

Example:

```
gui clockdomain hideDialog
```

gui clockdomain highlight

Highlight the given Clock Domain.

Usage:

```
gui clockdomain highlight ?-color colorValue? ?-domain domainValue?
```

Parameters:

`-color` **`colorValue`** `(optional default is "0")`

Number of the highlight color to use. If domain number is "all" then this will be the first used highlight color.

`-domain` **`domainValue`** `(optional default is "all")`

Number of the clock domain to highlight or "all". Clock domain numbers can be obtained by the "$db cdc" API commands.

Example:

```
gui clockdomain highlight
```

gui clockdomain showDialog

Show the Clock Domain Analyzer window.

Usage:

`gui clockdomain showDialog`

Parameters:

No parameters.

Example:

```
gui clockdomain showDialog
```

gui clockdomain unhighlight

Unhighlight the given Clock Domain.

Usage:

`gui clockdomain unhighlight ?-domain domainValue?`

Parameters:

`-domain domainValue (optional default is "all")`

Number of the clock domain to highlight or "all". Clock domain numbers can be obtained by the "$db cdc" API commands.

Example:

```
gui clockdomain unhighlight
```

gui cone

APIs to interact with the Cone window.

gui cone append

Identical to 'gui cone load' except that the previous contents of the window is not deleted. The objects in oidList will be appended to the Cone window.

Usage:

gui cone append ?-foldModules? ?-highlight? ?-window windowValue? oidList

---

Parameters:

-foldModules (optional)

Fold all module instances.

-highlight (optional)

Highlight the loaded objects using the goto color.

-window windowValue (optional default is NULL)

Path or name of a Cone window, "DEFAULT" or "" for the default Cone window.

oidList

List of objects to append.

Example:

```
gui cone append -highlight $oids
```

gui cone clear

Delete all objects from the given Cone window. The Cone window is re-initialized.

Usage:

gui cone clear ?-window windowValue?

Parameters:

-window windowValue (optional default is NULL)

Path or name of a Cone window, "DEFAULT" or "" for the default Cone window.

Example:

```
gui cone clear
```

gui cone contents

Returns all objects loaded to the given Cone window.

Usage:

```
gui cone contents ?-type typeValue? ?-window windowValue?
```

Parameters:

`-type typeValue` (optional default is "*")

If this parameter is given, the the result is filtered by the given type.

`-window windowValue` (optional default is NULL)

Path or name of a Cone window, "DEFAULT" or "" for the default Cone window.

Example:

```
gui cone contents
```

gui cone customFoldAction

Register a custom procedure for the fold button.

Usage:

```
gui cone customFoldAction ?-window windowValue? callback
```

Parameters:

`-window windowValue` (optional default is NULL)

Path or name of a Cone window, "DEFAULT" or "" for the default Cone window.

`callback`

Name of the callback procedure.

Example:

```
proc myFoldProc {inst} {
    # Do something with the instance to fold.
}
gui cone customFoldAction "myFoldProc"
```

gui cone customMoreAction

Register a custom procedure for the more command (double click).

Usage:

```
gui cone customMoreAction ?-window windowValue? callback
```

Parameters:

`-window windowValue` (optional default is NULL)

Path or name of a Cone window, "DEFAULT" or "" for the default Cone window.

`callback`

Name of the callback procedure.

Example:

```
proc myMoreProc {oid} {
    set oidList {}
    # Use the '$db cone' API to calculate the result.
    # set oidList [$db cone <OPTIONS> $oid]
    return $oidList
}
gui cone customMoreAction -window "Cone" myMoreProc
```

gui cone customUnfoldAction

Register a custom procedure for the unfold button.

Usage:

```
gui cone customUnfoldAction ?-window windowValue? callback
```

Parameters:

`-window windowValue` (optional default is NULL)

Path or name of a Cone window, "DEFAULT" or "" for the default Cone window.

`callback`

Name of the callback procedure.

Example:

```
proc myUnfoldProc {inst} {
    # Do something with the instance to unfold.
}
gui cone customUnfoldAction "myUnfoldProc"
```

gui cone fold

Fold the given modules in the Cone window.

Usage:

```
gui cone fold ?-window windowValue? oidList
```

Parameters:

`-window windowValue (optional default is NULL)`

Path or name of a Cone window, "DEFAULT" or "" for the default Cone window.

`oidList`

List of objects to fold.

Example:

```
set oidList [gui cone contents]
gui cone fold $oidList
```

gui cone isFolded

Check if the fold flag is set at a module.

Usage:

```
gui cone isFolded ?-window windowValue? oid
```

Parameters:

`-window windowValue (optional default is NULL)`

Path or name of a Cone window, "DEFAULT" or "" for the default Cone window.

`oid`

Module OID to check.

Example:

```
set oidList [gui cone contents]
gui cone isFolded [lindex $oidList 0]
```

gui cone isLoaded

Check if one of the given OID in oidList is loaded to the Cone window. If the oidList is empty then this function can be used to check if the Cone window is filled or empty.

Usage:

```
gui cone isLoaded ?-window windowValue? oidList
```

Parameters:

`-window windowValue (optional default is NULL)`

Path or name of a Cone window, "DEFAULT" or "" for the default Cone window.

`oidList`

List of objects to check.

Example:

```
set oidList [gui cone contents]
if {[gui cone isLoaded $oidList]} {
    # Do something
}
```

gui cone load

Load the given objects into the given Cone window. The objects are identified by OIDs; the 'oidList' is a Tcl-list of OIDs. If oidList contains nets, then those nets only display the connections to also loaded instances/ports. In other words, just loading a net (without loading at least one connecting instance or port will result in a blank window). An error is returned, if one of the OIDs can't be found in the database.

Usage:

```
gui cone load ?-foldModules? ?-highlight? ?-window windowValue? oidList
```

Parameters:

`-foldModules (optional)`
: Fold all module instances.

`-highlight (optional)`
: Highlight the loaded objects using the goto color.

`-window windowValue (optional default is NULL)`
: Path or name of a Cone window, "DEFAULT" or "" for the default Cone window.

`oidList`
: List of objects to load.

Example:

```
gui cone load $oids
```

gui cone loadModule

Load a module including the contents to the Cone window.

Usage:

```
gui cone loadModule ?-window windowValue? module
```

Parameters:

`-window windowValue (optional default is NULL)`
: Path or name of a Cone window, "DEFAULT" or "" for the default Cone window.

`module`
: OID of the module to load.

Example:

```
gui cone loadModule $moduleOid
```

gui cone nlv

Get the path of the Nlview widget.

Usage:

```
gui cone nlv ?-window windowValue?
```

Parameters:

`-window windowValue (optional default is NULL)`

Path or name of a Cone window, "DEFAULT" or "" for the default Cone window.

Example:

```
gui cone nlv
```

gui cone openSnapshot

Restore a Cone snapshot file.

Usage:

```
gui cone openSnapshot ?-greyout? ?-validate? ?-window windowValue? fname
```

Parameters:

`-greyout (optional)`

If greyout is set then grey out objects that do not match the loaded design database.

`-validate (optional)`

If validate is set then update the displayed connectivity to the loaded design database.

`-window windowValue (optional default is NULL)`

Path or name of a Cone window, "DEFAULT" or "" for the default Cone window.

`fname`

Name of the snapshot file.

Example:

```
gui cone openSnapshot -window "Cone" "mySnapshot.bm5" -validate
```

gui cone pages

Get the number of schematic pages displayed in the Cone window.

Usage:

```
gui cone pages ?-window windowValue?
```

Parameters:

`-window windowValue (optional default is NULL)`
: Path or name of a Cone window, "DEFAULT" or "" for the default Cone window.

Example:

```
gui cone pages -window "Cone"
```

gui cone regenerate

Regenerate the displayed schematic of the given Cone window.

Usage:

```
gui cone regenerate ?-window windowValue?
```

Parameters:

`-window windowValue (optional default is NULL)`
: Path or name of a Cone window, "DEFAULT" or "" for the default Cone window.

Example:

```
gui cone regenerate
```

gui cone registerAddCallback

Register a callback that is called after all objects of a load or append command have been added to a Cone window. The Cone window's path is appended to the callback.

Usage:

```
gui cone registerAddCallback callback
```

Parameters:

`callback`

The callback script.

Example:

```
proc my_callback {w} {
    gui console print "Some objects have been added to the Cone window: $w"
}

gui cone registerAddCallback my_callback
```

gui cone registerChangedCallback

Register a callback that is called after the contents of a Cone window have change. The Cone window's path is appended to the callback.

Usage:

```
gui cone registerChangedCallback callback
```

Parameters:

`callback`

The callback script.

Example:

```
proc my_callback {w} {
    gui console print "The Cone window has changed: $w"
}

gui cone registerChangedCallback my_callback
```

gui cone registerClearCallback

Register a callback that is called before a Cone window is cleared. The Cone window's path is appended to the callback.

Usage:

```
gui cone registerClearCallback callback
```

Parameters:

`callback`
: The callback script.

Example:

```
proc my_callback {w} {
    gui console print "The Cone window is about to be cleared: $w"
}

gui cone registerClearCallback my_callback
```

gui cone registerZoomChangedCallback

Register a callback that is evaluated every time current zoom factor has changed. Before evaluating the callback, the Cone window identifier and the current zoom factor value are appended. The current zoom factor may also be empty.

Usage:

```
gui cone registerZoomChangedCallback callback
```

Parameters:

`callback`

The callback script to be evaluated.

Example:

```
proc myZoomChanged {w zoomFactor} {
    gui console print "Cone window $w: the new zoom factor is $zoomFactor"
}

# add callback
gui cone registerZoomChangedCallback myZoomChanged

# remove callback
gui cone removeZoomChangedCallback myZoomChanged
```

gui cone remove

Delete the given objects from the given Cone window. An error is returned, if one of the OIDs can't be found in the database.

Usage:

`gui cone remove ?-window windowValue? oidList`

Parameters:

`-window windowValue (optional default is NULL)`

Path or name of a Cone window, "DEFAULT" or "" for the default Cone window.

`oidList`

List of objects to remove.

Example:

```
gui cone remove $oids
```

gui cone removeAddCallback

Remove the previously registered callback script.

Usage:

```
gui cone removeAddCallback callback
```

Parameters:

`callback`
: The callback script.

Example:

```
gui cone removeAddCallback my_callback
```

gui cone removeChangedCallback

Remove the previously registered callback script.

Usage:

```
gui cone removeChangedCallback callback
```

Parameters:

`callback`
: The callback script.

Example:

```
gui cone removeChangedCallback my_callback
```

gui cone removeClearCallback

Remove the previously registered callback script.

Usage:

```
gui cone removeClearCallback callback
```

Parameters:

`callback`
: The callback script.

Example:

```
gui cone removeClearCallback my_callback
```

gui cone removeZoomChangedCallback

Remove the callback previously registered with `gui cone registerZoomChangedCallback ⋯`.

Usage:

`gui cone removeZoomChangedCallback callback`

Parameters:

`callback`
: The callback script to remove.

Example:

```
proc myZoomChanged {w zoomFactor} {
    gui console print "Cone window $w: the new zoom factor is $zoomFactor"
}

# add callback
gui cone registerZoomChangedCallback myZoomChanged

# remove callback
gui cone removeZoomChangedCallback myZoomChanged
```

gui cone saveAs

Save the contents of the Cone window as a Verilog or Spice netlist or as a ZDB binfile.

Usage:

`gui cone saveAs ?-allTopPorts? ?-commonCells? ?-connectHiddenSupplies? ?-`

```
functionImplementations? ?-hierarchical? ?-namedConnectivity? ?-portsForUnconnectedPins? ?-
topName topNameValue? ?-unconnectedModulePorts? ?-visibleHierarchyOnly? ?-window windowValue?
mode fname
```

Parameters:

`-allTopPorts (optional)`

Create all top-level ports.

`-commonCells (optional)`

Create common cells for empty and full modules.

`-connectHiddenSupplies (optional)`

Connect all hidden connections to supply nets.

`-functionImplementations (optional)`

Add function implementations suitable for simulation (only for 'Verilog' type).

`-hierarchical (optional)`

Save as a hierarchical netlist.

`-namedConnectivity (optional)`

Create netlist with named connectivity (only for 'Verilog' type).

`-portsForUnconnectedPins (optional)`

Create ports for unconnected pins.

`-topName topNameValue (optional default is NULL)`

Name of created top module.

`-unconnectedModulePorts (optional)`

Create unconnected module ports.

`-visibleHierarchyOnly (optional)`

Create only visible hierarchy (only for 'Spice' type).

`-window windowValue (optional default is NULL)`

Path or name of a Cone window, "DEFAULT" or "" for the default Cone window.

`fname`

Name of the output file. When using '-' as an output file name in 'Binfile' mode, a ZDB handle is returned instead of writing a file.

`mode`

Output mode: 'Verilog', 'Spice', or 'Binfile'.

Example:

```
##
# Create a hierarchical Verilog netlist with function implementations for
# the current Cone window contents. Save it in the file $fname.
#
gui cone saveAs Verilog $fname -hierarchical -functionImplementations
```

gui cone saveSnapshot

Save a Cone snapshot file.

Usage:

```
gui cone saveSnapshot ?-window windowValue? fname
```

Parameters:

`-window windowValue` (optional default is NULL)

Path or name of a Cone window, "DEFAULT" or "" for the default Cone window.

`fname`

Name of the snapshot file.

Example:

```
gui cone saveSnapshot -window "Cone" "mySnapshot.bm5"
```

gui cone selection

Returns all currently selected objects of the Cone window.

Usage:

```
gui cone selection ?-window windowValue?
```

Parameters:

`-window windowValue` (optional default is NULL)

Path or name of a Cone window, "DEFAULT" or "" for the default Cone window.

Example:

```
gui cone selection
```

gui cone unfold

Unfold the given modules in the Cone window.

Usage:

```
gui cone unfold ?-window windowValue? oidList
```

Parameters:

`-window windowValue (optional default is NULL)`
: Path or name of a Cone window, "DEFAULT" or "" for the default Cone window.

`oidList`
: List of objects to unfold.

Example:

```
set oidList [gui cone contents]
gui cone unfold $oidList
```

gui cone zoom

Changes the zoom factor of the Cone window.

Usage:

```
gui cone zoom ?-window windowValue? factor
```

Parameters:

`-window windowValue (optional default is NULL)`
: Path or name of a Cone window, "DEFAULT" or "" for the default Cone window.

`factor`
: The argument $factor must be "fullfit" or a number ("fullfit" only works if that window is

currently visible).

Example:

```
gui window show "Cone"
gui cone zoom "fullfit"
```

gui connectivityBrowser

APIs to interact with the Connectivity Browser window.

gui connectivityBrowser hideWindow

Hide the Connectivity Browser window.

Usage:

```
gui connectivityBrowser hideWindow
```

Parameters:

Example:

```
gui connectivityBrowser hideWindow
```

gui connectivityBrowser showWindow

Show the Connectivity Browser window.

Usage:

```
gui connectivityBrowser showWindow ?-oid oidValue?
```

Parameters:

`-oid` **`oidValue`** `(optional default is NULL)`

Display this OID.

Example:

```
gui connectivityBrowser showWindow
```

gui console

APIs to interact with the Console window.

gui console hideWindow

Hide the Console window.

Usage:

```
gui console hideWindow
```

Parameters:

Example:

```
gui console hideWindow
```

gui console print

Print a message to the Console window.

Usage:

```
gui console print ?-callback callbackValue? ?-filename filenameValue? ?-line lineValue? ?-type typeValue? message
```

Parameters:

`-callback` **`callbackValue`** `(optional default is NULL)`

Add a binding on the left mouse button to the printed text. The given callback is evaluated if the user clicks on the message.

`-filename` **`filenameValue`** `(optional default is NULL)`

If a filename is specified then a left mouse button binding is added to jump to the file in the Source window.

`-line lineValue` (optional default is NULL)

If a filename and a line number is specified then a left mouse button binding is added to jump to the file and line in the Source window.

`-type typeValue` (optional default is "i")

Specify the severity type.

`message`

The message to print.

Example:

```
gui console print "This is a message"
gui console print  -type e  -filename "myfile.v"  -line 123  "This is an error message
with a file/line"
gui console print  -callback "gui console print clicked"  "This is a clickable
message"
```

gui console showWindow

Show the Console window.

Usage:

`gui console showWindow`

Parameters:

Example:

```
gui console showWindow
```

gui database

Get the Database

gui database changed

Update the GUI after loading or opening a new database.

Usage:

```
gui database changed db
```

Parameters:

`db`

Name of the new database.

Example:

```
set db [zdb new]
# load netlist data with '$db load ...'
gui database changed $db
```

gui database get

Returns the name of the currently loaded design database (or an empty string if no design data is loaded).

NOTE	Only after loading a design the database name is set - see also 'gui database registerChangedCallback'.

Usage:

```
gui database get
```

Parameters:

No parameters.

Example:

```
set db [gui database get]
```

gui database modified

Update the GUI after modifying the currently loaded database.

Usage:

```
gui database modified
```

Parameters:

No parameters.

Example:

```
gui database modified
```

gui database populated

Update the GUI after populating the currently loaded database.

Usage:

```
gui database populated
```

Parameters:

No parameters.

Example:

```
gui database populated
```

gui database registerChangedCallback

Register a procedure to be executed every time a new database got connected to the GUI. The database handle is appended to the callback before executing. This handle is identical to the return value of 'gui database get' at that time.

NOTE	The registered procedure is called before the schematic is rendered. If the registered Userware procedure accesses the Schem window then 'gui schem setCurrentModule' needs to be called before.

Usage:

```
gui database registerChangedCallback callback
```

Parameters:

`callback`

Name of the callback procedure to be called.

Example:

```
proc myFunc {db} {
    if {$db eq ""} {
        return
    }
    $db foreach top top {
        $db foreach inst $top inst {
            # Do something with $inst
        }
    }
}
gui database registerChangedCallback "myFunc"
```

gui database registerDesignReadyCallback

Register a callback procedure to be executed after the design is ready (all design files have been read). This callback is called only after loading design files which were given on the command line.

NOTE	The registered procedure is called before the schematic is rendered. If the registered procedure accesses the Schem window then 'gui schem setCurrentModule' needs to be called before.

Usage:

```
gui database registerDesignReadyCallback callback
```

Parameters:

`callback`

Name of the callback procedure to be registered.

Example:

```
proc myProc {} {
    set db [gui database get]
    # Do something with $db
}
gui database registerDesignReadyCallback "myProc"
```

gui database removeChangedCallback

Remove a previously registered callback procedure.

Usage:

`gui database removeChangedCallback callback`

Parameters:

`callback`
: Name of the callback procedure to be removed.

Example:

```
gui database removeChangedCallback "myFunc"
```

gui database removeDesignReadyCallback

Remove a previously registered callback procedure.

Usage:

`gui database removeDesignReadyCallback callback`

Parameters:

`callback`
: Name of the callback procedure to be removed.

Example:

```
gui database removeDesignReadyCallback "myProc"
```

gui database runAndRegisterChangedCallback

If there already is a database, immediately run the callback. Furthermore, register the callback to be executed every time a new database got connected. The database name is appended to the $callback before executing. This pointer is identical to the return value of GetDataBase at that time. If there already is a database, immediately run the callback before registering it.

NOTE	The registered procedure is called before the schematic is rendered. If the registered procedure accesses the Schem window then 'gui schem setCurrentModule' needs to be called before.

Usage:

```
gui database runAndRegisterChangedCallback callback
```

Parameters:

`callback`

Name of the callback procedure.

Example:

```
proc myFunc {db} {
    if {$db == ""} {
        return
    }
    $db foreach top top {
        $db foreach inst $top inst {
            # Do something with $inst
        }
    }
}
gui database runAndRegisterChangedCallback "myFunc"
```

gui database runOrRegisterChangedCallback

If there already is a database, immediately run the callback. Otherwise, register the callback to be executed every time a new database got connected. The database name is appended to the $callback before executing. This pointer is identical to the return value of GetDataBase at that time.

The return value is 0 if the procedure was executed immediately and 1 if the command was

registered for database changed callback.

Usage:

```
gui database runOrRegisterChangedCallback callback
```

Parameters:

`callback`
: Name of the callback procedure.

Example:

```
proc myFunc {db} {
    if {$db == ""} {
        return
    }
    $db foreach top top {
        $db foreach inst $top inst {
            # Do something with $inst
        }
    }
}
gui database runOrRegisterChangedCallback "myFunc"
```

gui dnd

APIs for drag-and-drop.

gui dnd registerCallback

Register a custom drag-and-drop callback which is called just before the drop code is executed. Adds the name of the window where the drop is happening and the dropped data as the last two parameters of the callback.

Usage:

```
gui dnd registerCallback callback ?only?
```

Parameters:

`callback`

Name of the callback procedure to be called.

`only (optional)`

Only call the callback for the specified window. Supported values are: `-class <WindowClass>` or `-name <WindowName>`.

Example:

```
proc myFunc {w data} {
    # Do something
}

# General callback:
gui dnd registerCallback "myFunc"

# Callback for windows of class Schem (Note the " around "-class Schem"):
gui dnd registerCallback "myFunc" "-class Schem"
```

gui dnd removeCallback

Remove a previously registered callback. Only works when `-class <WindowClass>` or `-name <WindowName>` is exactly set like in the "registerCallback" call.

Usage:

`gui dnd removeCallback callback ?only?`

Parameters:

`callback`

The first string of the previously registered callback script.

`only (optional)`

Has to be set like the callback was previously registered. Supported values are: `-class <WindowClass>` or `-name <WindowName>`.

Example:

```
# General callback:
gui extract removeCallback "myFunc"

# Callback for windows of class Schem (Note the " around "-class Schem"):
gui dnd removeCallback "myFunc" "-class Schem"
```

gui export

Schematic export APIs.

gui export edif

Export a schematic of the given window as an EDIF file.

Usage:

```
gui export edif ?-libname libnameValue? ?-mono? ?-optimize optimizeValue? ?-view viewValue?
what window fname
```

Parameters:

`-libname` **`libnameValue`** `(optional default is NULL)`

Specifies the created library name.

`-mono` `(optional)`

If set then the resulting EDIF is in mono color.

`-optimize` **`optimizeValue`** `(optional default is NULL)`

The created EDIF file is in standard EDIF 2.0.0 format. You can optimize the created edif for OrCAD (orcad), ViewDraw (viewdraw), Cadence (cadence) or Tanner SEdit (sedit).

`-view` **`viewValue`** `(optional default is NULL)`

Specifies the view name to create.

`fname`

Name of the created file.

`what`

Specify what to export. This can be tree (exports the whole design), window (exports the content of the window specified by the `window` parameter).

`window`

Path or name of a window.

Example:

```
# export the contents of the window named "Schem"
gui export edif window "Schem" -optimize orcad "schem.edif"
```

gui export pdf

Create the schematic of the given window as a PDF file.

Usage:

`gui export pdf fname window size`

Parameters:

`fname`

Name of the created PDF file.

`size`

This is one of the predefined paper sizes (e.g. `Letter`, `A4`, etc.) as defined in the GUI Print dialog.

`window`

Path or name of a window.

Example:

```
# export the contents of the window named "Schem"
gui export pdf export.pdf "Schem" A
```

gui export photo

Create a photo of the given window.

Usage:

`gui export photo fname window size type`

Parameters:

`fname`

Name of the created image file.

`size`

This is one of the predefined screen resolution sizes (e.g. `Full-HD`, `4K-UHD`, etc.) as defined in the GUI `Save Schematic as Image` dialog, a string specifying the width and height (e.g. `640x480`), or `auto` for the exact pixel size of the window on screen.

`type`

Currently "gif", "png", and "svg" images are supported.

`window`

Path or name of a window.

Example:

```
# export the contents of the window named "Schem"
gui export photo export.png "Schem" 300x200 png
```

gui export skill

Export a schematic of the given window as a Skill script which can be loaded into the Cadence environment.

Usage:

```
gui export skill ?-analog analogValue? ?-disabletrigger? ?-epilog epilogValue? ?-instsuffix instsuffixValue? ?-metric? ?-multifile? ?-netlabel? ?-overwrite? ?-prolog prologValue? ?-technology technologyValue? what window libname fname
```

Parameters:

`-analog` **`analogValue`** `(optional default is "0")`

Generate the schematic in analog mode.

`-disabletrigger (optional)`

Disable triggers.

`-epilog` **`epilogValue`** `(optional default is NULL)`

Specify the path to a Skill script that is evaluated after the actual schematic generation.

`-instsuffix` **`instsuffixValue`** `(optional default is NULL)`

Append this suffix to all created instances.

`-metric (optional)`

Use metric user units.

`-multifile (optional)`

If true then multiple files are generated.

`-netlabel (optional)`

If true then net name labels are created.

`-overwrite (optional)`

If true then existing cells will be overwritten.

`-prolog` **`prologValue`** `(optional default is NULL)`

Specify the path to a Skill script that is evaluated before the actual schematic generation.

`-technology` **`technologyValue`** `(optional default is NULL)`

Specify the technology.

`fname`

Name of the created file.

`libname`

Specifies the created Cadence library name.

`what`

Specify what to export. This can be tree (exports the whole design), module (exports the current module), cone (exports the contents of the Cone window) or symbol (exports only all symbols).

`window`

Name of the window to export.

Example:

```
gui export skill tree "" "sv_lib" "design.il"
```

gui extract

Commands for the interaction with the Result table in the Cone Extraction window.

gui extract addDataColumn

Add a new data column. If you do not specify a name, a default name "DataX" is given where "X" is the column index. The command returns a list containing the column index and the column name, e.g. {1 Data1}.

Usage:

```
gui extract addDataColumn ?-name nameValue? ?-window windowValue?
```

Parameters:

`-name nameValue (optional default is "")`

Specify a column name.

`-window windowValue (optional default is NULL)`

Path or name of a Cone window, "DEFAULT" or "" for the default Cone window.

Example:

```
gui extract addDataColumn
gui extract addDataColumn -name "myDataColumn"
```

gui extract registerCallback

Add a callback which is called whenever the result is changed. Returns the name of the Cone window as last argument.

Usage:

```
gui extract registerCallback callback
```

Parameters:

`callback`

Name of the callback procedure to be called.

Example:

```
proc myFunc {coneWindow} {
    for {set row 0} {$row < [gui extract resultCount]} {incr row} {
        # Do something (related to the Cone Extraction
        # dialog window of the given $coneWindow)
    }
}

gui extract registerCallback "myFunc"
```

gui extract removeCallback

Remove a previously registered callback.

Usage:

```
gui extract removeCallback callback
```

Parameters:

`callback`
: The first string of the previously registered callback script.

Example:

```
gui extract removeCallback "myFunc"
```

gui extract result

Get the result for the given row.

Usage:

```
gui extract result ?-window windowValue? row
```

Parameters:

`-window windowValue (optional default is NULL)`
: Path or name of a Cone window, "DEFAULT" or "" for the default Cone window.

`row`
: Specify the row to be returned.

Example:

```
for {set row 0} {$row < [gui extract resultCount]} {incr row} {
    set result [gui extract result $row]
}
```

gui extract resultCount

Get the number of displayed results in the Cone Extraction dialog.

Usage:

```
gui extract resultCount ?-row rowValue? ?-window windowValue?
```

Parameters:

`-row rowValue (optional default is "")`

If row is provided, the number of children of this path entry is returned. If no row is provided, the number of paths is returned.

`-window windowValue (optional default is NULL)`

Path or name of a Cone window, "DEFAULT" or "" for the default Cone window.

Example:

```
gui extract resultCount
gui extract resultCount -row 1
```

gui extract setData

Set data in the custom column in the given row. Already existing data will be overwritten.

Usage:

```
gui extract setData ?-child childValue? ?-column columnValue? ?-window windowValue? row data
```

Parameters:

`-child childValue (optional default is "0")`

Write to the specified child of the row. Child index 0 is the row itself, child index 1 is the first child of the row.

`-column columnValue (optional default is "1")`

Specify the column you want to write to. Column 0 represents the Result and can not be written to.

`-window windowValue (optional default is NULL)`

Path or name of a Cone window, "DEFAULT" or "" for the default Cone window.

`data`

Data to be written.

`row`

Specify the row to be written to.

Example:

```
set row 0
set data "some data"
gui extract setData $row $data
gui extract setData -child 1 -column 1 2  "second column, first child of row 1"
```

gui extract showDataColumn

Toggle the visibility of the data column. Nothing happens in case the desired visibility is already set.

Usage:

`gui extract showDataColumn ?-column columnValue? ?-window windowValue? visibility`

Parameters:

`-column columnValue (optional default is "1")`

Column index for the column to be changed. Index 0 is the result column.

`-window windowValue (optional default is NULL)`

Path or name of a Cone window, "DEFAULT" or "" for the default Cone window.

`visibility`

0/false → hide column. 1/true → show column.

Example:

```
# hide the data column with index 2
gui extract showDataColumn -column 2 0

# show the data column with index 1
gui extract showDataColumn 1
```

gui highlight

The tool maintains global highlight information, common to all windows like Schem, Cone, Source and Mem, via the Flat View API. There are up to 16 different color numbers available (each is associated to a foreground and background color) - the colors can be changed in the Preferences dialog.

gui highlight changed

Inform the GUI that the highlight information in the database has changed.

Usage:

```
gui highlight changed
```

Parameters:

No parameters.

Example:

```
set oidList1 {
    {inst TOP U1}
}
set oidList2 {
    {inst TOP U2 m1}
    {inst TOP U2 m2}
}
set oidList3 {
    {port TOP IN}
    {net TOP IN}
}
gui window show "Cone"
gui cone append [concat $oidList1 $oidList2 $oidList3]

foreach oid $oidList1 {
    $db flathilight $oid set 0
}
foreach oid $oidList2 {
    $db flathilight $oid set 1
}
foreach oid $oidList3 {
    $db flathilight $oid set 2
}
gui highlight changed

foreach oid $oidList1 {
    $db flathilight $oid delete
}
gui highlight changed

$db hilight * deleteAll 1
gui highlight changed

$db hilight * deleteAll all
gui highlight changed
```

gui highlight greymode

Toggle "greymode" in the specified window.

Usage:

```
gui highlight greymode window
```

Parameters:

`window`

Path or name of a window of class Cone or Schem.

Example:

```
# Toggle greymode in the default Schem window.
set schem [::WindowManager::DefaultWindow Schem]
gui highlight greymode $schem
```

gui highlight registerChangedCallback

Register a customer specific callback that is called every time the highlight in the GUI changes. To remove a previously registered highlight changed callback use `gui highlight removeChangedCallback ...`.

Usage:

`gui highlight registerChangedCallback callback`

Parameters:

`callback`

Name of the callback procedure.

Example:

```
proc myHighlightChangedHandler {} {
    # Do something
}
gui highlight registerChangedCallback "myHighlightChangedHandler"
```

gui highlight removeChangedCallback

Remove a previously registered changed callback.

Usage:

`gui highlight removeChangedCallback callback`

Parameters:

`callback`

Name of the callback procedure.

Example:

```
gui highlight removeChangedCallback "myHighlightChangedHandler"
```

gui imageset

APIs to interact with image sets.

gui imageset check

Check if the given `$name` refers to a proper image set.

Usage:

```
gui imageset check name
```

Parameters:

`name`

The image set's name.

Example:

```
if {[gui imageset check "myImageSet"]} {
    gui console print "'myImageSet' is a proper image set."
}
```

gui imageset delete

Delete an image set, i.e. remove all of its Tk images.

Usage:

```
gui imageset delete name
```

Parameters:

`name`

The image set's name.

Example:

```
gui imageset delete "myImageSet"
```

gui imageset load

Load an image set, e.g. a collection of 5 images of sizes 16, 24, 32, 48, and 64, that can be used in toolbars, etc. The Tk images will be available as `UserImage:$name-$size`, where `$name` is the image set's name, and `$size` is one of `16`, `24`, `32`, `48`, `64`.

Usage:

`gui imageset load name pattern`

Parameters:

`name`

The image set's name.

`pattern`

The file name pattern. The literal string `SIZE` is replaced with the images sizes (16, 24, 32, 48, 64) when loading the corresponding image. When `$pattern` is e.g. `myPath/myImage-SIZE.png`, the images `myPath/myImage-16.png`, `myPath/myImage-24.png`, etc. are loaded.

Example:

```
# Load "myPath/myImage-16.png", "myPath/myImage-24.png", ....
gui imageset load "myImageSet" "myPath/myImage-SIZE.png"
```

gui infobox

APIs to interact with the Infobox window.

gui infobox hideWindow

Hide the Infobox window.

Usage:

```
gui infobox hideWindow
```

Parameters:

Example:

```
gui infobox hideWindow
```

gui infobox showWindow

Show the Infobox window.

Usage:

```
gui infobox showWindow ?-oids oidsValue?
```

Parameters:

`-oids oidsValue` (optional default is "SELECTION")
: Display these OIDs.

Example:

```
gui infobox showWindow
```

gui mem

APIs to interact with the Mem window.

gui mem append

Appends the objects (specified in $oidList) to the given Mem window.

Usage:

```
gui mem append ?-select? ?-window windowValue? oidList
```

Parameters:

`-select (optional)`

Select the newly added objects.

`-window windowValue (optional default is NULL)`

Path or name of a Mem window, "DEFAULT" or "" for the default Mem window.

`oidList`

List of objects to append.

Example:

```
set oidList {
    {inst TOP U2 m1}
    {inst TOP U2 m2}
    {port TOP IN}
    {net TOP OUT}
}
gui mem append -select $oidList
```

gui mem clear

Removes all objects from the given Mem window.

Usage:

`gui mem clear ?-window windowValue?`

Parameters:

`-window windowValue (optional default is NULL)`

Path or name of a Mem window, "DEFAULT" or "" for the default Mem window.

Example:

```
gui mem clear
```

gui mem contents

Returns all objects loaded to the given Mem window.

Usage:

```
gui mem contents ?-window windowValue?
```

Parameters:

`-window windowValue (optional default is NULL)`

Path or name of a Mem window, "DEFAULT" or "" for the default Mem window.

Example:

```
gui mem contents
```

gui mem selection

Returns all currently selected objects of the Mem window.

Usage:

```
gui mem selection ?-window windowValue?
```

Parameters:

`-window windowValue (optional default is NULL)`

Path or name of a Mem window, "DEFAULT" or "" for the default Mem window.

Example:

```
gui mem selection
```

gui menu

Extend the main menu.

gui menu checkbutton

Creates a new menu checkbutton. An existing checkbutton entry with the same name will be updated. Returns the menu ID of the checkbutton entry.

Usage:

```
gui menu checkbutton ?-position positionValue? path command variableName
```

Parameters:

`-position` **`positionValue`** `(optional default is "end")`

Specify the position of the new entry in the menu; must be an integer, 'end' or 'fixed-end'. 'fixed-end' entries are always below of all other menu entries.

`command`

The command to be executed if the menu is selected.

`path`

Path to the menu entry. The last entry is displayed as the menu label. If an underscore (_) is part of the last entry, the character immediately following the underscore is used as a menu item's hotkey. The main menu (the path's first entry) and the sub-menus corresponding to the other individual path entries are created in case they are not existing.

`variableName`

Name of a global variable storing the state of the checkbutton.

Example:

```
gui menu checkbutton {"Userware" "_Check 1"} "" checkState
```

gui menu clearSubmenu

Remove the content of the specified sub-menu.

Usage:

```
gui menu clearSubmenu path
```

Parameters:

`path`

Path to the sub-menu.

Example:

```
# Add a sub-menu
gui menu submenu {"Userware" "MySubMenu"}
gui menu command {"Userware" "MySubMenu" "A"} "gui console print A"
gui menu command {"Userware" "MySubMenu" "B"} "gui console print B"

# Remove the sub-menu's children
gui menu clearSubmenu {"Userware" "MySubMenu"}

# Remove the sub-menu
gui menu removeEntry {"Userware" "MySubMenu"}
```

gui menu command

Creates a new menu command. An existing command entry with the same name will be updated. Returns the menu ID of the command entry.

Usage:

`gui menu command ?-position positionValue? path command`

Parameters:

`-position positionValue` (optional default is "end")

Specify the position of the new entry in the menu; must be an integer, 'end' or 'fixed-end'. 'fixed-end' entries are always below of all other menu entries.

`command`

The command to be executed if the menu is selected.

`path`

Path to the menu entry. The last entry is displayed as the menu label. If an underscore (_) is part of the last entry, the character immediately following the underscore is used as a menu item's hotkey. The main menu (the path's first entry) and the sub-menus corresponding to the other individual path entries are created in case they are not existing.

Example:

```
gui menu command {"Userware" "MyFunc_1"} "gui console print MyFunc1"
gui menu command {"Userware" "MyFunc_2"} "gui console print MyFunc2"
gui menu command {"Userware" "MyFunc_3"} "gui console print MyFunc3"
```

gui menu customizeEntry

Register a callback procedure to customize an existing menu entry. Returns the menu ID of the menu entry.

Usage:

```
gui menu customizeEntry path callback
```

Parameters:

`callback`

Name of the customization callback procedure.

`path`

Path to the menu entry.

Example:

```
proc _customizeOption {menuId} {
    $menuId entryconfigure end -state disabled
}
gui menu customizeEntry {"Userware" "MySubMenu"} _customizeOption
```

gui menu exists

Check if the menu with the given path exists.

Usage:

```
gui menu exists path
```

Parameters:

`path`

Path to the menu entry.

Example:

```
if {![gui menu exists {"Userware"}]} {
    ##
    # Create the menu.
    #
}
```

gui menu getCommand

Returns the command and arguments which is called for the menu entry.

Usage:

```
gui menu getCommand path
```

Parameters:

`path`

Path to the menu entry.

Example:

```
gui menu    command {"Userware" "MySubMenu" "A"} "gui console print A"
gui menu getCommand {"Userware" "MySubMenu" "A"}
```

gui menu getPosition

Returns a list with two entries. The first entry is the position of the menu in its sub-menu. The second entry is true if and only if the position is fixed to the end.

Usage:

```
gui menu getPosition path
```

Parameters:

`path`

Path to the menu entry.

Example:

```
gui menu getPosition {"Userware" "MySubMenu" "A"}
```

gui menu mainMenu

Creates a new menu button including the pull down menu in the main menu. Returns the menu ID of the main menu entry. If a menu button with this name already exists this command will initialize it again at the given position.

Usage:

```
gui menu mainMenu ?-position positionValue? name
```

Parameters:

`-position positionValue` (optional default is "end")

Specify the position of the menu label in the main menu; must be an integer, 'end' or 'fixed-end'. 'fixed-end' items are always right of all other menu items.

`name`

Text displayed as the menu's label. If an underscore (_) is part of the name, the character immediately following the underscore is used as the menu's hotkey. If no underscore character is part of the name the first suitable character will be used.

Example:

```
gui menu mainMenu "_Userware"
```

gui menu moveEntry

Move a menu entry to a different position. The entry is removed from the menu and then inserted at the position. The new position is determined after the removal.

Usage:

```
gui menu moveEntry ?-position positionValue? path
```

Parameters:

`-position positionValue` (optional default is "end")

Specify the new position of the entry in the menu; must be an integer, 'end' or 'fixed-end'. 'fixed-

end' entries are always below of all other menu entries.

`path`

Path to the menu entry to be moved.

Example:

```
gui menu moveEntry {"Userware" "MyFunc2"} -position 0
gui menu moveEntry {"Userware" "MyFunc2"} -position fixed-end
```

gui menu radiobutton

Creates a new menu radiobutton. An existing radiobutton entry with the same name will be updated. Returns the menu ID of the radiobutton entry.

Usage:

`gui menu radiobutton ?-position positionValue? path command variableName variableValue`

Parameters:

`-position positionValue (optional default is "end")`

Specify the position of the new entry in the menu; must be an integer, 'end' or 'fixed-end'. 'fixed-end' entries are always below of all other menu entries.

`command`

The command to be executed if the menu is selected.

`path`

Path to the menu entry. The last entry is displayed as the menu label. If an underscore (_) is part of the last entry, the character immediately following the underscore is used as a menu item's hotkey. The main menu (the path's first entry) and the sub-menus corresponding to the other individual path entries are created in case they are not existing.

`variableName`

Name of a global variable storing the state of the radiobutton.

`variableValue`

Value to store in the variable associated with the radiobutton.

Example:

```
gui menu radiobutton {"Userware" "Option _1"} "" radioValue option1
gui menu radiobutton {"Userware" "Option _2"} "" radioValue option2
```

gui menu removeEntry

Remove a menu entry.

Usage:

```
gui menu removeEntry path
```

Parameters:

`path`
: Path to the menu entry to be removed.

Example:

```
gui menu removeEntry {"Userware" "MyFunc1"}
gui menu removeEntry {"Userware" "MyFunc2"}
gui menu removeEntry {"Userware"}
```

gui menu removeSeparator

Remove a menu separator identified by the menu and separator ID.

Usage:

```
gui menu removeSeparator path sepId
```

Parameters:

`path`
: Path to the menu containing the separator.

`sepId`
: Unique ID to identify the separator.

Example:

```
gui menu command   {"Userware" "Function 1"} "gui console print Function1"
gui menu separator {"Userware"} -sepId "mySep1"
gui menu command   {"Userware" "Function 2"} "gui console print Function2"
gui menu removeSeparator {"Userware"} "mySep1"
```

gui menu separator

Creates a new menu separator. An existing separator with the same ID will be updated. The separator ID is returned.

Usage:

```
gui menu separator ?-position positionValue? ?-sepId sepIdValue? path
```

Parameters:

`-position` **`positionValue`** `(optional default is "end")`

Specify the position of the new entry in the menu; must be an integer, 'end' or 'fixed-end'. 'fixed-end' entries are always below of all other menu entries.

`-sepId` **`sepIdValue`** `(optional default is NULL)`

Unique ID to be able to identify the separator.

`path`

Path to the menu entry. The separator is added after the last entry of the given menu. The main menu (the path's first entry) and the sub-menus corresponding to the other individual path entries are created in case they are not existing.

Example:

```
gui menu separator {"Userware"}
```

gui menu separatorIdAfter

Return the ID of the menu separator directly following the menu item indicated by $path; if no such separator exists, return an empty string.

Usage:

```
gui menu separatorIdAfter path
```

Parameters:

`path`

Path to the menu entry immediately before the menu separator in question.

Example:

```
gui menu command {"Userware" "Before"} "gui console print Before"
gui menu separator {"Userware"}
gui menu separatorIdAfter {"Userware" "Before"}
```

gui menu submenu

Creates a new sub-menu. An existing sub-menu entry with the same name will be updated. Returns the menu ID of the sub-menu entry.

Usage:

`gui menu submenu ?-position positionValue? path`

Parameters:

`-position positionValue (optional default is "end")`

Specify the position of the new entry in the menu; must be an integer, 'end' or 'fixed-end'. 'fixed-end' entries are always below of all other menu entries.

`path`

Path to the menu entry. The last entry is displayed as the menu label. If an underscore (_) is part of the last entry, the character immediately following the underscore is used as a menu item's hotkey. The main menu (the path's first entry) and the sub-menus corresponding to the other individual path entries are created in case they are not existing.

Example:

```
gui menu submenu {"Userware" "Sub_Menu"}
```

gui messages

APIs to interact with the Messages window.

gui messages hideWindow

Hide the Messages window.

Usage:

```
gui messages hideWindow
```

Parameters:

Example:

```
gui messages hideWindow
```

gui messages showWindow

Show the Messages window.

Usage:

```
gui messages showWindow
```

Parameters:

Example:

```
gui messages showWindow
```

gui nlview

APIs to interact with an Nlview widget.

gui nlview getBoundingBox

Returns the combined bounding box (left, top, right, bottom coordinates) of the given $oids in the Nlview widget referenced by $nlview. If the bounding box cannot be determined, an empty list is returned.

Usage:

```
gui nlview getBoundingBox nlview oids
```

Parameters:

`nlview`

Nlview widget path.

`oids`

List of objects.

Example:

```
set oid1 {inst TOP INST1}
set oid2 {inst TOP INST1}
set nlview [gui schem nlv]
set bbox [gui nlview getBoundingBox $nlview [list $oid1 $oid2]]
lassign $bbox left top right bottom
gui console print "Combined bounding box of $oid1 and $oid2:"
gui console print "left=$left"
gui console print "top=$top"
gui console print "right=$right"
gui console print "bottom=$bottom"
```

gui nlview getOrientation

Returns the orientation of the instance referenced by $inst_oid in the Nlview widget referenced by $nlview.

Usage:

```
gui nlview getOrientation nlview inst_oid
```

Parameters:

`inst_oid`

OID of the instance.

`nlview`

Nlview widget path.

Example:

```
set oid [$db search inst [$db get_top_design] "instanceName"]
set nlview [gui schem nlv]
set orientation [gui nlview getOrientation $nlview $oid]
gui console print "Orientation of $oid: $orientation"
```

gui nlview getSymbol

Get the DEF-style symbol string for the given instance.

Usage:

```
gui nlview getSymbol instance_oid
```

Parameters:

`instance_oid`
: OID of the instance.

Example:

```
set instance [$db search inst [$db get_top_design] "instanceName"]
set symbol [gui nlview getSymbol $instance]
gui console print "The DEF-style symbol of '$instance' is '$symbol'."
```

gui nlview isDisplayed

Returns "true" if the object referenced by $oid is currently visible in the Nlview widget referenced by $nlview.

Usage:

```
gui nlview isDisplayed nlview oid
```

Parameters:

`nlview`
: Nlview widget path.

`oid`

OID of the object to check.

Example:

```
set oid [$db search inst [$db get_top_design] "instanceName"]
set nlview [gui schem nlv]
if {[gui nlview isDisplayed $nlview $oid]} {
    gui console print "$oid is displayed in the default Schem window."
}
```

gui nlview logFile

Open a log file for the given Nlview widget. If the 'fname' parameter is missing, close the current log file.

Usage:

`gui nlview logFile nlview ?fname?`

Parameters:

`fname (optional)`

File name of the log file. If this parameter is missing, the current log file is closed.

`nlview`

Nlview widget path.

Example:

```
set fname  "nlview-[pid].log"
set nlview [gui schem nlv]

# open a log file
gui nlview logFile $nlview $fname

# do something with the Schem window

# close the current log file
gui nlview logFile $nlview
```

gui nlview move

Move the instances in `$instance_oids` horizontally relative to the reference instance `$reference_inst_oid` in the Nlview widget `$nlview`. If `$direction` is `left`, move the instances to the left of the reference, if it is `right` move them to the right, if it is `same` move the instances to the same column as the reference. All instances must be in the same hierarchy level.

Usage:

```
gui nlview move nlview reference_inst_oid direction instance_oids
```

Parameters:

`direction`

Move direction: `left`, `right`, or `same`.

`instance_oids`

List of instances OIDs to move.

`nlview`

Nlview widget path.

`reference_inst_oid`

OID of the reference instance.

Example:

```
set reference {inst TOP I1}
set instances {{inst TOP I2} {inst TOP I3}}
set nlview [gui schem nlv]

##
# Move $instances to the left of $reference.
#
gui nlview move $nlview $reference "left" $instances
```

gui parasitic

APIs to interact with the Parasitic window.

gui parasitic append

Append the parasitic model for the given signal, net or parasitic OID(s).

Usage:

```
gui parasitic append ?-window windowValue? oidList
```

Parameters:

`-window windowValue` (optional default is NULL)

Path or name of a Parasitic window, "DEFAULT" or "" for the default Parasitic window.

`oidList`

List of objects to append to the Parasitic window.

Example:

```
set signalList {{module IN IN} {module OUT OUT}}
gui parasitic append $signalList
```

gui parasitic clear

Delete all objects from the Parasitic window.

Usage:

```
gui parasitic clear ?-window windowValue?
```

Parameters:

`-window windowValue` (optional default is NULL)

Path or name of a Parasitic window, "DEFAULT" or "" for the default Parasitic window.

Example:

```
gui parasitic clear
```

gui parasitic hideWindow

Hide the default Parasitic window.

Usage:

`gui parasitic hideWindow`

Parameters:

Example:

```
gui parasitic hideWindow
```

gui parasitic load

Load the parasitic model for the given signal, net or parasitic OID(s).

Usage:

`gui parasitic load ?-window windowValue? oidList`

Parameters:

`-window windowValue (optional default is NULL)`

Path or name of a Parasitic window, "DEFAULT" or "" for the default Parasitic window.

`oidList`

List of objects to load into the Parasitic window.

Example:

```
set signalList {{module IN IN} {module OUT OUT}}
gui parasitic load $signalList
```

gui parasitic nlv

Get the path of the Nlview widget.

Usage:

`gui parasitic nlv ?-window windowValue?`

Parameters:

`-window windowValue (optional default is NULL)`

Path or name of a Parasitic window, "DEFAULT" or "" for the default Parasitic window.

Example:

```
gui parasitic nlv
```

gui parasitic regenerate

Regenerate the displayed schematic of the given Parasitic window.

Usage:

```
gui parasitic regenerate ?-window windowValue?
```

Parameters:

`-window windowValue (optional default is NULL)`

Path or name of a Parasitic window, "DEFAULT" or "" for the default Parasitic window.

Example:

```
gui parasitic regenerate
```

gui parasitic registerZoomChangedCallback

Register a callback that is evaluated every time current zoom factor has changed. Before evaluating the callback, the Parasitic window identifier and the current zoom factor value are appended. The current zoom factor may also be empty.

Usage:

```
gui parasitic registerZoomChangedCallback callback
```

Parameters:

`callback`

The callback script to be evaluated.

Example:

```
proc myZoomChanged {w zoomFactor} {
    gui console print  "Parasitic window $w: the new zoom factor is $zoomFactor"
}

# add callback
gui parasitic registerZoomChangedCallback myZoomChanged

# remove callback
gui parasitic removeZoomChangedCallback myZoomChanged
```

gui parasitic removeZoomChangedCallback

Remove the callback previously registered with `gui parasitic registerZoomChangedCallback ⋯`.

Usage:

`gui parasitic removeZoomChangedCallback callback`

Parameters:

`callback`

The callback script to remove.

Example:

```
proc myZoomChanged {w zoomFactor} {
    gui console print  "Parasitic window $w: the new zoom factor is $zoomFactor"
}

# add callback
gui parasitic registerZoomChangedCallback myZoomChanged

# remove callback
gui parasitic removeZoomChangedCallback myZoomChanged
```

gui parasitic saveAs

Save the contents of the displayed RC network in the Parasitic window as a Spice, DSPF or SPEF file.

Usage:

```
gui parasitic saveAs ?-window windowValue? what filename
```

Parameters:

`-window windowValue (optional default is NULL)`

Path or name of a Parasitic window, "DEFAULT" or "" for the default Parasitic window.

`filename`

Specifies the created output filename.

`what`

This can be "Spice", "DSPF" or "SPEF".

Example:

```
gui parasitic saveAs "Spice" "parasitic.sp"
```

gui parasitic selection

Returns all currently selected objects of the Parasitic window.

Usage:

```
gui parasitic selection ?-window windowValue?
```

Parameters:

`-window windowValue (optional default is NULL)`

Path or name of a Parasitic window, "DEFAULT" or "" for the default Parasitic window.

Example:

```
gui parasitic selection
```

gui parasitic show

Show the RC network of the given signal or net OID(s).

Usage:

```
gui parasitic show ?-append? ?-window windowValue? signalList
```

Parameters:

`-append (optional)`

Append the given signals.

`-window windowValue (optional default is NULL)`

Path or name of a Parasitic window, "DEFAULT" or "" for the default Parasitic window.

`signalList`

List of signals to load into the Parasitic window.

Example:

```
set signalList {{module IN IN} {module OUT OUT}}
gui parasitic show $signalList
```

gui parasitic showWindow

Show the default Parasitic window.

Usage:

```
gui parasitic showWindow
```

Parameters:

Example:

```
gui parasitic showWindow
```

gui parasitic zoom

Changes the zoom factor of the Parasitic window.

Usage:

```
gui parasitic zoom ?-window windowValue? factor
```

Parameters:

`-window windowValue (optional default is NULL)`

Path or name of a Parasitic window, "DEFAULT" or "" for the default Parasitic window.

`factor`

The argument $factor must be "fullfit" or a number ("fullfit" only works if that window is currently visible).

Example:

```
gui window show "Parasitic"
gui parasitic zoom "fullfit"
```

gui parser

APIs to interact with the Parser.

gui parser getArgs

Get a list of command line arguments that can be used to start the given parser directly.

Usage:

`gui parser getArgs type`

Parameters:

`type`

The parser type.

Example:

```
set parser_args [gui parser getArgs Spice]
```

gui parser setArgs

Set the command line arguments used to invoke the given parser.

Usage:

```
gui parser setArgs type parserArgs
```

Parameters:

`parserArgs`

The parser Args.

`type`

The parser type.

Example:

```
set parser_args [gui parser setArgs Spice {-breakOnErr on}]
```

gui parser showReadDialog

Show the dialog for the given file type.

Usage:

```
gui parser showReadDialog type
```

Parameters:

`type`

The type of the read dialog to display. Depending on the available license this can be "verilog", "edif", "rtl", "spice", "binfile" or "wave".

Example:

```
# open the binfile read dialog
gui parser showReadDialog binfile
```

gui plugin

APIs to create Userware plugins.

gui plugin addConfig

Add the configuration option $name of the plugin identified by $namespace with the specified

$type, $defaultValue and $doc string.

Usage:

```
gui plugin addConfig namespace name defaultValue type doc
```

Parameters:

`defaultValue`
: The option's default value.

`doc`
: The documentation string to show in the GUI.

`name`
: The name of the configuration option.

`namespace`
: The plugin identifier.

`type`
: The option's type (text, bool, number).

Example:

```
gui plugin addConfig MyPlugin greeting "Hello world!" text "Welcome message"
```

gui plugin check

Returns true if the current script is loaded from the Plugins dialog, in contrast to being loaded as a plain userware script.

Usage:

```
gui plugin check
```

Parameters:

No parameters.

Example:

```
if {[gui plugin check]} {
    # The script is loaded from the Plugins dialog.
} else {
    # The script is loaded as an Userware script.
}
```

gui plugin getConfigValue

Retrieve the current value of the configuration option $name of the plugin identified by $namespace.

Usage:

`gui plugin getConfigValue namespace name`

Parameters:

`name`

The name of the configuration option.

`namespace`

The plugin identifier.

Example:

```
set value [gui plugin getConfigValue MyPlugin greeting]
```

gui popup

Extend the popup menu.

gui popup append

Extend the popup menu. The new entry is added at the bottom of the popup menu. The popup menu entry is only enabled if one or more objects are selected, otherwise the entry is disabled. A custom callback procedure registered by `gui popup customize` can be used to always enable the popup menu entry. This is a legacy command. Use 'gui popup command ...' instead.

Usage:

`gui popup append ?-menuname menunameValue? label command ?only?`

Parameters:

`-menuname menunameValue (optional default is "Userware")`

Name of the created menu entry.

`command`

If the menu entry is invoked then this $command is executed. The list of selected OIDs is appended to $command.

`label`

Label to display in the sub-menu.

`only (optional)`

Only show the Popup menu in the specified window. Supported values are: `-class <WindowClass>` or `-name <WindowName>`.

Example:

```
proc Userware:CustomCmd {oidList} {
    # Do something with $oidList
}
gui popup append "Custom Cmd" Userware:CustomCmd
```

gui popup checkbutton

Creates a new menu checkbutton. By default the command is affecting all popup menus for all clients. An existing checkbutton entry with the same name will be updated.

Usage:

`gui popup checkbutton ?-position positionValue? path command variableName ?only?`

Parameters:

`-position positionValue (optional default is "end")`

Specify the position of the new entry in the menu; must be an integer, 'end' or 'fixed-end'. 'fixed-end' entries are always below of all other menu entries.

`command`

The command to be executed if the menu is selected.

`only (optional)`

Only execute the command for the specified window(s). Supported values are: `-class <WindowClass>`, `-name <WindowName>` or `-client <WindowClient>`.

`path`

Path to the popup menu entry. The last entry is displayed as the menu label. If an underscore (_) is part of the last entry, the character immediately following the underscore is used as a menu item's hotkey. The path's first entry and the sub-menus corresponding to the other individual path entries are created in case they are not existing.

`variableName`

Name of a global variable storing the state of the checkbutton.

Example:

```
gui popup checkbutton {"Userware" "_Check 1"} "" checkState
```

gui popup clearSubmenu

Remove the content of the specified sub-menu. By default the command is affecting all popup menus for all clients.

Usage:

```
gui popup clearSubmenu path ?only?
```

Parameters:

`only (optional)`

Only execute the command for the specified window(s). Supported values are: `-class <WindowClass>`, `-name <WindowName>` or `-client <WindowClient>`.

`path`

Path to the sub-menu.

Example:

```
proc MyFunc {args} {
    # Do something
}

# Add a sub-menu
gui popup submenu {"Userware" "MySubMenu"}
gui popup command {"Userware" "MySubMenu" "A"} "MyFunc A"
gui popup command {"Userware" "MySubMenu" "B"} "MyFunc B"

# Remove the sub-menu's children
gui popup clearSubmenu {"Userware" "MySubMenu"}

# Remove the sub-menu
gui popup removeEntry {"Userware" "MySubMenu"}
```

gui popup command

Creates a new menu command. By default the command is affecting all popup menus for all clients. An existing command entry with the same name will be updated.

Usage:

`gui popup command ?-position positionValue? path command ?only?`

Parameters:

`-position positionValue (optional default is "end")`

Specify the position of the new entry in the menu; must be an integer, 'end' or 'fixed-end'. 'fixed-end' entries are always below of all other menu entries.

`command`

The command to be executed if the menu is selected.

`only (optional)`

Only execute the command for the specified window(s). Supported values are: `-class <WindowClass>`, `-name <WindowName>` or `-client <WindowClient>`.

`path`

Path to the popup menu entry. The last entry is displayed as the menu label. If an underscore (_) is part of the last entry, the character immediately following the underscore is used as a menu item's hotkey. The path's first entry and the sub-menus corresponding to the other individual path entries are created in case they are not existing.

Example:

```
proc MyFunc {number args} {
    gui console print "MyFunc $number: $args"
}

gui popup command {"Userware" "MyFunc1"} "MyFunc 1"
gui popup command {"Userware" "MyFunc2"} "MyFunc 2"
gui popup command {"Userware" "MyFunc3"} "MyFunc 3"
```

gui popup customize

Every time the popup menu is created (click the right mouse button) all registered callbacks are evaluated.

Usage:

```
gui popup customize callback
```

Parameters:

`callback`

Name of the callback procedure. The name of the popup menu as well as the list of selected OIDs is appended to the callback.

Example:

```
proc Userware:CustomPopupCmd {menu oidList} {
    foreach oid $oidList {
        $menu add command -label $oid
    }
}
gui popup customize Userware:CustomPopupCmd
```

gui popup customizeEntry

Register a callback procedure to customize an existing menu entry. Returns the menu ID of the menu entry.

Usage:

```
gui popup customizeEntry path callback ?only?
```

Parameters:

`callback`

Name of the customization callback procedure.

`only (optional)`

Only execute the command for the specified window(s). Supported values are: `-class <WindowClass>`, `-name <WindowName>` or `-client <WindowClient>`.

`path`

Path to the popup menu entry.

Example:

```
proc _customizeOption {menuId} {
    $menuId entryconfigure end -state disabled
}
gui popup customizeEntry {"Userware" "MySubMenu"} _customizeOption
```

gui popup exists

Check if the menu with the given path exists.

Usage:

`gui popup exists path ?only?`

Parameters:

`only (optional)`

Only call the callback for the specified window. Supported values are: `-class <WindowClass>`, `-name <WindowName>` or `-client <WindowClient>`.

`path`

Path to the popup menu entry.

Example:

```
if {![gui popup exists {"Userware"} "-name Schem"]} {
    ##
    # Create the menu.
    #
}
```

gui popup getCommand

Returns a list containing one sub-list for each command matching the arguments. Each sub-list has two entries with the first entry being the client and the second entry being the command and arguments.

Usage:

```
gui popup getCommand path ?only?
```

Parameters:

`only` `(optional)`

Only execute the command for the specified window(s). Supported values are: `-class <WindowClass>`, `-name <WindowName>` or `-client <WindowClient>`.

`path`

Path to the popup menu entry.

Example:

```
proc MyFunc {args} {
    # Do something
}

gui popup command {"Userware" "MySubMenu" "A"} "MyFunc"
gui popup getCommand {"Userware" "MySubMenu" "A"}
```

gui popup getPosition

Returns a list containing one sub-list for each entry matching the arguments. Each sub-list has two entries with the first entry being the client and the second entry being a list with the result. The first entry of the result is the position of the specified menu in its sub-menu. The second entry of the result is true if and only if the position is fixed to the end.

Usage:

```
gui popup getPosition path ?only?
```

Parameters:

`only (optional)`

Only execute the command for the specified window(s). Supported values are: `-class <WindowClass>`, `-name <WindowName>` or `-client <WindowClient>`.

`path`

Path to the popup menu entry.

Example:

```
gui popup getPosition {"Userware" "MySubMenu" "A"}
```

gui popup moveEntry

Move a menu entry to a different position. The entry is removed from the menu and then inserted at the position. The new position is determined after the removal. By default the command is affecting all popup menus for all clients.

Usage:

```
gui popup moveEntry ?-position positionValue? path ?only?
```

Parameters:

`-position positionValue (optional default is "end")`

Specify the new position of the entry in the menu; must be an integer, 'end' or 'fixed-end'. 'fixed-end' entries are always below of all other menu entries.

`only (optional)`

Only execute the command for the specified window(s). Supported values are: `-class <WindowClass>`, `-name <WindowName>` or `-client <WindowClient>`.

`path`

Path to the popup menu entry to be moved.

Example:

```
gui popup moveEntry {"Userware" "MyFunc2"} -position 0
gui popup moveEntry {"Userware" "MyFunc2"} -position fixed-end
```

gui popup radiobutton

Creates a new menu radiobutton. By default the command is affecting all popup menus for all clients. An existing radiobutton entry with the same name will be updated.

Usage:

```
gui popup radiobutton ?-position positionValue? path command variableName variableValue ?only?
```

Parameters:

`-position positionValue (optional default is "end")`

Specify the position of the new entry in the menu; must be an integer, 'end' or 'fixed-end'. 'fixed-end' entries are always below of all other menu entries.

`command`

The command to be executed if the menu is selected.

`only (optional)`

Only execute the command for the specified window(s). Supported values are: `-class <WindowClass>`, `-name <WindowName>` or `-client <WindowClient>`.

`path`

Path to the popup menu entry. The last entry is displayed as the menu label. If an underscore (_) is part of the last entry, the character immediately following the underscore is used as a menu item's hotkey. The path's first entry and the sub-menus corresponding to the other individual path entries are created in case they are not existing.

`variableName`

Name of a global variable storing the state of the radiobutton.

`variableValue`

Value to store in the variable associated with the radiobutton.

Example:

```
proc MyFunc {args} {
    # Do something
}

gui popup radiobutton {"Userware" "Option 1"} "" radioValue option1
gui popup radiobutton {"Userware" "Option 2"} "" radioValue option2
```

gui popup remove

Remove an entry added by the GUI API with 'gui popup append' from the Popup menu. This is a legacy command. Use 'gui popup command ...' instead.

Usage:

```
gui popup remove ?-menuname menunameValue? label
```

Parameters:

`-menuname` **`menunameValue`** `(optional default is "Userware")`

Name of the sub-menu entry.

`label`

Label of the entry to be removed.

Example:

```
gui popup remove "Custom Cmd"
```

gui popup removeCustomize

Remove a registered Popup customization callback.

Usage:

```
gui popup removeCustomize callback
```

Parameters:

`callback`

Name of the callback procedure.

Example:

```
gui popup removeCustomize Userware:CustomPopupCmd
```

gui popup removeEntry

Remove a popup menu entry. By default the command is affecting all popup menus for all clients.

Usage:

```
gui popup removeEntry path ?only?
```

Parameters:

`only (optional)`
: Only execute the command for the specified window(s). Supported values are: `-class <WindowClass>`, `-name <WindowName>` or `-client <WindowClient>`.

`path`
: Path to the popup menu entry to be removed.

Example:

```
gui popup removeEntry {"Userware" "MyFunc1"}
gui popup removeEntry {"Userware" "MyFunc2"}
gui popup removeEntry {"Userware"}
```

gui popup reset

Removes all custom popup menu entries.

Usage:

```
gui popup reset
```

Parameters:

No parameters.

Example:

```
gui popup reset
```

gui popup separator

Creates a new menu separator. An existing separator with the same ID will be updated. Returns a list containing one sub-list for each created separator. Each sub-list has two entries with the first entry being the client and the second entry being the sepId.

Usage:

```
gui popup separator ?-position positionValue? ?-sepId sepIdValue? path ?only?
```

Parameters:

`-position` **`positionValue`** `(optional default is "end")`

Specify the position of the new entry in the menu; must be an integer, 'end' or 'fixed-end'. 'fixed-end' entries are always below of all other menu entries.

`-sepId` **`sepIdValue`** `(optional default is NULL)`

Unique ID to be able to identify the separator.

`only` `(optional)`

Only execute the command for the specified window(s). Supported values are: `-class <WindowClass>`, `-name <WindowName>` or `-client <WindowClient>`.

`path`

Path to the popup menu entry. The separator is added after the last entry of the given menu or the given position. The path's first entry and the sub-menus corresponding to the other individual path entries are created in case they are not existing.

Example:

```
gui popup separator {"Userware"}
```

gui popup separatorIdAfter

Returns a list containing one sub-list for each matching separator. Each sub-list has two entries with the first entry being the client and the second entry being the sepId.

Usage:

```
gui popup separatorIdAfter path ?only?
```

Parameters:

`only (optional)`

Only execute the command for the specified window(s). Supported values are: `-class <WindowClass>`, `-name <WindowName>` or `-client <WindowClient>`.

`path`

Path to the popup menu entry immediately before the menu separator in question.

Example:

```
proc MyFunc {args} {
    # Do something
}

gui popup command {"Userware" "Before"} "MyFunc"
gui popup separator {"Userware"}
gui popup separatorIdAfter {"Userware" "Before"}
```

gui popup submenu

Creates a new sub-menu. By default the command is affecting all popup menus for all clients. An existing sub-menu entry with the same name will be updated.

Usage:

```
gui popup submenu ?-position positionValue? path ?only?
```

Parameters:

`-position positionValue (optional default is "end")`

Specify the position of the new entry in the menu; must be an integer, 'end' or 'fixed-end'. 'fixed-end' entries are always below of all other menu entries.

`only (optional)`

Only execute the command for the specified window(s). Supported values are: `-class <WindowClass>`, `-name <WindowName>` or `-client <WindowClient>`.

`path`

Path to the popup menu entry. The last entry is displayed as the menu label. If an underscore (_) is part of the last entry, the character immediately following the underscore is used as a menu item's hotkey. The path's first entry and the sub-menus corresponding to the other individual path entries are created in case they are not existing.

Example:

```
gui popup submenu {"Userware" "SubMenu"}
```

gui schem

APIs to interact with the Schem window.

gui schem clearCache

Clear the Schem window's cache. If neither '-window ...' nor '-all' are specified, operate on the default Schem window.

Usage:

```
gui schem clearCache ?-all? ?-clearCurrentModule? ?-removeAttribute? ?-window windowValue?
```

Parameters:

`-all (optional)`

Operate on all Schem windows.

`-clearCurrentModule (optional)`

Clear the current module.

`-removeAttribute (optional)`

Also remove the database schematic cache.

`-window windowValue (optional default is NULL)`

Path or name of a Schem window, "DEFAULT" or "" for the default Schem window.

Example:

```
##
# Clear the caches of all Schem windows.
#
gui schem clearCache -all

##
# Clear the cache of the window named "Schem".
#
gui schem clearCache -window "Schem"
```

gui schem contents

Returns all objects loaded to the given Schem window.

Usage:

```
gui schem contents ?-type typeValue? ?-window windowValue?
```

Parameters:

`-type typeValue (optional default is "*")`

If this parameter is given, the the result is filtered by the given type.

`-window windowValue (optional default is NULL)`

Path or name of a Schem window, "DEFAULT" or "" for the default Schem window.

Example:

```
gui schem contents
```

gui schem fold

Fold the given modules in the Schem window.

Usage:

```
gui schem fold ?-window windowValue? oidList
```

Parameters:

`-window windowValue` (optional default is NULL)

Path or name of a Schem window, "DEFAULT" or "" for the default Schem window.

`oidList`

List of objects to fold.

Example:

```
set oidList [gui schem contents]
gui schem fold $oidList
```

gui schem getCurrentModule

Get the OID of the module displayed in the Schem window.

Usage:

`gui schem getCurrentModule ?-window windowValue?`

Parameters:

`-window windowValue` (optional default is NULL)

Path or name of a Schem window, "DEFAULT" or "" for the default Schem window.

Example:

```
set mod [gui schem getCurrentModule]
```

gui schem isFolded

Check if the fold flag is set at a module.

Usage:

`gui schem isFolded ?-window windowValue? oid`

Parameters:

`-window windowValue` (optional default is NULL)

Path or name of a Schem window, "DEFAULT" or "" for the default Schem window.

`oid`

Module OID to check.

Example:

```
set oidList [gui schem contents]
gui schem isFolded [lindex $oidList 0]
```

gui schem nlv

Get the path of the Nlview widget.

Usage:

`gui schem nlv ?-window windowValue?`

Parameters:

`-window windowValue` (optional default is NULL)

Path or name of a Schem window, "DEFAULT" or "" for the default Schem window.

Example:

```
gui schem nlv
```

gui schem pages

Get the number of schematic pages displayed in the Schem window.

Usage:

`gui schem pages ?-window windowValue?`

Parameters:

`-window windowValue (optional default is NULL)`

Path or name of a Schem window, "DEFAULT" or "" for the default Schem window.

Example:

```
gui schem pages
```

gui schem regenerate

Regenerate the displayed schematic of the given Schem window.

Usage:

```
gui schem regenerate ?-window windowValue?
```

Parameters:

`-window windowValue (optional default is NULL)`

Path or name of a Schem window, "DEFAULT" or "" for the default Schem window.

Example:

```
gui schem regenerate
```

gui schem registerChangedCallback

Register a callback that is called after the contents of a Schem window have change. The Schem window's path is appended to the callback.

Usage:

```
gui schem registerChangedCallback callback
```

Parameters:

`callback`

The callback script.

Example:

```
proc my_callback {w} {
    gui console print "The Schem window has changed: $w"
}

gui schem registerChangedCallback my_callback
```

gui schem registerZoomChangedCallback

Register a callback that is evaluated every time current zoom factor has changed. Before evaluating the callback, the Schem window identifier and the current zoom factor value are appended. The current zoom factor may also be empty.

Usage:

```
gui schem registerZoomChangedCallback callback
```

Parameters:

`callback`

The callback script to be evaluated.

Example:

```
proc myZoomChanged {w zoomFactor} {
    gui console print "Schem window $w: the new zoom factor is $zoomFactor"
}

# add callback
gui schem registerZoomChangedCallback myZoomChanged

# remove callback
gui schem removeZoomChangedCallback myZoomChanged
```

gui schem removeChangedCallback

Remove the previously registered callback script.

Usage:

```
gui schem removeChangedCallback callback
```

Parameters:

`callback`
: The callback script.

Example:

```
gui schem removeChangedCallback my_callback
```

gui schem removeZoomChangedCallback

Remove the callback previously registered with `gui schem registerZoomChangedCallback …`.

Usage:

`gui schem removeZoomChangedCallback callback`

Parameters:

`callback`
: The callback script to remove.

Example:

```
proc myZoomChanged {w zoomFactor} {
    gui console print "Schem window $w: the new zoom factor is $zoomFactor"
}

# add callback
gui schem registerZoomChangedCallback myZoomChanged

# remove callback
gui schem removeZoomChangedCallback myZoomChanged
```

gui schem selection

Returns all currently selected objects of the Schem window.

Usage:

`gui schem selection ?-window windowValue?`

Parameters:

`-window windowValue (optional default is NULL)`

Path or name of a Schem window, "DEFAULT" or "" for the default Schem window.

Example:

```
gui schem selection
```

gui schem setCurrentModule

Set this module as the current module in the Schem window. The schematic for the module is generated. The return value indicates if setting the current module was successful. An error is thrown, if $oid is not a module OID.

Usage:

`gui schem setCurrentModule ?-activate? ?-window windowValue? moduleOid`

Parameters:

`-activate (optional)`

If the option is present, then the Schem window tab is activated.

`-window windowValue (optional default is NULL)`

Path or name of a Schem window, "DEFAULT" or "" for the default Schem window.

`moduleOid`

Module type OID to be set as the current module.

Example:

```
set oid {module TOP TOP}
gui schem setCurrentModule $oid
```

gui schem unfold

Unfold the given modules in the Schem window.

Usage:

```
gui schem unfold ?-window windowValue? oidList
```

Parameters:

`-window windowValue` (optional default is NULL)

Path or name of a Schem window, "DEFAULT" or "" for the default Schem window.

`oidList`

List of objects to unfold.

Example:

```
set oidList [gui schem contents]
gui schem unfold $oidList
```

gui schem zoom

Changes the zoom factor of the Schem window.

Usage:

```
gui schem zoom ?-window windowValue? factor
```

Parameters:

`-window windowValue` (optional default is NULL)

Path or name of a Schem window, "DEFAULT" or "" for the default Schem window.

`factor`

The argument $factor must be "fullfit" or a number ("fullfit" only works if that window is currently visible).

Example:

```
gui window show "Schem"
gui schem zoom "fullfit"
```

gui search

APIs to interact with the Search window.

gui search hideWindow

Hide the Search window.

Usage:

```
gui search hideWindow
```

Parameters:

Example:

```
gui search hideWindow
```

gui search selection

Returns all currently selected objects of the Search window.

Usage:

```
gui search selection ?-window windowValue?
```

Parameters:

`-window` **`windowValue`** `(optional default is NULL)`

Path or name of a Search window, "DEFAULT" or "" for the default Search window.

Example:

```
gui search selection
```

gui search showWindow

Show the Search window.

Usage:

```
gui search showWindow
```

Parameters:

Example:

```
gui search showWindow
```

gui selection

Work with the global selection.

gui selection get

Returns the most recently selected list of objects across all windows.

Usage:

```
gui selection get
```

Parameters:

No parameters.

Example:

```
foreach oid [gui selection get] {
    ##
    # Do something with each selected OID.
    #
}
```

gui selection registerCallback

Register a customer specific callback that is called every time the selection in the GUI changes. The list of selected OIDs is appended to the callback before evaluating it. To remove a previously registered "selection changed" callback use `gui selection removeCallback ⋯`.

Usage:

```
gui selection registerCallback callback
```

Parameters:

`callback`

The callback script.

Example:

```
proc mySelectionHandler {oidList} {
    ##
    # Selection changed to $oidList
    #
}
gui selection registerCallback "mySelectionHandler"
```

gui selection removeCallback

Remove a previously registered callback.

Usage:

```
gui selection removeCallback callback
```

Parameters:

`callback`

The first string of the previously registered callback script.

Example:

```
gui selection removeCallback "mySelectionHandler"
```

gui settings

All GUI settings can be controlled by the settings API. A list of all variables can be found in the manual of the Preferences dialog.

gui settings changed

Update the GUI after the settings have been modified.

Usage:

```
gui settings changed
```

Parameters:

No parameters.

Example:

```
##
# Display attributes in tooltips.
#
gui settings set "tooltipsWithAttrs" 1

##
# Autohide unconnected pins in the Cone window.
#
gui settings set "cone:autohide" 1

##
# Inform the GUI that the settings have been modified.
#
gui settings changed
```

gui settings get

Get the value of a global setting.

Usage:

```
gui settings get name
```

Parameters:

`name`
: The name of the global setting.

Example:

```
set value [gui settings get "my_setting"]
```

gui settings load

Load the settings from a file.

Note: After loading settings from a file, `gui settings changed` must be called to inform the application about the change.

Usage:

```
gui settings load filename scope
```

Parameters:

`filename`

File name.

`scope`

Scope to of the settings to load. Available scopes are workspace, project, history, volatile.

Example:

```
gui settings load "my-workspace.settings" "workspace"
gui settings changed
```

gui settings registerChangedCallback

Register a customer specific callback that is called every time the settings have changed. To remove a previously registered "settings changed" callback use `gui settings removeChangedCallback ⋯`.

Usage:

```
gui settings registerChangedCallback callback
```

Parameters:

`callback`

The callback script.

Example:

```
proc my_callback {} {
    ##
    # Settings have changed
    #
}
gui settings registerChangedCallback "my_callback"
```

gui settings removeChangedCallback

Remove a previously registered callback.

Usage:

```
gui settings removeChangedCallback callback
```

Parameters:

`callback`
: The first string of the previously registered callback script.

Example:

```
gui settings removeChangedCallback "my_callback"
```

gui settings reset

Reset all global settings matching the given name pattern to the built-in default value.

Usage:

```
gui settings reset ?-nameFilter nameFilterValue? scope
```

Parameters:

`-nameFilter nameFilterValue (optional default is "*")`
: Add a glob style pattern to filter the setting names.

`scope`
: Scope to reset settings to default. Available scopes are workspace, project, history, volatile.

Example:

```
gui settings reset workspace
```

gui settings save

Save the settings to a file.

Usage:

```
gui settings save filename scope
```

Parameters:

`filename`

File name.

`scope`

Scope to of the settings to save. Available scopes are workspace, project, history, volatile.

Example:

```
gui settings save "my-workspace.settings" "workspace"
```

gui settings set

Set a value of a global setting.

Usage:

```
gui settings set ?-type typeValue? name value
```

Parameters:

`-type typeValue (optional default is "String")`

The type of the setting value.

`name`

The name of the global setting.

`value`

The value to set to the global setting.

Example:

```
gui settings set "my_setting" 1
```

gui settings validate

Validate the value type of all settings.

Usage:

`gui settings validate`

Parameters:

No parameters.

Example:

```
gui settings validate
```

gui settings variable

Get the name of a variable for a global setting.

Usage:

`gui settings variable name`

Parameters:

`name`

The name of the global setting.

Example:

```
set variableName [gui settings variable "my_setting"]
```

gui source

APIs to interact with the Source window.

gui source displayFile

Display the contents of a file in the Source window.

Usage:

```
gui source displayFile ?-window windowValue? file lineno
```

Parameters:

`-window windowValue` `(optional default is NULL)`

Path or name of a Source window, "DEFAULT" or "" for the default Source window.

`file`

Filename of the file to display.

`lineno`

Number of the line to show.

Example:

```
gui source displayFile "design.v" 1
```

gui source gotoLine

Goto a given line in a given file in the Source window.

Usage:

```
gui source gotoLine ?-window windowValue? file lineno
```

Parameters:

`-window windowValue (optional default is NULL)`

Path or name of a Source window, "DEFAULT" or "" for the default Source window.

`file`

Filename of the source file.

`lineno`

Line number to goto.

Example:

```
gui source gotoLine "design.v" 1
```

gui source highlightLine

Highlight a line in the Source window. Each call to 'gui source highlightLine' will append the line highlight information.

Usage:

`gui source highlightLine ?-clear? ?-dontShow? ?-window windowValue? file lineno`

Parameters:

`-clear (optional)`

Clear the internal list of line highlights.

`-dontShow (optional)`

If this switch is set then the file will not be loaded and displayed at the given line number.

`-window windowValue (optional default is NULL)`

Path or name of a Source window, "DEFAULT" or "" for the default Source window.

`file`

Filename of the source file.

`lineno`

Line number to highlight.

Example:

```
gui source highlightLine "design.v" 100 -clear
```

gui source selection

Returns all currently selected objects of the Source window.

Usage:

`gui source selection ?-window windowValue?`

Parameters:

`-window windowValue (optional default is NULL)`

Path or name of a Source window, "DEFAULT" or "" for the default Source window.

Example:

```
gui source selection
```

gui source setActionbarDataCallback

Set the 'data' callback for the actionbar in all Source windows.

Usage:

`gui source setActionbarDataCallback callback`

Parameters:

`callback`

Callback to set. If this parameter is an empty string, the existing callback is removed.

Example:

```
proc myDataCallback {
    db
    oid
    fileName
    clickPosition
```

```
    oidBeginPosition
    oidEndPosition
} {
    # "db": the current database
    # "oid": the clicked OID
    # "fileName": the current file name
    # "clickPosition": the clicked position (byte index in the file)
    # "oidBeginPosition": the first byte index of the clicked OID
    # "oidEndPosition": the last byte index of the clicked OID

    # Note: if any of the following keys are missing from the dict, the
    # default functions are used to determine their values.
    # => If an empty dict is returned, the ActionBar will have default
    # behavior.
    set data [dict create]

    # "hide": bool
    # if 1, the ActionBar is hidden immediately
    dict set data "hide" 0

    # "label": string
    # the label that is displayed in the ActionBar's first row
    dict set data "label" "Clicked OID: [$db oid print $oid]"

    # "prev": empty or list<OID, string, number, number>
    # the previous occurrence of $oid;
    # empty or a list with
    # - an OID,
    # - a file-name,
    # - a begin position,
    # - an end position
    # The value is used for the ActionBar's "prev" button.
    dict set data "prev" {}

    # "next": empty or list<OID, string, number, number>
    # the next occurrence of $oid;
    # empty or a list with
    # - an OID,
    # - a file-name,
    # - a begin position,
    # - an end position
    # The value is used for the ActionBar's "next" button.
    dict set data "prev" [list $oid "myfile.v" 123 140]

    # "up": empty or OID
    # the up-object of $oid (see "$db oid up ...")
    # The value is used for the "up" button.
    dict set data "up" {}

    # "down": empty or OID
    # the down-object of $oid (see "$db oid down ...")
```

```
    # The value is used for the ActionBar's "down" button.
    dict set data "down" {module MOD MOD}

    # "instances": dict<string, OID>
    # the instantiations of $oid; the dict's keys are displayed in
    # insertion order in the ActionBar's "instances" combo box.
    # The special key "[...]" may be used to indicate
    # that there are additional instantiations that are not
    # computed/displayed e.g. due to performance reasons.
    dict set data "instances"  [dict create "inst1" {inst TOP INST1} "inst2" {inst TOP
INST2}]

    # "selectedInstance": string
    # the item initially selected in the ActionBar's "instances" combo box.
    dict set data "selectedInstance" "inst1"

    # "connections": dict<string, dict<string, list<OID, type>>>
    # The connections of $oid, i.e. the connected pins.
    # The outer dict is for wide, multi-bit connections; its keys are
    # displayed in the ActionBar's "bit select" combo box. If there's just
    # a single-bit connection, the outer dict must have exactly one key that
    # must be the empty string.
    # The inner dict is for the actual single-bit connections; its keys are
    # displayed in the ActionBar's "pin select" combo box; its values are
    # pairs of a pin/port OID and a type field which indicates if the
    # pin connection is a driver (type="driver"), a load (type="load") or
    # an ordinary connection (type="").
    # The special key "[...]" may be used to indicate that there are
    # additional connections that are not computed/displayed e.g. due to
    # performance reasons.
    # The driver pins (if there are any) are used for the ActionBar's
    # "driver" button. The load pins are used for the ActionBar's
    # "load" button.
    set connections [dict create]
    dict set connections  "bit 0" "pin 0 (driver)" [list {port TOP clk} "driver"]
    dict set connections  "bit 0" "pin 1 (load)" [list {pin TOP INST1 clk} "load"]
    dict set connections  "bit 0" "pin 2 (other)" [list {pin TOP INST2 in} ""]
    dict set connections  "bit 0" "\[...\]" [list {} ""]
    dict set connections  "bit 1" "pin 0" [list {pin TOP INST2 o} ""]
    dict set data "connections" $connections

    # "selectedConnection": list<string, string>
    # The items initially selected in the ActionBar's "select bit" and
    # "select pin" combo boxes.
    dict set data "selectedConnection" [list "bit 0" "pin 1 (load)"]

    return $data
}

# set the data callback
gui source setActionbarDataCallback myDataCallback
```

```
# clear the data callback
gui source setActionbarDataCallback {}
```

gui tab

API to work with the tabbed pane.

gui tab getActive

Return the ID of the active window in the given tab window.

Usage:

```
gui tab getActive window
```

Parameters:

`window`

Path or name of a Tab window, or "DEFAULT" or "" for the default Tab window.

Example:

```
set tab [gui window defaultClassWindow Tab]
set active [gui tab getActive $tab]
```

gui tab getAvailableClasses

Return the list of window classes that can be created by the "+" button.

Usage:

```
gui tab getAvailableClasses ?-tabwindow tabwindowValue?
```

Parameters:

`-tabwindow` **`tabwindowValue`** `(optional default is NULL)`

Path or name of a Tab window, "DEFAULT" or "" for the default Tab window. If specified, set the available classes only in this Tab window.

Example:

```
set classes [gui tab getAvailableClasses]
```

gui tab isWindowContainer

Check if the given Tab window is a general container for windows (i.e. Schem, Cone, Source, ...) windows can be added to it.

Usage:

```
gui tab isWindowContainer window
```

Parameters:

`window`

Path or name of a Tab window, or "DEFAULT" or "" for the default Tab window.

Example:

```
gui window foreach Tab w {
    if {[gui tab isWindowContainer $w]} {
        gui console print "$w is a window container"
    } else {
        gui console print "$w is NOT a window container"
    }
}
```

gui tab setAvailableClasses

Set the list of window classes that can be created by the "+" button.

Usage:

```
gui tab setAvailableClasses ?-tabwindow tabwindowValue? classes
```

Parameters:

`-tabwindow` **`tabwindowValue`** `(optional default is NULL)`

Path or name of a Tab window, "DEFAULT" or "" for the default Tab window. If specified, set the available classes only in this Tab window.

`classes`

List of window classes.

Example:

```
set classes {Schem Cone Source}
gui tab setAvailableClasses $classes
```

gui toolbar

APIs to interact with the Toolbars.

gui toolbar addButton

Add a button to the toolbar of the specified window.

Usage:

```
gui toolbar addButton ?-command commandValue? ?-position positionValue? ?-tooltip tooltipValue? window name imageset
```

Parameters:

`-command commandValue (optional default is NULL)`

The command to execute when the button is clicked.

`-position positionValue (optional default is "end")`

The position in the toolbar. Some number or "end".

`-tooltip tooltipValue (optional default is NULL)`

The tooltip to display.

`imageset`

Name of the image set to be used as the button's icon. Use `gui imageset load ⋯` to load a suitable image set.

`name`

The name of the toolbar button.

`window`

Path or name of a window. Use `""` or `"."` in order to modify the main window's toolbar.

Example:

```
proc buttonAction {} {
    ##
    # Do something...
    #
}

# Load "myPath/myImage-16.png", "myPath/myImage-24.png", ....
gui imageset load "myImageSet" "myPath/myImage-SIZE.png"

gui toolbar addButton "" "myButton" "myImageSet"  -command buttonAction  -tooltip
"this is my button"
```

gui toolbar addCheckButton

Add a checkable button to the toolbar of the specified window.

Usage:

`gui toolbar addCheckButton ?-command commandValue? ?-position positionValue? ?-tooltip tooltipValue? window name imageset variable`

Parameters:

`-command commandValue (optional default is NULL)`

The command to execute when the button is clicked.

`-position positionValue (optional default is "end")`

The position in the toolbar. Some number or "end".

`-tooltip tooltipValue (optional default is NULL)`

The tooltip to display.

`imageset`

Name of the image set to be used as the button's icon. Use `gui imageset load ⋯` to load a suitable image set.

`name`

The name of the toolbar button.

`variable`

The variable holding the check button's "checked" state.

`window`

Path or name of a window. Use `""` or `"."` in order to modify the main window's toolbar.

Example:

```
proc checkButtonAction {} {
    ##
    # Do something...
    #
}

# Load "myPath/myImage-16.png", "myPath/myImage-24.png", ....
gui imageset load "myImageSet" "myPath/myImage-SIZE.png"

set checkButtonState 0
gui toolbar addCheckButton "" "myCheckButton"  "myImageSet" "checkButtonState"
-command checkButtonAction  -tooltip "this is my check button"
```

gui toolbar addCustomWidget

Add a custom widget to the toolbar of the specified window.

Usage:

```
gui toolbar addCustomWidget ?-fill fillValue? ?-position positionValue? ?-tooltip tooltipValue?
window name
```

Parameters:

`-fill fillValue` (optional default is NULL)

Stretch direction. "x", "y", "xy", or "".

`-position positionValue` (optional default is "end")

The position in the toolbar. Some number or "end".

`-tooltip tooltipValue` (optional default is NULL)

The tooltip to display.

`name`

The widget to add. There must exist a widget $toolbar.$name where $toolbar is the widget path for the toolbar (see 'gui toolbar getPath ...').

`window`

Path or name of a window. Use "" or "." in order to modify the main window's toolbar.

Example:

```
set toolbar [gui toolbar getPath ""]
ttk::label $toolbar.customWidget -text "my label"
gui toolbar addCustomWidget "" "customWidget" -tooltip "this is my label"
```

gui toolbar addSeparator

Add a separator (vertical line) to the toolbar of the specified window.

Usage:

```
gui toolbar addSeparator ?-position positionValue? window name
```

Parameters:

`-position positionValue (optional default is "end")`

The position in the toolbar. Some number or "end".

`name`

The name of the separator to add.

`window`

Path or name of a window. Use `""` or `"."` in order to modify the main window's toolbar.

Example:

```
gui toolbar addSeparator "" "mySeparator"
```

gui toolbar addSpacer

Add a spacer to the toolbar of the specified window. A spacer pushes the following toolbar items to the right and takes all the remaining horizontal space.

Usage:

```
gui toolbar addSpacer ?-position positionValue? window name
```

Parameters:

`-position positionValue (optional default is "end")`

The position in the toolbar. Some number or "end".

`name`

The name of the spacer to add.

`window`

Path or name of a window. Use `""` or `"."` in order to modify the main window's toolbar.

Example:

```
gui toolbar addSpacer "" "mySpacer"
```

gui toolbar addTextButton

Add a text button to the toolbar of the specified window.

Usage:

```
gui toolbar addTextButton ?-command commandValue? ?-position positionValue? ?-tooltip tooltipValue? ?-width widthValue? window name text
```

Parameters:

`-command commandValue (optional default is NULL)`

The command to execute when the button is clicked.

`-position positionValue (optional default is "end")`

The position in the toolbar. Some number or "end".

`-tooltip tooltipValue (optional default is NULL)`

The tooltip to display.

`-width widthValue (optional default is NULL)`

The width of the button (in characters).

`name`

The name of the toolbar button.

`text`

The text to display in the toolbar button.

`window`

Path or name of a window. Use `""` or `"."` in order to modify the main window's toolbar.

Example:

```
proc textButtonAction {} {
    ##
    # Do something...
    #
}
gui toolbar addTextButton "" "myTextButton" "Text"  -command textButtonAction
-tooltip "this is my text button"
```

gui toolbar children

Get the names of all children of the toolbar of the specified window.

Usage:

```
gui toolbar children window
```

Parameters:

`window`

Path or name of a window. Use "" or "." in order to modify the main window's toolbar.

Example:

```
# Get the list of all children of the main toolbar.
set children [gui toolbar children ""]
```

gui toolbar getPath

Determine the path of the toolbar widget of the specified window.

Usage:

```
gui toolbar getPath window
```

Parameters:

`window`

Path or name of a window. Use "" or "." in order to get the path of the main window's toolbar.

Example:

```
set mainToolbar [gui toolbar getPath ""]
set schemToolbar [gui toolbar getPath "Schem"]
```

gui toolbar remove

Remove an item from the toolbar of the specified window.

Usage:

```
gui toolbar remove window name
```

Parameters:

`name`

The toolbar item's name.

`window`

Path or name of a window. Use `""` or `"."` in order to modify the main window's toolbar.

Example:

```
gui toolbar remove "" "mySpacer"
```

gui tooltip

Interact with the tooltips.

gui tooltip registerCallback

Register a callback to get additional information to be displayed in the tooltip for an object. The registered callback procedure is called with two additional arguments, the selected object and the name of a variable to customize the tooltip type. Supported types are 'help' or 'simple'. If the type is not set, then the default tooltip type is created.

Usage:

```
gui tooltip registerCallback callback
```

Parameters:

`callback`

The callback script.

Example:

```
proc getMyTooltip {oid tooltipTypeVarname} {
    upvar 1 $tooltipTypeVarname tooltipType

    set tooltipType "simple"

    return "This is my tooltip text for $oid."
}
gui tooltip registerCallback "getMyTooltip"
```

gui tooltip removeCallback

Remove a previously registered callback to get additional tooltip text.

Usage:

`gui tooltip removeCallback`

Parameters:

No parameters.

Example:

```
gui tooltip removeCallback
```

gui tree

APIs to interact with the Tree window.

gui tree getCurrentModule

Get the OID of the current/active module of the Tree window.

Usage:

```
gui tree getCurrentModule ?-window windowValue?
```

Parameters:

`-window windowValue` (optional default is NULL)

Path or name of a Tree window, "DEFAULT" or "" for the default Tree window.

Example:

```
set top {module top top}
set mod {module top i submodule}

gui tree setCurrentModule $mod
set oid [gui tree getCurrentModule] ;# => {module top i submodule}

gui tree setCurrentModule $top
set oid [gui tree getCurrentModule]  ;# => {module top top}
```

gui tree selection

Returns all currently selected objects of the Tree window.

Usage:

```
gui tree selection ?-window windowValue?
```

Parameters:

`-window windowValue` (optional default is NULL)

Path or name of a Tree window, "DEFAULT" or "" for the default Tree window.

Example:

```
gui tree selection
```

gui tree setCurrentModule

Set this module as the current/active module in the Tree window.

Usage:

```
gui tree setCurrentModule ?-window windowValue? moduleOid
```

Parameters:

`-window windowValue` (optional default is NULL)

Path or name of a Tree window, "DEFAULT" or "" for the default Tree window.

`moduleOid`

Module type OID to be set as the current module.

Example:

```
gui tree setCurrentModule {module top i submodule}
```

gui tree setTopModule

THIS COMMAND IS OBSOLETE!

Usage:

```
gui tree setTopModule ?-window windowValue? moduleOid
```

Parameters:

`-window windowValue` (optional default is NULL)

Path or name of a Tree window, "DEFAULT" or "" for the default Tree window.

`moduleOid`

Module type OID to be set as the top module.

Example:

```
catch {gui tree setTopModule {}}
```

gui wave

APIs to interact with the Wave window.

gui wave addAlias

Add wave alias, create scopes if needed.

Usage:

```
gui wave addAlias ?-window windowValue? sigOid newOid
```

Parameters:

`-window windowValue (optional default is NULL)`

Path or name of a Wave window, "DEFAULT" or "" for the default Wave window. If window parameter is empty, this command has effect on all Wave windows.

`newOid`

The alias OID.

`sigOid`

A signal OID.

Example:

```
gui wave addAlias {signal TOP CLK} {net TOP U1 IN}
```

gui wave addToGroup

Add the signals of the given nets to a existing wave group.

Usage:

```
gui wave addToGroup ?-oid? ?-window windowValue? groupIdentifier netOids
```

Parameters:

`-oid (optional)`

Boolean value to indicate if the identifier is an OID.

`-window windowValue (optional default is NULL)`

Path or name of a Wave window, "DEFAULT" or "" for the default Wave window. If window parameter is empty, this command has effect on all Wave windows.

`groupIdentifier`

Name or OID of the existing group.

`netOids`

The OID list of the nets.

Example:

```
gui wave addToGroup GrpIn {{net TOP U1 IN}}
```

gui wave bindToCone

Bind a Wave window to a Cone window for displaying logic values.

Usage:

`gui wave bindToCone wave cone`

Parameters:

`cone`

The Cone window. If the Cone window is an empty string then the binding is removed.

`wave`

The Wave window.

Example:

```
gui wave bindToCone Wave Cone
```

gui wave clear

Clear the contents of the Wave window.

Usage:

`gui wave clear ?-window windowValue?`

Parameters:

`-window windowValue` (optional default is NULL)

Path or name of a Wave window, "DEFAULT" or "" for the default Wave window. If window parameter is empty, this command has effect on all Wave windows.

Example:

```
gui wave clear
```

gui wave clearHighlightTime

Clear all highlighted time ranges in the Wave window.

Usage:

`gui wave clearHighlightTime ?-window windowValue? oidList`

Parameters:

`-window windowValue` (optional default is NULL)

Path or name of a Wave window, "DEFAULT" or "" for the default Wave window. If window parameter is empty, this command has effect on all Wave windows.

`oidList`

Clear all highlight information of the Wave window for all OIDs given with the argument $oidList. If $oidList is "*" then the highlighted time ranges for all displayed waveforms are cleared.

Example:

```
gui wave clearHighlightTime "*"
```

gui wave clearNameMarker

Clear the marker from the given signal names.

Usage:

`gui wave clearNameMarker ?-window windowValue? oidList`

Parameters:

`-window windowValue (optional default is NULL)`

Path or name of a Wave window, "DEFAULT" or "" for the default Wave window. If window parameter is empty, this command has effect on all Wave windows.

`oidList`

The list of OIDs to clear the name marker. If this option is an empty list or "*", then all name markers are cleared.

Example:

```
gui wave clearNameMarker $oidList
```

gui wave createGroup

Create a wave group and add the signals of the given nets to it. Return the OID of the created group.

Usage:

`gui wave createGroup ?-expandVectors? ?-hideMembers? ?-window windowValue? groupName netOids`

Parameters:

`-expandVectors (optional)`

Boolean value to indicate if the members of a given netBus should be expanded.

`-hideMembers (optional)`

Switch to indicate if the single bit members should be hidden.

`-window windowValue (optional default is NULL)`

Path or name of a Wave window, "DEFAULT" or "" for the default Wave window. If window parameter is empty, this command has effect on all Wave windows.

`groupName`

Name of the new group.

`netOids`

The OID list of the nets.

Example:

```
gui wave createGroup GrpIn {{net TOP U1 IN}}
```

gui wave customizeValueColor

Set a custom color for a specific signal and value.

Usage:

```
gui wave customizeValueColor ?-window windowValue? oid what color
```

Parameters:

`-window windowValue` (optional default is NULL)

Path or name of a Wave window, "DEFAULT" or "" for the default Wave window. If window parameter is empty, this command has effect on all Wave windows.

`color`

The color value to draw the waveform. If the color value is an empty string then the customization is removed.

`oid`

Customize the wave value color for this OID.

`what`

Specify what part of the waveform should be customized. What can be one of: ValueTop, ValueBot, ValueX, ValueXFill, ValueZ, ValueZFill or ValueChange.

Example:

```
gui wave customizeValueColor {net top CLK} ValueTop #ff1d77
gui wave customizeValueColor {net top CLK} ValueBot #ff1d77
```

gui wave databaseModified

Trigger update on the Wave window if wave database has been modified.

Usage:

```
gui wave databaseModified ?-window windowValue?
```

Parameters:

`-window windowValue (optional default is NULL)`

Path or name of a Wave window, "DEFAULT" or "" for the default Wave window. If window parameter is empty, this command has effect on all Wave windows.

Example:

```
gui wave databaseModified
```

gui wave getDatabase

Get the loaded wave database of the Wave window.

Usage:

`gui wave getDatabase ?-window windowValue?`

Parameters:

`-window windowValue (optional default is NULL)`

Path or name of a Wave window, "DEFAULT" or "" for the default Wave window. If window parameter is empty, this command has effect on all Wave windows.

Example:

```
set wdb [gui wave getDatabase]
```

gui wave getSignals

Get the list of loaded signals.

Usage:

`gui wave getSignals ?-window windowValue?`

Parameters:

`-window windowValue (optional default is NULL)`

Path or name of a Window window, "DEFAULT" or "" for the default Window window. If window parameter is empty, this loaded signals of the default Wave window are returned.

Example:

```
set signalList [gui wave getSignals]
```

gui wave getTimeRange

Returns a list containing the visible time in the Wave window, i.e. "From:" and "To:" timesteps. Returns {} if the timestep is not set.

Usage:

```
gui wave getTimeRange ?-timestep? ?-window windowValue?
```

Parameters:

`-timestep (optional)`

Returns the timesteps instead of the time.

`-window windowValue (optional default is NULL)`

Path or name of a Wave window, "DEFAULT" or "" for the default Wave window. If window parameter is empty, this command returns the values for the default window.

Example:

```
gui wave getTimeRange
# -> e.g. {10us 30us}

gui wave getTimeRange -timestep
# -> e.g. {100 300}
```

gui wave hideWindow

Hide the default Wave window.

Usage:

`gui wave hideWindow`

Parameters:

Example:

```
gui wave hideWindow
```

gui wave highlightTime

Highlight a given time range in the Wave window.

Usage:

`gui wave highlightTime ?-window windowValue? from to color oidList`

Parameters:

`-window windowValue (optional default is NULL)`

Path or name of a Wave window, "DEFAULT" or "" for the default Wave window. If window parameter is empty, this command has effect on all Wave windows.

`color`

Highlight color number.

`from`

Start time.

`oidList`

Apply the highlight information in the Wave window for all OIDs given with the argument $oidList. If $oidList is "*" then the highlight is added to all displayed waveforms.

`to`

End time.

Example:

```
gui wave highlightTime 100 300 0 "*"
```

gui wave highlightTimes

Provide a list with highlight information to highlight a given time range in the Wave window.

Usage:

```
gui wave highlightTimes ?-nocheck? ?-window windowValue? highlightInfoListName
```

Parameters:

`-nocheck (optional)`

Do not check the given values.

`-window windowValue (optional default is NULL)`

Path or name of a Wave window, "DEFAULT" or "" for the default Wave window. If window parameter is empty, this command has effect on all Wave windows.

`highlightInfoListName`

The name of the highlight info list variable, where the list consists of the same arguments as `gui wave highlightTime`. For each OID, the highlight info arguments can be repeated multiple times: {oid from to color ?oid from to color? ...}.

Example:

```
set highlightInfoList {
    $oid1  100  300 0
    $oid2  200  600 1
    $oid3 1000 2600 0
    $oid1  600  800 1
}
gui wave highlightTimes "highlightInfoList"
```

gui wave loadSignals

Load list of signals to file.

Usage:

```
gui wave loadSignals ?-window windowValue? fileName
```

Parameters:

`-window windowValue` (optional default is NULL)

Path or name of a Wave window, "DEFAULT" or "" for the default Wave window. If window parameter is empty, this command has effect on all Wave windows.

`fileName`

File name of input file.

Example:

```
gui wave loadSignals "file"
```

gui wave nextValueChange

Jump to next value change.

Usage:

`gui wave nextValueChange ?-window windowValue? oidList`

Parameters:

`-window windowValue` (optional default is NULL)

Path or name of a Wave window, "DEFAULT" or "" for the default Wave window. If window parameter is empty, this command has effect on all Wave windows.

`oidList`

List of objects to display.

Example:

```
gui wave nextValueChange {}
```

gui wave oid2varid

Convert an OID into a Varid.

Usage:

`gui wave oid2varid ?-window windowValue? oid`

Parameters:

`-window windowValue (optional default is NULL)`

Path or name of a Wave window, "DEFAULT" or "" for the default Wave window. If window parameter is empty, then the default Wave window is used.

`oid`

The OID to convert.

Example:

```
gui wave oid2varid $oid
```

gui wave open

Open a waveform database.

Usage:

```
gui wave open ?-buildGroups? ?-from fromValue? ?-nameOfDUT nameOfDUTValue? ?-pathToDUT pathToDUTValue? ?-restore? ?-to toValue? ?-window windowValue? filename
```

Parameters:

`-buildGroups (optional)`

Boolean value to indicate if groups should be created automatically.

`-from fromValue (optional default is "0")`

Limit the start time that is read from the waveform database.

`-nameOfDUT nameOfDUTValue (optional default is NULL)`

Name of the design under test. If not specified then the name to the DUT is guessed by a heuristic that compares hierarchical information from the waveform database against the loaded design.

`-pathToDUT pathToDUTValue (optional default is NULL)`

Path to the design under test. If not specified then the path to the DUT is guessed by a heuristic that compares hierarchical information from the waveform database against the loaded design.

`-restore (optional)`

Boolean value to indicate if the view of the windows should be restored.

`-to toValue (optional default is "end")`

Limit the end time until read from the waveform database.

`-window windowValue` `(optional default is NULL)`

Path or name of a Wave window, "DEFAULT" or "" for the default Wave window. If window parameter is empty, this command has effect on all Wave windows.

`filename`

The name of the waveform database to open.

Example:

```
gui wave open "design.wdb"
```

gui wave previousValueChange

Jump to previous value change.

Usage:

`gui wave previousValueChange ?-window windowValue? oidList`

Parameters:

`-window windowValue` `(optional default is NULL)`

Path or name of a Wave window, "DEFAULT" or "" for the default Wave window. If window parameter is empty, this command has effect on all Wave windows.

`oidList`

List of objects to display.

Example:

```
gui wave previousValueChange {}
```

gui wave registerAddSignalCallback

Register a callback that is evaluated every time before new signals are added to the Wave window. Before evaluating the callback, the window identifier of the Wave window responsible for the change is appended as well as the list of signals.

Usage:

```
gui wave registerAddSignalCallback callback
```

Parameters:

`callback`

The callback script to be evaluated.

Example:

```
proc wave_add_signal {w oidList} {
    gui console print "Wave window $w: add signal: $oidList"
}

# add callback
gui wave registerAddSignalCallback wave_add_signal

# remove callback
gui wave removeAddSignalCallback wave_add_signal
```

gui wave registerMarkerChangedCallback

Register a callback that is evaluated every time the time marker in a Wave window has changed. Before evaluating the callback, the Wave window identifier and the current time value are appended. If a Wave window is cleared (and thus the time marker is deleted), the empty string is appended to the callback instead of the current time value.

Usage:

```
gui wave registerMarkerChangedCallback callback
```

Parameters:

`callback`

The callback script to be evaluated.

Example:

```
proc wave_marker_changed {w time} {
    if {$time != ""} {
        set msg "Wave window $w: the time marker has been moved to $time"
    } else {
        set msg "Wave window $w: the time marker has been removed"
    }
    gui console print $msg
}

# add callback
gui wave registerMarkerChangedCallback wave_marker_changed

# remove callback
gui wave removeMarkerChangedCallback wave_marker_changed
```

gui wave registerSelectionChangedCallback

Register a callback that is evaluated every time the selection of the loaded signal list has changed. Before evaluating the callback, the Wave window identifier and the list of selected objects are added.

Usage:

```
gui wave registerSelectionChangedCallback callback
```

Parameters:

`callback`

The callback script to be evaluated.

Example:

```
proc selectionChanged {w oidList} {
    set msg "Wave window $w: the selection has changed to ${oidList}."
    gui console print $msg
}

# add callback
gui wave registerSelectionChangedCallback selectionChanged

# remove callback
gui wave removeSelectionChangedCallback selectionChanged
```

gui wave registerTimeRangeChangedCallback

Register a callback that is evaluated every time the displayed time range in a Wave window has changed. Before evaluating the callback, the Wave window identifier and the current from and to time values are appended.

Usage:

```
gui wave registerTimeRangeChangedCallback callback
```

Parameters:

`callback`

The callback script to be evaluated.

Example:

```
proc waveTimeRangeChanged {w from to} {
    gui console print "Wave window $w: Time range changed to $from : $to."
}

# add callback
gui wave registerTimeRangeChangedCallback waveTimeRangeChanged

# remove callback
gui wave removeTimeRangeChangedCallback waveTimeRangeChanged
```

gui wave registerTreeStateChangedCallback

Register a callback that is evaluated every time the state of the signal tree has changed. Before evaluating the callback, the Wave window identifier and the affected object as well as the open state is added.

Usage:

```
gui wave registerTreeStateChangedCallback callback
```

Parameters:

`callback`

The callback script to be evaluated.

Example:

```
proc tree_state_changed {w oid open} {
    set msg "Wave window $w: the tree item $oid state changed to "
    if {$open} {
        append msg "open"
    } else {
        append msg "close"
    }
    gui console print $msg
}

# add callback
gui wave registerTreeStateChangedCallback tree_state_changed

# remove callback
gui wave removeTreeStateChangedCallback tree_state_changed
```

gui wave registerValuesChangedCallback

Register a callback that is evaluated every time the @waveValue/@waveMarks DB attributes have been changed as an effect of e.g. the user selecting a time step in the Wave window. There may be spurious evaluations of the callback, e.g. when the user selects a new time step with exactly the same Wave values as before. Before evaluating the callback, the window identifier of the Wave window responsible for the change is appended.

Usage:

```
gui wave registerValuesChangedCallback callback
```

Parameters:

`callback`

The callback script to be evaluated.

Example:

```
proc wave_values_changed {w} {
    gui console print "Wave window $w: values changed"
}

# add callback
gui wave registerValuesChangedCallback wave_values_changed

# remove callback
gui wave removeValuesChangedCallback wave_values_changed
```

gui wave removeAddSignalCallback

Remove the callback previously registered with `gui wave registerAddSignalCallback ⋯`.

Usage:

`gui wave removeAddSignalCallback callback`

Parameters:

`callback`
: The callback script to remove.

Example:

```
proc wave_add_signal {w} {
    gui console print "Wave window $w: add signal: $oidList"
}

# add callback
gui wave registerAddSignalCallback wave_add_signal

# remove callback
gui wave removeAddSignalCallback wave_add_signal
```

gui wave removeMarkerChangedCallback

Remove the callback previously registered with `gui wave registerMarkerChangedCallback ⋯`.

Usage:

`gui wave removeMarkerChangedCallback callback`

Parameters:

`callback`

The callback script to remove.

Example:

```
proc wave_marker_changed {w time} {
    if {$time != ""} {
        set msg "Wave window $w: the time marker has been moved to $time"
    } else {
        set msg "Wave window $w: the time marker has been removed"
    }
    gui console print $msg
}

# add callback
gui wave registerMarkerChangedCallback wave_marker_changed

# remove callback
gui wave removeMarkerChangedCallback wave_marker_changed
```

gui wave removeSelectionChangedCallback

Remove the callback previously registered with `gui wave registerSelectionChangedCallback ⋯`.

Usage:

`gui wave removeSelectionChangedCallback callback`

Parameters:

`callback`

The callback script to remove.

Example:

```
proc selectionChanged {w oidList} {
    set msg "Wave window $w: the selection has changed to ${oidList}."
    gui console print $msg
}

# add callback
gui wave registerSelectionChangedCallback selectionChanged

# remove callback
gui wave removeSelectionChangedCallback selectionChanged
```

gui wave removeTimeRangeChangedCallback

Remove the callback previously registered with `gui wave registerTimeRangeChangedCallback ⋯`.

Usage:

`gui wave removeTimeRangeChangedCallback callback`

Parameters:

`callback`
: The callback script to remove.

Example:

```
proc waveTimeRangeChanged {w from to} {
    gui console print "Wave window $w: Time range changed to $from : $to."
}

# add callback
gui wave registerTimeRangeChangedCallback waveTimeRangeChanged

# remove callback
gui wave removeTimeRangeChangedCallback waveTimeRangeChanged
```

gui wave removeTreeStateChangedCallback

Remove the callback previously registered with `gui wave registerTreeStateChangedCallback ⋯`.

Usage:

`gui wave removeTreeStateChangedCallback callback`

Parameters:

`callback`
: The callback script to remove.

Example:

```
proc tree_state_changed {w oid open} {
    set msg "Wave window $w: the tree item $oid state changed to "
    if {$open} {
        append msg "open"
    } else {
        append msg "close"
    }
    gui console print $msg
}

# add callback
gui wave registerTreeStateChangedCallback tree_state_changed

# remove callback
gui wave removeTreeStateChangedCallback tree_state_changed
```

gui wave removeValuesChangedCallback

Remove the callback previously registered with `gui wave registerValuesChangedCallback ⋯`.

Usage:

`gui wave removeValuesChangedCallback callback`

Parameters:

`callback`
: The callback script to remove.

Example:

```
proc wave_values_changed {w} {
    gui console print "Wave window $w: values changed"
}

# add callback
gui wave registerValuesChangedCallback wave_values_changed

# remove callback
gui wave removeValuesChangedCallback wave_values_changed
```

gui wave saveSignals

Save list of signals to file.

Usage:

```
gui wave saveSignals ?-window windowValue? fileName
```

Parameters:

`-window windowValue` (optional default is NULL)

Path or name of a Window window, "DEFAULT" or "" for the default Window window. If window parameter is empty, this command has effect on all Wave windows.

`fileName`

File name of output file.

Example:

```
gui wave saveSignals "file"
```

gui wave see

Adjust the view of the Wave window that the given time is visible.

Usage:

```
gui wave see ?-window windowValue? time
```

Parameters:

`-window windowValue` (optional default is NULL)

Path or name of a Wave window, "DEFAULT" or "" for the default Wave window. If window parameter is empty, this command has effect on all Wave windows.

`time`

Time to see.

Example:

```
gui wave see 300
```

gui wave selection

Returns all currently selected objects of the Wave window.

Usage:

```
gui wave selection ?-window windowValue?
```

Parameters:

`-window windowValue` (optional default is NULL)

Path or name of a Wave window, "DEFAULT" or "" for the default Wave window. If window parameter is empty, this command has effect on all Wave windows.

Example:

```
gui wave selection
```

gui wave setCursor

Set a named cursor to the given time. If no cursor with this name exists then a new cursor is created.

Usage:

```
gui wave setCursor ?-color colorValue? ?-window windowValue? name time
```

Parameters:

`-color colorValue (optional default is NULL)`

Color value in #RGB format to draw the cursor.

`-window windowValue (optional default is NULL)`

Path or name of a Wave window, "DEFAULT" or "" for the default Wave window. If window parameter is empty, this command has effect on all Wave windows.

`name`

Set the name for this cursor.

`time`

Time to set the cursor. Use an empty string to remove the cursor.

Example:

```
gui wave setCursor "Test Cursor" 300 -color #007700
```

gui wave setDatabase

Set the wave database which is used by the Wave window.

Usage:

`gui wave setDatabase ?-nameOfDUT nameOfDUTValue? ?-pathToDUT pathToDUTValue? ?-window windowValue? database`

Parameters:

`-nameOfDUT nameOfDUTValue (optional default is NULL)`

Name of the design under test.

`-pathToDUT pathToDUTValue (optional default is NULL)`

Path to the design under test.

`-window windowValue (optional default is NULL)`

Path or name of a Wave window, "DEFAULT" or "" for the default Wave window. If window parameter is empty, this command has effect on all Wave windows.

`database`

The new wave database.

Example:

```
gui wave setDatabase $wdb
```

gui wave setLabel

Set a text label at the given OID and time.

Usage:

```
gui wave setLabel ?-window windowValue? oid time label
```

Parameters:

`-window windowValue` (optional default is NULL)

Path or name of a Wave window, "DEFAULT" or "" for the default Wave window. If window parameter is empty, this command has effect on all Wave windows.

`label`

The text label to show.

`oid`

Object to add the label.

`time`

Time to add the label. Use an empty string to remove a label.

Example:

```
gui wave setLabel {net top CLK} 100 "Test"
```

gui wave setMarker

Set the time marker to the given time.

Usage:

```
gui wave setMarker ?-window windowValue? time
```

Parameters:

`-window windowValue` (optional default is NULL)

Path or name of a Wave window, "DEFAULT" or "" for the default Wave window. If window parameter is empty, this command has effect on all Wave windows.

`time`

Time to set a marker. Use an empty string to clear the marker.

Example:

```
gui wave setMarker 600
```

gui wave setNameMarker

Set a marker at the signal name.

Usage:

```
gui wave setNameMarker ?-color colorValue? ?-window windowValue? oidList
```

Parameters:

`-color colorValue` (optional default is "red")

Specify a color for the marker.

`-window windowValue` (optional default is NULL)

Path or name of a Wave window, "DEFAULT" or "" for the default Wave window. If window parameter is empty, this command has effect on all Wave windows.

`oidList`

The list of OIDs to set a name marker.

Example:

```
gui wave setNameMarker $oidList -color red
```

gui wave setSelection

Set selection to given signals of the Wave window.

Usage:

```
gui wave setSelection ?-window windowValue? oidList
```

Parameters:

`-window windowValue` (optional default is NULL)

Path or name of a Wave window, "DEFAULT" or "" for the default Wave window. If window parameter is empty, this command has effect on all Wave windows.

`oidList`

List of objects to display.

Example:

```
gui wave setSelection {}
```

gui wave setTimeRange

Set the visible time range of the Wave window.

Usage:

```
gui wave setTimeRange ?-window windowValue? from to
```

Parameters:

`-window windowValue` (optional default is NULL)

Path or name of a Wave window, "DEFAULT" or "" for the default Wave window. If window parameter is empty, this command has effect on all Wave windows.

`from`

The start time.

`to`

The end time.

Example:

```
gui wave setTimeRange 0 100
```

gui wave show

Display the waveform of the given signals in the Wave window.

Usage:

```
gui wave show ?-force? ?-insertIdx insertIdxValue? ?-member? ?-window windowValue? oidList
```

Parameters:

`-force (optional)`

If force is true then the given objects are always added.

`-insertIdx insertIdxValue (optional default is "")`

Insert position

`-member (optional)`

Do not convert a member into the corresponding bus.

`-window windowValue (optional default is NULL)`

Path or name of a Wave window, "DEFAULT" or "" for the default Wave window. If window parameter is empty, this command has effect on all Wave windows.

`oidList`

List of objects to display.

Example:

```
gui wave show {{net TOP CLK} {netBus TOP DATI}}
```

gui wave showMembers

Expand or collapse the member view of a netBus in the Wave window.

Usage:

```
gui wave showMembers ?-show? ?-window windowValue? netBusOid
```

Parameters:

`-show (optional)`

Boolean value to indicate if the view should be expanded or collapsed.

`-window windowValue (optional default is NULL)`

Path or name of a Wave window, "DEFAULT" or "" for the default Wave window. If window parameter is empty, this command has effect on all Wave windows.

`netBusOid`

OID of the netBus to expand or collapse.

Example:

```
gui wave showMembers -show {netBus TOP DATI}
```

gui wave showWindow

Show the default Wave window.

Usage:

`gui wave showWindow`

Parameters:

Example:

```
gui wave showWindow
```

gui wave signalState

Get the open/close state of the given signal. For an open bus signal true is returned. For a collapsed vector, group or for a scalar signal false is returned.

Usage:

`gui wave signalState ?-window windowValue? oid`

Parameters:

`-window windowValue (optional default is NULL)`

Path or name of a Window window, "DEFAULT" or "" for the default Window window. If window parameter is empty, this loaded signals of the default Wave window are returned.

`oid`

Get the state of this OID.

Example:

```
gui wave signalState {netBus TOP DATI}
```

gui wave string2oid

Convert a Varid into an OID.

Usage:

```
gui wave string2oid ?-bit bitValue? ?-window windowValue? str
```

Parameters:

`-bit bitValue (optional default is "")`

Get the OID for a bit of the given vector.

`-window windowValue (optional default is NULL)`

Path or name of a Wave window, "DEFAULT" or "" for the default Wave window. If window parameter is empty, then the default Wave window is used.

`str`

The string to convert.

Example:

```
gui wave string2oid $str
```

gui wave updateScope

Update the Wave window after modifying the scope of the currently loaded waveform database.

Usage:

```
gui wave updateScope ?-window windowValue?
```

Parameters:

`-window windowValue` (optional default is NULL)

Path or name of a Wave window, "DEFAULT" or "" for the default Wave window. If window parameter is empty, this command has effect on all Wave windows.

Example:

```
gui wave updateScope
```

gui wave varid2oid

Convert a Varid into an OID.

Usage:

`gui wave varid2oid ?-window windowValue? varid`

Parameters:

`-window windowValue` (optional default is NULL)

Path or name of a Wave window, "DEFAULT" or "" for the default Wave window. If window parameter is empty, then the default Wave window is used.

`varid`

The Varid to convert.

Example:

```
gui wave varid2oid $varid
```

gui wave zoom

Set the zoom factor of the Wave window.

Usage:

`gui wave zoom ?-window windowValue? factor`

Parameters:

`-window windowValue (optional default is NULL)`

Path or name of a Wave window, "DEFAULT" or "" for the default Wave window. If window parameter is empty, this command has effect on all Wave windows.

`factor`

The zoom factor. A positive value is zoom in, negative is zoom out, 0 is fullfit. Or use 'in', 'out' and 'full' for 0.1, -0.1 and 0.

Example:

```
gui wave zoom 0.5
```

gui window

Sub command dispatcher for window related GUI API commands.

gui window autoscroll

Autoscroll implements a new layouter. It grids the given scrollable into the given frame and adds scrollbars if required. Whenever the frame window's scroll region changes, then the scrollbars are added/removed.

Usage:

```
gui window autoscroll ?-direction directionValue? frame scrollable
```

Parameters:

`-direction directionValue (optional default is "x y")`

The direction of the scrollbars.

`frame`

The frame where the scrollable is placed to.

`scrollable`

The scrollable object.

Example:

```
set f [gui window insertCustomWidget "AutoscrollFrame"]
ttk::treeview $f.tree
gui window autoscroll $f $f.tree
```

gui window createHorizontalPane

Create a horizontal Pane window and insert it into the given main vertical Pane window. The new Pane window is appended to the list of existing horizontal Pane windows, except in the main top-level window, where it is inserted right above the status tab (which contains the Console and Messages windows). The insert position can be specified using the `-before`/`-after` options. Returns the new Pane's path.

Usage:

```
gui window createHorizontalPane ?-after afterValue? ?-before beforeValue? mainVerticalPane
```

Parameters:

`-after` **`afterValue`** `(optional default is NULL)`

Insert the new Pane after the given window in the vertical Pane window.

`-before` **`beforeValue`** `(optional default is NULL)`

Insert the new Pane before the given window in the vertical Pane window.

`mainVerticalPane`

The vertical Pane window to create the new Pane window in.

Example:

```
##
# Get the main vertical Pane window of the main top-level window.
#
set mainVertical [gui window getMainVerticalPane .]

##
# Create a horizontal Pane window in the main window.
#
set pane [gui window createHorizontalPane $mainVertical]
```

gui window createTab

Create a new Tab container window and insert it into the given vertical Pane window (the referenced vertical Pane window **must not** be the main vertical Pane of a top-level window!). The

insert position can be specified using the -before/-after options. Returns the new Tab's path.

Usage:

```
gui window createTab ?-after afterValue? ?-before beforeValue? verticalPane
```

Parameters:

`-after afterValue` (optional default is NULL)

Insert the new Tab after the given window in the vertical Pane window.

`-before beforeValue` (optional default is NULL)

Insert the new Tab before the given window in the vertical Pane window.

`verticalPane`

The vertical Pane window to create a Tab in.

Example:

```
##
# Get the main vertical Pane window of the main top-level window.
#
set mainVertical [gui window getMainVerticalPane .]

##
# Create a horizontal Pane window.
#
set horizontal [gui window createHorizontalPane $mainVertical]

##
# Create a vertical Pane window.
#
set vertical [gui window createVerticalPane $horizontal]

##
# Create a Tab window in $vertical.
#
set tab [gui window createTab $vertical]

##
# Insert a Schem window into $tab.
#
set schem [gui window new "Schem" -tabwindow $tab]
```

gui window createToplevel

Create a new top-level window and return its path. This also creates an obligatory vertical pane window, which can be retrieved with `gui window getMainVerticalPane $top`. The top-level window then has to be resized and placed properly, e.g. with `gui window modelessDialog ⋯`.

Usage:

```
gui window createToplevel
```

Parameters:

No parameters.

Example:

```
##
# Create a new top-level window.
#
set top [gui window createToplevel]

##
# Resize and place $top.
gui window modelessDialog  $top  "My Custom Top-Level Window"  -place "CENTER"  -size
{800 600}  -closeCallback "destroy $top"
```

gui window createVerticalPane

Create a vertical Pane window and insert it into the given horizontal Pane window. The new Pane window is appended to the list of existing vertical Pane windows. The insert position can be specified using the `-before`/`-after` options. Return the new Pane's path.

Usage:

```
gui window createVerticalPane ?-after afterValue? ?-before beforeValue? horizontalPane
```

Parameters:

`-after` **`afterValue`** `(optional default is NULL)`

Insert the new Pane after the given window in the horizontal Pane window.

`-before beforeValue (optional default is NULL)`

Insert the new Pane before the given window in the horizontal Pane window.

`horizontalPane`

The horizontal Pane window to create the new Pane window in.

Example:

```
##
# Get the main vertical Pane window of the main top-level window.
#
set mainVertical [gui window getMainVerticalPane .]

##
# Create a horizontal pane.
#
set horizontal [gui window createHorizontalPane $mainVertical]

##
# Create a vertical Pane window in new horizontal Pane window.
#
set pane [gui window createVerticalPane $horizontal]
```

gui window defaultClassWindow

Get the default `$class` window (if one exists).

Usage:

`gui window defaultClassWindow class`

Parameters:

`class`

Path or name of a window.

Example:

```
# Get the default Schem window.
set schem [gui window defaultClassWindow "Schem"]
gui console print "The default Schem window is $schem"
```

gui window dialog

This procedure initializes the window manager (wm) relation between the dialog ($dlg) and the dialog's parent window; then it deiconifies the dialog (in case it is withdrawn), grabs all events, waits until the user presses a button, and then ungrabs the events.

Usage:

```
gui window dialog ?-buttons buttonsValue? ?-focus focusValue? ?-parent parentValue? ?-place placeValue? ?-size sizeValue? ?-validate validateValue? dialog title
```

Parameters:

`-buttons buttonsValue (optional default is "")`

List of buttons. Return index of the pressed button.

`-focus focusValue (optional default is "")`

The widget that should get the focus.

`-parent parentValue (optional default is "")`

The parent widget.

`-place placeValue (optional default is "")`

Available modes: MOUSE - place dialog at current mouse location CENTER - center the dialog on its parent

`-size sizeValue (optional default is "")`

The desired size. A pair of numbers, e.g. "600 400".

`-validate validateValue (optional default is "")`

A validation callback. The callback is evaluated when the user clicks a dialog button. The index of the pressed button is appended to the callback before evaluation. If the callback returns 0, the dialog is **not** closed.

`dialog`

Top-level widget of the dialog.

`title`

The title string.

Example:

```
toplevel .dialog
gui window dialog .dialog "MyDialog"
```

gui window exists

Check if a window widget exists.

Usage:

```
gui window exists window
```

Parameters:

`window`

The window name or path to check.

Example:

```
if {![gui window exists "My Frame"]} {
    # Create the "My Frame" window.
}
```

gui window fileDialog

Show one of the file selection dialogs.

Usage:

```
gui window fileDialog ?-parent parentValue? mode title fileTypes
```

Parameters:

`-parent parentValue (optional default is ".")`

Path of the parent window.

`fileTypes`

Specify a list of supported file types.

`mode`

Depending on the value of $mode show an "Open File" dialog to select one file (openFile) or to select multiple files (openFiles), a "Save File" dialog (saveFile) or a "Choose Directory" dialog (chooseDir).

`title`

Text to display in the title bar of the file dialog.

Example:

```
gui window fileDialog openFiles "Open Tcl/Tk Files" {{"Tcl/Tk Files" .tcl}}
```

gui window foreach

Evaluate $body for all $class windows.

Usage:

```
gui window foreach class variable body
```

Parameters:

`body`
: The Tcl script to run. Supports the usual loop commands (e.g. "break") to control the actual loop execution.

`class`
: The class of windows to iterate over, e.g. "Schem".

`variable`
: This variable will be set to the current window before evaluating $body.

Example:

```
gui console print "All Schem windows:"
gui window foreach "Schem" w {
    gui console print $w
}
```

gui window foreachToplevel

Evaluate $body for all top-level windows.

Usage:

```
gui window foreachToplevel variable body
```

Parameters:

`body`

The Tcl script to run. Supports the usual loop commands (e.g. "break") to control the actual loop execution.

`variable`

This variable will be set to the current top-level window before evaluating $body.

Example:

```
gui window foreachToplevel w {
    gui console print "toplevel: $w"
}
```

gui window fullscreen

Toggle the fullscreen mode.

Usage:

```
gui window fullscreen showFullscreen
```

Parameters:

`showFullscreen`

Enable or disable the fullscreen mode.

Example:

```
# enable fullscreen mode
gui window fullscreen 1

# disable fullscreen mode
gui window fullscreen 0
```

gui window getClass

Get class of the given window.

Usage:

```
gui window getClass window
```

Parameters:

`window`

Path of the window.

Example:

```
set w [gui window getCurrent]
set class [gui window getClass $w]
```

gui window getCurrent

Get the currently focused window.

Usage:

`gui window getCurrent`

Parameters:

No parameters.

Example:

```
set w [gui window getCurrent]
```

gui window getDesignTitle

Get the design name as displayed in title bar of the main window.

Usage:

`gui window getDesignTitle`

Parameters:

No parameters.

Example:

```
set title [gui window getDesignTitle]
```

gui window getMainVerticalPane

Return the given top-level window's obligatory vertical Pane window.

Usage:

```
gui window getMainVerticalPane ?toplevel?
```

Parameters:

`toplevel (optional)`

The top-level window to get the vertical Pane window from.

Example:

```
##
# Get the vertical Pane window of the main top-level window.
#
set vertical [gui window getMainVerticalPane .]
```

gui window getPaneSashes

Get the sash positions of the given Pane window. The number of sash positions must is one less than the number of children of $pane. The sash positions are relative, i.e. each sash position is be a number between 0 and 1.

Usage:

```
gui window getPaneSashes pane
```

Parameters:

`pane`

The Pane window.

Example:

```
set sashes [gui window getPaneSashes $pane]
```

gui window getState

Get the state of the given window.

Usage:

```
gui window getState window
```

Parameters:

`window`
: Path or name of a window of class Cone, Mem, Schem, Source, Tree, or Wave.

Example:

```
set state [gui window getState Schem]
```

gui window getTab

Get the parent Tab window of the given `$window`. If `$window` is not the child of a Tab window, return an empty string.

Usage:

```
gui window getTab window
```

Parameters:

`window`
: Path of the window.

Example:

```
set w [gui window defaultClassWindow "Schem"]
set tab [gui window getTab $w]
```

gui window getToplevelTitle

Get the title of a top-level window as set by 'gui window setToplevelTitle'. If no title has been set, return an empty string.

Usage:

`gui window getToplevelTitle window`

Parameters:

`window`
: Name or path of the top-level window of one of the top-level's sub-windows.

Example:

```
set t [gui window getToplevelTitle .]
```

gui window hasClassWindow

Check if a window of the given `$class` exists.

Usage:

`gui window hasClassWindow class`

Parameters:

`class`
: Path or name of a window.

Example:

```
# Check if a Cone window exists; if not, create one.
if {![gui window hasClassWindow "Cone"]} {
    gui window new "Cone"
}
```

gui window hide

Hide the given window inside of its tab container. Hidden windows still exist, but are not accessible from the Tab bar.

Usage:

```
gui window hide window
```

Parameters:

`window`

Path or name of a window.

Example:

```
# hide the window named "Schem".
gui window hide "Schem"
```

gui window image

Create and return a Tk image of the given window.

Usage:

```
gui window image window
```

Parameters:

`window`

Path or name of a window.

Example:

```
set img [gui window image "Schem"]
$img write "schem.png"
image delete $img
```

gui window insertCustomWidget

Insert a custom widget into the bottom tab. The widget path to a frame is returned.

Usage:

`gui window insertCustomWidget ?-destroyCallback destroyCallbackValue? ?-fixed? ?-pluginNamespace pluginNamespaceValue? ?-tabwindow tabwindowValue? name`

Parameters:

`-destroyCallback destroyCallbackValue (optional default is NULL)`

This callback is evaluated when the custom widget is destroyed (e.g. when the user clicks the 'x' in the widget's tab).

`-fixed (optional)`

A fixed tab window cannot be renamed or closed.

`-pluginNamespace pluginNamespaceValue (optional default is NULL)`

Namespace of the Plugin creating the custom widget. When the custom widget is destroyed (e.g. when the user clicks the 'x' in the widget's tab) the `Finit` procedure in the given namespace will be called to unload and therewith disable the plugin.

`-tabwindow tabwindowValue (optional default is NULL)`

Path or name of a Tab window, "DEFAULT" or "" for the default Tab window. If specified, create the custom widget in this Tab window.

`name`

Name of the custom widget.

Example:

```
set f1 [gui window insertCustomWidget "My Frame 1"]
label $f1.l -text "This is a custom widget."
pack $f1.l

set tab [gui window defaultClassWindow "Tab"]
set f2 [gui window insertCustomWidget -tabwindow $tab "My Frame 2"]
label $f2.l -text "This is a custom widget in the default Tab."
pack $f2.l
```

gui window internal

Create an internal window $child in the parent $w.

Usage:

```
gui window internal ?-anchor anchorValue? ?-button buttonValue? parent child title width height
```

Parameters:

`-anchor anchorValue (optional default is "se")`

The anchor of the window.

`-button buttonValue (optional default is "")`

The button binding of the window.

`child`

The child window.

`height`

The height of the window.

`parent`

The parent window.

`title`

The title of the window.

`width`

The width of the window.

Example:

```
set width  100
set height 100
set f [gui window insertCustomWidget "InternalWindow"]
canvas $f.minimap -width $width -height $height
gui window internal $f $f.minimap "Minimap" $width $height
```

gui window isMaximized

Check if the given top-level window is maximized.

Usage:

```
gui window isMaximized toplevel
```

Parameters:

`toplevel`

The top-level window.

Example:

```
##
# Check if the main window is maximized.
#
if {[gui window isMaximized .]} {
    ##
    # Yes, it is maximized.
    #
} else {
    ##
    # No, it is not maximized.
    #
}
```

gui window isStatusPaneVisible

Check if the 'Status Pane' (containing Console and Messages windows) is currently displayed.

Usage:

```
gui window isStatusPaneVisible
```

Parameters:

No parameters.

Example:

```
if {![gui window isStatusPaneVisible]} {
    # The 'Status Pane" is currently visible.
}
```

gui window maximize

Maximize the given top-level window.

Usage:

```
gui window maximize toplevel
```

Parameters:

`toplevel`

The top-level window.

Example:

```
##
# Maximize the main window.
#
gui window maximize .
```

gui window modelessDialog

This procedure initializes the window manager (wm) relation between the dialog ($dlg) and the dialog's parent window; then it deiconifies the dialog (in case it is withdrawn).

Usage:

```
gui window modelessDialog ?-closeCallback closeCallbackValue? ?-onTop? ?-place placeValue? ?-size sizeValue? dialog title
```

Parameters:

`-closeCallback` **`closeCallbackValue`** `(optional default is "")`

A callback that is run when the dialog is closed.

`-onTop` `(optional)`

Keep the dialog on top of other windows.

`-place` **`placeValue`** `(optional default is "")`

Available modes: MOUSE - place dialog at current mouse location CENTER - center the dialog on the main window

`-size` **`sizeValue`** `(optional default is "")`

The desired size. A pair of numbers, e.g. "600 400".

`dialog`

Top-level widget of the dialog.

`title`

The title string.

Example:

```
toplevel .dialog
gui window modelessDialog .dialog "MyDialog"
```

gui window name

Get the given window's name.

Usage:

`gui window name window`

Parameters:

`window`

Path or name of a window.

Example:

```
set w    [::WindowManager::DefaultWindow Schem]
set name [gui window name $w]
gui console print "The default Schem window is called $name"
```

gui window new

Creates a new $class window. If `-tabwindow $window` is specified, the `$class` window is created inside the Tab window $window, otherwise it is created inside a new toplevel window. The return value is the path to the new `$class` window.

Usage:

`gui window new ?-fixed? ?-name nameValue? ?-position positionValue? ?-tabwindow tabwindowValue? ?-tag tagValue? class`

Parameters:

`-fixed (optional)`

A fixed tab window cannot be renamed or closed.

`-name nameValue (optional default is "")`

Name of the new window.

`-position positionValue (optional default is "end")`

The position in the Tab window. Some number or "end".

`-tabwindow tabwindowValue (optional default is NULL)`

Path or name of a Tab window, "DEFAULT" or "" for the default Tab window. If specified, create the new window in this Tab window.

`-tag tagValue (optional default is "")`

Tag of the new window.

`class`

Class of the window to create.

Example:

```
# create a toplevel window containing a single Schem window.
set schem [gui window new "Schem" -name "MySchemWindow"]

# do something with $schem...

# close the window
gui window remove $schem
```

gui window path

Convert a window name into a widget path.

Usage:

`gui window path name`

Parameters:

`name`

Name of the window.

Example:

```
if {[gui window path "My Frame"] eq ""} {
    # Create the "My Frame" window.
}
```

gui window registerCreatedCallback

Register a callback that's called whenever a window of the specified $class is created.

Usage:

`gui window registerCreatedCallback class callback`

Parameters:

`callback`

Callback script. The window path of the created $class window is appended to the script before evaluating it.

`class`

Window class to register a "Created" callback for.

Example:

```
proc my_callback {w} {
    gui console print "A new Schem window has been created: $w"
}

gui window registerCreatedCallback "Schem" my_callback
```

gui window registerCurrentChangedCallback

Register a callback that is called when the current window (i.e. the focused window) has changed. Only registered windows are reported.

Usage:

```
gui window registerCurrentChangedCallback callback
```

Parameters:

`callback`

Callback script. The window path of the newly focused window is appended to the script before evaluating it.

Example:

```
proc my_callback {w} {
    gui console print "Current window: $w"
}

# register callback
gui window registerCurrentChangedCallback my_callback

# remove it
gui window removeCurrentChangedCallback my_callback
```

gui window registerDestroyedCallback

Register a callback that's called whenever a window of the specified $class has been destroyed.

Usage:

```
gui window registerDestroyedCallback class callback
```

Parameters:

`callback`

Callback script. The window path of the destroyed $class window is appended to the script before evaluating it.

`class`

Window class to register a "Destroyed" callback for.

Example:

```
proc my_callback {w} {
    gui console print "A Schem window has been destroyed: $w"
}

gui window registerDestroyedCallback "Schem" my_callback
```

gui window registerDoubleClickCallback

Register a callback for double-click events in all $class windows.

Usage:

```
gui window registerDoubleClickCallback class callback
```

Parameters:

`callback`

Callback script. The window path of the double-clicked $class window is appended to the script before evaluating it. If the script returns "break" (e.g. "return -code break"), no other double-click actions are performed.

`class`

Window class to register a double-click callback for.

Example:

```
proc my_callback {w} {
    gui console print "Double click in Schem: $w"
}

gui window registerDoubleClickCallback "Schem" my_callback
```

gui window registerMoveCallback

Register a callback that is called when a window is moved to another pane.

Usage:

```
gui window registerMoveCallback callback
```

Parameters:

`callback`

Callback script. The new window path as well as the name of the window are appended to the script before evaluating it.

Example:

```
proc my_callback {w name} {
    gui console print "$name window moved, new path '$w'."
}

# register callback
gui window registerMoveCallback my_callback

# remove it
gui window removeMoveCallback my_callback
```

gui window registerNameChangedCallback

Register a callback that is called when the name of a window has changed.

Usage:

```
gui window registerNameChangedCallback callback
```

Parameters:

`callback`

Callback script. The window path of the window as well as the new name are appended to the script before evaluating it.

Example:

```
proc my_callback {w name} {
    gui console print "Rename window '$w' to '$name'"
}

# register callback
gui window registerNameChangedCallback my_callback

# remove it
gui window removeNameChangedCallback my_callback
```

gui window remove

Remove the specified window.

Usage:

`gui window remove window`

Parameters:

`window`

Path or name of a window.

Example:

```
# create a toplevel window containing a single Schem window.
set schem [gui window new "Schem"]

# remove the window
gui window remove $schem
```

gui window removeCreatedCallback

Remove a previously registered callback for "Created" events for $class windows.

Usage:

```
gui window removeCreatedCallback class callback
```

Parameters:

`callback`

The previously registered callback script.

`class`

Window class to remove a "Created" callback from.

Example:

```
proc my_callback {w} {
    gui console print "A new Schem window has been created: $w"
}

gui window registerCreatedCallback "Schem" my_callback

# remove it
gui window removeCreatedCallback "Schem" my_callback
```

gui window removeCurrentChangedCallback

Remove a previously registered callback for current changed events.

Usage:

```
gui window removeCurrentChangedCallback callback
```

Parameters:

`callback`

The previously registered callback script.

Example:

```
proc my_callback {w} {
    gui console print "Current window: $w"
}

# register callback
gui window registerCurrentChangedCallback my_callback

# remove it
gui window removeCurrentChangedCallback my_callback
```

gui window removeCustomWidget

Remove the custom widget given by its name.

Usage:

```
gui window removeCustomWidget name
```

Parameters:

`name`
: Name of the custom widget to be removed.

Example:

```
gui window removeCustomWidget "My Frame"
```

gui window removeDestroyedCallback

Remove a previously registered callback for "Destroyed" events for $class windows.

Usage:

```
gui window removeDestroyedCallback class callback
```

Parameters:

`callback`
: The previously registered callback script.

`class`
: Window class to remove a "Destroyed" callback from.

Example:

```
proc my_callback {w} {
    gui console print "A new Schem window has been destroyed: $w"
}

gui window registerDestroyedCallback "Schem" my_callback

# remove it
gui window removeDestroyedCallback "Schem" my_callback
```

gui window removeDoubleClickCallback

Remove a previously registered callback for double-click events in all $class windows.

Usage:

```
gui window removeDoubleClickCallback class callback
```

Parameters:

`callback`
: The previously registered callback script.

`class`
: Window class to remove a double-click callback from.

Example:

```
proc my_callback {w} {
    gui console print "Double click in Schem: $w"
}

gui window registerDoubleClickCallback "Schem" my_callback

# remove it
gui window removeDoubleClickCallback "Schem" my_callback
```

gui window removeMoveCallback

Remove a previously registered callback for window move events.

Usage:

```
gui window removeMoveCallback callback
```

Parameters:

`callback`

The previously registered callback script.

Example:

```
proc my_callback {w name} {
    gui console print "$name window moved, new path '$w'."
}

# register callback
gui window registerMoveCallback my_callback

# remove it
gui window removeMoveCallback my_callback
```

gui window removeNameChangedCallback

Remove a previously registered callback for name changed events.

Usage:

```
gui window removeNameChangedCallback callback
```

Parameters:

`callback`

The previously registered callback script.

Example:

```
proc my_callback {w name} {
    gui console print "Rename window '$w' to '$name'"
}

# register callback
gui window registerNameChangedCallback my_callback

# remove it
gui window removeNameChangedCallback my_callback
```

gui window rename

Rename the given window.

Usage:

```
gui window rename window newname
```

Parameters:

`newname`

The new name. Only ASCII characters minus ||, &&, ^, !, ^ and \\ are allowed for the name. Empty strings and strings only containing whitespaces are not allowed. If you have spaces in your name, you have to group it with "" or {}.

`window`

Path or name of a window.

Example:

```
set w [gui window new "Schem"]
set name [gui window name $w]
gui console print "Initial name: $name"

gui window rename $w "New Schem Name"

set name [gui window name $w]
gui console print "New name: $name"
```

gui window setDesignTitle

Set the design title and update the title bars of all top-level windows accordingly.

Usage:

```
gui window setDesignTitle ?-filename filenameValue? parserTitle
```

Parameters:

`-filename filenameValue` (optional default is NULL)
: Set a parser file name for the design title.

`parserTitle`
: The design title string to set.

Example:

```
gui window setDesignTitle "New Title" -filename "design.v"
```

gui window setGeometry

Set the given top-level window's geometry.

Usage:

```
gui window setGeometry toplevel geometry
```

Parameters:

`geometry`
: The window geometry in `WIDTHxHEIGHT+X+Y`-format, e.g. `1000x800+0+100`.

`toplevel`
: The top-level window.

Example:

```
gui window setGeometry . 1000x800+0+100
```

gui window setLabel

Give the window a custom label.

Usage:

```
gui window setLabel window label
```

Parameters:

`label`

The label the window shall get.

`window`

Path or name of the window (e.g. Tree, Schem, Cone, ...). Restrictions for the label are the same as for "rename".

Example:

```
# give the default Schem window a label:
gui window setLabel Schem "some label"
```

gui window setPaneSashes

Set the sash positions of the given Pane window.

Usage:

```
gui window setPaneSashes pane sashes
```

Parameters:

`pane`

The Pane window.

`sashes`

List of sash positions. The number of sash positions must be one less than the number of children of $pane. The sash positions are relative, i.e. each sash position must be a number between 0 and 1.

Example:

```
##
# Get the main vertical Pane window of the main top-level window.
#
set mainVertical [gui window getMainVerticalPane .]

##
# Create a horizontal Pane window.
#
set horizontal [gui window createHorizontalPane $mainVertical]

##
# Create a vertical Pane window.
#
set vertical [gui window createVerticalPane $horizontal]

##
# Create some Tab windows in $pane.
#
set tab1 [gui window createTab $vertical]
set tab2 [gui window createTab $vertical]
set tab3 [gui window createTab $vertical]

##
# Set the sash positions for $vertical.
# As $vertical has 3 children, we need to specify 2 sash positions.
# We set the sashes at 0.2 (20%) and 0.7 (70%) of the $vertical's height.
# => $tab1 will take 20% of $vertical's height, $tab2 will take 50%, and
#    $tab3 will take 30%.
#
gui window setPaneSashes $vertical {0.2 0.7}
```

gui window setState

Set the state of the given window.

Usage:

```
gui window setState window state
```

Parameters:

`state`

The state to restore. Must be the result of a previous call to `gui window getState ⋯`.

`window`

Path or name of a window of class Cone, Mem, Schem, Source, Tree, or Wave.

Example:

```
gui window setState Schem $state
```

gui window setTag

Tag the given window.

Usage:

```
gui window setTag window tag
```

Parameters:

`tag`
: The tag.

`window`
: Path or name of a window.

Example:

```
set mainVertical [gui window getMainVerticalPane .]
set pane [gui window createHorizontalPane $mainVertical]
gui window setTag $pane "MyTag"
```

gui window setToplevelTitle

Set the title of a top-level window. This overrides other title commands, e.g. 'gui window title ...'.

Usage:

```
gui window setToplevelTitle window title
```

Parameters:

`title`
: The string to display as the title.

`window`
: Name or path of the top-level window of one of the top-level's sub-windows.

Example:

```
gui window setToplevelTitle . "New Title"
```

gui window show

Activate/raise the given window inside of its tab container.

Usage:

```
gui window show window
```

Parameters:

`window`
: Path or name of a window.

Example:

```
# activate the window named "Schem".
gui window show "Schem"
```

gui window split

Split the given window by creating a new Tab window in the given direction. Returns the path of the newly created Tab window.

Usage:

```
gui window split window direction
```

Parameters:

`direction`
: Split direction (left, right, top, bottom).

`window`

Path or name of a window.

Example:

```
# create a toplevel window containing a single Schem window.
set schem [gui window new "Schem"]

# create a Tab window to the right
set tab [gui window split $schem right]

# insert a new Cone window into $tab
set cone [gui window new -tabwindow $tab "Cone"]

# remove everything ($cone is removed automatically when removing $tab)
gui window remove $schem
gui window remove $tab
```

gui window title

Set the title of the main window.

Usage:

```
gui window title title
```

Parameters:

`title`

The string to display as the title.

Example:

```
gui window title "New Title"
```

gui window unhide

Unhide the given window inside of its tab container.

Usage:

```
gui window unhide window
```

Parameters:

`window`
: Path or name of a window.

Example:

```
# unhide the window named "Schem".
gui window unhide "Schem"
```

gui window unmaximize

Unmaximize the given top-level window.

Usage:

```
gui window unmaximize toplevel
```

Parameters:

`toplevel`
: The top-level window.

Example:

```
##
# Unmaximize the main window.
#
gui window unmaximize .
```

gui busy

Show busy cursor and discard any pointer events for all windows.

Usage:

```
gui busy enable
```

Parameters:

`enable`

Reset busy to normal state.

Example:

```
gui busy true
gui busy false
```

gui goto

Display (at least one of) the objects in $oidList in the opened Schem, Cone and Source window (however, if the Cone window does not already contain those objects nothing is displayed). The objects in each window get selected - and in Schem and Cone additionally temporarily red-colored. Each window tries to avoid flickering, i.e. if one of the objects is already visible, then the display does not change. The target window(s) can be specified with the mutually exclusive '-class' and '-window' options. If neither of the two options is given, the 'goto' command is applied to all windows.

Usage:

`gui goto ?-class classValue? ?-noSelect? ?-window windowValue? oidList`

Parameters:

`-class` **`classValue`** `(optional default is NULL)`

If this parameter is given, the 'goto' command affects all existing windows of the specified class.

`-noSelect (optional)`

Select the newly added objects.

`-window` **`windowValue`** `(optional default is NULL)`

Path or name of a window of class Cone, Mem, Schem, Search, Source, or Tree. If this parameter is given, the 'goto' command only affects the specified window.

`oidList`

List of goto objects.

Example:

```
set oidList {
    {inst TOP U2 m2}
}
gui goto -class "Schem" $oidList
```

gui quit

Exit the application.

Usage:

`gui quit ?returnCode?`

Parameters:

`returnCode (optional)`

The returnCode is returned to the system as the exit status.

Example:

```
##
# Exit with the error code 2.
#
gui quit 2
```

gui registerQuitCallback

Register a callback that is called when the application is about to quit. If the callback evaluates to 0/false, the quit operation is aborted, otherwise it continues normally. Note that only one quit callback can be registered.

Usage:

`gui registerQuitCallback callback`

Parameters:

`callback`

Callback script.

Example:

```
proc my_callback {} {
    gui console print "The application will quit now!"
    return 1
}

# register callback
gui registerQuitCallback my_callback

# remove it
gui removeQuitCallback
```

gui removeQuitCallback

Remove the previously registered quit callback.

Usage:

```
gui removeQuitCallback
```

Parameters:

No parameters.

Example:

```
proc my_callback {} {
    gui console print "The application will quit now!"
    return 1
}

# register callback
gui registerQuitCallback my_callback

# remove it
gui removeQuitCallback
```

gui zoomTo

Like 'goto' but in addition zoom to the selected object. The target window(s) can be specified with the mutually exclusive '-class' and '-window' options. If neither of the two options is given, the 'goto' command is applied to all windows.

Usage:

```
gui zoomTo ?-class classValue? ?-window windowValue? oidList
```

Parameters:

`-class classValue` (optional default is NULL)

If this parameter is given, the 'zoomTo' command affects all existing windows of the specified class.

`-window windowValue` (optional default is NULL)

Path or name of a window of class Cone or Schem. If this parameter is given, the 'zoomTo' command only affects the specified window.

`oidList`

List of goto objects.

Example:

```
set oidList {
    {inst TOP U2 m2}
}
gui window show "Schem"
gui zoomTo -window "Schem" $oidList
```

The RTLvision PRO API

RTLvision PRO provides a rich API to access both, the database and the GUI by using the Tcl/Tk scripting language. Tcl/Tk scripts can be added to the tool at startup time (with the command line option `-userware script.tcl`), or later with the `source` command from the GUI Console window.

ZDB Database API

Document	Description
Database API	The **ZDB API** Reference Manual describes the functions to query and modify the database.
Primitives API	describes the primitive functions (like AND, OR, etc) of the database.
Flat View API	describes the API to the Flat View - e.g. set/get flat attributes.
Spos API	describes the API to set/get the Source-File positions of the database objects.
Cone Extraction	describes the "cone extraction" command (used to load the GUI Cone window).
Operator API	describes functions (algorithms) to modify the database.
Tools API	describes database tools (user-level functions).
Command Kit	describes database high-level commands created by a collection of already existing ZDB database commands.
Meta Attributes	describes how the "Meta Attributes" control the visibility of database attributes in the schematic windows (in the GUI Schem and Cone windows).
GUI Attributes	describes how the GUI behavior can be influenced by certain ZDB attributes.
CDC API	describes the Clock Domain Analyzer API.
ZDB Service Functions	Misc service functions.

ZDB API Examples and Tutorials

Document	Description
API Tutorial	The ZDB API Tutorial creates, step by step, some design rule checkers for finding floating nodes, heavy nodes or heavy C and R. It also shows how to customize the popup menu and add custom methods.
Examples	The API Examples documentation lists all Userware examples from the demo/api directory with a short description.

The GUI API

Document	Description
GUI API	The GUI API Reference Manual describes the functions to control the Graphical User Interface. This includes attaching attributes, highlighting objects, preloading the Cone window and much more.

Utility and Service Functions

Document	Description
Message Callbacks	Functions for messages.
Progress Bar	Functions for progress bars.
License Management	The license management API.
Utility Functions	Utility Functions
OS functions	Platform dependent system functions.
Debug Flags	List of all debug flags.

The Wave Database (WDB) API

Document	Description
WDB Tcl API	describes the Wave database Tcl-API.
WDB C API	describes the Wave database C-API.
WDB Read Tcl-API	describes the Wave database Read Tcl-API.
WDB Read C-API	describes the Wave database Read C-API.
wdbdemo.c	WDB usage example.

The Parser API

Document	Description
RTL Parser	describes the RTL parser call options
VHDL Library Compiler	describes the VHDL library parser call options to compile a VHDL library into a binary format.
Verilog Parser	describes the structural Verilog (netlist) parser call options
EDIF Parser	describes the EDIF parser call options
LEF Parser	describes the LEF parser call options
DEF Parser	describes the DEF parser call options
Liberty Parser	describes the Liberty parser call options
SDF API	describes the SDF parser call and how to extract timings from a database.

The ZDB Database API

Introduction

This Document describes the API for the database as an extension to the Tcl scripting language. This extension is a native (C-level) implementation to provide the user with the highest possible speed and minimum resource consumption to maintain huge net-list databases. All the API is based on one new Tcl command "**zdb**".

Extensions to this document are: Flat View, Spos, Cone Extraction, Operators (modify), and Prim Functions.

For further information please checkout the API Tutorial (that also includes calls to the GUI API).

The Index is an overview of all API documents.

Index

Overview

A database — later referred to as **$db** — stores one structural circuit design. More than one database can exist simultaneously. The Tcl API introduces a new variable type called OID (for object identification) that is used as a "pointer" to database objects. This document uses **$oid** but also **$mod**, **$inst**, **$pin** etc. to name OID variables.

Here is a list of the most important commands:

Return Value	Command	Description
`$db`	`zdb new|open|check|valid|foreach⋯`	
	`$db save|close ⋯`	
	`$db load ⋯`	Load netlist objects into the database

Return Value	Command	Description
$oid	$db search ⋯	Search netlist object by name
...	$db oper ⋯	Operators to modify the database
$oidList	$db tools ⋯	Tools to query the database
...	$db cone ⋯	Do Cone Extraction
...	zdb cone ⋯	user defined Cone Arcs
...	sdf $db ⋯	Get Data from SDF File

Return Value	Command
$oid	$db oid create|createIcase $oid
$oid	$db oid createFromString ⋯
<string>	$db oid type|root|path|module|oname|pname|cname|tail|print $oid
<string>	$db oid iname0|pname0|iname1|pname1 $oid
<bool>	$db oid isModBased|isTopRoot $oid
$oid	$db oid createModBased|searchTreeBased $oid
$oid	$db oid createNetSeg $pin $pin
$oid	$db oid convertTo $type $oid ?$name?
$oid	$db oid down|up $oid
$oid	$db oid concat $inst $pin
<bool>	$db oid exists $oid
<bool>	$db oid isnull $oid
<string>	$db oid nulltype $oid
<bool>	$db oid isequal $oid1 $oid2
<bool>	$db oid cmp $oid1 $oid2
<bool>	$db oid isparasitic $oid
<bool>	$db oid isparasiticmodinst $oid
<bool>	$db oid insideprim $oid
$oid	$db oid convertFromParasitic $oid
$oid	$db oid convertToParasitic $oid $pTopOid
<integer>	$db oid resetAllOIDs
<bool>	$db isBusMember $pin\|$port|$net
$oid	$db busOf $pin\|$port|$net
<integer>	$db widthOf $pinBus\|$portBus|$netBus
<bool>	$db isTop $mod
<bool>	$db isModule $inst\|$pin|$pinBus\|$mod|$prim
<bool>	$db isEmpty $module ?-checkDanglingNets?

Return Value	Command
$oid	$db moduleOf $inst\|$pin|$pinBus
$oid	$db primitiveOf $inst\|$pin|$pinBus
$oid	$db refCount $prim\|$mod
<bool>	$db hasParentInst $inst\|$net|$netBus
$inst	$db parentInst $inst\|$net|$netBus
<bool>	$db hasParentMod $oid
$mod	$db parentModule $oid
<string>	$db primFuncOf $prim\|$mod|$inst\|$pin
	$db setPrimitive $level
<string>	$db directionOf $pin\|$port|$pinBus\|$portBus
<bool>	$db isConnected $pin\|$port
$oid	$db connectedNet $pin\|$port
<bool>	$db isOneToOneConnection $netBus $pinBus\|$portBus
	$db top set|unset $mod
$oid	$db clone $primitive\|$module ?$rename?
	$db cloneDB ?options?
	$db mergeDB ?options?

Return Value	Command	Description
…	$db attr $oid …	set, get or delete attributes
…	$db flag $oid …	set, get, toggle or delete flags
…	$db hilight $oid …	set, get or delete highlight data
…	$db highlight $oid …	set, get or delete highlight data
	$db deleteZombies	
…	$db spos …	get source file positions
…	$db flat|flatattr|flatflag|flathighlight …	flat view sub-commands
<bool>	$db identicalInterface	
…	$db report sizeOf|instCount|netCount|portCount|objCount …	
<integer>	$db count $type …	count the number of database objects of the given $type
…	$db tdevice …	
	$db symlib …	read/update symlib file

Return Value	Command	Description
	`$db symdef set|get|delete|list …`	
`<string>`	`$db info $binfile`	

Foreach Loops

`$db foreach top|module|primitive|cell|parasitic`		cur	`$body`	examine database root
`$db foreach port|portBus|oPort`	`$mod`	cur	`$body`	examine module/primitive interface
`$db foreach inst|net|netBus`	`$mod`	cur	`$body`	examine module
`$db foreach pin|pinBus|oPin`	`$inst`	cur	`$body`	examine instance
`$db foreach net|member|pin|port`	`$bus`	cur	`$body`	examine buses
`$db foreach pin|pinCon|portCon`	`$net`	cur	`$body`	examine connectivity
`$db attr $oid foreach`		cur	`$body`	examine attributes

Maintain a Database

To create a new database, use the `zdb new` or `zdb open` commands and load netlist data with the `$db load` command or with one of the shipped parser executables (e.g. `rtl2zdb`).

The command

```
zdb new
```

creates a new and empty database and returns a "database handle" (later referred to as **`$db`**), usually something like `zdb01`, `zdb02`, etc.

```
zdb open ?-quick? $binfile
```

Returns a "database handle" for the created database (like the `zdb new` command) but also loads the database from the specified binfile. Opening the binfile in quick mode by specifying the `-quick` option will load the cell definitions and hierarchy information only. The contents for each module needs to be populated separately.

```
$db save $binfile
```

stores the contents of the database `$db` into the specified `$binfile`.

The command

```
zdb valid $db
```

can be used to check whether $db is a valid database command returned from e.g. zdb new.

The command

```
zdb foreach db {
    # do something with $db, e.g. '$db close'
}
```

can be used to loop over all active databases.

The command

```
$db write tcl ?-gzip? ?-compress t? ?-comments 0|1? ?-nospos? ?-noattr?
              ?-noflag? ?-noTsort? ?-flat? ?-sposidx? ?-mangle? $tclfile
```

writes the contents of the database $db as Tcl $db load commands into the specified $tclfile.

- If one of the options -nospos, -noflag or -noattr is given then no -spos, -attr/-pinattr or -flag options will be added to the $db load commands.
- The option -flat adds flat infos.
- The option -gzip may be used to produce gzip compressed output.
- With -comments 0 no comments are written to the output file; with -comments 1 only a file header is written.
- The option -sposidx generates more compact -spos options using indices instead of filenames.
- With -mangle all names get mangled.

The command

```
$db write verilog ?-named? ?-gzip? ?-compress t?
                  ?-comments 0|1|2? ?-objcomments?
                  ?-preservenet?
                  ?-implementFunction?
                  ?-ignoreport $flag? ?-ignorecell $flag? ?ignorercell $flag?
                  ?-ignoreinst $flag? ?-ignoreautogen? $verilogfile
                  ?-noCelldefine?
```

writes the contents of the database $db as Verilog into the specified Verilog file using implicit connectivity.

- With -named named connectivity is used.

- With `-ignoreport` flagged ports are not written to the output file.
- With `-ignorecell` flagged cells are not written to the output file.
- With `-ignorercell` flagged cells and referenced instances are not written to the output file.
- With `-ignoreinst` flagged instances are not written to the output file.
- With `-ignoreautogen` all cells auto generated by a parser are ignored.
- The option `-gzip` may be used to produce gzip compressed output.
- With `-comments 0` no comments are written to the output file. With `-comments 1` only a file header, with `-comments 2` info at each module are written.
- The option `-preservenet` may be used to give local net names priority over instance names if there are name clashes.
- With `-implementFunction` function implementations for selected primitives is added to run simulations.
- If `-objcomments` is specified then the module-based `@comment` attribute is printed before a module, port or instance object.
- If `-noCelldefine` is specified then no `` `celldefine `` and `` `endcelldefine `` is written.

The command

```
$db write spice ?-gzip? ?-compress t?
                ?-comments 0|1? ?-subckt? ?-subcktx? ?-nomodel? ?-noEnd?
                ?-value0net $n? ?-value1net $n? ?-valuexnet $n? ?-valueznet $n?
                $spicefile
```

writes the contents of the database `$db` as Spice into the specified spicefile.

- The option `-gzip` may be used to produce gzip compressed output.
- With `-comments 0` no comments are written to the output file. With `-comments 1` only a file header is written.
- With `-subckt` devices are written as subckts instances and corresponding wrappers are created.
- With `-subcktx` devices are written as subckts instances.
- With `-nomodel` no `.model` statements nor empty sub-circuits are created for models or macro models in the generated netlist.
- With `-noEnd` no `.END` is added at the end of the file.
- The `-value0net`, `-value1net`, `-valuexnet`, and `-valueznet` options can be used to specify names for constant nets which have a value.

The command

```
$db write dspf  ?-gzip? ?-compress t?
                ?-comments 0|1? ?-ignoreempty?
                ?-top $top? ?-design $d? ?-version $v? ?-vendor $v?
                ?-date $dt? ?-program $p?
                ?-divider $div? ?-delimiter $del? ?-busdelimiter $bd?
                ?-gndnet $gnd?
                ?-only $netname?
                ?-noesc
                $dspffile
```

writes the contents of the database `$db` as DSPF into the specified dspffile.

- The option `-gzip` may be used to produce gzip compressed output.
- With `-comments 0` no comments are written to the output file. With `-comments 1` only a file header is written.
- The `-top` option defines the top module.
- The `-design`, `-version`, `-vendor`, `-date` and `-program` overwrite/set the corresponding header information.
- The `-divider`, `-delimiter`, `-busdelimiter` define the divider and delimiter used in the output file.
- The `-gndnet` defines a ground net used in the output file.
- With the `-ignoreempty` option empty parasitics are ignored.
- With the `-only` option only the given net is written. This option can exist multiple times.
- With the `-noesc` option strings are written as is. This may produce incorrect syntax.

The command

```
$db write spef  ?-gzip? ?-compress t?
                ?-comments 0|1? ?-namemap? ?-ignoreempty?
                ?-top $top? ?-design $d? ?-version $v? ?-vendor $v?
                ?-date $dt? ?-program $p? ?-designflow $df?
                ?-divider $div? ?-delimiter $del? ?-busdelimiter $bd?
                ?-timescale $ts? ?-timeunit $tu?
                ?-capscale $cs?  ?-capunit $cu?
                ?-resscale $rs?  ?-resunit $ru?
                ?-indscale $is?  ?-indunit $iu?
                ?-gndnet $gnd? ?-powernet $pwr?
                ?-only $netname?
                ?-noesc
                $speffile
```

writes the contents of the database `$db` as SPEF into the specified speffile.

- The option `-gzip` may be used to produce gzip compressed output.
- With `-comments 0` no comments are written to the output file. With `-comments 1` only a file header

is written.

- The `-namemap` option creates a namemap in the outputfile.
- The `-top` option defines the top module.
- The `-design`, `-version`, `-vendor`, `-date`, `-program`, `-designflow -timescale`, `-timeunit`, `-capscale`, `-capunit`, `-resscale`, `-resunit -indscale` and `-indunit` overwrite/set the corresponding header information.
- The `-divider`, `-delimiter`, `-busdelimiter` define the divider and delimiters used in the output file.
- The `-powernet` and `-gndnet` define power and ground nets used in the output file.
- With the `-ignoreempty` option empty parasitics are ignored.
- With the `-only` option only the given net is written. This option can exist multiple times.
- With the `-noesc` option strings are written as is. This may produce incorrect syntax.

The command

```
$db write vhdl ?-gzip? ?-comments 0|1? ?-implementFunction? $vhdlfile
```

writes the contents of the database `$db` as VHDL into the specified vhdlfile.

- The option `-gzip` may be used to produce gzip compressed output.
- With `-comments 0` no comments are written to the output file. With `-comments 1` a file header is written.
- With `-implementFunction` function implementations for selected primitives is added to run simulations.

The command

```
$db write liberty ?-gzip? ?-allCells? ?-library name? $libertyfile
```

writes the cells of the database `$db` and the corresponding clock information as Liberty into the specified libertyfile.

- The option `-gzip` may be used to produce gzip compressed output.
- With `-allCells` all cells are written to the file.
- The `-library` option defines the library's name of the exported cells.

The command

```
$db close
```

closes the database and frees all associated resources including memory.

The command

```
$db info binfile
```

will print the binfile associated with the database or an empty string if no binfile was used.

The commands

```
zdb check $binfile
zdb validate $binfile
```

check if the `$binfile` is valid for the current architecture and if the `$binfile` is currently not in use by anyone else.

`zdb check` returns an error code if the check is negative. `zdb validate` returns true if the `$binfile` is ok, false otherwise.

```
zdb fileinfo $binfile
```

returns additional information about the given `$binfile`.

```
zdb parserbits $binfile
```

returns a list of used licenses in the given `$binfile`.

The command

```
zdb version
```

returns the internal version of the ZDB including the required license version, the binfile version and the stubs version as well as the build date.

```
zdb sizeof ?$type?
```

returns the amount of bytes that is needed by the in memory structures to store a database object of the given `$type`. If no `$type` is given then a list with the sizes of all types is returned.

Possible values for the `$type` are: `port`, `portBus`, `primitive`, `module`, `pin`, `inst`, `pinRef`, `net`, `netbus`, `spos`, and `sfile`.

Error Conditions

Some of the database commands above and below may return because of an error condition (e.g. file access problems, no disk space, out of memory, etc.). In those cases, an error message is returned that indicates the problem.

Quick Open Mode

A database can be opened in quick mode, then only the cell definitions and hierarchy information is loaded into memory. The contents for each module can be populated separately. The following chapter describes the database populate API.

$db populate

Database populate sub-command.

$db populate active

Returns true if populate is active (-quick mode).

Usage:

```
$db populate active
```

Parameters:

No parameters.

Example:

```
set active [$db populate active]
```

$db populate changed

Check if a database item was populated.

Usage:

```
$db populate changed ?-clear?
```

Parameters:

`-clear (optional)`
: Clear current state.

Example:

```
set changed [$db populate changed]
```

$db populate changedFlat

Check if a flat item was populated.

Usage:

```
$db populate changedFlat ?-clear?
```

Parameters:

`-clear (optional)`
: Clear current state.

Example:

```
set changed [$db populate changedFlat]
```

$db populate changedModule

Check if a module was populated.

Usage:

```
$db populate changedModule ?-clear?
```

Parameters:

`-clear (optional)`
: Clear current state.

Example:

```
set changed [$db populate changedModule]
```

$db populate changedSfile

Check if a sfile item was populated.

Usage:

```
$db populate changedSfile ?-clear?
```

Parameters:

`-clear (optional)`
: Clear current state.

Example:

```
set changed [$db populate changedSfile]
```

$db populate populatedCount

Get the number of populated items.

Usage:

```
$db populate populatedCount
```

Parameters:

No parameters.

Example:

```
set total [$db populate populatedCount]
```

$db populate populatedFlatCount

Get the number of populated flat trees.

Usage:

```
$db populate populatedFlatCount
```

Parameters:

No parameters.

Example:

```
set total [$db populate populatedFlatCount]
```

$db populate populatedModuleCount

Get the number of populated modules.

Usage:

```
$db populate populatedModuleCount
```

Parameters:

No parameters.

Example:

```
set total [$db populate populatedModuleCount]
```

$db populate populatedSfileCount

Get the number of populated sfiles.

Usage:

```
$db populate populatedSfileCount
```

Parameters:

No parameters.

Example:

```
set total [$db populate populatedSfileCount]
```

$db populate totalCount

Get the total number of populatable items.

Usage:

`$db populate totalCount`

Parameters:

No parameters.

Example:

```
set total [$db populate totalCount]
```

$db populate totalFlatCount

Get the total number of populatable flat trees.

Usage:

`$db populate totalFlatCount`

Parameters:

No parameters.

Example:

```
set total [$db populate totalFlatCount]
```

$db populate totalModuleCount

Get the total number of populatable modules.

Usage:

`$db populate totalModuleCount`

Parameters:

No parameters.

Example:

```
set total [$db populate totalModuleCount]
```

$db populate totalSfileCount

Get the total number of populatable sfiles.

Usage:

```
$db populate totalSfileCount
```

Parameters:

No parameters.

Example:

```
set total [$db populate totalSfileCount]
```

Load a Database

As an alternative to the parsers above, the database can also be loaded through the API with `$db load ⋯` commands.

```
$db load primitive $name $function  ?opt?
$db load module    $name ?-top?     ?opt?  ?-primfunc $function? ?-parasitic?
$db load port      $name $direction ?opt?
$db load portBus   $name $direction $width $pname1 $pname2 ... ?-range $from $to?
?opt?
$db load inst      $name $cellref   ?opt? ?pinOpt?
$db load net       $name ?-pin inst pin? ?-port port? ?-value $constantValue?
                         ?-pinspos $iname $pname $fname $start $end? ?opt?
$db load netBus    $name $width $net1 $net2 ... ?opt?
```

The Primitive's `$function` argument (and the Module's `-primfunc $function` option) is defined in the document The Primitive Functions.

The option -parasitic creates a parasitic module instead of a regular one (it can only be instantiated in a special parasitic top module).

The ?opt? is zero or one or more of:

```
-attr  $name=$value
-spos  $fname $start $end
-sposx $fidx $start $end
-flag  $flag
```

For loading instances the additional option -orient can be used to specified the orientation. Possible orientation values are: R0, R90, R180, R270, MY, MYR90, MX, and MXR90.

The ?pinOpt? is zero or one or more of:

```
-pinattr  $pin $name=$value
-pinspos  $pin $fname $start $end
-pinsposx $pin $fidx $start $end
-pinflag  $pin $flag
```

Attributes can be defined for all objects with the -attr option. These options can be repeated to add multiple attributes (alternatively, attributes can be set with the attr add command).

Source positions can be added with the -spos option. They can be repeated multiple times. See the spos API for more information.

Flags can be set with the -flag option. See the $db flag set for more information.

For pins attributes, flags and spos can be added with the -pinattr, -pinflag, and -pinspos options.

Instead of the -spos and the -pinspos options, -sposn and -pinsposn may be used to store the filenames as is. This may be useful, if the default conversion to absolute filenames is not applicable.

As a more compact form to add source positions the -sposx and the -pinsposx options use a numerical index for each referenced file.

The port $direction argument is one of: input, output, inout, unknown, and * (the * means that the port direction is derived from the Primitive's or Module's function). The suffix .neg is supported for some ports.

If a portBus range is specified using the -range option then the portBus member names need to conform to a supported bus member syntax (i.e. NAME[0], NAME(0), NAME<0>, NAME{0}, NAME_0_, NAME_0, and NAME0, where NAME is the bus' name and `0`is the member's index).

The order of the load command is important. A $db load primitive ⋯ command must be followed by calls to $db load port ⋯ and $db load portBus ⋯ to define the primitive's interface. A $db load module ⋯ command must be followed by calls to $db load port ⋯ and $db load portBus ⋯ to define the module's interface and by calls to $db load inst ⋯, $db load net ⋯, and $db load netBus ⋯ to define the module's contents.

Further, **declared before used** is required, that means, 1. a primitive or module must be defined before an instance of it can be created; 2. ports and instances must be defined before a net can refer to them with `-pin` or `-port`.

The `$db load netBus ⋯` command finds existing sub-nets or creates new sub-nets. That means, nets can be defined before or after they are "grouped" by a call to `$db load netBus ⋯`.

The same net can be loaded multiple times, if so, then the connections (defined by `-pin` and `-port`) are accumulated. However, other objects cannot be accumulated, each load must define a new object with a unique name.

The `$cellref` argument of the `$db load inst` command must be either a module or a primitive name. Modules and primitives share the same name space.

To create an unique name, the command

```
$db uniqueName $module $type $name
```

can be used.

Delete Objects from a Database

```
$db deleteZombies
```

The `$db deleteZombies` command deletes all objects with the `zombie` flag (must be `set` before calling `$db deleteZombies`).

The table below displays the objects that accept the `zombie` flag and the detailed delete semantics (ports cannot be deleted, i.e. this command cannot change the module/primitive interface).

Object Type	Effects of `$db deleteZombies`
`inst`	automatically disconnect from nets.
`net`	automatically disconnect from ports and instance pins.
`netBundle`	automatically expand the bus into single-bits (technically, the single-bit nets existed before and are not touched).
`module`	automatically delete all module contents and all instances of that module.
`primitive`	automatically delete all instances of that primitive.

Reload Objects to a Database

```
$db reloadModule   $name
$db load   inst    $name ...
$db load   net     $name ...
$db load   netBus  $name ...
```

The `$db reloadModule ⋯` command "goes to" the given module, so that further `$db load ⋯` commands can add more objects to the specified module. Only `$db load inst ⋯`, `$db load net ⋯` and `$db load netBus ⋯` are supported (this "reload" mode does not support ports, i.e. it cannot change the module/primitive interface).

To reload parasitic modules please use `$db reloadParasitic $name`.

Other commands that modify the database are described in the "Operator API" document.

Set Hierarchy Separator

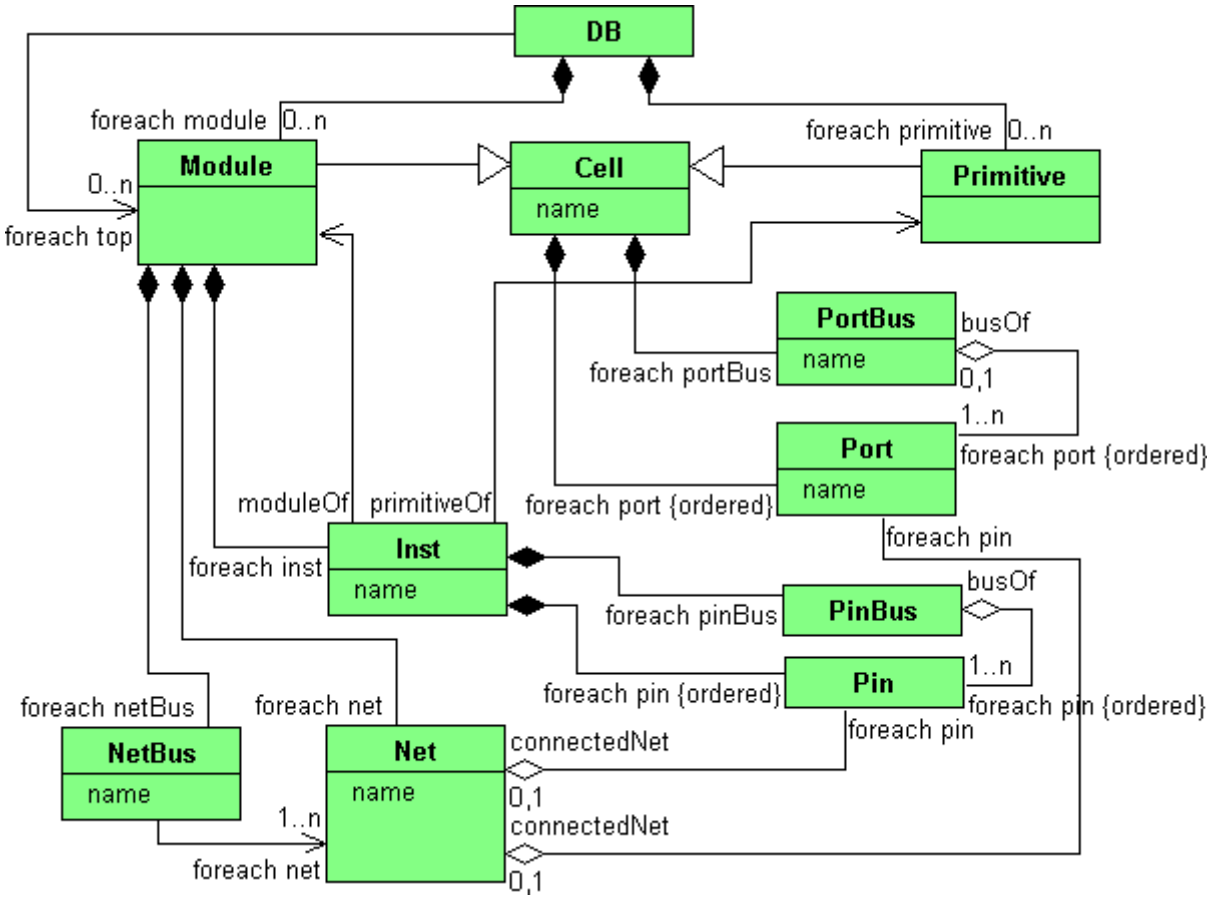
```
$db hiersep $character
```

The `$db hiersep ⋯` command sets the hierarchy separator for the database without any further checks if the specified `$character` is an unused character.

There is a more user friendly (but slower) version of this function in the "Operator API".

Access a Database

The following UML diagram displays the major database objects and their static relations. The labels at the relations refer to the query API functions (basically to the foreach loops below).



The database can be traversed with foreach loops and implicitly searched with the `$db oid create ⋯` command. A new Tcl variable type called OID is defined by this Tcl API to provide fast access to database objects. Each OID variable (object identification) points to a database entry.

Object IDs

The Tcl interpreter is extended by a new Tcl variable type called OID (later referred to as `$oid`). The string representation of these OIDs follow the Tcl-rules for a list, however, internally they are stored as pointers to the database objects for efficiency (Tcl variable's native representation). Here is a description of the OID string representation:

`$otype`	The object type; one of `inst`, `net`, `netBus`, `pin`, `pinBus`, `port`, `portBus`, `netSeg`, `signal`, `module`, and `primitive`.		
`$mname`	The name of the module containing the object (if no path is specified) — or the root of the instance path (if a path is specified) usually the design root.		
`$path1` … `$pathn`	The instance names `$path1` to `$pathn` define the path from the root `$mname` to the module that actually contains the object. The path is optional, i.e. it may be empty.		
`$oname`	The name of the object that is one of instance name, net name, netBus name, etc.		
`$pname`	For objects of type `pin` and `pinBus` only: The name of the pin or pinBus (`$oname` is the instance name).	`$iname0` `$pname0` `$iname1` `$pname1`	For objects of type `netSeg` only: The instance-name and pin-name for two connected pins. If instance-name is an empty string, then the pin-name refers to a port (`$oname` is the net name). A "netSeg" describes a pin-to-pin connection within a net and is only meaningful in some cases (e.g. highlighting in the GUI).

Basically, the database and API provides two ways (or styles) of addressing objects, a **module-based** way with no instance path, and a **tree-based** way with a valid instance path from the design root to the object. The module-based OIDs can be sub-divided into those that refer to the design root (alias top module) and those that refer to an ordinary non-top module — the `$db oid isTopRoot …` command explains the difference.

The following table assumes `GL85` is the design root (a top module), and `REG8` is an ordinary non-top module (the 2nd and 3rd columns refer to the same database objects):

module-based OIDs rooted at top module	module-based OIDs rooted at non-top module	tree-based OIDs rooted at top module

`inst GL85 IR`
`net GL85 CLK`
`netBus GL85 DIN(7:0)`
`pin GL85 IR C`
`pinBus GL85 IR BIN(7:0)`
`port GL85 C`
`portBus GL85 DIN(7:0)`
`netSeg GL85 CLK "" C IR C`
`signal GL85 CLK`
	`inst REG8 R3`
	`net REG8 ST`
	`netBus REG8 L(7:0)`
	`pin REG8 R3 SET`
	`pinBus REG8 R3 D(7:0)`
	`port REG8 C`
	`portBus REG8 Q(7:0)`
	`netSeg REG8 ST "" C R3 SET`
		`inst GL85 IR B8 R3`
		`net GL85 IR B8 ST`
		`netBus GL85 IR B8 L(7:0)`
		`pin GL85 IR B8 R3 SET`
		`pinBus GL85 IR B8 R3 D(7:0)`
		`port GL85 IR B8 C`
		`portBus GL85 IR B8 Q(7:0)`
		`netSeg GL85 IR B8 ST "" C R3 SET`
		`signal GL85 IR B8 ST`

module-based OIDs rooted at top module	module-based OIDs rooted at non-top module	tree-based OIDs rooted at top module
`module GL85 GL85`	`module REG8 REG8` `primitive DFF DFF`	`module GL85 IR B8 REG8` `primitive GL85 IR B8 DFF`

Create Object IDs

Creating an OID from a string representation is a powerful command. It searches the database and figures out if the addressed object really exists. If not, an error is returned.

```
$db oid create      $oid
$db oid createIcase $oid
```

This commands create and return an OID (internal representation) from a string representation. The given `$oid` is expected to be a Tcl list like: `[list $otype $mname $path1 ⋯ $pathn $oname ?$pname?]`.

If, however, the given `$oid` already is an OID, then the OID is simply returned. The `$db oid createIcase ⋯` is equal to `$db oid create ⋯`, except that the internal database searches are performed in "case-insensitive" mode (however, the returned OID is always "case sensitive" — as the ZDB internally is).

```
$db oid createFromString $type $mod $str $hiersep
                         ?-delim $d? ?-escape $e? ?-icase? ?-guess?
                         ?-topInstName $t?
```

This command can be used if the instance path and name of the object is given by one string `$str`. The string is split by the given hierarchy separator `$hiersep`. If no optional delimiter `$d` is given, the hierarchy separator is used to find the last part of the string, used as an object name. Hierarchy separator and delimiter may be protected by preceding them with an optional escape character `$e`. Optional guessing by prepending an `X` to the instance name used during searching may be enabled by `-guess`. The option `-topInstName` can be used to short the path until the first match.

Access Object IDs

Following the Tcl standard each OID is automatically converted back to its string representation whenever needed. That means `lindex` and other Tcl commands can always operate on OIDs, but for efficiency we recommend using these access methods:

```
$db oid type   $oid
$db oid root   $oid
$db oid path   $oid
$db oid module $oid
$db oid oname  $oid
$db oid pname  $oid
$db oid cname  $oid
$db oid tail   $oid
$db oid iname0 $oid
$db oid pname0 $oid
$db oid iname1 $oid
$db oid pname1 $oid
```

The first three commands return object type, the root module name and the instance path starting at the root module.

The `$db oid module $oid` command returns the name of the module at the end of the path (is identical to the root if the path is empty; for if `$oid` is the port or portBus of a primitive, the primitive's name is returned).

The `$db oid oname $oid` returns the object's name, e.g. instance name, net name, etc. However, for pins and pinBuses, the `oname` is the instance name and the `$db oid pname $oid` command returns the pin name. For all object types except pins and pinBuses, the `pname` command returns a empty string.

The `$db oid cname $oid` returns the cell name. This is the module or primitive name of the given inst (or pin or pinBus).

The `$db oid tail $oid` command returns a two or three element list, consisting of the object type plus `oname` plus the optional `pname`.

The `$db oid iname0 $oid` and the `$db oid iname1 $oid` commands return the instance names of `pin1` and `pin2` of a netSeg OID. If the instance name is a empty string then the connected object is a port, not a pin.

The `$db oid pname0 $oid` and the `$db oid pname1 $oid` commands return the pin names of the connected pins/ports of a netSeg `$oid`.

Some examples:

`$oid`	**type**	**root**	**path**	**module**	**oname**	**pname**	**cname**	**tail**
`inst gl85 IR B8 R3`	inst	gl85	IR B8	REG8BITT	R3		DFF	inst R3
`inst REG8BITT R3`	inst	REG8BITT		REG8BITT	R3		DFF	inst R3
`pin gl85 IR B8 R3 SET`	pin	gl85	IR B8	REG8BITT	R3	SET	DFF	pin R3 SET

$oid	type	root	path	module	oname	pname	cname	tail
pin REG8BITT R3 SET	pin	REG8BITT		REG8BITT	R3	SET	DFF	pin R3 SET
pinBus gl85 IR B8 U9 I	pinBus	gl85	IR B8	REG8BITT	U9	I	BUF8	pinBus U9 I
net gl85 IR B8 Q0	net	gl85	IR B8	REG8BITT	Q0			net Q0
port gl85 IR B8 Q0	port	gl85	IR B8	REG8BITT	Q0			port Q0
port REG8BITT Q0	port	REG8BITT		REG8BITT	Q0			port Q0
portBus gl85 IR B8 U9 I	portBus	gl85	IR B8 U9	BUF8	I			portBus I
netSeg gl85 IR C {} C B8 CK	netSeg	gl85	IR	INST_REG	C			netSeg C {} C B8 CK
signal gl85 IR C	signal	gl85	IR	INST_REG	C			signal C
module gl85 IR B8 U9 BUF8	module	gl85	IR B8 U9	BUF8	BUF8			module BUF8
module BUF8 BUF8	module	BUF8		BUF8	BUF8			module BUF8
primitive gl85 IR B8 R3 DFF	primitive	gl85	IR B8 R3	DFF	DFF			primitive DFF

Additional examples for netSeg:

$oid	iname0	pname0	iname1	pname1
netSeg gl85 IR C {} C B8 CK	{}	C	B8	CK
netSeg top o2 and1 o m1 CLK	and1	o	m1	CLK

The signal OID refers to a set of hierarchically interconnected nets and are only meaningful for flat traversals.

Print Object IDs

The command

```
$db oid print $oid
        ?-hiersep $sep? ?-delim $del? ?-joinchar $char?
        ?-notype? ?-noroot? ?-nopath? ?-noname?
        ?-addwidth? ?-addrange?
        ?-typeastitle? ?-tclname? ?-nameAttr?
```

prints the given `$oid` (i.e. converts it to a string representation). The actual content/layout of the string representation can be controlled by the following options:

- `-hiersep $sep`: use the character `$sep` to join the items of the OID's hierarchical path (default: space character),
- `-delim $del`: use the character `$del` to join the pin name when printing a pin OID (default: the `-hiersep` character),
- `-joinchar $char`: use the character `$char` to join all OID components,
- `-notype`: omit the OID's type,
- `-noroot`: don't print the root (top-level) module,
- `-nopath`: omit the OID's path,
- `-noname`: omit the OID's name,
- `-addwidth`: add the width of bus objects to the type,
- `-addrange`: add the range of bus objects to the type,
- `-typeastitle`: print the type as a title,
- `-tclname`: use Tcl conforming object name.
- `--nameAttr`: use the @name attribute as the object name.

Convert Object IDs

The following commands handle the difference between module-based and tree-based OIDs:

```
$db oid isModBased      $oid
$db oid createModBased  $oid
$db oid searchTreeBased $oid  ?$hint?
$db oid isTopRoot       $oid
$db oid createNetSeg    $pin1 $pin2
$db oid convertTo $type $oid ?$name?
```

the first three commands handle the difference between module-based and tree-based OIDs:

- `$db oid isModBased $oid` returns a boolean (true if the `$oid` is a module-based OID, that means, it has no path).
- `$db oid createModBased $oid` creates and returns a module-based OID from the given `$oid`. If the given `$oid` already is module-based, then that `$oid` is just returned.
- `$db oid searchTreeBased $oid` searches the design hierarchy for a tree-based OID that refers to

the same module as the given module-based `$oid` does (if `$oid` already is tree-based, then that `$oid` is just returned). The `$hint` variable is optional — if specified, then it should be a tree-based OID that's used as a starting point for the search algorithm. E.g. `$db oid searchTreeBased {pin AUX U12 A}` might return `{pin TOP B2 C2 U12 A}` (assuming `TOP.B2.C2` is an instance of module `AUX`) — and `$db oid searchTreeBased {module AUX AUX}` would return `{module TOP B2 C2 AUX}`.
As a special case, if a **module** or **primitive** is not instantiated, then `$db oid searchTreeBased ⋯` will return the given OID as is. Ditto for a port/portBus of a not instantiated module or primitive. `$db oid searchTreeBased ⋯` should never return an error.

`$db oid isTopRoot $oid` returns a boolean; it is true if the `$oid`'s root module is a top module, that means, the foreach top iterator will return it and it can be created with `$db load module -top ⋯` (a top module is not instantiated at all). Most tree-based OIDs are rooted at a top module.

`$db oid createNetSeg $pin1 $pin2` creates a `netSeg` OID (net segment) from two pins or ports (`$pin1` and `$pin2` can be pin or port OIDs). They both must connect to the same net — if not, then an error is returned.

`$db oid convertTo $type $oid ?$name?` converts the given OID into the given `$type` if possible (these conversions run on the variable's native representation and are faster than using `lreplace` on the OID's Tcl-list representation).

Some conversions require an extra argument to define the port/pin `$name`.

`$oid`	`$type`	`?$name?`	**Comments**
`pin ⋯` `pinBus ⋯`	`inst`		
`signal ⋯`	`net`		
`netSeg ⋯`	`net`		
`port ⋯` `portBus ⋯`	`module/primitive`		
`pin ⋯`	`port`		identical to `$db oid down $oid`
`pinBus ⋯`	`portBus`		
`inst ⋯`	`module/primitive`		
`port ⋯`	`pin`		identical to `$db oid up $oid`
`portBus ⋯`	`pinBus`		
`module ⋯` `primitive ⋯`	`inst`		
`net ⋯`	`signal`		identical to `$db flat signalOf $oid`
`inst ⋯`	`pin` `pinBus`	`$name`	search for a port/pinBus named `$name` in the given instance
`module ⋯` `primitive ⋯`	`port` `portBus`	`$name`	search for a port/portBus named `$name` in the given module/primitive

Dive Up and Down with Object IDs

The following commands convert an OID in order to dive down and up the hierarchy:

```
$db oid down $oid
$db oid up   $oid
```

The first command dives down the hierarchy tree and returns an OID with a longer path; the second command is the reverse function and returns an OID with a shorter path (the second only works, if the given `$oid` is a tree-based OID).

`$db oid down $oid` accepts an `$oid` of type `inst`, `pin`, and `pinBus` and returns and OID of type `module`/`primitive`, `port`, and `portBus`, respectively.

`$db oid up $oid` accepts an `$oid` of type `module`, `primitive`, `port`, and `portBus` and returns an OID of type `inst`, `pin`, and `pinBus`, respectively.

The `$db oid down ⋯` command is similar to `$db moduleOf ⋯` / `$db primitiveOf ⋯` and the `$db oid up ⋯` command is similar to `$db parentInst ⋯` / `$db parentModule ⋯`.

Concat a Pin OID to a Module Inst

The following command converts a hierarchical instance OID to a tree-based pin oid, by appending a module-based pin OID or a relative path to a pin:

```
$db oid concat $inst $pin
$db oid concat $inst -pinpath $path
```

The works only if the `$pin` OID is relative to the given `$inst` OID.

The `-pinpath` option can be used, if `$path` is a list containing the instance names and the pin name which should be appended to `$inst`.

Check for Existence of OIDs

```
$db oid exists $oid
```

returns true if the given `$oid` exists in `$db`. The given OID may be created from an other database.

Null OIDs

```
$db oid isnull $oid
```

returns true if the given `$oid` is a "null OID". Those "null OIDs" can show up in the `$db foreach member ⋯` loop to identify "gaps" in a netBus definition; or as a return value from the `$db search ⋯` commands.

Type of Null OIDs

```
$db oid nulltype $oid
```

returns the type for some special "null OIDs" or an empty string. They are used to identify e.g. source pos info for comments etc.

Compare OIDs

```
$db oid isequal $oid1 $oid2
$db oid cmp $oid1 $oid2
```

The first command returns true if both given OIDs are identical, otherwise it returns false.

The second command returns `-1`, `0`, `1` for sorting the given OIDs (suitable for sorting a list of OIDs using Tcl's `lsort` command with the `-command ⋯` option).

These functions can be a speedup over "stringifying" them and comparing the strings.

Parasitic OIDs

```
$db oid isparasitic $oid
$db oid isparasiticmodinst $oid
$db oid convertFromParasitic $oid
$db oid convertToParasitic $oid $pTopOid
```

The first command returns true if the given OID is a parasitic module, or the referred object is contained in a parasitic module.

The second returns true if the given OID is a instance of a parasitic module.

The third converts the given OID to a "normal" OID.

The fourth converts a "normal" OID to a parasitic OID relative to the given parasitic top module `$pTopOid` or, if `$oid` is a signal/net OID `$pTopOId` can be omitted and a parasitic module OID is returned.

Reset all OIDs

```
$db oid resetAllOIDs
```

resets the string representation of all Tcl variables referring to OIDs. The next access to the string representation of the variable will re-create the representation. The command returns the number of reset OIDs.

Additional OID Commands

$db oid

Oid sub-command.

$db oid cleanup_mem

No Documentation yet.

Usage:

`$db oid cleanup_mem`

Parameters:

No parameters.

Example:

```
    # No example
```

$db oid cmp

Returns `-1`, `0`, `1` for sorting the given OIDs (suitable for sorting a list of OIDs using Tcl's `lsort` command with the `-command …` option).

Usage:

`$db oid cmp oid1 oid2`

Parameters:

`oid1`
: Input OID.

`oid2`
: Input OID.

Example:

```
$db oid cmp $oid1 $oid2
```

$db oid cname

Get the cell name. That is the module or primitive name of the given inst (or pin or pinBus).

Usage:

```
$db oid cname oid
```

Parameters:

`oid`
: Input OID.

Example:

```
$db oid cname $oid
```

$db oid concat

No Documentation yet.

Usage:

```
$db oid concat
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db oid convertable

No Documentation yet.

Usage:

```
$db oid convertable
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db oid convertFromParasitic

Converts the given OID to a "normal" OID.

Usage:

```
$db oid convertFromParasitic oid
```

Parameters:

`oid`
: Input OID.

Example:

```
$db oid convertFromParasitic $oid
```

$db oid convertPin

No Documentation yet.

Usage:

```
$db oid convertPin oid pinNo
```

Parameters:

`oid`

Input OID.

`pinNo`

Number of the pin.

Example:

```
$db oid convertPin $oid 0
```

$db oid convertTo

No Documentation yet.

Usage:

`$db oid convertTo`

Parameters:

No parameters.

Example:

```
    # No example
```

$db oid convertToParasitic

No Documentation yet.

Usage:

`$db oid convertToParasitic`

Parameters:

No parameters.

Example:

```
    # No example
```

$db oid convertToPin

No Documentation yet.

Usage:

```
$db oid convertToPin
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db oid convertToPort

No Documentation yet.

Usage:

```
$db oid convertToPort oid portNo
```

Parameters:

`oid`
: Input OID.

`portNo`
: Number of the port.

Example:

```
$db oid convertToPort $oid 0
```

$db oid create

No Documentation yet.

Usage:

```
$db oid create oidObj
```

Parameters:

`oidObj`

String or Oid String representation.

Example:

```
$db oid create {module top top}
$db oid create null
```

$db oid createFromString

This command can be used if the instance path and name of the object is given by one string `$str`. The string is split by the given hierarchy separator `$hiersep`.

Usage:

```
$db oid createFromString ?-autoPopulate? ?-delim delim? ?-escape escape? ?-escapeVerilog? ?-guess? ?-icase? ?-topInstName topInstName? type topModule str hiersep
```

Parameters:

`-autoPopulate (optional)`

Auto populate DB.

`-delim delim (optional default is NULL)`

If no optional delimiter is given, the hierarchy separator is used to find the last part of the string, used as an object name.

`-escape escape (optional default is NULL)`

Hierarchy separator and delimiter may be protected by preceding them with an optional escape character.

`-escapeVerilog (optional)`

Escape the identifiers using the Verilog standard.

`-guess (optional)`

Optional guessing by prepending an `X` to the Instance name used during searching may be enabled by `-guess`.

`-icase (optional)`

Ignore case.

`-topInstName topInstName (optional default is NULL)`

Can be used to short the path until the first match.

`hiersep`

Hierarchy separator.

`str`

Name to the object.

`topModule`

The oid of the containing top module. If {null} OID is given, top module is extracted from the given path string.

`type`

Type of oid.

Example:

```
$db oid createFromString inst {module top top} i1 .
```

$db oid createFromStringList

No Documentation yet.

Usage:

```
$db oid createFromStringList ?-autoPopulate? ?-guessName? ?-guessPath? ?-icase? type pathList name
```

Parameters:

`-autoPopulate (optional)`

Auto populate DB.

`-guessName (optional)`

Use extended rules to match the name.

`-guessPath (optional)`

Use extended rules to match the path.

`-icase (optional)`

Ignore case.

`name`

Name of the object.

`pathList`

List of the names to the object.

`type`

Type of oid.

Example:

```
$db oid createFromStringList inst {top} i1
```

$db oid createIcase

No Documentation yet.

Usage:

`$db oid createIcase oidObj`

Parameters:

`oidObj`

String or Oid String representation.

Example:

```
$db oid createIcase {module TOP TOP}
$db oid createIcase NULL
```

$db oid createModBased

No Documentation yet.

Usage:

```
$db oid createModBased oid
```

Parameters:

`oid`
: No Documentation yet.

Example:

```

```

$db oid createNetSeg

No Documentation yet.

Usage:

```
$db oid createNetSeg
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db oid deleteSchematicCache

Delete the database schematic cache.

Usage:

```
$db oid deleteSchematicCache module
```

Parameters:

`module`

Delete the database schematic cache for this module.

Example:

```
$db oid deleteSchematicCache $module
```

$db oid down

No Documentation yet.

Usage:

```
$db oid down oid
```

Parameters:

`oid`

No Documentation yet.

Example:

```

```

$db oid exists

No Documentation yet.

Usage:

```
$db oid exists
```

Parameters:

No parameters.

Example:

```
# No example
```

$db oid getSchematicCache

Get the database schematic cache.

Usage:

```
$db oid getSchematicCache module
```

Parameters:

`module`
: Get the database schematic cache for this module.

Example:

```
set layout [$db oid getSchematicCache $module]
```

$db oid iname0

No Documentation yet.

Usage:

```
$db oid iname0 oid
```

Parameters:

`oid`
: No Documentation yet.

Example:

```

```

$db oid iname1

No Documentation yet.

Usage:

```
$db oid iname1 oid
```

Parameters:

`oid`
: No Documentation yet.

Example:

```
```

$db oid insideprim

No Documentation yet.

Usage:

```
$db oid insideprim oid
```

Parameters:

`oid`
: No Documentation yet.

Example:

```
```

$db oid isequal

Returns true if both given OIDs are identical, otherwise it returns false.

Usage:

```
$db oid isequal oid1 oid2
```

Parameters:

`oid1`

Input OID.

`oid2`

Input OID.

Example:

```
$db oid isequal $oid1 $oid2
```

$db oid isModBased

No Documentation yet.

Usage:

```
$db oid isModBased oid
```

Parameters:

`oid`

No Documentation yet.

Example:

```

```

$db oid isnull

No Documentation yet.

Usage:

```
$db oid isnull oid
```

Parameters:

`oid`

No Documentation yet.

Example:

```
```

$db oid isparasitic

No Documentation yet.

Usage:

```
$db oid isparasitic oid
```

Parameters:

`oid`

: No Documentation yet.

Example:

```
```

$db oid isparasiticmodinst

No Documentation yet.

Usage:

```
$db oid isparasiticmodinst oid
```

Parameters:

`oid`

: No Documentation yet.

Example:

```
```

$db oid isTopRoot

Returns a boolean; it is true if the `$oid`'s root module is a top module.

Usage:

```
$db oid isTopRoot oid
```

Parameters:

`oid`
: Input OID.

Example:

```
$db oid isTopRoot $oid
```

$db oid isvirtual

No Documentation yet.

Usage:

```
$db oid isvirtual oid
```

Parameters:

`oid`
: No Documentation yet.

Example:

```

```

$db oid module

No Documentation yet.

Usage:

```
$db oid module oid
```

Parameters:

`oid`

No Documentation yet.

Example:

```
```

$db oid nulltype

No Documentation yet.

Usage:

```
$db oid nulltype oid
```

Parameters:

`oid`

No Documentation yet.

Example:

```
```

$db oid oname

Get the object's name, e.g. instance name, net name, etc.

Usage:

```
$db oid oname oid
```

Parameters:

`oid`

Input OID.

Example:

```
$db oid oname $oid
```

$db oid path

No Documentation yet.

Usage:

```
$db oid path oid
```

Parameters:

`oid`

: No Documentation yet.

Example:

```

```

$db oid pname

Get the pin name.

Usage:

```
$db oid pname oid
```

Parameters:

`oid`

: Input OID.

Example:

```
$db oid pname $oid
```

$db oid pname0

No Documentation yet.

Usage:

```
$db oid pname0 oid
```

Parameters:

`oid`
: No Documentation yet.

Example:

```
```

$db oid pname1

No Documentation yet.

Usage:

```
$db oid pname1 oid
```

Parameters:

`oid`
: No Documentation yet.

Example:

```
```

$db oid print

Converts the given oid to a string representation.

Usage:

```
$db oid print ?-addrange? ?-addwidth? ?-delim delim? ?-escapeVerilog? ?-hiersep hiersep? ?-joinchar joinchar? ?-nameAttr? ?-noname? ?-nopath? ?-noroot? ?-notype? ?-tclname? ?-typeastitle? oid
```

Parameters:

`-addrange` `(optional)`

Add the range of a bus object.

`-addwidth` `(optional)`

Add the width of a bus object.

`-delim` **`delim`** `(optional default is NULL)`

Use the given character to separate the pin name.

`-escapeVerilog` `(optional)`

Escape the identifiers using the Verilog standard.

`-hiersep` **`hiersep`** `(optional default is NULL)`

Use the given hierarchy separator to print the OID.

`-joinchar` **`joinchar`** `(optional default is NULL)`

Join all OID elements using this char.

`-nameAttr` `(optional)`

Use the @name attribute for the object name.

`-noname` `(optional)`

Add the object name.

`-nopath` `(optional)`

Add the path to the object.

`-noroot` `(optional)`

Do not print the path of the OID.

`-notype` `(optional)`

Do not print the object type.

`-tclname` `(optional)`

Add Tcl conform object names.

`-typeastitle` `(optional)`

Print the type as title.

`oid`

The oid to print.

Example:

```
$db oid print {inst top i1}
```

$db oid resetAllOIDs

No Documentation yet.

Usage:

```
$db oid resetAllOIDs
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db oid root

No Documentation yet.

Usage:

```
$db oid root oid
```

Parameters:

`oid`
: No Documentation yet.

Example:

```

```

$db oid searchTreeBased

No Documentation yet.

Usage:

```
$db oid searchTreeBased
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db oid setSchematicCache

Set the database schematic cache.

Usage:

```
$db oid setSchematicCache module schematicLayout
```

Parameters:

`module`
: Set the database schematic cache for this module.

`schematicLayout`
: The schematic layout string.

Example:

```
$db oid setSchematicCache $module "schematic layout"
```

$db oid statistics

No Documentation yet.

Usage:

```
$db oid statistics
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db oid tail

No Documentation yet.

Usage:

```
$db oid tail oid
```

Parameters:

`oid`
: No Documentation yet.

Example:

```

```

$db oid type

Get the object type.

Usage:

```
$db oid type oid
```

Parameters:

`oid`
: Input OID.

Example:

```
$db oid type $oid
```

$db oid up

No Documentation yet.

Usage:

```
$db oid up oid
```

Parameters:

`oid`

No Documentation yet.

Example:

$db oid validate

No Documentation yet.

Usage:

```
$db oid validate oid
```

Parameters:

`oid`

No Documentation yet.

Example:

$db oid virtualtype

No Documentation yet.

Usage:

```
$db oid virtualtype oid
```

Parameters:

`oid`

No Documentation yet.

Example:

```
```

Additional ZDB Commands

zdb check

Check if the `$binfile` is valid for the current architecture. `zdb check` returns an error code if the check is negative.

Usage:

```
zdb check binfile
```

Parameters:

`binfile`

Path to a binfile.

Example:

```
zdb check "design.zdb"
```

zdb cone

No Documentation yet.

Usage:

```
zdb cone
```

Parameters:

No parameters.

Example:

```
    # No example
```

zdb fileinfo

No Documentation yet.

Usage:

```
zdb fileinfo
```

Parameters:

No parameters.

Example:

```
    # No example
```

zdb foreach

Can be used to loop over all active databases.

Usage:

```
zdb foreach var body
```

Parameters:

`body`
: The executed code for each database.

`var`
: The loop variable is set to the current database.

Example:

```
zdb foreach db {
    # Do something with $db
}
```

zdb keyword

No Documentation yet.

Usage:

`zdb keyword`

Parameters:

No parameters.

Example:

```
    # No example
```

zdb new

Create a new zdb object.

Usage:

`zdb new ?-malloc? ?binfile?`

Parameters:

`-malloc` `(optional)`
: Create a malloc based database.

`binfile` `(optional)`
: Path to a binfile.

Example:

```
set db [zdb new]
```

zdb open

Returns a "database handle" for the created database (like the `zdb new` command) but also loads the database from the specified binfile. The binfile is opened in read-only mode.

Usage:

```
zdb open ?-malloc? ?-populate? ?-quick? ?-writable? binfile
```

Parameters:

`-malloc (optional)`

Create a malloc based database.

`-populate (optional)`

The `-populate` option can be used to map the whole database into memory. This may slowdown the open command but speedup processing later. It has only an effect on Linux.

`-quick (optional)`

Opening the binfile in quick mode will load the cell definitions and hierarchy information only. The contents for each module needs to be populated separately.

`-writable (optional)`

The binfile is opened writable and is updated whenever the database changes and (after a `$db close`) it is usable for subsequent `zdb open` calls.

`binfile`

Path to a binfile.

Example:

```
set db [zdb open "design.zdb"]
```

zdb parserbits

No Documentation yet.

Usage:

```
zdb parserbits
```

Parameters:

No parameters.

Example:

```
# No example
```

zdb primFuncList

No Documentation yet.

Usage:

`zdb primFuncList`

Parameters:

No parameters.

Example:

```
# No example
```

zdb primPrefix2Func

No Documentation yet.

Usage:

`zdb primPrefix2Func`

Parameters:

No parameters.

Example:

```
# No example
```

zdb profile

No Documentation yet.

Usage:

`zdb profile`

Parameters:

No parameters.

Example:

```
    # No example
```

zdb sizeof

No Documentation yet.

Usage:

`zdb sizeof`

Parameters:

No parameters.

Example:

```
    # No example
```

zdb source

No Documentation yet.

Usage:

`zdb source`

Parameters:

No parameters.

Example:

```
    # No example
```

zdb synbool

No Documentation yet.

Usage:

`zdb synbool`

Parameters:

No parameters.

Example:

```
    # No example
```

zdb valid

Can be used to check whether `$db` is a valid database command returned from e.g. `zdb new`.

Usage:

`zdb valid database`

Parameters:

`database`
: Name of a database.

Example:

```
zdb valid $db
```

zdb validate

Returns true if the `$binfile` is ok, false otherwise.

Usage:

```
zdb validate binfile
```

Parameters:

`binfile`
: Path to a binfile.

Example:

```
zdb validate "design.zdb"
```

zdb version

Prints the database version number.

Usage:

```
zdb version ?what?
```

Parameters:

`what (optional)`
: What

Example:

```
zdb version
```

$db annotate

No Documentation yet.

Usage:

```
$db annotate
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db assertionModel

Create a model of the parasitic database.

Usage:

```
$db assertionModel
```

Parameters:

No parameters.

Example:

```
set assertionModel [$db assertionModel]
```

$db attr

No Documentation yet.

Usage:

```
$db attr
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db blocklevel

Create a new blocklevel object.

Usage:

`$db blocklevel`

Parameters:

No parameters.

Example:

```
set targetDB [zdb new]
set blocklevel [$db blocklevel $targetDB]
```

$db busOf

get bus of given member pin/port/net.

Usage:

`$db busOf pinPortOrNet`

Parameters:

`pinPortOrNet`
: pin, port or net OID.

Example:

```
set oid [$db busOf $pin]
```

$db calc

No Documentation yet.

Usage:

```
$db calc
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db cdc

No Documentation yet.

Usage:

```
$db cdc
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db checkFunction

No Documentation yet.

Usage:

```
$db checkFunction
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db clearSchematicCache

Clear the database schematic cache for all modules.

Usage:

`$db clearSchematicCache`

Parameters:

Example:

```
$db clearSchematicCache
```

$db clone

No Documentation yet.

Usage:

`$db clone`

Parameters:

No parameters.

Example:

```
    # No example
```

$db cloneDB

No Documentation yet.

Usage:

`$db cloneDB`

Parameters:

No parameters.

Example:

```
    # No example
```

$db close

The `$db close` command frees the used memory and deletes the `$db` Tcl command.

Usage:

`$db close`

Parameters:

No parameters.

Example:

```
$db close
```

$db cone

No Documentation yet.

Usage:

`$db cone`

Parameters:

No parameters.

Example:

```
    # No example
```

$db coneToPG

No Documentation yet.

Usage:

```
$db coneToPG
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db connectedNet

get connected net of given pin or port.

Usage:

```
$db connectedNet pinOrPort
```

Parameters:

`pinOrPort`
: pin or port OID.

Example:

```
set oid [$db connectedNet $pin]
```

$db count

No Documentation yet.

Usage:

```
$db count
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db deleteAllButFlat

No Documentation yet.

Usage:

```
$db deleteAllButFlat
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db deleteZombies

No Documentation yet.

Usage:

```
$db deleteZombies
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db directionOf

Get the direction of a pin or port.

Usage:

```
$db directionOf pin
```

Parameters:

`pin`
: The OID of a parasitic module.

Example:

```
$db directionOf $pin
```

$db down

get down of given inst/pin.

Usage:

```
$db down instPinOrPinBus
```

Parameters:

`instPinOrPinBus`
: inst, pin or PinBus.

Example:

```
set oid [$db down $inst]
```

$db dump

Dump the database contents to a file.

Usage:

```
$db dump ?-cell cell? ?-compress compress? ?flags? filename
```

Parameters:

`-cell cell (optional default is NULL)`
: Dump flags.

`-compress compress (optional default is ce_IOTypeStd)`
: Compression type.

`filename`
: Name of the dump output file.

`flags (optional)`
: Dump flags

Example:

```
$db dump "database.dmp"
```

$db efile

No Documentation yet.

Usage:

```
$db efile
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db extract

No Documentation yet.

Usage:

```
$db extract
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db find

No Documentation yet.

Usage:

```
$db find
```

Parameters:

No parameters.

Example:

```
# No example
```

$db flag

No Documentation yet.

Usage:

```
$db flag
```

Parameters:

No parameters.

Example:

```
# No example
```

$db flat

No Documentation yet.

Usage:

```
$db flat
```

Parameters:

No parameters.

Example:

```
# No example
```

$db flatattr

No Documentation yet.

Usage:

```
$db flatattr
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db flatflag

No Documentation yet.

Usage:

```
$db flatflag
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db flathilight

No Documentation yet.

Usage:

```
$db flathilight
```

Parameters:

No parameters.

Example:

```
# No example
```

$db flatid

No Documentation yet.

Usage:

```
$db flatid
```

Parameters:

No parameters.

Example:

```
# No example
```

$db flatoomr

No Documentation yet.

Usage:

```
$db flatoomr
```

Parameters:

No parameters.

Example:

```
# No example
```

$db foreach

No Documentation yet.

Usage:

```
$db foreach
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db get_all_bits

No Documentation yet.

Usage:

```
$db get_all_bits
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db get_attribute

No Documentation yet.

Usage:

```
$db get_attribute
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db get_cell

No Documentation yet.

Usage:

```
$db get_cell
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db get_driver

No Documentation yet.

Usage:

```
$db get_driver
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db get_inst

No Documentation yet.

Usage:

```
$db get_inst
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db get_instances_of_cell

No Documentation yet.

Usage:

```
$db get_instances_of_cell
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db get_load

No Documentation yet.

Usage:

```
$db get_load
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db get_net

No Documentation yet.

Usage:

```
$db get_net
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db get_para_port_name

No Documentation yet.

Usage:

```
$db get_para_port_name
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db get_pins

No Documentation yet.

Usage:

```
$db get_pins
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db get_ports

No Documentation yet.

Usage:

```
$db get_ports
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db get_top_design

No Documentation yet.

Usage:

```
$db get_top_design
```

Parameters:

No parameters.

Example:

```
# No example
```

$db getFuncPort

No Documentation yet.

Usage:

`$db getFuncPort`

Parameters:

No parameters.

Example:

```
# No example
```

$db getMember

No Documentation yet.

Usage:

`$db getMember`

Parameters:

No parameters.

Example:

```
# No example
```

$db getOriginalName

No Documentation yet.

Usage:

```
$db getOriginalName
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db guessedTopModules

No Documentation yet.

Usage:

```
$db guessedTopModules
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db hasParentInst

No Documentation yet.

Usage:

```
$db hasParentInst
```

Parameters:

No parameters.

Example:

```
# No example
```

$db hasParentMod

No Documentation yet.

Usage:

`$db hasParentMod`

Parameters:

No parameters.

Example:

```
# No example
```

$db hasRange

No Documentation yet.

Usage:

`$db hasRange`

Parameters:

No parameters.

Example:

```
# No example
```

$db hiersep

No Documentation yet.

Usage:

```
$db hiersep
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db highlight

No Documentation yet.

Usage:

```
$db highlight
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db hilight

No Documentation yet.

Usage:

```
$db hilight
```

Parameters:

No parameters.

Example:

```
# No example
```

$db htree

No Documentation yet.

Usage:

`$db htree`

Parameters:

No parameters.

Example:

```
# No example
```

$db identicalInterface

No Documentation yet.

Usage:

`$db identicalInterface`

Parameters:

No parameters.

Example:

```
# No example
```

$db ignore

No Documentation yet.

Usage:

```
$db ignore
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db info

No Documentation yet.

Usage:

```
$db info
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db infobox

No Documentation yet.

Usage:

```
$db infobox
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db isBusMember

No Documentation yet.

Usage:

```
$db isBusMember
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db isConnected

No Documentation yet.

Usage:

```
$db isConnected
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db isEmpty

No Documentation yet.

Usage:

`$db isEmpty`

Parameters:

No parameters.

Example:

```
    # No example
```

$db isModule

No Documentation yet.

Usage:

`$db isModule`

Parameters:

No parameters.

Example:

```
    # No example
```

$db isOneToOneConnection

No Documentation yet.

Usage:

`$db isOneToOneConnection`

Parameters:

No parameters.

Example:

```
    # No example
```

$db isOperator

No Documentation yet.

Usage:

```
$db isOperator
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db isPgNet

No Documentation yet.

Usage:

```
$db isPgNet
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db istmptop

No Documentation yet.

Usage:

```
$db istmptop
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db isTop

No Documentation yet.

Usage:

```
$db isTop
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db load

No Documentation yet.

Usage:

```
$db load
```

Parameters:

No parameters.

Example:

```
# No example
```

$db memusage

No Documentation yet.

Usage:

```
$db memusage
```

Parameters:

No parameters.

Example:

```
# No example
```

$db mergeDB

No Documentation yet.

Usage:

```
$db mergeDB
```

Parameters:

No parameters.

Example:

```
# No example
```

$db moduleListModel

Create a new module list object.

Usage:

`$db moduleListModel`

Parameters:

No parameters.

Example:

```
set moduleListModel [$db moduleListModel]
```

$db moduleOf

get module of given inst/pin/pinBus.

Usage:

`$db moduleOf instPinOrPinBus`

Parameters:

`instPinOrPinBus`
: inst, pin or PinBus.

Example:

```
set oid [$db moduleOf $inst]
```

$db oper

No Documentation yet.

Usage:

`$db oper`

Parameters:

No parameters.

Example:

```
# No example
```

$db orient

No Documentation yet.

Usage:

```
$db orient
```

Parameters:

No parameters.

Example:

```
# No example
```

$db parasiticNetModel

Create a model of the parasitic database.

Usage:

```
$db parasiticNetModel
```

Parameters:

No parameters.

Example:

```
set parasiticNetModel [$db parasiticNetModel]
```

$db parasiticSchemModel

Create a new Parasitic view object.

Usage:

```
$db parasiticSchemModel
```

Parameters:

No parameters.

Example:

```
set parasiticSchemModel [$db parasiticSchemModel]
```

$db parentInst

get parentInst of given inst|net|netBus|netSeg|signal.

Usage:

```
$db parentInst instNetNetBusNetSegSignal
```

Parameters:

`instNetNetBusNetSegSignal`
: inst, net, netBus, netSeg or signal.

Example:

```
set oid [$db parentInst $inst]
```

$db parentModule

get parentModule of given oid.

Usage:

```
$db parentModule oid
```

Parameters:

`oid`
: oid.

Example:

```
set oid [$db parentModule $inst]
```

$db populateCount

Populate sub-command to get current count of populated data.

Usage:

`$db populateCount`

Parameters:

No parameters.

Example:

```
set count [$db populateCount]
```

$db populateFlat

Populate sub-command to incrementally fill the database from a binfile.

Usage:

`$db populateFlat ?moduleName?`

Parameters:

`moduleName (optional)`
: Populate the database with the flat data of the given module.

Example:

```
$db populateFlat "TOP"
```

$db populateModule

Populate sub-command to incrementally fill the database from a binfile.

Usage:

```
$db populateModule ?-noflat? ?-parasitic? ?moduleName?
```

Parameters:

`-noflat` `(optional)`
: Do not automatically populate flat data.

`-parasitic` `(optional)`
: Treat the given module name as a parasitic module.

`moduleName` `(optional)`
: Populate the database with the contents of the given module.

Example:

```
$db populateModule "TOP"
```

$db populateSourceFileInfo

Populate sub-command to incrementally fill the source file information in the database for the source file with the given index.

Usage:

```
$db populateSourceFileInfo ?sfileIdx?
```

Parameters:

`sfileIdx` `(optional)`
: Populate the database with the line file information of the sfile index.

Example:

```
$db populateSourceFileInfo 0
```

$db primFuncOf

No Documentation yet.

Usage:

`$db primFuncOf`

Parameters:

No parameters.

Example:

```
    # No example
```

$db primitiveFuncOf

No Documentation yet.

Usage:

`$db primitiveFuncOf`

Parameters:

No parameters.

Example:

```
    # No example
```

$db primitiveOf

get primitive of given inst/pin/pinBus.

Usage:

```
$db primitiveOf instPinOrPinBus
```

Parameters:

`instPinOrPinBus`
: inst, pin or PinBus.

Example:

```
set oid [$db primitiveOf $inst]
```

$db print_oid_as_slash_path

No Documentation yet.

Usage:

```
$db print_oid_as_slash_path
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db print_oid_as_spef

No Documentation yet.

Usage:

```
$db print_oid_as_spef
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db rangeLsb

No Documentation yet.

Usage:

`$db rangeLsb`

Parameters:

No parameters.

Example:

```
    # No example
```

$db rangeMsb

No Documentation yet.

Usage:

`$db rangeMsb`

Parameters:

No parameters.

Example:

```
    # No example
```

$db refCount

No Documentation yet.

Usage:

```
$db refCount
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db reloadModule

No Documentation yet.

Usage:

```
$db reloadModule
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db reloadParasitic

No Documentation yet.

Usage:

```
$db reloadParasitic
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db report

No Documentation yet.

Usage:

```
$db report
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db save

Save the database as a binfile.

Usage:

```
$db save ?-legacy? binfile
```

Parameters:

`-legacy (optional)`
: Use legacy file format.

`binfile`
: The name of the created binfile.

Example:

```
$db save "design.zdb"
```

$db schematic

Create a new schematic object.

Usage:

```
$db schematic
```

Parameters:

No parameters.

Example:

```
set schem [$db schematic]
```

$db schematicModel

Create a new schematicModel object.

Usage:

```
$db schematicModel
```

Parameters:

No parameters.

Example:

```
set schem [$db schematicModel]
```

$db search

No Documentation yet.

Usage:

`$db search`

Parameters:

No parameters.

Example:

```
    # No example
```

$db setPrimitive

No Documentation yet.

Usage:

`$db setPrimitive`

Parameters:

No parameters.

Example:

```
    # No example
```

$db setXY

No Documentation yet.

Usage:

`$db setXY`

Parameters:

No parameters.

Example:

```
    # No example
```

$db spos

No Documentation yet.

Usage:

`$db spos`

Parameters:

No parameters.

Example:

```
    # No example
```

$db sposFilesModel

Create a SPOS file model.

Usage:

`$db sposFilesModel`

Parameters:

No parameters.

Example:

```
set model [$db sposFilesModel]
```

$db symdef

No Documentation yet.

Usage:

`$db symdef`

Parameters:

No parameters.

Example:

```
    # No example
```

$db symlib

No Documentation yet.

Usage:

`$db symlib`

Parameters:

No parameters.

Example:

```
    # No example
```

$db tableAlloc

No Documentation yet.

Usage:

`$db tableAlloc`

Parameters:

No parameters.

Example:

```
# No example
```

$db tdevice

No Documentation yet.

Usage:

`$db tdevice`

Parameters:

No parameters.

Example:

```
# No example
```

$db tools

No Documentation yet.

Usage:

`$db tools`

Parameters:

No parameters.

Example:

```
# No example
```

$db top

No Documentation yet.

Usage:

```
$db top
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db treeModel

Create a new hierarchy tree model object.

Usage:

```
$db treeModel
```

Parameters:

No parameters.

Example:

```
set treeModel [$db treeModel]
```

$db uniqueName

No Documentation yet.

Usage:

```
$db uniqueName
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db unusedTopCandidates

No Documentation yet.

Usage:

`$db unusedTopCandidates`

Parameters:

No parameters.

Example:

```
    # No example
```

$db up

get up of given module/primitive/port/portBus.

Usage:

`$db up modPrimPortOrPBus`

Parameters:

`modPrimPortOrPBus`
: module, primitive, port or portBus.

Example:

```
set oid [$db up $mod]
```

$db utils

No Documentation yet.

Usage:

```
$db utils
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db validate

No Documentation yet.

Usage:

```
$db validate
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db value

No Documentation yet.

Usage:

```
$db value
```

Parameters:

No parameters.

Example:

```
    # No example
```

$db widthOf

No Documentation yet.

Usage:

```
$db widthOf
```

Parameters:

No parameters.

Example:

```
    # No example
```

zdb formatvalue

zdb formatvalue sub-command.

zdb formatvalue changeverilogradix

Change the radix of a Verilog value. Returns the original value in case of errors.

Usage:

```
zdb formatvalue changeverilogradix radix inputValue
```

Parameters:

`inputValue`
: Input value in the specified format to be formatted.

`radix`

Specify the output format: -bin|-oct|-dec|-hex.

Example:

```
zdb formatvalue changeverilogradix -hex 4'b1010
```

zdb formatvalue frombinary

Convert a given binary value into an octal, decimal or hexadecimal value.

Usage:

`zdb formatvalue frombinary ?-prefix? ?-verilogPrefix? ?-width widthVal? radix inputValue`

Parameters:

`-prefix (optional)`

Adds 'O' for octal and '0x' for hexadecimal values at the beginning of the output value. prefix and verilogPrefix are mutually exclusive.

`-verilogPrefix (optional)`

Add [0-9]'[bodh] prefix in Verilog style to the value. prefix and verilogPrefix are mutually exclusive.

`-width widthVal (optional default is 0)`

Adds leading zeros to the input value before conversion to match the given width. Ignored for '-dec'. Ignored if width is too small.

`inputValue`

Input value in binary format to be formatted. Leading zeros are ignored.

`radix`

Set the target format with -bin|-oct|-dec|-hex.

Example:

```
zdb formatvalue frombinary -hex -prefix 1101
```

zdb formatvalue fromspice

Convert values from Spice notation to a real number. Returns '0.0' for erroneous inputs.

Usage:

```
zdb formatvalue fromspice ?-hspice? inputValue
```

Parameters:

`-hspice (optional)`
: For inputs in hspice format. Enables the usage of 'a' (atto → 1e-18).

`inputValue`
: Input value in Spice notation to be formatted.

Example:

```
zdb formatvalue fromspice 10m
```

zdb formatvalue fromspicestrict

Convert values in Spice format to decimal. Returns empty string for erroneous inputs.

Usage:

```
zdb formatvalue fromspicestrict ?-hspice? inputValue
```

Parameters:

`-hspice (optional)`
: For inputs in hspice format. Enables the usage of 'a' (atto → 1e-18).

`inputValue`
: Input value in Spice format to be formatted.

Example:

```
zdb formatvalue fromspicestrict 10m
```

---

zdb formatvalue fromverilog

Convert a given verilog value in binary, decimal, octal or hexadecimal format into a decimal value.

Usage:

```
zdb formatvalue fromverilog inputValue
```

Parameters:

`inputValue`
: Input value in verilog format to be formatted.

Example:

```
zdb formatvalue fromverilog 8'b00000101
```

zdb formatvalue thousands

Add thousands separators to the input value.

Usage:

```
zdb formatvalue thousands inputValue ?sep?
```

Parameters:

`inputValue`
: Input value in dec format to be formatted.

`sep (optional)`
: Separator to be used. By default ',' is used.

Example:

```
zdb formatvalue thousands 1234567890
```

zdb formatvalue tobinary

Convert a number in binary, octal, decimal or hexadecimal format to a number in binary format.

Usage:

```
zdb formatvalue tobinary ?-prefix? ?-verilogPrefix? ?-width widthVal? radix inputValue
```

Parameters:

`-prefix (optional)`
: Has to be set if the input starts with an 'O'/'0x' for octal/hexadecimal input formats.

`-verilogPrefix (optional)`
: Add [0-9]'[bodh] prefix to the value.

`-width widthVal (optional default is 0)`
: Number of bits for the result value. Is ignored if the number is too small.

`inputValue`
: Input value in the specified format to be formatted.

`radix`
: Specify the input format: -bin|-oct|-dec|-hex.

Example:

```
zdb formatvalue tobinary -oct -prefix 012
```

zdb formatvalue tospice

Convert a real number into a Spice value.

Usage:

```
zdb formatvalue tospice ?-hspice? inputValue
```

Parameters:

`-hspice (optional)`
: For inputs in hspice format. Enables the usage of 'a' (atto → 1e-18).

`inputValue`
: Input value to be formatted.

Example:

```
zdb formatvalue tospice 1e-7
```

zdb formatvalue toverilog

Convert a decimal number to a number in Verilog format with bin|oct|dec|hex radix.

Usage:

```
zdb formatvalue toverilog ?-separatorDistance separatorDistance? ?-width widthVal? radix inputValue
```

Parameters:

`-separatorDistance separatorDistance` (optional default is 0)

Set to insert underscore separators with the distance of <separatorDistance> between them.

`-width widthVal` (optional default is 0)

Number of bits for the result value. Is ignored if the number is too small.

`inputValue`

Input value in the specified format to be formatted.

`radix`

Specify the output format: -bin|-oct|-dec|-hex.

Example:

```
zdb formatvalue toverilog -bin -width 10 123
```

Foreach Loops

```
$db foreach $iter_type $oid variable $body
```

loops over all elements of type `$iter_type` within the specified object. For each element, the specified loop `variable` is set and the `$body` is executed.

The body can access the loop `variable`.

The `$body` may execute `break`, `continue`, and `return` with the expected results.

The loop variable is actually another OID. For each iteration step, a new OID is created internally and assigned to the loop variable. These new OIDs always have the same module root and path as

the specified $oid.

It is not safe to run operators modifying the database while using a $db foreach ⋯ loop. Especially if the body of the $db foreach ⋯ loop modifies the current iteration type. E.g. $db foreach pin $net while using $db oper disconnect $pin in the body of the foreach loop will cause unpredictable results.

If the specified $oid does not exist or architecturally cannot "contain" elements of $iter_type then the foreach-loop will immediately return an error message (and will not execute the body).

Loop over Modules and Primitives

```
$db foreach top       cur $body
$db foreach module    cur $body
$db foreach primitive cur $body
$db foreach cell      cur $body
$db foreach parasitic cur $body
```

loop over top-modules, modules or primitives.

- the top-modules are a subset of the "modules" (often just one);
- $db foreach cell ⋯ loops over both modules and primitives;
- $db foreach parasitic ⋯ loops over the so-called parasitic modules in random order (they are disjoint to the "module" list).

For each iteration step, the loop variable cur is set to a new module OID or primitive OID. These five foreach-loops are the only ones that do not require an argument $oid (for initialization), since they are based on the database root.

Loop over Interface Ports

```
$db foreach port       $mod cur $body
$db foreach portBus    $mod cur $body
$db foreach oPort $dir $mod cur $body
```

Assuming $mod is a module or primitive OID, these foreach-commands loop over the interface ports and portBuses and set the loop variable cur accordingly.

$db foreach port ⋯ loops over single-bit ports (including bus members);

$db foreach portBus loops over portBuses;

$db foreach oPort ⋯ loops over both simultaneously, setting cur to either a port or a portBus; it does not include bus members and visits the ports/portBuses in a predefined order (as defined by the bus-level functions, where single-bit ports and portBuses are mixed); additionally, oPort can filter ports by direction $dir:

*	loop over all ports and portBuses

`input`	loop over input ports and input portBuses, skip output, inout and unknown directions
`output`	loop over output ports/portBuses
`inout`	loop over inout ports/portBuses

The commands

```
$db foreach couplingPort $parasitic cur $body
$db foreach couplingInst $parasitic cur $body
```

can be used to loop over all coupling connections and coupling instances of a parasitic module.

Loop over Instances, Nets and Buses

```
$db foreach inst     $mod cur $body
$db foreach clocked  $mod cur $body
$db foreach operator $mod cur $body
$db foreach primInst $mod cur $body
$db foreach net      $mod cur $body
$db foreach netBus   $mod cur $body
```

Assuming `$mod` is a module OID, these foreach-commands loop over the module contents and set the loop variable `cur` accordingly.

```
$db foreach pin       $inst cur $body
$db foreach pinBus    $inst cur $body
$db foreach oPin $dir $inst cur $body
```

Assuming `$inst` is an instance OID, `$db foreach pin/pinBus ⋯` loop over pins or pinBuses of a given instance and set the `cur` variable accordingly; `$db foreach oPin ⋯` works analogous to "oPort".

Loop over Bus Information

```
$db foreach net    $netBus  cur $body
$db foreach member $netBus  cur $body
$db foreach pin    $pinBus  cur $body
$db foreach port   $portBus cur $body
```

Loop over single bit nets, pins or ports of a given bus object (as before, `$netBus`, `$pinBus`, and `$portBus` are OIDs).

The `$db foreach net ⋯` and `$db foreach member ⋯` are identical, except `$db foreach net ⋯` will skip null OIDs (representing gaps in a netBus definition).

Search Objects by Name

```
$db search ?-icase? top        $name
$db search ?-icase? module     $name
$db search ?-icase? primitive  $name
$db search ?-icase? cell       $name
$db search ?-icase? parasitic  $name
```

Analogous to the foreach loop above, these commands search for a top-module, module, or primitive by `$name`;

`$db search cell` ⋯ searches for a module or a primitive;

`$db search parasitic` ⋯ searches for a so-called parasitic module.

The found object is returned as an OID — or a null OID is returned if the object cannot be found.

If the option `-icase` is specified then the object is searched in a case insensitive manner.

```
$db search ?-icase? port     $cell $name
$db search ?-icase? portBus  $cell $name
```

Analogous to the foreach loop above, these commands search for a port or portBus in the given module or primitive (as specified by `$cell`).

```
$db search ?-icase? inst     $mod  $name
$db search ?-icase? net      $mod  $name
$db search ?-icase? netBus   $mod  $name
$db search ?-icase? pin      $inst $name
$db search ?-icase? pinBus   $inst $name
$db search ?-icase? member   $nbus $name (deprecated)
```

Analogous to the foreach loop above, these commands search for an inst, net or netBus in the given module (`$mod`) or for a pin or pinBus in the given instance (`$inst`).

```
$db search ?-icase? net   $netBus  $name
$db search ?-icase? pin   $pinBus  $name
$db search ?-icase? port  $portBus $name
```

Analogous to the foreach loop above, these commands search for a sub-net, sub-pin or sub-port in the given netBus, pinBus or portBus respectively.

Getting Bus Information

```
$db isBusMember $net
$db isBusMember $pin
$db isBusMember $port
$db busOf   $net
$db busOf   $pin
$db busOf   $port
$db widthOf $netBus
$db widthOf $pinBus
$db widthOf $portBus
$db hasRange $bus
$db rangeLsb $bus
$db rangeMsb $bus
```

`$db isBusMember $net/$pin/$port` returns a boolean value indicating whether the object is member of a bus;

`$db busOf $net/$pin/$port` return the OID of the netBus, pinBus or portBus (or an error if the specified `$net`/`$pin`/`$port` is not a member of a bus);

`$db widthOf $netBus/$pinBus/$portBus` returns an integer defining the bus width.

`$db hasRange $bus` returns true if the given bus has a range direction.

`$db rangeLsb $bus` returns the index of the bus' LSB.

`$db rangeMsb $bus` returns the index of the bus' MSB.

By convention the parsers store the LSB of a bus first.

Getting Hierarchy Information

```
$db isTop       $mod
$db isModule    $inst|$pin|$pinBus|$mod|$prim
$db isOperator  $inst
$db moduleOf    $inst|$pin|$pinBus
$db primitiveOf $inst|$pin|$pinBus
```

The first three commands return a boolean value; the fourth command returns the down-module OID (the module referred to by the specified instance) or an error if `$inst` is not a module; the last command returns the primitive OID as referred to by the specified instance, or an error if `$inst` is not a primitive. Instead of `$inst`, a `$pin` or `$pinBus` is also accepted with the same effect.

```
$db isEmpty $module ?-checkDanglingNets?
```

`$db isEmpty …` command returns true if the given module is empty (contains no instance or net). If the option `-checkDanglingNets` is given then nets with no or only one connection are ignored.

```
$db refCount $oid
```

`$db refCount ⋯` returns an integer representing the number of instances referring to the Module or Primitive given as `$oid`.

But, any other kind of OID can be given as `$oid` — in that case, the `refCount` of the `$oid`'s parent module is returned.

```
$db hasParentInst $inst
$db parentInst    $inst
```

The first command returns a boolean value; the second command returns an inst OID for the instance one hierarchy level above. Instead of `$inst`, a `$net` or `$netBus` is also accepted with the same effect. This only works for tree-based OIDs.

```
$db hasParentMod $oid
$db parentModule $oid
```

the first command returns a boolean value; the second command returns a module OID, pointing to the module that "contains" the given `$oid`. If the `$oid` itself is a tree-based module or primitive, then the path is shortened by one element.

More hierarchy information is returned by the `$db oid down/up ⋯` commands.

Getting Primitive Function

```
$db primFuncOf $prim
$db primFuncOf $mod
$db primFuncOf $inst
$db primFuncOf $pin
```

returns the name of the primitive function, e.g. `NAND`, `NMOS` etc. or `UNKNOWN`. For compatibility, the old command name `$db primitiveFuncOf ⋯` is also supported with the same function.

```
$db setPrimitive $level
```

Set a new primitive level according to the given value of $level which is a combination of `-clear`, `-operator`, `-libcell`, `-celldefine`, `-flag` and `-symbol`.

- The option `-clear` removes all primitive level settings.
- The option `-operator` specifies all cells with a known function (Operators) as primitives.
- The option `-libcell` specifies all cells flagged as library cells (e.g. coming from a binlib) as primitives.

- The option `-celldefine` specifies all cells surrounded by the Verilog macros `` `celldefine `` and `` `endcelldefine `` as primitives.
- The option `-flag` specifies all modules flagged with the leafcell flag as primitives.
- The option `-symbol` specifies all cells with a `@symbol` attribute (from a symlib) as primitives.

```
$db oid insideprim $oid
```

returns true if the given `$oid` is inside a primitive "module". These OIDs are returned by the picklist command if the `-insideprim` option is given.

Compare Cell Interfaces

```
$db identicalInterface ?-icase? $cellA $cellB
```

Compare if the interfaces of two given cells are identical. If identical, then this function returns true, false otherwise.

The interface includes port and portBus names, direction and order. Update `$cellA`'s port directions from `$cellB` and vice versa, whatever has more information: update `unknown` by `inout`/`input`/`output` and update `inout` by `input`/`output`.

If the optional argument `-icase` is specified, then the port names are compared in a case insensitive manner.

Report Module Size/Insts/Nets

```
$db report sizeOf    $mod
$db report instCount $result $mod
$db report netCount  $mod
$db report portCount $mod
$db report objCount  ?-cell|-inst|-pin|-net|-netBus|port|portBus|-all?
                     ?$mod|-hier?
$db report histogram
$db report designStatistics $filename
```

`$db report sizeOf ⋯` returns a triplet (list with 3 elements) containing the number of ports, instances and nets of the given module.

`$db report instCount ⋯` computes the number of primitives and modules (all instantiations) and stores the result into the given table (`$result` is the name of the array variable where the results are stored). For each entry, the key is the primitive or module name (prefixed by the character `P` or `M`) and the value is the number of instances.

`$db report netCount/portCount ⋯` compute and return the number of nets and ports, respectively (in all instantiations).

`$db report objCount ⋯` reports the number of inst, port and net objects in all instantiations. If the option `-hier` is given then only one instance of a module is used to count the total number of objects. By adding a type option the returned object count can be limited to the given type.

`$db report histogram` report statistic (histogram) data for the given database.

`$db report designStatistics` prints statistic information of the loaded design database into the given file.

Count Objects

```
$db count cell
$db count module
$db count primitive
$db count parasitic
$db count inst $mod
$db count net $mod
$db count netBus $mod
$db count port $cell
$db count portBus $cell
$db count member $netBus|$portBus
$db count pin $net ?-noport?
```

The `$db count ⋯` commands return the number of elements of the given type in the database respectively given owner.

When using `$db count pin $net`, connected ports may be excluded with `-noport`.

Is Transistor Device

```
$db tdevice $inst|$prim
```

returns true if the given instance or primitive OID refers to a device function.

Is Polar Device

```
$db tdevice polar $inst|$prim
```

returns true if the given instance or primitive OID refers to a polar device function.

Get Opposite Pin

```
$db tdevice oppositePin $pin
```

For transistor-level primitives, return the "opposite pin" of the given pin or the "opposite port" of the given port; that is for RES, CAP, etc. just the other pin.

For NMOS and PMOS jump from drain to source and vice versa — does not work for gate and bulk pins.

For NPN and PNP jump from collector to emitter and vice versa — does not work for base and substrate pins.

Get Pin by Function

```
$db getFuncPort $cell|$inst ?-name? ?-noError? $orderedPort
```

Get the instance pin or cell port at the given ordered port number.

- The option `-noError` will return a empty list for not matching ports.
- The option `-name` allows specifying one of the port functions `select`, `set`, `reset`, `clock`, `enable`, `drain`, and `gate` as `$orderedPort`.

Get Bus Member by Index

```
$db getMember $portBus|$netBus -index|-bit $idx ?-noError?
```

- Get the member of a portBus or netBus at the given index number, if `-index` is used.
- Get the member of a portBus or netBus with the given bit subscript, if `-bit` is used.
- The option `-noError` will return a null oid for invalid options instead of raising a Tcl error.

Getting Pin Direction

```
$db directionOf $port
$db directionOf $pin
$db directionOf $portBus
$db directionOf $pinBus
```

returns one of `input`, `output`, `inout`, and `unknown`.

For primitive ports (and primitive instance pins), the return string may be extended by a `.neg` suffix to indicate that their function is inverted, e.g. `input.neg`.

Getting Connectivity Information

```
$db isConnected $pin
$db isConnected $port
$db connectedNet $pin
$db connectedNet $port
$db isOneToOneConnection ?-checksequence? $netBus $pinBus
$db isOneToOneConnection ?-checksequence? $netBus $portBus
```

- The `$db isConnected ⋯` command returns a boolean value indicating whether the object is connected to a net.
- The `$db connectedNet ⋯` command returns a net OID (or an error if the object is not connected to a net).
- The `$db isOneToOneConnection ⋯` command checks if all subnets of the given netBus are connected to the given pinBus or portBus. If `-checksequence` is used, the first member of the netbus must be connected to first member of the pinBus or portBus and all other connections must be in sequence.

```
$db foreach pin $net cur $body
```

loops over the net connections. For each iteration step, the loop variable `cur` is set to either a port OID or a pin OID.

The loops

```
$db foreach pinCon  $net cur $body
$db foreach portCon $net cur $body
```

can be used to get only the pins or the ports connected to the given net object.

Top Module

```
$db top set $mod
$db top unset $mod
```

The `$db top ⋯` commands add or remove a module from the top list.

Clone Object

```
$db clone ?-bitblasted? ?-skipflaggedport $flag? ?-skipflaggedinst $flag?
          ?-skipcontent? ?-nospos? $oid ?$rename?
```

The `$db clone ⋯` command clones an object and returns the OID of the newly created object. If the optional `$rename` is missing, a unique name is created. Currently only primitives and modules are supported. If the option `-bitblasted` is specified, then a bit-blasted version of the primitive or module is created. If the original cell has a bus level function then this function is not cloned.

The option `-skipflaggedport $flag` will not clone ports flagged with the given `$flag`. If the original cell has a function then this function is not cloned if `-skipflaggedport` is specified.

The option `-skipflaggedinst $flag` will not clone instances inside the module to be cloned flagged with the given flag. If `-skipcontent` is given then this option is not used.

The `-nospos` option can be used to save memory by not cloning spos information.

Clone Database

```
$db cloneDB ?-into $otherdb?
            ?-keepcell $flag? ?-keepinst $flag?
            ?-keepnet $flag?  ?-keepport $flag?
            ?-keeppin $flag?
            ?-nospos?
```

The `$db cloneDB ⋯` command clones the database with all flagged objects and returns the new database.

The `-keep⋯` options define the flags for objects which should be cloned. Flagged nets/netbuses and ports/portbuses are only cloned if the parent module is flagged. Flagged instances are only cloned if the parent module and the referenced cell is flagged. The connections of pins is only cloned if the pins and nets are flagged.

In none of the `-keep⋯` options is given, all objects will be cloned.

The `-into` option can be used to merge or clone flagged or all object into the given database. This can be used to merge multiple databases.

The `-nospos` option can be used to save memory by not cloning spos information.

Merge Database

```
$db mergeDB $fromDb ?-rename? ?-nospos?
```

The `$db mergeDB ⋯` command clones all cells from `$fromDb` into `$db`.

If the `-rename` option is given, existing cells with the same name get renamed. Otherwise the interface must match. If a existing module has already contents the new contents is ignored.

Get/Set Attributes

```
$db attr $oid add $attr ?$attr $attr ...?
$db attr $oid set $attr ?$attr $attr ...?
$db attr $oid delete ?$name $name ...?
$db attr $oid get ?-sorted?
$db attr $oid getValue $name
$db attr $oid foreach cur_attr $body
```

Both, `$db attr $oid add ⋯` and `$db attr $oid set ⋯` append one or more attributes to the given `$oid` — however, `set` overwrites existing attributes (avoids duplicates) while `add` simply appends.

The format of `$attr` is `name=value`.

`$db attr $oid delete ⋯` removes the attributes with the specified names from the given OID — or all attributes (if no names are specified).

`$db attr $oid get ⋯` returns a list of all attributes (list of `name=value` elements). If the optional switch `-sorted` is given then a sorted list of all attributes is returned.

`$db attr $oid getValue ⋯` returns just the value of the specified attribute name (or an empty string if not found).

`$db attr $oid foreach cur_attr $body` loops over the attributes of the specified object. The loop variable `cur_attr` is set to the current attribute string (in `name=value` format).

All six functions accept `-db` instead of `$oid` — in this case, the database root attributes are processed.

Get/Set Flags

```
$db flag $oid|<SCOPE> set   $flag
$db flag $oid|<SCOPE> clear $flag
$db flag $oid is            $flag
$db flag $oid is_set        $flag
$db flag $oid is_clear      $flag
$db flag $oid get
$db flag $oid clear
$db flag $oid toggle        $flag
$db flag $oid supported     $flag
$db flag $type available
```

The `set` and `clear` sub-commands set and clear a flag at the specified object of at all objects of the specified scope.

If the scope is `-db` then the given flag will be set or cleared at all database objects. Further possible scopes are: `-cell`, `-module`, `-primitive`, `-inst`, `-net`, `-netBus`, `-port`, `-portBus`, and `-pin` to limit the processed object type.

`$db flag $oid is $flag` returns true if the `$flag` is set at the specified object;

`$db flag $oid is_set|is_clear $flag` also return the current flag state (like `$db flag $oid is $flag`), but afterwards set/clear the flag.

`$db flag $oid get` returns a list of all set flags at the given object.

`$db flag $oid supported $flag` returns true if `$flag` is supported at the given `$oid`.

`$db flag $type available` returns a list of all supported flags for the given OID `$type`.

`$flag`	**Description**	**Supported at**			
		cell	**inst**	**net**	**port/pin**
`zombie`	used by the `$db deleteZombies` command	Yes	Yes	Yes	

red green blue yellow target	general purpose, for Tcl Userware, they have no special meaning in the database. The clear command without a flag-name argument clears all these "general purpose" flags.	Yes	Yes	Yes	Yes
visit visit2 visit3	general purpose	Yes Yes Yes	Yes	Yes Yes Yes	Yes
hide	hide this symbol port or portBus in the schematic windows (all instantiations of modules and primitives are affected); hiding primitive ports is limited to the gate-level functions UNKNOWN and DFF and the bulk pins at some Spice-level functions (e.g. NMOS, PMOS, etc). The hide flag at a pin displays that pin disconnected in the schematic windows. (In addition to the hide flag, the Meta Attributes also control visibility in the schematic windows).				Yes
neg	bubble symbol port				Yes
top bottom left right	place this port at the top, bottom, left or right side.				Yes
clock	mark this port as a clock input pin				Yes
driver	mark this pin/port as a driver				Yes
weakflow	declare transistor primitives and instances to drive only a small current (is a weak transistor).	Yes	Yes		
power ground negpower	this net is a power or ground or negative power net (or constant 1 or 0 in digital logic)			Yes	
undefined	this is a black-box (the parser was not able to get any information about that module)	Yes			
feedthru	for incremental schematic navigation pass this buffer-like cell	Yes			
libcell	this is considered to be a library cell, e.g. defined by `celldefine in Verilog	Yes			
leafcell	a module flagged with as a leafcell can be changed into a primitive by $db setPrimitive ⋯.	Yes (module)			
group	group flagged instances in the tree		Yes		
autogen	this object is not declared by the user		Yes	Yes	

global	for future use			Yes	

Power/Ground Nets

```
$db isPgNet $oid
```

Service function to test if the given net or signal is flagged as power or ground.

Constant Nets

```
$db value $oid ?$value?
```

`$db value $oid` returns the constant value of the `$oid` (`$oid` must be of type net, netBus or signal).

If `$oid` is a signal and there's a value mismatch at the nets of the signal, and error is returned.

If called with the optional `$value` argument, then a constant value is set at the given net, netBus or signal object.

Possible values are: `0`, `1`, `X`, and `Z`. If an empty string is specified, then the value at the net, netBus or signal object is deleted.

Deal with Highlight Colors

The following commands can be used to highlight objects module based. An object inside a module that is instantiated multiple times will be highlighted with the given color in each instance of this module. The `flathighlight` API can be used to highlight individual objects only in one specific instantiation path.

```
$db hilight $oid set $color ?-permanent?
$db hilight $oid get    ?-permanent|-both?
$db hilight $oid delete ?-permanent|-both?
$db hilight *    deleteAll ?$color? ?-permanent|-both?
```

The `set` sub-command sets the highlight color for the object specified by `$oid` to `$color`.

The `get` sub-command returns the current highlight color or an empty list (`{}`) if nothing is set.

The `delete` sub-command removes the highlight color from the object specified by `$oid`.

With the `deleteAll` sub-command all highlights are removed. The optional `$color` argument specifies the color of the objects which should be removed. It may be omitted if all highlights should be removed.

There are two sets of highlight colors for each object. "normal" and `-permanent` colors. They can be set and deleted independent of each other. If both are set, the "normal" color has priority over the permanent color.

$color may be in the range 0..48.

Using the option -both to get either the normal or permanent highlight color will return the permanent value as a negative number to be able to distinguish between normal highlight.

Helper with Highlight Colors

The following commands can be used to highlight objects in module based and flat data. Parasitic and database object are both handled if possible.

```
$db highlight $oid set $color ?-permanent?
$db highlight $oid get        ?-permanent|-both?
$db highlight $oid delete     ?-permanent|-both?
```

Get/Set Instance Orientation

```
$db orient $inst ?R0|R90|R180|R270|MY|MYR90|MX|MXR90?
$db orient $inst ?+R0|+R90|+R180|+R270|+MY|+MYR90|+MX|+MXR90?
$db orient $inst ?-R0|-R90|-R180|-R270|-MY|-MYR90|-MX|-MXR90?
```

$db orient ⋯ gets or sets the orientation of the given instance object.

If the value is prefixed with +/- the orientation is applied relative to the current position. The rotation is applied counter-clockwise if prefix is + and clockwise if prefix is -.

If the optional "orientation" argument is omitted, then the current orientation of the instance is returned, otherwise the orientation is set to the given value.

Validate

```
$db validate ?options?
```

Validate the integrity of the database.

The supported ?options? are:

-namespace	additionally validate object names in the database — to check if they are unique
-icasenamespace	ditto, but perform the check in a case-insensitive way
-spos	additionally validate source file positions
-mode9	perform extra checks on Prim Functions
-flat	additionally check flat data
-torder	additionally check for a cycle in the design hierarchy.

Symlib Support

The symlib operations are used to maintain graphical symbol information in the database.

The typical usage is:

```
$db symlib add     -fromattr|-fromfunc|-fromstr $str|$fname
$db symlib scan    ?-force?
$db symlib clear
$db symlib remove
$db symlib copy
$db symlib scandef ?-icase? $symdef
$db symlib info
$db symlib create  mySymbols.sym
```

The `add` sub-command reads the given symlib file, `@symbol` attributes or cell functions and adds it to the database.

The `scan` sub-command updates the symbol information of the database primitives, modules, ports and power/ground nets.

The `-force` option will overwrite symbol information if it already exists at a given cell. Otherwise it will be left unchanged.

The `clear` command removes all symlib information while the `remove` command only removes the symbol libraries add using the `add` command.

The `copy` command can be used to create database primitives and modules from added symlib information.

`scandef` splits the given symbol definition string into a list of separate entries for each contained keyword.

`info` can be used to get information contained in the symlib header of an added file.

`create` extracts the symbol information from the given database and creates a symlib file. An existing file will be overwritten.

Details about the symlib file can be found in the Symlib Tutorial.

Symdef Support

The graphical symbol information for port and power ground symbols is stored in the symdef table of a database and referenced by `@sym` attributes at the ports and power ground nets. The symdef table is maintained by the following commands:

```
$db symdef set    $name $def
$db symdef get    $name
$db symdef delete $name
$db symdef list
```

The `set` sub-command adds or overwrites a symbol definition with the given name.

The `get` sub-command returns a list containing the stored symbol definition and the optional io mapping info or an empty string if the given name is not stored in the table.

With the `delete` sub-command the information for the given symbol name is removed from the table.

The `list` sub-command returns a list of the names of all stored symbols.

The Primitive Functions

Introduction

This document describes the "Primitive Functions" of the DataBase API; The primitives in the design hierarchy are usually primitive objects with a known function (e.g. `AND`, `NOR`, etc.). However, some algorithms, e.g. Cone Extraction will not traverse the design hierarchy tree down to the primitives but may "stop" at bus-level (sometimes called "operator-level") where the function of a module is known (these modules are treated as "primitives" for that algorithm).

Overview

Here is an overview of the various module and primitive types. The **function** is returned by `primFuncOf`. The **flags** are returned or set by the `flag` command.

OID Type	Function	Has Sub-Netlist	Flags	Description	Index
module	NONE	Yes		Any hierarchy module that is defined by the lower-level netlist only.	#1
		No		Empty module.	#2
			undefined	Undefined module (= **blackbox** module); ports may have unknown directions.	#3
		Yes/No	libcell	Any library cell, normally with lower-level netlist (eventually with attached symbol shape).	#4
	some function (e.g. WIDE_BUF)	Yes/No		Module with a known (bus-level) function (sometimes called "operator"), may have an implementing lower-level netlist (redundant data).	#5
primitive	some function (e.g. NAND)	No		Primitive with a known function (similar to #5).	#6
	UNKNOWN	No		Unknown primitive from a source like EDIF that does not know anything about library cells (similar to #2).	#7

The functions NONE and UNKNOWN are identical, NONE is used at modules and UNKNOWN at primitives.

Note that some GUI features, like the Tree window, traverse the design hierarchy down to the real primitive objects; that means, it distinguishes modules (#1 to #5) from primitives (#6 to #7). However some algorithms may stop as soon as a function is known, and therefore consider #5 to #7 as "primitives"; other algorithms may stop at library cells and consider #4 to #7 as "primitives".

For a better understanding, here is what the various parsers create:

Parser	Module/Primitive Objects Created
liberty2zdb, lib2zdb	Creates library cells (#4) with symbol shape definition and with optional content lower-level netlist consisting of operators (#5) and primitives (#6).
verilog2zdb, rtl2zdb	Creates hierarchy modules (#1) with netlists consisting of operators (#5) and primitives (#6); if a module is defined inside `celldefine or read from a -v or -y file, then it is flagged as library cell (#4); undefined instantiation creates an undefined module (#3).
edif2zdb	Creates hierarchy modules (#1) and unknown primitives (#7).
spice2zdb	Creates hierarchy modules (#1) and Spice primitives (#6).

Parser	Module/Primitive Objects Created
symlib2zdb	Creates library cells (#4) with symbol shape definition.
lef2zdb	Creates empty library cells (#4).
def2zdb	Creates hierarchy modules (#1) with netlists of cells (e.g. from lef2zdb).

Port Function

This database applies the same port-order rules to Primitives as well as to Modules with defined function. That means, if a function is defined, then the number of ports and their directions and their function is ruled by the database built-in specification, as defined in this document. The database enforces that those rules are not violated.

Ordinary modules (with function = NONE) as well as primitives with UNKNOWN function have no port-restrictions at all (any number of ports and portBuses in arbitrary order). All ports can be bubbled/negated.

Primitives (as well as modules) with a known function define the direction and function of their ports by the port-order. For each function, only a certain number and order of the ports is possible. Also depending on the function, some ports may be bubbled/negated (details below).

All port names are arbitrary (the names in this document are chosen only for better understanding). Of course, all port names within a primitive or module are unique.

Load and Retrieve the Port Direction

For UNKNOWN primitives and modules without a function (function = NONE), loading a port must specify a port direction. However, for a primitive or module with a known function, loading a port can (and should) use a * for the port direction (to apply the built-in direction) — or if a direction is specified, then it must match the built-in specification or an error is returned. Independent from the function, the command directionOf returns the correct direction.

Bubbled Ports

Some ports or portBuses can be bubbled to define negated port functions (and to display them bubbled). If so, then the command directionOf will append .neg to the returned port direction, e.g. input.neg (instead of input). The .neg extension can also be used for API-loading, including *.neg for known functions, as described in the tables below.

Definition of Primitive Functions

Special Primitives

UNKNOWN

Description

An unknown function with an arbitrary number and order of input/output/inout ports (no restrictions).

A mix of single-bit ports and portbuses with arbitrary widths is supported.

All ports and portbuses can be bubbled.

Ports

No	Name	Type
…	…	…

Gate-Level Functions

Description

The primitive functions below describe gate-level functions.

The function names and port orders partly follow the Verilog gate definitions.

All ports are single-bit (except for `UNKNOWN`).

On the API-level, a cell's function is specified in the corresponding `load` command call (`$db load primitive $name $function ⋯` for primitives, `$db load module $name ?-primfunc $function? ⋯` for modules), while the primitive function of a cell can be queried using the `primFuncOf` command (`$db primFuncOf $cell`).

Combinational Gates

AND

Description

Boolean "AND" function.

The first port is the output, followed by one or more input ports.

The input ports can be bubbled.

Ports

No	Name	Type
0	Y	output
1	A	input
2	B	input
…	…	input

NAND

Description

Boolean "NAND" function.

The first port is the output, followed by one or more input ports.

The input ports can be bubbled.

Ports

No	Name	Type
0	Y	output
1	A	input
2	B	input
…	…	input

OR

Description

Boolean "OR" function.

The first port is the output, followed by one or more input ports.

The input ports can be bubbled.

Ports

No	Name	Type
0	Y	output
1	A	input
2	B	input
…	…	input

NOR

Description

Boolean "NOR" function.

The first port is the output, followed by one or more input ports.

The input ports can be bubbled.

Ports

No	Name	Type
0	Y	output
1	A	input
2	B	input

No	Name	Type
...	...	input

XOR

Description

Boolean "XOR" function.

The first port is the output, followed by one or more input ports.

The input ports can be bubbled.

Example

```
$db load primitive xor4 XOR
$db load port Y output
$db load port A input
$db load port B input.neg
$db load port C input.neg
$db load port D input
```

Ports

No	Name	Type
0	Y	output
1	A	input
2	B	input
...	...	input

XNOR

Description

Boolean "XNOR" function.

The first port is the output, followed by one or more input ports.

The input ports can be bubbled.

Ports

No	Name	Type
0	Y	output
1	A	input

No	Name	Type
2	B	input
…	…	input

BUF

Description

(Multi-output) buffer.

All ports but the last port are outputs, the last port is the input (as Verilog defines 'buf').

Ports

No	Name	Type
0	Y	output
…	…	output (optional)
last	I	input

FORCE

Description

(Multi-output) "forced" buffer, typically generated from a "force"-type assignment in Verilog/VHDL.

All ports but the last port are outputs, the last port is the input.

Ports

No	Name	Type
0	Y	output
…	…	output (optional)
last	I	input

INV

Description

(Multi-output) Boolean "Inverter" function.

All ports but the last port are outputs, the last port is the input (as Verilog defines 'inv').

Ports

No	Name	Type
0	Y	output
...	...	output (optional)
last	I	input

MUX

Description

2-input multiplexer.

The first port is the output, the second and third ports are the inputs `A` and `B`, the fourth port is the select port `Sel`; if `Sel` is `0`, `A` is switched to the output, if `Sel` is `1`, `B` is switched to the output.

Ports

No	Name	Type
0	Y	output
1	A	input
2	B	input
3	Sel	input

ADD

Description

1-bit full adder with carry-in and carry-out.

The first port is the **sum** output, the second port is **carry-out**, the third and fourth ports are the **adder inputs**, and the fifth port is the **carry-in**.

Example

```
$db load primitive adder ADD
$db load port Y  output
$db load port Co output
$db load port A  input
$db load port B  input
$db load port Ci input
```

Ports

No	Name	Type
0	Y	output

No	Name	Type
1	Co	output
2	A	input
3	B	input
4	Ci	input

State Elements

DFF

Description

Simple flip-flop.

The first port is the **output**, the second port is the data **input**, the third port is the **clock** (active on positive edge), the optional fourth and fifth ports are **reset** and **set** inputs. The clock, set and reset inputs may be bubbled.

Ports

No	Name	Type
0	Y	output
1	D	input
2	Clk	input
3	Reset	input (optional)
4	Set	input (optional)

DFFN

Description

Simple flip-flop with a non-inverting output and an inverting output.

The first port is the non-inverting **output**, The second port is the inverting **output**, the third port is the data **input**, the fourth port is the **clock** (active on positive edge), the optional fifth and sixth ports are **reset** and **set** inputs. The clock, set and reset inputs may be bubbled.

Ports

No	Name	Type
0	Y	output
1	YN	output

No	Name	Type
2	D	input
3	Clk	input
4	Reset	input (optional)
5	Set	input (optional)

DLATCH

Description

Simple latch.

The first port is the **output**, the second port is the data **input**, the third port is the **enable** input, the optional fourth and fifth ports are **reset** and **set** inputs. The enable, set and reset inputs may be bubbled.

Ports

No	Name	Type
0	Y	output
1	D	input
2	En	input
3	Reset	input (optional)
4	Set	input (optional)

DLATCHN

Description

Simple latch with a non-inverting output and an inverting output.

The first port is the non-inverting **output**, The second port is the inverting **output**, the third port is the data **input**, the fourth port is the **enable** input, the optional fifth and sixth ports are **reset** and **set** inputs. The enable, set and reset inputs may be bubbled.

Ports

No	Name	Type
0	Y	output
1	YN	output

No	Name	Type
2	D	input
3	En	input
4	Reset	input (optional)
5	Set	input (optional)

DFFCV

Description

Condition/value flip-flop.

The first port is the **output**, the second port is the data **input**, the third port is the **clock** (active on positive edge), the fourth and fifth ports are **condition** and **value** inputs. The clock may be bubbled.

Ports

No	Name	Type
0	Y	output
1	D	input
2	Clk	input
3	Condition	input
4	Value	input

DLATCHCV

Description

Condition/value latch.

The first port is the **output**, the second port is the data **input**, the third port is the **enable** input, the fourth and fifth ports are **condition** and **value** inputs. The enable input may be bubbled.

Ports

No	Name	Type
0	Y	output
1	D	input
2	En	input
3	Condition	input (optional)

No	Name	Type
4	Value	input (optional)

Other Functions

BUFIF0

Description

A tri-state buffer.

The first port is the **output**, the second port is the **input**, the third port is the **control** input.

If the control input is `1`, the buffer is set to high-impedance state (as Verilog defines `bufif0`).

Ports

No	Name	Type
0	Y	output
1	I	input
2	Ctrl	input

BUFIF1

Description

A tri-state buffer.

The first port is the **output**, the second port is the **input**, the third port is the **control** input.

If the control input is `0`, the buffer is set to high-impedance state (as Verilog defines `bufif1`).

Ports

No	Name	Type
0	Y	output
1	I	input
2	Ctrl	input

INVIF0

Description

A tri-state inverting buffer.

The first port is the **output**, the second port is the **input**, the third port is the **control** input.

If the control input is 1, the buffer is set to high-impedance state (as Verilog defines `notif0`).

Ports

No	Name	Type
0	Y	output
1	I	input
2	Ctrl	input

INVIF1

Description

A tri-state inverting buffer.

The first port is the **output**, the second port is the **input**, the third port is the **control** input.

If the control input is `0`, the buffer is set to high-impedance state (as Verilog defines `notif1`).

Ports

No	Name	Type
0	Y	output
1	I	input
2	Ctrl	input

TRAN

Description

A transfer gate.

This primitive has exactly two **inout** (bidirectional) ports (as Verilog defines `tran`).

Ports

No	Name	Type
0	P1	inout
1	P2	inout

TRANIF0

Description

A bi-directional transistor (low).

This primitive has two **inout** (bidirectional) ports, and a **control** input (as Verilog defines `tranif0`).

Ports

No	Name	Type
0	P1	inout
1	P2	inout
2	Ctrl	input

TRANIF1

Description

A bi-directional transistor (high).

This primitive has two **inout** (bidirectional) ports, and a **control** input (as Verilog defines `tranif1`).

Ports

No	Name	Type
0	P1	inout
1	P2	inout
2	Ctrl	input

RTRANIF1

Description

A resistive transistor (high).

This primitive has two **inout** (bidirectional) ports, and a **control** input (as Verilog defines `rtranif1`).

Ports

No	Name	Type
0	P1	inout
1	P2	inout
2	Ctrl	input

GNMOS

Description

A gate-level NMOS transistor.

(as Verilog defines `nmos`)

Example

```
$db load primitive gnmos GNMOS
$db load port Y *
$db load port A *
$db load port C *
```

Ports

No	Name	Type
0	Y	output
1	I	input
2	Ctrl	input

GPMOS

Description

A gate-level PMOS transistor.

(as Verilog defines `pmos`)

Example

```
$db load primitive gpmos GPMOS
$db load port Y *
$db load port A *
$db load port C *
```

Ports

No	Name	Type
0	Y	output
1	I	input
2	Ctrl	input

GRNMOS

Description

A gate-level resistive NMOS transistor.

(as Verilog defines `rnmos`)

Example

```
$db load primitive grnmos GRNMOS
$db load port Y *
$db load port A *
$db load port C *
```

Ports

No	Name	Type
0	Y	output
1	I	input
2	Ctrl	input

GRPMOS

Description

A gate-level resistive PMOS transistor.

(as Verilog defines `rpmos`)

Example

```
$db load primitive grpmos GRPMOS
$db load port Y *
$db load port A *
$db load port C *
```

Ports

No	Name	Type
0	Y	output
1	I	input
2	Ctrl	input

GCMOS

Description

A combination of gate-level NMOS and PMOS transistors with common data input and output ports.

(as Verilog defines `cmos`)

Example

```
$db load primitive gcmos GCMOS
$db load port Y *
$db load port A *
$db load port nC *
$db load port C *
```

Ports

No	Name	Type
0	Y	output
1	A	input
2	nCtrl	input
3	Ctrl	input

GRCMOS

Description

A combination of gate-level resistive NMOS and PMOS transistors with common data input and output ports.

(as Verilog defines `rcmos`)

Example

```
$db load primitive grcmos GRCMOS
$db load port Y *
$db load port A *
$db load port nC *
$db load port C *
```

Ports

No	Name	Type
0	Y	output
1	A	input
2	nCtrl	input
3	Ctrl	input

PULLUP

Description

A "pull-up" resistor.

Example

```
$db load primitive pu PULLUP
$db load port a *
```

Ports

No	Name	Type
0	pt1	output

PULLDOWN

Description

A "pull-down" resistor.

Example

```
$db load primitive pd PULLDOWN
$db load port a *
```

Ports

No	Name	Type
0	pt1	output

Bus-Level Functions

Description

The primitive functions below describe bused functions (word-level).

The function is defined e.g. by the load API and returned by primFuncOf.

All ports are usually a mix of multi-bit buses and single-bits. In the table below, the column "Ports/Type" uses a square-bracket notation to identify buses (portBus instead of port); Buses which are required to have the same width use the same index denominator, while buses with potentially different widths use different index denominators (e.g. `[n]` and `[m]`).

Reducing Functions (n to 1)

REDUCE_AND

Description

Bus-input reducing Boolean "AND" function.

The first port is the **output**, followed by one **input** portbus of arbitrary width.

```
Y = AND(A[0], ⋯, A[n-1])
```

Ports

No	Name	Type
0	Y	output
1	A	{input[n]}

REDUCE_NAND

Description

Bus-input reducing Boolean "NAND" function.

The first port is the **output**, followed by one **input** portbus of arbitrary width.

```
Y = NAND(A[0], ⋯, A[n-1])
```

Ports

No	Name	Type
0	Y	output
1	A	{input[n]}

REDUCE_OR

Description

Bus-input reducing Boolean "OR" function.

The first port is the **output**, followed by one **input** portbus of arbitrary width.

```
Y = OR(A[0], ⋯, A[n-1])
```

Ports

No	Name	Type
0	Y	output
1	A	{input[n]}

REDUCE_NOR

Description

Bus-input reducing Boolean "NOR" function.

The first port is the **output**, followed by one **input** portbus of arbitrary width.

```
Y = NOR(A[0], ⋯, A[n-1])
```

Ports

No	Name	Type
0	Y	output
1	A	{input[n]}

REDUCE_XOR

Description

Bus-input reducing Boolean "XOR" function.

The first port is the **output**, followed by one **input** portbus of arbitrary width.

```
Y = XOR(A[0], ⋯, A[n-1])
```

Ports

No	Name	Type
0	Y	output
1	A	{input[n]}

REDUCE_XNOR

Description

Bus-input reducing Boolean "XNOR" function.

The first port is the **output**, followed by one **input** portbus of arbitrary width.

```
Y = XNOR(A[0], ⋯, A[n-1])
```

Ports

No	Name	Type
0	Y	output
1	A	{input[n]}

Wide Functions (n to n)

WIDE_AND

Description

Wide Boolean "AND" function.

The first portbus is the **output** bus, it is followed by an arbitrary number of **input** portbuses.

All buses must have the same width.

The input buses may be bubbled, indicating a negation of the complete input bus.

```
Y[0] = AND(A[0], B[0], ...)
Y[1] = AND(A[1], B[1], ...)
...
Y[n-1] = AND(A[n-1], B[n-1], ...)
```

Ports

No	Name	Type
0	Y	{output[n]}
1	A	{input[n]}
2	B	{input[n]}
...	...	{input[n]}

WIDE_NAND

Description

Wide Boolean "NAND" function.

The first portbus is the **output** bus, it is followed by an arbitrary number of **input** portbuses.

All buses must have the same width.

The input buses may be bubbled, indicating a negation of the complete input bus.

```
Y[0] = NAND(A[0], B[0], ...)
Y[1] = NAND(A[1], B[1], ...)
...
Y[n-1] = NAND(A[n-1], B[n-1], ...)
```

Ports

No	Name	Type
0	Y	{output[n]}
1	A	{input[n]}
2	B	{input[n]}
...	...	{input[n]}

WIDE_OR

Description

Wide Boolean "OR" function.

The first portbus is the **output** bus, it is followed by an arbitrary number of **input** portbuses.

All buses must have the same width.

The input buses may be bubbled, indicating a negation of the complete input bus.

```
Y[0] = OR(A[0], B[0], ...)
Y[1] = OR(A[1], B[1], ...)
...
Y[n-1] = OR(A[n-1], B[n-1], ...)
```

Ports

No	Name	Type
0	Y	{output[n]}
1	A	{input[n]}
2	B	{input[n]}
...	...	{input[n]}

WIDE_NOR

Description

Wide Boolean "NOR" function.

The first portbus is the **output** bus, it is followed by an arbitrary number of **input** portbuses.

All buses must have the same width.

The input buses may be bubbled, indicating a negation of the complete input bus.

```
Y[0] = NOR(A[0], B[0], ...)
Y[1] = NOR(A[1], B[1], ...)
...
Y[n-1] = NOR(A[n-1], B[n-1], ...)
```

Ports

No	Name	Type
0	Y	{output[n]}
1	A	{input[n]}
2	B	{input[n]}
...	...	{input[n]}

WIDE_XOR

Description

Wide Boolean "XOR" function.

The first portbus is the **output** bus, it is followed by an arbitrary number of **input** portbuses.

All buses must have the same width.

The input buses may be bubbled, indicating a negation of the complete input bus.

```
Y[0] = XOR(A[0], B[0], ...)
Y[1] = XOR(A[1], B[1], ...)
...
Y[n-1] = XOR(A[n-1], B[n-1], ...)
```

Ports

No	Name	Type
0	Y	{output[n]}
1	A	{input[n]}
2	B	{input[n]}
...	...	{input[n]}

WIDE_XNOR

Description

Wide Boolean "XNOR" function.

The first portbus is the **output** bus, it is followed by an arbitrary number of **input** portbuses.

All buses must have the same width.

The input buses may be bubbled, indicating a negation of the complete input bus.

```
Y[0] = XNOR(A[0], B[0], ...)
Y[1] = XNOR(A[1], B[1], ...)
...
Y[n-1] = XNOR(A[n-1], B[n-1], ...)
```

Ports

No	Name	Type
0	Y	{output[n]}
1	A	{input[n]}
2	B	{input[n]}
...	...	{input[n]}

WIDE_BUF

Description

Wide buffer.

The first portbus is the **output** bus, it is followed by an **input** portbus.

The buses must have the same width.

Ports

No	Name	Type
0	Y	{output[n]}
1	A	{input[n]}

WIDE_INV

Description

Wide Boolean inverter.

The first portbus is the **output** bus, it is followed by an **input** portbus.

The buses must have the same width.

```
Y[0] = INV(A[0])
Y[1] = INV(A[1])
...
Y[n-1] = INV(A[n-1])
```

Ports

No	Name	Type
0	Y	{output[n]}
1	A	{input[n]}

WIDE_TRI

Description

Wide tri-state buffer.

The first portbus is the **output** bus, it is followed by an **input** portbus and a single **control** port.

The buses must have the same width.

Ports

No	Name	Type
0	Y	{output[n]}
1	A	{input[n]}
2	Ctrl	input

Mixed Functions

WIDE_NTO1MUX

Description

Multi-input multiplexer.

The first portbus is the **output** bus, it is followed by an arbitrary number of **input** portbuses `D0` ... `Dx`, and a **select** input bus `Sel`.

All buses except `Sel` must have the same width.

`Sel` selects which input bus is switched to the output:

- If `Sel` is `0`, `D0` is switched to the output,
- if `Sel` is `1`, `D1` is switched to the output,
- ...
- if `Sel` is `x`, `Dx` is switched to the output.

Ports

No	Name	Type
0	Y	{output[n]}
1	D0	{input[n]}
2	D1	{input[n]}
...	...	...
(x+1)	Dx	{input[n]}
(x+2)	Sel	{input[m]}

WIDE_MUX

Description

2-input bus multiplexer.

The first portbus is the **output** bus, it is followed by two **input** buses `A` and `B`, and a **select** input `Sel`.

All buses must have the same width.

`Sel` selects which input is switched to the output:

- If `Sel` is `0`, `A` is switched to the output,
- if `Sel` is `1`, `B` is switched to the output.

Ports

No	Name	Type
0	Y	{output[n]}
1	A	{input[n]}
2	B	{input[n]}
3	Sel	input

ADDER

Description

Multi-bit full adder with carry-in and carry-out.

The first portbus is the **output** bus, it is followed by the **carry-out** port; next are the two adder **input** buses, followed by the **carry-in** port.

The buses can have pairwise different widths.

Ports

No	Name	Type
0	Y	{output[o]}
1	Co	output
2	A	{input[m]}
3	B	{input[n]}
4	Ci	input

MULTIPLIER

Description

Arithmetic "MULTIPLIER" function.

The first portbus is the **output** bus, next are the two operand **input** buses.

The buses can have pairwise different widths.

Ports

No	Name	Type
0	Y	{output[o]}
1	A	{input[m]}
2	B	{input[n]}

POW

Description

Arithmetic "POW" function.

The first portbus is the **output** bus, next are the two operand **input** buses.

The buses can have pairwise different widths.

Ports

No	Name	Type
0	Y	{output[o]}
1	A	{input[m]}
2	B	{input[n]}

DIVIDER

Description

Arithmetic "DIVIDER" function.

The first portbus is the **output** bus, next are the two operand **input** buses.

The buses can have pairwise different widths.

Ports

No	Name	Type
0	Y	{output[o]}
1	A	{input[m]}
2	B	{input[n]}

MODULO

Description

Arithmetic "MODULO" function.

The first portbus is the **output** bus, next are the two operand **input** buses.

The buses can have pairwise different widths.

Ports

No	Name	Type
0	Y	{output[o]}
1	A	{input[m]}
2	B	{input[n]}

REMAINDER

Description

Arithmetic "REMAINDER" function.

The first portbus is the **output** bus, next are the two operand **input** buses.

The buses can have pairwise different widths.

Ports

No	Name	Type
0	Y	{output[o]}
1	A	{input[m]}
2	B	{input[n]}

SHIFT_LEFT

Description

Bitwise left shift.

The first portbus is the **output** bus, it is followed by the **input** bus, the **shift amount** input bus S, and a **carry-in** port.

All portbuses except S must have the same width.

Ports

No	Name	Type
0	Y	{output[n]}
1	I	{input[n]}
2	S	{input[m]}
3	Ci	input

SHIFT_RIGHT

Description

Bitwise right shift.

The first portbus is the **output** bus, it is followed by the **input** bus, the **shift amount** input bus S, and a **carry-in** port.

All portbuses except S must have the same width.

Ports

No	Name	Type
0	Y	{output[n]}
1	I	{input[n]}
2	S	{input[m]}
3	Ci	input

ROTATE_LEFT

Description

Bitwise left rotation.

The first portbus is the **output** bus, it is followed by the **input** bus, and the **rotate amount** input bus S.

All portbuses except S must have the same width.

Ports

No	Name	Type
0	Y	{output[n]}
1	I	{input[n]}
2	S	{input[m]}

ROTATE_RIGHT

Description

Bitwise right rotation.

The first portbus is the **output** bus, it is followed by the **input** bus, and the **rotate amount** input bus S.

All portbuses except S must have the same width.

Ports

No	Name	Type
0	Y	{output[n]}
1	I	{input[n]}
2	S	{input[m]}

LESSTHAN

Description

The arithmetic "less than" or "less or equal" operation.

The first port is the result output, followed by two input portbuses, and a carry-in.

Both portbuses may have different widths.

The carry-in selects the actual operation:

- carry-in = 0 ⇒ "less than"
- carry-in = 1 ⇒ "less or equal"

The result output can be bubbled.

Ports

No	Name	Type
0	O	output
1	A	{input[n]}
2	B	{input[m]}
3	Cin	input

NTO1MUX

Description

A portbus-input multiplexer with single bit output.

The first port is the output, the following two portbuses are the input portbus `D`, and the selection input `Sel`.

`Sel` selects which bit of `D` is switched to the output:

- if `Sel` is 0, `D[0]` is switched to the output,
- if `Sel` is 1, `D[1]` is switched to the output,
- ...
- if `Sel` is n-1, `D[n-1]` is switched to the output.

The portbuses may have different widths.

Ports

No	Name	Type
0	Y	output

No	Name	Type
1	D	{input[n]}
2	Sel	{input[m]}

SELECTOR

Description

A "selector" with single bit output, a portbus input `A`, and a selection input portbus `Sel`.

Both portbuses must have the same width.

Ports

No	Name	Type
0	Y	output
1	A	{input[n]}
2	Sel	{input[n]}

DECODER

Description

A "decoder".

The first portbus is the output, followed by an input portbus, and an optional control port.

The portbuses may have different widths.

If the control port is given, the decoder is actually an "enabled decoder", i.e. its output signals are "AND-ed" with the control signal.

Ports

No	Name	Type
0	Y	{output[n]}
1	A	{input[m]}
2	Ctrl	input (optional)

PRIO_SELECTOR

Description

A "priority selector".

The first port is the a single bit output, followed by an input portbus, a selection input portbus `Sel`, and a carry-in port.

The carry-in port is used as the "else value".

The portbuses must have the same width.

Ports

No	Name	Type
0	Y	output
1	A	{input[n]}
2	Sel	{input[n]}
3	Cin	input

WIDE_PRIO_SELECTOR

Description

A "wide priority selector".

The carry-in portbus is used as the "else value".

Ports

No	Name	Type
0	Y	{output[m]}
1	A0	{input[m]}
2	A1	{input[m]}
…	…	…
n	AN	{input[m]}
2+n	Sel	{input[n]}
3+n	Cin	{input[m]}

PRIO_ENCODER

Description

A "priority encoder".

Ports

No	Name	Type
0	Y	{output[n+1]}
1	A	{input[n]}

WIDE_CASE_SELECT_BOX

Description

This primitive represents a "case statement".

Ports

No	Name	Type
0	out	{output[m]}
1	sel	{input[s]}
2	sel_val	{input[n*s]}
3	data	{input[n*m]}
4	default	{input[m]}

LUT

Description

A lookup table.

Ports

No	Name	Type
0	Y	output
1	D	{input[n]}

ABS

Description

The arithmetic "ABS" operation.

Ports

No	Name	Type
0	Y	{output[n]}
1	A	{input[n]}

MINUS

Description

The arithmetic "subtraction" operation.

The first portbus is the result output, followed by two input portbuses (the "minuend" and the "subtrahend").

All portbuses must have same width.

Ports

No	Name	Type
0	Y	{output[n]}
1	A	{input[n]}
2	B	{input[n]}

UMINUS

Description

The "unary minus" operation.

The first portbus is the result output, followed by one input portbus.

Both portbuses must have same width.

Ports

No	Name	Type
0	Y	{output[n]}
1	A	{input[n]}

EQUAL

Description

The "equal" operation.

The first port is the result output, followed by two input portbuses of equal width.

Ports

No	Name	Type
0	Y	output
1	A	{input[n]}
2	B	{input[n]}

NEQUAL

Description

The "not equal" operation.

The first port is the result output, followed by two input portbuses of equal width.

Ports

No	Name	Type
0	Y	output
1	A	{input[n]}
2	B	{input[n]}

WIDE_SELECTOR

Description

A bus selector.

Ports

No	Name	Type
0	Y	{output[n]}
1	D0	{input[n]}
…	…	…
1+x	Dx	{input[n]}
2+x	Sel	{input[m]}

RAM Elements

RAM

Description

A generic RAM with unrestricted ports.

Ports

No	Name	Type
…	…	…

DUAL_PORT_RAM

Description

A "dual port RAM";

The data ports (`Y` and `D`) must have equal widths, and the address ports (`Wadr` and `Radr`) must have equal widths.

Ports

No	Name	Type
0	Y	{output[n]}
1	D	{input[n]}
2	Wadr	{input[m]}
3	Radr	{input[m]}
4	Wen	input

READ_PORT

Description

A "read port" of a multi-port RamNet.

The `RamNet` port is normally connected to a net which interconnects all read/write-ports of the same RAM.

Ports

No	Name	Type
0	D	{output[m]}
1	Adr	{input[n]}
2	RamNet	input
3	Ren	input

WRITE_PORT

Description

A "write port" of a multi-port RamNet.

The `RamNet` port is normally connected to a net which interconnects all read/write-ports of the same RAM.

Ports

No	Name	Type
0	RamNet	output
1	Adr	{input[m]}
2	D	{input[n]}
3	Wen	input

CLOCKED_WRITE_PORT

Description

A "clocked write port" of a multi-port RamNet.

The `RamNet` port is normally connected to a net which interconnects all read/write-ports of the same RAM.

Ports

No	Name	Type
0	RamNet	output
1	Adr	{input[m]}
2	D	{input[n]}
3	Clk	input
4	Wen	input

Wide State Elements

WIDE_DFF

Description

A wide flip-flop.

Ports

No	Name	Type
0	Y	{output[n]}
1	D	{input[n]}
2	Clk	input
3	Reset	{input[n]} (optional)
4	Set	{input[n]} (optional)

WIDE_DLATCH

Description

A wide dlatch.

Ports

No	Name	Type
0	Y	{output[n]}
1	D	{input[n]}
2	En	input
3	Reset	{input[n]} (optional)
4	Set	{input[n]} (optional)

WIDE_DFFCV

Description

A wide condition/value flip-flop.

Ports

No	Name	Type
0	Y	{output[n]}
1	D	{input[n]}
2	Clk	input
3	Condition	{input[n]}
4	Value	{input[n]}

WIDE_DLATCHCV

Description

A wide condition/value dlatch.

Ports

No	Name	Type
0	Y	{output[n]}
1	D	{input[n]}
2	En	input
3	Condition	{input[n]}
4	Value	{input[n]}

Assertions

Description

The following primitive functions represent SystemVerilog Assertions (SVA), as created by our SystemVerilog front-end. All ports are single-bit.

Assertion Primitives (1 Input)

HDL_ASSERTION

Description

Generic HDL assertion.

Ports

No	Name	Type
0	A	input

SVA_IMMEDIATE_ASSERT

Description

```
assert <expr>
```

Ports

No	Name	Type
0	A	input

SVA_ASSERT

Description

```
assert property <expr>
```

Ports

No	Name	Type
0	A	input

SVA_COVER

Description

```
cover property <expr>
```

Ports

No	Name	Type
0	A	input

SVA_IMMEDIATE_COVER

Description

```
cover  #0 <expr>
```

Ports

No	Name	Type
0	A	input

SVA_ASSUME

Description

```
assume property <expr>
```

Ports

No	Name	Type
0	A	input

SVA_IMMEDIATE_ASSUME

Description

```
assume #0 <expr>
```

Ports

No	Name	Type
0	A	input

SVA_EXPECT

Description

```
expect property <expr>
```

Ports

No	Name	Type
0	A	input

Unary SVA Operators (1 Output, 1 Input)

SVA_POSEDGE

Description

```
posedge <expr>
```

Ports

No	Name	Type
0	Y	output
1	A	input

SVA_NOT

Description

```
not <property_expr>
```

Ports

No	Name	Type
0	Y	output
1	A	input

SVA_FIRST_MATCH

Description

```
first_match <seq_expr>
```

Ports

No	Name	Type
0	Y	output
1	A	input

SVA_ENDED

Description

```
<seq_expr>.ended
```

Ports

No	Name	Type
0	Y	output
1	A	input

SVA_MATCHED

Description

```
<seq_expr>.matched
```

Ports

No	Name	Type
0	Y	output
1	A	input

SVA_CONSECUTIVE_REPEAT

Description

```
<seq_expr> [* <const_or_range_expr>]
```

Ports

No	Name	Type
0	Y	output
1	A	input

SVA_NON_CONSECUTIVE_REPEAT

Description

```
<seq_expr> [*= <const_or_range_expr>]
```

Ports

No	Name	Type
0	Y	output
1	A	input

SVA_GOTO_REPEAT

Description

```
<seq_expr> [*-> <const_or_range_expr>]
```

Ports

No	Name	Type
0	Y	output
1	A	input

SVA_MATCH_ITEM_TRIGGER

Description

```
(<sequence_expr>,<local_var>=<expr>)
```

Ports

No	Name	Type
0	Y	output
1	A	input

SVA_WITHIN_RANGE

Description

```
##[range] <seq_expr>
```

Ports

No	Name	Type
0	Y	output
1	A	input

SVA_STRONG

Description

```
strong (<seq_expr>)
```

Ports

No	Name	Type
0	Y	output
1	A	input

SVA_WEAK

Description

```
weak (<seq_expr>)
```

Ports

No	Name	Type
0	Y	output
1	A	input

SVA_RESTRICT

Description

```
restrict (<property_expr>)
```

Ports

No	Name	Type
0	Y	output
1	A	input

SVA_TRIGGERED

Description

```
<sequence_expr>.triggered
```

Ports

No	Name	Type
0	Y	output
1	A	input

SVA_NEXTTIME

Description

```
nexttime <property_expr>
```

or

```
nexttime [<constant_expr>] <property_expr>
```

Ports

No	Name	Type
0	Y	output
1	A	input

SVA_S_NEXTTIME

Description

```
s_nexttime <property_expr>
```

or

```
s_nexttime [<constant_expr>] <property_expr>
```

Ports

No	Name	Type
0	Y	output
1	A	input

SVA_ALWAYS

Description

```
always <property_expr>
```

or

```
always [<cycle_delay_constant_range_expr>] <property_expr>
```

Ports

No	Name	Type
0	Y	output
1	A	input

SVA_S_ALWAYS

Description

```
s_always [<constant_range>] <property_expr>
```

Ports

No	Name	Type
0	Y	output

No	Name	Type
1	A	input

SVA_S_EVENTUALLY

Description

```
s_eventually <property_expr>
```

or

```
s_eventually [<cycle_delay_constant_range_expr>] <property_expr>
```

Ports

No	Name	Type
0	Y	output
1	A	input

SVA_EVENTUALLY

Description

```
eventually [<constant_range>] <property_expr>
```

Ports

No	Name	Type
0	Y	output
1	A	input

Binary SVA Operators (1 Output, 2 Inputs)

SVA_AND

Description

```
<property_expr> and <property_expr>
```

Ports

No	Name	Type
0	Y	output
1	A	input
2	B	input

SVA_OR

Description

```
<property_expr> or <property_expr>
```

Ports

No	Name	Type
0	Y	output
1	A	input
2	B	input

SVA_SEQ_AND

Description

```
<seq_expr> and <seq_expr>
```

Ports

No	Name	Type
0	Y	output
1	A	input
2	B	input

SVA_SEQ_OR

Description

```
<seq_expr> or <seq_expr>
```

Ports

No	Name	Type
0	Y	output

No	Name	Type
1	A	input
2	B	input

SVA_EVENT_OR

Description

Event 'or' in the argument of a sequence or property, e.g.

```
@(posedge clk) seq1(in1 or in2);
```

Ports

No	Name	Type
0	Y	output
1	A	input
2	B	input

SVA_OVERLAPPED_IMPLICATION

Description

```
<seq_expr> \|-> <seq_expr>
```

Ports

No	Name	Type
0	Y	output
1	A	input
2	B	input

SVA_NON_OVERLAPPED_IMPLICATION

Description

```
<seq_expr> \|=> <seq_expr>
```

Ports

No	Name	Type
0	Y	output

No	Name	Type
1	A	input
2	B	input

SVA_OVERLAPPED_FOLLOWED_BY

Description

```
<seq_expr> #-# <seq_expr>
```

Ports

No	Name	Type
0	Y	output
1	A	input
2	B	input

SVA_NON_OVERLAPPED_FOLLOWED_BY

Description

```
<seq_expr> #=# <seq_expr>
```

Ports

No	Name	Type
0	Y	output
1	A	input
2	B	input

SVA_INTERSECT

Description

```
<seq_expr> intersect <seq_expr>
```

Ports

No	Name	Type
0	Y	output
1	A	input

No	Name	Type
2	B	input

SVA_THROUGHOUT

Description

```
<expr> throughout <seq_expr>
```

Ports

No	Name	Type
0	Y	output
1	A	input
2	B	input

SVA_WITHIN

Description

```
<seq_expr> within <seq_expr>
```

Ports

No	Name	Type
0	Y	output
1	A	input
2	B	input

SVA_AT

Description

```
@(<clock_expr) <seq_expr>
```

Ports

No	Name	Type
0	Y	output
1	A	input
2	B	input

SVA_DISABLE_IFF

Description

```
disable iff (<expr>) <property_expr>
```

Ports

No	Name	Type
0	Y	output
1	A	input
2	B	input

SVA_SAMPLED

Description

```
$sampled(<expr>,<clock_expr>)
```

Ports

No	Name	Type
0	Y	output
1	A	input
2	B	input

SVA_ROSE

Description

```
$rose(<expr>,<clock_expr>)
```

Ports

No	Name	Type
0	Y	output
1	A	input
2	B	input

SVA_FELL

Description

```
$fell(<expr>,<clock_expr>)
```

Ports

No	Name	Type
0	Y	output
1	A	input
2	B	input

SVA_STABLE

Description

```
$stable(<expr>,<clock_expr>)
```

Ports

No	Name	Type
0	Y	output
1	A	input
2	B	input

SVA_PAST

Description

```
$past(<expr>,<no_of_ticks>,<clock_expr>)
```

Ports

No	Name	Type
0	Y	output
1	A	input
2	B	input

SVA_MATCH_ITEM_ASSIGN

Description

```
(<sequence_expr>, <local_var> = <expression>)
```

Ports

No	Name	Type
0	Y	output
1	A	input
2	B	input

SVA_SEQ_CONCAT

Description

```
<seq_expr> ##(cycle_delay_range) <seq_expr>
```

Ports

No	Name	Type
0	Y	output
1	A	input
2	B	input

SVA_UNTIL

Description

```
<property_expr> until <property_expr>
```

Ports

No	Name	Type
0	Y	output
1	A	input
2	B	input

SVA_S_UNTIL

Description

```
<property_expr> s_until <property_expr>
```

Ports

No	Name	Type
0	Y	output
1	A	input
2	B	input

SVA_UNTIL_WITH

Description

```
<property_expr> until_with <property_expr>
```

Ports

No	Name	Type
0	Y	output
1	A	input
2	B	input

SVA_S_UNTIL_WITH

Description

```
<property_expr> s_until_with <property_expr>
```

Ports

No	Name	Type
0	Y	output
1	A	input
2	B	input

SVA_IMPLIES

Description

```
<property_expr> implies <property_expr>
```

Ports

No	Name	Type
0	Y	output
1	A	input
2	B	input

SVA_IFF

Description

```
<property_expr> iff <property_expr>
```

Ports

No	Name	Type
0	Y	output
1	A	input
2	B	input

SVA_ACCEPT_ON

Description

```
accept_on (<expr_or_dist>) <property_expr>
```

Ports

No	Name	Type
0	Y	output
1	A	input
2	B	input

SVA_REJECT_ON

Description

```
reject_on (<expr_or_dist>) <property_expr>
```

Ports

No	Name	Type
0	Y	output

No	Name	Type
1	A	input
2	B	input

SVA_SYNC_ACCEPT_ON

Description

```
sync_accept_on (<expr_or_dist>) <property_expr>
```

Ports

No	Name	Type
0	Y	output
1	A	input
2	B	input

SVA_SYNC_REJECT_ON

Description

```
sync_reject_on (<expr_or_dist>) <property_expr>
```

Ports

No	Name	Type
0	Y	output
1	A	input
2	B	input

Ternary SVA Operators (1 Output, 3 Inputs)

SVA_IF

Description

```
if (expr) <seq_expr> else <seq_expr>
```

Ports

No	Name	Type
0	Y	output

No	Name	Type
1	A	input
2	B	input
3	C	input

Global Clocking

SVA_GLOBAL_CLOCKING_DEF

Description

SVA Global Clocking Definition.

```
global clocking <clk_expr>; endclocking
```

Ports

No	Name	Type
0	A	input

SVA_GLOBAL_CLOCKING_REF

Description

Global clocking reference.

Ports

No	Name	Type
0	Y	output

SVA_WIDE_SAMPLED

Description

```
$sampled(<expr>,<clock_expr>)
```

Ports

No	Name	Type
0	Y	{output[m]}
1	A0	{input[m]}
2	A1	{input[n]}

SVA_WIDE_STABLE

Description

```
$stable(<expr>,<clock_expr>)
```

Ports

No	Name	Type
0	Y	{output[m]}
1	A0	{input[m]}
2	A1	{input[n]}

Spice-Level Functions

Description

The primitive functions below describe transistor devices. The function names are borrowed from Spice devices (with the same meaning). Of course, all ports are single-bit.

On the API-level, a cell's function can be queried using the `primFuncOf` command (`$db primFuncOf $cell`).

NMOS

Description

NMOS transistor.

The first three ports are **drain**, **gate**, and **source** respectively, the fourth port (**bulk**) is optional;

Additional ports may be added (e.g. for substrate).

Example

```
$db load primitive nmos NMOS
$db load port D *
$db load port G *
$db load port S *
$db load port B *
```

Ports

No	Name	Type
0	drain	inout

No	Name	Type
1	gate	input
2	source	inout
3	bulk	input (optional)
…	…	input (optional)

PMOS

Description

A PMOS transistor.

The first three ports are **drain**, **gate**, and **source** respectively, the fourth port (**bulk**) is optional;

Additional ports may be added (e.g. for substrate).

Ports

No	Name	Type
0	drain	inout
1	gate	input
2	source	inout
3	bulk	input (optional)
…	…	input (optional)

NPN

Description

A bipolar NPN transistor.

The first three ports are **collector**, **base**, and **emitter** respectively, the fourth port (**substrate**) is optional;

Additional ports may be added.

Example

```
$db load primitive npn NPN
$db load port C *
$db load port B *
$db load port E *
```

Ports

No	Name	Type
0	collector	inout
1	base	input
2	emitter	inout
3	substrate	input (optional)
…	…	input (optional)

PNP

Description

A bipolar PNP transistor.

The first three ports are **collector**, **base**, and **emitter** respectively, the fourth port (**substrate**) is optional;

Additional ports may be added.

Example

```
$db load primitive pnp PNP
$db load port C *
$db load port B *
$db load port E *
```

Ports

No	Name	Type
0	collector	inout
1	base	input
2	emitter	inout
3	substrate	input (optional)

No	Name	Type
…	…	input (optional)

NMOSM

Description

NMOS transistor with multiple gates.

The first three ports are **drain**, **gate**, and **source** respectively, the other optional ports are gates or bulk;

Additional ports may be added (e.g. for substrate).

Example

```
$db load primitive nmos NMOSM
$db load port D *
$db load port G *
$db load port S *
$db load port G2 * -flag multigate
$db load port B *
```

Ports

No	Name	Type
0	drain	inout
1	gate	input
2	source	inout
3	gate2	input (optional)
…	…	input (optional)

PMOSM

Description

A PMOS transistor with multiple gates.

The first three ports are **drain**, **gate**, and **source** respectively, the other optional ports are gates or bulk; the fourth port (**bulk**) is optional;

Additional ports may be added (e.g. for substrate).

Ports

No	Name	Type
0	drain	inout
1	gate	input
2	source	inout
3	bulk	input (optional)
...	...	input (optional)

NPNM

Description

A bipolar NPN transistor with optional bases.

The first three ports are **collector**, **base**, and **emitter** respectively, the other optional ports are bases or bulk;

Additional ports may be added.

Example

```
$db load primitive npn NPNM
$db load port C *
$db load port B *
$db load port E *
$db load port B2 * -flag multigate
```

Ports

No	Name	Type
0	collector	inout
1	base	input
2	emitter	inout
3	base2	input (optional)
...	...	input (optional)

PNPM

Description

A bipolar PNP transistor with multiple bases.

The first three ports are **collector**, **base**, and **emitter** respectively, the other optional ports are bases or bulk;

Additional ports may be added.

Example

```
$db load primitive pnp PNPM
$db load port C *
$db load port B *
$db load port E *
$db load port B2 * -flag multigate
```

Ports

No	Name	Type
0	collector	inout
1	base	input
2	emitter	inout
3	base2	input (optional)
…	…	input (optional)

RES

Description

A resistor.

Two **inout** (bidirectional) ports, the third port (**bulk**) is optional;

Additional ports may be added (e.g. for substrate).

Example

```
$db load primitive res RES
$db load port a *
$db load port b *
```

Ports

No	Name	Type
0	p1	inout
1	p2	inout
2	bulk	input (optional)
…	…	input (optional)

CAP

Description

A capacitor.

Two **inout** (bidirectional) ports, the third port (**bulk**) is optional;

Additional ports may be added (e.g. for substrate).

Example

```
$db load primitive cap CAP
$db load port a *
$db load port b *
```

Ports

No	Name	Type
0	p1	inout
1	p2	inout
2	bulk	input (optional)
…	…	input (optional)

INDUCTOR

Description

An inductor.

Two **inout** (bidirectional) ports, the third port (**bulk**) is optional;

Additional ports may be added (e.g. for substrate).

Example

```
$db load primitive ind INDUCTOR
$db load port a *
$db load port b *
```

Ports

No	Name	Type
0	p1	inout
1	p2	inout
2	bulk	input (optional)
…	…	input (optional)

DIODE

Description

A diode.

The first port is the **anode**, the second port is the **cathode**, the optional third port is **bulk**;

Additional ports may be added (e.g. for substrate).

Example

```
$db load primitive diode DIODE
$db load port anode *
$db load port catho *
```

Ports

No	Name	Type
0	anode	inout
1	cathode	inout
2	bulk	input (optional)
…	…	input (optional)

ZDIODE

Description

A Zener diode.

The first port is the **anode**, the second port is the **cathode**, the optional third port is **bulk**;

Additional ports may be added (e.g. for substrate).

Example

```
$db load primitive zdiode ZDIODE
$db load port anode *
$db load port catho *
```

Ports

No	Name	Type
0	anode	inout
1	cathode	inout
2	bulk	input (optional)
…	…	input (optional)

SWITCH

Description

A (controlled) switch.

Two **inout** (bidirectional) ports, the optional third and fourth ports are used as **control** inputs.

Example

```
$db load primitive switch SWITCH
$db load port a *
$db load port b *
$db load port c1 *
$db load port c2 *
```

Ports

No	Name	Type
0	p1	inout

No	Name	Type
1	p2	inout
2	Ctrl+	input (optional)
3	Ctrl-	input (optional)

VSOURCE

Description

A voltage source.

The primitive has either two ports (`+`, `-`) or four (`+`, `-`, `ctrl+`, `ctrl-`). If there are only two ports it is a "normal" voltage source, if there are four ports it's a "voltage controlled" voltage source — in this case the additional two ports are the control inputs.

Example

```
$db load primitive vsrc VSOURCE
$db load port + *
$db load port - *
```

Ports

No	Name	Type
0	+	inout
1	-	inout
2	Ctrl+	input (optional)
3	Ctrl-	input (optional)

ISOURCE

Description

A current source.

The primitive has either two ports (`+`, `-`) or four (`+`, `-`, `ctrl+`, `ctrl-`). If there are only two ports it is a "normal" current source, if there are four ports it's a "voltage controlled" current source — in this case the additional two ports are the control inputs.

Example

```
$db load primitive isrc ISOURCE
$db load port + *
$db load port - *
$db load port c+ *
$db load port c- *
```

Ports

No	Name	Type
0	+	inout
1	-	inout
2	Ctrl+	input (optional)
3	Ctrl-	input (optional)

TRANSLINE

Description

A transmission line.

The primitive has `2*n` ports; the first `n` ports are on the one side of the transmission line, the second `n` ports are on the other side.

Example

```
$db load primitive transline TRANSLINE
$db load port a0 *
$db load port a1 *
$db load port a2 *
$db load port b0 *
$db load port b1 *
$db load port b2 *
```

Ports

No	Name	Type
0	in0	inout
1	in1	inout
...	...	...
(n-1)	in(n-1)	inout

No	Name	Type
n	out0	inout
n+1	out1	inout
…	…	…
2*n-1	out(n-1)	inout

UDRCLINE

Description

An UDRC line.

The first two ports are the actual line, the third port is ground.

Example

```
$db load primitive udrcline UDRCLINE
$db load port a *
$db load port b *
$db load port g *
```

Ports

No	Name	Type
0	in	inout
1	out	inout
2	ground	inout

AMP

Description

An operational amplifier.

This primitive comes with 3, 5, or 7 ports.

In the 3-port version, only the positive input (`+`), negative input (`-`), and output (`out`) are present.

In the 5-port version, only the positive input (`+`), negative input (`-`), output (`out`), `vcc`, and `vee` are present.

In the 7-port version, all ports are present — including the two compensation ports `comp1` and `comp2`.

Example

```
$db load primitive amp AMP
$db load port - *
$db load port + *
$db load port out *
$db load port vcc *
$db load port vee *
```

Ports

No	Name	Type
0	-	input
1	+	input
2	out	output
3	comp1	inout (only in the 7-port version)
4	comp2	inout (only in the 7-port version)
5	vcc	input (only in the 5- and 7-port versions)
6	vee	input (only in the 5- and 7-port versions)

JOSEPHSON_JUNCTION

Description

A Josephson Junction.

This primitive requires 2 ports.

Example

```
$db load primitive jj JOSEPHSON_JUNCTION
$db load port n1 *
$db load port n2 *
```

Ports

No	Name	Type
0	n1	inout
1	n2	inout

UNKNOWNDEV

Description

An unknown device.

This primitive is expected to have at least two inout (bidirectional) ports.

Additional input ports may be added (e.g. for substrate).

Ports

No	Name	Type
0	p1	inout
1	p2	inout
...	...	input (optional)

The Flat View API

Introduction

This Document describes the "Flat View" extension to the Database API and provides the commands listed below.

A Flat View is an instance-tree-based data structure linked to a top module. That data structure is created implicitly when needed, but can be compressed or freed explicitly. Each flat view can store flat attributes, flat flags and highlight colors. In addition it can be used for flat traversal.

Index

- Getting Flat Instances
- Deal with Flat Attributes
- Deal with Flat Ids
- Deal with Flat Flags
- Deal with Flat Highlight colors
- Compare Flag, Attr, and Highlight commands
- Out of module reference commands

Overview

Here is a reference list of the flat view commands:

- $db flat compress ?$top?
- $db flat free ?$top?
- $db flat signalOf ?-autoPopulate? $net
- $db flatattr $oid add|set|delete ?$attr ...?
- $db flatattr $top deleteAll ?$attr?
- $db flatattr $oid get|getMerged
- $db flatattr $oid getValue|getMergedValue $name
- $db flatid $oid set|get $name
- $db flatflag $oid set|clear|is $name
- $db flatflag $oid get
- $db flatflag $oid getOIDs
- $db flathilight $oid set $col
- $db flathilight $oid get|getMerged
- $db flathilight $oid delete
- $db flathilight $top deleteAll ?$col?
- $db flat foreach signal ?-autoPopulate? $top cur $body (Examine all signals)
- $db flat foreach net|pin ?-autoPopulate? $net cur $body (Examine all nets/pins of a signal)
- $db flat foreach netseg ?-autoPopulate? $pin1 $pin2 cur $body (Examine all netsegment between two pins)
- $db flat foreach instOfCell $flag ?-autoPopulate? $mod cur $body (Examine all instances of flagged cells)
- $db flat foreach instOfMod ?-autoPopulate? cur $body (Examine all instances of modules)
- $db flat foreach instOfModOnce ?-autoPopulate? cur $body (Examine all instances of modules (each module only once).)
- $db flat foreach instOfCellOnce ?-autoPopulate? cur $body (Examine all instances of cells (each cell only once).)

- $db flat foreach hilight $top oid col $body (Examine all flat objects which have hilight set)
- $db flat foreach obj ?-error? ?-autoPopulate? $top oid $body (Examine objects in flat tree which have attr, flag or hilight set)
- $db flatoomr add $pin $net (Add out of module references)
- $db flatoomr del $pin (Delete out of module references)
- $db flatoomr get $oid (Get out of module references)
- $db flatoomr foreach $top oid1 oid2 (Loop over out of module references)

The Flat View

The Flat View data is addressed by (usually tree-based) OIDs. Each OID's root module must refer to a top module; more precisely, isTopRoot must return true for the OID - if not, then the OID cannot address a Flat View data structure and the commands will return an error.

The Flat View provides a "flat" view to a certain design tree; it is linked to a top module in the database. The Flat View data structure is created automatically when needed.

```
$db flat free      ?$top?
$db flat compress  ?$top?
$db flat foreach hilight           ?-autoPopulate? $top oid color $body
$db flat foreach flagged $flagname ?-autoPopulate? $top oid       $body
$db flat foreach attr    $attrname ?-autoPopulate? $top oid value $body
$db flat foreach obj ?-error?      ?-autoPopulate? $top oid       $body
```

The `free` command deletes the Flat View at the given top or all Flat Views if $top is not specified. If a database is closed by $db close then all Flat Views are automatically freed.

The `compress` command frees unneeded data nodes from the Flat View data structure (saves memory space). If a $top is specified, then only the Flat View at the given top module is compressed (else all Flat Views are compressed).

The `foreach hilight` command loops over the flat view and executes the body for each object which has the hilight info set. A variable with the name of color is set with the current hilight value.

The `foreach flagged` command loops over the flat view and executes the body for each object which has a flat attribute with the given flagname set.

The `foreach attr` command loops over the flat view and executes the body for each object which has a attribute with the given attrname set. A variable with the name of value is set with the current attribute value.

The `foreach obj` command loops over the flat view and executes the body for each object which has attribute, flags or hilight info set. If a object in flat view does not match the database it is skipped. With the '-error' option set, unmatched objects generate an error.

The **$top**, if specified, must be an OID of a top module (it must not be tree-based and isTopRoot must be true). The database must not be modified while one or more Flat Views exists (or the Flat

View data get invalid).

Getting Flat Connectivity

Here, the term "**signal**" is used to describe a set of interconnected nets (in the design hierarchy tree rooted at a top module). Technically, signal-OIDs are identical to the net-OIDs, but the different type name identifies that it is the top-most net of the interconnected nets. Most of the API functions that accept a net OID also accept a signal OID with the same meaning as the net OID.

In contrast to the $db functions, the flat functions are not local to a certain module, but traverse all instance paths below the top module (in other words: in the Flat View of the top module). The **top module** - and so the Flat View - is identified by the OID's root module, that means, isTopRoot must be true for $top and $net below.

```
$db flat foreach signal ?-autoPopulate? ?-skipGlobal? $top cur $body
$db flat foreach net    ?-autoPopulate? $signal cur $body
$db flat foreach pin    ?-autoPopulate? ?-addHier? ?-stopHier? ?-skipConst? $net cur
$body
$db flat foreach netseg ?-autoPopulate? $pin1 $pin2 cur $body
$db flat signalOf       ?-autoPopulate? $net
```

The 1st command loops over all signals in the flat-view (`cur` is set to the signal-OID of each signal in the design tree). If the optional argument -addHier is specified global nets are skipped. The 2nd command loops over all interconnected nets (of the given net or signal OID) and the 3rd command loops over all pins (and top-level ports) of all interconnected nets (identified by the given net or signal OID). If the net has the 'global' flag set, all nets with the same name in all hierarchies are visited. Keep in mind that this may be a large number for big designs. If the optional argument -addHier is specified then all intermediate hierarchical pins and ports are processed. If the optional argument -stopHier is specified then hierarchical pins which connect to nothing down the hierarchy are processed in addition. If the optional argument -skipConst is specified then connected constant nets are skipped. The 4th command "'foreach netseg'" executes $body for each netsegment, which lies on the path between $pin1 and $pin2. If there is no signal between $pin1 and $pin2 an error is generated. The 5th command "'signalOf'" returns the signal-OID that is the top-most net connected to $net (the top-most of all candidates that "'foreach net'" would traverse). For nets which have the 'global' flag set a net with the same name is searched in the top module.

Getting Flat Count

To retrieve the number of loops the above commands will use, the following helpers can be used:

```
set count [$db flat count signal ?-autoPopulate? $top]
set count [$db flat count net    ?-autoPopulate? $signal]
set count [$db flat count pin ?-autoPopulate? ?-addHier? ?-stopHier? $net]
set count [$db flat count netseg ?-autoPopulate? $pin1 $pin2]
```

Getting Flat Instances

The following commands exceptionally do not use the Flat View data structure:

```
$db flat foreach instOfCell     $flag ?-autoPopulate? $mod cur $body
$db flat foreach instOfCellOnce $flag ?-autoPopulate? $mod cur $body
$db flat foreach instOfMod            ?-autoPopulate? $mod cur $body
$db flat foreach instOfModOnce        ?-autoPopulate? $mod cur $body
```

The first command traverses the hierarchy tree, rooted at $mod (module OID, usually the top module) and loops over each instance of flagged Primitives or Modules (flagged with the given $flag. In each iteration, "cur" is set to an instance OID. The instance path of the "cur" OID starts with the path given to "$mod" and gets longer while the command dives down the hierarchy tree. If `$flag`is an empty string the command loops over all instances including all module instances. The second command traverses the hierarchy tree like the first command, but only one instance of each referenced cell is visited. The third command traverses the hierarchy tree rooted at $mod and loops over each instance of all modules. The fourth command traverses the hierarchy tree like the previous command, but only one instance of each referenced module is visited.

Deal with Flat Attributes

In addition to the module-based attributes, the database also supports "flat" attributes that are stored in an instance-tree-based data structure called Flat View. The Flat View is addressed by the $flag OID's root module as described above.

```
$db flatattr $oid add $attr ?$attr $attr ...?
$db flatattr $oid set $attr ?$attr $attr ...?
$db flatattr $oid delete    ?$name ...?
$db flatattr $oid get ?-sorted?
$db flatattr $oid getMerged
$db flatattr $oid getValue       $name
$db flatattr $oid getMergedValue $name
$db flatattr $top deleteAll     ?$name ...?
```

Both, `add` and `set` append one or more attributes to the given OID - however, `set` overwrites existing attributes (avoids duplicates) and `add` simply appends. The format of `$attr`is "**name=value**".

`delete` removes all or the named attributes from the given OID.

`get` returns a list of all attributes (list of `name=value`` elements). If the optional switch `-sorted` is given then a sorted list of all attributes is returned.

`getValue` returns just the attribute value (or an empty string if not found).

`getMerged` and `getMergedValue` have the same functionality as "get" and "getValue" but additionally merge the result with the module-based object attributes (module-based attributes have lower priority than flat attributes).

The flat-view's attributes resist in the database and therefore also in the binfile. The number of attributes per OID is currently limited to 32.

With the `deleteAll` command, all flat attributes are removed. The '$name' arguments specify the attributes which should be removed. It may be omitted if all attributes should be removed. The "'$top'" is "*" or a module-based module OID; if "*" is used, then all trees are scanned otherwise only flat objects within the tree below the given '$top'. isTopRoot must return true for all OIDs.

Deal with Flat Flags

In addition to the module-based flags, the database also supports "flat" flags that are stored in an instance-tree-based data structure called Flat View. The Flat View is addressed by the OID's root module as described above.

```
$db flatflag $oid set      name
$db flatflag $oid clear    name
$db flatflag $oid is       name
$db flatflag $oid is_set   name
$db flatflag $oid is_clear name
$db flatflag $oid get
$db flatflag $top clearall name
$db flatflag $top getOIDs  name
```

`set` or `clear` a flag at the specified object in the tree-based data structure. The `is` command returns a Boolean, `is_set` and `is_clear` do both, return the old flag (like `is`) and afterwards set or clear that flag. The `get` command returns a list of all set flags at the given object. The `clearall` command clears the given flag in the complete tree-based data structure, rooted at the given $oid's top module. The general purpose flag name must be one of "**orange**", "**cyan**", "**black**", "**white**", "**magenta**" and "**target**".

Deal with Flat Ids

The database also supports "flat" ids that are stored in an instance-tree-based data structure called Flat View. The Flat View is addressed by the $flag OID's root module as described above.

```
$db flatid $oid set $id
$db flatid $oid get
```

`set` the given numeric $id to the given OID. `get` returns the numeric id for the given OID. If there is none set `0` is returned. Currently only net, netBus or signal oids are supported.

Deal with Highlight Colors

The following commands can be used to highlight individual objects only in one specific instantiation path. An object inside a module that is instantiated multiple times will be highlighted with the given color only in the specified instantiation path of this module. The `highlight` API can be used to highlight objects in each instance of a module.

```
$db flathilight $oid set $col  ?-permanent?
$db flathilight $oid get       ?-permanent|-both?
$db flathilight $oid getMerged ?-permanent|-both?
$db flathilight $oid delete    ?-permanent|-both?
$db flathilight $top deleteAll ?$col? ?-permanent|-both?
```

The `set` command set the flat highlight color for the object specified by $oid to $col. The `get` command returns the current flat highlight color or an empty list ("{}") if nothing set. The `getMerged` command returns the flat highlight color, if not set it looks for the module based highlight color and if none found an empty list ("{}") is returned. The `delete` command removes the highlight color from the object specified by $oid. With the `deleteAll` command, all highlights are removed. The '$col' argument specifies the color of the objects which should be removed. It may be omitted if all highlights should be removed. The "'$top'" is "*" or a module-based module OID; if "*" is used, then all trees are scanned otherwise only flat objects within the tree below the given '$top'. isTopRoot must return true for all OIDs.

A flathilight on a signal OID is supported but only stores the highlight information for the signal. It does not show the highlight color in the GUI. To do so the same color needs to be set at each net of the signal.

There are two sets of highlight colors for each object. 'normal' and '-permanent'. They can be set and deleted independent of each other. If both are set the 'normal' color has priority over the permanent color. '$col' may be in the range 0..48.

Using the option -both to get either the normal or permanent highlight color will return the permanent value as a negative number to be able to distinguish between normal highlight.

Compare Flag, Attr, and Highlight Commands

flatattr	flatflag	flathilight	attr	flag	hilight	description (write access)
`set`	`set` `clear`	`set`	`set`	`set` `clear`	`set`	store the given value to the object (overwrite)
`add`			`add`			store the given value to the object (duplicates are ok)
`delete`		`delete`	`delete`		`delete`	remove entry from the object
`deleteAll`		`deleteAll`			`deleteAll`	remove entry from multiple objects ("*" supported)
	`clearall`					store value at multiple objects.

flatattr	flatflag	flathilight	attr	flag	hilight	description (write access)
	is_set is_clear			is_set is_clear		return value from the object and then modify it
						description (read access)
get getMerged`	get		get foreach	get		return all entries at the object
getValue getMergedValue	is	get getMerged	getValue	is	get	return value from the object
				supported		return true if the object supports the given flag

Out Of Module Reference Commands

The following command can be used to add, get and loop over out of module references (OOMR).

```
$db flatoomr add $pin $net
$db flatoomr get $pin
$db flatoomr get $net
$db flatoomr foreach $top oid1 oid2 {}
```

Adding OOMR connectivity to a pin which has already a net connected in the module based world results in a warning. It is not fully supported and will be treated like disconnected in all hierarchies except the one defined by the OOMR commands.

To avoid these limitations (and the warning) use a singlized design (e.g. $db oper singlize) and disconnect the pins before adding OOMR connectivity.

Additionally, most of the operators are not aware of the OOMR connectivity and should be avoided.

The Spos API

Introduction

This Document describes the "Source Position API" - this is an extension to the Database API and bases on the zdb sub-command $db spos.

Spos data is usually created by a parser, but it can also be created through the API with load -spos, spos add and spos addline.

Overview

The spos data defines source file positions for database objects (if the parser provides them). Each database object (e.g. a net, instance, pin, etc) has zero or more associated spos data entries. Each spos entry consist of a filename and a begin and end file position. Both ways are supported, (a) finding the file positions of a database object as well as (b) finding the object at a certain file position. Most commands which use a file name string to identify a specific file can use an index prefixed by `-idx`. Command which return a file name or set a file name string value have a `-idx` option to set index values instead.

The commands are:

- `$db load -spos $file $begin $end`
- `$db spos pick ?-acceptCallback cmd? (-idx $idx)|$fname $filepos`
- `$db spos picklist ?-limit $limit? (-idx $idx)|$fname $filepos`
- `$db spos foreach $oid ?-idx? fname begin end $body`
- `$db spos foreach -all|-grey|.. (-idx $idx)|$fname oid begin end $body`
- `$db spos foreachfiltered (-idx $idx)|$fname $first $last $oidList begin end $body`
- `$db spos foreachrange ?-acceptCallback cmd? ?-uniq $cnt? ?-column $b $e? ?-idx? $fname $first $last oid begin end $body`
- `$db spos foreachfile ?-idx? fname modtime $body`
- `$db spos exists $fname`
- `$db spos mtime (-idx $idx)|$fname`
- `$db spos maxcolumn (-idx $idx)|$fname`
- `$db spos filetype (-idx $idx)|$fname`
- `$db spos neighbor $oid (-idx $idx)|$fname $pos prevFname prevBegin prevEnd nextFname nextBegin nextEnd`
- `$db spos add ($oid|-grey|-comment) $file $begin $end`
- `$db spos addfile ?-type $t? $file`
- `$db spos addline $file $lineno ?$filepos?`
- `$db spos lineno ?-no|-beg|-end? (-idx $idx)|$fname $filepos`
- `$db spos filepos ?-beg|-end? (-idx $idx)|$fname $lineno`
- `$db spos delete $oid`
- `$db spos clone $src $dest ?$limit?`
- `$db spos fnamebase ?$fname?`
- `$db spos setfname (-idx $idx)|$fname ?-short? $newfname`
- `$db spos getfname (-idx $idx)|$fname ?-short?`

The `$oid` and `$oidList` arguments refer to database objects (OIDs).

The `$file` and `fname` arguments refer to file names.

The `modtime` argument refers to the file modification time stamp.

The `$lineno` argument refers to the line number within a file — starting at 1.

The `$begin`, `$end`, `$filepos`, `$first` and `$last` as well as `begin` and `end` arguments refer to byte positions within a file (seek position) — starting at 0. The `$begin`, `$first` and `begin` point to the first byte of a range, but `$end`, `$last` and `end` point to one byte after the range (i.e. `end - begin` is the size of a range).

Line Numbers

The spos data additionally maintains a mapping from line numbers to byte positions and vice versa. Line numbers start at 1, byte positions start at 0.

spos add

```
$db spos add $oid (-idx $idx)|$fname $begin $end
```

Appends a new spos entry to the given object.

spos addfile

```
$db spos addfile ?-type $t? $file
```

Creates a new file entry with the filename `$file`. Returns a unique index which can be used in `-sposx` options of the `$db load ⋯` commands.

spos addline

```
$db spos addline $file $lineno ?$filepos?
```

Creates a new line-number entry that maps the given `$lineno` to the given file byte position `$filepos` — that must be the end-of-line position, in other words, `$filepos` must point to a newline (`\n`) character. The file `$file` must already exist; the `$lineno` should be increased for each call. If it is not in sequence or some line numbers are skipped, the missing lines are calculated internally on demand. If `$lineno`'s value is `all` the whole file is scanned for end-of-line positions. The size of the file should be stored by using `end` as `$lineno`'s value.

spos lineno

```
$db spos lineno ?-no|-beg|-end? (-idx $idx)|$fname $filepos
```

Returns a triplet with **line number**, the **begin position** and the **end position** of the line. If `$filepos` is out of range it generates an error. The begin file position is the first character in the line. The end file position is one character after the end of the line — that is the first character of the next line. If

`$filepos` is `end` the triplet of the last line is returned. If one of the convenience options is used, only one value instead of a triplet is returned.

Here are some examples for different files:

command	returns	comment
`$db spos lineno file 0`	error	for an empty file
	1 0 1	for a file containing only "\n"
	1 0 1	for a file containing only "\n\n\n"
	1 0 3	for a file containing only "AB\n\n\n"
`$db spos lineno file 2`	1 0 3	for a file containing only "AB\n\n\n"
`$db spos lineno file 3`	2 3 4	for a file containing only "AB\n\n\n"
`$db spos lineno file 4`	3 4 5	for a file containing only "AB\n\n\n"
`$db spos lineno file end`	3 4 5	for a file containing only "AB\n\n\n"
`$db spos lineno file 5`	3 4 5	for a file containing only "AB\n\n\n"
`$db spos lineno file 6`	error	for a file containing only "AB\n\n\n"

spos filepos

```
$db spos filepos ?-beg|-end? (-idx $idx)|$fname $lineno
```

Returns a pair with the **begin position** and the **end position** of the given line. If `$lineno` is out of range it generates an error. If `$lineno` is `end` the file position of the last line is returned. The begin file position is the first character in the line. The end file position is one character after the end of the line — that is the first character of the next line. If one of the convenience options is used, only one value instead of a pair is returned.

Here are some examples for different files:

command	returns	comment
`$db spos filepos file 1`	error	for an empty file
	0 1	for a file containing only "\n\n\n"

Object Positions

The spos data maintains a mapping from object-ids to source file positions (defined by **filename**, **begin** and **end**) and vice versa. The file positions "**begin**" and "**end**" are byte positions in the file (number of bytes from the begin of the file).

spos pick

```
$db spos pick ?-acceptCallback $cmd? (-idx $idx)|$fname $filepos
```

This command results the OID of the object that maps to the source file range at the given `$filepos` (`begin` <= `$filepos` < `end`). If multiple objects' source file position ranges match with the given `$filepos`, then the smallest range (smallest **end**-**begin**) has priority and is returned. If no object matches, then an empty string is returned. If `-acceptCallback` is given, for each result candidate the candidate's cell name is appended to `$cmd` before evaluating it. If it evaluates to false the candidate is skipped.

spos picklist

```
$db spos picklist ?-limit $limit? (-idx $idx)|$fname $filepos
```

This command returns a list of OIDs of the object that map to the source file range at the given `$filepos` (`begin` <= `$filepos` < `end`). The order of the result list is random, except the first element is the one with the smallest range (the same returned by pick). If no object matches, then an empty list is returned.

If `-limit` is used, the length of the returned list is limited to the given number of elements.

spos foreach

```
$db spos foreach $oid ?-idx? fname begin end $body
$db spos foreach -all|-grey|... (-idx $idx)|$fname oid begin end $body
```

The first command loops over all source positions of a given `$oid` (for some objects like ports or instances there are usually only one source position per object, but in general, objects can define multiple source positions). The second command loop over source positions which are not connected to an object. The `$body` is executed for each source position at 'OID'; the three variables (specified by their names `fname`, `begin` and `end`) are pre-set before each loop iteration. These loop variables define each source position range.

Here is a trivial "dump" example:

```
$db spos foreach $oid fname begin end {
    puts "$fname $begin $end"
}
$db spos foreach -all -idx $idx oid begin end {
    puts "[$db oid nulltype $oid] $begin $end"
}
```

spos foreachfiltered

```
$db spos foreachfiltered (-idx $idx)|$fname \
        $first $last $oidList begin end $body
```

This command is an optimized version of `$db spos foreach` and can be used if only the source positions in a given file range is needed. Also, this command accepts a list of objects `$oidList` instead of a single object. The `$body` is executed for each source position found, for each OID of `$oidList`, which match the given `$fname` and lie inside the range `$first` to `$last`. To match the whole file, you can call `$db spos foreachfiltered` with `first` = `0` and `last` = `end`.

Here is a trivial "dump" example:

```
$db spos foreachfiltered "design.sp" 0 end $oidList b e {
    puts "design.sp $b $e"
}
```

spos foreachrange

```
$db spos foreachrange \
        ?-acceptCallback $cmd? \
        ?-uniq $cnt? ?-sort? ?-column $b $e? (-idx $idx)|$fname \
        $first $last oid begin end $body
```

This command is a version of `$db spos foreach` and can be used if only the source positions in a given file range are needed. The `$body` is executed for each source position found inside the range `$first` to `$last` regardless to which OID it belongs. To match the whole file, you can call `$db spos foreachrange` with `first` = `0` and `last` = `end`.

Here is a trivial "dump" example:

```
$db spos foreachrange "design.sp" 0 end o b e {
    puts "design.sp $o $b $e"
}
```

The `-uniq $cnt` option suppresses OIDs which have the same type and lie on the same begin and end position as previous OIDs. Instance OIDs are suppressed if there are more than `$cnt` occurrences. The `-column` option suppresses OIDs which are not in the given column range.

The `-acceptCallback $cmd` is called for each OID candidate with appended cell name. If the callback return false the candidate is skipped.

spos foreachfile

```
$db spos foreachfile ?-idx? fname modtime $body
```

This command loops over all known source files. The `$body` is executed for each file and the two loop variables `fname` and `modtime` are pre-set before each loop iteration (defining the file name (or index) and the modification time). The file's modification time is registered at parsing time (when the spos data is created).

Here is a trivial "dump" example:

```
$db spos foreachfile fname mod {
    puts "$fname $mod"
}
```

spos exists

```
$db spos exists $fname
```

Check if the given file exists in the spos database.

spos mtime

```
$db spos mtime (-idx $idx)|$fname
```

This command returns the modification time of the given filename at parsing time (when the spos data is created).

spos maxcolumn

```
$db spos maxcolumn (-idx $idx)|$fname
```

This command returns the length of the longest line of the given file.

spos filetype

```
$db spos filetype (-idx $idx)|$fname
```

This command returns the file type of the given file.

spos neighbor

```
$db spos neighbor $oid (-idx $idx)|$fname $pos \
        prevFname prevBegin prevEnd \
        nextFname nextBegin nextEnd
```

This command fills the given variables with the source positions before and after the source position defined by `$fname` and `$pos`. If there are no more source positions in front or after the given `$pos`, `prevFname` / `nextFname` are set to `{}`.

Miscellaneous Commands

spos delete

```
$db spos delete $oid
```

This command deletes all spos entries for the given object. If there are no more spos entries for a file the whole file is removed from the database.

spos clone

```
$db spos clone $src $dest ?$limit?
```

This command clones all spos entries from the source (`$src`) object to the destination (`$dest`) object. If `$limit` (a number) is given only the first `$limit` spos entries are cloned.

spos fnamebase

```
$db spos fnamebase ?$base?
```

This command sets/gets the filename base which is used when retrieving relative filenames added with `-sposn` and `-pinsposn`.

spos setfname

```
$db spos setfname (-idx $idx)|$fname ?-short? $newfname
```

This command changes the stored file referenced by spos to a new file. The new file should be identical to the old file but can be located somewhere else. With the `-short` option a short name can

be set. It has only comment character and can not be used to identify a file in any spos command.

spos getfname

```
set fn [$db spos getfname (-idx $idx)|$fname] ?-short?
```

This command returns the stored file name for a file. With the `-short` option the short name can be retrieved. If none is set an empty string is returned.

The Cone Extraction

Introduction

This Document describes the "Cone Extraction" - this is an extension to the Database API and bases on the zdb command $db cone.

This is an implementation of a cone extraction algorithm, whose result can be sent to the GUI Cone window. It does not modify the database.

The cone extraction algorithm only returns meaningful results for gate level or block level designs.

Overview

Three major modes are supported:

```
$db cone            options $start
$db cone            options -startList $list
$db cone -reachable options $start
$db cone -reachable options -startList $list
$db cone -paths     options $start

$db cone definearccheck
zdb cone            options
```

The first mode extracts the **Cone Of Influence** between the start point and a set of (potential) end points.

The third mode (with **-reachable** option) computes a list of **Reachable Targets**.

The fifth mode (with **-paths** option) performs a **Path Extraction** between the start point and a set of (potential) end points.

The second and forth mode is equivalent to the first and third, but allows to specify **multiple start points** (multiple start points are not supported for -paths).

The start point(s) are specified by **start**-OIDs, either Net, Pin, Port or Inst (Inst is not supported for the **-paths** mode).

The potential end points (we call them "Targets") are specified by the Target Options below.

The algorithm checks the direction of the traversed pins and ports: inout directions are traversed in both directions and unknown directions are not traversed at all (unknown directions are usually created by the Verilog parser (or other parser) if undefined blocks are instantiated) - but option -unknown2IO changes this behavior.

With the last command global **User Defined Arcs** for the cone extraction can be defined. To Check the user defined arcs definition the **definearccheck** command can be used.

The Cone Of Influence

The cone extraction algorithm searches all paths from the given **start** points to all reachable Targets in O(n) (n is the number of searched nets, maximum all nets in the database). The search result is a list of **pin**-OIDs and **port**-OIDs describing all signal paths that form the "Cone". The result list may include some duplicate OIDs.

The Basic Options

`-in`	Perform an input cone extraction (search the signals towards the input).
`-out`	Perform an output cone extraction; this option is mutually exclusive with `-in`; either `-out` or `-in` is required, if the direction is not implicitly defined by the start-pin or port.
`-ignoreDir`	Ignore all directions while extracting the cone.
`-limit` 'num'	Limit the search depth to the given number (number of combinational levels). If the extraction algorithm hits the given limit, then the result is most probably incomplete. Default is no limit.
`-dontDive`	Don't cross hierarchy borders: all module Instances will be treated like primitives without a known function, so the search won't dive down into sub-modules and would not dive up. It speeds up both path- and cone-extraction. If a sub-module pin is reached the cone extraction will continue at the opposite pin.
`-emptyModAsPrim`	Empty modules are treated like primitives. If a cone extraction hits a pin of an instance of an empty module then the cone extraction will continue at the opposite pins. With this option the Cone extraction will not dive into empty modules.
`-checkArcs`	Don't traverse the internals of Modules with a known function but directly jump from an input to output or vice versa. This can significantly speedup the search, but the search will not check any target/exclude flags inside those known Modules. See **User Defined Arcs** for exceptions.
`-unknown2IO`	Normally, unknown pin directions stop the cone extraction, but this option makes them to be handled like IO pins.
`-addStartPin`	If the search was not successful, but the start-point(s) were pins, ports or instances, then add those start point(s) to the result list anyway.
`-startList` '$list'	Specify a list of start points instead of just one (see overview).

-flat	Exclude hierarchical pins in the result.
-processPGNet	Continue tracing the cone through power/ground nets.
-autoPopulate	If the database was opened from a binfile using quick mode, then the -autoPopulate option can be used to automatically populate the contents of modules to find a path through the module.
-createNetSeg	Additionally include net segments in the result.
-customCB 'proc'	The callback procedure **proc** get called with $db and OID as arguments, for each OID in the result. If the function returns '-code continue', the OID is omitted from the result.
-filterLogicalInvalid	Ignore paths through logical gate input if other inputs are connected to constants nets.

The Target Options

The target options can specify multiple targets, the algorithm uses them as an OR-combination of end points for the search. Most target options can be repeated multiple times, but the -targetObj option can not.

-targetIO	This is a Boolean switch. If specified, then all top-level IO ports are targets (potential end points for the cone extraction).
-targetDangle	This is a Boolean switch. If specified, then all dangling nets (only connected to one pin) are targets.
-targetPGNet	This is a Boolean switch. If specified, then all power/ground nets are targets.
-targetConstNet	This is a Boolean switch. If specified, then all constant values are targets.
-targetOpenPin	This is a Boolean switch. If specified, then all unconnected pins (of the desired direction) are targets.
-targetOpenPort	This is a Boolean switch. If specified, then all unconnected ports are targets.
-targetLoop	This is a Boolean switch. If specified, then all reachable feedback loops are targets.
-targetObj '$obj'	Specify the given OID as target object (in addition to the targets specified by other options - but this option can not be repeated). Supported target object types are: net, inst, pin, pinBus, port and portBus. If a pinBus is given, then all bus members are targets, if a inst is given, then all pins are targets. A port and portBus work identical to pin and pinBus, except that ports refer to the inner side (down level) pins. All the target object IDs can be module-based or tree-based. If they are tree-based, then they address the only one instance - however, if they are module-based, then they address all instance-paths that lead to the OID.

`-targetFlatFlagged` 'flag'	All flagged objects (with the given 'flag' set) are targets. Note: these flags are tree-based (not module-based as for `-targetFlaggedInst` below) and therefore match exactly the specified instance path. The flat-flags at `net`, `inst`, `pin`, `pinBus`, `port` and `portBus` objects are considered; this option works pretty similar to the `-targetObj` above, but of course many objects may have been flagged before the cone extraction runs.
`-targetCell` 'cellname'	Specify all instances of the given cell to be targets. It is often used to specify latches - for cone extraction from an arbitrary start point to all latches.
`-targetCell_icase` 'cellname'	Like -targetCell, but the cellname search is case-insensitive.
`-targetCell_glob` 'pat'	Like -targetCell, but a glob-style pattern is used to identify the target cells.
`-targetCell_icase_glob` 'pat'	Like -targetCell_glob, but the pattern is case-insensitive.
`-targetFlaggedInst` 'flag'	All flagged instance-objects (with the given 'flag' set) are targets. Note: these flags are module-based (we don't use the tree-based flat flags here) and therefore match all instance paths that lead to the given instance-object.
`-targetFlaggedCell` 'flag'	All instances of flagged cells (with the given 'flag' set) are targets. Note: the cell flag "undefined" is set by the Verilog parser for undefined module or prim instantiations - this flag can be used here to make them targets.
`-targetFlaggedNet` 'flag'	Like `-targetFlaggedInst`, but for Nets: all flagged net-objects are targets.
`-targetFlaggedPin` 'flag'	Like `-targetFlaggedInst`, but for instance Pins: all flagged instance pin-objects are targets.
`-targetFlaggedPort` 'flag'	Like `-targetFlaggedPin`, but at each instance Pins, the referenced Port is checked.

The Exclude Options

The exclude options can specify objects that make the algorithm stop searching (without finding a target in a given search branch). If the same object is both a Target and an Exclude object, then the target meaning has priority. Multiple Exclude objects are treated as an OR-combination of (non-successive) end points for the search. Each Exclude option can be repeated multiple times.

`-excludeCell` 'cellname'	All instances of the given cell make the algorithm stop searching. It can be used to specify certain modules to be excluded from the cone extraction search.
`-excludeFlaggedCell` 'flag'	All instances of cells with this flag set make the algorithm stop searching (analog to the -targetFlaggedCell option).
`-excludeFlaggedInst` 'flag'	All instances with this flag set make the algorithm stop searching (analog to the -targetFlaggedInst option).
`-excludeFlaggedPin` 'flag'	All pins with this flag set make the algorithm stop searching (analog to the -targetFlaggedInst option).

`-excludeFlaggedPort` 'flag'	All ports with this flag set make the algorithm stop searching (analog to the -targetFlaggedCell option).
`-excludeFlaggedNet` 'flag'	All nets with this flag set make the algorithm stop searching (analog to the -targetFlaggedPin option).
`-excludeFlatFlagged` 'flag'	All flagged objects (with the given 'flag' set) make the algorithm stop searching (analog to the -targetFlatFlagged option). Note: these flags are tree-based (not module-based as for `-excludeFlaggedInst`) and therefore match exactly the specified instance path.
`-excludeFuncPort`	Ports with a function (e.g. clock, reset, set, select) get flagged internally and are ignored like `-excludeFlaggedPort`.

Path Extraction

The path extraction algorithm searches all paths from the given **start** point to all reachable Targets and returns the -pathLimit shortest paths.

Internally, this algorithm first computes the Cone Of Influence and then operates on that data. This means, all cone extraction options above apply.

The search result is a list of paths - sorted by path depth (the shortest path is the first). Each path itself is a list of **pin**-OIDs and **port**-OIDs describing the signal path, but the first element is an integer (rather than an OID) storing the depth of the path.

More Options for Path Extraction

`-pathLimit` 'num'	Limit the number of created paths to the given number. If the extraction algorithm hits the given limit, then only the shortest n paths are returned. Default is 100. If the given limit is 0 then up to 65535 paths are returned. If the memory consumption for this extraction algorithm would be too high (because the search space is too big), then eventually less paths as defined by -pathLimit are returned and a warning is issued.
`-shortestPath`	If only the shortest path is wanted, this option can be issued, to use a faster breadth first algorithm, with a different implementation. Not all options make sense here. For the special case of multiple target nets the option **-targetNet** can be used multiple times here.
`-customCB` 'proc'	The callback procedure **proc** gets called with $db and OID as arguments, for each OID in the result paths. If the function returns '-code continue', the OID is omitted from the result path.

Reachable Targets

This mode is identical to extracting the Cone Of Influence, but the search result is a list of the target OIDs only; that is a list of **pin**-OIDs and **port**-OIDs of the end points of the search.

User Defined Arcs

If needed, Arcs for some cells can be defined in a global table. Four commands can be used to maintain this table:

```
zdb cone cleararc  ?$cellName?
zdb cone definearc  $cellname $enterport $leaveportList ...
zdb cone invertarc ?$cellName?
$db cone definearccheck
```

cleararc clears the whole table or one given cell. **definearc** adds entries to the table. Multiple $enterport / $leaveportList pairs can be used in one **definearc** command. Use an empty $leaveportList to block arcs from the given $enterport. **invertarc** may be used to use all arcs which are **not** defined. The table is asked only if the cone -checkArcs option is used. The **definearccheck** command looks whether all or no ports are defined in the table. For Cells with only partial definitions a warning is issued, because missing port are treated like a complete arc to all possible ports.

Cone to Power/Ground

For a given start pin the coneToPG command tries to search a path to power ground nets through transistor devices.

With the **-opposite** option the opposite pin of $startPin is used.

With the **-limit** option only path with limited length are returned.

With the **-connlimit** option nets with more than the given number of connections are ignored.

With the **-only** option only supply nets with the given flag are targets.

With the **-excludeFlaggedInst** instances flagged with the given flag are not allowed in the result.

With the **-excludeFlaggedCell** instances of cells flagged with the given flag are not allowed in the result.

```
$db coneToPG $startPin  ?-opposite?
                        ?-limit num?
                        ?-connlimit num?
                        ?-only power|ground|negpower?
                        ?-excludeFlaggedCell flag?
                        ?-excludeFlaggedInst flag?
```

The Operator API

This Document describes the "Operator API" - this is an extension to the Database API and bases on the `zdb` sub-command `$db oper`.

In contrast to the API commands, these operators usually execute algorithms that **modify** the database.

Overview

The operators modify the Database.

It is not safe to run operators modifying the database while using a `$db foreach ⋯` loop - especially if the body of the foreach loop modifies the current iteration type.

Here is an overview of the operator commands:

```
$db oper maketop $topCnt
$db oper guessTopModule ?-singleTop? ?-useAllUnreferencedCells? ?-ignoreLibcells?
$db oper sorttop
$db oper tsort
$db oper deftop $mod
$db oper deleteUnused
$db oper zombieUnused

$db oper validatePG

$db oper mergeNet $net $net2
$db oper mergeNetBus $netBus $netBus2
$db oper connect $pin $net
$db oper disconnect $pin

$db oper hiersep scan|get|wish
$db oper unused wish

$db oper singlize $inst
$db oper singlizePath $inst
$db oper singlizeTree $mod

$db oper addhier $mname $iname $instList
$db oper rmhier $inst ?$prefix? ?-delzombies?

$db oper createHier ?-doNotGuessPortDir? $top $hiersep ?$prefix?

$db oper collectSignalData ?-pg? ?-value? ?-spos? ?-attr? ?-hilight? ?-flatattr? ?-flathilight?
?$top?
$db oper flatdesign ?-spos? ?-keep? $top

$db oper flattenSubtree $inst

$db oper gate $mod ?-fold|-autofold? ?-noflat? ⋯
$db oper gateadd $type $key $stype $suffix $iorder
$db oper gateadd -clear
$db oper hierStart $top ?-fold|-autofold? ?-noflat? ⋯
$db oper hierAdd $cname $iname $iList $oList ⋯
$db oper hierFinish
```

```
$db oper hierFoldCount
$db oper hierFoldDupl foldkey
$db oper hierFoldList

$db oper hierPathCount
$db oper hierPathDupl $foldkey

$db oper rename ?-updateOIDs? ?-checkName? ?$oid $newname?
$db oper renameUniq ?-updateOIDs?
$db oper renameDigit

$db oper changecellref ?-updateOIDs? ?-similar? ?$inst $newcell?
$db oper setDirection $oid $dir ?-updatePins?

$db oper bulk wrong|all|none|nopg

$db oper guessPortBus $prim|$mod open close
$db oper guessNetBus ?-down? $mod open close
$db oper guessInstArray $mod open close ?-delzombies?
$db oper sortPorts ?-down?
$db oper guessBusses ?-down? $open ?$close?
$db oper verilogBusses ?$cell?
$db oper distributeSubBits netBus attrName

$db oper removeBuf $mod|-all ?-delzombies?
$db oper removeInv $mod|-all ?-delzombies?
$db oper createConst $mod|-all ?-netstub? ?-bool? ?-delzombies?
$db oper guessWide $mod|-all ?-equal? ?-mux? ?-prio? ?-dlatch? ?-repeat? ?-delzombies?
$db oper guessWideBus $mod|-all ?-delzombies?
$db oper chain $mod|-all ?-mux? ?-bool? ?-inv? ?-delzombies?
$db oper reducePins $mod|-all ?-created? ?-modules? ?-delzombies?
$db oper removeDangle $mod|-all ?-delzombies?
$db oper instarray $mod|-all open close ?-delzombies?
$db oper bubbles $mod|-all ?-delzombies?
$db oper negedge $mod|-all ?-delzombies?
$db oper removeUnused ?-delzombies?
$db oper mergeRams ?-delzombies? ?$mod?
$db oper deletePort $cell $pname …
$db oper guessDir

$db oper groupmultifinger $sep
$db oper weakflow ?-hspice? ?-reslimit limit?
$db oper poweranddirsettings ?-node m n t? ?-icase?
$db oper guesspower
$db  oper poweranddirguess ?-icase?  ?-avoidshorted?  ?-evalvsrc2p?  ?-evalvsrc2i?  ?-addtopports?
?-pwrprop? ?-force?
$db oper hidepowerports
$db oper cleanup
$db oper expand ?-auto? ?-auto0? ?-icase? ?str …..?
```

```
$db oper shortRes limit ?-hspice? ?-delzombies?
$db oper removeMOS ?-delzombies?
$db oper removeUseless ?-delzombies?
$db oper removeRes ?-delzombies?
$db oper removeCap ?-delzombies?
$db oper mergeParallelCap ?-hspice? ?-parasitic? ?-delzombies? ?mod?
$db oper mergeParallelRes ?-hspice? ?-parasitic? ?-delzombies? ?mod?
$db oper mergeParallelDiode ?-hspice? ?-parasitic? ?-delzombies? ?mod?
$db oper mergeParallelInst ?-hspice? ?-parasitic? ?-delzombies? ?-module mod? ⋯
$db oper reportParallelInst ?-parasitic? ?-module mod? ⋯
$db oper mergeSerialRes ?-hspice? ?-delzombies? ?mod?
$db oper mergeSerialCap ?-hspice? ?-delzombies? ?mod?
$db oper mergeParallel ?-hspice? ?-multi m? ?-delzombies?
$db oper mergeSerial ?-delzombies?
$db oper mergeSerialParallel ?-hspice? ?-multi m? ?-equality val? `?-delzombies?
$db oper removeEmptyModule ?-delzombies? ?-checkDanglingNets? ?-checkZombies?

$db oper analyzeCoupling
$db oper paraRmHier ?-delzombies? ?-prefix $pref? ?$inst|$mod?
$db oper paraRename $mod
$db oper paraMinMax $mod
$db oper paraNetCap $mod
$db oper paraFinish ?-deleteUnmatched?
$db oper paraInline ?-inlinemod? ?-delzombies?
$db oper paraRollback ?-delzombies?
```

Make and Sort Top Modules, Topological Sort

```
$db oper maketop $topCnt
$db oper guessTopModule ?-singleTop? ?-useAllUnreferencedCells? ?-ignoreLibcells?
$db oper sorttop
$db oper tsort
```

The `maketop` command searches for up to $topCnt "top-level" modules and adds them to the list of top-modules; see the `$db foreach top ⋯` loop.

The `guessTopModule` command flags modules as top which were most likely top modules. If only library cells are present then all of them become top module.

- If the `singleTop` option is given then the largest module of the found top candidates is used as top module.
- If the `useAllUnreferencedCells` option is given then all unreferenced cells become top.
- If the `ignoreLibcells` option is given then cells with the `libcell` flag are ignored.

The `sorttop` command divides the list of top-modules into two sections and sorts each section (this affects the order of the `$db foreach top ⋯` loop); the first section stores the "real tops" (modules that instantiate other modules) and the second section may store unreferenced library cells. The

`tsort` command performs a topological sort to get a "declared before used" order of the modules. This re-orders the list of modules; see the `$db foreach module ⋯` loop.

Define a New User-Defined Top Module

```
$db oper deftop $mod
```

The `deftop` changes the potentially existing top modules (see `$db foreach top ⋯`) into ordinary modules and establish one new top module. If the new top module is already instantiated, then those instances are deleted (and disconnected). This function internally uses the zombie flag; if that flag is set through the API, then additional objects may be deleted as if `$db deleteZombies` was called.

Delete Unused Modules

```
$db oper deleteUnused
$db oper zombieUnused
```

The `deleteUnused` function deletes all not-instantiated modules (except the top modules). This function can be useful after `$db oper deftop ⋯` was called to remove the unneeded parts of the design hierarchy tree. This function internally uses the zombie flag; if that flag is set through the API, then additional objects may be deleted as if `$db deleteZombies` was called.

The `zombieUnused` function can be used to set only the zombie flag without deleting the objects.

Validate Power/Ground

```
$db oper validatePG
```

This operator validates the database for power and ground nets. If no power or ground net was found then an error will be raised. This operator is useful for databases with transistor devices.

Change Connectivity

```
$db oper mergeNet  $net $net2
$db oper mergeNetBus $netBus $netBus2
$db oper disconnect  $pin
$db oper connect     $pin $net
```

The first command merges two nets by appending `$net2`'s connectivity to `$net`. All connectivity (as e.g. specified by `$db load net`'s `-pin` and `-port`) is moved from `$net2` to `$net`. Net `$net2` survives with no connections at all. Power/ground/negpower flags from `$net2` are also merged into `$net`.

The second command merges all members of two netBuses with the same behavior as described above for `mergeNet`.

The third command disconnects a Pin or Port from a Net. The `$pin` must be a pin or port OID - if the pin/port already is unconnected then this command has no effect.

The fourth command connects a Pin or Port to a Net. The `$pin` must be a pin or port OID - if the pin/port is already connected to the given net then this command has no effect. If the pin/port is already connected to another net then this command throws an error.

Scan and Get a Hierarchy Separator Character

```
$db oper hiersep scan
$db oper hiersep get
$db oper hiersep wish $character
$db oper unused $wish
```

The `scan` option scans all instance and net names in the database and returns a character that's not used in any name - so it can be used as a separator character for path names.

The `get` option returns the previously scanned character - or if no "scan" was performed before, then it performs a scan exactly like the "scan" option do.

The `wish` option sets the desired hierarchy separator `$character`. If this character is not used in any name then it will be used as a separator character for path names.

The `unused` operator can be used to scan the database and return characters which are not used in the design. For each character in `$wish` an unused character is returned in the result.

Unfold Hierarchy

In the zdb, each hierarchical module is either instantiated zero times (e.g. a top module) or one time (is "singlized") or multiple times, i.e. it is referred by more than one instance (is not "singlized"). We use the term "singlized module" if the module is only referred by one instance. The following commands clone modules if required to get a "singlized" hierarchy:

```
$db oper singlize     $inst
$db oper singlizePath $inst
$db oper singlizeTree $mod
```

The `singlize` command checks only the one given instance `$inst`. If its down-module's refcount is more than one (used by other instances also), then the module is cloned (same contents but the module name is extended by a unique number) and attached to that instance only. Then, the cloned module's refcount is one - so it is private to that instance.

The `singlizePath` command walks down the instance path and performs `singlize` for each instance on the path as well as for the final `$inst` (`$inst` must be a tree-based OID).

The `singlizeTree` traverses the full sub-tree rooted at `$mod` and performs the "singlizing" for each hierarchy instance.

The singlize and singlizePath commands also "go to" the singlized module (exactly as reloadModule does), so that further load commands can add more objects to the singlized module. The singlize and singlizePath commands also accept tree-based module-OIDs (instead of instance-OIDs).

These commands update the native representation of all affected Tcl OID variables. That means that all existing Tcl OID variables stay at their objects. This kind of OID updates usually don't change their textual representation, except for Module and Primitive OIDs, e.g. module top U2 U43 MUX will change to module top U2 U43 MUX#1 (if instance top.U2.U43 was singlized).

Recreate Hierarchy

```
$db oper createHier ?-doNotGuessPortDir? $top $hiersep ?$prefix?
```

Split all instances and nets of a flat design at the given hierarchy separator and re-create the hierarchy.

Collect Signal Data

Collect signal data from all connected nets. With the options -pg, -value, -spos, -attr, -hilight, -flatattr and -flathilight only selected data is copied to the main signal net. If no options are used all data is copied. If a top module is given, only this hierarchy is processed. Otherwise all hierarchies are processed.

```
$db oper collectSignalData ?-pg? ?-value? ?-spos? \
                           ?-attr? ?-hilight? ?-flatattr? ?-flathilight? \
                           ?$top?
```

Flat a Hierarchical Design

The whole design get flattened into the given top module. If the -spos option is given, source file position are created for the flattened objects.

If the -keep option is given then the original hierarchical top will be preserved.

```
$db oper flatdesign ?-spos? ?-keep? $top
```

Add and Remove Hierarchy Level

```
$db oper addhier $mname $iname $instList ?$prefix? ?-usefirstinst? -pwrgndtoo?
$db oper rmhier  $inst ?$prefix? ?-delzombies?
```

The addhier command adds a new level of hierarchy, grouping the given $instList (list of Instance OIDs). All elements in $instList must have the same root and path - meaning they must be located

in the same *parent* module. The $mname is the name of the new Module (created to group the $instList) and $iname is the name of the new Instance of that Module. That new component is created within the parent module and its OID is returned. If the optional $prefix argument is given, then it specifies the prefix of characters removed from each inst/net/netBus name moved to the new Module. If the optional -usefirstinst argument is given, the first instance of $instList is used to determine the ports. If the optional -pwrgndtoo argument is given, power/ground nets create ports too.

The $instList may store tree-based and/or module-based OIDs, but they never singlize the path to the parent, meaning the command applies to all instantiations (if there are multi-instantiated Modules in the design).

The rmhier command removes the specified Module Instance and moves its contents up one level of hierarchy. The given $inst must be a tree-based Instance-OID (of the object to be removed). The optional $prefix argument specifies a prefix-string that is used to prefix all Instances, Nets and NetBuses names. If $prefix is not specified, then rmhier first checks if the existing names would create a name clash and if so, then it ties different prefixes until it finds one that will not create a name clash. As prefixes, it uses the $inst's name plus one of these single characters: _ - : <space> , # $ % & (but not the current hiersep). Like addhier, this command also never singlizes the path to the $inst, meaning if the path to $inst is multi-instantiated, then rmhier returns an error.

To remove the deleted objects the -delzombies options is needed. To optimize the performance for multiple rmhier commands this option can be omitted for all but the last commands.

The algorithm for addhier splits the crossing nets and routes them through the new module's interface ($mname) - except for power/ground/negpower nets (those nets are split but not routed through the new module's interface); rmhier is the undo-function to addhier and merges the split nets into a single net.

Flat a Complete Subtree

The subtree rooted at the specified hierarchical instance is flattened. All hierarchical instances in the subtree must be <<SINGLIZE,singlized>.

```
$db oper flattenSubtree $inst
```

Update Tcl OID Variables

As the singlize command above, the addhier and rmhier commands update the native representation of all affected Tcl OID variables. That means that all existing Tcl OID variables stay at their objects.

Tcl OID variables, pointing to objects that are removed by the rmhier command, turn into null-string variables (they are no longer OID variables).

Typically the instance-paths of the affected OID variables are enlarged (by addhier) or reduced (by rmhier). E.g.

pin top U2 U43 U3 IN will change to pin top U2 U43_U3 IN by the command $db oper rmhier {inst

`top U2 U43} U43_` (note the prefix `U43_`), and

`pin top U2 U43 U3 IN` will change to `pin top U2 U77 U43 U3 IN` by the command `$db oper addhier $mname U77 {{inst top U2 U43} ⋯}`.

This note is for Tcl experts only:

There is a potential problem for Tcl userware scripts that use the Tcl variables by their text representation (see `oid1` below), or by replacing the Tcl variable's native OID representation by another native representation (see `oid2` below):

```
set oid1 [$db oid create ....]
set tab($oid1) 1            ;# uses text-representation for hash table
```

```
set oid2 [$db oid create ....]
set type [lindex $oid2 0]   ;# overwrite oid2's native OID representation
                            ;# by a native LIST representation.
```

There is no risk when using Tcl lists of OID variables, but hash tables (`oid1` above) use the current text representation to store the hash key; if `oid1` later changes because of `addhier` or `rmhier` or `singlize`, then the hash table `tab` still stores the old text representation. Turning an OID variable into a list (`oid2` above) is usually not required, because access functions for all OID members exist.

Internal Hierarchical Operators

```
$db oper gateadd   $type $key $stype $suffix $iorder
$db oper gateadd   -clear
$db oper hierStart $top ?-fold|-autofold? ?-noflat? ?-notran? ?-nopar?
$db oper hierAdd   $cname $iname $iList $oList $sList $mList \
                   ?-stype $stype? ?-oneg? ?-fold foldkey?
$db oper hierFinish
$db oper hierFoldCount
$db oper hierPathCount
$db oper hierPathDupl $foldkey
$db oper hierFoldList
$db oper hierFoldDupl foldkey
$db oper hierFoldList
```

These oper functions are for internal use only.

Multiple calls to `hierAdd` need to be surrounded by one call to `hierStart` and `hierFinish`.

The other opers are useful for statistics.

Rename Objects

```
$db oper rename ?-updateOIDs? ?-checkName? ?$oid $newname?
$db oper renameUniq ?-updateOIDs?
$db oper renameDigit
```

The `rename` command changes the name of the given OID to `$newname`. The `$db oid oname` command will return `$newname`.

If the option `-updateOIDs` is specified, then all Tcl variables referring to object ids will be updated. The same effect can be achieved using the `$db oid resetAllOIDs` command.

The option `-checkName` will check if the rename would create a name clash and throws an error.

Please note that the rename operator works module based. If you rename an object with either flat attributes or flat highlight in a module that is instantiated multiple times, then the flat tree structure is not updated. To work around this limitation you can use the singlize operator to unfold the hierarchy.

The `renameUniq` command renames all Modules, Primitives, Ports and PortBusses such that they have a unique name (Modules and Primitives are renamed to `C{number}`, Ports are renamed to `p{number}`, and PortBusses are renamed to `b{number}`, where `{number}` is an increasing integer value).

The `renameDigit` command renames all Nets and Ports to avoid starting with a digit.

Change Cellref

```
$db oper changecellref ?-updateOIDs? ?-similar? ?$inst $newcell?
```

The `changecellref` command changes the referenced cell of the given instance OID to the cell specified as `$newcell`. The interface of `$newcell` must be identical to the original cell - unless the `-similar` option is specified.

If the option `-similar` is specified, then `$newcell` does not need to have the same interface as the original cell - instead, the instance's connectivity is restored by port names: If `$newcell` is missing a port, the originally connected net will be disconnected, if `$newcell` has additional ports, the corresponding instance pins will not be connected to any net.

If the option `-updateOIDs` is specified, then all Tcl variables referring to object ids will be updated.

Set Direction

```
$db oper setDirection $port    $dir ?-updatePins?
$db oper setDirection $pin     $dir
$db oper setDirection $portBus $dir ?-updatePins?
$db oper setDirection $pinBus  $dir
```

`$dir` may be on one of `input`, `output`, `inout`, or `unknown`.

The `setDirection` command may be used to change the direction of a given pin, port, pinBus or portBus. The direction can only be changed for cells without a function.

To get a valid database the direction of pins and ports must be in sync. The `-updatePins` option can be used to make this automatically. Because this is a relative costly operation it should be done once for all changes.

Toggle Visibility of Bulk Pins

```
$db oper bulk wrong|all|none|nopg
```

The `bulk` command toggles the visibility of the bulk pin at NMOS, PMOS, NPN and PNP transistors. If devices with the functions CAP, RES, INDUCTOR, DIODE or ZDIODE are loaded with the optional bulk pin then these devices are also processed by this operator.

If *mode* is `wrong` then only bulk connections at PMOS or PNP transistors are shown not connected to a power or negpower node and bulk connections at NMOS or NPN transistors are shown not connected to a ground node.

If *mode* is `all` then all bulk connections are shown.

If *mode* is `none` then no bulk connections are shown.

If *mode* is `nopg` then only bulks not connected to a power, negpower or ground node are shown.

Please note that not connected bulk pins are always hidden and that bulk hiding for symbol shapes coming from a symbol library is not supported.

Guess Port Buses of a Cell

```
$db oper guessPortBus $cell $open ?$close?
```

Loops over all ports of a cell and tries to guess PortBuses. Sequential ports with same basename followed by a bit subscript will create a port bus. The bit subscript must consist of numerical digits prefixed by one character out of `$open` and ending with a character out of the optional `$close` argument. For the special case that `$open` contains `0`, no character between base name and bit subscript is needed. If the ports with same basename are not in a sequential order they can be sorted by calling the `sortPorts` operator in advance.

Guess Net Buses of a Module

```
$db oper guessNetBus ?-down? $mod $open ?$close?
```

Loops over all nets of a module and tries to guess NetBuses. Sequential net with same basename followed by a bit subscript will create a net bus. The bit subscript must consist of numerical digits prefixed by one character out of `$open` and ending with a character out of the optional `$close`

argument. For the special case that $open contains 0, no character between base name and bit subscript is needed. Subscripts will get numerical sorted downwards if -down is specified.

Guess Inst Arrays of a Module

```
$db oper guessInstArray $mod $open ?$close? ?-delzombies?
```

Loops over all instances of a module and tries to guess arrayed instances. Instances with same basename followed by a bit subscript will create an arrayed instance. The bit subscript must consist of numerical digits prefixed by one character out of $open and ending with a character out of the optional $close argument. An optional suffix after the $close character will be appended to the generated base name. For the special case that $open contains 0, no character between base name and bit subscript is needed.

Sort all Module Ports

```
$db oper sortPorts ?-down?
```

Sorts all modules ports. Subscripts will get numerical sorted downwards if -down is specified.

Create Buses

```
$db oper guessBusses ?-down? $open ?$close?
$db oper verilogBusses ?$cell?
```

The guessBusses command calls all bus guessing operators in the correct order to re-create bus structures from single bit ports.

The verilogBusses command only creates Verilog conform buses from single bit ports with consecutive numbering.

Distribute Bus Values

```
$db oper distributeSubBits $netBus $attrName
```

Distribute value of the given $netBus to all sub-bits.

Remove Buffer

```
$db oper removeBuf $mod|-all ?-delzombies?
```

Remove all BUF and WIDE_BUF instances and merge the connected nets.

Remove Inverter

```
$db oper removeInv $mod|-all ?-delzombies?
```

Remove all INV instances if the input connects to a power or ground net. The net connected to the output is flagged with the inverted value of the input net.

Create Constant

```
$db oper createConst $mod|-all ?-bool? ?-delzombies?
```

Replace all power ground nets connected to input pins with a 'constant' net stub and annotate a 'value' attribute with the constant value.

The option `-bool` looks for AND/OR primitives with constant 0/1 at one input pin. If found, the constant is propagated to the output pin and the instance is removed. Nets connected to X and Z primitives are treated as a value of 'X' and 'Z'. These instances get removed if all connections can be replaced by a constant.

Guess Wide

```
$db oper guessWide $mod|-all ?-equal? ?-mux? ?-prio? ?-dlatch? ?-repeat? ?-delzombies?
$db oper guessWideBus $mod|-all ?-delzombies?
```

Replace multiple scalar primitives connected to same bus pin by corresponding wide function primitive.

The option `-repeat` may be used to recognize wide functions which are found only after the creation of some other wide functions.

The option `-equal` looks for equal/nequal/lessthan primitives connected to a bus pin and converts them to a hierarchical block.

The option `-mux` looks for mux primitives connected to a bus pin and converts them to a hierarchical block or a WIDE_MUX depending nets connected to the `sel` pins.

The option `-prio` looks for PRIO_SELECTOR primitives connected to a bus pin and converts them to a WIDE_PRIO_SELECTOR, but only if the `sel` pins or all connected in the same way.

The option `-dlatch` looks for DLATCH primitives connected to a bus pin and converts them to a hierarchical block.

The `guessWideBus` operator guesses buses from members connected to one output pinBus.

Chain

```
$db oper chain $mod|-all ?-mux? ?-bool? ?-inv? ?-delzombies?
```

Replace chains of MUX/WIDE_MUX primitives by one PRIO_SELECTOR/WIDE_PRIO_SELECTOR instance (`-mux`).

Replace chains of AND/OR primitives by one reduced instance (REDUCE_AND/REDUCE_OR) (`-bool`).

Replace odd number of INVs in a chain by one INV and remove an even number of INVs in a chain.

Reduce Pins

```
$db oper reducePins $mod|-all ?-created? ?-modules? ?-delzombies?
```

Creates an extra hierarchy for primitive instances which have same nets connected to multiple pins.

The option `-created` extends the search to hierarchical instances created by one of the operators above.

The option `-modules` extends the search to module instances.

Remove Dangle

```
$db oper removeDangle $mod|-all ?-delzombies?
```

Remove nets connecting to nothing. Remove NetBuses with no subnets.

Remove Unused

```
$db oper removeUnused ?-delzombies?
```

Remove all cell which not referenced anymore or only referenced by zombie flagged instances.

Merge RAMs

```
$db oper mergeRams ?-delzombies? ?$mod?
```

Merge ReadPort and WritePort instances to one RAM instance. If `$mod` is specified, only consider instances within this module, otherwise work on the whole DB.

Guess Port Directions

```
$db oper guessDir
```

Guess direction of unknown module ports.

Inst Array

```
$db oper instarray $mod|-all ?-delzombies?
```

Group instances which have digits enclosed in one of [<(brackets in their name.

Bubble Pins

```
$db oper bubble $mod|-all ?-delzombies?
```

Replace inverters by bubbles at connected input pins or change function for inverters at output pins.

Bubble Flip-Flop Pins

```
$db oper negedge $mod|-all ?-delzombies?
```

Remove all inverter directly connected to a clock, set or reset pin of a flip-flop and add a bubble to the pin.

Delete Port

```
$db oper deletePort $cell $pname ...
```

Delete named ports or portBuses from given cell. All instance pins and affected nets are updated. This is a relative costly operator, because Spos and OIDs need to be updated.

Device Operators

```
$db oper groupmultifinger $sep
$db oper weakflow ?-hspice? ?-reslimit $limit?
$db oper poweranddirsettings ?-node m n t? ?-icase?
$db oper guesspower
$db oper poweranddirguess ?-icase? ?-avoidshorted? ?-evalvsrc2p? ?-evalvsrc2i? ?-
addtopports? ?-pwrprop? ?-force?
$db oper hidepowerports
$db oper cleanup
$db oper expand ?-auto? ?-auto0? ?-icase? ?str .....?
$db oper shortRes limit ?-hspice? ?-delzombies?
$db oper removeMOS ?-useless? ?-delzombies?
```

Remove spurious MOS transistors:

1. remove NMOS/PMOS with dangling source or drain (only if `-useless` is not given)
2. remove NMOS/PMOS with shortened source, gate, and drain
3. remove NMOS/PMOS with dangling gate, and both source and drain connected to power or ground
4. remove NMOS/PMOS with gate connected to power or ground, and source and drain shortened
5. remove NMOS with gate connected to ground (only if `-useless` is not given)
6. remove PMOS with gate connected to power (only if `-useless` is not given)
7. remove NMOS/PMOS with dangling source, gate, and drain
8. remove NMOS and merge nets connected to source and drain, if gate is connected to power
9. remove PMOS and merge nets connected to source and drain, if gate is connected to ground

```
$db oper removeUseless ?-delzombies?
$db oper removeRes ?-delzombies?
$db oper removeCap ?-delzombies?
$db oper mergeParallelCap?-hspice? ?-parasitic? ?-delzombies? ?$mod?
$db oper mergeParallelRes ?-hspice? ?-parasitic? ?-delzombies? ?$mod?
$db oper mergeParallelDiode ?-hspice? ?-parasitic? ?-delzombies? ?$mod?
$db oper mergeSerialRes ?-hspice? ?-delzombies??-hspice? ?-delzombies? ?$mod?
$db oper mergeSerialCap ?-hspice? ?-delzombies? ?$mod?
$db oper mergeParallel ?-hspice? ?-multi $m? ?-delzombies?
$db oper mergeSerial ?-delzombies?
$db oper mergeSerialParallel `?-hspice? ?-multi m? ?-equality val? `?-delzombies?`
$db oper removeEmptyModule ?-delzombies? ?-checkDanglingNets? ?-checkZombies?
```

Operators related to transistor devices.

Merge Parallel Instances

```
$db oper mergeParallelInst ?-hspice? ?-parasitic? ?-icase? ?-delzombies? ?-module $mod
RULES...
```

The `mergeParallelInst` operator can be used to merge parallel instances controlled by the following options:

- `-hspice`, controls the handling of values with units.
- `-parasitic`, includes parasitic modules in the processing.
- `-icase`, be case insensitive.
- `-delzombies`, deletes unneeded instances.
- `-module $mod`, limits the processing to the given module.
- one or more rules consisting of four strings `<cell>`, `<attrs>`, `<eqfunc>`, `<grpfunc>` each.

`<cell>` controls the instances which will be processed by a given rule. It can be:

- a primitive function like e.g. `RES`, `NMOS`, etc.
- a model name glob pattern, if prefixed by `model:`.
- a cell name glob pattern, if prefixed by `cell:`.
- a module name glob pattern, if prefixed by `module:`.
- a primitive name glob pattern, if prefixed by `primitive:`.

`<attrs>` is a comma separated list of attribute names followed by a equality value enclosed by round brackets, or `*` if the value is not compared. Each group of recognized parallel instances is divided into groups of instances which have equal attribute values and one group with the remaining instances. There is the special case `ARRAYEDINST(str)` where the instance names are used to cluster the parallel instances. `str` defines the open/close characters, separated by a blank, which are used to get the bit subscript.

The `<eqfunc>` describes how the equal instances should be processed. The `<grpfunc>` how the remaining group should be processed. Possible func names are:

- `NOOP`, do nothing.
- `MULTI(<attrname>)`, add an attribute `<attrname>` with the sum of all `<attrname>` values. Missing attributes count as 1.
- `AVG`, calc the average value for each relevant attribute.
- `RES`, calc the resistance of all parallel resistor values.
- `CAP`, calc the sum of all parallel capacitor values.
- `CALCATTR`, calc the W/L and optional nfin value of all instances.
- `INSTARRAY(<openSepClose>)`, create an arrayed instance.
- `MULTIHIER(<attrname>)`, same as `MULTI` but create a hierarchy of all parallel instances.
- `AVGHIER`, same as `AVG` but create a hierarchy of all parallel instances.

- CALCATTRHIER, same as CALCATTR but create a hierarchy of all parallel instances.

<openSepClose> defines the characters needed to create bus range for arrayed instances.

Examples for rules:

```
{RES R(0.3) AVG NOOP}
```

```
{model:cap2 C(0.3) CAP NOOP}
```

```
{NMOS W(0.3),L(0.2) MULTI(M) CALCATTR} \
{PMOS W(0.2),L(0.4) MULTI(M) CALCATTR}
```

```
{NMOS {ARRAYEDINST(<( )>)} {INSTARRAY(( : ))} NOOP}
```

Report Parallel Instances

```
$db oper reportParallelInst ?-hspice? ?-parasitic? ?-icase? ?-module $mod RULES...
```

The reportParallelInst operator can be used to get a list of parallel instances oid controlled by the following options:

- -hspice, controls the handling of values with units.
- -parasitic, includes parasitic modules in the processing.
- -icase, be case insensitive.
- -module $mod, limits the processing to the given module.
- one or more rules consisting of four strings <cell>, <attrs>, <eqfunc>, <grpfunc> each.

<cell> and <attrs> are used like in mergeParallelInst. <eqfunc> and grpfunc are ignored and should be NOOP.

Parasitic Operators

```
$db oper analyzeCoupling
$db oper paraRmHier ?-delzombies? ?-prefix $pref? ?$inst|$mod?
$db oper paraRename $mod
$db oper paraMinMax $mod
$db oper paraNetCap $mod
$db oper paraFinish ?-deleteUnmatched?
$db oper paraInline ?-inlinemod? ?-delzombies?
$db oper paraRollback ?-delzombies?
```

Operators related to parasitic modules.

The Tools API

Introduction

This Document describes the "Tools API" - this is an extension to the "Database API" and bases on the zdb sub-command "$db tools".

In contrast to the "$db oper" API commands, these tools usually execute algorithms that **query** the database.

Overview

The Tools query the database.

Here is an overview of the tools commands:

- $db tools floatingNodes
- $db tools flatfloatNodes
- $db tools cCoupling
- $db tools heavyCR
- $db tools wrongBulk
- $db tools heavyNodes
- $db tools multiDriverCheck
- $db tools zeroDriverCheck
- $db tools createHier
- $db tools findGuessedSupplyNets

Query the Database for Floating Nodes

```
$db tools floatingNodes
```

The **floatingNodes** command searches each module for floating nodes. A floating node is a net which only connects to one pin. Power and Ground nets are skipped. The return value is a list of net OIDs.

Query the Database for Flat Floating Nodes

```
$db tools flatfloatNodes
```

The **flatfloatNodes** command searches the database in a flat manner for floating nodes. A floating node is a signal which only connects to one pin. Power and Ground nets are skipped. The return

value is a list of signal OIDs.

Query the Database for Coupling Capacitors

```
$db tools cCoupling
```

The **cCoupling** command searches each module for coupling capacitors. A coupling capacitor is a capacitor which neither connects to power nor to ground - that means they couple two data wires. The return value is a list of inst OIDs.

Query the Database for Heavy C and R

```
$db tools heavyCR
```

The **heavyCR** command searches each module for capacitors and resistors and checks each value (capacitance or resistance) and collects 10 objects with the biggest values. The 10 biggest Cs and Rs are returned as a list of inst OIDs.

Query the Database for Wrong Bulk Connections

```
$db tools wrongBulk
```

The **wrongBulk** command searches each module for PMOS and NMOS transistors and checks if their bulks are connected to power and ground respectively. If not, then the transistor is part of the result list.

Query the Database for Heavy Nodes

```
$db tools heavyNodes
```

The **heavyNodes** command counts the number of connections for each signal in the design. Power/Ground nets are skipped. The 10 biggest nodes are returned as a list of signal OIDs.

Query the Database for Signals with Multiple Drivers

```
$db tools multiDriverCheck
```

The **multiDriverCheck** command counts the number of driver pins for each signal in the design. Power/Ground nets are skipped. The return value is a list of signal OIDs.

Query the Database for Signals with no Drivers

```
$db tools zeroDriverCheck
```

The **zeroDriverCheck** command searches for signals with no driver. Power/Ground nets are skipped. The return value is a list of signal OIDs.

Recreate Hierarchy from Flat Instance Names

```
$db tools createHier <hiersep> <prefix>
```

The **createHier** command recreates hierarchy from flat instance names of a flat design. The instance names are split at the given hiersep character. If prefix is given, the created instances get sequential names. If no prefix given, the instance names are numbered in hierarchical groups.

Find all Guessed Supply Nets

```
$db tools findGuessedSupplyNets
```

The **findGuessedSupplyNets** command reports all nets that have been guessed by "oper guesspower" as a supply net.

The ZDB Command Kit

The ZDB Command Kit

The Command Kit provides 'high level' commands created by a collection of already existing ZDB database commands.

The following commands are available:

- get_top_design
- get_cell
- get_pins
- get_ports
- get_all_bits
- get_attribute
- get_instances_of_cell
- get_inst
- get_net
- get_para_port_name
- get_driver

- get_load
- print_oid_as_slash_path
- print_oid_as_spef

Access Design Object

Basic Access Functions

get_top_design

```
get_top_design ?-name?
```

Get the top level design. If no top level design could be found then an empty string is returned.

The following options are available:

`-name`	Return the name of the top design.

Example

```
set top  [$db get_top_design]
      set name [$db get_top_design -name]
```

get_cell

```
get_cell ?-modbased? pin|port
```

Get the cell (module or primitive) of a pin or port.

The following options are available:

`-modbased`	Return a module based OID.
`pin|port`	Either a pin or port OID for which the cell OID should be returned.

Example

```
set cell [$db get_cell {pin TOP U1 A}]
```

get_pins

```
get_pins ?-addHier? cell|inst|net|signal
```

Get all pins or ports from a cell/inst/net/signal without having to write a loop.

The following options are available:

`-addHier`	Only valid for a given signal OID. If present then also add hierarchical ports.
`cell|inst|net|signal`	OID to get the pins/ports from.

Example

```
set pins [$db get_pins {signal TOP CLK}]
```

get_ports

```
get_ports ?-name? ?-match pattern? cell
```

Get all ports of the given cell (module, primitive or parasitic).

The following options are available:

`-name`	Return the names only.
`-match pattern`	Only return ports matching the given glob style pattern.
`cell`	Either a primitive, module or parasitic module OID for which the ports should be returned.

Example

```
set ports [$db get_ports {module AND2 AND2}]
```

get_all_bits

```
get_all_bits busOID
```

Get all bits of the given bus.

The following options are available:

`busOID`	Either a pinBus, portBus or netBus OID for which the member bits should be returned.

Example

```
set bits [$db get_all_bits {portBus TOP DATA_IN}]
```

get_attribute

```
get_attribute oid attrName
```

Get the value for an attribute at the given OID.

The following options are available:

`oid`	Any OID type that supports attributes.
`attrName`	Name of the requested attribute value.

Example

```
set value [$db get_attribute {inst TOP U1} width]
```

get_instances_of_cell

```
get_instances_of_cell cellName
```

Get all instances of the given cell type.

The following options are available:

`cellName`	Name of the requested cell.

Example

```
set instList [$db get_instances_of_cell "AND2"]
```

get_inst

```
get_inst ?-hiersep sep? path
```

Get the inst for a given partial flat path.

The following options are available:

`-hiersep sep`	Set hiersep different from default '/'.
`path`	partial flat path name for which the inst OID should be returned.

Example

```
set inst [$db get_inst "sub/U1"]
```

get_net

```
get_net ?-hiersep sep? path
```

Get the net or netBus for a given partial flat path.

The following options are available:

`-hiersep sep`	Set hiersep different from default '/'.
`path`	partial flat path name for which the net OID should be returned.

Example

```
set net [$db get_net "sub/n1"]
```

get_para_port_name

```
get_para_port_name ?-div d? ?-del d?
```

Get parasitic port name from given top port or instance pin

The following options are available:

`-div d`	Use given string instead of divider stored in db.
`-del d`	Use given string instead of delimiter stored in db.

Example

```
set name [$db get_para_port_name {pin TOP U1 A}]
```

get_driver

```
get_driver ?-flat? ?-stopatfirst? ?-ignorebidir? ?-limit count? net|signal
```

Get the driver for a given net or signal.

The following options are available:

`-flat`	Traverse hierarchies when searching for the driver.
`-stopatfirst`	Stop search at the first driver.

`-ignorebidir`	Ignore bidirectional pins/ports.
`-limit count`	Stop search after 'count' pins have been visited.
`net|signal`	net or signal OID.

Example

```
set driver  [$db get_driver -flat {net TOP net1}]
      set driver2 [$db get_driver -flat -limit 1000 {net TOP net1}]
```

get_load

```
get_load ?-flat? ?-stopatfirst? ?-ignorebidir? ?-limit count? net|signal
```

Get the load for a given net or signal.

The following options are available:

`-flat`	Traverse hierarchies when searching for the load.
`-stopatfirst`	Stop search at the first load.
`-ignorebidir`	Ignore bidirectional pins/ports.
`-limit count`	Stop search after 'count' pins have been visited.
`net|signal`	net or signal OID.

Example

```
set load  [$db get_load -flat {net TOP net1}]
      set load2 [$db get_load -flat -limit 1000 {net TOP net1}]
```

print_oid_as_slash_path

```
print_oid_as_slash_path oid
```

Print the given oid as a slash separated name.

The following options are available:

`oid`	The OID to print.

Example

```
$db print_oid_as_slash_path {net TOP net1}
```

print_oid_as_spef

```
print_oid_as_spef oid
```

Print the given oid as a SPEF name.

The following options are available:

`oid`	The OID to print.

Example

```
$db print_oid_as_spef {net TOP net1}
```

The Meta Attributes

The "Meta Attributes" control the visibility of attributes in the Schem and Cone windows. The "Meta Attributes" are identified by the predefined names `@nlv` or `@nlv:⋯` — they are attached to the database root, to the Primitives and Modules and to the Primitives' and Modules' Ports and PortBuses.

Overview

This Document describes the "Meta Attributes" and how they control the display of ordinary database attributes. In addition to this meta attributes, the hide flags also control visibility in the schematic windows.

Those Meta Attributes can be set through the API when loading the database. The Meta Attributes can be modified with ordinary attr commands — usually followed by a call to `gui attribute changed` to update the schematics.

If customer symbols are used (by specifying `-symlib`), then the symbols should define display locations (called `attrdsp`) for the Instance Attributes to be displayed.

Format String

Each "Meta Attribute" defines a format string that is displayed after replacing all occurrences of `%xxx` by the contents of attribute `xxx`. As common to Tcl and other interpreters, the attribute name ends with the first special character or space. Braces may be used to precisely define the end of the attribute name: that means, `%{xx}x` addresses the attribute `xx`, not `xxx`. A double `%%` creates a single `%` character. For backward compatibility, a `$` character has the same meaning as the `%` character.

Newline characters (`\n`) can be used to create multi-line outputs, this is recommended to display multiple Instance attributes, one per line.

Conditional Expressions

Meta attributes may contain conditional expressions of the form:

`?{<attr>{<then>}:{<else>}}`

or

`?{<attr><check><number>{<then>}:{<else>}}`

where the `:{<else>}` part is optional.

The first conditional expression is replaced by the `<then>` part if the referenced attribute `<attr>` exists, otherwise it is replaced by the `<else>` part (if the `<else>` part is missing, an empty string is used).

The second conditional expression contains a `<check>` operator (one of `<`, `=`, `!`, `>`) and a floating point number `<number>`. It is replaced by an empty string if the referenced attribute `<attr>` does not exist, it's replaced by the `<then>` part if the attribute's value fulfills the `<check>` condition (if the value if less (`<`), equal (`=`), not equal (`!`), or greater (`>`) than `<number>`), and it's replaced by the `<else>` part if the condition is not met (if the `<else>` part is missing, an empty string is used).

The `<then>` and `<else>` parts of the conditional expression are themselves full expressions, including references to attributes, or even other conditional expressions.

Display Instance Attributes

The Meta Attribute `@nlv` at an Instance's cellref (Primitive or Module) controls what is displayed at the built-in symbols (for customer symbols, see below). Built-in symbols only define ONE general purpose display location for attributes — but newline characters can be used to separate lines to display multiple attributes. This works identical for both Instances of Primitives and Instances of Modules (UML diagram). One pseudo attribute name `@cell@` is always defined to grant access to the Instance's Primitive or Module name (however, if a `@cell@` attribute exists, then the attribute is used).

Example

Assuming the Instance attributes are `L=12u`, `W=24u` and `AS=8.3`:

Meta Attribute	Display in Schematic
`@nlv=%AS`	`8.3`
`@nlv=W=%W`	`W=24u`
`@nlv=L=%L\nW=%W`	`L=12u` `W=24u`

`@nlv=L=%L\nW=%W\nA=%AS`	`L=12u`
	`W=24u`
	`A=8.3`

`@nlv=L=%L?{M{\nm=%M}:{\nno M}`	`L=12u` `no M`

Instance Attributes at Customer Symbols

If customer symbols are used (e.g. with command line option `-symlib`), then the symbols should define display locations (called "attrdsp") for the instance name (called `@name`) and a general purpose display location for attributes (called `@value` for transistor level (Spice) devices — and `@cell` for gate level Primitives) as well as for Modules. Here are two customer symbol examples (see "The Symlib Format" for more information):

```
symbol _RES_ * DEF device nonpolar \
    attrdsp @name -ll 10 0 12 \
    attrdsp @value -ul 10 3 12 \
    ....
symbol and2 * DEF \
    attrdsp @name -ll 5 -22 12 \
    attrdsp @cell -cl 5 22 10 \
    ....
```

Alternatively, the database root attribute `@nlv:all` could be set (to an arbitrary value) — to make all Instance attributes copied over to the schematic windows and those attributes with a dedicated display information (`attrdsp`) will get displayed (with some performance drawback for big circuits).

API Example

Here is an API example that sets or modifies the `@nlv` meta attribute at all NMOS and PMOS primitives:

```
set meta "L=%L\nW=%W\nA=%AS"
$db foreach primitive p {
    set func [$db primFuncOf $p]
    switch $func {
        "NMOS" -
        "PMOS" {
            $db attr $p set @nlv=$meta
        }
    }
}
```

The next API example just sets the `@nlv:all` database root attribute (see above):

```
$db attr -db set @nlv:all=1
```

Display Pin Attributes

The Meta Attribute `@nlv:pin` at the database root controls what is displayed at the built-in symbol pins (for customer symbols, see below). Built-in symbols only define ONE general purpose display location for pin attributes (newline characters can be used to display multiple pin attributes but because of space limitations, this is not recommended). This works identical for both, Pin and

PinBus objects (UML diagram).

Example

Assuming the Instance Pin attributes include `delay=12.5` at pin `A` and `delay=7.0` at pin `B` — and assuming the meta attribute `@nlv:pin` is set to `%delay` — then the table below shows what is displayed at pins `A` and `B`:

Set Meta Attribute `@nlv:pin` via API	Pin	Display in Schematic
`$db attr -db set @nlv:pin=%delay`	`A`	`12.5`
	`B`	`7.0`

Pin Attributes at Customer Symbols

If customer symbols are used (e.g. with command line option `-symlib`), then the symbols may define display locations (called `pinattrdsp`) for the pin name (called `@name`) and a general purpose display location for pin attributes (called `@attr`). However, if `pinattrdsp @name` is not defined, then the pin name is not displayed, and if `pinattrdsp @attr` is not defined, then the `@attr` is displayed anyway, automatically placed close to the pin location (this means, a customer symbol may predefine `@attr` at the preferred location).

Here is a customer symbol example that defines `@name` and `@attr` for pins (see "The Symlib Format" for more information):

```
symbol enand4 * DEF \
    port O output -loc 55 0 47 0 \
    pinattrdsp @name -cr 52 -4 8i \
    port A input -loc -15 -30 0 -30 \
    pinattrdsp @name -cl -10 -35 8i \
    pinattrdsp @attr -cl -10 -30 8 \
    port B input -loc -15 -10 0 -10 \
    pinattrdsp @name -cl -10 -15 8i \
    pinattrdsp @attr -cl -10 -10 8 \
    port C input -loc -15 10 0 10 \
    pinattrdsp @name -cl -10 5 8i \
    pinattrdsp @attr -cl -10 10 8 \
    port D input -loc -15 30 0 30 \
    pinattrdsp @name -cl -10 25 8i \
    pinattrdsp @attr -cl -10 30 8 \
    ....
```

Display Port Attributes

The Meta Attribute `@nlv:port` at the database root controls what is displayed at the built-in IO connectors (for customer IO symbols, see below). IO connectors only define ONE general purpose display location for attributes — but newline characters (`\n`) can be used to display multiple attributes. This works identical for both Port and PortBus objects (UML diagram).

Example

Assuming Port `A`'s attributes include `delay=3.7`, Port `B`'s attributes include `delay=5.5`, and PortBus `DATA`'s attributes include `delay=9.1` — and assuming the meta attribute `@nlv:port` is set to `%delay ns` — then the table below shows what is displayed at ports `A`, `B` and `DATA`:

Set Meta Attribute `@nlv:port` via API	Port	Display in Schematic
`$db attr -db set "@nlv:port=%delay ns"`	`A`	`3.7 ns`
	`B`	`5.5 ns`
	`DATA`	`9.1 ns`

Port Attributes at Customer IO Symbols

If customer IO symbols are used (e.g. with command line option `-symlib`), then the symbols should define display locations (called `attrdsp`) for the port name (called `@name`) and a general purpose display location for attributes (called `@attr`). Here are three complete customer IO symbol examples (the names are fixed for input, output and bidir IO connector):

```
symbol sv_extInOutPort * DEF \
        attrdsp @name -cl 22 0 10 \
        attrdsp @attr -ll 0 -5 10 \
        port {} input -loc 0 0 0 0 \
        path 15 5 15 -5 20 0 15 5 \
        path 5 5 0 0 5 -5 5 5 \
        path 15 0 5 0
symbol sv_extInPort * DEF \
        attrdsp @name -cr -17 0 10 \
        attrdsp @attr -lr 0 -5 10 \
        port {} output -loc 0 0 0 0 \
        path -15 5 -15 -5 -10 0 -15 5 \
        path -10 0 0 0
symbol sv_extOutPort * DEF \
        attrdsp @name -cl 22 0 10 \
        attrdsp @attr -ll 0 -5 10 \
        port {} input -loc 0 0 0 0 \
        path 15 5 15 -5 20 0 15 5 \
        path 15 0 0 0
```

Display Net Attributes

The Meta Attribute `@nlv:net` at the database root controls what attributes are displayed at the Net and NetBus objects. Those attributes are only visible, if they are enabled in the Preferences dialog at `Display > Net Attributes at wire`.

For power and ground nets, the Meta Attributes `@nlv:powernet` and `@nlv:groundnet` at the database root additionally control what attributes are displayed at the Net's power or ground stubs. Those attributes are only visible, if they are enabled in the Preferences dialog at `Display > Power/Ground Label`. One pseudo attribute name `netname` is always defined to grant access to the Net name

(however, if a `netname` attribute exists, then the attribute is used).

Example

Assuming Net `A` is a power net and its attributes include `width=3.7`, `voltage=1.9` and `crit=420` — and assuming the meta attributes `@nlv:net` and `@nlv:powernet` are set as in the table below — then the table below shows what is displayed at the net and at the net's power stubs:

Set Meta Attributes via API	Display in Schematic	
`$db attr -db set "@nlv:net=%width\ncrit=%crit"` `$db attr -db set "@nlv:powernet=%{voltage}V\n%netname"`	At Net	At Power Stubs
	`w=3.7` `crit=420`	`1.9V` `A`

Attributes at Virtual Objects

To set a format string for a virtual OID, the @nlv:virtual meta attribute can be used.

Formatting Attributes

In addition to the attribute name and value an `@format` attribute at the same object can be added to specify the color and font scale of the displayed attribute.

It allows to specify a foreground and background color and a font scaling factor to overwrite the default foreground color of the attribute and scale the font size to the given value.

The syntax of this option is:

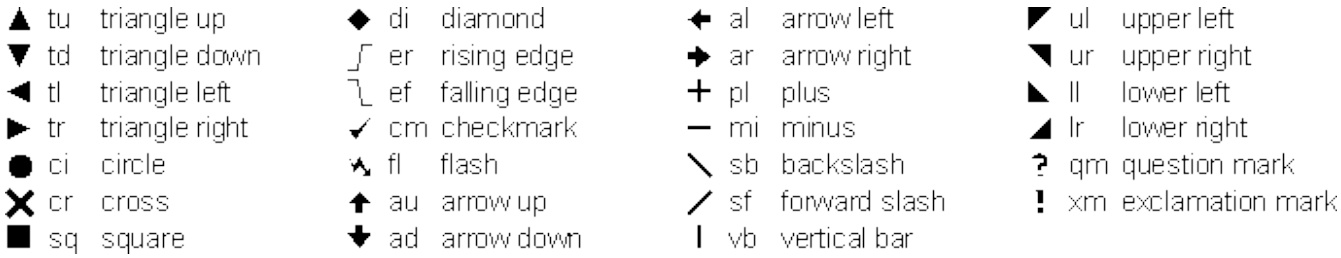
```
(?+?<foreground>,-|<background>,<fontscale>)
```

The `<foreground>` and `<background>` values must be in `#RRGGBB` color format. The `<fontscale>` is a floating point number that specifies a factor for the text height (e.g. 1.5 makes it 50% bigger; 0.7 makes it 30% smaller). The optional leading `+` gives the `<foreground>` color priority over the highlight colors. A `-` as background color defines a transparent background.

Display Graphical Marks

In addition to the text attributes special built-in graphical attributes (marks) can be displayed at Net/NetBus, Pin/PinBus and Port/PortBus objects. The Meta Attribute `@nlv:marks` at the database root controls what graphical marks are displayed. Those attributes are only visible if they are enabled in the Preferences dialog at `Display > Graphical Marks`.

The following mark types are supported:



The format of a graphical mark attribute value is a white-space separated list of single mark definitions containing the mark type and a format string.

```
<name>=<type>(<color>) <type>(<color>)
<name>=<type>(<color>,<bgcolor>) ...
<name>=<type>(<color>,,<ratio>) ...
```

`<name>` is the mark identifier; in order to display this mark, the DB root attribute `@nlv:marks=%<name>` has to be set.

The `<bgcolor>` and `<ratio>` items in the format are optional, but the comma must be given, if `<bgcolor>` is omitted but a `<ratio>` is present.

Two (or more) marks can optionally be combined into more complex shapes by concatenating them with a `&` character like this:

```
<name>=<type>(<color>)&<type>(<color>)
```

`<type>` is one of the predefined mark types.

`<color>` is a color definition to set the (foreground) color of the mark — optionally prefixed by a `+` character to indicate priority over the highlight and selection colors.

`<bgcolor>` is an optional color definition to set the background color for this individual mark. If not specified, the background will be transparent.

`<ratio>` is an optional floating point number which specifies the width:height aspect ratio of the mark. A value of 2.0 will display the mark twice as wide as the default size, with a value of 0.5 only half the default size is used as width. The height is always the internal default marksize and cannot be changed. When `<ratio>` is omitted, a default value of 1.0 is used internally.

The marks are displayed in the same left-to-right order as specified in the attribute value.

Example

Assuming the Instance Pin attributes include `edge=er(#ff0000)` at pin `A` and `edge=ef(#00ff00)` at pin `B` — and assuming the meta attribute `@nlv:marks` is set to `%edge` — then the table below shows what is displayed at pins `A` and `B`:

Set Meta Attributes	Pin	Display in Schematic
`$db attr -db set "@nlv:marks=%edge"`	A	Rising edge symbol (colored red)
	B	Falling edge symbol (colored green)

ZDB Attributes That Influence the GUI Behavior

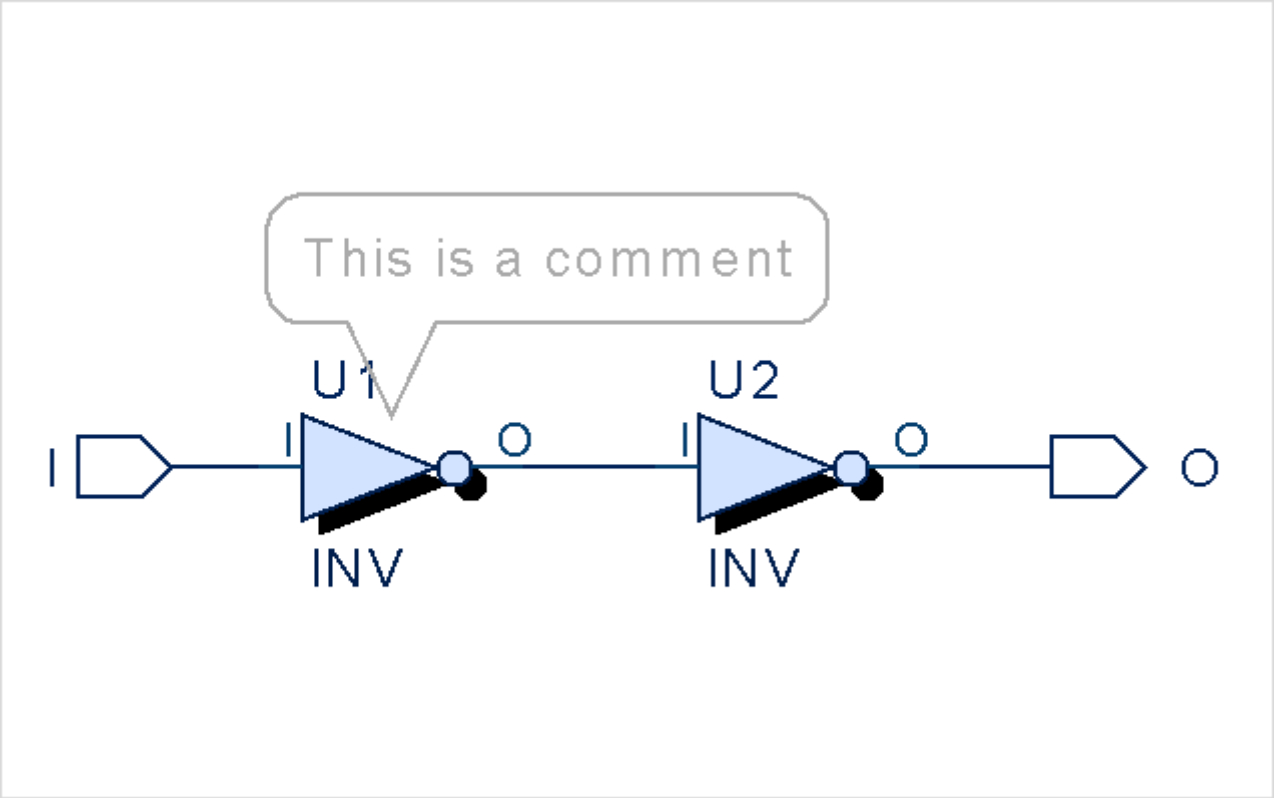
NOTE All of the following ZDB attributes can be set with $db attr commands, e.g.

```
$db attr $oid set {@comment=MY COMMENT}
```

`@comment` and `@commentclickaction` Object Attributes

Textual comments can be attached to objects using the `@comment` attribute. The attribute can be added/modified via the GUI's popup menu (i.e. **Edit › Add Comment ...**). Objects with an `@comment` attribute are displayed with a "speech bubble" icon in certain windows (e.g. in the Mem window); the attribute value is also displayed in tooltips.

If the option Show Comments is enabled via the Preferences dialog, objects' comments are graphically displayed in Schem and Cone windows with speech bubbles. If such a "speech bubble" is clicked, the value of the object's `@commentclickaction` attribute is evaluated as a Tcl script (if the attribute exists).



`@name` Object Attribute

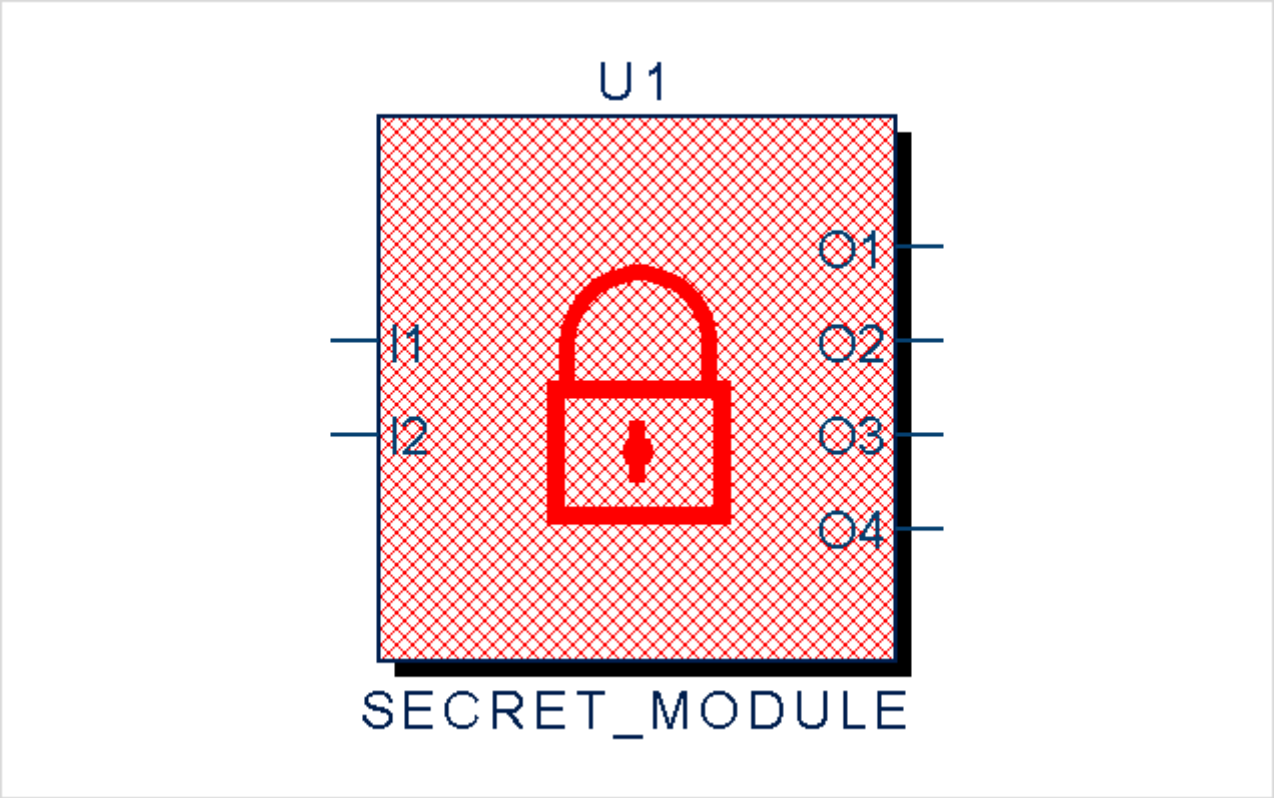
If an object has a `@name` attribute, its value will be displayed in the Schem and Cone windows instead of the original name.

`@cell` Instance Attribute

If an instance has a `@cell` attribute, its value will be displayed in the Schem and Cone windows instead of the original cell name.

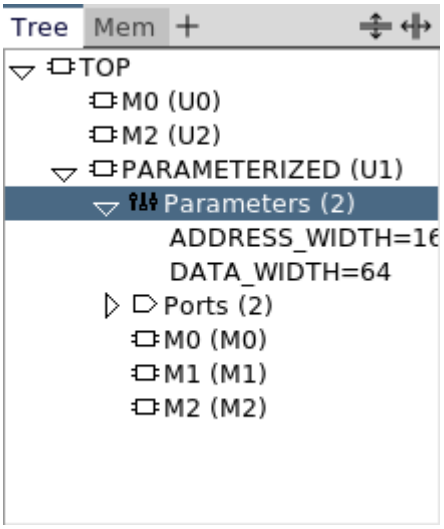
`@encrypted` Module Attribute

Modules with an `@encrypted` attribute are displayed with a red padlock to indicate that their contents are encrypted.



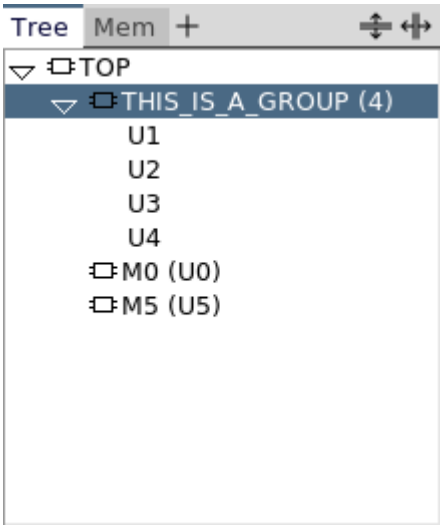
`@PARAMETERS` Cell Attribute

If a cell has a `@PARAMETERS` attribute, the attribute's value is displayed in a "Parameters" section in the Tree window. The RTL parser automatically adds this attribute for parameterized modules.



`@GROUP_NAME` Module Attribute

In the Tree window, instances flagged with the `group` flag will grouped in a special node "Custom Group". You can change this node's name by setting the parent module's `@GROUP_NAME`.



The Clock Domain Analyzer API

The CDC API

This Document describes the "Clock Domain Analyzer" extension to the "Database API". This API provides commands to get information about clock domains and clock domain crossings.

The following commands are available:

- $db cdc new
- $cdc getDomainCount
- $cdc getDomainList
- $cdc getSrc
- $cdc getAllSources
- $cdc getClkPinList
- $cdc getClkPinCount
- $cdc getReducedClkPinList
- $cdc getTreeList
- $cdc getReducedTreeList
- $cdc getClkPinDomains
- $cdc getTreeCrossCount
- $cdc getTreeCrossList
- $cdc getTreeCrossSrcList
- $cdc getTreeCrossClkPinList
- $cdc getTreeCrossCone
- $cdc calcDomainCross
- $cdc getDomainCrossCount

- $cdc getDomainCrossList
- $cdc getDomainCrossDomainPair
- $cdc getDomainCrossTrgList
- $cdc getDomainCrossCone
- $cdc free

Initialize

$db cdc new

```
$db cdc new $top ?-skipundriven? ?-cmpr? ?-skipless <int>?
```

The Clock Domain Analyzer API needs to be initialize to a top level module. During this initialization some options can be specified. This command will return a reference to the CDC in-memory structure later referred to as $cdc.

The following options are available:

`$top`	The OID of the top level module.
`-skipundriven`	The clock domain search will stop with an error message if it finds any undriven clock signal. Use this option to ignore such errors.
`-cmpr`	Comprehensive Mode: collects all possible clock domain sources for all FFs rather than trying to estimate the clock domain sources. As this will slow down the process and may create complex results it should be only enabled if clock domain violations are expected. As an advantage a list of all found clock domain crossings is created below the clock domain list.
`-skipless <int>`	Skip clock domains with less FFs than...: this option can be used to ignore small clock domains (e.g. asynchronous Flip-Flops).

Example

```
set cdc [$db cdc new $top]
```

Get Clock Domain Information

$cdc getDomainCount

```
$cdc getDomainCount
```

Get the number of clock domains.

Example

```
$cdc getDomainCount
```

$cdc getDomainList

```
$cdc getDomainList ?-incr? ?-decr?
```

Get list of clock domain indices, sorted by number of contained clockPins.

The following options are available:

-incr	Sort the result in increasing order ("smallest" items first).
-decr	Sort the result in decreasing order ("largest" items first).

Example

```
set domains [$cdc getDomainList]
```

Get Clock Domain Details

$cdc getSrc

```
$cdc getSrc $domainIdx
```

Get clock source driver pin/port for given clock domain index.

The following options are available:

$domainIdx	The clock domain index.

Example

```
set domainIdx [lindex $domains 0]
      set clkSrc    [$cdc getSrc $domainIdx]
```

$cdc getAllSources

```
$cdc getAllSources $domainIdx
```

Get all clock source driver pins/ports for given clock domain index.

The following options are available:

`$domainIdx`	The clock domain index.

Example

```
set domainIdx  [lindex $domains 0]
      set clkSources [$cdc getAllSources $domainIdx]
```

$cdc getClkPinList

```
$cdc getClkPinList $domainIdx
```

Get list of clock pins for given clock domain index.

The following options are available:

`$domainIdx`	The clock domain index.

Example

```
set clkPinList [$cdc getClkPinList $domainIdx]
```

$cdc getClkPinCount

```
$cdc getClkPinCount $domainIdx
```

Get the number of physical clock pins for given clock domain index.

The following options are available:

`$domainIdx`	The clock domain index.

Example

```
set clkPinCount [$cdc getClkPinCount $domainIdx]
```

$cdc getReducedClkPinList

```
$cdc getReducedClkPinList $domainIdx
```

Get reduced list of clock pins for given clock domain index for each hierarchy level only one clock pin connected to one clock net is returned.

The following options are available:

`$domainIdx`	The clock domain index.

Example

```
set redClkPinList [$cdc getReducedClkPinList $domainIdx]
```

$cdc getTreeList

```
$cdc getTreeList $domainIdx ?-nocache? ?-skipcontrollogic?
```

Get cone to all clock pins for one clock domain.

The following options are available:

`$domainIdx`	The clock domain index.
`-nocache`	Do not use a previously calculated clock domain, force the recalculation of the clock domain structure.
`-skipcontrollogic`	Skip control logic.

Example

```
set clkTree [$cdc getTreeList $domainIdx]
```

$cdc getReducedTreeList

```
$cdc getReducedTreeList $domainIdx ?-nocache? ?-skipcontrollogic?
```

Get reduced cone for one clock domain (see getReducedClkPinList).

The following options are available:

`$domainIdx`	The clock domain index.
`-nocache`	Do not use a previously calculated clock domain, force the recalculation of the clock domain structure.
`-skipcontrollogic`	Skip control logic.

Example

```
set clkTree [$cdc getReducedTreeList $domainIdx]
```

$cdc getClkPinDomains

```
$cdc getClkPinDomains $clkPinOid
```

Get list of clock source drivers (domains) for given clock pin.

The following options are available:

`$clkPinOid`	The clock pin OID.

Example

```
set clkPin    [lindex $clkPinList 0]
      set clkDomain [$cdc getClkPinDomains $clkPin]
```

$cdc getTreeCrossCount

```
$cdc getTreeCrossCount
```

Get number of clock tree crossings.

Example

```
$cdc getTreeCrossCount
```

$cdc getTreeCrossList

```
$cdc getTreeCrossList ?-incr? ?-decr? ?-srcincr? ?-srcdecr?
```

Get list of tree crossings indices, sorted by number of contained clk pins or clkSrc pins.

The following options are available:

`-incr`	Sort the result in increasing order ("smallest" items first).
`-decr`	Sort the result in decreasing order ("largest" items first).
`-srcincr`	Sort the result in increasing order of the clock source pins.
`-srcdecr`	Sort the result in decreasing order of the clock source pins.

Example

```
set treeCrossList [$cdc getTreeCrossList]
```

$cdc getTreeCrossSrcList

```
$cdc getTreeCrossSrcList $crossIdx
```

Get list of clock source driver for one tree crossing.

The following options are available:

`$crossIdx`	Tree crossing index.

Example

```
set crossIdx [lindex $treeCrossList 0]
      if {$crossIdx != {}} {
          $cdc getTreeCrossSrcList $crossIdx
      }
```

$cdc getTreeCrossClkPinList

```
$cdc getTreeCrossClkPinList $crossIdx
```

Get list of clock pins which are clocked by the multiple clock sources return by getTreeCrossSrcList.

The following options are available:

`$crossIdx`	Tree crossing index.

Example

```
if {$crossIdx != {}} {
          $cdc getTreeCrossClkPinList $crossIdx
      }
```

$cdc getTreeCrossCone

```
$cdc getTreeCrossCone $crossIdx
```

Get the cone with the tree crossing.

The following options are available:

`$crossIdx`	Tree crossing index.

Example

```
if {$crossIdx != {}} {
          $cdc getTreeCrossCone $crossIdx
      }
```

Get Clock Domain Crossings

$cdc calcDomainCross

```
$cdc calcDomainCross ?-removedrs? ?$domainIdxList?
```

Generate data structures and return domainCrossCount.

The following options are available:

`-removedrs`	Remove Dual-Rank-Synchronizer.
`$domainIdxList`	The list of clock domain indices.

Example

```
$cdc calcDomainCross
```

$cdc getDomainCrossCount

```
$cdc getDomainCrossCount
```

Get number of domain crossings.

Example

```
$cdc getDomainCrossCount
```

$cdc getDomainCrossList

```
$cdc getDomainCrossList ?-incr? ?-decr?
```

Get list of domain crossings indices, sorted by number of contained clk pins.

The following options are available:

`-incr`	Sort the result in increasing order ("smallest" items first).

`-decr` Sort the result in decreasing order ("largest" items first).

Example

```
set domainCrossList [$cdc getDomainCrossList]
```

$cdc getDomainCrossDomainPair

```
$cdc getDomainCrossDomainPair $domainCrossIdx
```

Get list of clock source driver for one domain crossing.

The following options are available:

`$domainCrossIdx` Index of the domain crossing.

Example

```
set domainCrossIdx [lindex $domainCrossList 0]
      $cdc getDomainCrossDomainPair $domainCrossIdx
```

$cdc getDomainCrossTrgList

```
$cdc getDomainCrossTrgList $domainCrossIdx
```

Get target list for given domain crossing.

The following options are available:

`$domainCrossIdx` Index of the domain crossing.

Example

```
$cdc getDomainCrossTrgList $domainCrossIdx
```

$cdc getDomainCrossCone

```
$cdc getDomainCrossCone $domainCrossIdx
```

Get cone for given domain crossing.

The following options are available:

`$domainCrossIdx` Index of the domain crossing.

Example

```
$cdc getDomainCrossCone $domainCrossIdx
```

Finalize

$cdc free

```
$cdc free
```

The free command can be used to clear all internal data structures.

Example

```
$cdc free
```

ZDB Service Functions

Overview

```
zdb source filename
```

Re-implementation of the Tcl source command. The input file is evaluated line by line. This can be helpful to source very large scripts, e.g. to fill the zdb with a large netlist.

```
zdb primFuncList
```

Return a list of all supported primitive functions.

```
    zdb profile init
    zdb profile reset
    zdb profile print
    zdb profile get
    zdb profile finit
```

The above commands control the built-in Tcl profiler. The "init" and "finit" sub-commands start respective stop a profile session. Use "reset" to start a new profile session after a previous initialization. With the "print" command a profiling table is printed as a debug message to the message interface. The "get" command returns a list of triplets containing the command name, the

number of calls and the total consumed time (in nanoseconds). This output can be used to create a custom profile report.

The Htree API

The Tcl Database API for Hierarchical Tree Commands.

$db htree

This command provides access to the hierarchical tree information of the current database.

$db htree foreachClass

This command iterates over all hierarchical children of the given startOid. This loop iterates only over the first instance of a module and ignores all instantiations beginning at the second instantiation.

Usage:

```
htree foreachClass startOid child type cellName instName body
```

Parameters:

`startOid`

The given startOid can be empty, a module OID or a inst OID of a module. If it is an inst oid, then the referenced module of the instance is used. If startOid is empty only the top modules are unrolled.

`child`

The loop variable "child" is set to module oids or to inst oids (whatever this foreachClass function decides - to get highest speed).

`type`

The loop variable type can be encoded bitwise, where `x` means don't care:

Value	Description
`x0`	current child is node
`x1`	current child is leave
`0x`	current child no brothers
`1x`	current child has brothers

`cellName`

The loop variable is set to cell name of the current child node.

`instName`

The loop variable is set to instance name of the child (leader of instance group). Empty if `startOid` is empty.

`body`

Executed body for each child.

Example:

```
set startOid {module top top}
$db htree foreachClass $startOid child type cellName instName {}
```

$db htree foreachInst

Iterate over all instance names which have the same reference module as the given instance. Iterates only over the instances which are located in the same parent module as the start instance.

Usage:

`htree foreachInst inst instName body`

Parameters:

`inst`

Start instance.

`instName`

The members's instance name.

`body`

Executed body for each instance.

Example:

```
$db htree foreachInst {inst top i1} instName {}
```

$db htree moduleOf

This command converts a node-id into a database object id (oid).

Usage:

```
htree moduleOf instance treeLeadName
```

Parameters:

`instance`

If this parameter is empty or a module OID the function return the value without conversion.

`treeLeadName`

Name of the tree leader table.

Example:

```
$db htree moduleOf {inst top i1} {inst top i1}
```

$db htree foreachNode

The "foreachNode" gets a database oid and must return all node-ids on the way from lowest node (that maps to the given oid) up to the top node.

Usage:

```
htree foreachNode oid node instName body
```

Parameters:

`oid`

The given $oid (must be a module-oid) is translated into a set of node-ids. If $oid is a null-string, then we just return without an error.

`node`

The node as returned by foreachClass (as child).

`instName`

The members's instance name.

`body`

Executed body for each node.

Example:

```
$db htree foreachNode oid node instName {}
```

$db htree portCount

Get the total number of external ports of the given module. The number is the sum of single bit ports which are not member of a bus and multi bit ports (portBuses).

Usage:

```
htree portCount module
```

Parameters:

`module`
: The ports are counted in this module.

Example:

```
$db htree portCount {module top top}
```

$db htree portCountByDir

Count the single bit ports which are not member of a bus and multi bit ports and group the results by direction.

Usage:

```
htree portCountByDir module resultArray
```

Parameters:

`module`
: The ports are counted in this module.

`resultArray`
: Array where the results is stored.

Example:

```
array set portCount {}
$db htree portCountByDir {module top top} portCount
```

$db htree netCount

Count the single bit nets which are not member of a bus and multi bit nets in the given module. Ignore all auto generated nets.

Usage:

```
htree netCount module
```

Parameters:

`module`
: The nets are counted in this module.

Example:

```
$db htree netCount {module top top}
```

$db htree instCount

Count the instances of modules and primitives in the given module.

Usage:

```
htree instCount module
```

Parameters:

`module`
: The instances are counted in this module.

Example:

```
$db htree instCount {module top top}
```

$db htree clockCount

Count the number of clocked instances in the given module. Members of tree groups are ignored.

Usage:

```
htree clockCount module
```

Parameters:

`module`
: The clocked instances are counted in this module.

Example:

```
$db htree clockCount {module top top}
```

$db htree operCount

Count the operator instances in the given module. Members of tree groups are ignored.

Usage:

```
htree operCount module
```

Parameters:

`module`
: The operator instances are counted in this module.

Example:

```
$db htree operCount {module top top}
```

$db htree primCount

Count the primitive instances in the given module. Clocked primitives and operators are ignored.

Usage:

`htree primCount module`

Parameters:

`module`
: The primitives instances are counted in this module.

Example:

```
$db htree primCount {module top top}
```

$db htree combinedCount

Get the detailed count of instances grouped by primitive, clocked and operator. Count all instances which tree group members separately. Returns tcl list with the order `{primitiveCount clockedCount operatorCount groupCount}`.

Usage:

`htree combinedCount module`

Parameters:

`module`
: The primitives instances are counted in this module.

Example:

```
$db htree combinedCount {module top top}
```

$db htree brotherCount

Count all instances of the same type in the containing module of the given instance.

Usage:

`htree brotherCount instance`

Parameters:

`instance`

Reference instance.

Example:

```
$db htree brotherCount {inst top i1}
```

List of API Examples

This document lists all Userware examples that can be found in the demo/api directory of the RTLvision PRO release package.

There are different possibilities to execute an Userware script:

- Load the Userware script file using the **File › Load Userware** menu entry.
- Execute the Tcl command `source <userware.tcl>` in the Console window, where `<userware.tcl>` is your Userware script.
- Start RTLvision PRO with commandline option `-userware <userware.tcl>`, where `<userware.tcl>` is your Userware script file. If you use this method, make sure that you register your script to be executed using the API call `gui database registerChangedCallback`.
- Use the Plugins dialog. Only scripts that follow certain rules (see demo/api/demoPlugin.tcl for an example) can be loaded as a Plugin. The API examples in the table below that can be loaded as a Plugin are marked with the icon.

Cds

`cust42/browseCdsLib.tcl`	BrowseCdsLib

Analyze the Loaded Database

`calculateArea.tcl`	Calculate Area
`cCoupling.tcl`	Coupling Capacitors
`cdcExample.tcl`	CDC Example
`checkDeviceModel.tcl`	Check Device Model
`clockTree.tcl`	Clock Tree Extraction
`configCGC.tcl`	Identify Clock Gates
`configureClockedCells.tcl`	Configure Clocked Cells
`createUnisimSymbols.tcl`	Create UNISIM Symbols
`cust1/heavyR.tcl`	Find Defined Resistors

cust1/patternSearch.tcl	Search for Interconnected Resistors
cust11/hireconvlogic.tcl	Find Reconvergent Logic
cust13/pathToIP.tcl	Find Paths to IP Blocks
cust14/hi2lo.tcl	Check Voltage Zones
cust22/checkOneLoad.tcl	Check Connectivity to IP Cells
cust22/directlyConnectedFlops.tcl	Detect Cells Directly Connected to Input
cust22/directlyConnectedFlops2.tcl	Detect Cells Directly Connected to Input 2
cust24/clockedWithoutReset.tcl	Find Clocked Cells Without Reset
cust27/colorNets.tcl	Color Nets
cust27/duplicateSubckt.tcl	Find Duplicate Sub-Circuits
cust27/relatedPGtoIO.tcl	Report Related P/G for each I/O
cust32/transistorStatistics.tcl	Transistor Device Statistics
cust35/pinSearch.tcl	Pin Search And Driver Extraction
cust35/switchCell.tcl	Switch Cell Representation
cust36/histogram.tcl	Histogram of W/L
deviceStatistics.tcl	Compute and Display Device Statistics
erc/runerc.tcl	Run ERC Checks
esd/runesd.tcl	Run ESD Checks
expandConnectivity.tcl	Expand Connectivity
findBlackbox.tcl	Find Blackboxes
findGuessedSupplyNets.tcl	Find Guessed Supply Nets
flatfloatNodes.tcl	Find Flat Floating Nodes
floatingGate.tcl	Find Floating Gates
floatingInputs.tcl	Find Floating Inputs
floatingNets.tcl	Find Floating Members in NetBuses
floatingNodes.tcl	Find Floating Nodes
floatingOutputs.tcl	Find Floating Outputs
heavyCR.tcl	Find Heavy Cs and Rs
heavyCRt.tcl	Find Heavy Cs and Rs (Tree-Based)
heavyNodes.tcl	Find Heavy Signals

hierarchyStatistics.tcl	Compute and Display Hierarchy Statistics
libertyArea.tcl	Liberty Area
listIP.tcl	Find IP Modules
multiDriverCheck.tcl	Check for Multiple Drivers
parasitic/debugSPEF.tcl	Debug SPEF
parasitic/filterParasitics.tcl	Filter Parasitic Elements
parasitic/linkToLayout.tcl	Link to Layout
parasitic/markNets.tcl	Mark Nets Without Parasitic Info
parasitic/parasiticStatistics.tcl	Show Parasitic Statistics
parasitic/pinToPinRes.tcl	Calculate the Pin-to-Pin Resistance
parasitic/showContributingCap.tcl	Show Parasitic Coupling Connections
parasitic/topCapNets.tcl	Top Capacitance Nets
parasitic/undriven.tcl	Find Parasitic Nets With No Driver
pathBetweenModules.tcl	Find Path Between Modules
reportConstantNets.tcl	Report Constant Nets
saveSelectedSymbols.tcl	Save Selected Symbols
showAllSDFInst.tcl	Find Insts Without SDF Info
wrongBulk.tcl	Check for Wrong Bulks
zeroDriverCheck.tcl	Check for Zero Drivers

Read Side Files and Annotate Data

cust1/readSideFile.tcl	Read Side File
cust2/showInstGroups.tcl	Read Group Information
cust10/faultVisu.tcl	Read Fault List
cust26/showReport.tcl	ShowReport
cust31/analyzerReport.tcl	Read ERC Analyzer Report
cust31/voltageCollision.tcl	Read Voltage Collision Report
fastscan/fastscan.tcl	Load FastScan Report
fsm.tcl	FSM
hyperfault/hyperfault.tcl	Load HyperFault Report
hyperfault/hyperfault_log.tcl	Load HyperFault Logfile

nanotime/nanotime.tcl	NanoTime
oem/bus85-primetime.tcl	Highlight and Annotate Timing Values
pathmill/pathmill.tcl	Parse PathMill Reports
primetime/primetime.tcl	PrimeTime
tempus/tempus.tcl	Tempus Timing Report

Modify the Loaded Database

analyzeBlackboxConnectivity.tcl	Analyze Blackbox Connectivity
createHier.tcl	Re-Create Hierarchy
cust1/allResistors1K.tcl	Set Resistors to Value
cust1/renameResistors.tcl	Rename Resistors
cust1/rmAllCapacitors.tcl	Remove All Capacitors
cust1/rmAllResistors.tcl	Remove All Resistors
cust5/postProcess.tcl	Move R/C Objects
cust7/editWL.tcl	Edit and Export W/L Attributes
cust11/flatModule.tcl	Flat Modules in the Design
cust27/mergeSerialVeriRes.tcl	Merge Serial Verilog Resistors
cust28/createTopBlock.tcl	Create Top Block
cust33/flatDesign.tcl	Flat the Design
deleteBuses.tcl	Delete NetBuses
deleteObjects.tcl	Delete Objects
deviceName.tcl	Additional Information in Device Name
eco.tcl	Engineering Change Order
fixSkillBusChars.tcl	Fix Bus Characters
fixSkillNamespace.tcl	Fix Colliding Names
flagLeafCells.tcl	Flag Leaf Cells
groupAnd.tcl	Group Connected And Gates
lutSymbols.tcl	LUT Symbols
makeBuses.tcl	Create Port- and Netbuses (Pattern Based)
makeBuses2.tcl	Create Port- and NetBuses
manipulateDesign/manipulateDesign.tcl	Manipulate Design
operpp.tcl	Beautify Schematics

parasitic/lump.tcl	Lump RC Modules
parasitic/pruneSPF.tcl	Prune SPF
removeHierarchy.tcl	Remove Hierarchy (Oper-Command)
renameObj.tcl	Manually Rename Objects
replaceAnd.tcl	Replace AND-Gates
replaceDevices.tcl	Replace Devices
setNetValue.tcl	Set Net Value
setPG.tcl	Find Power/Ground Nets
toggleBitblasted.tcl	Toggle Bitblasted
updateDiodes.tcl	Update Diodes Width and Height Parameters
wrapTops.tcl	Wrap Top Modules

GUI Specific Userware Examples

addDirectConnections.tcl	Add Direct Connections
configAtStartup.tcl	Set Preferences
cust22/coneConnectTwoInsts.tcl	Connect two Selected Instances in the Cone
cust22/coneExpandOneLogicLevel.tcl	Expand one Logic Level in the Cone
cust22/customizeVision.tcl	Initialize Customization
cust22/customSettings.tcl	Set Custom Settings at Tool Startup
cust39/highlightAndRemoveDevices.tcl	Highlight And Remove Devices
customPopup.tcl	Demo for API Tutorial
ERC_Cockpit.tcl	ERC Cockpit
graphicalMarks.tcl	Use Graphical Marks
groupCone.tcl	Group Cone Content
guiCustomization/customizeLayout.tcl	Customize Layout
guiCustomization/customizeMenu.tcl	Customize Menu
guiCustomization/customizePopup.tcl	Custom Popup
guiCustomization/customizeToolbar.tcl	Customize Toolbar
guiCustomization/customWidget.tcl	Custom Widget
oem/progressBar.tcl	Progress Bar Example
quickHiCol.tcl	Quick Highlight Color Change

toggleNetName.tcl	Custom Key Binding
unfoldHierarchy.tcl	Unfold Hierarchy in Schem

Create Reports

createOverview/createOverview.tcl	Create HTML/Text Overview
cust3/printPDF.tcl	Print Design as PDF
cust3/printSpice.tcl	Create Images from Spice
cust11/flopsAtIP.tcl	Find Paths to Clocked Cells
cust12/nomenclature.tcl	Create Nomenclature
cust13/trace.tcl	Trace Path to File
cust16/traceCell.tcl	Trace Cell
cust22/statistics.tcl	Collect Statistics
cust25/designMetrics.tcl	Dump Design Metrics as JSON
cust27/designHistogram.tcl	Design Histogram
cust33/reportAllInstanceParameters.tcl	Report All Instance Parameters
cust33/reportArrayInformation.tcl	Report Array Information
cust33/reportClockTrees.tcl	Report Clock Trees
cust33/reportDesignHierarchy.tcl	Report Design Hierarchy
cust33/reportFanInCone.tcl	Report Fan-In Cone
cust33/reportGateCount.tcl	Report Gate Count
cust33/reportInstanceParameters.tcl	Report Instance Parameters
cust33/reportLogicLoop.tcl	Report Logic Loop
cust33/reportSignalInformation.tcl	Report Signal Information
dumpHier.tcl	Dump Hierarchy
dumpIO.tcl	Dump I/O Ports
erc/example.tcl	ERC Usage Example
extractParam.tcl	Create Instance's Attribute List
parasitic/analyzeRC.tcl	Analyze RC Networks
saveHilight.tcl	Save/Restore Highlights
saveInstCount.tcl	Save Design Statistics

Miscellaneous Userware Examples

analogWave/create_attributes_for_dc.tcl	Create Attributes for DC
analogWave/create_attributes_for_tran.tcl	Create Attributes for Transient

analogWave/highlight_for_dc.tcl	Highlight for DC
analogWave/highlight_for_tran.tcl	Highlight for Transient
analogWave/print_functions.tcl	Functions to Print Data from the Analog Wave Database
analogWave/search.tcl	Search the Analog Waveform Database
analogWave/waveform_parser.tcl	Access the Analog Waveform Parser
batchExportSchematics.tcl	Headless Schematic Export
binlibExport.tcl	Export Binlib as TCL
binlibImport.tcl	Import Binlib from TCL
calculateNewWidth.tcl	Calculate New Width
commentGraphic.tcl	Annotate Cone
comments.tcl	Edit And Show Comments
coneKeepSelected.tcl	Keep Selected Cone Objects
coneStatus.tcl	Cone Status
configureMOSDisplayAttributes.tcl	Configure Attributes At MOS Devices
coupled.tcl	Coupled Elements
cust9/fpga-symbols.tcl	FPGA Symbols
cust11/extractCone2.tcl	Transistor Cone Extraction
cust13/traceThrough.tcl	Trace Through Cells
cust15/additionalCDC.tcl	Do CDC Checks
cust22/exportNetlistBatch.tcl	Batch Netlist Export
cust22/exportSkill.tcl	Automatically Map Symbols and Run Skill Export
cust22/saveView.tcl	Save/Restore Schematic Layout and Highlighting Colors
cust26/displayVoltage.tcl	Display Voltage
cust30/showLibertyTiming.tcl	Cross-Probe To Liberty Sources
cust32/hierGen.tcl	Hierarchical Netlist Generation
cust33/compileDesign.tcl	Compile a Design
cust34/highlightNodes.tcl	Highlight Nodes
cust38/excludeCell.tcl	Exclude Cells from the Cone Extraction
cust41/connectPG.tcl	Connect Pin to a PG Net

cust41/readPinList.tcl	Read a Pinlist from a File and Display the Path
demoPlugin.tcl	Demo Plugin
Diff/diff.tcl	Compute Differences
Diff/diff_nogui.tcl	DB Diff (Batch Mode Version)
exportMangled/exportMangled.tcl	Export Mangled/Obfuscated Netlist
extractCone.tcl	Cone Extraction Example
findInst.tcl	Find Transistor Devices by Name
getSymbol.tcl	Get Symbol String for Built-In Shapes
hierarchyOverview.tcl	Hierarchy Overview
highlightDevices.tcl	Highlight Devices
highlightPulldown.tcl	Highlight Pulldown
highlightsToCone.tcl	Load/Append Highlighted Objects into the Cone Window
hireddev.tcl	Highlight Reduced Devices
loadMembers.tcl	Load Members
logicCloud.tcl	Extract Clocked Elements
longPath.tcl	Find Longest Path
mkSym.tcl	Assign Symbols
navigateHier.tcl	Navigate Hierarchy
obfuscate/obfuscate.tcl	ObfuscateVerilog
oem/dummySpos.tcl	Add Source Code Highlight
oem/socket.tcl	Open Socket Connection
parasitic/createSpiceNetlist.tcl	Generate Spice From Parasitic
pathToInput.tcl	Path to Input
pathToPG.tcl	Path to P/G
permanentHighlight.tcl	Permanent Highlight Colors
preplace.tcl	Save/Restore Schematic Layout
readBookmark.tcl	Restore Bookmarks
recognizeGate.tcl	Pattern for Gate Recognition
restoreMem.tcl	Restore Contents of the Mem Window
saveConeAsSpice.tcl	Save Cone as Spice
saveNetsInCone.tcl	Save Nets in Cone

saveRestoreClockedCells.tcl	Save and Restore Configured Clocked Cells
saveRotation.tcl	Save/Restore Rotation and Orientation
showAllCells.tcl	Show All Cells
showFunc.tcl	Show Function
showICValues.tcl	Show IC Value
showInstComment.tcl	Show Instance Comment Attribute
showModelAttributes.tcl	Show Model Attributes
showMultiplier.tcl	Show Multiplier
showNetName.tcl	Show Selected Net Name
skill/skillClient.tcl	Send Skill Commands To Skill Server
skill/skillServer.tcl	Socket Server for Virtuoso
SkillExport.tcl	Skill Export
SkillExportBatch.tcl	Batch Skill Export
slib2skill.tcl	Convert Slib To Zdb And Show All Symbols
viewLiberty.tcl	Show Contents of Liberty
voltageZones.tcl	Voltage Zones
writeEDIF.tcl	Export Schematic as EDIF

Link to Other Tools

importSynplicity/importSynplicity.tcl	Import Synplicity Project
importXilinx/importXilinx.tcl	Import Xilinx Project
quartus/accessDB.tcl	Export Quartus DB as TCL
quartus/getNetlist.tcl	Quartus/TimeQuest Netlist Exporter
quartus/makeBinfile.tcl	Create Timing Netlist from TimeQuest
quartus/makePrj.tcl	Create Quartus Project
quartus/pathVision.tcl	Import Timings from TimeQuest
quartus/quartusLink.tcl	Quartus Link
quartus/server.tcl	Quartus Link Server
quartus/sta_report.tcl	TimeQuest Export Tool

Load Netlist Data

cust13/readTSV.tcl	Parse TSV Files
oem/alu.tcl	Stripped-Down BUS85
oem/bus85.tcl	BUS85 Example
oem/bus85gates.tcl	BUS85 Example (Gate Level)
oem/dac.tcl	CSIM90 Example
oem/ex_adder.tcl	Adder Example (Transistor Level)
oem/gm1.tcl	Transistor Example
oem/nand.tcl	Load NAND
oem/pchip.tcl	Load PMOS Example

C Level Examples

cust13/tsv2zdb/tsv2zdb.c	Parse Tab Separated Values
cust23/ndl2zdb.c	Parse NDL Data
wdb/wdbdemo.c	WDB API Demo
wdb/wdbdemo.tcl	Example for Wdb Gui API
zdb/createFlat.c	Create Flat Binary Database
zdb/createZdb.c	Create And Fill ZDB Via C-Level API
zdb/debugZdb.c	Example for Debug Messages
zdb/extract.c	Display Path Extraction Results
zdb/skeleton.c	C-Code Skeleton
zdb/userware.c	Example for C Userware
zdb/zdbDump.c	Dump Zdb to ASCII Text File

API Examples Details

Access the Analog Waveform Parser

This file contains an example for the use of the Analog Waveform library access functions.

The used functions allow to parse spice simulation results and to retrieve the contained data.

The comments in the example code give more explanations on the performed API function calls.

Section

Miscellaneous Userware Examples

Files

`analogWave/waveform_parser.tcl`

Example

`demo/spice/gl85.sp demo/spice/gl85.tr0`

Add Direct Connections

Add a new main menu entry to loop over all objects loaded to the Cone window and automatically load all direct connections between objects.

Plugin

Section

GUI Specific Userware Examples

Files

`addDirectConnections.tcl`

Example

`demo/spice/buf.sp`

Add Source Code Highlight

Create dummy objects and associate arbitrary file positions with these objects. Now any position in a source file can be highlighted.

Section

Miscellaneous Userware Examples

Files

`oem/dummySpos.tcl`

Example

`demo/rtl/aquarius/aquarius.f`

Adder Example (Transistor Level)

Small transistor example.

Section

Load Netlist Data

Files

`oem/ex_adder.tcl`

Additional Information in Device Name

Update the displayed name of all modules to show additional information.

Section

Modify the Loaded Database

Files

`deviceName.tcl`

Analyze Blackbox Connectivity

Analyze the connectivity of a blackbox instance and try to arrange the pins on the left or right side depending on the connectivity.

Plugin

Section

Modify the Loaded Database

Files

`analyzeBlackboxConnectivity.tcl`

Analyze RC Networks

Report the top 5 resistance values in the selected RC network. Usage:

1. Add the signals (turn on "Signal Mode") to investigate into the Mem window (e.g. by Drag&Drop).
2. Invoke the "Top 5 Resistors" menu entry to run the analysis.

Plugin

Section

Create Reports

Files

`parasitic/analyzeRC.tcl`

Example

`demo/spef/usb_phy.spef`

Annotate Cone

This is a prototype implementation of a Cone window annotation feature. To access this feature the main menu is extended by an `Annotate Cone` button.

The `Start Annotation` menu entry will activate the annotation mode and disables the normal mouse actions like zoom, double-click, etc. Now simple elements like rectangles, circles, lines and text boxes can be added on top of the schematic.

Annotation items can be moved or deleted using the corresponding menu entry. The `Clear Annotation` menu entry will remove all created annotation objects. The `End Annotation` entry will switch back to normal mode an re-install all normal mouse bindings. This will also remove all created annotation objects.

Plugin

Section

Miscellaneous Userware Examples

Files

`commentGraphic.tcl`

Example

`demo/rtl/aquarius/aquarius.f`

Assign Symbols

Assign a built-in symbol shape to a loaded primitive. The first character of the instance name defines the symbol type.

Section

Miscellaneous Userware Examples

Files

`mkSym.tcl`

Automatically Map Symbols and Run Skill Export

Open a design database, automatically map all symbols for all database cells matching by name, render the schematic for each module and export the schematic as a Skill file.

Section

Miscellaneous Userware Examples

Files

cust22/exportSkill.tcl

Batch Netlist Export

Read a System Verilog design and export the created netlist as a structural Verilog netlist.

Usage

```
starsh demo/api/cust22/exportNetlistBatch.tcl    \
    -rtl                  <READ_RTL_OPTIONS>     \
    -netlist outputFile.v <NETLIST_EXPORT_OPTIONS>
READ_RTL_OPTIONS:
    All supported options of the zrtl Tcl command.
NETLIST_EXPORT_OPTIONS:
    -namedConnectivity: Create named connectivity.
    -implementFunction: Add implementations for selected primitives.
    -gzip:              Write a gzipped output file.
    -writeSymlib:       Also write a .sym file.
    -writeLiberty:      Also write a .lib file.
```

Section

Miscellaneous Userware Examples

Files

cust22/exportNetlistBatch.tcl

Batch Skill Export

Export the schematic as Skill in batch mode using starsh.

Usage

```
starsh demo/api/SkillExportBatch.tcl \
    -spice <READ_SPICE_OPTIONS>      \
    -skill <SKILL_EXPORT_OPTIONS>
```

Section

Miscellaneous Userware Examples

Files

SkillExportBatch.tcl

Beautify Schematics

Call post process operators to beautify the schematic.

Section

Modify the Loaded Database

Files

`operpp.tcl`

Example

`demo/rtl/aquarius/aquarius.f`

BrowseCdsLib

Display the contents of a Cadence dds.lib file.

Plugin

Section

Cds

Files

`cust42/browseCdsLib.tcl`
`cust42/img/cell-16.png`
`cust42/img/cell-24.png`
`cust42/img/cell-32.png`
`cust42/img/cell-48.png`
`cust42/img/cell-64.png`
`cust42/img/layoutView-16.png`
`cust42/img/layoutView-24.png`
`cust42/img/layoutView-32.png`
`cust42/img/layoutView-48.png`
`cust42/img/layoutView-64.png`
`cust42/img/library-16.png`
`cust42/img/library-24.png`
`cust42/img/library-32.png`
`cust42/img/library-48.png`
`cust42/img/library-64.png`
`cust42/img/schematicView-16.png`
`cust42/img/schematicView-24.png`
`cust42/img/schematicView-32.png`
`cust42/img/schematicView-48.png`
`cust42/img/schematicView-64.png`
`cust42/img/symbolView-16.png`
`cust42/img/symbolView-24.png`

cust42/img/symbolView-32.png
cust42/img/symbolView-48.png
cust42/img/symbolView-64.png

BUS85 Example

Fill the database with the bus85 example design. The `bus85.tcl` example refers to the `bus85prim.sp` spice source. The `bus85prim.sym` Symbol library fits to the `bus85.tcl` example.

Section

Load Netlist Data

Files

`oem/bus85.tcl`
`oem/bus85prim.sp`
`oem/bus85prim.sym`

BUS85 Example (Gate Level)

Use this userware to load the `bus85.tcl` without transistors.

Section

Load Netlist Data

Files

`oem/bus85gates.tcl`

C-Code Skeleton

Skeleton that can be used to create a shared object that can be loaded into the *Vision tools to register new Tcl Object Commands.

Section

C Level Examples

Files

`zdb/skeleton.c`

Calculate Area

Add a main menu entry to calculate the estimated chip area.

Add a comment to the top module with the total area.

A text report with details about the chip area consumption can be created.

Plugin

Section

Analyze the Loaded Database

Files

`calculateArea.tcl`

Example

`demo/spice/alu8bit.sp`

Calculate New Width

Calculate new width based on the product of m and w.

Plugin

Section

Miscellaneous Userware Examples

Files

`calculateNewWidth.tcl`

Example

`demo/spice/aquarius.sp`

Calculate the Pin-to-Pin Resistance

Report the pin-to-pin resistance for two given pins on the same net without opening the Parasitic window. Select two pins on the same net and access this function from the extended Popup menu.

Plugin

Section

Analyze the Loaded Database

Files

`parasitic/pinToPinRes.tcl`

Example

`demo/dspf/example.dspf`

CDC Example

Userware example for the ClockTree API. Do additional CDC checks.

Section

Analyze the Loaded Database

Files

`cdcExample.tcl`

Check Connectivity to IP Cells

Loop over all output pins of all instances of the given start cell list and check that each output pin only connects to one instance of the given IP cell list.

Plugin

Section

Analyze the Loaded Database

Files

`cust22/checkOneLoad.tcl`

Check Device Model

Find and highlight io devices matching the given model name pattern.

Plugin

Section

Analyze the Loaded Database

Files

`checkDeviceModel.tcl`

Example

`demo/spice/multivolt.sp`

Check for Multiple Drivers

Loop over all signals and check if a signal has more than one driver.

Plugin

Section

Analyze the Loaded Database

Files

`multiDriverCheck.tcl`

Example

`demo/verilog/bus85.f`

Check for Wrong Bulks

Loop over all PMOS and NMOS transistors and check if their bulks are connected to power and ground respectively. If not, the transistor is reported in the Mem window.

Section

Analyze the Loaded Database

Files

`wrongBulk.tcl`

Example

`demo/spice/amd2901.sp demo/spice/csim90/g1310.sp`

Check for Zero Drivers

Loop over all signals and check if a signal has no driver.

Plugin

Section

Analyze the Loaded Database

Files

`zeroDriverCheck.tcl`

Example

`demo/rtl/wb_sys/conmax_top.f`

Check Voltage Zones

Find signals that connect transistors in a low voltage zone with transistors in a high voltage zone.

Plugin

Section

Analyze the Loaded Database

Files

`cust14/hi2lo.tcl`

Example

`demo/spice/multivolt.sp`

Clock Tree Extraction

Extract the clock tree structure of the loaded design.

Usage: - Right click on an input port (e.g. CLK) and invoke - "Userware→Clock Tree" from the context menu. - Now switch to the Cone window to see the result.

Plugin

Section

Analyze the Loaded Database

Files

`clockTree.tcl`

Example

`demo/rtl/aquarius/aquarius.f`

Collect Statistics

Traverse Design and collect statistics.

Section

Create Reports

Files

`cust22/statistics.tcl`

Color Nets

Color nets in the loaded database.

Plugin

Section

Analyze the Loaded Database

Files

`cust27/colorNets.tcl`

Example

`demo/spice/parity1.sp`

Compile a Design

Example Tcl script to compile all HDL design files into an in-memory (ZDB) database to be used with the Vision SDT platform. In the binary directory (linux64 or win64) of the Vision platform package you can find the starsh binary. This is a Tcl shell extended by all HDL parsers provided by the Vision platform. Either use the option -help to get a list of all possible options or see doc/parser/*parser.html for a more detailed description of the parser options.

Section

Miscellaneous Userware Examples

Files

`cust33/compileDesign.tcl`

Compute and Display Device Statistics

Compute and display statistics about the devices used in the design.

Plugin

Section

Analyze the Loaded Database

Files

`deviceStatistics.tcl`

Example

`demo/spice/gl85.sp`

Compute and Display Hierarchy Statistics

Compute and display statistics about the design hierarchy, e.g. number of instances of each cell, depth of hierarchical modules, etc. The statistics can also be exported as a CSV file.

The plugin script can also be run in batch mode via: starsh hierarchyStatistics.tcl -batch ZDB_FILE OUTPUT_FILE

Plugin

Section

Analyze the Loaded Database

Files

`hierarchyStatistics.tcl`

Example

`demo/rtl/aquarius/aquarius.f`

Compute Differences

Compute and display the differences of the currently loaded database and a binary database (.zdb file). Can be run directly from the command line using the command line options `-userware3 diff.tcl db1.zdb db2.zdb`.

Plugin

Section

Miscellaneous Userware Examples

Files

`Diff/diff.tcl`
`Diff/diff_common.tcl`

Example

`demo/api/Diff/netlistA.v demo/api/Diff/netlistB.v`

Cone Extraction Example

API example how to use the Cone extraction API: extract the logic cone or path between a start node and several targets.

Section

Miscellaneous Userware Examples

Files

`extractCone.tcl`

Example

`demo/rtl/aquarius/aquarius.f`

Cone Status

Add a main menu entry to highlight complete or empty modules loaded in the Cone window in different colors.

Plugin

Section

Miscellaneous Userware Examples

Files

`coneStatus.tcl`

Example

`demo/rtl/aquarius/aquarius.f`

Configure Attributes At MOS Devices

Configure which attributes are displayed at *MOS devices.

Section

Miscellaneous Userware Examples

Files

`configureMOSDisplayAttributes.tcl`

Example

`demo/spice/gl85.sp`

Configure Clocked Cells

Userware to flag cells as clocked cells and ports of clocked cells as the clock ports.

Section

Analyze the Loaded Database

Files

`configureClockedCells.tcl`

Example

`demo/verilog/gl85.v`

Connect Pin to a PG Net

Extend the Popup menu and add a new entry to disconnect the connected net and connect a constant 1 or 0 to the pin. It is also possible to restore the original connectivity.

Plugin

Section

Miscellaneous Userware Examples

Files

`cust41/connectPG.tcl`

Example

`demo/verilog/gl85.v`

Connect two Selected Instances in the Cone

Extend the popup menu and add an entry to add all direct connections between two selected instance in the Cone window.

Plugin

Section

GUI Specific Userware Examples

Files

`cust22/coneConnectTwoInsts.tcl`

Convert Slib To Zdb And Show All Symbols

Convert symbols from slib format to zdb and show all symbols.

```
starvisionpro -userware3 slib2skill.tcl slibfname libname
```

Section

Miscellaneous Userware Examples

Files

`slib2skill.tcl`

Coupled Elements

Add a main menu entry to collect information about all coupled K-objects and load all instances to the Mem window.

Coupled instances can be displayed in the Cone window including all connected R, C or L devices. If option `Full LRC` is set the corresponding R and C to the L are loaded into the cone, too.

Coupled instances can be grouped into a new hierarchy.

Plugin

Section

Miscellaneous Userware Examples

Files

`coupled.tcl`

Example

`demo/spice/examples.sp`

Coupling Capacitors

Report coupling Cs in the Mem window.

Plugin

Section

Analyze the Loaded Database

Files

`cCoupling.tcl`

Example

`demo/spice/amd2901.sp demo/spice/csim90/gm19.sp`

Create And Fill ZDB Via C-Level API

Create and fill a binary database using the C-Level API.

Section

C Level Examples

Files

`zdb/createZdb.c`

Create Attributes for DC

This example shows how to use the waveform parser to - read a DC analysis result file - traverse all curves defined in this file - create value attributes in the schematic from this data

- Load the spice netlist "demo/spice/Fig24_30/Fig24_30.sp".
- Load this userware file. A small GUI will open
- Open the DC simulation result (1) "demo/spice/Fig24_30/dcOpInfo.info"
- Select a subtype such as "bsim3v3.region".
- Click "Create"
- Check the attributes for the transistors in the schematic. Attributes such as "bsim3v3.region=1" have been added. (2) "demo/spice/Fig24_30/dcOp.dc"
- Select a type such as "volt".
- Click "Create"
- From the menu select "View→Select Attributes" and change "Net Attributes → Net" to "%volt"
- The generated attributes are visible on the nets of the schematic.

Section

Miscellaneous Userware Examples

Files

`analogWave/create_attributes_for_dc.tcl`

Example

```
demo/spice/gl85.sp demo/spice/gl85.tr0
```

Create Attributes for Transient

This example shows how to use the waveform parser to - read a transient analysis result file - traverse all curves defined in this file - create value attributes in the schematic from this data

- Load the spice netlist "demo/spice/Fig24_30/Fig24_30.sp".

- Load this userware file. A small GUI will open
- Open the tran simulation result "demo/spice/Fig24_30/Fig24_30.csv"
- Select a type such as "volt".
- Click "Create"
- From the menu select "View→Select Attributes" and change "Net Attributes → Net" to "%volt"
- The generated attributes are visible on the nets of the schematic.
- Change the x-axis value to 0.7e-6
- Click "Create"
- The attribute values are adjusted

Section

Miscellaneous Userware Examples

Files

`analogWave/create_attributes_for_tran.tcl`

Example

`demo/spice/gl85.sp demo/spice/gl85.tr0`

Create Flat Binary Database

Create and fill flat binary database using the C-Level API.

Section

C Level Examples

Files

`zdb/createFlat.c`

Create HTML/Text Overview

Create a design overview report of the loaded design either in HTML or ASCII text format. This example can easily be extended to create an other output format.

The HTML pages contain many typical information about the design, such as: module and submodule names with their instantiations, schematics, input/output/inout ports, clocked elements, all with the places of definition in the source files. Furthermore Operator types and Primitive functions are printed with the number of their occurrences.

Alternately you can create the same information in text files - without connecting links, of course.

In addition to the schematics in HTML, linked pdf files are generated; they have better zoom view options than the former.

DIRECTIONS:

1. Look through the BASIC OPTIONS in this `createOverview.tcl` file. If necessary, edit and save it.
2. Have this tcl file together with the included file `htmlFormat.tcl` (or `txtFormat.tcl`) stored in one directory.
3. Open a verilog design in StarVision PRO.
4. Click Menu-File-LoadUserware and navigate to `createOverview.tcl` from 2.
5. The createOverview.tcl script will create the depicted html (or txt) files in the specified $outDir. Start browsing with index.html.

Plugin

Section

Create Reports

Files

`createOverview/createOverview.tcl`
`createOverview/htmlFormat.tcl`
`createOverview/txtFormat.tcl`

Example

`demo/rtl/aquarius/aquarius.f`

Create Images from Spice

Load all Spice files in a given folder and create a PNG image of each sub-circuit.

Loop over spice files; read each spice file (by calling `zspice <options>`) and then load each sub-circuit into the Schem window and print it to postscript.

Terminology: ZDB uses the Verilog name Module for a Spice "sub-circuit" and the name Primitive for a Spice "model".

Section

Create Reports

Files

`cust3/printSpice.tcl`

Example

`demo/spice/adder.sp`

Create Instance's Attribute List

Print all attributes of all instances to a text file.

Plugin

Section

Create Reports

Files

`extractParam.tcl`

Example

`demo/spice/aquarius.sp`

Create Nomenclature

Create a list of instance names with their corresponding page and sector. If the instance has an attribute named 'PartValue' then the value of this attribute is added to the created instance list.

Plugin

Section

Create Reports

Files

`cust12/nomenclature.tcl`

Example

`demo/rtl/aquarius/aquarius.f`

Create Port- and NetBuses

Create portbuses and netbuses from single bit ports and nets.

Section

Modify the Loaded Database

Files

`makeBuses2.tcl`

Example

`demo/spice/aquarius.sp`

Create Port- and Netbuses (Pattern Based)

Create portbuses and netbuses from single bit ports and nets.

The script can be used as a template for creating special purpose versions. The hard coded pattern for collecting simple buses may need modifications. Currently the pattern for collecting single bits matches simple names like `A[1]`, `B{1}`, `C_1_` or `A1`.

Foreach module all nets are examined. If the name (e.g. `BUS[77]`) matches a pattern containing a basename (e.g. `BUS`), some letters preceding the bitsubscript (e.g. `[`) and some trailing letters (e.g. `]`) this triplet is used as an index identifying the bus. All bitsubscripts (e.g. `77`) are collected and sorted in a list. After all subscripts are collected, a netBus or portBus is created. For PortBuses there are no "gaps" in the bitsubscripts allowed. If a net is already member of a bus the bitsubscript list is cleared to mark the bus as already bundled.

Section

Modify the Loaded Database

Files

`makeBuses.tcl`

Example

`demo/spice/gl85.sp`

Create Quartus Project

This script can be used to create a Quartus project.

Section

Link to Other Tools

Files

`quartus/makePrj.tcl`

Create Timing Netlist from TimeQuest

A script that can be used to generate a zdb binfile of a timing netlist from TimeQuest.

This script can be loaded as a Userware in RTLvision PRO or as a script in the batch tool starsh. In both cases a zdb binfile is created based on the netlist in the Tcl file `<PROJECT>.rtlvision.tcl`.

Section

Link to Other Tools

Files

`quartus/makeBinfile.tcl`

Create Top Block

Create a new top block around all instances in the old top.

Plugin

Section

Modify the Loaded Database

Files

`cust28/createTopBlock.tcl`

Example

`demo/rtl/aquarius/aquarius.f`

Create UNISIM Symbols

Create symbols for cells with UNISIM-style names.

The function is activated by the new main menu entry `Userware > UNISIM Symbols`.

Plugin

Section

Analyze the Loaded Database

Files

`createUnisimSymbols.tcl`

Example

`demo/edif/gl85.edf`

Cross-Probe To Liberty Sources

Extend the popup menu with an entry `Show Liberty` to cross-probe from the schematic to Liberty source code.

Plugin

Section

Miscellaneous Userware Examples

Files

`cust30/showLibertyTiming.tcl`

Example

`demo/api/cust4/example.lib demo/api/cust4/example.v`

CSIM90 Example

API example of demo/spice/csim90 spice files.

Section

Load Netlist Data

Files

`oem/dac.tcl`

Custom Key Binding

This Userware binds a special hotkey to toggle the visibility of all netnames.

Section

GUI Specific Userware Examples

Files

`toggleNetName.tcl`

Custom Popup

Example code for doc/tutorial/guiCustomization.html

Section

GUI Specific Userware Examples

Files

`guiCustomization/customizePopup.tcl`

Custom Widget

Example code for doc/tutorial/guiCustomization.html

Section

GUI Specific Userware Examples

Files

guiCustomization/customWidget.tcl

Customize Layout

Example code for doc/tutorial/guiCustomization.html

Section

GUI Specific Userware Examples

Files

guiCustomization/customizeLayout.tcl

Customize Menu

Example code for doc/tutorial/guiCustomization.html

Section

GUI Specific Userware Examples

Files

guiCustomization/customizeMenu.tcl

Customize Toolbar

Example code for doc/tutorial/guiCustomization.html

Section

GUI Specific Userware Examples

Files

guiCustomization/customizeToolbar.tcl
guiCustomization/show-net-16.png
guiCustomization/show-net-24.png
guiCustomization/show-net-32.png
guiCustomization/show-net-48.png
guiCustomization/show-net-64.png

DB Diff (Batch Mode Version)

Compare two ZDBs. Differences are printed to STDOUT. If there are no differences, the script terminates with exit code 0, otherwise with exit code 1.

Usage:

```
starsh diff_nogui.tcl [OPTIONS...] <ZDB_FILE1> <ZDB_FILE2>
   --ignore-case
      Ignore case when comparing names.
   --ignore-equivalent-primitives
      Consider primitives with the same function to be equivalent.
   --normalize-arrays
      Normalize array/bus names.
   --skip-cells
      Don't report cells that only exist in one DB.
   --skip-connectivity
      Don't report connectivity differences (nets, netbuses, ...).
   --skip-instances
      Don't report instances that only exist in one DB.
   --skip-interfaces
      Don't report interface differences (e.g. ports, portbuses).
```

Section

Miscellaneous Userware Examples

Files

Diff/diff_nogui.tcl
Diff/diff_common.tcl

Debug SPEF

Internal SPEF debugging.

Plugin

Section

Analyze the Loaded Database

Files

parasitic/debugSPEF.tcl

Example

demo/spef/uart_layout.spef

Delete NetBuses

Delete all netBus information in the loaded database.

Plugin

Section

Modify the Loaded Database

Files

`deleteBuses.tcl`

Example

`demo/rtl/aquarius/aquarius.f`

Delete Objects

Extend the Popup menu to delete selected objects from the loaded database.

Plugin

Section

Modify the Loaded Database

Files

`deleteObjects.tcl`

Example

`demo/rtl/aquarius/aquarius.f`

Demo for API Tutorial

Extend the Popup menu to show all connections of a net/signal.

Section

GUI Specific Userware Examples

Files

`customPopup.tcl`

Example

`demo/spice/gl85.sp`

Demo Plugin

A demo plugin to demonstrate all features of the plugin API:

1. Register a callback function to change the display attributes on each database reload.
2. Add a menu item to print all module names to demo GUI modifications.
3. Add some configuration options for the plugin.

Plugin

Section

Miscellaneous Userware Examples

Files

`demoPlugin.tcl`

Example

`demo/rtl/aquarius/aquarius.f`

Design Histogram

Display a "Design Histogram" of the loaded design database.

Plugin

Section

Create Reports

Files

`cust27/designHistogram.tcl`

Example

`demo/spice/aquarius.sp`

Detect Cells Directly Connected to Input

Detect and highlight any FF/latches (with specific std cell name pattern) that is directly connected to the input ports (directly driven by input ports without any buffers/inverters).

Section

Analyze the Loaded Database

Files

`cust22/directlyConnectedFlops.tcl`

Detect Cells Directly Connected to Input 2

Detect and highlight any FF/latches (with specific std cell name pattern) that is directly connected to the input ports (directly driven by input ports without any buffers/inverters).

Section

Analyze the Loaded Database

Files

`cust22/directlyConnectedFlops2.tcl`

Display Path Extraction Results

This example script demonstrates how to display path extraction results generated by the extract C demo and stored as database root attributes in the Cone window.

Section

C Level Examples

Files

`zdb/extract.c`
`zdb/extract.tcl`

Display Voltage

Display the voltages in the given file at the nets.

Plugin

Section

Miscellaneous Userware Examples

Files

`cust26/displayVoltage.tcl`
`cust26/ram2k.ic`

Example

`demo/spice/ram2k.sp`

Do CDC Checks

Do additional CDC checks. The user can specify that input ports are in sync with a specific clock.

Section

Miscellaneous Userware Examples

Files

`cust15/additionalCDC.tcl`

Dump Design Metrics as JSON

Analyze a given ZDB and produce the following design metrics for each design (top level module):

- Module name
- Number of ports
- Number of input ports
- Number of output ports
- Number of bi-dir ports
- Number of instances in the module
- Number of nets in the module
- Number of registers
- Number and types of sub-module instances (including standard cells) instantiated.
- Length of scan-chains
- Number of clock domains

Section

Create Reports

Files

`cust25/designMetrics.tcl`

Dump Hierarchy

Dump hierarchy information to a text file.

Right click on a hierarchical instance or module to invoke the `Dump Hierarchy` function from the Userware Popup menu. This function will write hierarchy information starting from the selected object.

From the Userware main menu the `Dump Hierarchy Tree` function can be used to dump the hierarchy tree for each top module.

Plugin

Section

Create Reports

Files

`dumpHier.tcl`

Example

`demo/rtl/aquarius/aquarius.f`

Dump I/O Ports

Add a main menu entry to loop over all modules and dump the I/O ports to a text file.

Plugin

Section

Create Reports

Files

`dumpIO.tcl`

Example

`demo/rtl/aquarius/aquarius.f`

Dump Zdb to ASCII Text File

Dump the contents of the zdb database into an ASCII text file. This example do not need Tcl and can be used as a reference how to access the ZDB structures.

Section

C Level Examples

Files

`zdb/zdbDump.c`

Edit and Export W/L Attributes

Change all W/L attributes and update the Spice file.

Section

Modify the Loaded Database

Files

cust7/editWL.tcl

Example

demo/api/cust7/example.sp

Edit And Show Comments

This plugin extends the Popup menu to add comments to the selected object.

From the extended main menu a report can be displayed or saved as a text file.

Also, all objects with a comment can be loaded to the Mem window.

The plugin also lets the user toggle the display of comments at nets in the schematic view.

Plugin

Section

Miscellaneous Userware Examples

Files

comments.tcl

Example

demo/rtl/aquarius/aquarius.f

Engineering Change Order

Allow simple ECO changes.

This userware assist the user to do ECO changes. The Popup menu is extended to perform the following operations on insts/pins:

```
Disconnect         - Disconnect a pin from a net.
Connect            - Connect a pin to a net.
Change Connections - Change all connections of an instance.
Change Cellref     - Change the cell referenced by an instance.
Replace Instance   - Replace an instance by an instance of an other
                     cell.
```

For each operation a string with the performed action is printed to the Console window. Instead of this text a DC-script could be created to perform the ECO action on the netlist.

Section

Modify the Loaded Database

Files

`eco.tcl`

Example

`demo/verilog/gl85.v`

ERC Cockpit

Add various ERC checks to the main menu.

Section

GUI Specific Userware Examples

Files

`ERC_Cockpit.tcl`

Example

`demo/spice/amd2901.sp`

ERC Usage Example

Usage example for ERC:check02 "Floating Gates".

Section

Create Reports

Files

`erc/example.tcl`

Example for C Userware

C implementation of the floatingNodes Userware. This example script demonstrates how to use the userware extension compiled as a shared library in the *Vision GUI as well as in the starsh.

Section

C Level Examples

Files

`zdb/userware.c`
`zdb/userware.tcl`

Example for Debug Messages

Message and debug C-Level API.

Section

C Level Examples

Files

`zdb/debugZdb.c`

Example for Wdb Gui API

Shows the usage of the Gui Wave API functions and the extension of a wave database via the TCL API.

Section

C Level Examples

Files

`wdb/wdbdemo.tcl`

Example

`demo/rtl/aquarius/aquarius.f`

Exclude Cells from the Cone Extraction

Read the side file 'excludeCell.txt' and match the name patterns against the loaded database to flag cells exclude from the cone extraction.

Section

Miscellaneous Userware Examples

Files

`cust38/excludeCell.tcl`

Expand Connectivity

Expand the connectivity of devices in the Cone window excluding all the MOS devices that are connected to the net by the 'gate' pin.

Plugin

Section

Analyze the Loaded Database

Files

`expandConnectivity.tcl`

Example

`demo/spice/gl85.sp`

Expand one Logic Level in the Cone

Extend the popup menu and add an entry to expand the first logic level of all selected hierarchical instance in the Cone window.

Plugin

Section

GUI Specific Userware Examples

Files

`cust22/coneExpandOneLogicLevel.tcl`

Example

`demo/verilog/bus85.f`

Export Binlib as TCL

Since the binlib file format becomes incompatible if the database structure changes here are two scripts that can be used to migrate the contents of a compiled binlib to a new binlib version.

Run the old version of *Vision to export the binlib to Tcl:

```
starvisionpro -userware3 binlibExport.tcl oldBinlib.zdb /tmp/binlib.tcl
```

Run the new version of *Vision to import the Tcl and save a new version of the binlib:

```
starvisionpro -userware3 binlibImport.tcl /tmp/binlib.tcl newBinlib.zdb
```

Section

Miscellaneous Userware Examples

Files

`binlibExport.tcl`

Export Mangled/Obfuscated Netlist

Export a mangled/obfuscated version (stripped attributes, replaced names) of the currently displayed module or the whole database.

Plugin

Section

Miscellaneous Userware Examples

Files

`exportMangled/exportMangled.tcl`

Export Quartus DB as TCL

This script contains procedures to query netlist information from the Quartus database and writes out `$db load ⋯` statements.

Section

Link to Other Tools

Files

`quartus/accessDB.tcl`

Export Schematic as EDIF

Export the schematic as EDIF.

Section

Miscellaneous Userware Examples

Files

`writeEDIF.tcl`

Example

`demo/spice/gl85.sp`

Extract Clocked Elements

Extract logic cloud from all given start points to all clocked elements and load the result to the Cone window. Flip-flops belonging together are lined up in columns and get the same color.

Section

Miscellaneous Userware Examples

Files

`logicCloud.tcl`

Example

`demo/rtl/aquarius/aquarius.f`

Extract Path

Extract a path using the C-Level API.

Section

C Level Examples

Files

`zdb/extract.c`

Filter Parasitic Elements

This script can be used to filter parasitic elements of the RC network displayed in the Parasitic window. Sometimes the RC network even for a single net is too complicated and therefore when viewed for multiple nets, it can be very complicated to for debugging and visualization.

This script enables the user to:

1. Remove all the resistors or capacitors from a selected net in the RC window (NOT from the entire database).
2. Merge series resistors or parallel capacitors of a selected net.
3. Filter resistors or capacitors below a specified threshold value of a selected net.

Plugin

Section

Analyze the Loaded Database

Files

`parasitic/filterParasitics.tcl`

Example

`demo/dspf/example.dspf`

Find Blackboxes

Add a main menu entry to find all blackbox cells in the loaded database and show the result in the Mem window.

Plugin

Section

Analyze the Loaded Database

Files

`findBlackbox.tcl`

Example

`demo/rtl/aquarius/top.v`

Find Clocked Cells Without Reset

Find all instances of clocked cells and check the nets connected to the clock and reset pins.

Plugin

Section

Analyze the Loaded Database

Files

`cust24/clockedWithoutReset.tcl`

Find Defined Resistors

Traverse the design hierarchy tree for all Rs, check the resistance value, and collect the objects bigger than the defined threshold value. The result is stored in the Mem window.

Section

Analyze the Loaded Database

Files

`cust1/heavyR.tcl`

Example

`demo/spice/amd2901.sp`

Find Duplicate Sub-Circuits

Find duplicate sub-circuits loaded from a SPICE netlist.

Plugin

Section

Analyze the Loaded Database

Files

`cust27/duplicateSubckt.tcl`

Example

`demo/api/cust27/duplicateSubckt.sp`

Find Flat Floating Nodes

Loop over all nets/signals and count the number of pins for each net/signal. All nets/signals with exactly one pin are stored in the Mem window.

Plugin

Section

Analyze the Loaded Database

Files

`flatfloatNodes.tcl`

Example

`demo/spice/amd2901.sp`

Find Floating Gates

Loop over all instances and find floating gates. The result is stored in the Mem window.

Plugin

Section

Analyze the Loaded Database

Files

`floatingGate.tcl`

Example

demo/spice/gl85.sp

Find Floating Inputs

Loop over all signals and report signals that have no driver. Also loop over all primitive instance pins and report unconnected inputs.

Plugin

Section

Analyze the Loaded Database

Files

floatingInputs.tcl

Example

demo/spice/aquarius.sp

Find Floating Members in NetBuses

Loop over the database and find floating/dangling member nets in netbuses. Results are displayed in the Mem window.

Plugin

Section

Analyze the Loaded Database

Files

floatingNets.tcl

Example

demo/rtl/aquarius/aquarius.f

Find Floating Nodes

Loop over all nets/signals and count the number of pins for each net/signal. All nets/signals with exactly one pin are stored in the Mem window.

Plugin

Section

Analyze the Loaded Database

Files

`floatingNodes.tcl`

Example

`demo/spice/amd2901.sp demo/spice/csim90/gm1.sp`

Find Floating Outputs

Loop over all signals report signals that do not drive anything.

Plugin

Section

Analyze the Loaded Database

Files

`floatingOutputs.tcl`

Example

`demo/spice/amd2901.sp`

Find Guessed Supply Nets

Add a main menu entry to find all guessed supply nets in the loaded database and show the result in the Mem window.

Plugin

Section

Analyze the Loaded Database

Files

`findGuessedSupplyNets.tcl`

Example

`demo/spice/amd2901.sp`

Find Heavy Cs and Rs

The heavyCR procedure loops over all C or R and checks each value (capacitance or resistance) and collects the 10 objects with the biggest values. The 10 biggest Cs and 10 biggest Rs are stored in the Mem window. heavyCR is the tree-based version.

Plugin

Section

Analyze the Loaded Database

Files

`heavyCR.tcl`

Example

```
demo/spice/amd2901.sp demo/spice/csim90/g1310.sp demo/spice/csim90/gm1.sp demo/spice/gl85.sp
```

Find Heavy Cs and Rs (Tree-Based)

The heavyCR procedure loops over all C or R and checks each value (capacitance or resistance) and collects the 10 objects with the biggest values. The 10 biggest Cs and 10 biggest Rs are stored in the Mem window. heavyCRt is the tree-based version.

Plugin

Section

Analyze the Loaded Database

Files

`heavyCRt.tcl`

Example

```
demo/spice/amd2901.sp demo/spice/csim90/chip2.sp demo/spice/gl85.sp
```

Find Heavy Signals

Loop over all signals and count the number of pins for each signal. The 10 biggest signals are stored in the Mem window.

Plugin

Section

Analyze the Loaded Database

Files

`heavyNodes.tcl`

Example

`demo/spice/gl85.sp demo/spice/csim90/chip2.sp`

Find Insts Without SDF Info

Find all instances without SDF info. Load to mem or mark them.

Plugin

Section

Analyze the Loaded Database

Files

`showAllSDFInst.tcl`

Example

`demo/rtl/pulpino/pulpino.f`

Find IP Modules

Based on a given list of cell name patterns create an output file to list all instantiation paths to cells matching the given pattern list.

Section

Analyze the Loaded Database

Files

`listIP.tcl`

Find Longest Path

Extract and show the longest path between two selected objects.

Features:

- Define start and endpoint by selecting each object in the Schem window and press the corresponding button in the dialog window created by this userware.
- Pressing the "Run Search" button will perform a path extraction and display the result in a listbox. Each entry in this listbox can be double clicked to show it in the Cone window.
- There is a limit to 100000 paths, feel free to increase this value for fast machines.

- A progress bar appears which let you interrupt the path search process.

Section

Miscellaneous Userware Examples

Files

`longPath.tcl`

Example

`demo/rtl/aquarius/aquarius.f`

Find Parasitic Nets With No Driver

Find parasitic nets with no driver. A driver is either:

- a top level input port or
- an output pin or
- a pin with the name `Y`.

Section

Analyze the Loaded Database

Files

`parasitic/undriven.tcl`

Find Path Between Modules

Find all paths between the output pins of a given start module instance and any pin of a given target module instance.

Section

Analyze the Loaded Database

Files

`pathBetweenModules.tcl`

Example

`demo/rtl/aquarius/aquarius.f`

Find Paths to Clocked Cells

Read an input file that contains cell names and pin names. Then a path search is performed and the name of the first reachable clocked cell is dumped into an output file.

Section

Create Reports

Files

`cust11/flopsAtIP.tcl`

Example

`demo/spice/gl85.sp`

Find Paths to IP Blocks

Find paths from all nets of a given block to all IP blocks (flag undefined).

Section

Analyze the Loaded Database

Files

`cust13/pathToIP.tcl`

Find Power/Ground Nets

Flag all nodes that matches one of the given patterns with a power, negpower or ground flag to get a more readable schematic.

Section

Modify the Loaded Database

Files

`setPG.tcl`

Example

`demo/spice/gl85.sp`

Find Reconvergent Logic

Read a side file that contains pairs of FF pins. The task of this userware is to find paths with reconvergent logic between these pins. The first column in the file is used as the start pin the second column is used as the target pin.

Section

Analyze the Loaded Database

Files

`cust11/hireconvlogic.tcl`

Example

demo/spice/gl85.sp

Find Transistor Devices by Name

Search for transistor devices by name. The device might be moved to a new hierarchy created by the 'Recognize Gate' feature.

Section

Miscellaneous Userware Examples

Files

findInst.tcl

Example

demo/spice/gl85.sp

Fix Bus Characters

Rename net objects to follow the Cadence Virtuoso bus naming semantic.

Section

Modify the Loaded Database

Files

fixSkillBusChars.tcl

Example

demo/spice/aquarius.sp

Fix Colliding Names

Adjust instance and net names to satisfy the Cadence Virtuoso namespace rules. - Instances cannot contain a dot character. - Scalar net names and netbus names share the same namespace.

Section

Modify the Loaded Database

Files

fixSkillNamespace.tcl

Example

demo/spice/buf.sp

Flag Leaf Cells

Flag all modules listed in the provided side file as a leaf cell.

Plugin

Section

Modify the Loaded Database

Files

`flagLeafCells.tcl`

Example

```
demo/spice/gl85.sp demo/api/flagLeafCells.txt
```

Flat Modules in the Design

Flat all instances of a given module. For the simple case of only one hierarchy '$db oper expand' may be an simpler approach.

Section

Modify the Loaded Database

Files

`cust11/flatModule.tcl`

Example

```
demo/spice/param.sp
```

Flat the Design

Example Tcl script that uses ZDB API commands to flatten the design in the given binary ZDB file and write out a structural Verilog netlist of the flattened database. Usage: starsh flatDesign.tcl design.zdb design.v

Section

Modify the Loaded Database

Files

`cust33/flatDesign.tcl`

Floating Nodes Example

C implementation of the floatingNodes Userware. This example script demonstrates how to use the

userware extension compiled as a shared library in the *Vision GUI as well as in the starsh.

Section

C Level Examples

Files

`zdb/userware.c`
`zdb/userware.tcl`

FPGA Symbols

Create nice symbol shapes for all LUT cells.

Plugin

Section

Miscellaneous Userware Examples

Files

`cust9/fpga-symbols.tcl`

Example

`demo/edif/pulpino.edf`

FSM

Extract FSM information copied from the parse tree to the netlist database and display FSMs using a graphical and a table view.

Plugin

Section

Read Side Files and Annotate Data

Files

`fsm.tcl`

Example

`demo/rtl/wb_sys/conmax_top.f`

Functions to Print Data from the Analog Wave Database

Functions to print parsed data.

Section

Miscellaneous Userware Examples

Files

`analogWave/print_functions.tcl`

Example

`demo/spice/gl85.sp demo/spice/gl85.tr0`

Generate Spice From Parasitic

Read a text file containing names of hierarchical instances. For all nets inside these sub-circuits the RC network will be loaded to the Parasitic window and a Spice netlist is created.

Section

Miscellaneous Userware Examples

Files

`parasitic/createSpiceNetlist.tcl`

Get Symbol String for Built-In Shapes

Example code to extract the DEF string for the built-in symbol shapes.

Section

Miscellaneous Userware Examples

Files

`getSymbol.tcl`

Group Cone Content

Group the contents of the Cone window into an artificial hierarchy.

Plugin

Section

GUI Specific Userware Examples

Files

groupCone.tcl

Example

demo/rtl/aquarius/aquarius.f

Group Connected And Gates

Create artificial hierarchy around any two connected and2 gates in the database.

Section

Modify the Loaded Database

Files

groupAnd.tcl

Example

demo/verilog/gl85.v

Headless Schematic Export

Load a binary database (ZDB file), and create a graphics file for each module (SVG, PDF or POSTSCRIPT). This script works without a GUI and must be executed directly with starsh.

Usage:

```
starsh batchExportSchematics.tcl <ZDB_FILE> <FILE_TYPE> <TARGET_FOLDER>
```

Section

Miscellaneous Userware Examples

Files

batchExportSchematics.tcl

Hierarchical Netlist Generation

Read a Spice netlist with X/Y annotations for each device and a side file that contains polygon definitions then this Plugin can be used to re-create a hierarchical database.

Plugin

Section

Miscellaneous Userware Examples

Files

`cust32/hierGen.tcl`

Hierarchy Overview

Add a main menu entry to load all hierarchical instances into the Cone window.

Plugin

Section

Miscellaneous Userware Examples

Files

`hierarchyOverview.tcl`

Example

`demo/rtl/aquarius/aquarius.f`

Highlight and Annotate Timing Values

Userware example to highlight objects and annotate timing values.

Section

Read Side Files and Annotate Data

Files

`oem/bus85-primetime.tcl`

Highlight And Remove Devices

Highlight and remove PODE and DUMMY devices by model/cell name patterns. Highlight devices py parameter name patterns.

Section

GUI Specific Userware Examples

Files

`cust39/highlightAndRemoveDevices.tcl`

Highlight Devices

Loop over the loaded database and highlight different devices (based on the model name) in a different color.

Plugin

Section

Miscellaneous Userware Examples

Files

`highlightDevices.tcl`

Example

`demo/spice/gl85.sp`

Highlight for DC

This example shows how to use the waveform parser to - read a DC analysis result file - traverse all curves defined in this file - highlight elements in the schematic from this data

- Load the spice netlist "demo/spice/Fig24_30/Fig24_30.sp".
- Load this userware file. A small GUI will open
- Open the DC simulation result (1) "demo/spice/Fig24_30/dcOpInfo.info"
- Select a subtype such as "bsim3v3.region".
- Define a min / max value such as 2.0 and 2.0
- Click "Highlight"
- The concerned components are highlighted in the schematic. (2) "demo/spice/Fig24_30/dcOp.dc"
- Select a type such as "volt".
- Define a min / max value such as 0.0 and 0.01
- Click "Highlight"
- The concerned nets are highlighted in the schematic.

Section

Miscellaneous Userware Examples

Files

`analogWave/highlight_for_dc.tcl`

Example

`demo/spice/gl85.sp demo/spice/gl85.tr0`

Highlight for Transient

This example shows how to use the waveform parser to - read a DC analysis result file - traverse all curves defined in this file - highlight elements in the schematic from this data

- Load the spice netlist "demo/spice/Fig24_30/Fig24_30.sp".
- Load this userware file. A small GUI will open
- Open the tran simulation result "demo/spice/Fig24_30/Fig24_30.csv"
- Select a type such as "volt".
- Define a min / max value such as 0.0 and 0.01
- Click "Highlight"
- The concerned nets are highlighted in the schematic.

Section

Miscellaneous Userware Examples

Files

`analogWave/highlight_for_tran.tcl`

Example

`demo/spice/gl85.sp demo/spice/gl85.tr0`

Highlight Nodes

Read a side file that contains node names. Highlight all interconnected nodes in the same color.

Plugin

Section

Miscellaneous Userware Examples

Files

`cust34/highlightNodes.tcl`
`cust34/report.txt`

Example

`demo/spice/adder.sp`

Highlight Pulldown

Loop over the loaded database and highlight all pulldown devices and nets.

Plugin

Section

Miscellaneous Userware Examples

Files

`highlightPulldown.tcl`

Example

`demo/spice/ram2k.sp`

Highlight Reduced Devices

Loop over the loaded database and highlight reduced objects that are marked with the `$CONSISTINGOF` attribute (e.g. after `$db oper mergeParallel`).

Plugin

Section

Miscellaneous Userware Examples

Files

`hireddev.tcl`

Example

`demo/spice/asiclib.sp`

Histogram of W/L

Create a histogram of all W/L values in the database.

Plugin

Section

Analyze the Loaded Database

Files

`cust36/histogram.tcl`

Example

`demo/spice/gl85.sp`

Identify Clock Gates

Open a binfile and loop over all cells in the database and remove the clock flag from all CGC cells.

Section

Analyze the Loaded Database

Files

`configCGC.tcl`

Import Binlib from TCL

Since the binlib file format becomes incompatible if the database structure changes here are two scripts that can be used to migrate the contents of a compiled binlib to a new binlib version.

Run the old version of *Vision to export the binlib to Tcl:

```
starvisionpro -userware3 binlibExport.tcl oldBinlib.zdb /tmp/binlib.tcl
```

Run the new version of *Vision to import the Tcl and save a new version of the binlib:

```
starvisionpro -userware3 binlibImport.tcl /tmp/binlib.tcl newBinlib.zdb
```

Section

Miscellaneous Userware Examples

Files

`binlibImport.tcl`

Import Synplicity Project

Insert a Userware menu to import Synplicity project files.

Plugin

Section

Link to Other Tools

Files

`importSynplicity/importSynplicity.tcl`

Import Timings from TimeQuest

A Userware to visualize the results of the `sta_report.tcl` script.

This is a Userware to provide a widget to manage timing paths. The script `sta_report.tcl` executed in Altera's TimeQuest tool will generate an output file that uses this PathVision Userware.

Section

Link to Other Tools

Files

`quartus/pathVision.tcl`

Import Xilinx Project

Insert a Userware menu to import Xilinx Vivado and Xilinx ISE project files.

Just create the userware menu

```
$tool -userware  importXilinx.tcl
```

Create userware menu and immediately import `project.xpr`

```
$tool -userware2 importXilinx.tcl project.xpr
```

Create userware menu, immediately import `project.xpr` and create `project.set`

```
$tool -userware3 importXilinx.tcl project.xpr project.set
```

Inspired by code from the XML Parser YAXMLP in wiki.tcl.tk/20146 from George Peter Staplin

Plugin

Section

Link to Other Tools

Files

`importXilinx/importXilinx.tcl`

Initialize Customization

Customize *Vision at startup by loading multiple scripts.

Section

GUI Specific Userware Examples

Files

`cust22/customizeVision.tcl`

Keep Selected Cone Objects

Extend the Popup menu of the Cone window and add the option to keep the selected and delete all other objects.

Plugin

Section

Miscellaneous Userware Examples

Files

`coneKeepSelected.tcl`

Example

`demo/rtl/aquarius/aquarius.f`

Liberty Area

Add a new main menu entry to calculate the estimated chip area based on the Liberty area attribute.

Plugin

Section

Analyze the Loaded Database

Files

`libertyArea.tcl`

Link to Layout

Link the Parasitic view with a Layout viewer.

Plugin

Section

Analyze the Loaded Database

Files

`parasitic/linkToLayout.tcl`

Example

`demo/dspf/example.dspf`

Load FastScan Report

Parse a FastScan report file and display a pin list.

Plugin

Section

Read Side Files and Annotate Data

Files

`fastscan/fastscan.tcl`
`fastscan/bus85.fas`
`fastscan/bus85_small.fas`
`fastscan/minirisc.fas`

Example

`demo/verilog/bus85.f`

Load HyperFault Logfile

Parse and display a HyperFault report or log file.

Plugin

Section

Read Side Files and Annotate Data

Files

`hyperfault/hyperfault_log.tcl`
`hyperfault/bus85.flt`

Example

`demo/verilog/bus85.f demo/verilog/bus85.sym`

Load HyperFault Report

Parse and display a HyperFault report or log file.

Plugin

Section

Read Side Files and Annotate Data

Files

`hyperfault/hyperfault.tcl`
`hyperfault/bus85.dictionary`
`hyperfault/bus85_small.dictionary`

Example

`demo/verilog/bus85.f demo/verilog/bus85.sym`

Load Members

Extend the Popup menu with a function `Load Members Into Memory` that loads the members of the selected buses into the Mem window.

Plugin

Section

Miscellaneous Userware Examples

Files

`loadMembers.tcl`

Example

`demo/rtl/aquarius/aquarius.f`

Load NAND

Basic transistor example.

Section

Load Netlist Data

Files

`oem/nand.tcl`

Load PMOS Example

Another transistor example.

Section

Load Netlist Data

Files

`oem/pchip.tcl`

Load/Append Highlighted Objects into the Cone Window

Add main menu entries to load/append all highlighted objects/nets/instances into the main Cone window.

Plugin

Section

Miscellaneous Userware Examples

Files

`highlightsToCone.tcl`

Lump RC Modules

Merges parallel and serial transistors and resistors of parasitic networks.

Plugin

Section

Modify the Loaded Database

Files

`parasitic/lump.tcl`

Example

`demo/spef/usb_phy.spef`

LUT Symbols

Add a main menu entry to apply LUT/PLA symbols to LUT-style instances.

In order to be recognized as a LUT-style instance, there must be `n` inputs `(2⇐n⇐10)`, exactly 1

output, and the instance must have an attribute `INIT` with an exact bit-width of `2^n`.

Plugin

Section

Modify the Loaded Database

Files

`lutSymbols.tcl`

Example

`demo/edif/pulpino.edf`

Manipulate Design

Allow for deleting & creating hierarchies, rotating & moving transistor-level device instances, etc.

Plugin

Section

Modify the Loaded Database

Files

`manipulateDesign/manipulateDesign.tcl`
`manipulateDesign/annotate.tcl`
`manipulateDesign/appendcone.tcl`
`manipulateDesign/flatten.tcl`
`manipulateDesign/history_tab.tcl`
`manipulateDesign/merge.tcl`
`manipulateDesign/merge_tab.tcl`
`manipulateDesign/move.tcl`
`manipulateDesign/replace.tcl`
`manipulateDesign/rotate.tcl`
`manipulateDesign/search_tab.tcl`
`manipulateDesign/session.tcl`
`manipulateDesign/gl85-example.session`

Example

`demo/spice/gl85.sp`

Manually Rename Objects

Extend the Popup menu with a `Rename` entry that allows for renaming of the selected object.

Warning: bad names (e.g. empty names, names of an existing objects, names containing hierarchy separators) may result in an invalid data base.

Plugin

Section

Modify the Loaded Database

Files

`renameObj.tcl`

Example

`demo/rtl/aquarius/aquarius.f`

Mark Nets Without Parasitic Info

Mark all nets without parasitic info.

Plugin

Section

Analyze the Loaded Database

Files

`parasitic/markNets.tcl`

Example

`demo/dspf/example.dspf`

Merge Serial Verilog Resistors

Replace all modules matching any of the given name patterns in the provided side file as a resistor device. Then merge all serial resistor devices.

Plugin

Section

Modify the Loaded Database

Files

`cust27/mergeSerialVeriRes.tcl`

Example

`demo/api/cust27/mergeSerialVeriRes.v demo/api/cust27/mergeSerialVeriRes.txt`

Move R/C Objects

Scan the database for top level R/C objects with a period (.) in a connected node name and move that object down the hierarchy to the path specified in the node name.

Section

Modify the Loaded Database

Files

`cust5/postProcess.tcl`

Example

`demo/api/cust5/example.sp`

NanoTime

Parse a NanoTime report file and display all paths in a custom widget. Timing values are annotated in the schematic view. Individual paths can be displayed in the Cone window.

Usage:

After loading a NanoTime report, the tab 'NanoTime' shows the original report file, as well as the list of report items found in the file. Clicking a report item scrolls the report file to the corresponding position and highlights the path nodes of the report item.

The button 'Load Cone' loads the path nodes of the selected report item into StarVision's/SpiceVision's 'Cone window'. You can annotate the path with attribute values by checking some check boxes in the 'Show' section.

The button 'Load Mem' loads the path nodes into the 'Mem window'.

Checking 'Show Only Violating Items' updates the items list to only show report items which violate slack constraints; unchecking it will display all items.

Plugin

Section

Read Side Files and Annotate Data

Files

`nanotime/nanotime.tcl`
`nanotime/gl85.rpt`

Example

demo/spice/gl85.sp

Navigate Hierarchy

Extend the Popup menu to traverse the design hierarchy.

Plugin

Section

Miscellaneous Userware Examples

Files

navigateHier.tcl

Example

demo/rtl/aquarius/aquarius.f

ObfuscateVerilog

Obfuscates all names of the loaded design and writes Verilog code to obfuscated directory structure. Generates obfuscated filesets and an corresponding map to reverse the obfuscation (e.g. in a NanoTime report).

Plugin

Section

Miscellaneous Userware Examples

Files

obfuscate/obfuscate.tcl
obfuscate/obfuscatedNanoTimeReport_aquarius.rpt
obfuscate/hash.tcl
obfuscate/keywords.tcl

Example

demo/rtl/aquarius/aquarius.f

Open Socket Connection

This Userware example script establishes a socket connection between *Vision PRO (server) and an arbitrary Tcl-based application, e.g. tclsh (client). This file contains the code for both sides, the

server and the client side.

Try it out: Source this file into a *Vision PRO tool AND into a standard "tclsh" and type commands at the tclsh prompt like

```
set oid {inst gl85 C1 M2 U2 m10}
set db  [sv set db]
set val [sv $db flatattr $oid getMergedValue w]
```

or

```
sveval $script
```

The first executes `set db` and `$db flatattr $oid getMergedValue w` in the *Vision process and sets the result to the `val` variable, the second executes the contents of the `script` variable.

Section

Miscellaneous Userware Examples

Files

`oem/socket.tcl`

Parse NDL Data

Parse NDL files and create zdb. The proprietary netlist format is the gate-level design description language named 'Network Description Language' (NDL) used by LSI.

Section

C Level Examples

Files

`cust23/ndl2zdb.c`
`cust23/ndlparse.y`
`cust23/ndlscan.l`

Parse PathMill Reports

Parse a Pathmill report file and load the paths into the Cone window. On click, the values are displayed on the schematic.

Select Pathmill→Open Pathmill File (from main menu) and open the report file.

Plugin

Section

Read Side Files and Annotate Data

Files

`pathmill/pathmill.tcl`
`pathmill/compound.out`
`pathmill/compound.spi`
`pathmill/gl85.out`

Example

`demo/spice/gl85.sp`

Parse Tab Separated Values

Parse tab-separated-value (TSV) file into zdb database.

```
b   block_type
d   bus_name      bus_direction
i   instance_name block_type
n   is_global     net_name
p   instance_name port_name
```

where:

```
b is the  block_type/module_name
d is the  portbus_name, and its direction (input/output/bidir)
i is the  instance_name and the module type of an instance
          in module_name
n is the  net_name which may be global
p is the  port_name on inst_name which connects to the
          previous defined net_name. if inst_name is same as
          the previous module_name, the nets connects to
          interface port.
```

This is the C implementation of the `readTSV.tcl` userware. It is faster on processing large files but must be compiled be before usage.

Section

C Level Examples

Files

`cust13/tsv2zdb/tsv2zdb.c`

Parse TSV Files

Parse TSV (tab separated value) files and load the connectivity to zdb.

Parser for a tab-separated-value (TSV) file which looks like this:

```
b    block_type
d    bus_name       bus_direction
i    instance_name  block_type
n    is_global      net_name
p    instance_name  port_name
```

where:

```
b is the module_name
d is the port_name, and its direction (input/output/bidir)
i is the instance_name and the module type of an instance in module_name
n is the net_name which may be global
p is the port_name on an instance which connects to the last
  net_name being defined.
```

The C implementation `tsv2zdb.c` of this userware can be used to speed up the parsing process. The TCL implementation can be adjusted to the user's requirements without recompiling.

Section

Load Netlist Data

Files

`cust13/readTSV.tcl`
`cust13/tsv2zdb/demo.tsv`

Path to Input

Use the Cone Extraction API to find a path from a given start pin or node to primary input. If a cell contains only transistors then this cell is treated as a leaf cell, because the Cone Extraction cannot trace through transistors.

Plugin

Section

Miscellaneous Userware Examples

Files

`pathToInput.tcl`

Example

demo/rtl/aquarius/aquarius.f

Path to P/G

Extend the Popup menu to extract a path from a S, D or G pin of a transistor to power and ground. Go through resistors, inductors and D→S or S→D of NMOS or PMOS transistors.

Plugin

Section

Miscellaneous Userware Examples

Files

pathToPG.tcl

Example

demo/spice/gl85.sp

Pattern for Gate Recognition

Add more patterns to the gate recognition code.

Section

Miscellaneous Userware Examples

Files

recognizeGate.tcl
recognizeGate.sp

Permanent Highlight Colors

Demonstrate the usage more highlight colors using permanent highlight.

Section

Miscellaneous Userware Examples

Files

permanentHighlight.tcl

Pin Search And Driver Extraction

Search for hierarchical pins and extract the driver. The menu entry "Search Pins/Search" opens a dialog to search for hierarchical pins. Results are displayed in the "Mem" window. The "Mem" window's popup menu entry "Search Pins/Extract to Driver" searches for the driver of the selected pin and appends the result to the "Cone" window.

Plugin

Section

Analyze the Loaded Database

Files

`cust35/pinSearch.tcl`

PrimeTime

Parse a PrimeTime report file and display all paths in a dialog window. Timing values are annotated in the schematic view. Individual paths can be displayed in the Cone window.

Features:

- one file may have multiple paths (sets of faultlists)
- one path may have multiple pins
- for each pin, different attributes may be stored and displayed
- source pinlist will be displayed
- parsed pinlist can be loaded to Mem window
- paths can be loaded into Cone

NOTE	Only one faultlist file may be active the same time.

Plugin

Section

Read Side Files and Annotate Data

Files

`primetime/primetime.tcl`
`primetime/bus85.timing`
`primetime/bus85_single.timing`

Example

`demo/verilog/bus85.f`

Print Design as PDF

Print each module of the loaded design as a PDF file.

Section

Create Reports

Files

`cust3/printPDF.tcl`

Example

`demo/spice/adder.sp`

Progress Bar Example

Create a progress bar for a long running function.

Section

GUI Specific Userware Examples

Files

`oem/progressBar.tcl`

Prune SPF

General purpose script to prune any kind of loaded netlist data.

Plugin

Section

Modify the Loaded Database

Files

`parasitic/pruneSPF.tcl`

Example

`demo/dspf/example.dspf`

Quartus Link

RTLvision PRO userware to implement a GUI showing the use of `server.tcl` and `client.tcl`.

Start Quartus in the background, start a server inside Quartus and connect to this server (RTLvision is the client). Now a Quartus project can be created, the timing netlist can be transferred and the

timing paths can be queried from Quartus and visualized in RTLvision.

Section

Link to Other Tools

Files

`quartus/quartusLink.tcl`

Quartus Link Client

Tcl client for quartus socket server

> **IMPORTANT** The script 'quartus/server.tcl' is NOT a *Vision Userware.

It is a Tcl script that can be executed in Altera's Quartus/TimeQuest tools to evaluate Tcl commands in the Quartus environment. The script 'quartus/client.tcl' is a *Vision Userware and implements the client side.

Section

Link to Other Tools

Files

`quartus/server.tcl`
`quartus/client.tcl`

Quartus Link Server

> **IMPORTANT** The script `quartus/server.tcl` is NOT a *Vision Userware.

It is a Tcl script that can be executed in Altera's Quartus/TimeQuest tools to evaluate Tcl commands in the Quartus environment. The script `quartus/client.tcl` is a *Vision Userware and implements the client side.

Because quartus has no TCL main loop, vwait is used to process incoming data (and freeze gui) until Server:Shutdown is evaluated.

Section

Link to Other Tools

Files

`quartus/server.tcl`
`quartus/client.tcl`

Quartus/TimeQuest Netlist Exporter

Transfer the timing netlist from Quartus to a zdb tcl file.

Script that can be executed in TimeQuest to transfer the timing netlist.

IMPORTANT	This script is NOT a *Vision Userware.

It is a Tcl script that can be executed in Altera's Quartus/TimeQuest tools to export the netlist for RTLvision PRO as a Tcl Userware file. Run this script as:

```
quartus_sta -report=getNetlist.tcl <Project>
```

It will generate the file `<Project>.rtlvision.tcl` containing `$db load ⋯` commands. This file can be loaded as a Userware in RTLvision PRO.

Section

Link to Other Tools

Files

`quartus/getNetlist.tcl`

Quick Highlight Color Change

Show all highlight colors below the Console window. From there the user can select his favorite color. The visibility of the widget can be toggled from the Userware menu.

Section

GUI Specific Userware Examples

Files

`quickHiCol.tcl`

Re-Create Hierarchy

Re-create the hierarchy structure of a flat design based on the instance names using `$db oper addhier`.

Section

Modify the Loaded Database

Files

`createHier.tcl`

Example

`demo/verilog/bus85.f`

Read a Pinlist from a File and Display the Path

Read a pin list from a side file and load each pin into the Cone window.

Plugin

Section

Miscellaneous Userware Examples

Files

`cust41/readPinList.tcl`

Example

`demo/verilog/gl85.v`

Read ERC Analyzer Report

Parse a AnalyzerReport report file and display all paths in a custom widget.

Plugin

Section

Read Side Files and Annotate Data

Files

`cust31/analyzerReport.tcl`

Read Fault List

Read a faultlist sidefile and add a colored graphical mark at each pin of the read pinlist.

Input

Hierarchical Verilog netlist and a textfile with pinnames and error types (one per line).

Format of the report file:

```
O1  testchip_digital.DUT.U418.O     UD
O0  testchip_digital.DUT.U418.O     UD
I0  testchip_digital.DUT.U1367.I    EQ
O0  testchip_digital.DUT.U1366.O    EQ
O1  testchip_digital.DUT.U1357.O    UD
I1  testchip_digital.DUT.U1357.I    UD
```

First character of first field can be ignored. The second character of the first field is the type: can be either 0 or 1. The second field is the hierarchical pin name. Only lines with `UD` (undetected) as the third field are interesting and should be processed.

Result

Add a colored graphical mark at each pin of the pinlist, e.g. red for pins of type 0 and green for pins of type 1. Blocks in upper hierarchy display two values showing the number of type 0 and type 1 pins inside the hierarchy.

Section

Read Side Files and Annotate Data

Files

`cust10/faultVisu.tcl`
`cust10/faultlist`

Example

`demo/verilog/gl85.v`

Read Group Information

Read a text file that contains grouped instance information. All instances of a group are loaded into the Memory window and the Cone window.

Section

Read Side Files and Annotate Data

Files

`cust2/showInstGroups.tcl`
`cust2/input_file.txt`

Example

`demo/spice/csim90/chip2.sp`

Read Side File

Read a sidefile and display the values of the sidefile in the schematic window.

Section

Read Side Files and Annotate Data

Files

`cust1/readSideFile.tcl`

Read Voltage Collision Report

Parse a VoltageCollision report file and display all paths in a custom widget.

Plugin

Section

Read Side Files and Annotate Data

Files

`cust31/voltageCollision.tcl`

Remove All Capacitors

Traverse the database and remove all capacitors.

Section

Modify the Loaded Database

Files

`cust1/rmAllCapacitors.tcl`

Example

`demo/spice/amd2901.sp`

Remove All Resistors

Traverse the database and remove all resistors.

Section

Modify the Loaded Database

Files

`cust1/rmAllResistors.tcl`

Example

`demo/spice/amd2901.sp`

Remove Hierarchy (Oper-Command)

Remove all levels of hierarchy and create a flat version of the loaded design (using `$db oper rmhier`).

Section

Modify the Loaded Database

Files

`removeHierarchy.tcl`

Example

`demo/rtl/aquarius/aquarius.f`

Rename Resistors

Simple example to traverse the DB and rename all resistors primitives and instances.

Section

Modify the Loaded Database

Files

`cust1/renameResistors.tcl`

Replace AND-Gates

Replace any two connected and2 gates by one module.

Section

Modify the Loaded Database

Files

`replaceAnd.tcl`

Example

`demo/verilog/bus85.f`

Replace Devices

Extend the Popup menu to replace selected device-type instances by non-device instances to allow for rotation, moving, etc.

Plugin

Section

Modify the Loaded Database

Files

`replaceDevices.tcl`

Report All Instance Parameters

Example Tcl script that uses ZDB API commands to extract all instance parameters (default and non-default) and print a report to an output file. The script may get confused by "(* ... *)"-style Verilog attributes and parameters with the same name as internal Vision attributes.

Usage: starsh reportAllInstanceParameters.tcl design.zdb report.txt

Section

Create Reports

Files

`cust33/reportAllInstanceParameters.tcl`

Report Array Information

Example Tcl script that uses ZDB API commands to get information about arrays in the design. Usage: starsh reportArrayInformation.tcl design.zdb report.txt

Section

Create Reports

Files

`cust33/reportArrayInformation.tcl`

Report Clock Trees

Example Tcl script that uses ZDB API commands to get information about clock domains in a given design and print a report to an output file. Usage: starsh reportClockTrees.tcl design.zdb report.txt

Section

Create Reports

Files

`cust33/reportClockTrees.tcl`

Report Constant Nets

Loop over all nets of each module and search for constant values. For the found nets all connected pins are stored in the Mem window.

Plugin

Section

Analyze the Loaded Database

Files

`reportConstantNets.tcl`

Example

`demo/rtl/uar.v`

Report Design Hierarchy

Example Tcl script that uses ZDB API commands to get information about the design hierarchy in a given design and print a report to an output file. Usage: starsh reportDesignHierarchy.tcl design.zdb report.txt

Section

Create Reports

Files

`cust33/reportDesignHierarchy.tcl`

Report Fan-In Cone

Example Tcl script that uses Zdb API commands to get fanIn cones of a node to a DFF driver. Usage: starsh reportFanInCone.tcl design.zdb report.txt

Section

Create Reports

Files

`cust33/reportFanInCone.tcl`

Report Gate Count

Example Tcl script that uses Zdb API commands to extract instance count for hierarchical modules. Usage: starsh reportGateCount.tcl design.zdb report.txt

Section

Create Reports

Files

`cust33/reportGateCount.tcl`

Report Instance Parameters

Example Tcl script that uses ZDB API commands to extract non-default instance parameters and print a report to an output file. Usage: starsh reportInstanceParameters.tcl design.zdb report.txt

Section

Create Reports

Files

`cust33/reportInstanceParameters.tcl`

Report Logic Loop

Example Tcl script that uses Zdb API commands to get logic loops. Usage: starsh reportLogicLoop.tcl design.zdb report.txt

Section

Create Reports

Files

`cust33/reportLogicLoop.tcl`

Report Related P/G for each I/O

Extract the related power/gnd of all IO signals.

Plugin

Section

Analyze the Loaded Database

Files

`cust27/relatedPGtoIO.tcl`

Example

`demo/api/cust27/relatedPGtoIO.sp`

Report Signal Information

Example Tcl script that uses ZDB API commands to report all signals with the type (input, output, wire, reg, logic type of every electrical connection, sequential and combinational logic). Usage: starsh reportSignalInformation.tcl design.zdb report.txt

Section

Create Reports

Files

`cust33/reportSignalInformation.tcl`

Restore Bookmarks

Restore a bookmark file specified on the command line.

Section

Miscellaneous Userware Examples

Files

`readBookmark.tcl`
`aquarius.bm`

Example

`demo/rtl/aquarius/aquarius.f`

Restore Contents of the Mem Window

Open a file saved from the Mem window to restore the contents.

Plugin

Section

Miscellaneous Userware Examples

Files

`restoreMem.tcl`

Example

`demo/rtl/aquarius/aquarius.f`

Run ERC Checks

Load the ERC checks. - check01.tcl: Check that the gate pin of a MOSFET device is not connected to a

power or ground node.

- check02.tcl: Detect floating gates at the MOS devices, either directly or through pins of a resistor.
- check03.tcl: Check for wrong bulk connections.
- check04.tcl: Forward bias diode between power and ground.
- check05.tcl: Capacitor like devices between power and ground.
- check06.tcl: Floating input, power and ground ports.
- check07.tcl: Cell tie-off.

Plugin

Section

Analyze the Loaded Database

Files

`erc/runerc.tcl`
`erc/utils.tcl`
`erc/check01.tcl`
`erc/check02.tcl`
`erc/check03.tcl`
`erc/check04.tcl`
`erc/check05.tcl`
`erc/check06.tcl`
`erc/check07.tcl`
`erc/check14.tcl`

Example

```
demo/spice/ram2k.sp demo/spice/amd2901.sp
```

Run ESD Checks

Load the ESD checks.

Plugin

Section

Analyze the Loaded Database

Files

`esd/runesd.tcl`
`esd/utils.tcl`
`esd/check01.tcl`
`esd/check02.tcl`

esd/check03.tcl
esd/check04.tcl
esd/check05.tcl
esd/check06.tcl
esd/check07.tcl
esd/check08.tcl
esd/check09.tcl
esd/check10.tcl

Example

```
demo/api/esd/testcase01.sp demo/api/esd/testcase02.sp demo/api/esd/testcase03.sp demo/api/esd/testcase04.sp
demo/api/esd/testcase05.sp demo/api/esd/testcase06.sp demo/api/esd/testcase07.sp demo/api/esd/testcase08.sp
demo/api/esd/testcase09.sp demo/api/esd/testcase10.sp
```

Save and Restore Configured Clocked Cells

This plugin allows you to save manually configured clocked cell information to a text file and restore it from the file.

Plugin

Section

Miscellaneous Userware Examples

Files

saveRestoreClockedCells.tcl

Example

demo/verilog/gl85.v

Save Cone as Spice

Save Cone as Spice uses spos information at the instances loaded to the Cone window to do a "Copy and Paste" like Spice export. Works only for flat netlist.

Section

Miscellaneous Userware Examples

Files

saveConeAsSpice.tcl

Example

demo/spice/gl85.sp

Save Design Statistics

Save the design statistic (as displayed in the `Tools→Report InstCount` dialog) to a file.

Section

Create Reports

Files

`saveInstCount.tcl`

Example

`demo/rtl/aquarius/aquarius.f`

Save Nets in Cone

Add a main menu entry to save a text file with the names of all nets loaded in the Cone window.

Plugin

Section

Miscellaneous Userware Examples

Files

`saveNetsInCone.tcl`

Example

`demo/rtl/aquarius/aquarius.f`

Save Selected Symbols

Save a symlib file containing mappings for all cells with a symbol attribute.

Plugin

Section

Analyze the Loaded Database

Files

`saveSelectedSymbols.tcl`

Example

`demo/verilog/gl85.v`

Save/Restore Highlights

Traverse the loaded design database and collect all highlight information. The collected information is stored as an userware file that can be sourced to restore the highlights or loaded via the corresponding menu entry. The stored file contains: 0. Database call command 1. Used highlighting colors 2. List of module-based highlights 3. List of flat highlights 4. Code to import used colors 5. Code to import and create module-based highlights 6. Code to import and create flat highlights 7. If enabled as config option: Load objects of the highlights into the mem window 8. Code to restore highlights

Plugin

Section

Create Reports

Files

`saveHilight.tcl`

Example

`demo/rtl/aquarius/aquarius.f`

Save/Restore Rotation and Orientation

Save/Restore the module rotation and pin orientation

Plugin

Section

Miscellaneous Userware Examples

Files

`saveRotation.tcl`

Example

`demo/rtl/aquarius/aquarius.f`

Save/Restore Schematic Layout

Save/Restore the current schematic layout ('preplace' information) to/from a set of files.

Plugin

Section

Miscellaneous Userware Examples

Files

`preplace.tcl`

Example

`demo/rtl/aquarius/aquarius.f`

Save/Restore Schematic Layout and Highlighting Colors

This script combines: - preplace.tcl: save/restore the current schematic layout - saveHilight.tcl: save/restore highlight information - saveRotation.tcl: save/restore module rotation and pin orientation

Section

Miscellaneous Userware Examples

Files

`cust22/saveView.tcl`

Example

`demo/rtl/aquarius/aquarius.f`

Search for Interconnected Resistors

Search the loaded database to extract lists of interconnected resistors.

Section

Analyze the Loaded Database

Files

`cust1/patternSearch.tcl`

Search the Analog Waveform Database

Example to parse spice simulation results and to retrieve the contained data.

Section

Miscellaneous Userware Examples

Files

`analogWave/search.tcl`

Example

`demo/spice/gl85.sp demo/spice/gl85.tr0`

Send Skill Commands To Skill Server

Send skill commands to skill server. Configure host name and port number to match the skill server. The Skill server script needs to be loaded into the Cadence Virtuoso environment.

Plugin

Section

Miscellaneous Userware Examples

Files

`skill/skillClient.tcl`

Set Custom Settings at Tool Startup

Configure *Vision at startup by setting `global settings`. A mapping between the options in the Preferences dialog and the corresponding `global settings` can be found in the Preferences dialog section of the reference manual (doc/manual.html#Dialog).

Section

GUI Specific Userware Examples

Files

`cust22/customSettings.tcl`

Set Net Value

Extend the Popup menu with functions to set value of selected nets to `0`/`1`/`X`/`Z` or flag them as `power`/`negpower`/`ground`.

Plugin

Section

Modify the Loaded Database

Files

`setNetValue.tcl`

Example

`demo/spice/gl85.sp`

Set Preferences

Configure *Vision at startup by setting `global settings`. A mapping between the options in the Preferences dialog and the corresponding `global settings` can be found in the Preferences dialog section of the reference manual (doc/manual.html#Dialog).

Section

GUI Specific Userware Examples

Files

`configAtStartup.tcl`

Set Resistors to Value

Traverse the database and set the value of all resistors to 1k.

Section

Modify the Loaded Database

Files

`cust1/allResistors1K.tcl`

Show All Cells

Add a main menu entry to load an instance of each cell into the Cone window.

Plugin

Section

Miscellaneous Userware Examples

Files

`showAllCells.tcl`

Example

`demo/rtl/aquarius/aquarius.f`

Show Contents of Liberty

Read the given Liberty file, create a top module and load an instance of each Liberty cell.

Usage:

```
starvisionpro -userware2 viewLiberty.tcl <LIBERTY_FILE.lib>
```

Section

Miscellaneous Userware Examples

Files

`viewLiberty.tcl`

Example

`demo/liberty/gl85.lib`

Show Function

Add a main menu entry to toggle the display of the function at each primitive instance.

Plugin

Section

Miscellaneous Userware Examples

Files

`showFunc.tcl`

Example

`demo/rtl/aquarius/aquarius.f`

Show IC Value

Display IC values coming from the `.IC`, `.DCVOLT` or `.NODESET` Spice commands as net attributes in the schematic view.

Plugin

Section

Miscellaneous Userware Examples

Files

showICValues.tcl

Example

demo/spice/aquarius.sp demo/spice/aquarius.ic

Show Instance Comment Attribute

Show special instance comment attribute (`$ICOMMENT`) at MOS devices created by the Spice parser in the schematic view. Each instance with this attribute set can be added to the Mem window. The Spice parser option `-icomment` needs to be enabled to get the `$ICOMMENT` attribute at instances.

Plugin

Section

Miscellaneous Userware Examples

Files

showInstComment.tcl

Example

demo/spice/multivolt.sp

Show Model Attributes

Add a main menu entry to toggle the display of the model name and all model attributes at each transistor device.

Plugin

Section

Miscellaneous Userware Examples

Files

showModelAttributes.tcl

Example

demo/spice/gl85.sp

Show Multiplier

Add a new display attribute to show the multiplier value as an instance attribute in the schematic

view.

Plugin

Section

Miscellaneous Userware Examples

Files

`showMultiplier.tcl`

Example

`demo/spice/aquarius.sp`

Show Parasitic Coupling Connections

Extend the Popup menu in the Parasitic window to show the contributing capacitance of coupled nets.

Plugin

Section

Analyze the Loaded Database

Files

`parasitic/showContributingCap.tcl`

Show Parasitic Statistics

Show parasitic statistics.

Plugin

Section

Analyze the Loaded Database

Files

`parasitic/parasiticStatistics.tcl`

Example

`demo/dspf/example.dspf`

Show Selected Net Name

Extend the Popup menu and add a function to show the net name only at the selected nets.

Plugin

Section

Miscellaneous Userware Examples

Files

`showNetName.tcl`

Example

`demo/rtl/aquarius/aquarius.f`

ShowReport

Read a report file and display all objects in a dialog window. Selecting a line of the report file will goto the object.

Plugin

Section

Read Side Files and Annotate Data

Files

`cust26/showReport.tcl`
`cust26/showReport.txt`

Example

`demo/spice/parity1.sp`

Skill Export

Export the schematic as Skill. SkillExport Userware example: this Userware can be used to export a design as Skill in batch mode, e.g. start SpiceVision with the following commandline options:

```
spicevisionpro -userware ../demo/api/SkillExport.tcl                \
    -symlib ../demo/spice/parity1.sym -symlib /path/to/analogLib.sym \
    ../demo/spice/parity1.sp
```

The design `parity1.sp` is read into the SpiceVision database, all primitive symbols are mapped as specified in the spice lines of your symbol library `analogLib.sym`. All sub-circuit symbols are mapped

from demo/spice/parity1.sym. If you don't know how to create your symbol library (analogLib.sym) and how to map the symbols, then please refer to the doc/skillExportTutorial.html.

After the Skill file is written SpiceVision will quit.

Section

Miscellaneous Userware Examples

Files

SkillExport.tcl

Example

demo/spice/parity1.sp demo/spice/parity1.sym

Socket Server for Virtuoso

Socket Server started from skillServer.il

This file should be installed in the same directory as skillServer.il and needs **execute permission**

It needs path access to a working tclsh interpreter.

Section

Miscellaneous Userware Examples

Files

skill/skillServer.tcl

Stripped-Down BUS85

Stripped-down version of bus85.tcl

Stripped down by:

```
$db foreach module m {
    if {[$db refCount $m] == 0} {
        puts $m
    }
}
```

Section

Load Netlist Data

Files

oem/alu.tcl

Switch Cell Representation

Provide different views for insts. Requires the design to be read with '-resolveDuplicates off' which automatically rename the cells and set the required attributes for this plugin.

Plugin

Section

Analyze the Loaded Database

Files

`cust35/switchCell.tcl`

Example

`demo/rtl/aquarius/aquarius.f demo/spice/aquarius.sp demo/spice/aquarius.sym`

Tempus Timing Report

Parse a Tempus report file and display all paths in a custom window. Timing values are annotated in the schematic view. Individual paths can be displayed in the Cone window.

Plugin

Section

Read Side Files and Annotate Data

Files

`tempus/tempus.tcl`
`tempus/timingpaths.txt`

Example

`demo/verilog/bus85.f`

TimeQuest Export Tool

Get STA timing report from TimeQuest.

IMPORTANT	This script is NOT a RTLvision PRO Userware.

It is a Tcl script that can be executed in Altera's TimeQuest tool to generate an Userware to visualize timing paths.

Run this script as:

```
quartus_sta -report=sta_report.tcl <Project>
```

It will generate the file `<Project>.rtlvision.tcl` that can be loaded as a Userware in conjunction with the PathVision Userware.

Section

Link to Other Tools

Files

`quartus/sta_report.tcl`

Toggle Bitblasted

Extend the Popup menu with a `Toggle Bitblasted` entry that allows for exchanging buses in the selected instances by bitblasted nets, and vice versa.

Plugin

Section

Modify the Loaded Database

Files

`toggleBitblasted.tcl`

Example

`demo/rtl/aquarius/aquarius.f`

Top Capacitance Nets

Report the top total capacitance nets in the design.

Plugin

Section

Analyze the Loaded Database

Files

`parasitic/topCapNets.tcl`

Example

`demo/dspf/example.dspf`

Trace Cell

Find all possible paths from all primary input ports of all loaded modules to all internal nodes and to all primary output ports. Also find paths from all internal nodes to all primary output ports. The result can be visualized in the Cone window and in addition it is written to an ASCII file.

Section

Create Reports

Files

`cust16/traceCell.tcl`

Trace Path to File

Userware script that reads in a configuration file containing names of start net/nodes and target cells. Search paths between the start net and the target cell and dump this path into an output file.

Section

Create Reports

Files

`cust13/trace.tcl`

Trace Through Cells

Flag cells as feed through cells. A double click in the Cone window will go through these cells.

Section

Miscellaneous Userware Examples

Files

`cust13/traceThrough.tcl`

Example

`demo/verilog/bus85.f`

Transistor Cone Extraction

Do a special cone extraction on a transistor netlist.

Section

Miscellaneous Userware Examples

Files

`cust11/extractCone2.tcl`

Example

`demo/spice/gl85.sp`

Transistor Device Statistics

Report all the unique combinations for L and W in the design to gain insight into how many different combinations of L and W transistors exist in the design and where the different combinations are in the hcells.

Plugin

Section

Analyze the Loaded Database

Files

`cust32/transistorStatistics.tcl`

Transistor Example

Small transistor example from `demo/spice/csim90`.

Section

Load Netlist Data

Files

`oem/gm1.tcl`

Unfold Hierarchy in Schem

Unfold the visible hierarchy modules in the Schem window.

Plugin

Section

GUI Specific Userware Examples

Files

`unfoldHierarchy.tcl`

Example

`demo/rtl/aquarius/aquarius.f`

Update Diodes Width and Height Parameters

Update all diode instances and add the width and height parameters according to the area and perimeter parameters.

Plugin

Section

Modify the Loaded Database

Files

`updateDiodes.tcl`

Use Graphical Marks

Demonstrate the usage of attributes and graphical marks.

Section

GUI Specific Userware Examples

Files

`graphicalMarks.tcl`

Voltage Zones

Highlight power nets with different names or different voltage values in a different color.

Plugin

Section

Miscellaneous Userware Examples

Files

`voltageZones.tcl`

Example

`demo/spice/multivolt.sp`

WDB API Demo

Create and fill wdb.

Section

C Level Examples

Files

`wdb/wdbdemo.c`

Wrap Top Modules

Add a main menu entry to create a wrapper module that instantiates all top cells in an artificial new top module.

Plugin

Section

Modify the Loaded Database

Files

`wrapTops.tcl`

Example

`demo/spice/asiclib.sp`

Utility and Service Functions

Platform Dependent System Functions

The zos command provides platform dependent service functions.

zos configdir

Get the configuration directory.

Usage:

```
zos configdir ?-legacy?
```

Parameters:

`-legacy` (optional)
: Get the legacy name of the config directory.

Example:

```
set config [zos configdir]
```

zos convertFilename

Convert paths and filenames from relative to absolute and vice versa.

Usage:

```
zos convertFilename filename mode ?dir?
```

Parameters:

`dir` (optional)
: The conversion is relative to the given directory "dir" (dir must be absolute) or if dir is not given, then the conversion is relative to the current working directory.

`filename`
: The file to convert.

`mode`

The convert mode.

Example:

```
# make an absolute filename relative to a given dir
# => my/file.txt
zos convertFilename "/my/path/my/file.txt" relative "/my/path"

# make a relative filename absolute
# => /my/test.txt
zos convertFilename "../test.txt" absolute "/my/path"
```

zos convertFilename2

Converts a given filename into relative-to-todir.

Usage:

`zos convertFilename2 filename fromdir ?todir?`

Parameters:

`filename`

The file to convert.

`fromdir`

The given filename is relative to the fromdir.

`todir (optional)`

The given filename is converted relative to the todir.

Example:

```
# => ../my/path/file.txt
zos convertFilename2 "demo/file.txt" "/my/path" "/other"
```

zos currentCputime

Can be used for cpu time measurement. The returned time value is in milliseconds. The option -ns can be used to return nanoseconds.

Usage:

```
zos currentCputime ?-ns?
```

Parameters:

`-ns (optional)`
: Nanoseconds

Example:

```
set ns1 [zos currentCputime -ns]
# do some expensive work
set ns2 [zos currentCputime -ns]
set delta [expr {$ns2 - $ns1}]
puts "The expensive work took ${delta}ns CPU time."
```

zos currentLocalTime

Return the current time.

Usage:

```
zos currentLocalTime
```

Parameters:

No parameters.

Example:

```
set time [zos currentLocalTime]
set timeStr [clock format $time -format "%Y-%m-%d %T"]
puts $timeStr
```

zos currentLocalTimeString

Return the current time as a string.

Usage:

```
zos currentLocalTimeString
```

Parameters:

No parameters.

Example:

```
set timeStr [zos currentLocalTimeString]
puts $timeStr
```

zos currentRuntime

Can be used for runtime measurement. The returned time value is in milliseconds. The option -ns can be used to return nanoseconds.

Usage:

```
zos currentRuntime ?-ns?
```

Parameters:

`-ns (optional)`
: Nanoseconds

Example:

```
set ms1 [zos currentRuntime]
# do some expensive work
set ms2 [zos currentRuntime]
set delta [expr {$ms2 - $ms1}]
puts "The expensive work took ${delta}ms wall clock time."
```

zos dirname

Return the directory name of the given filename.

Usage:

```
zos dirname filename
```

Parameters:

`filename`

The filename.

Example:

```
# => /my/path
zos dirname /my/path/file.txt
```

zos expandenv

Expand environment variables in the given string.

The variable names are terminated by any character other than "[A-z][0-9]_". But each name may be enclosed in '{}'. Dollar may be escaped by \\ - except on Windows.

Non-existing environment variables are replaced by the empty string.

On Windows only: '~' is replaced by $USERPROFILE.

Usage:

```
zos expandenv filename
```

Parameters:

`filename`

The path to a filename to expand environment variables.

Example:

```
# => /my/path/file.txt if the environment variable PROJECT_DIR is "/my/path"
zos expandenv "\${PROJECT_DIR}/file.txt"
```

zos filenametype

Return a number to indicate the type of the given path.

Possible values are: * -1: empty file name. * 0: file is absolute. * 1: file is relative * 2: file is volumerelative (Windows only) * 3: file is no volume absolute (Windows only)

Usage:

```
zos filenametype filename
```

Parameters:

`filename`

The filename.

Example:

```
# => 0 = absolute
zos filenametype /my/path/file.txt

# => 1 = relative
zos filenametype ../file.txt
```

zos homedir

Get the home directory.

Usage:

```
zos homedir ?user?
```

Parameters:

`user (optional)`

Get the homedir for this user.

Example:

```
set home [zos homedir]
```

zos numberOfProcessors

Return the number of processors.

Usage:

`zos numberOfProcessors`

Parameters:

No parameters.

Example:

```
set cpuCount [zos numberOfProcessors]
```

zos sanitizeFilename

Create a sanitized version of the given `$filename`, e.g. replace special characters (`<`, `>`, `:`, `"`, `'`, `/`, `\\`, `|`, `?`, `*`) by `_`, avoid forbidden/special file names (`.`, `..`, `CON`, `PRN`, `AUX`, `NUL`, `COM1-9`, `LPT1-9`, and variations thereof), etc. This function is useful when automatically creating file names from DB names, e.g.

Usage:

`zos sanitizeFilename filename`

Parameters:

`filename`
: The filename to sanitize.

Example:

```
set name [$db oid oname $oid]
# $name may contain special characters; we need to sanitize it
set fileName [zos sanitizeFilename $name.txt]
set f [open $fileName w]
```

zos strcmpAlphanum

An alphanumeric string compare independent from any locale setting.

Usage:

`zos strcmpAlphanum str1 str2`

Parameters:

`str1`

The string to compare.

`str2`

The string to compare.

Example:

```
# => <0
zos strcmpAlphanum "A23" "A123"

# => >0
zos strcmpAlphanum "A42suffix" "A1other"

# => =0
zos strcmpAlphanum "Test123" "Test123"
```

zos tempdir

Return the path to temporary space.

Usage:

`zos tempdir`

Parameters:

No parameters.

Example:

```
set temp [zos tempdir]
```

zos tempFilename

Return the name of an unique temporary file.

Usage:

```
zos tempFilename prefix ?tempDir?
```

Parameters:

`prefix`

A prefix used to generate the temp filename.

`tempDir (optional)`

Path to a tmp directory.

Example:

```
# => something like /tmp/my_prefix_123456
zos tempFilename my_prefix
```

zos uniqueFileName

Make the given filename unique. Path can be relative or absolute. Removes extension (string after last .) and appends '_' and a number.

Usage:

```
zos uniqueFileName filename
```

Parameters:

`filename`

The filename to unify.

Example:

```
set filename [file join [zos tempdir] file.txt]
# => something like /tmp/file_123.txt if /tmp/file.txt already exists.
set unique [zos uniqueFileName $filename]
```

Message Callbacks

Introduction

To manage message callbacks the following API commands are supported.

Overview

Some database operations (like the spice parser, cone extraction, etc) may send warning or information messages by executing a callback procedure.

The callback procedure must be registered by `zmessage setCallback ⋯`.

The typical usage is:

```
proc myMsgCallback {time type message file line tclScript} {
    puts "$time $type $message $file $line $tclScript"
}
zmessage init
zmessage setCallback myMsgCallback
update idletasks
zspice parse ...
zmessage finit
```

This callback procedure is called from `zspice` command during execution. The 'type' argument can be used to distinguish between different message types. Possible types are `i`, `w` and `e` for information, warning and error messages. Additionally `t` for title information.

The init command tests if the callback was successfully installed by issue a special test message with the type `x`. If this test fails then a background error is thrown and `update idletasks` is needed to see the actual error message.

```
zmessage setCallbackFilter ?types?
```

can be used to only send certain message types to the registered callback.

To re-enable all message types, use `zmessage clearCallbackFilter 1`.

The call `zmessage finit` disables messaging output.

API Commands

To manage messages the following API commands are supported.

zmessage clear

Clear all messages.

Usage:

`zmessage clear`

Parameters:

No parameters.

Example:

```
# clear all messages
zmessage clear
```

zmessage clearCallbackFilter

Clear the callback filter.

Usage:

```
zmessage clearCallbackFilter value
```

Parameters:

`value`
: Value for all message types.

Example:

```
# set all type filters to 0 (=> no messages are forwarded to the callback)
zmessage clearCallbackFilter 0

# set all type filters to 1 (=> all messages are forwarded to the callback)
zmessage clearCallbackFilter 1
```

zmessage closeLogfile

Close the current log file.

Usage:

```
zmessage closeLogfile
```

Parameters:

No parameters.

Example:

```
# close the previously opened logfile
zmessage closeLogfile
```

zmessage count

Return the number of existing messages.

Usage:

```
zmessage count
```

Parameters:

No parameters.

Example:

```
set count [zmessage count]
puts "There are $count messages"
```

zmessage countFiltered

Return the number of existing filtered messages.

Usage:

```
zmessage countFiltered
```

Parameters:

No parameters.

Example:

```
set count [zmessage countFiltered]
puts "There are $count filtered messages"
```

zmessage enableDebug

Enable the given debug flag.

Usage:

```
zmessage enableDebug flag
```

Parameters:

`flag`
: Debug flag.

Example:

```
zmessage enableDebug "zdb/spos"
```

zmessage errorsSinceLastCall

Return the number of 'fatal' and 'error' messages since last call.

Usage:

```
zmessage errorsSinceLastCall
```

Parameters:

No parameters.

Example:

```
set errors [zmessage errorsSinceLastCall]
puts "errors since last call: $errors"
```

zmessage expandType

Return a human readable version of a one character type.

Usage:

```
zmessage expandType type
```

Parameters:

`type`
: Message type.

Example:

```
set expanded [zmessage expandType w]
puts "w -> $expanded"
```

zmessage finit

Tear down the messaging subsystem.

Usage:

```
zmessage finit
```

Parameters:

No parameters.

Example:

```
zmessage finit
```

zmessage foreach

Iterate over a range of messages.

Usage:

```
zmessage foreach first last indexVar timeVar typeVar messageVar filenameVar linenoVar
tclScriptVar body
```

Parameters:

`body`

The Tcl script body.

`filenameVar`

The file name variable.

`first`

The first index.

`indexVar`

The index variable.

`last`

The last index (may be 'end').

`linenoVar`

The file line variable.

`messageVar`

the message text variable.

`tclScriptVar`

The tclScript variable.

`timeVar`

The timestamp variable.

`typeVar`

the type variable.

Example:

```
zmessage foreach 0 "end"  index time type message fileName fileLine tclScript  {
    puts "[zmessage expandType $type]: $message"
}
```

zmessage foreachFiltered

Iterate over a range of filtered messages.

Usage:

```
zmessage foreachFiltered first last indexVar timeVar typeVar messageVar filenameVar linenoVar tclScriptVar body
```

Parameters:

`body`

The Tcl script body.

`filenameVar`

The file name variable.

`first`

The first index.

`indexVar`

The index variable.

`last`

The last index (may be 'end').

`linenoVar`

The file line variable.

`messageVar`

the message text variable.

`tclScriptVar`

The tclScript variable.

`timeVar`

The timestamp variable.

`typeVar`

the type variable.

Example:

```
zmessage foreachFiltered 0 "end"  index time type message fileName fileLine tclScript
{
    puts "[zmessage expandType $type]: $message"
}
```

zmessage getCallbackFilter

Return the current callback filter.

Usage:

```
zmessage getCallbackFilter
```

Parameters:

No parameters.

Example:

```
foreach type [zmessage getCallbackFilter] {
    puts "[zmessage longType $type] messages are forwarded"
}
```

zmessage getKnownDebugFlags

Return the list of known debug flags (list of name + description) pairs.

Usage:

```
zmessage getKnownDebugFlags
```

Parameters:

No parameters.

Example:

```
foreach {name description} [zmessage getKnownDebugFlags] {
    # do something with $name, $description
}
```

zmessage init

Initialize the messaging subsystem.

Usage:

```
zmessage init
```

Parameters:

No parameters.

Example:

```
zmessage init
```

zmessage isDebugEnabled

Check if the given debug flag is enabled.

Usage:

```
zmessage isDebugEnabled flag
```

Parameters:

`flag`
: Debug flag.

Example:

```
if {[zmessage isDebugEnabled "zdb/spos"]} {
    # do something
}
```

zmessage isInitialized

Check if the messaging subsystem is initialized.

Usage:

```
zmessage isInitialized
```

Parameters:

No parameters.

Example:

```
if {[zmessage isInitialized]} {
    # do something
}
```

zmessage longType

Return a long human readable version of a one character type.

Usage:

```
zmessage longType type
```

Parameters:

`type`
: Message type.

Example:

```
set long [zmessage longType w]
puts "w -> $long"
```

zmessage openLogfile

Open a log file.

Usage:

```
zmessage openLogfile ?-append? ?-buffered? ?-printold? filename
```

Parameters:

`-append (optional)`
: Open the log file in append mode.

`-buffered (optional)`

Enable buffering (don't flush the log file on each new message).

`-printold (optional)`

Print all existing old message to the log file after opening it.

`filename`

The file name of the log file to be opened.

Example:

```
# open a log file, also print all old messages to the file
zmessage openLogfile -printold my_log.txt
```

zmessage print

Print a message.

Usage:

`zmessage print type message ?filename? ?lineno?`

Parameters:

`filename (optional)`

The file the message corresponds to.

`lineno (optional)`

The file line the message corresponds to.

`message`

The actual message text.

`type`

Message type.

Example:

```
# print an info message
zmessage print INF "Info message"

# print a warning message with file/line info
zmessage print WAR "Syntax error" "my_file.v" 123
```

zmessage printWithTclScript

Print a message.

Usage:

```
zmessage printWithTclScript type message tclScript ?filename? ?lineno?
```

Parameters:

`filename (optional)`
: The file the message corresponds to.

`lineno (optional)`
: The file line the message corresponds to.

`message`
: The actual message text.

`tclScript`
: A Tcl script.

`type`
: Message type.

Example:

```
# print an info message
zmessage printWithTclScript INF "Info message" "myScript"

# print a warning message with file/line info
zmessage printWithTclScript WAR "Syntax error" "myScript" "my_file.v" 123
```

zmessage removeCallback

Remove a message callback.

Usage:

`zmessage removeCallback`

Parameters:

No parameters.

Example:

```
# remove the previously registered message callback
zmessage removeCallback
```

zmessage setCallback

Install a message callback.

Usage:

`zmessage setCallback ?-replay? callback`

Parameters:

`-replay` `(optional)`

: Replay old messages.

`callback`

: The callback proc to install. The callback proc will be called with `time`, `type`, `message`, `file`, and `line` parameters.

Example:

```
proc my_callback {time type message file line tclScript} {
    set formatTime [clock format $time -format "%Y-%m-%d %T"]
    set formatType [zmessage expandType $type]
    puts "$formatTime $formatType $message ($file:$line) $tclScript"
}

zmessage setCallback -replay my_callback
```

zmessage setCallbackFilter

Set a callback filter.

Usage:

```
zmessage setCallbackFilter types
```

Parameters:

`types`
: List of message types.

Example:

```
# only forward error and warning messages to the registered callback
zmessage setCallbackFilter {ERR WAR}
```

zmessage suppress

Suppress messages matching the given type and pattern.

Usage:

```
zmessage suppress type pattern
```

Parameters:

`pattern`
: Case insensitive message text pattern.

`type`
: Message type.

Example:

```
# suppress warning messages that match the pattern "*bad syntax:*"
zmessage suppress WAR "*bad syntax:*"
```

zmessage suppressCount

Return the number of suppressed messages for the given type and pattern.

Usage:

```
zmessage suppressCount type ?pattern?
```

Parameters:

`pattern (optional)`
: Case insensitive message text pattern. If no pattern is specified, return the number of all suppressed messages for the given type.

`type`
: Message type.

Example:

```
set count1 [zmessage suppressCount WAR]
puts "$count1 warning messages have been suppressed"

set count2 [zmessage suppressCount WAR "*bad syntax:*"]
puts "$count2 '*bad syntax:*' warning messages have been suppressed"
```

zmessage typeCount

Return the number of existing messages of the given type.

Usage:

```
zmessage typeCount type
```

Parameters:

`type`
: Message type.

Example:

```
set count [zmessage typeCount WAR]
puts "There are $count warning messages"
```

zmessage usedMemory

Return the number of bytes used.

Usage:

```
zmessage usedMemory
```

Parameters:

No parameters.

Example:

```
set mem [zmessage usedMemory]
puts "zmessage uses $mem bytes of memory"
```

Progress Bar

Simple Example

Some long-running database operations can periodically execute a callback function to inform the caller about the progress of the operation.

The typical usage is:

```
proc myProgress {w percent info} {
    puts "$w $percent $info"
    return 0
}
zprogress init myProgress .progress
zprogress begin
... zdb operation ...
zprogress end
zprogress finit
```

The initial `zprogress init` registers "myProgress" as callback function with ".progress" as first argument. The next `zprogress begin` enables progress calls for the following zdb operation. Here, the "...zdb operation..." will call myProgress with $w = ".progress" and two additional arguments

$percent and $info. The 'percent' argument is set to a floating number between 0.0 and 1.0 (representing the progress done so far) and the 'info' argument is set to a text message that gives some hint about what the database is currently doing. The terminating `zprogress end` executes a final call to myProgress with **percent** = **"END"** and then resets the internal state.

Nested Example

If a combination of multiple database operations and a long running custom procedure should appear as one progress bar, then the typical usage is (assuming 4 operations, the first takes 60%, the second and the third take 20%):

```
zprogress begin
zprogress push "operation 1" 0.60
... zdb operation #1 ...
zprogress pop
if {![zprogress isinterrupted]} {
    zprogress push "operation 2"  0.80
    ... zdb operation #2 ...
    zprogress pop
    if {![zprogress isinterrupted]} {
        zprogress push "operation 3"  1.00
        set count 1000
        for {set i 0} {$i < $count} {incr i} {
            zprogress update "" $i $count
            # Do something
        }
        zprogress pop
    }
}
zprogress end
zprogress finit
```

The user can interrupt the current database operation. An interrupt is released if the callback procedure returns "**1**" (instead of "0"). If so, then the database operation will return with an error. The `isinterrupted` subcommand checks for this special condition. Please check out a full example here: demo/api/oem/interrupt.tcl.

API Commands

To manage progress callbacks the following API commands are supported.

zprogress begin

Enable progress calls for the following operation.

Usage:

`zprogress begin`

Parameters:

No parameters.

Example:

```
zprogress begin
# ...
zprogress end
```

zprogress check

Return true if the interrupted flag is set. Alias for `zprogress isinterrupted`.

Usage:

```
zprogress check ?setLastErrorMsg?
```

Parameters:

`setLastErrorMsg (optional)`
: Write the text "interrupted" in the last error message.

Example:

```
if {[zprogress check]} {
    # the current progress has been interrupted by the user
}
```

zprogress end

Call progress callback with end mark and disable the progress bar.

Usage:

```
zprogress end
```

Parameters:

No parameters.

Example:

```
zprogress begin
# ...
zprogress end
```

zprogress finit

Delete registered progress callback.

Usage:

```
zprogress finit
```

Parameters:

No parameters.

Example:

```
proc my_callback {w ratio message} {
    puts "$w $ratio $message"
    return 0
}
zprogress init my_callback .progress 10 2
# ...
zprogress finit
```

zprogress init

Register a progress update callback procedure.

Usage:

```
zprogress init progressCallback pathName ?cbPerSec? ?firstDelay?
```

Parameters:

`cbPerSec` `(optional)`

Number of callback calls per second.

`firstDelay (optional)`

The number of first calls which where suppressed.

`pathName`

The widget path of the progress bar. This is appended as the first parameter to the progressCallback. May be the empty string if not needed.

`progressCallback`

The name of the progress callback procedure. When calling the callback, pathName, the current progress ratio (or `END`), and the current progress message are appended as parameters. If the callback returns `1`/`true` the progress is interrupted.

Example:

```
proc my_callback {w ratio message} {
    puts "$w $ratio $message"
    return 0
}
zprogress init my_callback .progress
# ...
zprogress finit
```

zprogress isinterrupted

Return true if the interrupted flag is set.

Usage:

`zprogress isinterrupted ?setLastErrorMsg?`

Parameters:

`setLastErrorMsg (optional)`

Write the text "interrupted" in the last error message.

Example:

```
if {[zprogress isinterrupted]} {
    # the current progress has been interrupted by the user
}
```

zprogress istimeoutreached

Return true if the timeoutreached flag is set.

Usage:

```
zprogress istimeoutreached ?setLastErrorMsg?
```

Parameters:

`setLastErrorMsg (optional)`

Write the text "timeout reached" in the last error message.

Example:

```
if {[zprogress istimeoutreached]} {
    # the current progress has timeoutreached
}
```

zprogress pop

Pop range from the stack.

Usage:

```
zprogress pop ?-nocallback?
```

Parameters:

`-nocallback (optional)`

Do not run the registered progress callback.

Example:

```
zprogress push "progress at 10%" 0.1
# do some work
zprogress pop

zprogress push "progress at 50%" 0.5
# do some work
zprogress pop -nocallback
```

zprogress push

Push a new progress bar range to the stack.

Usage:

```
zprogress push msg ratio
```

Parameters:

`msg`

: The push message to display.

`ratio`

: The amount to push on the stack (0.0 ... 1.0).

Example:

```
zprogress push "progress at 10%" 0.1
# do some work
zprogress pop

zprogress push "progress at 50%" 0.5
# do some work
zprogress pop
```

zprogress settimeout

Set the timeout period in msec. 0 = disable.

Usage:

```
zprogress settimeout timeout
```

Parameters:

`timeout`
: timeout in msec. 0 = disable.

Example:

```
zprogress settimeout 10
```

zprogress timestep

Toggle runtime statistics.

Usage:

```
zprogress timestep enable
```

Parameters:

`enable`
: Enable or disable the runtime statistic.

Example:

```
# enable runtime statistics (printed via `zmessage print DBG ...`)
zprogress timestep 1

# disable statistics
zprogress timestep 0
```

zprogress update

Update the progress bar (same as push/pop).

Usage:

```
zprogress update ?-nocallback? msg curItem totalItems
```

Parameters:

`-nocallback (optional)`

Do not run the registered progress callback.

`curItem`

The current processed item.

`msg`

The update message to display.

`totalItems`

The total number of items.

Example:

```
zprogress update "at item 17/100" 17 100
zprogress update -nocallback "at item 18/100" 18 100
```

License Management

Introduction

To manage licenses the following API commands are supported.

Overview

```
zlicense checkout ?-wait_for_license $sec? $feature
zlicense checkin   $feature
zlicense version
zlicense features
zlicense available $feature
zlicense notice    $feature
zlicense init
zlicense location  $feature
zlicense permit    sub-feature
zlicense idle      $state
zlicense renew
zlicense finit
zlicense master    sub-feature
```

Utility Functions

Overview

```
zutil signal default signame
zutil signal ignore  signame
zutil signal trap    signame signalhandler
zutil signal get     signame
zutil signal send    signame pid
```

The 'signal default' command installs the default signal handler of the system for the given signal name.

The 'signal ignore' command set the signal state to ignore for the specified signal.

The 'signal trap' command installs a custom signal handler for the given signal name that will be evaluated. The substitution '%S' can be used to pass the name of the signal to the callback.

The 'signal get' command returns the signal states for the specified signal.

The 'signal send' command sends the signal with the given signal name to the process with the given pid.

List of Debug Flags

This document lists all valid debug flags, which can be used to increase the verbosity of specific operations/features in order to debug issues with RTLvision PRO. Debug flags can be set with the `-debugFlag FLAGNAME` command line option, where `FLAGNAME` is one of the debug flags from the list below. Multiple flags can be set by either using multiple `-debugFlag` options or glob style pattern to match multiple flag names, e.g. `zdb/*`. If debug flags are used, it's often advisable to also set the message level to "debug" (e.g. `-info debug`) and specify a log file location (`-logfile log.txt`).

- `all`: Enable all debug flags (except `*/syntax_parser`).
- `assert`: Assertions are enabled and reported as fatal error messages. (enabled with "all")
- `dspf/lex`: Enable DSPF tokenizer (lexer) debug output.
- `dspf/parse`
- `dump/filled` (enabled with "all")
- `edif/compatible`: Enable compatibility mode with older versions of the EDIF parser. (enabled with "all")
- `edif/lex`: Enable full EDIF tokenizer (lexer) debug output.
- `edif/lex:recover`: Enable only EDIF tokenizer (lexer) error recovery debug output.
- `edif/lex:tok`: Enable only the EDIF tokenizer (lexer) debug output.
- `edif/merge`: Enable EDIF net merge debug output. (enabled with "all")
- `edif/syntax_parser`: Enable EDIF syntax parser (bison) runtime debug output to stderr.
- `liberty/process`: Enable group processing debug output. (enabled with "all")
- `liberty/trace`: Turn on trace mode to dump two trace files, a .c file, and a .h file. These files will be in ANSI C format, and record each function call, along with its arguments, in a format that

will allow the later compilation and execution of the generated code without sharing any confidential information (except the input file name).

- `liberty/syntax_parser`: Enable Liberty syntax parser (bison) runtime debug output to stderr.
- `rtl/copy2zdb`: Enable debug output for copying the Verific netlist database into ZDB. (enabled with "all")
- `rtl/preprocess`: Enable debug output for the RTL -preProcessOutputFile option. (enabled with "all")
- `sdf/syntax_parser`: Enable SDF syntax parser (bison) runtime debug output to stderr.
- `spef/lex`: Enable SPEF tokenizer (lexer) debug output.
- `spef/namemap` (enabled with "all")
- `spef/parse`
- `spef/strpool` (enabled with "all")
- `spef/syntax_parser`: Enable SPEF syntax parser (bison) runtime debug output to stderr.
- `spf/createtop` (enabled with "all")
- `spf/hier` (enabled with "all")
- `spf/info` (enabled with "all")
- `spf/proc` (enabled with "all")
- `spf/spos` (enabled with "all")
- `spf/stat` (enabled with "all")
- `spice/parse`
- `spice/spc2zdb`
- `spice/lex`
- `spice/save` (enabled with "all")
- `spice/spectrelpe`: Enable debug output for Spectre LPE mode.
- `spice/spectrepex`: Enable debug output for Spectre PEX mode.
- `vcd/syntax_parser`: Enable VCD header syntax parser (bison) runtime debug output to stderr.
- `verilog/msg:duplicates` (enabled with "all")
- `verilog/scanner`: Enable Verilog netlist tokenizer (lexer) debug output.
- `verilog/syntax_parser`: Enable Verilog netlist syntax parser (bison) runtime debug output to stderr.
- `zdb/calcres` (enabled with "all")
- `wdb/create`: Debug wdb creation. (enabled with "all")
- `wdb/util`: Debug wdb util. (enabled with "all")
- `zdb/cdc:clkPinList` (enabled with "all")
- `zdb/cdc:clkReduced` (enabled with "all")
- `zdb/cdc:clkTree` (enabled with "all")

- zdb/cdc:clkTreeMore (enabled with "all")
- zdb/cdc:domainCross (enabled with "all")
- zdb/cdc:domainCrossCone (enabled with "all")
- zdb/cdc:domainCrossDRS (enabled with "all")
- zdb/cdc:domainCrossMore (enabled with "all")
- zdb/cdc:treeCross (enabled with "all")
- zdb/cdc:treeCrossCone (enabled with "all")
- zdb/cone:shortest (enabled with "all")
- zdb/cone:shortestFrom (enabled with "all")
- zdb/cone:shortestNet (enabled with "all")
- zdb/cone:shortestProg (enabled with "all")
- zdb/cone:shortestTo (enabled with "all")
- zdb/cone:toPG (enabled with "all")
- zdb/cone:extract (enabled with "all")
- zdb/extract:follow (enabled with "all")
- zdb/extract:filterLogicalInvalid (enabled with "all")
- zdb/extract:free (enabled with "all")
- zdb/extract:new (enabled with "all")
- zdb/extract:pop (enabled with "all")
- zdb/extract:process (enabled with "all")
- zdb/extract:result (enabled with "all")
- zdb/extract:push (enabled with "all")
- zdb/flat:symlibscan (enabled with "all")
- zdb/mem (enabled with "all")
- zdb/mem:freelist (enabled with "all")
- zdb/mem:size (enabled with "all")
- zdb/oid2nlv (enabled with "all")
- zdb/oper:changeblackboxinterface (enabled with "all")
- zdb/oper:createpreplace (enabled with "all")
- zdb/oper:deleteunused (enabled with "all")
- zdb/oper:expand (enabled with "all")
- zdb/oper:guessdir (enabled with "all")
- zdb/oper:guessnetbus (enabled with "all")
- zdb/oper:guessportbus (enabled with "all")
- zdb/oper:guesstop (enabled with "all")

- zdb/oper:layoutcomment (enabled with "all")
- zdb/oper:propagateignoreinwrite (enabled with "all")
- zdb/oper:removeemptymod (enabled with "all")
- zdb/oper:subckt2dev (enabled with "all")
- zdb/oper:verilogbusses (enabled with "all")
- zdb/oper:zombieunused (enabled with "all")
- zdb/operdevice:cleanup (enabled with "all")
- zdb/operdevice:groupmultifinger (enabled with "all")
- zdb/operdevice:guesspower (enabled with "all")
- zdb/operdevice:mergecurmirror (enabled with "all")
- zdb/operdevice:mergeparallelinst (enabled with "all")
- zdb/operdevice:mergeserial (enabled with "all")
- zdb/operdevice:mergeserialcap (enabled with "all")
- zdb/operdevice:mergeserialparallel (enabled with "all")
- zdb/operdevice:mergeserialres (enabled with "all")
- zdb/operdevice:poweranddir (enabled with "all")
- zdb/operdevice:removemos (enabled with "all")
- zdb/operdevice:removerescap (enabled with "all")
- zdb/operdevice:removeuseless (enabled with "all")
- zdb/opergate (enabled with "all")
- zdb/opergateppinv (enabled with "all")
- zdb/operhier:createhier (enabled with "all")
- zdb/operparasitic:analyzecoupl (enabled with "all")
- zdb/operparasitic:inline (enabled with "all")
- zdb/operparasitic:rename (enabled with "all")
- zdb/operparasitic:rollback (enabled with "all")
- zdb/operpp:bubbles (enabled with "all")
- zdb/operpp:chain (enabled with "all")
- zdb/operpp:const (enabled with "all")
- zdb/operpp:guessinstarray (enabled with "all")
- zdb/operpp:guesswide (enabled with "all")
- zdb/operpp:guesswidebus (enabled with "all")
- zdb/operpp:negedge (enabled with "all")
- zdb/operpp:reducepins (enabled with "all")
- zdb/operpp:removebuf (enabled with "all")

- zdb/operpp:removedangle (enabled with "all")
- zdb/operpp:removeinv (enabled with "all")
- zdb/operpp:removeunused (enabled with "all")
- zdb/operpp:rtlschem (enabled with "all")
- zdb/reduceres (enabled with "all")
- zdb/reduceresdumpspice (enabled with "all")
- zdb/spos (enabled with "all")
- zdb/spos:bucket (enabled with "all")
- zdb/spos:getfilepos (enabled with "all")
- zdb/spos:insline (enabled with "all")
- zdb/spos:newfile (enabled with "all")
- zdb/spos:pos (enabled with "all")
- zdb/spos:searchfile (enabled with "all")
- zdb/spos:searchline (enabled with "all")
- zdb/spos:invalidate (enabled with "all")
- zdb/spos:validatedump (enabled with "all")
- zdb/spos:validateredundant (enabled with "all")
- zdb/symlibcopy (enabled with "all")
- zdb/symlibscan (enabled with "all")
- zdb/tapi (enabled with "all")
- zdb/writeSpice (enabled with "all")
- zdb/writeVerilog (enabled with "all")
- zdb/writeVerilog:Assign (enabled with "all")
- zdb/writeVerilog:CheckNames (enabled with "all")
- zdb/writeVerilog:CheckRanges (enabled with "all")
- zdb/writeVerilog:FindConn (enabled with "all")
- zdb/writeVerilog:Write (enabled with "all")
- zdb/zDuplicate (enabled with "all")
- zdb/zclone (enabled with "all")
- zdb/zdb:searchEscapedName (enabled with "all")
- zdb/zdi (enabled with "all")
- zdb/zinfobox (enabled with "all")
- zdb/zsynbool (enabled with "all")
- zdb/writeAscii:dbgRecord (enabled with "all")
- zdb/readAscii:ignoreErr

- `zdb/readAscii:dbgRecord` (enabled with "all")
- `gui/traceProcedureCalls`: Trace all procedure calls in the GUI code. (enabled with "all")
- `gui/instrument`: Instrument all GUI procedures and log all calls into the logfile. (enabled with "all")
- `gui/instrument2`: Instrument all GUI procedures and log all calls with the parameter values into the logfile. (enabled with "all")
- `gui/waveMove`: Add debug output while moving signals in the Wave window. (enabled with "all")
- `zdb/renamePortFromNet`

The Wave Database (WDB) API

The Waveform Database Tcl API

wdb check

Return true if file is a readable binfile.

Usage:

```
wdb check fname
```

Parameters:

`fname`

The name of the binary file to check

Example:

```
set ok [wdb check "test.wdb"]
```

wdb create

The `wdb create` command creates a WDB database object which can be accessed by the returned unique Tcl command. If the optional file name is given a binary wdb file is created. Otherwise the database exists only in memory.

Usage:

```
wdb create ?-analyze? ?-dump dumpmode? ?-noCollapse? ?-nogcd? ?-noop? ?-noSkipDupl? ?-nozip? ?-ptime ptime? ?-skipLeadX? ?-vcd? ?fname?
```

Parameters:

`-analyze (optional)`

Create analyze output.

`-dump dumpmode (optional default is NULL)`

Create dump output.

`-noCollapse (optional)`

Do not collapse value changes at same timestep.

`-nogcd (optional)`

Do not compress storage of time steps.

`-noop (optional)`

Create noop output.

`-noSkipDupl (optional)`

Do not skip duplicate values.

`-nozip (optional)`

Do not compress whole database.

`-ptime ptime (optional default is -1)`

Create ptime output.

`-skipLeadX (optional)`

Ignore leading 'X' values.

`-vcd (optional)`

Create text vcd file output.

`fname (optional)`

Optional name of the binary file to create. One of the above options cannot be used as the filename.

Example:

```
set wdb1 [wdb create]
set wdb2 [wdb create test.wdb]
set wdb3 [wdb create -skipLeadX test.wdb]
```

wdb info

Get verbose description about binfile.

Usage:

`wdb info fname`

Parameters:

`fname`

The name of the binary file

Example:

```
set info [wdb info "test.wdb"]
```

wdb open

Restore a binary WDB file into memory and return the name of the new Tcl command for accessing the WDB database.

Usage:

```
wdb open fname
```

Parameters:

`fname`

Name of the WDB file to open.

Example:

```
set wdb [wdb open "test.wdb"]
```

wdb read

Parse a VCD file into memory and return the name of the new Tcl command for accessing the created in-memory WDB database.

Usage:

```
wdb read ?-from from? ?-noCollapse? ?-nogcd? ?-noSkipDupl? ?-skipLeadX? ?-to to? fname
```

Parameters:

`-from from (optional default is 0)`

Skip data until from

`-noCollapse (optional)`

Do not collapse value changes at same timestep.

`-nogcd (optional)`

Do not compress storage of time steps.

`-noSkipDupl (optional)`

Do not skip duplicate values.

`-skipLeadX (optional)`

Ignore leading 'X' values.

`-to to (optional default is 0)`

Stop parsing at to

`fname`

Name of the VCD file to read.

Example:

```
set wdb [wdb read -from 0 -to 1000 "test.vcd"]
```

wdb version

Prints the parser version number

Usage:

`wdb version`

Parameters:

No parameters.

Example:

```
wdb version
```

$wdb addalias

The `$wdb addalias` command adds a new alias variable to the current scope. If the `width` parameter is greater than 1, a vector is defined. Multiple aliases/variables (in different scopes) can share the same signal id `signo`.

Usage:

`$wdb addalias aliastype name width signo ?iname?`

Parameters:

`aliastype`
: The parameter vartype can be one of the following values: pin, port, net

`iname (optional)`
: Instance name (only for pin aliases).

`name`
: Name of the variable.

`signo`
: Signal number.

`width`
: Width of the variable.

Example:

```
set sigNo2 [$wdb nextsigno]
$wdb addalias "port" {test[0:1]} 2 $sigNo2
```

$wdb adddefinitions

The `$wdb adddefinitions` command reenables adding scopes, variable.

Usage:

`$wdb adddefinitions`

Parameters:

No parameters.

Example:

```
$wdb adddefinitions
```

$wdb addevent

Add an event. Because storing an event in wdb is currently not supported using this command creates an error.

Usage:

```
$wdb addevent signo
```

Parameters:

`signo`
: Signal number.

Example:

```
$wdb addevent [$wdb nextsigno]
```

$wdb addreal

Add a real value. Because storing a real value in wdb is currently not supported using this command creates an error.

Usage:

```
$wdb addreal signo value
```

Parameters:

`signo`
: Signal number.

`value`
: Value.

Example:

```
$wdb addreal [$wdb nextsigno] 1.23
```

$wdb addscalar

The `$wdb addscalar` command defines a new value at the current time for the given signal `signo`.

Usage:

```
$wdb addscalar signo value
```

Parameters:

`signo`
: Signal number.

`value`
: Possible values are: x/X, z/Z, 0, 1

Example:

```
$wdb addscalar $sigNo1 0
```

$wdb addtime

The `$wdb addtime` command adds a new time mark for upcoming value changes.

Usage:

```
$wdb addtime time
```

Parameters:

`time`
: Add a time.

Example:

```
$wdb addtime 100
```

$wdb addvar

The `$wdb addvar` command adds a new variable to the current scope. If the `width` parameter is greater than 1, a vector is defined. Multiple variables (in different scopes) can share the same signal id `signo`.

Usage:

```
$wdb addvar vartype valtype name width signo
```

Parameters:

`name`

Name of the variable.

`signo`

Signal number.

`valtype`

The parameter valtype can be one of the following values: scalar, vector, real

`vartype`

The parameter vartype can be one of the following values: unknown, event, integer, parameter, real, reg, supply0, supply1, time, tri, triand, trior, trireg, tri0, tri1, wand, wire, wor

`width`

Width of the variable.

Example:

```
set sigNo1 [$wdb nextsigno]
$wdb addvar "wire" "scalar" "clk"       1 $sigNo1
set sigNo2 [$wdb nextsigno]
$wdb addvar "reg"  "vector" {test[0:1]} 2 $sigNo2
```

$wdb addvector

The `$wdb addvector` command defines a new value at the current time for the given signal `signo`. The given `width` should match with the width defined in `$wdb addvar`. The given `val` string has one character for each bit. LSB comes first.

Usage:

`$wdb addvector signo value width`

Parameters:

`signo`
: Signal number.

`value`
: Possible values are the same as described in `$wdb addscalar`.

`width`
: Width of the vector.

Example:

```
    $wdb addvector $sigNo2 "1X" 2
```

$wdb close

The `$wdb close` command writes all data, frees used memory and deletes the `$wdb` Tcl command.

Usage:

`$wdb close`

Parameters:

No parameters.

Example:

```
$wdb close
```

$wdb dirvar

The `$wdb dirvar` command look for a variable in the current scope. If found it sets the direction.

Usage:

`$wdb dirvar name dir`

Parameters:

`dir`
: direction.

`name`
: Name of the variable.

Example:

```
$wdb dirvar "foo" inout
```

$wdb done

The `$wdb done` command marks the end of all value changes.

Usage:

`$wdb done`

Parameters:

No parameters.

Example:

```
$wdb done
```

$wdb enddefinitions

The `$wdb enddefinitions` command ends the variable definitions and starts adding value changes.

Usage:

`$wdb enddefinitions`

Parameters:

No parameters.

Example:

```
$wdb enddefinitions
```

$wdb findalias

The `$wdb findalias` command look for a alias variable in the the current scope. If found it returns the signal id `signo` else -1.

Usage:

```
$wdb findalias name type ?iname?
```

Parameters:

`iname (optional)`
: Instance name of pin alias.

`name`
: Name of the variable.

`type`
: Alias type.

Example:

```
set sigNo2 [$wdb findalias "foo" Net]
```

$wdb findvar

The `$wdb findvar` command look for a variable in the current scope. If found it returns the signal id `signo` else -1.

Usage:

```
$wdb findvar name
```

Parameters:

`name`
: Name of the variable.

Example:

```
set sigNo2 [$wdb findvar "foo"]
```

$wdb hidevar

The `$wdb hidevar` command look for a variable in the current scope. If found it sets the hide flag.

Usage:

`$wdb hidevar name h`

Parameters:

`h`
: Hide flag.

`name`
: Name of the variable.

Example:

```
$wdb hidevar "foo" 1
```

$wdb isReady

Check if wdb is ready to create readers.

Usage:

`$wdb isReady`

Parameters:

No parameters.

Example:

```
$wdb isReady
```

$wdb memdump

dump memory to file.

Usage:

```
$wdb memdump fname
```

Parameters:

`fname`
: The name of the dump file

Example:

```
$wdb memdump "out.dmp"
```

$wdb memory

Return memory statistics of the WDB database.

Usage:

```
$wdb memory
```

Parameters:

No parameters.

Example:

```
set mem [$wdb memory]
```

$wdb nextsigno

The `$wdb nextsigno` command returns the next usable signal id.

Usage:

`$wdb nextsigno`

Parameters:

No parameters.

Example:

```
set sigNo [$wdb nextsigno]
```

$wdb pathscope

The `$wdb pathscope` command sets the current scope as path for upcoming variable definitions.

Usage:

`$wdb pathscope ?-relative? type names`

Parameters:

`-relative (optional)`
: Start path as current scope.

`names`
: Path of the scope names.

`type`
: Type of the scope. The following scope types are possible: unknown, module, task, function, begin and fork

---

Example:

```
$wdb pathscope "module" {"ALU" "ADDER"}
```

$wdb reader

Create a new reader on the given database. Multiple readers can be created. Each reader should be freed with `$rdr free` command.

Usage:

```
$wdb reader
```

Parameters:

No parameters.

Example:

```
set rdr [$wdb reader]
```

$wdb save

Save memory to binfile.

Usage:

```
$wdb save fname
```

Parameters:

`fname`

The name of the binary file

Example:

```
$wdb save "out.wdb"
```

$wdb scope

The `$wdb scope` command sets the current scope for upcoming variable definitions.

Usage:

```
$wdb scope type name
```

Parameters:

`name`

Name of the scope.

`type`

Type of the scope. The following scope types are possible: unknown, module, task, function, begin and fork

---

Example:

```
$wdb scope "module" "CPU"
```

$wdb setdate

The `$wdb setdate` command sets the date string in the header data.

Usage:

```
$wdb setdate date
```

Parameters:

`date`

Set the given date string.

Example:

```
$wdb setdate "03.11.14"
```

$wdb settimescale

The `$wdb settimescale` command sets the timescale string in the header data.

Usage:

`$wdb settimescale scale`

Parameters:

`scale`
: Set the given timescale string.

Example:

```
$wdb settimescale "1ns"
```

$wdb setversion

The `$wdb setversion` command sets the version string in the header data.

Usage:

`$wdb setversion version`

Parameters:

`version`
: Set the given version string.

Example:

```
$wdb setversion "1.3"
```

$wdb upscope

The `$wdb upscope` command sets to current scope one level up.

Usage:

```
$wdb upscope
```

Parameters:

No parameters.

Example:

```
$wdb upscope
```

The Waveform Database Reader Tcl API

$wdb reader

new reader subcommand.

Usage:

`$wdb reader`

Parameters:

No parameters.

Example:

```
set rdr [$wdb reader]
```

$wread foreach

Iterate over object of given type. Evaluate body with loop variable set to current object. 'varitem' is the combination of variables and groups.

$wread foreach clone

Loop over clones.

Usage:

`$wread foreach clone ?scopeid? var body`

Parameters:

`body`

Loop Body.

`scopeid (optional)`

Scope of clone.

`var`
: Name of loop variable.

Example:

```
$rdr foreach clone             cvar {set cname [$rdr varname $cvar]}
$rdr foreach clone    $scopeid cvar {set cname [$rdr varname $cvar]}
```

$wread foreach group

Loop over groups.

Usage:

```
$wread foreach group ?scopeid? var body
```

Parameters:

`body`
: Loop Body.

`scopeid (optional)`
: Scope of group.

`var`
: Name of loop variable.

Example:

```
$rdr foreach group             gvar {set gname [$rdr varname $gvar]}
$rdr foreach group    $scopeid gvar {set gname [$rdr varname $gvar]}
```

$wread foreach member

Loop over member.

Usage:

```
$wread foreach member ?-lsbfirst? ?-msbfirst? varid mvar body
```

Parameters:

`-lsbfirst (optional)`

LSB first.

`-msbfirst (optional)`

MSB first.

`body`

Loop Body.

`mvar`

Name of loop variable.

`varid`

Variable ID.

Example:

```
$rdr foreach member   $varid   mvar {set mname [$rdr varname $mvar]}
$rdr foreach member -lsbfirst $varid mvar {}
$rdr foreach member -msbfirst $varid mvar {}
```

$wread foreach scope

Loop over scopes.

Usage:

`$wread foreach scope ?scopeid? svar body`

Parameters:

`body`

Loop Body.

`scopeid (optional)`

Scope ID.

`svar`

Name of loop variable.

Example:

```
$rdr foreach scope             svar {set sname [$rdr scopename $svar]}
$rdr foreach scope    $scopeid svar {set sname [$rdr scopename $svar]}
```

$wread foreach variable

Loop over scopes.

Usage:

```
$wread foreach variable ?parentScopeId? vvar body
```

Parameters:

`body`

Loop Body.

`parentScopeId` `(optional)`

Scopeid ID.

`vvar`

Name of loop variable.

Example:

```
$rdr foreach variable          vvar {set vname [$rdr varname $vvar]}
$rdr foreach variable $scopeid vvar {set vname [$rdr varname $vvar]}
```

$wread foreach varitem

Loop over variable items.

Usage:

```
$wread foreach varitem ?scopeid? var body
```

Parameters:

`body`

Loop Body.

`scopeid (optional)`

Scope of group.

`var`

Name of loop variable.

Example:

```
$rdr foreach varitem           gvar {set gname [$rdr varname $gvar]}
$rdr foreach varitem  $scopeid gvar {set gname [$rdr varname $gvar]}
```

$wread calcZoom

set new from/to range for zoom.

Usage:

`$wread calcZoom fromName toName factor ?center?`

Parameters:

`center (optional)`

optional center of zoom.

`factor`

zoom factor, positive is zoom in, negative is zoom out, 0 is fullfit. or use 'in', 'out' 'full' for 0.1, -0.1 and 0.

`fromName`

name of from variable.

`toName`

name of to variable.

Example:

```
set from 0
set to   1000
$rdr calcZoom from to 0.1 250
```

$wread clonecreate

create a new clone.

Usage:

```
$wread clonecreate varid
```

Parameters:

`varid`
: Variable referenced from clone.

Example:

```
set c [$rdr clonecreate $varid]
```

$wread clonedelete

delete a clone or all clones.

Usage:

```
$wread clonedelete ?varid?
```

Parameters:

`varid (optional)`
: Clone to delete.

Example:

```
$rdr clonedelete $c
```

$wread date

return date string from header.

Usage:

```
$wread date
```

Parameters:

No parameters.

Example:

```
set str [$rdr date]
```

$wread DUTscope

get current scope to design under test.

Usage:

```
$wread DUTscope
```

Parameters:

No parameters.

Example:

```
set dutScopeId [$rdr DUTscope]
```

$wread DUTtopName

get current top module name of design under test.

Usage:

```
$wread DUTtopName
```

Parameters:

No parameters.

Example:

```
set topName [$rdr DUTtopName]
```

$wread firsttime

return the smallest (first) time in wdb.

Usage:

```
$wread firsttime
```

Parameters:

No parameters.

Example:

```
set t [$rdr firsttime]
```

$wread free

Free all memory used by given reader and remove TCL command.

Usage:

```
$wread free
```

Parameters:

No parameters.

Example:

```
$rdr free
```

$wread group

return the value at the given time. It is an error if the given 'varId' is not a vector.

Usage:

```
$wread group time varId
```

Parameters:

`time`

Current time.

`varId`

Variable ID.

Example:

```
set v [$rdr group $time $varid]
```

$wread groupbuild

Create groups for variables with same basename.

Usage:

```
$wread groupbuild ?-hidemember hidemember? ?-rangedown rangedown? ?-scope scopeId? ?-type vartype?
```

Parameters:

`-hidemember` **`hidemember`** `(optional default is zFalse)`

Hide members contained in group.

`-rangedown` **`rangedown`** `(optional default is zFalse)`

Group members with range downwards.

`-scope scopeId (optional default is WreadNullScopeId)`

Scope of group.

`-type vartype (optional default is WdbVarInvalid)`

Type of group.

Example:

```
$rdr groupbuild -scope $scopeid -type reg -hidemember on -rangedown on
```

$wread groupcreate

create a new group.

Usage:

```
$wread groupcreate ?-autoScope? ?-expandVectors? ?-hidemember hidemember? ?-scope scope? ?-type type? name varidList
```

Parameters:

`-autoScope (optional)`

Derive the scope from the variable list. Uses the common scope or the null scope.

`-expandVectors (optional)`

If this flag is enabled, vectors are not expanded to their members.

`-hidemember hidemember (optional default is zFalse)`

Hide members contained in group.

`-scope scope (optional default is WreadNullScopeId)`

Scope of group.

`-type type (optional default is WdbVarGroup)`

Type of group.

`name`

Name of group.

`varidList`

Variables contained in group.

Example:

```
set g [$rdr groupcreate -scope $scopeid -hidemember on  -type reg $name [list $varid]]
```

$wread groupdelete

delete a group or all groups.

Usage:

```
$wread groupdelete ?varid?
```

Parameters:

`varid (optional)`

Group to delete.

Example:

```
$rdr groupdelete $g
```

$wread groupfind

Find a group by name.

Usage:

```
$wread groupfind name
```

Parameters:

`name`

Name of the group.

Example:

```
$rdr groupfind Grp1
```

$wread groupredefine

redefine a given group

Usage:

```
$wread groupredefine ?-expandVectors? grpid varidList ?hidemember?
```

Parameters:

`-expandVectors` `(optional)`

If this flag is enabled, vectors are not expanded to their members.

`grpid`

Group to redefine.

`hidemember` `(optional)`

Hide members contained in group.

`varidList`

Variables contained in group.

Example:

```
$rdr groupredefine $g [list $varid]
```

$wread grouprename

rename a given group.

Usage:

```
$wread grouprename varid name
```

Parameters:

`name`

New name of group.

`varid`

Group to delete.

Example:

```
$rdr grouprename $g "newname"
```

$wread isscopeid

return true if id is a valid scopeid.

Usage:

```
$wread isscopeid scopeid
```

Parameters:

`scopeid`

Scope ID.

Example:

```
if {[$rdr isscopeid $varid]} {
    # ...
}
```

$wread isvarid

return true if id is a valid varid.

Usage:

```
$wread isvarid varid
```

Parameters:

`varid`

Variable or member ID.

Example:

```
if {[$rdr isvarid $varid]} {
    # ...
}
```

$wread lasttime

return the biggest (last) time in wdb.

Usage:

```
$wread lasttime
```

Parameters:

No parameters.

Example:

```
set t [$rdr lasttime]
```

$wread member

return the value at the given time. It is an error if the given 'varId' is not a vector.

Usage:

```
$wread member time varId
```

Parameters:

`time`
: Current time.

`varId`
: Variable ID.

Example:

```
set v [$rdr member $time $varid]
```

$wread memvar

Return var id of the given group member variable.

Usage:

```
$wread memvar memid
```

Parameters:

`memid`
: Group member variable ID.

Example:

```
set memvarid [$rdr memvar $vmid]
```

$wread nexttime

search for the next value change in given list of variables 'varIdLst' before 'time'. If multiple variables have a value change before the given 'time' the nearest is returned. If none is found until the optional 'tlimit' is reached '{}' is returned.

Usage:

```
$wread nexttime time ?timeLimit? varIds
```

Parameters:

`time`
: Current time.

`timeLimit (optional)`
: Specify a time to stop the search.

`varIds`
: List of variable IDs.

Example:

```
set t [$rdr nexttime 1000 [list $varid]]
```

$wread nextvalue

search for the next value change in given list of variables 'varIdLst' before 'time', which matches the given value. If multiple variables have a value change before the given 'time' the nearest is returned. If none is found until the optional 'tlimit' is reached '{}' is returned.

Usage:

```
$wread nextvalue time ?timeLimit? varIds value
```

Parameters:

`time`
: Current time.

`timeLimit (optional)`
: Specify a time to stop the search.

`value`
: Possible values are: x/X, z/Z, 0, 1

`varIds`
: List of variable IDs.

Example:

```
set t [$rdr nextvalue 1000 [list $varid] "101X"]
```

$wread pix2time

return time for given pix index by from and to time and pixelcnt.

Usage:

```
$wread pix2time from to pixelCnt pix
```

Parameters:

`from`
: from time step.

`pix`
: pixel index which should be converted into time step.

`pixelCnt`
: width of drawing rectangle.

`to`
: to time step.

Example:

```
set pix [$rdr pix2time 0 1000 100 42]
```

$wread prevtime

search for the previous value change in given list of variables 'varIdLst' before 'time'. If multiple variables have a value change before the given 'time' the nearest is returned. If none is found until the optional 'tlimit' is reached '{}' is returned.

Usage:

`$wread prevtime time ?timeLimit? varIds`

Parameters:

`time`
: Current time.

`timeLimit (optional)`
: Specify a time to stop the search.

`varIds`
: List of variable IDs.

Example:

```
set t [$rdr prevtime 1000 [list $varid]]
```

$wread prevvalue

search for the previous value change in given list of variables 'varIdLst' before 'time', which matches the given value. If multiple variables have a value change before the given 'time' the nearest is returned. If none is found until the optional 'tlimit' is reached '{}' is returned.

Usage:

```
$wread prevvalue time ?timeLimit? varIds value
```

Parameters:

`time`
: Current time.

`timeLimit (optional)`
: Specify a time to stop the search.

`value`
: Possible values are: x/X, z/Z, 0, 1

`varIds`
: List of variable IDs.

Example:

```
set t [$rdr prevvalue 1000 [list $varid] "101X"]
```

$wread real

return the value at the given time. It is an error if the given 'varId' is not a real.

Usage:

```
$wread real time varId
```

Parameters:

`time`
: Current time.

`varId`
: Variable ID.

Example:

```
set v [$rdr real $time $varid]
```

$wread scalar

return the value at the given time. It is an error if the given 'varId' is not a scalar.

Usage:

```
$wread scalar time varId
```

Parameters:

`time`
: Current time.

`varId`
: Variable ID.

Example:

```
set v [$rdr scalar $time $varid]
```

$wread scopedown

return the first scope id of one level below the given scope.

If there is no scope below given scope an empty list '{}' is returned.

If the optional 'scopeid' parameter is missing the top scope, which contains all toplevel scopes, is returned.

Usage:

```
$wread scopedown ?scopeid?
```

Parameters:

`scopeid (optional)`
: Scope ID.

Example:

```
set downScopeId [$rdr scopedown $scopeid]
```

$wread scopefind

search the scope with given name below the given parent scopeid.

If there is no scope found an empty list "{}" is returned.

Usage:

```
$wread scopefind ?scopeid? name
```

Parameters:

`name`
: Name of scope.

`scopeid (optional)`
: Scope ID.

Example:

```
set scopeId [$rdr scopefind "top"]
set scopeId [$rdr scopefind $scopeid "foo"]
```

$wread scopefindnamepath

search the scope with given name path starting at given scope or root.

If there is no scope found an empty list "{}" is returned.

Usage:

```
$wread scopefindnamepath ?scopeid? name
```

Parameters:

`name`
: List of scope names.

`scopeid (optional)`
: Scope ID.

Example:

```
set scopeId [$rdr scopefindnamepath {"top"}]
set scopeId [$rdr scopefind $scopeid "foo"]
```

$wread scopename

return name of given scope.

Usage:

```
$wread scopename scopeid
```

Parameters:

`scopeid`
: Scope ID.

Example:

```
set name [$rdr scopename $scopeid]
```

$wread scopenamepath

Return path of scopenames. Empty names are ignored.

Usage:

```
$wread scopenamepath ?-fromDUT? scopeid
```

Parameters:

`-fromDUT (optional)`

Scope path from DUT.

`scopeid`

Start scope ID.

Example:

```
set newscopeid [$rdr scopenamepath $scopeid]
```

$wread scopepath

Return of path of scopeids.

Usage:

```
$wread scopepath ?-fromDUT? scopeid
```

Parameters:

`-fromDUT (optional)`

Scope path from DUT.

`scopeid`

Start scope ID.

Example:

```
set newscopeid [$rdr scopepath $scopeid]
```

$wread scopeTreeModel

Create a new module list object.

Usage:

```
$wread scopeTreeModel
```

Parameters:

No parameters.

Example:

```
set scopeTreeModel [$rdr scopeTreeModel]
```

$wread scopetype

return type of given scope.

Usage:

```
$wread scopetype scopeid
```

Parameters:

`scopeid`
: Scope ID.

Example:

```
set type [$rdr scopetype $scopeid]
```

$wread scopeup

return the scope id of the parent scope of given scope.

Usage:

```
$wread scopeup scopeid
```

Parameters:

`scopeid`
: Scope ID.

Example:

```
set upScopeId [$rdr scopeup $scopeid]
```

$wread setDUT

set scope and topname to design under test.

Usage:

```
$wread setDUT scopeid topName
```

Parameters:

`scopeid`
: Start scope ID.

`topName`
: Name of top design module.

Example:

```
$rdr setDUT $scopeid "top"
```

$wread signalTreeModel

Create a new module list object.

Usage:

```
$wread signalTreeModel
```

Parameters:

No parameters.

Example:

```
set signalTreeModel [$rdr signalTreeModel $scopeId]
```

$wread signo

return the signal number of the given variable.

Usage:

```
$wread signo varid
```

Parameters:

`varid`
: Variable ID.

Example:

```
set signo [$rdr signo $varid]
```

$wread str2time

return time step in relation to timescale.

Usage:

```
$wread str2time ?-fraction? str
```

Parameters:

`-fraction` (optional)
: allow fraction if string can be converted into timescale.

`str`
: string which should be converted. Special 'first' and 'last' are allowed and return the reader limits.

Example:

```
set str [$rdr str2time 42ns]
```

$wread str2varid

Try to convert a string to varId.

Usage:

```
$wread str2varid str
```

Parameters:

`str`
: The source string

Example:

```
set varid [$rdr str2varid $str]
```

$wread time2pix

return pix index for given time scaled by from and to time and pixelcnt.

Usage:

```
$wread time2pix from to pixelCnt time
```

Parameters:

`from`
: from time step.

`pixelCnt`
: width of drawing rectangle.

`time`
: time step which should be converted into pixel index.

`to`
: to time step.

Example:

```
set pix [$rdr time2pix 0 1000 100 42]
```

$wread time2str

return time step scaled with time factor and unit of timescale.

Usage:

```
$wread time2str ?-fraction? ?-timescale? time
```

Parameters:

`-fraction (optional)`
: do not try to use shortest string.

`-timescale (optional)`
: do not try to use shortest string.

`time`
: time step which should be converted.

Example:

```
set str [$rdr time2str 42]
```

$wread timescale

return timescale string from header.

Usage:

```
$wread timescale
```

Parameters:

Example:

```
set str [$rdr timescale]
```

$wread valarray

return a list with pixel strings for each variable of 'varIdLst'.

Each string is 'pixelCnt' characters long and represents the time range from 'tfrom' to 'tto'.

Each character of the pixel string contains one of the following values:

- ' ' - no values on this pixel
- '0' - all values on this pixel are 0
- '1' - all values on this pixel are 1
- 'B' - constant bus value (no x or z)
- '*' - there are '0' and '1' on this pixel
- 'C' - value change in bus
- 'Z' - all values on this pixel are Z
- 'X' - all values on this pixel are X
- 'Y' - all values in the bus on this pixel are X
- 'z' - multiple values on this pixel, at least one 'Z'
- 'x' - multiple values on this pixel, at least one 'X'

Usage:

```
$wread valarray fromTime toTime vidList pixelCount
```

Parameters:

`fromTime`
: Start time.

`pixelCount`
: Number of available pixels to display.

`toTime`
: End time.

`vidList`
: Number of available pixels to display.

Example:

```
set vidLst {}
$rdr foreach variable vid {
    lappend vidLst $vid
}
set btime 1000
set etime 4000
set result [$rdr valarray $btime $etime $vidLst 80]
foreach pixStr $result {
    # Each pixStr is 80 character long.
}
```

$wread valtype

return the value type of the given variable.

Usage:

```
$wread valtype varid
```

Parameters:

`varid`
: Variable ID.

Example:

```
set valtype [$rdr valtype $varid]
```

$wread value

return the value at the given time. It is an error if the given 'varId' is not a vector.

Usage:

```
$wread value time varId
```

Parameters:

`time`
: Current time.

`varId`
: Variable ID.

Example:

```
set v [$rdr value $time $varid]
```

$wread varaliasfind

find alias and return it's id.

Usage:

```
$wread varaliasfind scopeid type name ?iname?
```

Parameters:

`iname (optional)`
: Instance name of (pin) alias.

`name`
: Name of variable.

`scopeid`

Scope ID.

`type`

alias type of variable.

Example:

```
set newVid [$rdr varaliasfind $scopeid pin "foo"]
```

$wread varaliasinstname

return the inst name of the given alias.

Usage:

```
$wread varaliasinstname varid
```

Parameters:

`varid`

Variable ID.

Example:

```
set iname [$rdr varaliasinstname $varid]
```

$wread varaliastype

return the aliastype of the given variable.

Usage:

```
$wread varaliastype varid
```

Parameters:

`varid`
: Variable ID.

Example:

```
set type [$rdr varaliastype $varid]
```

$wread varbit

Return the bit of the given vector member variable. It is an error if 'vmid' is not a member variable.

Usage:

```
$wread varbit varid
```

Parameters:

`varid`
: Variable ID.

Example:

```
set mem [$rdr varbit $vmid]
```

$wread varbus

return the group or vector variable to which the given member variable belongs. It is an error if 'vmid' is not a member variable.

Usage:

```
$wread varbus varid
```

Parameters:

`varid`
: Variable member ID.

Example:

```
set varid [$rdr varbus $vmid]
```

$wread vardir

return the direction of a variable.

Usage:

```
$wread vardir varid
```

Parameters:

`varid`
: Variable ID.

Example:

```
set d [$rdr vardir $varid]
```

$wread vardirset

set the direction of a variable.

Usage:

```
$wread vardirset varid dir
```

Parameters:

`dir`
: Set the direction.

`varid`
: Variable ID.

Example:

```
$rdr vardirset $varid in
```

$wread varfind

find var and return it's id.

Usage:

```
$wread varfind scopeid name
```

Parameters:

`name`
: Name of variable.

`scopeid`
: Scope ID.

Example:

```
set newVid [$rdr varfind $scopeid "foo"]
```

$wread varhaschanges

Return true if variable id has value changes.

Usage:

```
$wread varhaschanges varid
```

Parameters:

`varid`
: Variable ID.

Example:

```
if {[$rdr varhaschanges $varid]} {
    # ...
}
```

$wread varhasmem

return true if variable id has member variable (e.g. vector or group).

Usage:

```
$wread varhasmem varid
```

Parameters:

`varid`

Variable ID or member ID.

Example:

```
if {[$rdr varhasmem $varid]} {
    # ...
}
```

$wread varhasvectormem

Return true if variable group id contains vector members.

Usage:

```
$wread varhasvectormem varid
```

Parameters:

`varid`

Variable ID.

Example:

```
if {[$rdr varhasvectormem $varid]} {
    # ...
}
```

$wread varhide

return the hide flag of a variable.

Usage:

```
$wread varhide varid
```

Parameters:

`varid`
: Variable ID.

Example:

```
set h [$rdr varhide $varid]
```

$wread varhideset

return the hide flag of a variable or set it.

Usage:

```
$wread varhideset varid hide
```

Parameters:

`hide`
: Set the hide flag on or off.

`varid`
: Variable ID.

Example:

```
$rdr varhideset $varid true
```

$wread varid2str

Try to convert a varId to string.

Usage:

```
$wread varid2str varId
```

Parameters:

`varId`
: The varid which should be converted.

Example:

```
set str [$rdr varid2str $varId]
```

$wread varismem

Return true if variable id is a member variable.

Usage:

```
$wread varismem varid
```

Parameters:

`varid`
: Variable ID or member ID.

Example:

```
if {[$rdr varismem $varid]} {
    # ...
}
```

$wread varmem

Return member index for given member variable. It is an error if 'vmid' is not a member variable.

Usage:

```
$wread varmem varid
```

Parameters:

`varid`

Variable ID.

Example:

```
set found 0
$rdr foreach variable varid {
    if {![$rdr varhasmem $varid]} {
        continue
    }
    $rdr foreach member $varid vmid {
        set found 1
        break
    }
    if {$found} {
        break
    }
}
set mem [$rdr varmem $vmid]
```

$wread varname

return the name of the given variable.

Usage:

```
$wread varname varid
```

Parameters:

`varid`
: Variable ID.

Example:

```
set name [$rdr varname $varid]
```

$wread varscope

Return the scope containing of the given variable.

Usage:

```
$wread varscope varid
```

Parameters:

`varid`
: Variable ID.

Example:

```
set scope [$rdr varscope $varid]
```

$wread vartype

return the type of the given variable.

Usage:

```
$wread vartype varid
```

Parameters:

`varid`
: Variable ID.

Example:

```
set type [$rdr vartype $varid]
```

$wread varwidth

return the width of the given variable.

Usage:

```
$wread varwidth varid
```

Parameters:

`varid`
: Variable ID.

Example:

```
set width [$rdr varwidth $varid]
```

$wread vector

return the value at the given time. It is an error if the given 'varId' is not a vector.

Usage:

```
$wread vector time varId
```

Parameters:

`time`
: Current time.

`varId`
: Variable ID.

Example:

```
set v [$rdr vector $time $varid]
```

$wread version

return version string from header.

Usage:

```
$wread version
```

Parameters:

No parameters.

Example:

```
set str [$rdr version]
```

WDB (Wave Database) API Header File to Create a WDB

API Functions to create a file or in memory Wave Database.

WDB Datatypes

Time Data

All time data uses the WdbTime data type.

```
typedef zUInt64 WdbTime;
#define WdbTimeInvalid  ((WdbTime)(-1))
#define WdbTimeNoChange ((WdbTime)(-2))

typedef enum {
    WdbTimeUnitUndef  = 0,
    WdbTimeUnitSecond = 1,
    WdbTimeUnitMilli  = 2,
    WdbTimeUnitMicro  = 3,
    WdbTimeUnitNano   = 4,
    WdbTimeUnitPico   = 5,
    WdbTimeUnitFemto  = 6
} WdbTimeUnit;
```

Value Types

Signal value use the WdbValue enumeration.

```
typedef enum {
    WdbVx=0,
    WdbVz=1,
    WdbV0=2,
    WdbV1=3
} WdbValue;
```

Variable Types

Variable types use the WdbVarType enumeration.

```
typedef enum {
    WdbVarUnknown,
    WdbVarEvent,
    WdbVarInteger,
    WdbVarParameter,
    WdbVarReal,
    WdbVarReg,
    WdbVarSupply0,
    WdbVarSupply1,
    WdbVarTime,
    WdbVarTri,
    WdbVarTriand,
    WdbVarTrior,
    WdbVarTrireg,
    WdbVarTri0,
    WdbVarTri1,
    WdbVarWand,
    WdbVarWire,
    WdbVarWor,

    WdbVarGroup,
    WdbVarAlias,

    WdbVarInvalid
} WdbVarType;
```

Alias Types

Types of alias variables.

```
typedef enum {
    WdbAliasNone,
    WdbAliasPin,
    WdbAliasPort,
    WdbAliasNet,

    WdbAliasInvalid
} WdbAliasType;
```

Var Dir

Directions of variables.

```
typedef enum {
    WdbVarDirNone,
    WdbVarDirIn,
    WdbVarDirOut,
    WdbVarDirInOut,

    WdbVarDirInvalid
} WdbVarDir;
```

Signal Types

Type of a signal and it's value.

```
typedef enum {
    WdbValUnknown,
    WdbValScalar,
    WdbValVector,
    WdbValReal,

    WdbValMember, /* of a vector or group */
    WdbValGroup,

    WdbValInvalid
} WdbValType;
```

Scope Types

Scope types use the WdbScopeType enumeration.

```
typedef enum {
    WdbScopeUnknown,
    WdbScopeModule,
    WdbScopeTask,
    WdbScopeFunction,
    WdbScopeBegin,
    WdbScopeFork,

    WdbScopeInvalid
} WdbScopeType;
```

String Conversion Helpers

Helper functions to convert types to strings.

```
const char*  WdbValue2Str(WdbValue value);
const char*  WdbVarType2Str(WdbVarType vartype);
const char*  WdbAliasType2Str(WdbAliasType aliastype);
const char*  WdbValType2Str(WdbValType valtype);
const char*  WdbScopeType2Str(WdbScopeType stype);
const char*  WdbVarDir2Str(WdbVarDir vardir);
WdbVarType   WdbStr2VarType(const char* str);
WdbAliasType WdbStr2AliasType(const char* str);
WdbValType   WdbStr2ValType(const char* str);
WdbScopeType WdbStr2ScopeType(const char* str);
WdbVarDir    WdbStr2VarDir(const char* str);
void         WdbVarTypeUsageList(const char*** list);
void         WdbAliasTypeUsageList(const char*** list);
void         WdbValTypeUsageList(const char*** list);
void         WdbScopeTypeUsageList(const char*** list);
void         WdbVarDirUsageList(const char*** list);
```

Wave Database Handle

The opaque Wdb handle is used by all API functions, to store the internal state.

```
typedef struct WdbStruct Wdb;
```

Callbacks

Wdb Callbacks structure is used to generate progress information and message output.

```
typedef struct {
    int  (*progress)(const char* what, double percent);
    void (*msg)(zMessage_Type type, const char* msg);
} WdbCallbacks;
```

Default Callbacks

Default functions for progress and messaging.

```
#define WdbDefaultCallbacks ((WdbCallbacks*)1)
```

No Callbacks

No functions for progress and messaging.

```
#define WdbNoCallbacks ((WdbCallbacks*)NULL)
```

Signal Number

The data type WdbSigNo is returned and used as arguments by the other API functions.

```
typedef int WdbSigNo;

#define WdbInvalidSigNo (-1)
```

Create Options

Options are use to control the creation of the WDB database by the WdbCreateMem and WdbCreateFile functions.

- gcd: saves space by finding the greatest common divisor (GCD) for stored timesteps.
- zip: saves space by compressing a whole buffer with zip.
- skipdupl: save space by ignoring value changes to same value as previous change
- collapse: save space by collapsing multiple value changes at same timestep.
- skipLeadX: ignore value changes until first non X value.

```
```

WDB API Functions

Create a WDB

The WdbCreateMem() function creates a memory-located file (this is not useful for the write interface, but can be used together with the read interface). This function always returns an empty Wdb.

The WdbCreateFile() function creates an external WDB file and uses core memory only for buffering. This function may fail but return a Wdb and set ok to zFalse - call WdbErrorMsg() to get the error message.

During execution the callback functions are called if the are not set to NULL.

```
Wdb* WdbCreateMem(
    zBool zip,
    zBool gcd,
    zBool skipDupl,
    zBool collapse,
    zBool skipLeadX,
    WdbCallbacks*);

Wdb* WdbCreateFile(
    const char* fname,
    zBool zip,
    zBool gcd,
    zBool skipDupl,
    zBool collapse,
    zBool skipLeadX,
    WdbCallbacks*,
    zBool* ok);
```

Restore a WDB File

The WdbRestore() function restores a WDB from the binary wdb file.

If ok is set to zFalse there was an error - call WdbErrorMsg() to get the error message.

During execution the callback functions are called if the are not set to NULL.

```
Wdb* WdbRestore( const char* fname, WdbCallbacks*, zBool* ok);
```

Check a WDB File

The WdbCheck() function tries to open a binary WDB file.

It returns zTrue if everything is ok. If there are problems zFalse is returned. If msg != NULL it contains more details..

```
zBool WdbCheck(const char* fname, WdbCallbacks*, char msg[256]);
```

Close the WDB

The WdbClose() function empties all internal buffer, writes all pending data and close the database. The given handling is not valid anymore.

It returns zTrue if everything is ok. If there are problems zFalse is returned - call WdbErrorMsg() to get the error message.

```
zBool WdbClose(Wdb*);
```

Save the WDB

The WdbSave() function saves the in memory database to a file.

It returns zTrue if everything is ok. If there are problems zFalse is returned - call WdbErrorMsg() to get the error message.

```
zBool WdbSave(Wdb*, const char* fname);
```

Dump the WDB

The WdbMemDump() function dumps the in memory database to a file.

It returns zTrue if everything is ok. If there are problems zFalse is returned - call WdbErrorMsg() to get the error message.

```
zBool WdbMemDump(Wdb*, const char* fname);
```

Store Header Information

The WdbSetDate(), WdbSetVersion() functions are used to store header information in the WDB. The data is store as strings. The WdbSetTimescale() is used to store header information in the WDB needed to convert from WdbTime steps into real time and vice versa.

They return zTrue if everything is ok. If there are problems zFalse is returned - call WdbErrorMsg() to get the error message.

```
zBool WdbSetDate(     Wdb*, const char* str);
zBool WdbSetVersion(  Wdb*, const char* str);
zBool WdbSetTimeScale(Wdb*, unsigned timeFactor, WdbTimeUnit unit);
```

Store Variable and Scope Definitions

All variables belong to a hierarchical scope. During the definition of variables with WdbVar() the scope can be changed by calling WdbScope(). WdbUpScope() or WdbPathScope(). The signal identifier sigNo can be used multiple times in different scopes. The next usable sigNo is returned by a call to WdbNextSigNo().

The end the variable definitions and start adding value changes the function WdbEndDefinitions() needs to be called.

The function WdbAddDefinitions() reenables adding Scopes, Variables.

The function WdbFindVar() returns the sigNo of a named variable or -1 if not found. This works only for in memory databases.

The function WdbFindAlias() returns the sigNo of a named and typed alias variable or -1 if not found. This works only for in memory databases.

The function WdbHideVar() hides the variable referenced by sigNo, if hide is zTrue, otherwise unhides it. This works only for in memory databases.

The functions, except WdbNextSigNo(), return zTrue if everything is ok. If there are problems zFalse is returned - call WdbErrorMsg() to get the error message.

```
zBool    WdbScope(         Wdb*, WdbScopeType, const char* name);
zBool    WdbPathScope(     Wdb*, WdbScopeType, zBool rel,
                                               const char** names, int namesLen);
zBool    WdbUpScope(       Wdb*);
WdbSigNo WdbNextSigNo(     Wdb*);
zBool    WdbVar(           Wdb*, WdbVarType, WdbValType,
                   const char* name, int w, WdbSigNo sigNo);
zBool    WdbAlias(         Wdb*, WdbAliasType,
                   const char* name, int w, WdbSigNo sigNo, const char* iname);
zBool    WdbEndDefinitions(Wdb*);
zBool    WdbAddDefinitions(Wdb*);
zBool    WdbFindVar(       Wdb*, const char* name, WdbSigNo* sigNo);
zBool    WdbFindAlias(     Wdb*, const char* name, WdbAliasType,
                                                   const char* iname, WdbSigNo*);
zBool    WdbHideVar(       Wdb*, const char* name, zBool hide);
zBool    WdbDirVar(        Wdb*, const char* name, WdbVarDir dir);
```

Store Signal Value Changes

The function WdbAddTime() defines a new time which is used for all following value changes until a new call to WdbAddTime() is made.

The function WdbAddScalar() stores a new value for given signal.

WdbAddVector() store a complete bit vector. The given width w must match the width given to WdbVar() during variable definition.

The function WdbDone() marks the end of all value changes.

If there are problems zFalse is returned - call WdbErrorMsg() to get the error message.

NOTE	*Todo* WdbAddEvent() and WdbAddReal() are not implemented yet.

```
zBool WdbAddTime(  Wdb*, WdbTime tm, zBool* stopPtr);
zBool WdbAddScalar(Wdb*, WdbSigNo sigNo, WdbValue val);
zBool WdbAddVector(Wdb*, WdbSigNo sigNo, WdbValue* values, int w);
zBool WdbAddEvent( Wdb*, WdbSigNo sigNo);
zBool WdbAddReal(  Wdb*, WdbSigNo sigNo, double value);
zBool WdbDone(     Wdb*);
```

Get Error Message

The WdbErrorMsg() is used to get a more detailed on the last error occurred.

```
const char* WdbErrorMsg(Wdb*);
```

WDB (Wave Database) API Header File to Access WDB

API Functions to process data contained in the Wave Database.

The read API uses WDB types defined in the wdb.h header file.

WDB-Read Datatypes

Reader Handle

The opaque Wread handle is used by all API functions.

```
typedef struct WreadStruct Wread;
```

Scope ID

The opaque data type WreadScopeId is returned and used as arguments by the other API functions.

```
typedef int WreadScopeId;
#define WreadNullScopeId    (WreadScopeId)(-1)
#define WreadInvalidScopeId (WreadScopeId)(-2)
#define WreadScopeIdIsNull(scpId)    ((scpId) == WreadNullScopeId)
#define WreadScopeIdIsInvalid(scpId) ((scpId) == WreadInvalidScopeId)
```

Variable ID

The opaque data type WreadVarId is returned and used as arguments by the other API functions.

```
typedef struct {
    int id;
    int mem;
} WreadVarId;

#define WreadNullVarId     (-1)
#define WreadInvalidVarId  (-2)
#define WreadFirstGrpVarId (-3)
#define WreadFirstCloneVarId -((int)(((unsigned)-1)>>2))
#define WreadNullMem       (((unsigned)-1)>>2)

#define WreadVarIdIsNull(varId)    ((varId).id  == WreadNullVarId)
#define WreadVarIdIsInvalid(varId) ((varId).id  == WreadInvalidVarId)
#define WreadVarIdIsNullMem(varId) ((varId).mem == WreadNullMem)
#define WreadVarIdSet(varId,i,m)   ((varId).id  = (i), (varId).mem = (m))
```

WDB-Read API Functions

Allocate and Free a WDB Reader

All reader access functions need a 'Wread*' handle, which is allocated by WreadAlloc() and freed by WreadFree(). After a call to WreadFree() the handle is invalid.

```
Wread* WreadAlloc(Wdb*);
void   WreadFree( Wread*);
```

Get Associated Wdb

```
Wdb*   WreadGetWdb(Wread*);
```

Get Header Information

The functions WreadDate() and WreadVersion() return date and version string. The functions WreadTimeFactor() and WreadTimeUnit() return timescale information.

```
const char* WreadDate(     Wread*);
const char* WreadVersion(  Wread*);
unsigned    WreadTimeFactor(Wread*);
WdbTimeUnit WreadTimeUnit(  Wread*);
```

Convert Functions

The function WreadTime2Str() converts a given time step into a string. The returned value contains a integer followed by the time unit. If useBest is true it uses the smallest possible integer, by changing the unit given by timescale. The returned string is only valid until the next call to a Wread

API function. The functions WreadStr2Time converts a string including unit into a time step. The functions WreadTime2Pix and WreadPix2Time convert pixel coordinates into time steps and vice versa. They return -1 or WdbTimeInvalid for wrong parameter. The function WreadCalcZoom changes the tfrom and tto time by a factor relative to the time range and tcenter. Factor can be positive to zoom in, or negative to zoom out. tcenter time is tried to stay near the same pixel position. If tcenter is WdbTimeInvalid the center of tfrom/tto is used.

```
const char* WreadTime2Str(Wread*, WdbTime, zBool useFraction, zBool useBest);
WdbTime     WreadStr2Time(Wread*, const char* str, zBool allowFraction);
int         WreadTime2Pix(Wread*, WdbTime tfrom, WdbTime tto,
                                                 int pixelCnt, WdbTime t);
WdbTime     WreadPix2Time(Wread*, WdbTime tfrom, WdbTime tto,
                                                 int pixelCnt, int pix);

zBool       WreadCalcZoom(Wread* wread, WdbTime* tfrom, WdbTime* tto,
                          double factor, WdbTime tmarker);
```

Get Scope Information

The functions WreadScopeType() WreadScopeName() return the type and name of a given scope.

The function WreadScopeDown() is used to go down the hierarchy from a given scope, using 'WreadNullScopeId' as a start value.

The function WreadScopeSibling() returns the next scope in the same hierarchy as the given scope.

The function WreadScopeFind() search a scope under the given scope, using 'WreadNullScopeId' as the root value. Returns the scopeId or WreadNullScopeId if not found.

The function WreadScopeFindNamePath works like WreadScopeFind, but searches a list of scope names down the hierarchy, stating at the given scope or root.

In the case of an error the functions return 'WreadInvalidScopeId'.

The function WreadScopeDownVar() return the first variable for given scope or 'WreadNullScopeId' is there are no variables. In the case of an error the function return 'WreadInvalidVarId'.

The function WreadScopePath() is a service function to set a scope list (and it's length) containing all scopes from top to the given scope. In case of an error False is returned. If fromDUT is true, the path does begin at the DUT hierarchy.

The function WreadScopeNamePath() is a service function works like WreadScopePath() but returns the names of the scopes. If fromDUT is true, the path does begin at the DUT hierarchy. The top name is taken from the DUTtopName.

```
WdbScopeType  WreadScopeType(   Wread*, WreadScopeId);
const char*   WreadScopeName(   Wread*, WreadScopeId);
WreadScopeId  WreadScopeUp(     Wread*, WreadScopeId);
WreadScopeId  WreadScopeDown(   Wread*, WreadScopeId);
WreadScopeId  WreadScopeSibling(Wread*, WreadScopeId);
WreadScopeId  WreadScopeFind(   Wread*, WreadScopeId, const char* name);
WreadScopeId  WreadScopeFindNamePath(Wread*, WreadScopeId, const char**, int);
WreadScopeId  WreadScopeFindIcase(   Wread*, WreadScopeId, const char* name,
    ce_Bool icase);
WreadVarId    WreadScopeDownVar(Wread*, WreadScopeId);
zBool         WreadScopePath(   Wread*, WreadScopeId, zBool fromDUT,
                                                    WreadScopeId**, int* len);
zBool         WreadScopeNamePath(Wread*, WreadScopeId, zBool fromDUT,
                                                    const char***, int* len);
```

DUT Information

The functions WreadSetDUT() and WreadDUTscope()/WreadDUTtopName set and get the DUT scope and top name.

```
void         WreadSetDUT(    Wread*, WreadScopeId, const char* topName);
WreadScopeId WreadDUTscope(  Wread*);
const char*  WreadDUTtopName(Wread*);
```

Get Variable Information

The functions WreadVarType() WreadVarName() return the type and name of a given variable.

The function WreadVarScope() is used to get the scope of a given variable.

The function WreadVarSibling() returns the next scope in the same hierarchy as the given variable.

WreadVarWidth() gets the width of a vector or group. All other types get 0.

WreadVarBit() gets the bit number of a vector member

The functions WreadVarFirst() and WreadVarNext() can be used to iterate over all variables.

The functions WreadVarFirstMem(), WreadVarLastMem, WreadVarNextMem() and WreadVarPrevMem() can be used to iterate over all members of one variable. Returns 'WreadNullVarId' if there no more members.

WreadVarSibling(), WreadVarFirst() and WreadVarNext() return 'WreadNullVarId' if there no more variables.

WreadMem2Var() returns the parent variable of a member variable.

WreadVar2MemIdx() returns the member index.

WreadVar2Bus() returns the group or vector of the member.

WreadVarHasMem() get zTrue if id has member variable (vector or group).

WreadVarHasChanges() get zTrue if there are any value changes.

WreadVarHasVectorMem() get zTrue if group contains vector members.

WreadVarFind() returns a variable with the given name under the given scope or WreadNullVarId if it is not found.

WreadVarAliasType() returns the alias type or VarAliasInvalid for 'normal' variables.

WreadVarAliasInstName() return the inst name of pin aliases.

In the case of an error the functions return 'WreadInvalidVarId' or zFalse.

```
WdbVarType   WreadVarType(   Wread*, WreadVarId);
const char*  WreadVarName(   Wread*, WreadVarId);
WreadScopeId WreadVarScope(  Wread*, WreadVarId);
WreadVarId   WreadVarSibling(Wread*, WreadVarId);
WreadVarId   WreadVarFirst(  Wread*);
WreadVarId   WreadVarNext(   Wread*, WreadVarId);
zBool        WreadVarWidth(  Wread*, WreadVarId, int*);
zBool        WreadVarBit(    Wread*, WreadVarId, int*);
WreadVarId   WreadVarFirstMem(Wread*, WreadVarId);
WreadVarId   WreadVarNextMem( Wread*, WreadVarId);
WreadVarId   WreadVarLastMem( Wread*, WreadVarId);
WreadVarId   WreadVarPrevMem( Wread*, WreadVarId);
WreadVarId   WreadMem2Var(    Wread*, WreadVarId);
int          WreadVar2MemIdx( Wread*, WreadVarId);
WreadVarId   WreadVar2Bus(    Wread*, WreadVarId);
zBool        WreadVarHasMem(  Wread*, WreadVarId);
zBool        WreadVarHasChanges(Wread*, WreadVarId, zBool*);
zBool        WreadVarHasVectorMem(Wread*, WreadVarId, zBool*);
WreadVarId   WreadVarFind(    Wread*, WreadScopeId, const char* name);
WreadVarId   WreadVarAliasFind(Wread*, WreadScopeId, WdbAliasType,
                        const char* name, const char* iname);
WdbAliasType WreadVarAliasType(    Wread*, WreadVarId);
const char*  WreadVarAliasInstName(Wread*, WreadVarId);
```

Hide Variable

WreadVarSetHide() and WreadVarGetHide() get be used to set/get the hide flag.

In the case of an error the functions return zFalse.

```
zBool WreadVarSetHide( Wread*, WreadVarId, zBool);
zBool WreadVarGetHide( Wread*, WreadVarId, zBool*);
```

Dir Variable

WreadVarSetDir() and WreadVarGetDir() get be used to set/get the direction.

In the case of an error the functions return zFalse.

```
zBool WreadVarSetDir( Wread*, WreadVarId, WdbVarDir);
zBool WreadVarGetDir( Wread*, WreadVarId, WdbVarDir*);
```

Get Signal Change Times

The functions WreadTimeFirst() and WreadTimeLast() return the minimum and maximum time.

The functions WreadTimePrev() and WreadTimeNext() return the time of the previous/next value change of the given array of variables 'varIdLst' starting with the given 'time' and a limit of 'tlimit'. The argument 'varIdLstLen' gives the size of the 'varIdLst' array. If no value change exits 'WdbTimeNoChange' is returned.

If there is an error 'WdbTimeInvalid' is returned.

```
WdbTime WreadTimeFirst(Wread*);
WdbTime WreadTimeLast( Wread*);

WdbTime WreadTimePrev(Wread*, WdbTime time, WdbTime tlimit,
                      WreadVarId varIdLst[], int varIdLstLen);
WdbTime WreadTimeNext(Wread*, WdbTime time, WdbTime tlimit,
                      WreadVarId varIdLst[], int varIdLstLen);
```

Get the Value Type of a Variable

The function WreadValType() returns the type of a given signal. If there is an error 'WdbValInvalid' is returned.

```
WdbValType WreadValType(Wread*, WreadVarId varId);
```

Get the Signal No of a Variable

The function WreadSigNo() returns the signal number of a given signal. Multiple variables may return the same signal number. If there is an error 'WdbValInvalid' is returned.

```
int WreadSigNo(Wread*, WreadVarId varId);
```

Get Signal Value Changes

The functions WreadValueScalar() and WreadValueVector() set the value for a given variable at the given time into 'val'. For a vector the argument 'sz' defines the available space.

NOTE

Todo

The function WreadValueReal() is not yet implemented.

```
zBool WreadValueScalar(Wread*, WdbTime, WreadVarId, WdbValue*);
zBool WreadValueVector(Wread*, WdbTime, WreadVarId, WdbValue*, int sz);
zBool WreadValueReal(  Wread*, WdbTime, WreadVarId, double*);
zBool WreadValueMember(Wread*, WdbTime, WreadVarId, WdbValue*, int sz);
zBool WreadValueGroup( Wread*, WdbTime, WreadVarId, WdbValue*, int sz);

zBool WreadValueScalarS(Wread*, WdbTime, WreadVarId, const char**);
zBool WreadValueVectorS(Wread*, WdbTime, WreadVarId, const char**);
zBool WreadValueRealS(  Wread*, WdbTime, WreadVarId, const char**);
zBool WreadValueMemberS(Wread*, WdbTime, WreadVarId, const char**);
zBool WreadValueGroupS( Wread*, WdbTime, WreadVarId, const char**);
zBool WreadValueS(      Wread*, WdbTime, WreadVarId, const char**);
```

The function WreadValArray() sets an internal buffer with "pixel"-information for a width of 'pixelCnt' corresponding to a time range 'tfrom' to 'tto' (and + 1 '\0'). The start of each line is set into pixelLinesPtr. The number of lines (which is the same as valIdLstLen) is set into pixelLineCntPtr for a consistent API interface.

Each "pixel" stores one character depending on the different values which lie an this "pixel".

- ' ' - no values on this pixel
- '0' - all values on this pixel are 0
- '1' - all values on this pixel are 1
- 'B' - constant bus value (no x or z)
- '*' - there are '0' and '1' on this pixel
- 'C' - value change in bus
- 'Z' - all values on this pixel are Z
- 'X' - all values on this pixel are X
- 'Y' - all values in the bus on this pixel are X
- 'z' - multiple values on this pixel, at least one 'Z'
- 'x' - multiple values on this pixel, at least one 'X'

If there is an error '0' is returned.

```
zBool WreadValArray(
    Wread* wread,
    WdbTime tfrom,
    WdbTime tto,
    WreadVarId* varIdLst,
    int varIdLstLen,
    int pixelCnt,
    const char*** pixelLinesPtr,
    int* pixelLineCntPtr
);
```

Group Scalars to Virtual Vectors

- WreadGroupCreate - return group varId or WreadInvalidVarId.
- WreadGroupDelete - return zFalse on error.
- WreadGroupFirst - returns group varId or WreadNullVarId.
- WreadGroupNext - returns group varId or WreadNullVarId.
- WreadGroupBuild - find groups by looking at scalar bus member names.
- WreadGroupRename - set new name for given group.
- WreadGroupRedefine - set new member list for group. The varIdList which defines the members of the group is stored with the LSB first.

```
WreadVarId   WreadGroupCreate( Wread*,
                               const char* name,
                               WreadVarId varIdList[], int len,
                               WreadScopeId,
                               WdbVarType varType,
                               zBool hide,
                               zBool autoScope,
                               zBool expandVectors
                               );
zBool        WreadGroupDelete( Wread*, WreadVarId);
WreadVarId   WreadGroupFirst(  Wread*);
WreadVarId   WreadGroupNext (  Wread*, WreadVarId);
zBool        WreadGroupBuild(  Wread*, WreadScopeId,
                                    WdbVarType varType, zBool hide,
                                    zBool rangeDown);
zBool        WreadGroupRename( Wread*, WreadVarId, const char* name);
zBool        WreadGroupRedefine(Wread*,WreadVarId,
                               WreadVarId varIdLst[],
                               int len,
                               zBool hide,
                               zBool expandVectors
                               );

WreadVarId WreadGroupFind(
    Wread*      wread,
    const char* groupName
);
```

Clones

- WreadCloneCreate - return Clone varId or WreadInvalidVarId.
- WreadCloneDelete - return zFalse on error.
- WreadCloneFirst - returns Clone varId or WreadNullVarId.
- WreadCloneNext - returns Clone varId or WreadNullVarId.

```
WreadVarId   WreadCloneCreate( Wread*, WreadVarId varId);
zBool        WreadCloneDelete( Wread*, WreadVarId);
WreadVarId   WreadCloneFirst(  Wread*);
WreadVarId   WreadCloneNext (  Wread*, WreadVarId);
```

Get Time For Given Value

The functions WreadValuePrev() and WreadValueNext() return the time of the previous/next value change of the given array of variables 'varIdLst' starting with the given 'time' and a limit of 'tlimit', which matches the given value. The argument 'varIdLstLen' gives the size of the 'varIdLst' array. If no matching value exits 'WdbTimeNoChange' is returned.

If there is an error 'WdbTimeInvalid' is returned.

```
WdbTime WreadValuePrev(Wread*, WdbTime time, WdbTime tlimit,
                       WreadVarId varIdLst[], int varIdLstLen,
                       WdbValue* values, int w);
WdbTime WreadValueNext(Wread*, WdbTime time, WdbTime tlimit,
                       WreadVarId varIdLst[], int varIdLstLen,
                       WdbValue* values, int w);
```

Get Error Message

If there was an error reported by one of the Wread API functions WreadErrorMsg returns a detailed message.

```
const char* WreadErrorMsg(Wread*);
```

Set Error Message

Set API error message. For convenience returns always false.

```
zBool WreadSetErrorMsg(Wread*, const char* fmt, ...);
```

Get Debug Flag For Util

Return true if util debug messages are enabled.

```
zBool  WreadDebugUtil(Wread*);
```

Scope Iterator Functions

Scope iterator support.

```
void* WreadForeachScopeInit(Wread* ,WreadScopeId);
zBool WreadForeachScopeNext(Wread* ,void*, WreadScopeId*);
void WreadForeachScopeFinit(Wread* ,void*);
```

Var Iterator Functions

Var iterator support.

```
void* WreadForeachVarInit(Wread* ,WreadScopeId);
zBool WreadForeachVarNext(Wread* ,void*, WreadVarId*);
void WreadForeachVarFinit(Wread* ,void*);
```

Clone Iterator Functions

Clone iterator support.

```
void* WreadForeachCloneInit(Wread* ,WreadScopeId);
zBool WreadForeachCloneNext(Wread* ,void*, WreadVarId*);
void WreadForeachCloneFinit(Wread* ,void*);
```

Group Iterator Functions

Group iterator support.

```
void* WreadForeachGroupInit(Wread* ,WreadScopeId);
zBool WreadForeachGroupNext(Wread* ,void*, WreadVarId*);
void WreadForeachGroupFinit(Wread* ,void*);
```

VarItem Iterator Functions

VarItem iterator support.

```
void* WreadForeachVarItemInit(Wread* ,WreadScopeId);
zBool WreadForeachVarItemNext(Wread* ,void*, WreadVarId*);
void WreadForeachVarItemFinit(Wread* ,void*);
```

Member Iterator Functions

Member iterator support.

```
void* WreadForeachMemberInit(Wread* ,WreadVarId,
                             zBool lsbFirst, zBool msbFirst);
zBool WreadForeachMemberNext(Wread* ,void*, WreadVarId*);
void WreadForeachMemberFinit(Wread* ,void*);
```

WreadVarId2Str

Converts a VarId to string. result needs to be freed.

```
zBool WreadVarId2Str(
    Wread*       reader,
    WreadVarId   varId,
    char**       str
);
```

WreadStr2VarId

Converts a string to VarId.

```
zBool WreadStr2VarId(
    Wread*       reader,
    const char*  str,
    WreadVarId*  varId
);
```

Usage Example

The following examples show how the above functions are used together.

Open a WDB for Reading and Create a Reader

WdbRestore is used to open an existing WDB file. WreadAlloc then creates a reader, which must be freed after use.

Example

```
#include "wdb/wdbread.h"
      zBool  ok = zTrue;
      Wdb*   wdb;
      Wread* rdr;

      wdb = WdbRestore("file.wdb", &callbacks, &ok);
      if (!ok) ...
      rdr = WreadAlloc(wdb);
      ...
      WreadFree(wdb);
```

Get Header Information

WreadDate, WreadVersion, WreadTimeScale are used to retrieve header information.

Example

```
Wread*      rdr;
      const char* str;
      ...
      str = WreadDate(rdr);
      str = WreadVersion(rdr);
      str = WreadTimeScale(rdr);
```

Get Scope Information

The hierarchical scope data is stored in a tree structure. It can be recursively traversed with the WreadScopeDown, WreadScopeSibling functions. The type, name and first variable of the scope can then be retrieved.

Example

```
...
      void traverseScopes(Wread* rdr, WreadScopeId* parent) {
          WreadScopeId* scopeid;
          scopeid = WreadScopeDown(rdr, parent);
          while (scopeid != WreadNullScopeId) {
              WdbScopeType scopetype;
              const char*  name;
              WreadVarId   vid;
              scopetype = WreadScopeType(rdr, scopeid);
              name      = WreadScopeName(rdr, scopeid);
              vid       = WreadScopeDownVar(rdr, scopeid);
              ...
              traverseScopes(rdr, scopeid);
              scopeid = WreadScopeSibling(rdr, scopeid);
          }
      }
      ...
      traverseScopes(rdr, WreadNullScopeId);
```

Get Variable Information

The variables can be iterated with two pair functions. Use WreadVarFirst and WreadVarNext for looping over all variables.

Example

```
WreadVarId  vid;
      ...
      vid = WreadVarFirst(rdr);
      while (vid != WreadNullVarId) {
          ... process all variables
          vid = WreadVarNext(rdr, vid);
      }
```

Use WreadScopeDownVar and WreadVarSibling for looping over all variables of a given scope.

Example

```
WreadScopeId* scopeid;
      ...
      vid = WreadScopeDownVar(rdr, scopeid);
      while (vid != WreadNullVarId) {
          ... process variables of current scope
          vid = WreadVarSibling(rdr, vid);
      }
```

Get Value Changes

The value changes of a signal can be retrieved at any timestep with WreadScalarValue, WreadVectorValue etc. To get timesteps where some signals really change, WreadTimePrev or WreadTimeNext can be used.

Example

```
WdbTime    t1, t2, t3;
      WreadVarId vid1, vid2;
      WreadVarId vidList[2];
      WdbValue   val1;
      int        width;
      WdbValue   val2[10];
      const char* str;
      ...
      t1 = WreadTimeFirst(rdr);
      t2 = WreadTimeLast(rdr);
      ...
      ok = WreadScalarValue(rdr, t1+100, vid1, &val1);
      ok = WreadVectorValue(rdr, t2-500, vid2, val2, width);
      ...
      vidList[0] = vid1;
      vidList[1] = vid2;
      t3 = WreadTimePrev(rdr, t1+200, 10000, vidList, 2);
      t3 = WreadTimeNext(rdr, t1+900, 0,     vidList, 2);

      str = WreadTime2Str(rdr, t3);
```

The Parser API

The RTL Parser

This document describes the function of the software unit that reads and elaborates RTL files and stores the connectivity (with parameters) in the hierarchical database ZDB for later processing. It supports all variants of Verilog, SystemVerilog and VHDL.

Overview

Introduction

In addition of using the RTLvision PRO GUI to read RTL files either the stand alone binary `rtl2zdb` or the Tcl command `zrtl` can be used.

The `rtl2zdb` binary reads the given RTL files and creates a binfile containing the elaborated netlist of the input files:

```
rtl2zdb -o binfile.zdb <OPTIONS> <FILES>
```

The `zrtl` Tcl command reads the given RTL files and returns the elaborated netlist of the input files as an in-memory database:

```
set db [zrtl <OPTIONS> <FILES>]
```

The RTL parser can use precompiled VHDL libraries. The precompiled library format is called VDB and can be created using the RTLvision PRO GUI, the `vhdl2vdb` stand alone binary or the Tcl command `zvdb`.

The Tcl commands are available in the Console window of the RTLvision PRO GUI .

This document describes the usage of this flow.

RTL Parser Options

Option	Parameters	Description
`-allowRamInLoop`	`on|off`	Elaborate RAM used within loops. (The default value for this option is `on`.)
`-argsFromFile`	`<file>`	Read cmdline arguments from file.

Option	Parameters	Description
-binlib	<file>	Open this binfile as a precompiled library.
-breakOnError | -breakOnErr	on|off	Stop on errors during parsing. (The default value for this option is off.)
-compact	off|low|med|full	Adjust the level of compaction for a RTL schematic. (The default value for this option is med.)
-compileMode	default|mfcu|sfcu	Set the RTL compile mode. Mode can either be the default of the specified input language, multi file compilation unit (mfcu) or single file compilation unit (sfcu). (The default value for this option is default.)
-connectByName	<netnamepattern>	Connect matching net by name, i.e. don't route it.
-createBus	on|off	Create buses for ports with Verilog conform, consecutive numbering. (The default value for this option is off.)
-createMacroObjects	on|off	Store macro definition and reference info in virtual objects. (The default value for this option is off.)
-createUniqConsts	on|off	Create unique nets for all constants. (The default value for this option is off.)
-cse	on|off	Perform common subexpression elimination and merge logic that creates the same functional behavior. (The default value for this option is off.)
-debugFlag	<flag>	Enable a specific debug flag.
-define	<macro>	Define a Verilog macro on the command line.
+define+	<m>=<v>	Define Verilog macros on the command line.
-defParam	<param>=<value>	Define generic value (VHDL only).
-dontCut		Don't cut long filenames in messages.
-dontElaborate	<pattern>	Don't elaborate modules matching the name pattern.
-F	<fileset>	Read Verilog fileset file. The files in the fileset are relative to the fileset file.
-f	<fileset>	Read Verilog fileset file. Files in the fileset are relative to the current working directory.
-forceSymlib		Overwrite existing symbols with the symbols from the given symbol library files (option -symlib).
-funcHier	on|off	Create hierarchy for function calls. (The default value for this option is off.)

Option	Parameters	Description
-globalInclude	<file>	Define global Verilog include files. Global includes are processed before any other source files (this option can be repeated multiple times).
-help | -h		Print a help text with a short description of each option.
-hierSep	<hiersepchar>	Set the desired hierarchy separator 'hiersepchar' of your choice. Any character can be used. To be able to identify the hierarchy separator, a character that is not already used in an identifier should be used.
-ignoreCase	on|off	Case-insensitive parser. (The default value for this option is on.)
-ignorePragmas	on|off	Ignore all Pragmas. (The default value for this option is off.)
-ignoreTranslate	on|off	Ignore only Translate and Synthesis Pragmas. (The default value for this option is off.)
-ignoreUnit	<unit>	Do not elaborate the VHDL unit <unit>; any VHDL unit whose name matches <unit> case-insensitively will not be elaborated.
-incdir	<dir>	Define an include directory (this option can be repeated multiple times).
+incdir+	<d1>+<d2>…	Define include directories.
-info	None|Error|Warning|Verbose|Debug	Level of verbosity for issued messages. (The default value for this option is Error.) (This option is only available for the rtl2zdb executable.)
-into	<dbname>	Compile the given files into an already existing database. (This option is only available for the zrtl command.)
-L	<library>	Search for Verilog modules and packages in <library>; multiple libraries can be specified using multiple -L options; libraries are searched in the order of the -L options.
+libext+	<e1>+<e2>…	Define the file name extensions for the -y option.
-library	<name>	Parse all files following this option into a library with the given name.
-localdefine	<macro>	Define a Verilog macro local to the next file.
+localdefine+	<m>=<v>	Define Verilog macros on the command line.
-localincdir	<dir>	Define an include directory local to next file(this option can be repeated multiple times).

Option	Parameters	Description
+localincdir+	<d1>+<d2>…	Define include directories local to next file.
-logAppend		Append to logfile.
-logfile	<file>	Generate log file. (This option is only available for the rtl2zdb executable.)
-maxErrCnt	<num>	Set maximum number of errors that can occur before the parser stops reading the input file(s). (The default value for this option is 0.)
-minRamSize	<min>	Minimum size (in bits) a RAM needs to be before RTL elaboration extracts it. Value 0 means no lower limit. (The default value for this option is 4096.)
-netlistPattern	<pattern>	Matching files are structural Verilog files.
-noHierSep		Don’t search for a hierarchy separator.
-nologo		Print no greeting message.
-o | -out	<file>	Name of the zdb output binfile. (This option is only available for the rtl2zdb executable.)
-operContents	on|off	Create operator implementation. (The default value for this option is on.)
-pedantic	on|off	Toggle pedantic language checking mode (in relaxed mode some errors are just warnings and some warnings are suppressed). (The default value for this option is on.)
-preProcessOutputFile	<fileName>	Preprocess verilog macros and includes. Write output to given file.
-preserveAssign	on|off	Preserve assignments in the netlist. (The default value for this option is off.)
-preserveX	on|off	Preserve X values when generating netlists from RTL. (The default value for this option is on.)
-procHier	on|off	Create hierarchy for always and process blocks. (The default value for this option is off.)
-prog		Print progress information. (This option is only available for the rtl2zdb executable.)
-renameSuffix	<suffix>	Suffix for renaming duplicate cells.
-resolveDuplicates	on|off	Resolve duplicate cells. If "off" all duplicate cells are renamed. (The default value for this option is on.)
-sdbl	<file>	Name of Verific sdbl file.

Option	Parameters	Description
-spos	on|off	Create source code references. (The default value for this option is on.)
-suppress	<type> <pattern>	Suppress messages which match pattern. (This option is only available for the rtl2zdb executable.)
-sym2zdb		Preload symbol library file(s) given with the -symlib option.
-symlib	<symlib>	Specify a symbol library file.
-systemVerilog | -sysverilog | -sverilog | -sv		Read RTL SystemVerilog 2009.
+systemverilogext+	<e1>+<e2>…	Define SystemVerilog file name extensions.
-sysVerilog2005		Read RTL SystemVerilog 2005.
-time		Print CPU time consumption (requires enabled progress updates). (This option is only available for the rtl2zdb executable.)
-top	<name>	Define this module as the top module. If * is set, then all unreferenced cells are used as top.
-topLibrary	<name>	The library containing the top-level design.
-v	<libfile>	Read <libfile> as Verilog library file.
-validate		Validate DB before creating the zdb output file.
+verilog1995ext+	<e1>+<e2>…	Define Verilog 1995 file name extensions.
-verilog2001 | -v2k | +v2k		Read RTL Verilog 2001.
+verilog2001ext+	<e1>+<e2>…	Define Verilog 2001 file name extensions.
-verilog95		Read RTL Verilog 95.
-verilogAMS		Read RTL Verilog AMS.
-vhdl2000		Read RTL VHDL 2000.
-vhdl2008		Read RTL VHDL 2008.
-vhdl2019		Read RTL VHDL 2019.
-vhdl87		Read RTL VHDL 87.
-vhdl93		Read RTL VHDL 93.
-vhdlLibPath	<dir>	Look for and store precompiled VHDL libraries in <dir>.

Option	Parameters	Description
-wait_for_license | -waitForLicense	<sec>	Wait 'sec' seconds for a license. If the value is -1 then the started tool will not wait for the next free license. Use a value of 0 to wait forever. (The default value for this option is -1.) (This option is only available for the rtl2zdb executable.)
-y	<libdir>	Read files matching the extension given with +libext+ from the specified directory as Verilog library files.

VDB Creation

The vhdl2vdb binary reads the given VHDL files and creates a binfile for later use by the RTL parser:

```
vhdl2vdb -vhdlLibPath /path/to/vdbdir <OPTIONS> <FILES>
```

The zvdb Tcl command reads the given VHDL files and creates a binfile for later use by the RTL parser:

```
zvdb -vhdlLibPath /path/to/vdbdir <OPTIONS> <FILES>
```

VDB Options

Option	Parameters	Description
-argsFromFile	<file>	Read cmdline arguments from file.
-breakOnError | -breakOnErr	on|off	Stop on errors during parsing. (The default value for this option is off.)
-connectByName	<netnamepattern>	Connect matching net by name, i.e. don't route it.
-debugFlag	<flag>	Enable a specific debug flag.
-dontCut		Don't cut long filenames in messages.
-help | -h		Print a help text with a short description of each option.
-ignorePragmas	on|off	Ignore all Pragmas. (The default value for this option is off.)
-ignoreTranslate	on|off	Ignore only Translate and Synthesis Pragmas. (The default value for this option is off.)
-ignoreUnit	<unit>	Do not elaborate the VHDL unit <unit>; any VHDL unit whose name matches <unit> case-insensitively will not be elaborated.
-info	None|Error|Warning |Verbose|Debug	Level of verbosity for issued messages. (The default value for this option is Error.)

Option	Parameters	Description
-library	<name>	Parse all files following this option into a library with the given name.
-logAppend		Append to logfile.
-logfile	<file>	Generate log file.
-maxErrCnt	<num>	Set maximum number of errors that can occur before the parser stops reading the input file(s). (The default value for this option is 0.)
-nologo		Print no greeting message.
-pedantic	on|off	Toggle pedantic language checking mode (in relaxed mode some errors are just warnings and some warnings are suppressed). (The default value for this option is on.)
-prog		Print progress information.
-renameSuffix	<suffix>	Suffix for renaming duplicate cells.
-resolveDuplicates	on|off	Resolve duplicate cells. If "off" all duplicate cells are renamed. (The default value for this option is on.)
-spos	on|off	Create source code references. (The default value for this option is on.)
-suppress	<type> <pattern>	Suppress messages which match pattern.
-time		Print CPU time consumption (requires enabled progress updates).
-topLibrary	<name>	The library containing the top-level design.
-unit	<name>	Unit name for the following files.
-vhdl2000		Read RTL VHDL 2000.
-vhdl2008		Read RTL VHDL 2008.
-vhdl2019		Read RTL VHDL 2019.
-vhdl87		Read RTL VHDL 87.
-vhdl93		Read RTL VHDL 93.
-vhdlLibPath	<dir>	Look for and store precompiled VHDL libraries in <dir>.
-wait_for_license | -waitForLicense	<sec>	Wait 'sec' seconds for a license. If the value is -1 then the started tool will not wait for the next free license. Use a value of 0 to wait forever. (The default value for this option is -1.)

The Verilog Netlist Parser

This document describes the function of the software unit that reads Verilog netlist files and stores the connectivity (with parameters) in the hierarchical database ZDB for later processing.

Overview

Introduction

In addition of using the RTLvision PRO GUI to read Verilog netlist files either the stand alone binary `verilog2zdb` or the Tcl command `zverilog` can be used.

The `verilog2zdb` binary reads the given Verilog files and creates a binfile containing the connectivity of the input file:

```
verilog2zdb -o binfile.zdb <OPTIONS> <FILE.V>
```

The `zverilog` Tcl command reads the given Verilog files and returns the created in-memory database:

```
set db [zverilog <OPTIONS> <FILE.V>]
```

The Tcl command is available in the Console window of the RTLvision PRO GUI .

This document describes the usage of this flow.

Options

Option	Parameters	Description
`-argsFromFile`	`<file>`	Read cmdline arguments from file.
`-binlib`	`<file>`	Open this binfile as a precompiled library.
`-breakOnError` | `-breakOnErr`	`on|off`	Stop on errors during parsing. (The default value for this option is `off`.)
`-connectByName`	`<netnamepattern>`	Connect matching net by name, i.e. don't route it.
`-createBus`	`on|off`	Create buses for ports with Verilog conform, consecutive numbering. (The default value for this option is `off`.)

Option	Parameters	Description
-createHier	<sep>	Create hierarchy from flat instance names. Split the instance names at the given hierarchy separator. If the given hierarchy separator character is an empty string then the hierarchy separator character is guessed.
-createUniqConsts	on|off	Create unique nets for all constants. (The default value for this option is off.)
-debugFlag	<flag>	Enable a specific debug flag.
-define	<macro>	Define a Verilog macro on the command line.
+define+	<m>=<v>	Define Verilog macros on the command line.
-dontCut		Don't cut long filenames in messages.
-F	<fileset>	Read Verilog fileset file. The files in the fileset are relative to the fileset file.
-f	<fileset>	Read Verilog fileset file. Files in the fileset are relative to the current working directory.
-forceSymlib		Overwrite existing symbols with the symbols from the given symbol library files (option -symlib).
-globalInclude	<file>	Define global Verilog include files. Global includes are processed before any other source files (this option can be repeated multiple times).
-guessBus	<open> <close>	Guess buses based on net and port names with a bit subscript enclosed in the given 'open' and 'close' characters.
-guessInstArray	<open> <close>	Guess instance arrays based on instance names with a bit subscript enclosed in the given 'open' and 'close' characters.
-help | -h		Print a help text with a short description of each option.
-hierSep	<hiersepchar>	Set the desired hierarchy separator 'hiersepchar' of your choice. Any character can be used. To be able to identify the hierarchy separator, a character that is not already used in an identifier should be used.
-incdir	<dir>	Define an include directory (this option can be repeated multiple times).
+incdir+	<d1>+<d2>…	Define include directories.
-info	None|Error|Warning|Verbose|Debug	Level of verbosity for issued messages. (The default value for this option is Error.) (This option is only available for the verilog2zdb executable.)

Option	Parameters	Description
-into	<dbname>	Compile the given files into an already existing database. (This option is only available for the zverilog command.)
+libext+	<e1>+<e2>···	Define the file name extensions for the -y option.
-logAppend		Append to logfile.
-logfile	<file>	Generate log file. (This option is only available for the verilog2zdb executable.)
-netlistPattern	<pattern>	Matching files are structural Verilog files.
-noHierSep		Don’t search for a hierarchy separator.
-nologo		Print no greeting message.
-o | -out	<file>	Name of the zdb output binfile. (This option is only available for the verilog2zdb executable.)
-pedantic	on|off	Toggle pedantic language checking mode (in relaxed mode some errors are just warnings and some warnings are suppressed). (The default value for this option is on.)
-preserveAssign	on|off	Preserve assignments in the netlist. (The default value for this option is off.)
-prog		Print progress information. (This option is only available for the verilog2zdb executable.)
-reduceInvChain	on|off	Replace odd number of INVs in a chain by one INV and remove an even number of INVs in a chain. (The default value for this option is off.)
-removeBuffer	on|off	Remove all BUF and WIDE_BUF instances and merge the connected nets. (The default value for this option is off.)
-renameSuffix	<suffix>	Suffix for renaming duplicate cells.
-resolveDuplicates	on|off	Resolve duplicate cells. If "off" all duplicate cells are renamed. (The default value for this option is on.)
-spos	on|off	Create source code references. (The default value for this option is on.)
-suppress	<type> <pattern>	Suppress messages which match pattern. (This option is only available for the verilog2zdb executable.)

Option	Parameters	Description
-sym2zdb		Preload symbol library file(s) given with the -symlib option.
-symlib	<symlib>	Specify a symbol library file.
+systemverilogext+	<e1>+<e2>…	Define SystemVerilog file name extensions.
-tempDir	<dir>	Specify a directory for temporary files.
-time		Print CPU time consumption (requires enabled progress updates). (This option is only available for the verilog2zdb executable.)
-top	<name>	Define this module as the top module. If * is set, then all unreferenced cells are used as top.
-v	<libfile>	Read <libfile> as Verilog library file.
-validate		Validate DB before creating the zdb output file.
+verilog1995ext+	<e1>+<e2>…	Define Verilog 1995 file name extensions.
+verilog2001ext+	<e1>+<e2>…	Define Verilog 2001 file name extensions.
-wait_for_license | -waitForLicense	<sec>	Wait 'sec' seconds for a license. If the value is -1 then the started tool will not wait for the next free license. Use a value of 0 to wait forever. (The default value for this option is -1.) (This option is only available for the verilog2zdb executable.)
-y	<libdir>	Read files matching the extension given with +libext+ from the specified directory as Verilog library files.

The EDIF Parser

This document describes the function of the software unit that reads EDIF files and stores the connectivity (with parameters) to the hierarchical database ZDB for later processing.

Overview

Introduction

In addition of using the RTLvision PRO GUI to read EDIF files either the stand alone binary edif2zdb or the Tcl command zedif can be used.

The edif2zdb binary reads the given EDIF file and creates a binfile containing the connectivity of the input file:

```
edif2zdb -o binfile.zdb <OPTIONS> <FILE.EDF>
```

The zedif Tcl command reads the given EDIF file and returns the created in-memory database:

```
set db [zedif <OPTIONS> <FILE.EDF>]
```

The Tcl command is available in the Console window of the RTLvision PRO GUI .

This document describes the usage of this flow.

Options

Option	Parameters	Description
-argsFromFile	<file>	Read cmdline arguments from file.
-binlib	<file>	Open this binfile as a precompiled library.
-breakOnError | -breakOnErr	on|off	Stop on errors during parsing. (The default value for this option is off.)
-connectByName	<netnamepattern>	Connect matching net by name, i.e. don't route it.
-createBus	on|off	Create buses for ports with Verilog conform, consecutive numbering. (The default value for this option is off.)
-createHier	<sep>	Create hierarchy from flat instance names. Split the instance names at the given hierarchy separator. If the given hierarchy separator character is an empty string then the hierarchy separator character is guessed.
-debugFlag	<flag>	Enable a specific debug flag.
-dontCut		Don't cut long filenames in messages.
-forceSymlib		Overwrite existing symbols with the symbols from the given symbol library files (option -symlib).
-guessBus	<open> <close>	Guess buses based on net and port names with a bit subscript enclosed in the given 'open' and 'close' characters.
-guessInstArray	<open> <close>	Guess instance arrays based on instance names with a bit subscript enclosed in the given 'open' and 'close' characters.
-help | -h		Print a help text with a short description of each option.

Option	Parameters	Description
-hierSep	<hiersepchar>	Set the desired hierarchy separator 'hiersepchar' of your choice. Any character can be used. To be able to identify the hierarchy separator, a character that is not already used in an identifier should be used.
-ignoreCase	on|off	Case-insensitive parser. (The default value for this option is on.)
-info	None|Error|Warning|Verbose|Debug	Level of verbosity for issued messages. (The default value for this option is Error.) (This option is only available for the edif2zdb executable.)
-into	<dbname>	Compile the given files into an already existing database. (This option is only available for the zedif command.)
-logAppend		Append to logfile.
-logfile	<file>	Generate log file. (This option is only available for the edif2zdb executable.)
-noHierSep		Don't search for a hierarchy separator.
-nologo		Print no greeting message.
-o | -out	<file>	Name of the zdb output binfile. (This option is only available for the edif2zdb executable.)
-prog		Print progress information. (This option is only available for the edif2zdb executable.)
-reduceInvChain	on|off	Replace odd number of INVs in a chain by one INV and remove an even number of INVs in a chain. (The default value for this option is off.)
-removeBuffer	on|off	Remove all BUF and WIDE_BUF instances and merge the connected nets. (The default value for this option is off.)
-renameSuffix	<suffix>	Suffix for renaming duplicate cells.
-resolveDuplicates	on|off	Resolve duplicate cells. If "off" all duplicate cells are renamed. (The default value for this option is on.)
-spos	on|off	Create source code references. (The default value for this option is on.)
-suppress	<type> <pattern>	Suppress messages which match pattern. (This option is only available for the edif2zdb executable.)

Option	Parameters	Description
-sym2zdb		Preload symbol library file(s) given with the -symlib option.
-symlib	<symlib>	Specify a symbol library file.
-time		Print CPU time consumption (requires enabled progress updates). (This option is only available for the edif2zdb executable.)
-top	<name>	Define this module as the top module. If * is set, then all unreferenced cells are used as top.
-topView	<view>	View name for the -top option (-edif only).
-validate		Validate DB before creating the zdb output file.
-wait_for_license | -waitForLicense	<sec>	Wait 'sec' seconds for a license. If the value is -1 then the started tool will not wait for the next free license. Use a value of 0 to wait forever. (The default value for this option is -1.) (This option is only available for the edif2zdb executable.)

The LEF Parser

This document describes the function of the software unit that reads LEF files and stores all library elements as primitive cells in ZDB for later use.

Overview

Introduction

In addition of using the RTLvision PRO GUI to read LEF files either the stand alone binary lef2zdb or the Tcl command zlef can be used.

The lef2zdb binary reads the given LEF file and creates a binfile containing the libraries of the input file:

```
lef2zdb -o binfile.zdb <OPTIONS> <FILE.LEF>
```

The zlef Tcl command reads the given LEF file and returns the created in-memory database:

```
set db [zlef <OPTIONS> <FILE.LEF>]
```

The Tcl command is available in the Console window of the RTLvision PRO GUI .

This document describes the usage of this flow.

Options

Option	Parameters	Description
`-argsFromFile`	`<file>`	Read cmdline arguments from file.
`-binlib`	`<file>`	Open this binfile as a precompiled library.
`-breakOnError` | `-breakOnErr`	`on|off`	Stop on errors during parsing. (The default value for this option is `off`.)
`-connectByName`	`<netnamepattern>`	Connect matching net by name, i.e. don't route it.
`-debugFlag`	`<flag>`	Enable a specific debug flag.
`-dontCut`		Don't cut long filenames in messages.
`-help` | `-h`		Print a help text with a short description of each option.
`-ignoreCase`	`on|off`	Case-insensitive parser. (The default value for this option is `on`.)
`-info`	`None|Error|Warning|Verbose|Debug`	Level of verbosity for issued messages. (The default value for this option is `Error`.) (This option is only available for the `lef2zdb` executable.)
`-into`	`<dbname>`	Compile the given files into an already existing database. (This option is only available for the `zlef` command.)
`-logAppend`		Append to logfile.
`-logfile`	`<file>`	Generate log file. (This option is only available for the `lef2zdb` executable.)
`-nologo`		Print no greeting message.
`-o` | `-out`	`<file>`	Name of the zdb output binfile. (This option is only available for the `lef2zdb` executable.)
`-prog`		Print progress information. (This option is only available for the `lef2zdb` executable.)
`-reduceInvChain`	`on|off`	Replace odd number of INVs in a chain by one INV and remove an even number of INVs in a chain. (The default value for this option is `off`.)

Option	Parameters	Description
-removeBuffer	on|off	Remove all BUF and WIDE_BUF instances and merge the connected nets. (The default value for this option is off.)
-renameSuffix	<suffix>	Suffix for renaming duplicate cells.
-resolveDuplicates	on|off	Resolve duplicate cells. If "off" all duplicate cells are renamed. (The default value for this option is on.)
-scale	<scale>	Scale all coordinates by this factor. (The default value for this option is 1.00.)
-skipSupplyPorts	on|off	Skip power/ground supply ports. (The default value for this option is off.)
-spos	on|off	Create source code references. (The default value for this option is on.)
-suppress	<type> <pattern>	Suppress messages which match pattern. (This option is only available for the lef2zdb executable.)
-time		Print CPU time consumption (requires enabled progress updates). (This option is only available for the lef2zdb executable.)
-validate		Validate DB before creating the zdb output file.
-wait_for_license | -waitForLicense	<sec>	Wait 'sec' seconds for a license. If the value is -1 then the started tool will not wait for the next free license. Use a value of 0 to wait forever. (The default value for this option is -1.) (This option is only available for the lef2zdb executable.)

The DEF Parser

This document describes the function of the software unit that reads DEF files and stores the connectivity into the hierarchical database ZDB for later processing.

Overview

Introduction

In addition of using the RTLvision PRO GUI to read DEF files either the stand alone binary def2zdb or the Tcl command zdef can be used.

The def2zdb binary reads the given DEF file and creates a binfile containing the connectivity of the input file:

```
def2zdb -o binfile.zdb <OPTIONS> <FILE.DEF>
```

The zdef Tcl command reads the given DEF file and returns the created in-memory database:

```
set db [zdef <OPTIONS> <FILE.DEF>]
```

The Tcl command is available in the Console window of the RTLvision PRO GUI .

This document describes the usage of this flow.

Options

Option	Parameters	Description
-argsFromFile	<file>	Read cmdline arguments from file.
-binlib	<file>	Open this binfile as a precompiled library.
-breakOnError | -breakOnErr	on|off	Stop on errors during parsing. (The default value for this option is off.)
-connectByName	<netnamepattern>	Connect matching net by name, i.e. don't route it.
-createHier	<sep>	Create hierarchy from flat instance names. Split the instance names at the given hierarchy separator. If the given hierarchy separator character is an empty string then the hierarchy separator character is guessed.
-createPreplace	on|off	Use the placement information from a DEF file to create preplace strings for the schematic view. (The default value for this option is off.)
-debugFlag	<flag>	Enable a specific debug flag.
-dontCut		Don't cut long filenames in messages.
-forceSymlib		Overwrite existing symbols with the symbols from the given symbol library files (option -symlib).
-help | -h		Print a help text with a short description of each option.
-hierSep	<hiersepchar>	Set the desired hierarchy separator 'hiersepchar' of your choice. Any character can be used. To be able to identify the hierarchy separator, a character that is not already used in an identifier should be used.
-ignoreCase	on|off	Case-insensitive parser. (The default value for this option is on.)

Option	Parameters	Description
-info	None|Error|Warning|Verbose|Debug	Level of verbosity for issued messages. (The default value for this option is Error.) (This option is only available for the def2zdb executable.)
-into	<dbname>	Compile the given files into an already existing database. (This option is only available for the zdef command.)
-logAppend		Append to logfile.
-logfile	<file>	Generate log file. (This option is only available for the def2zdb executable.)
-noHierSep		Don't search for a hierarchy separator.
-nologo		Print no greeting message.
-o | -out	<file>	Name of the zdb output binfile. (This option is only available for the def2zdb executable.)
-prog		Print progress information. (This option is only available for the def2zdb executable.)
-reduceInvChain	on|off	Replace odd number of INVs in a chain by one INV and remove an even number of INVs in a chain. (The default value for this option is off.)
-removeBuffer	on|off	Remove all BUF and WIDE_BUF instances and merge the connected nets. (The default value for this option is off.)
-renameSuffix	<suffix>	Suffix for renaming duplicate cells.
-resolveDuplicates	on|off	Resolve duplicate cells. If "off" all duplicate cells are renamed. (The default value for this option is on.)
-scale	<scale>	Scale all coordinates by this factor. (The default value for this option is 1.00.)
-skipSupplyPorts	on|off	Skip power/ground supply ports. (The default value for this option is off.)
-spos	on|off	Create source code references. (The default value for this option is on.)
-storePlacement	on|off	Store placement information from DEF file as attributes. (The default value for this option is off.)

Option	Parameters	Description
-suppress	<type> <pattern>	Suppress messages which match pattern. (This option is only available for the def2zdb executable.)
-sym2zdb		Preload symbol library file(s) given with the -symlib option.
-symlib	<symlib>	Specify a symbol library file.
-time		Print CPU time consumption (requires enabled progress updates). (This option is only available for the def2zdb executable.)
-top	<name>	Define this module as the top module. If * is set, then all unreferenced cells are used as top.
-validate		Validate DB before creating the zdb output file.
-wait_for_license | -waitForLicense	<sec>	Wait 'sec' seconds for a license. If the value is -1 then the started tool will not wait for the next free license. Use a value of 0 to wait forever. (The default value for this option is -1.) (This option is only available for the def2zdb executable.)

The Liberty Parser

This document describes the function of the software unit that reads Liberty files and stores all library elements as primitive cells in ZDB for later use.

Overview

Introduction

In addition of using the RTLvision PRO GUI to read Liberty files either the stand alone binary liberty2zdb or the Tcl command zliberty can be used.

The liberty2zdb binary reads the given Liberty file and creates a binfile containing the libraries of the input file:

```
liberty2zdb -o binfile.zdb <OPTIONS> <FILE.LIB>
```

The zliberty Tcl command reads the given Liberty file and returns the created in-memory database:

```
set db [zliberty <OPTIONS> <FILE.LIB>]
```

The Tcl command is available in the Console window of the RTLvision PRO GUI .

This document describes the usage of this flow.

Options

Option	Parameters	Description
-argsFromFile	<file>	Read cmdline arguments from file.
-boolSym	on|off	Store BOOL symbol instead of the analyzed shape. (The default value for this option is off.)
-breakOnError | -breakOnErr	on|off	Stop on errors during parsing. (The default value for this option is off.)
-createNetlist	on|off	Create a netlist for the equation representing the function. (The default value for this option is off.)
-debugFlag	<flag>	Enable a specific debug flag.
-dontCut		Don't cut long filenames in messages.
-help | -h		Print a help text with a short description of each option.
-info	None|Error|Warning|Verbose|Debug	Level of verbosity for issued messages. (The default value for this option is Error.) (This option is only available for the liberty2zdb executable.)
-into	<dbname>	Compile the given files into an already existing database. (This option is only available for the zliberty command.)
-libertyDetails		If not specified then all information not required to create a library cell are skipped. The purpose of this option is to get syntax error reports in irrelevant sections of the read Liberty file or to do semantic checks.
-logAppend		Append to logfile.
-logfile	<file>	Generate log file. (This option is only available for the liberty2zdb executable.)
-nologo		Print no greeting message.

Option	Parameters	Description
-o | -out	<file>	Name of the zdb output binfile. (This option is only available for the liberty2zdb executable.)
-oSymlib	<file>	Name of symlib output file.
-pedantic	on|off	Toggle pedantic language checking mode (in relaxed mode some errors are just warnings and some warnings are suppressed). (The default value for this option is on.)
-prog		Print progress information. (This option is only available for the liberty2zdb executable.)
-renameSuffix	<suffix>	Suffix for renaming duplicate cells.
-renderEquation	on|off	Render symbols representing the equation netlist. (The default value for this option is off.)
-resolveDuplicates	on|off	Resolve duplicate cells. If "off" all duplicate cells are renamed. (The default value for this option is on.)
-semanticCheck		Enable semantic checks. Semantic issues will produce warnings or, if -pedantic is enabled, error messages.
-setPrimFunc	on|off	Additionally to the @symbol attribute also set the primitive function at the created cell. (The default value for this option is off.)
-skipMuxDetection	on|off	Skip detection of MUX cells and do not assign a symbol shape. (The default value for this option is off.)
-skipSupplyPorts	on|off	Skip power/ground supply ports. (The default value for this option is off.)
-spos	on|off	Create source code references. (The default value for this option is on.)
-storeGroup	<group name>	Name of a Liberty group to store as a database attribute.
-suppress	<type> <pattern>	Suppress messages which match pattern. (This option is only available for the liberty2zdb executable.)
-time		Print CPU time consumption (requires enabled progress updates). (This option is only available for the liberty2zdb executable.)
-validate		Validate DB before creating the zdb output file.

Option	Parameters	Description
-wait_for_license | -waitForLicense	<sec>	Wait 'sec' seconds for a license. If the value is -1 then the started tool will not wait for the next free license. Use a value of 0 to wait forever. (The default value for this option is -1.) (This option is only available for the liberty2zdb executable.)

The SDF API

Introduction

This document describes the "SDF API". It is an extension to the Database API and bases on the Tcl command sdf $db.

Overview

The sdf command links an SDF file to an existing database or extracts timings for a given OID.

- sdf $db link ?-portdirupdate? ?-topInstName name? $filename
- sdf $db getTiming $signal|$inst $filter1 $filter2

Link an SDF File to a Database

```
sdf $db link ?-portdirupdate? ?-topInstName name? $filename
```

Links a given SDF file to the matching top module of the given database. As the SDF timings are not copied to the database, the SDF file must be present when timings are extracted with the sdf $db getTiming command.

Parameters

- $db - the database containing the module to connect with.
- $filename - the filename of an SDF file to link with the database.
- -portdirupdate - optional switch to reconstruct port directions of instances in the database from an SDF file. As proper port directions are needed to extract valid timings, this is strongly recommended for designs with unknown port directions (e.g. if no cell-library is provided).
- -topInstName - optional argument to set the name of the top instance.

Results

If executed without error, additional spos items described in the SDF file are created in the database. They can be extracted by $db spos commands or are used by the sdf $db getTiming command to extract timing information.

Access SDF Timing Info from Database

```
sdf $db getTiming $OID $filter1 $filter2
```

Gets SDF timing information for a given OID (must be either a signal or an instance).

Parameters

- `$db` - the Database containing the module.
- `$OID` - the OID the timing shall be extracted for; must be either a signal OID or an instance OID.
- `$filter1` - one of `Delay`, `PulseR` or `PulseX`. Depending of the option provided, delay-, pulse-rejection-limit- or X-filter-limit-values are extracted.
- `$filter2` - one of `Min`, `Typ` or `Max`, determining the process factor for which the timings shall be extracted.

Results if $OID is a Signal

If executed without error, a list of triplets (source OID \+ target OID + timing table) is created:

```
$sig_tim := $src_oid_1 $trg_oid_1 $time_table_1 \
            $src_oid_2 $trg_oid_2 $time_table_2 \
            [...] \
            $src_oid_n $trg_oid_n $time_table_n
```

- `$src_oid_x` - OID of source for path "x"
- `$trg_oid_x` - OID of target for path "x"
- `$time_table_x` - timing table for path "x"

Timing table `$time_table` is a Tcl list of 12 tuples (value + type):

```
$time_table := $value_1 $type_1 \
               $value_2 $type_2 \
               [..] \
               $value_12 $type_12
```

- `$value_x` - delay value of transition "x"; or `-` if no value is given.
- `$type_x` - type of value. `t` if the value is explicitly given in the SDF file, `c` if it's a calculated value not explicitly given in the SDF file.

Each tuple represents the timing of a transition as described in IEEE 1497-2001, page 30:

Tuple #	Transition
1	`0 → 1`

Tuple #	Transition
2	`1 → 0`
3	`0 → Z`
4	`Z → 1`
5	`1 → Z`
6	`Z → 0`
7	`0 → X`
8	`X → 1`
9	`1 → X`
10	`X → 0`
11	`X → Z`
12	`Z → X`

Results if $OID is an Instance

If executed without error, a list of two lists is created.

```
$inst_tim := $port_tim $io_tim
```

- `$port_tim` - Tcl list of port delays
- `$io_tim` - Tcl list of IO path delays

A port delay list `$port_tim` is a Tcl list of tuples (pin OID + timing table):

```
$port_tim := $pin_oid_1 $time_table_1 \
             $pin_oid_2 $time_table_2 \
             [...] \
             $pin_oid_n $time_table_n
```

- `$pin_oid_x` - pin OID of actual instance.
- `$time_table_x` - timing table as described in the previous results section.

An IO path delay list is a Tcl list of triplets (source-pin OID + target-pin OID + conditional timing list):

```
$io_tim := $src_oid_1 $trg_oid_1 $cond_tim_1 \
           $src_oid_2 $trg_oid_2 $cond_tim_2 \
           [...] \
           $src_oid_n $trg_oid_n $cond_tim_n
```

- `$src_oid_x` - OID of source for path "x"

- `$trg_oid_x` - OID of target for path "x"
- `$cond_tim_x` - conditional timing list

A conditional timing list `$cond_tim_x` is a Tcl list of tuples (timing table + condition):

```
$cond_tim := $time_table_1 $condition_1 \
             $time_table_2 $condition_2 \
             [...] \
             $time_table_n $condition_n
```

- `$time_table_x` - timing table as described in the previous results section.
- `$condition_x` - Tcl string describing the condition under which the corresponding `$time_table_x` is valid.

The VCD Reader

This document describes the function of the software unit that reads VCD files and stores the value changes in a compressed and accelerated format to the wave database WDB for later processing.

Overview

- Introduction
- Options

Introduction

In addition of using the RTLvision PRO GUI to read VCD files either the stand alone binary `vcdcompile` or the Tcl command `wdb read` can be used.

The `vcdcompile` binary reads the given VCD file and creates a WDB binfile containing the value changes of the input file:

```
vcdcompile -o binfile.zdb <OPTIONS> <FILE.VCD>
```

The `wdb read` Tcl command reads the given VCD file and returns the created in-memory WDB database:

```
set wdb [wdb read <OPTIONS> <FILE.VCD>]
```

The Tcl command is available in the Console window of the RTLvision PRO GUI .

This document describes the usage of this flow.

Options

Option	Parameters	Description
-argsFromFile	<file>	Read cmdline arguments from file.
-breakOnError | -breakOnErr	on|off	Stop on errors during parsing. (The default value for this option is off.)
-debugFlag	<flag>	Enable a specific debug flag.
-dontCut		Don't cut long filenames in messages.
-from	<time>	Ignore value changes before time.
-help | -h		Print a help text with a short description of each option.
-info	None|Error|Warning|Verbose|Debug	Level of verbosity for issued messages. (The default value for this option is Error.)
-logAppend		Append to logfile.
-logfile	<file>	Generate log file.
-noCollapse		Do not collapse multiple value changes at same t.
-nogcd		Compile without gcd for time deltas.
-nologo		Print no greeting message.
-noSkipDupl		Do not skip identical value changes.
-nozip		Compile without zip.
-o	<file>	Name of binary wave output file.
-prog		Print progress information.
-skipLeadX		Skip lead 'X' values, if all signal values are equal.
-suppress	<type> <pattern>	Suppress messages which match pattern.
-time		Print CPU time consumption (requires enabled progress updates).
-to	<time>	Ignore value changes after time.
-wait_for_license | -waitForLicense	<sec>	Wait 'sec' seconds for a license. If the value is -1 then the started tool will not wait for the next free license. Use a value of 0 to wait forever. (The default value for this option is -1.)

The Symbol to ZDB Converter

This document describes the function of the software unit that reads a Symlib file and stores the symbol shapes found in the symbol library in a ZDB binfile.

Overview

Introduction

In addition of using the RTLvision PRO GUI to read Symbol library files the stand alone binary `sym2zdb` can be used.

The `sym2zdb` binary reads the given Symbol library file and creates a ZDB binfile containing the symbol shapes found in the input file:

```
sym2zdb -o binfile.zdb <OPTIONS> <FILE.SYM>
```

This document describes the usage of this flow.

Options

Option	Parameters	Description
`-argsFromFile`	`<file>`	Read cmdline arguments from file.
`-binlib`	`<file>`	Open this binfile as a precompiled library.
`-breakOnError` | `-breakOnErr`	`on|off`	Stop on errors during parsing. (The default value for this option is `off`.)
`-debugFlag`	`<flag>`	Enable a specific debug flag.
`-dontCut`		Don't cut long filenames in messages.
`-help` | `-h`		Print a help text with a short description of each option.
`-info`	`None|Error|Warning|Verbose|Debug`	Level of verbosity for issued messages. (The default value for this option is `Error`.)
`-logAppend`		Append to logfile.
`-logfile`	`<file>`	Generate log file.
`-nologo`		Print no greeting message.
`-o` | `-out`	`<file>`	Name of the zdb output binfile.
`-prog`		Print progress information.
`-renameSuffix`	`<suffix>`	Suffix for renaming duplicate cells.
`-resolveDuplicates`	`on|off`	Resolve duplicate cells. If "off" all duplicate cells are renamed. (The default value for this option is `on`.)
`-suppress`	`<type> <pattern>`	Suppress messages which match pattern.

Option	Parameters	Description
-time		Print CPU time consumption (requires enabled progress updates).
-validate		Validate DB before creating the zdb output file.
-wait_for_license | -waitForLicense	<sec>	Wait 'sec' seconds for a license. If the value is -1 then the started tool will not wait for the next free license. Use a value of 0 to wait forever. (The default value for this option is -1.)

The ZDB to Symbol Converter

This document describes the function of the software unit that reads a ZDB binfile and stores the symbol shapes found in the binfile as a Symlib library file.

Overview

Introduction

In addition of using the RTLvision PRO GUI to read a ZDB binfile the stand alone binary zdb2sym can be used.

The zdb2sym binary reads the given ZDB binfile and creates a Symbol library containing the symbol shapes found in the input file:

```
zdb2sym -o file.sym <OPTIONS> <BINFILE.ZDB>
```

This document describes the usage of this flow.

Options

Option	Parameters	Description
-argsFromFile	<file>	Read cmdline arguments from file.
-breakOnError | -breakOnErr	on|off	Stop on errors during parsing. (The default value for this option is off.)
-debugFlag	<flag>	Enable a specific debug flag.
-dontCut		Don't cut long filenames in messages.
-help | -h		Print a help text with a short description of each option.

Option	Parameters	Description
-info	None|Error|Warning|Verbose|Debug	Level of verbosity for issued messages. (The default value for this option is Error.)
-logAppend		Append to logfile.
-logfile	<file>	Generate log file.
-nologo		Print no greeting message.
-o	<file>	Name of symlib output file.
-prog		Print progress information.
-renameSuffix	<suffix>	Suffix for renaming duplicate cells.
-resolveDuplicates	on|off	Resolve duplicate cells. If "off" all duplicate cells are renamed. (The default value for this option is on.)
-suppress	<type> <pattern>	Suppress messages which match pattern.
-time		Print CPU time consumption (requires enabled progress updates).
-wait_for_license | -waitForLicense	<sec>	Wait 'sec' seconds for a license. If the value is -1 then the started tool will not wait for the next free license. Use a value of 0 to wait forever. (The default value for this option is -1.)

The C-API

The Utilities Header Files

Define all Basic Data Types

Boolean Data Type

ce_Bool

Definition for Boolean true and false.

```
typedef enum {
    ce_False = 0,
    ce_True  = 1
} ce_Bool;
```

Signed Integer Data Types

ce_Int8

8 Bit signed integer.

```
typedef signed char ce_Int8;
#define CE_INT8_MIN   SCHAR_MIN
#define CE_INT8_MAX   SCHAR_MAX
```

ce_Int16

16 Bit signed integer.

```
typedef signed short ce_Int16;
#define CE_INT16_MIN   SHRT_MIN
#define CE_INT16_MAX   SHRT_MAX
```

ce_Int32

32 Bit signed integer.

```
typedef signed int ce_Int32;
#define CE_INT32_MIN INT_MIN
#define CE_INT32_MAX INT_MAX
```

ce_Int64

64 Bit signed integer.

```
#if defined(WINDOWS)
typedef signed __int64     ce_Int64;
#define CE_INT64_MIN        _I64_MIN
#define CE_INT64_MAX        _I64_MAX
#else
typedef signed long long   ce_Int64;
#define CE_INT64_MIN        (-CE_INT64_MAX-1LL)
#define CE_INT64_MAX        (9223372036854775807LL)
#endif
```

Unsigned Integer Data Types

ce_UInt8

8 Bit unsigned integer.

```
typedef unsigned char ce_UInt8;
#define CE_UINT8_MAX    UCHAR_MAX
```

ce_UInt16

16 Bit unsigned integer.

```
typedef unsigned short ce_UInt16;
#define CE_UINT16_MAX    USHRT_MAX
```

ce_UInt32

32 Bit unsigned integer.

```
typedef unsigned int ce_UInt32;
#define CE_UINT32_MAX  UINT_MAX
```

ce_UInt64

64 Bit unsigned integer.

```
#if defined(WINDOWS)
typedef unsigned __int64   ce_UInt64;
#define CE_UINT64_MAX      _UI64_MAX
#else
typedef unsigned long long ce_UInt64;
#define CE_UINT64_MAX      (18446744073709551615ULL)
#endif
```

Integer Type for Pointer Precision

ce_IntPtr

Signed integer in the size of a pointer; use when casting a pointer to a long to perform pointer arithmetic.

```
#if defined(WINDOWS)
typedef   signed __int64 ce_IntPtr;
#else
typedef signed long      ce_IntPtr;
#endif
```

ce_UIntPtr

Unsigned integer in the size of a pointer; use when casting a pointer to a long to perform pointer arithmetic.

```
#if defined(WINDOWS)
typedef unsigned __int64 ce_UIntPtr;
#else
typedef unsigned long    ce_UIntPtr;
#endif
```

Integer Type for File Size, Position and Offset

ce_Filepos

File position.

```
typedef ce_Int64 ce_Filepos;
#define CE_FILEPOS_MAX   CE_INT64_MAX
```

Integer Type for Platform Independent Long Precision

ce_Long

Signed integer in the size of a pointer.

```
#if defined(WINDOWS)
typedef signed __int64 ce_Long;
#define CE_LONG_MAX    _I64_MAX
#else
typedef signed long    ce_Long;
#define CE_LONG_MAX    LONG_MAX
#endif
```

ce_ULong

Unsigned integer in the size of a pointer.

```
#if defined(WINDOWS)
typedef unsigned __int64 ce_ULong;
#define CE_ULONG_MAX     _UI64_MAX
#else
typedef unsigned long    ce_ULong;
#define CE_ULONG_MAX     ULONG_MAX
#endif
```

Integer Type for Time

ce_Time

Unsigned 64 bit integer to store time information.

```
typedef ce_UInt64 ce_Time;
```

ce_Date

Signed integer to store date information.

```

```

Define Custom Assertions

Error Message Handling

File System Functions

Collection of functions to interact with the file system.

File Functions

Many ce_File* functions get an "filename" argument that refers to a file or to a directory. If not stated different below, it can be either an absolute path or a relative path (relative to the current

working directory). The "dirname" argument refers to a directory (can be either an absolute path or a relative path).

ce_Filesize

Returns the size of a file (in # of bytes).

```
ce_Filepos ce_Filesize(const char* filename);
```

ce_FileModificationTime

Returns the modification time stamp of a file or directory.

```
ce_Date ce_FileModificationTime(const char* filename);
```

ce_FileExist

Return true if given file or directory exists and false if it does not exist.

```
ce_Bool ce_FileExist(const char* filename);
```

ce_FileIsDirectory

Returns true if filename refers to a directory or false if not.

```
ce_Bool ce_FileIsDirectory(const char* filename);
```

ce_FileIsSymbolicLink

Returns true if filename refers to a symbolic link or false if not.

```
ce_Bool ce_FileIsSymbolicLink(const char* filename);
```

ce_FileReadSymbolicLink

Returns the target of a symbolic link.

```
int ce_FileReadSymbolicLink(
    const char* filename,
    char*       buf,
    unsigned    maxlen
);
```

ce_FilePathToExe

Return the absolute path to the running executable.

```
int ce_FilePathToExe(
    const char* filename,
    char*       buf,
    unsigned    maxlen
);
```

ce_FileIsWritable

Return true if given dir/file is writable.

```
ce_Bool ce_FileIsWritable(const char* filename);
```

ce_FileIsReadable

Return true if given dir/file is readable.

```
ce_Bool ce_FileIsReadable(const char* filename);
```

ce_FileIsExecutable

Return true if the given file is executable.

```
ce_Bool ce_FileIsExecutable(const char* filename);
```

ce_FileRename

Renames a file. Returns 0 on success or 1 on error. This function is limited to file names without any directory path prefix (neither relative nor absolute).

```
int ce_FileRename(
    const char* oldName,
    const char* newName
);
```

ce_FileDelete

Delete a file. Returns 0 on success or 1 on error. This function is limited to file names without any directory path prefix (neither relative nor absolute).

```
int ce_FileDelete(const char* filename);
```

ce_FileCaseName

Converts the given filename (by changing upper/lower case of the file name characters) to fit an existing filename (in the file system). If no existing file name can be found, then 0 is returned and filename is not modified. On success, 1 is returned.

```
int ce_FileCaseName(char* filename);
```

ce_FileHasTTY

Return true if there is a TTY

```
ce_Bool ce_FileHasTTY(void);
```

ce_FileOpenDir

Open a directory - like UNIX opendir(). Opening "/.." maps to "/" on some UNIX systems.

```
void* ce_FileOpenDir(const char* dirname);
```

ce_FileReadDir

Read the next entry of the directory - like UNIX readdir().

```
const char* ce_FileReadDir(void* dir);
```

ce_FileCloseDir

Close the directory - like UNIX closedir(). Returning 0 on success or -1 on failure (the global errno is set to indicate the error).

```
int ce_FileCloseDir(void* dir);
```

ce_FileCopy

Copy infile to outfile.

```
int ce_FileCopy(
    const char* infile,
    const char* outfile
);
```

ce_FileTempDirectory

Return a good temp dir.

```
const char* ce_FileTempDirectory(void);
```

ce_FileTempFilename

Return unique filename in temp dir.

```
const char* ce_FileTempFilename(
    const char* prefix,
    const char* tempDir,
    char*       buf,
    int         maxlen
);
```

ce_FileDirname

Find directory of given filename.

```
int ce_FileDirname(
    const char* filename,
    char*       result,
    unsigned    maxlen
);
```

ce_FileTail

Return all of the characters in the last filesystem component of the given filename. If the filename contains no separators, then the filename is returned.

```
int ce_FileTail(
    const char* filename,
    char*       result,
    unsigned    maxlen
);
```

ce_FileNameType

Return the type of the file: 0: absolute 1: relative -1: empty file name 2: volumerelative (on Windows only) 3: novolumeabsolute (on Windows only)

```
int ce_FileNameType(const char* filename);
```

ce_FileNativeName

Convert / to \\ on Windows.

```
void ce_FileNativeName(char* filename);
```

ce_FileUniqueName

Makes the given filename unique. The returned string must be freed after usage.

```
char* ce_FileUniqueName(const char* filename);
```

ce_FileMkdir

Create given directory. If successful returns true. If failed or the if the directory already exists return false.

```
ce_Bool ce_FileMkdir(
    const char* directory,
    ce_Bool     recursive
);
```

ce_FileConvertName

Converts a given filename depending on mode. The given filename is converted to absolute or relative (more on Windows) depending on the "mode" argument. The conversion is relative to the given directory "dir" (dir must be absolute) or if dir == NULL, then relative to the current working directory. The converted path is copied into result buffer with maxlen space. Returns the mode of given filename or -1 for error. The following modes are defined: 0 : absolute Unix: /dir/file Win32: c:\dir\file.txt, \\host\service\file.txt 1 : relative Unix: ./file, file.txt Win32: .\file, file.txt dir\file.txt 2 : volumerelative Win32: c:.\file, c:file.txt c:dir\file.txt 3 : novolumeabsolute Win32: /dir/file 4 : relative(1) or absolute(0) - whatever is shorter

```
int ce_FileConvertName(
    const char* filename,
    int         mode,
    const char* dir,
    char*       result,
    unsigned    maxlen
);
```

ce_FileConvertName2

Converts a given "filename" into relative-to-todir. Converted path is copied into "result" buffer with maxlen space. Returns the mode of given "filename" or -1 for error. The given "filename" may be relative to "fromdir". If the given "fromdir" and "todir" are not absolute paths, then they are first

converted to absolute by the current working directory. If the given "filename" cannot converted into relative-to-todir, then the absolute path is returned. On Windows, the details are: filename result 0:absolute 0:absolute 1:relative 1:relative-to-todir or 0:absolute 2:volumerelative 1:relative-to-todir or 0:absolute 3:novolumeabsolute 3:novolumeabsolute or 0:absolute

The ambiguous case 2:volumerelative will call getdcwd() to get Window's drive-specific directory if the drive character in the given filename does not match to the drive character in todir. Examples:

filename fromdir todir result bla C:\path C:\path\to ..\bla bla C:\path\to C:\path to\bla bla C:\path F:\path\to C:\path\bla bla C:\path\to F:\path C:\path\to\bla F:\path\bla ??????? ??????? F:\path\bla F:bla F:\path\to C:\path F:\path\to\bla F:bla F:\path\to F:\path\any ..\to\bla \ok\bla F:\path\to C:\path F:\ok\bla \ok\bla F:\path\to F:\path\any \ok\bla

More examples for the cases if fromdir or todir are not absolute:

filename fromdir todir result bla C:/path C:/path bla bla C:/path/ok C:/path ok\bla bla . . bla bla .. . ..\bla bla . ok/ok ..\..\bla

If todir==NULL, then "." (the current working directory) is assumed, and in addition, the returned filename is absolute if that is shorter than the relative version.

```
int ce_FileConvertName2(
    const char* filename,
    const char* fromdir,
    const char* todir,
    char*       result,
    unsigned    maxlen
);
```

ce_FileHomeDir

Get home dir of given user, use NULL for current user. Returns true if ok, false on Error (error message is in dest).

```
int ce_FileHomeDir(
    const char* user,
    char*       dest,
    unsigned    maxLen
);
```

ce_FileConfigDir

Get the configuration directory for current user. Returns -1 on Error (error message is in dest).

```
int ce_FileConfigDir(
    ce_Bool  legacy,
    char*    dest,
    unsigned maxLen
);
```

ce_FilePathCompare

Compare the two given paths. On Windows '\\' and '/' are equal, also ignore case.

```
int ce_FilePathCompare(
    const char* path1,
    const char* path2
);
```

ce_FileGetCwd

Get the current working directory.

```
char* ce_FileGetCwd(
    char*    result,
    unsigned maxLen
);
```

ce_SanitizeFilename

Return a sanitized version of the given file name, e.g. a new string without any special characters, following the OS rules for file naming. The returned must be freed by the caller.

```
char* ce_FileSanitizeName(const char* filename);
```

Operating System Calls

Reimplementation of the most important clib functions. == Operating System Calls to Allocate Memory

Allocate and Free Memory

Define a Read/Write Interface

Hash Table Template

The Progress Bar Header File

Manage Progress Bar and Interrupt

The zdb Header Files

The zdb Database Core Functions and Objects

Version of the Binfile Format

Incompatible changes are only permitted for major releases.

```
#define BINFILE_VERSION_MAJOR 16  /* Increment this for in-compatible changes */
#define BINFILE_VERSION_MINOR  0  /* Increment this for    compatible changes */
```

Database Hash Table Definition

Hash Table Key Types

```
typedef enum HashKeyType {
    HASHKEY_STRING,
    HASHKEY_ICASE,
    HASHKEY_PTR,
    HASHKEY_LENGTH
} HashKeyType;
```

Hash Table Struct Definition

```
typedef struct HashTable {
    struct HashTableEntry**  bucketList;
    struct HashTableMemory*  memoryBuffer;
    struct HashTableEntry*   freeList;
    int                      nBuckets;
    int                      count;
    int                      valueSize;
    HashKeyType              ktype;
} HashTable;
```

Hash Table Iterator Struct Definition

```
typedef struct HashIter {
    struct HashTable*      table;
    struct HashTableEntry* cur;
    int                    slot;
} HashIter;
```

Hierarchy Separator

Default Hierarchy Separators

The visible hierarchy separator can be any character. The internal hierarchy separator needs to be a character that is not used as an object name in the entire database. The default value is the non-printable character GS (group separator).

```
#define VISIBLE_HIERARCHY_SEPARATOR     '.'
#define INTERNAL_HIERARCHY_SEPARATOR    0x1D
```

Database Object Flags

Port Flags

```
typedef unsigned short PortFlags;
#define PortFUnknown    0x0000  /* Port direction: Unknown                    */
#define PortFOut        0x0001  /* Port direction: Output                     */
#define PortFIn         0x0002  /* Port direction: Input                      */
#define PortFInOut      0x0003  /* Port direction: Bidir                      */
#define PortFTarget     0x0004  /* General purpose flag, use for Cone Extract */
#define PortFVisit      0x0008  /* Visit flag for processing                  */
#define PortFRed        0x0010  /* General purpose flag                       */
#define PortFGreen      0x0020  /* General purpose flag                       */
#define PortFBlue       0x0040  /* General purpose flag                       */
#define PortFYellow     0x0080  /* General purpose flag                       */

#define PortFNeg        0x0100  /* Primitive pin function: port is inverted   */
#define PortFClk        0x0200  /* Primitive pin function: port is clocked    */
#define PortFMultiGate  0x0400  /* Port is a multi gate                       */
#define PortFHide       0x0800  /* Primitive/module port is hidden in zdi.c   */

#define PortFTop        0x1000  /* Display port at top                        */
#define PortFBot        0x2000  /* Display port at bottom                     */
#define PortFLeft       0x4000  /* Force port to left side                    */
#define PortFRight      0x8000  /* Force port to right side                   */
#define PortFTBLR      (PortFTop|PortFBot|PortFLeft|PortFRight)

#define PortFAll        0xffff  /* all bits set as error flags */

typedef unsigned short PortFlagsInternal;
/* Port has different power           */
#define PortFInternalMixed          0x0001
/* Port has guessed power             */
#define PortFInternalGuessed        0x0002
/* Port has IO set from -node         */
#define PortFInternalIOSet          0x0004
/* Port has weak (e.g. diode)         */
#define PortFInternalWeak           0x0008
/* Port has Power                     */
#define PortFInternalPower          0x0010
/* Port has Ground                    */
#define PortFInternalGround         0x0020
/* Port has NegPower                  */
#define PortFInternalNegPower       0x0040
/* Port is ignored in cone search     */
#define PortFInternalConeIgn        0x0080
/* Port direction is define in a library */
#define PortFInternalFixedDirection  0x0100

#define PortFInternalPG       (PortFInternalPower   \
                              |PortFInternalGround  \
                              |PortFInternalNegPower)
```

Cell Flags

```
typedef unsigned short CellFlags;
#define CellFModule     0x0001  /* This is a Module                          */
#define CellFTarget     0x0002  /* general purpose flag, use for Cone Extract */
#define CellFZombie     0x0004  /* This object is to be deleted              */
#define CellFVisit      0x0008  /* Visit flag for processing                 */
#define CellFVisit2     0x0010  /* Visit flag for processing                 */
#define CellFVisit3     0x0020  /* Visit flag for processing                 */
#define CellFUndefined  0x0040  /* Is a blackbox instantiation               */
#define CellFWeakFlow   0x0080  /* Set for Weakflow devices                  */
#define CellFFeedthru   0x0100  /* Nlview VdiInstFFeedthru                   */
#define CellFLibCell    0x0200  /* Cell is a Library Cell                    */
#define CellFClk        0x0400  /* Cell has a Clock-Port                     */
#define CellFModuleMem  0x0800  /* Remember original CellFModule flag        */

#define CellFRed        0x1000  /* general purpose flag                      */
#define CellFGreen      0x2000  /* general purpose flag                      */
#define CellFBlue       0x4000  /* general purpose flag                      */
#define CellFYellow     0x8000  /* general purpose flag                      */

typedef unsigned short CellFlagsInternal;
/* `celldefine                                      */
#define CellFInternalCelldefine                     0x0001
/*The Cell is preloaded                             */
#define CellFInternalPreloaded                      0x0002
/* Visit flag for processing                        */
#define CellFInternalVisit                          0x0004
/* UserdefinedArcs are checked                      */
#define CellFInternalUserDefArcsChecked             0x0008
/* Auto generated cell                              */
#define CellFInternalAutoGen                        0x0010
/* Generated in Lpe                                 */
#define CellFInternalLpeParasitic                   0x0020
/* Generated by spice                               */
#define CellFInternalCreatedBySpice                 0x0040
/* Generated by symlib                              */
#define CellFInternalCreatedBySymlib                0x0080
/* Generated by zverilog                            */
#define CellFInternalCreatedByZverilog              0x0100
/* If cell is undefined and has generic port names */
#define CellFInternalUndefinedWithNamedPorts        0x0200
/* If cell is ignored in zwrite */
#define CellFInternalIgnoreInWrite                  0x0400
/* Generated by Liberty                             */
#define CellFInternalCreatedByLiberty               0x0800
/* Exclude cell from cone extraction                */
#define CellFInternalConeExtractExclude             0x8000
```

Module Flags

```
typedef unsigned short ModuleFlags;
#define ModuleFTop                  0x0001 /* Top Module (topList)             */
#define ModuleFParasitic            0x0002 /* Parasitic Module (parasiticTable)*/
#define ModuleFParasiticTop         0x0004 /* Parasitic Top Module.            */
#define ModuleFQuick                0x0008 /* Quick Parser Mode.               */
#define ModuleFAutoGen              0x0010 /* Auto-generated module            */
#define ModuleFOperpp               0x0020 /* Created by operpp                */
#define ModuleFOperGate             0x0040 /* Created by recognize gates       */
#define ModuleFLeafCell             0x0080 /* Treated module as a primitive    */
#define ModuleFParasiticHier        0x0100 /* Para-Module for hierarchical net */
#define ModuleFParasiticParent      0x0200 /* Para-Module for parent hier net  */
#define ModuleFParasiticNeedRename  0x0400 /* Parasitic Module need rename     */
#define ModuleFParasiticProcess     0x0800 /* Parasitic Module need processing */
#define ModuleFParasiticComplete    0x1000 /* Para Module is filled/processed  */
#define ModuleFGroupPorts           0x2000 /* @group attribute set at ports    */
#define ModuleFInvisible            0x4000 /* Hide the module in the Tcl API   */
#define ModuleFPreplace             0x8000 /* Module instances have @X/@y      */

typedef unsigned short ModuleFlagsInternal;
#define ModuleFInternalTopCandidate 0x0001 /* Top Candidate                    */
#define ModuleFInternalGuessedTop   0x0002 /* Guessed Top Module               */
#define ModuleFInternalIgnoreContentInWrite 0x0004 /* Ignore Content in zWrite*/
#define ModuleFInternalContentNotPopulated 0x0008 /* Contents is not populated*/
#define ModuleFInternalFlatNotPopulated    0x0010 /* Contents is not populated*/
```

Instance Flags

```
typedef unsigned short InstFlags;
#define InstFAutoGen     0x0001    /* Is an auto generated instance          */
#define InstFTarget      0x0002    /* Cone Extraction target                 */
#define InstFZombie      0x0004    /* This object is to be deleted           */
#define InstFVisit       0x0008    /* Visit flag for processing              */
#define InstFCouplLocal  0x0010    /* Coupling local                         */
#define InstFCouplExtern 0x0020    /* Coupling external                      */
#define InstFTreeGroup   0x0040    /* Flag to group instances in tree        */
#define InstFWeakFlow    0x0080    /* Set for Weakflow devices               */

#define InstFRed         0x1000    /* general purpose flag */
#define InstFGreen       0x2000    /* general purpose flag */
#define InstFBlue        0x4000    /* general purpose flag */
#define InstFYellow      0x8000    /* general purpose flag */

typedef unsigned short InstFlagsInternal;
/* Set if attributes were modified        */
#define InstFInternalAttrMod                0x01
/* Internal general purpose flag          */
#define InstFInternalVisit                  0x02
/* Ignore instance in zdi (ictrl).        */
#define InstFInternalIgnore                 0x04
/* Set the unfold flag is zdi.            */
#define InstFInternalUnfold                 0x08
/* Internal general purpose flag          */
#define InstFInternalVisit2                 0x10
/* Internal pfilter shorten flag          */
#define InstFInternalPFilterShorten         0x20
/* Internal pfilter remove flag           */
#define InstFInternalPFilterRemove          0x40
```

Net Flags

```
typedef unsigned short  NetFlags;
#define NetFMember   0x0001        /* This net is a subnet member of a NetBus */
#define NetFGlobal   0x0002        /* Global Net, connectivity by name        */
#define NetFZombie   0x0004        /* This object is to be deleted            */
#define NetFVisit    0x0008        /* Visit flag for processing               */
#define NetFPower    0x0010        /* Power Net                               */
#define NetFGround   0x0020        /* Ground Net                              */
#define NetFNegPower 0x0040        /* Negative Power Net                      */
#define NetFPG      (NetFPower   \
                    |NetFGround  \
                    |NetFNegPower) /* Net is either power, ground or negpower */
#define NetFVisit2   0x0080        /* Visit flag for processing               */
#define NetFVisit3   0x0100        /* Visit flag for processing               */
#define NetFAutoGen  0x0200        /* Is an auto generated net                */
#define NetFHide     0x0400        /* Hide net                                */
#define NetFTarget   0x0800        /* Cone Extraction target                  */
#define NetFRed      0x1000        /* general purpose flag                    */
#define NetFGreen    0x2000        /* general purpose flag                    */
#define NetFBlue     0x4000        /* general purpose flag                    */
#define NetFYellow   0x8000        /* general purpose flag                    */

typedef unsigned short NetFlagsInternal;
/* Net is a potential power.           */
#define NetFInternalPotentialPower             0x0001
/* Net is a potential ground.          */
#define NetFInternalPotentialGround            0x0002
/* Ignore net in zdi (ictrl).          */
#define NetFInternalIgnore                     0x0004
/* Mark OOM nets.                      */
#define NetFInternalOOMNet                     0x0008
/* Don't route in schematic.           */
#define NetFInternalConnectByName              0x0010
/* Defined as wire in Verilog.         */
#define NetFInternalWire                       0x0020
/* All bus members are one-to-one connected. */
#define NetFInternalNetBusWideConnection       0x0040
```

Constant Net Values

```
enum zNetValue {
    NetVUndef  = 0,
    NetV0      = 1,
    NetVGround = 1,
    NetV1      = 2,
    NetVPower  = 2,
    NetVZ      = 3,
    NetVX      = 4,
    NetVNeg    = 5,
    NetBusVSet = 6
};
```

Primitive Level Flags

```
typedef unsigned short PrimitiveLevel;
#define PrimitiveLevelDefault  0x01 /* Only Primitive types are prim          */
#define PrimitiveLevelOperator 0x02 /* Cells with a known function are prim   */
#define PrimitiveLevelLibCell  0x04 /* Cells flagged as library cells are prim*/
#define PrimitiveLevelSymbol   0x08 /* Cells with @symbol attr are primitive  */
#define PrimitiveLevelFlagged  0x10 /* Modules with flag leafcell are prim    */
#define PrimitiveLevelCelldefine 0x20 /* Modules inside `celldefine are prim  */
```

Instance Orientation

```
enum zOrientation {
    OrientFR0,
    OrientFR90,
    OrientFR180,
    OrientFR270,
    OrientFMY,
    OrientFMYR90,
    OrientFMX,
    OrientFMXR90
};
```

Database Flags

```
typedef unsigned short DBFlags;
#define DBFPreplace     0x0001    /* Preplace info exists                  */
#define DBFTopUnused    0x0002    /* DB has unused top modules             */
#define DBFTopGuessed   0x0004    /* DB has guessed top modules            */
```

zFlagObjType

```
enum zFlagObjType {
    FlagObjCell     = 0,
    FlagObjMod      = 1,
    FlagObjInst     = 2,
    FlagObjNet      = 3,
    FlagObjPort     = 4,
    FlagObjPin      = 5,
    FlagObjDB       = 6,
    FlagObjVirtual  = 7
};
```

Highlight Colors

Highlight Color Info Struct

```
typedef struct HiCol {
    unsigned char norm;    /* Normal highlight color.     */
    unsigned char perm;    /* Permanent highlight color.  */
} HiCol;
```

Object Definition Structures

Definition of a Single Bit Interface Port

```
typedef struct Port {
    char*             name;          /* Port name                              */
    struct Net*       net;           /* Inside net or NULL (Module Ports only)*/
    char**            attrList;      /* Dynamic list of name=value attributes */
    int*              arcList;       /* List of arcs for cone extraction      */
    struct Spos*      spos;          /* Pointer to first Spos or NULL         */
    int               busMember;     /* Member of PortBus or -1               */
    PortFlags         flags;         /* Port direction and other flags        */
    HiCol             hicol;         /* Highlight Color                       */
    PortFlagsInternal internalFlags; /* Internal Port Flags                   */
    char              unused[6];
} Port;
```

Definition of an Interface PortBus

A PortBus is a group of Ports.

```
typedef struct PortBus {
    char*        name;          /* PortBus name                                */
    char**       attrList;      /* Dynamic list of name=value attributes       */
    struct Spos* spos;          /* pointer to first Spos or NULL               */
    int          rangestart;    /* range index at Cell.portList[first]         */
    int          first;         /* First Member: Cell.portList[first] (LSB)    */
    int          width;         /* Bus width: members [first...first+width)    */
    PortFlags    flags;         /* Port direction and other flags              */
    HiCol        hicol;         /* Highlight Color                             */
    signed char  rangedir;      /* range direction: 0=off 1=ascent -1=descent */
    char         unused[7];
} PortBus;
```

Definition of a Cell Object

```
typedef struct Cell {
    char*             name;          /* Cell name                              */
    Port*             portList;      /* List of single bit Ports - ordered     */
    PortBus*          portBusList;   /* List of PortBus - ordered, or NULL     */
    char**            attrList;      /* Dynamic list of name=value attributes */
    struct Spos*      spos;          /* pointer to first Spos or NULL          */
    HashTable         portTable;     /* String HashTable for Ports             */
    int               refCount;      /* Number of referencing instances        */
    PrimitiveFunc     func;          /* Cell function                          */
    CellFlags         flags;         /* Cell Flags                             */
    CellFlagsInternal internalFlags; /* Internal use only (no Tcl API)         */
    HiCol             hicol;         /* Highlight Color                        */
    char              unused[2];
} Cell;
```

Definition of a Primitive

```
typedef struct Primitive {
    Cell                c;              /* Is a Cell                          */
} Primitive;
```

Definition of a Module

```
typedef struct Module {
    Cell                c;             /* Is a Cell                          */
    struct Inst**       instList;      /* Dynamic list of Instances          */
    struct Net**        netList;       /* Dynamic list of Nets               */
    struct NetBus**     netBusList;    /* Dynamic list of Buses              */
    struct FTopNode*    topNode;       /* ptr to top flat node               */
    HashTable           instTable;     /* String HashTable for Instances     */
    HashTable           netTable;      /* String HashTable for Nets          */
    HashTable           nbusTable;     /* String HashTable for Buses         */
    char*               schematicCache; /* Store schematic layout information */
    ModuleFlags         mflags;        /* Flags in addition to c.flags       */
    ModuleFlagsInternal internalFlags; /* Internal module Flags              */
    char                unused[4];
} Module;
```

Pin Flags

```
typedef unsigned short PinFlags;
#define PinFTarget     0x0004  /* General purpose flag, use for Cone Extract */
#define PinFVisit      0x0008  /* Visit flag for processing                  */
#define PinFRed        0x0010  /* General purpose flag                       */
#define PinFGreen      0x0020  /* General purpose flag                       */
#define PinFBlue       0x0040  /* General purpose flag                       */
#define PinFYellow     0x0080  /* General purpose flag                       */
#define PinFHide       0x0800  /* Primitive/module pin is hidden in zdi.c    */

typedef unsigned char PinFlagsInternal;
#define PinFInternalOOMR    0x01  /* Pin has OOMR connected in some hierarchy */
#define PinFInternalAutoGen 0x02  /* Pin is auto generated (not in netlist)   */
```

Definition of Instance Pins

The instance pins refer to the Cell pins in the same order.

```
typedef struct Pin {            /* Single Bit Pin                             */
    struct Net*  net;           /* connected Net - may be NULL if unconnected */
    struct Spos* spos;          /* pointer to first Spos or NULL              */
    char**       attrList;      /* Dynamic list of name=value attributes      */
    HiCol        hicol;         /* Highlight Color                            */
    PinFlags     flags;         /* Pin flags                                  */
    PinFlags     internalFlags; /* Internal Pin flags                         */
    char         unused[2];
} Pin;
```

Definition of an Instance Object

```
typedef struct Inst {
    char*             name;        /* Instance name                          */
    Cell*             cellRef;     /* Instance cell interface                */
    char**            attrList;    /* Dynamic list of name=value attributes  */
    struct Spos*      spos;        /* pointer to first Spos or NULL          */
    HiCol             hicol;       /* Highlight Color                        */
    InstFlags         flags;       /* Instance Flags                         */
    InstFlagsInternal internalFlags; /* Internal Instance flags             */
    unsigned short    orient:3;    /* enum zOrientation for VdiInstOrientMask */
    unsigned short    unused:13;
    Pin               pin[1];      /* ordered list of Pins                   */
} Inst;
```

Definition of Net Connectivity

The Net's pinRefList refers to instance pins or to this module's Ports (if PinRef.inst == NULL).

```
typedef struct PinRef {         /* Reference to an inst Pin                  */
    Inst*       inst;           /* Pointer to the instance or NULL           */
    int         pinNo;          /* inst->pin[pinNo] or Cell.portList[pinNo]  */
    char unused[4];
} PinRef;
```

Definition of a Net Object

```
typedef struct Net {
    char*             name;          /* Net name                            */
    PinRef*           pinRefList;    /* Dynamic list defines Connectivity   */
    char**            attrList;      /* Dynamic list of name=value attributes */
    struct Spos*      spos;          /* pointer to first Spos or NULL       */
    zInt32            busMember;     /* Member of NetBus or -1              */
    HiCol             hicol;         /* Highlight Color                     */
    NetFlagsInternal internalFlags; /* Internally used net flags.           */
    NetFlags          flags;         /* Net type and other flags            */
    char              value;         /* Value of the net                    */
    char              unused[5];
} Net;
```

Definition of a NetBus Object

```
typedef struct NetBus {
    char*        name;        /* NetBus name                                  */
    Net**        subnetList;  /* SubNets (LSB first) may contain NULL nets    */
    char**       attrList;    /* Dynamic list of name=value attributes        */
    struct Spos* spos;        /* pointer to first Spos or NULL                */
    int          rangestart;  /* range index at subnetList[0]                 */
    NetFlags     flags;       /* Net type and other flags                     */
    NetFlagsInternal internalFlags; /* Internally used netBus flags.          */
    signed char  rangedir;    /* range direction: 0=off 1=ascent -1=descent   */
    char         value;       /* Value of the netBus                          */
    HiCol        hicol;       /* Highlight Color                              */
    char         unused[4];
} NetBus;
```

Definition of a VirtualObject

```
typedef unsigned short VirtualFlags;
#define VirtualFZombie 0x0004 /* This object is to be deleted                */

typedef enum zVirtualObjectType {
    VIRTUALOBJECT_NONE       = 0,
    VIRTUALOBJECT_MACRO      = 1,
    VIRTUALOBJECT_PARAMETER  = 2,
    VIRTUALOBJECT_MACRO_DEF  = 3,
    VIRTUALOBJECT_MACRO_REF  = 4,
    VIRTUALOBJECT_TASK       = 5,
    VIRTUALOBJECT_FUNCTION   = 6,
    VIRTUALOBJECT_WDB_VAR_ID = 7,
    VIRTUALOBJECT_INVALID    = 8
} zVirtualObjectType;

zVirtualObjectType zVirtualObject_str2type(const char* str, zBool icase);
const char*  zVirtualObject_type2str(zVirtualObjectType type);

typedef struct VirtualObject {
    char*              name;     /* VirtualObject name                       */
    char**             attrList; /* Dynamic list of name=value attributes    */
    struct Spos*       spos;     /* pointer to first Spos or NULL            */
    zVirtualObjectType type;     /* Type                                     */
    VirtualFlags       flags;
    char               unused[2];
} VirtualObject;
```

Forward Definition of OIDs

OIDs do not resist in DB memory.

```
struct OID;
struct OIDMemBlock;
struct OIDMemory {
    struct OIDMemBlock* blk;
    struct OID*         free;
    int                 blkCount;
    int                 freeCount;
};
```

Definition of the Database Hooks

```
typedef struct Hooks {
    struct FlatCache* flatCache;            /* for flat node cache           */
    struct TNode*     tree;
    struct OIDMemory  oid;
    struct Efile**    efileList;
    char*             sposBase;             /* zSposNewFile noconvert=zTrue  */
    char*             binfileName;          /* Name of the binfile.          */
    struct PopulateInfo* populateInfo;
    struct ParasiticInfo* parasiticInfo;
    zBool             debugMem;
    zBool             debugMemSize;
    zBool             debugMemFreeList;
    zBool             useTree;
    zBool             legacyBinfile;
} Hooks;
```

Definition of the Database Root

```
typedef struct DB {
    struct Strmem*     strmem;
    Cell**             cellList;       /* Dynamic list of Cells               */
    Module**           topList;        /* Dynamic list of Top modules or NULL */
    Module**           savedTopList;   /* Dynamic list of Top modules or NULL */
    HashTable          cellTable;      /* String HashTable for Cell namespace */
    HashTable          parasiticTable; /* Str HashTable for parasitic Modules */
    VirtualObject**    virtualObjects; /* Dynamic list of virtual objects     */
    struct Sfile**     sfileList;      /* Dynamic list of Sfiles or NULL      */
    char**             attrList;       /* List of name=value attributes       */
    struct Symlib*     symlibs;        /* list of symlibs                     */
    HashTable          symdefTable;    /* Str HashTable for port/pgnet symdef */
    int                parserBits;     /* each used parser sets bits          */
    DBFlags            flags;          /* DB Flags                            */
    char               hiersep;        /* usable hierarchy separator character*/
    char               internalHierSep;
    Hooks              hooks;          /* DB hooks must be at the end of DB   */
} DB;
```

Database Hash Table Implementation

Each HashTable struct must get initialized by zTableInit() before it can be used. A hash table stores key-value pairs. Currently, STRING and PTR are supported as hash-keys but anything can be added in future. The valueSize argument defines the number of bytes of the hash-value.

For STRING hash-key: the key "const char*" points to a zero-terminated string; that must be non-volatile memory if given to zTableInsert(). For PTR hash-key: the key "const char*" is used as "void*" and is not de-referenced (just the pointer value is used).

The zTableIgnoreCase() has only effect on STRING and ICASE Tables; those Tables are switched from STRING to ICASE and back. This only affects the behavior of zTableFind()

not the behavior of zTableInsert() and zTableRemoveObj().

zTableRemoveObj() removes the entry with identical key and identical value.

zTableInit

```
void zTableInit(
    DB*,
    HashTable*,
    int        valueSize,
    HashKeyType
);
```

zTableAlloc

```
void zTableAlloc(
    DB*,
    HashTable*,
    int        nBuckets
);
```

zTableFree

```
void zTableFree(
    DB*,
    HashTable*
);
```

zTableInsert

```
void  zTableInsert(
    DB*,
    HashTable*,
    const char* key,
    const void* valuePtr
);
```

zTableFind

```
zBool zTableFind(
    DB*,
    HashTable*,
    const char* key,
    void*       valuePtr_return
);
```

zTableRemoveObj

```
zBool zTableRemoveObj(
    DB*,
    HashTable*,
    const char* key,
    const void* valuePtr
);
```

zTableIgnoreCase

```
void  zTableIgnoreCase(
    DB*,
    HashTable*,
    zBool      icase
);
```

Database Hash Table Iterator

zIterInit

```
void  zIterInit(
    HashIter*,
    HashTable*
);
```

zIterMore

```
zBool zIterMore(
    HashIter*
);
```

zIterNext

```
zBool zIterNext(
    HashIter*
);
```

zIterKey

```
void  zIterKey(
    HashIter*,
    const char** keyPtr_return
);
```

zIterValue

```
void  zIterValue(
    HashIter*,
    void*     valPtr_return
);
```

Database Service and Access Function

zUniqueName

Append _%d at the end of the given name and repeats incrementing until this name is unique in the given hash table.

```
const char* zUniqueName(
    DB*,
    HashTable*,
    const char* name,
    char*       buf,
    int         maxlen,
    zBool       icase
);
```

zUniqueNameAlloc

If 'name' is unique in the hash table, NULL is returned, otherwise a new unique string is returned, which must be 'zFree'd by the caller.

```
char* zUniqueNameAlloc(
    DB*,
    HashTable*,
    const char* name,
    zBool       icase
);
```

zIdenticalInterface

Test if the interface of two given Cells is identical. If icase is zTrue, then the port names are compared using zStrcasecmp.

```
zBool zIdenticalInterface(
    Cell* oldCell,
    Cell* newCell,
    zBool icase
);
```

zSimilarInterface

Test if the interface of two given Cells is identical. If icase is zTrue, then the port names are compared using zStrcasecmp. Ports at oldCell will match portBusses on newCell.

```
zBool zSimilarInterface(
    DB*   db,
    Cell* oldCell,
    Cell* newCell,
    zBool icase
);
```

zModifyFlags

Modify the given flag at all database objects, mode can be: 0 set 1 clear flagsToModify are in the same order as enum zFlagObjType.

```
typedef enum zModifyFlags_Mode {
    ZMODIFYFLAGS_MODE_SET=0,
    ZMODIFYFLAGS_MODE_CLEAR=1
} zModifyFlags_Mode;

zBool zModifyFlags(
    DB*      db,
    unsigned flagsToModify[8],
    int      mode,
    zBool    procModule,
    zBool    procPrimitive,
    zBool    procInst,
    zBool    procNet,
    zBool    procNetBus,
    zBool    procPort,
    zBool    procPortBus,
    zBool    procPin,
    zBool    procVirtual
);
```

zStr2Flag

Get flag bit for given string.

```
zBool zStr2Flag(
    enum zFlagObjType type,
    const char*       name,
    unsigned short*   flagsPtr,
    unsigned short*   iflagsPtr,
    unsigned short*   mflagsPtr,
    unsigned short*   miflagsPtr
);
```

zFlag2Str

Get string list from bits.

```
int zFlag2Str(
    enum zFlagObjType type,
    unsigned short    flags,
    unsigned short    iflag,
    unsigned short    mflags,
    unsigned short    miflags,
    const char**      strs,
    unsigned          maxStrs
);
```

zFlag2Dump

Get string list from bits for dump.

```
int zFlag2Dump(
    enum zFlagObjType type,
    unsigned short    flags,
    unsigned short    iflag,
    unsigned short    mflags,
    unsigned short    miflags,
    const char**      strs,
    unsigned          maxStrs
);
```

zGetVersion

Return zdb version string.

```
void zGetVersion(
    char[512],
    const char* what
);
```

zGetMemoryUsage

Return memory usage in given buf (human readable).

```
void zGetMemoryUsage(
    DB*  db,
    char sep,
    char buf[800]
);
```

zIsEmpty

Check if a cell refers to an empty module.

```
zBool zIsEmpty(
    Cell* cell,
    zBool checkDanglingNets,
    zBool checkZombie
);
```

zCellIsModule

Check if a given cell is a module.

```
#define zCellIsModule(cell)     \
    ((cell)->flags & CellFModule)
```

zCellIsModuleMem

Check if a given cell originally is a module.

```
#define zCellIsModuleMem(cell)     \
    ((cell)->flags & CellFModuleMem)
```

zCellIsPrimitive

Check if a given cell is a primitive.

```
#define zCellIsPrimitive(cell)     \
    (!((cell)->flags & CellFModule))
```

zCellIsPrimitiveMem

Check if a given cell originally is a primitive.

```
#define zCellIsPrimitiveMem(cell) \
    (!((cell)->flags & CellFModuleMem))
```

zCellIsParasitic

Check if a given cell is a parasitic module.

```
#define zCellIsParasitic(cell) \
    (zCellIsModule(cell) && (((Module*)(cell))->mflags & ModuleFParasitic))
```

zModuleIsTop

Check if a given module is a top module.

```
#define zModuleIsTop(mod) \
    ((mod)->mflags & ModuleFTop)
```

PINCOUNT

Get the number of instance pins.

```
#define PINCOUNT(inst) \
    LISTLEN((inst)->cellRef->portList)
```

Create and Access Database Objects

Many functions below accept a const char* (const char*) for the object name that can point to volatile memory (like stack memory). The zNew functions internally allocate private memory and copy the name.

Those functions that return a pointer to a DB object may return NULL to indicate an error. In that case, the error message is stored in zLastErrorMsg (see zerror.h).

The functions zNewPort and zSearchPort return the index into the cell's portList, or -1 to indicate an error (message in zLastErrorMsg). The Function zNewPortBus return the index into the cell's portBusList, or -1 to indicate an error (message in zLastErrorMsg).

The zBool-returning functions return zTrue in case of success or zFalse in case of an error (message in zLastErrorMsg).

zNewModule

```
Module* zNewModule(
    DB*,
    const char* name
);
```

zTopModule

```
zBool zTopModule(
    DB*,
    Module* mod
);
```

zUnsetTopModule

```
zBool zUnsetTopModule(
    DB*,
    Module* mod
);
```

zIsTmpTop

Check if tmp top is used.

```
zBool zIsTmpTop(
    DB*
);
```

zUnusedTopCandidates

```
zBool zUnusedTopCandidates(
    DB*
);
```

zGuessedTopModules

```
zBool zGuessedTopModules(
    DB*
);
```

zNewPrimitive

```
Primitive* zNewPrimitive(
    DB*,
    const char*   name,
    PrimitiveFunc func
);
```

zSetPrimitives

```
void zSetPrimitives(
    DB*,
    PrimitiveLevel level
);
```

zSearchCell

```
Cell* zSearchCell(
    DB*,
    const char* name,
    zBool       icase,
    zBool       errMsg
);
```

zSearchTopModule

```
Module* zSearchTopModule(
    DB*,
    const char* name,
    zBool       icase,
    zBool       errMsg
);
```

zRenameCell

```
void zRenameCell(
    DB*,
    Cell*,
    const char* name
);
```

zDeleteCell

```
void zDeleteCell(
    DB*,
    Cell*
);
```

zNewParasitic

```
Module* zNewParasitic(
    DB*,
    const char* name
);
```

zSearchParasitic

```
Module* zSearchParasitic(
    DB*,
    const char* name,
    zBool       icase,
    zBool       errMsg
);
```

zRenameParasitic

```
void zRenameParasitic(
    DB*,
    Module*     para,
    const char* name
);
```

zDeleteParasitic

```
void zDeleteParasitic(
    DB*,
    Module*
);
```

zIsParasiticModInst

```
zBool zIsParasiticModInst(
    Inst*
);
```

zIsCouplingParasiticPort

```
zBool zIsCouplingParasiticPort(
    Module* parasitic,
    int     portNo
);
```

zIsCouplingParasiticInst

```
zBool zIsCouplingParasiticInst(
    Inst* inst,
    zBool local
);
```

zNewInst

```
Inst* zNewInst(
    DB*,
    Module*,
    const char* name,
    Cell*       cellRef
);
```

zNewInstByIndex

```
Inst* zNewInstByIndex(
    DB*,
    Module*,
    const char* name,
    unsigned    index,
    Cell*       cellRef
);
```

zSearchInst

```
Inst* zSearchInst(
    DB*,
    Module*,
    const char* name,
    zBool       icase,
    zBool       errMsg
);
```

zRenameInst

```
void zRenameInst(
    DB*,
    Module*,
    Inst*,
    const char* name
);
```

zDeleteLastInst

```
zBool zDeleteLastInst(
    DB*,
    Module*,
    Inst*
);
```

zInstSetXY

```
void zInstSetXY(
    DB*,
    Module*,
    Inst*,
    const char* x,
    const char* y
);
```

zNewNet

```
Net* zNewNet(
    DB*,
    Module*,
    const char* name
);
```

zNewNetBus

```
NetBus* zNewNetBus(
    DB*,
    Module*,
    const char* name,
    int         width
);
```

zSearchNet

```
Net* zSearchNet(
    DB*,
    Module*,
    const char* name,
    zBool       icase,
    zBool       errMsg
);
```

zSearchNetBus

```
NetBus* zSearchNetBus(
    DB*,
    Module*,
    const char* name,
    zBool       icase,
    zBool       errMsg
);
```

zRenameNet

```
void zRenameNet(
    DB*,
    Module*,
    Net*,
    const char* name
);
```

zRenameNetBus

```
void zRenameNetBus(
    DB*,
    Module*,
    NetBus*,
    const char* name
);
```

zNewPort

```
int zNewPort(
    DB*,
    Cell*,
    const char* name,
    PortFlags   dir
);
```

zNewPortBus

```
int zNewPortBus(
    DB*,
    Cell*,
    const char* name,
    PortFlags   dir
);
```

zSearchPort

```
int zSearchPort(
    DB*,
    Cell*,
    const char* name,
    zBool       icase,
    zBool       errMsg
);
```

zSearchPortEscapedName

```
int zSearchPortEscapedName(
    DB*,
    Cell*,
    const char* name,
    zBool       icase,
    zBool       errMsg
);
```

zSearchPortBus

```
int zSearchPortBus(
    DB*,
    Cell*,
    const char* name,
    zBool       icase,
    zBool       errMsg
);
```

zSearchPortBusEscapedName

```
int zSearchPortBusEscapedName(
    DB*,
    Cell*,
    const char* name,
    zBool       icase,
    zBool       errMsg
);
```

zRenamePort

```
void zRenamePort(
    DB*,
    Cell*,
    Port*,
    const char* name
);
```

zRenamePortBus

```
void zRenamePortBus(
    DB*,
    Cell*,
    PortBus*,
    const char* name
);
```

zNewAttribute

```
char* zNewAttribute(
    DB*,
    char***     attrListPtr,
    const char* nameValue
);
```

zNewAttributeChk

```
char* zNewAttributeChk(
    DB*         db,
    char***     attrListPtr,
    const char* nameValue
);
```

zNewAttributeNv

```
char* zNewAttributeNv(
    DB*,
    char***     attrListPtr,
    const char* name,
    const char* val
);
```

zSetAttribute

```
char* zSetAttribute(
    DB*,
    char***     attrListPtr,
    const char* nameValue,
    zBool       ic
);
```

zSetAttributeNv

```
char* zSetAttributeNv(
    DB*         db,
    char***     attrListPtr,
    const char* name,
    const char* value,
    zBool       icase

);
```

zDeleteAttribute

```
void zDeleteAttribute(
    DB*,
    char**      attrList,
    const char* name,
    zBool       ic

);
```

zSearchAttributeValue

```
char* zSearchAttributeValue(
    DB*,
    char**      attrList,
    const char* name,
    zBool       ic
);
```

zSearchAttributeValues

```
const char** zSearchAttributeValues(
    DB*,
    char**      attrList,
    const char* name,
    zBool       ic
);
```

zSetAttributeList

```
void zSetAttributeList(
    DB*,
    char*** destListPtr,
    char**  srcList
);
```

zSetSchematicCache

```
void zSetSchematicCache(Module* module, const char* schematicLayout);
```

zGetSchematicCache

```
const char* zGetSchematicCache(Module* module);
```

zClearSchematicCache

```
void zClearSchematicCache(DB* db, Module* module);
```

zDeleteHilight

```
void zDeleteHilight(
    DB*,
    int col,
    int permanent
);
```

zNewPinRef

```
zBool zNewPinRef(
    DB*,
    Net*,
    Inst*,
    int   pinNo
);
```

zNewPortRef

```
zBool zNewPortRef(
    DB*,
    Net*,
    Cell*,
    int   portNo
);
```

zNewNetBusMember

```
zBool zNewNetBusMember(
    DB*,
    Module*,
    NetBus*,
    Net*    subnet
);
```

zNewNetBusMemberByIndex

```
zBool zNewNetBusMemberByIndex(
    DB*,
    Module*,
    int     busIndex,
    Net*    subnet
);
```

zValidateNetBus

```
zBool zValidateNetBus(
    DB*,
    Module*,
    NetBus*,
    int     width
);
```

zNewPortBusMember

```
zBool zNewPortBusMember(
    DB*,
    Cell*,
    int   portBusNo,
    int   portNo
);
```

zValidatePortBus

```
zBool zValidatePortBus(
    DB*,
    Cell*,
    int   portBusNo,
    int   width
);
```

zNewVirtualObject

```
VirtualObject* zNewVirtualObject(
    DB*                db,
    const char*        name,
    zVirtualObjectType type
);
```

zDeleteVirtualObject

```
void zDeleteVirtualObject(
    DB*            db,
    VirtualObject* obj
);
```

zSearchVirtualObject

```
VirtualObject* zSearchVirtualObject(
    DB*         db,
    const char* name,
    zBool       icase,
    zBool       errMsg
);
```

zDefineNetBusRange

Define a bus range like [31:0].

```
void zDefineNetBusRange(
    NetBus*,
    int     from,
    int     to
);
```

zDefinePortBusRange

Define a bus range like [31:0].

```
void zDefinePortBusRange(
    PortBus*,
    int      from,
    int      to
);
```

zUndefineNetBusRange

Remove bus range.

```
void zUndefineNetBusRange(
    NetBus*
);
```

zUndefinePortBusRange

Remove bus range.

```
void zUndefinePortBusRange(
    PortBus*
);
```

zHasNetBusRange

Return zTrue if a range is defined.

```
zBool zHasNetBusRange(
    NetBus*,
    int*    from,
    int*    to
);
```

zHasPortBusRange

Return zTrue if a range is defined.

```
zBool zHasPortBusRange(
    PortBus*,
    int*     from,
    int*     to
);
```

zGetSubnetByRangeIdx

Get a member by range index.

```
zBool zGetSubnetByRangeIdx(
    NetBus*,
    int     idx,
    Net**   net
);
```

zGetSubportByRangeIdx

Get a member by range index.

```
zBool zGetSubportByRangeIdx(
    Cell*  cell,
    int    portBus,
    int    rangeIdx,
    Port** port
);
```

zValidateNetBusRange

Validate the bus range information.

```
zBool zValidateNetBusRange(
    Cell*,
    NetBus*
);
```

zValidatePortBusRange

Validate the bus range information. Checks the bus range info if defined: The member names must comply with the Verilog/VHDL member naming rules. Example: bus name: DATA bus range from: 31 (MSB) bus range to: 8 (LSB) member names: DATA[31], DATA[30], ..., DATA[0]

```
zBool zValidatePortBusRange(
    Cell* cell,
    int   portBusNo
);
```

zBusOfNet

```
NetBus* zBusOfNet(
    Module*,
    Net*
);
```

zOneToOneBusConnection

```
zBool zOneToOneBusConnection(
    NetBus* netBus,
    Module* mod,
    Inst*   inst,
    int     busNo,
    zBool   checkSequence
);
```

zCheckRangeName

```
zBool zCheckRangeName(
    Cell*,
    const char*,
    const char*,
    int
);
```

Symbol Definitions

The functions below manage the symdefTable needed for graphical port and pg symbols. And symioList used for the mapping information.

zSetSymdef

Add/overwrite a symdef entry

```
void zSetSymdef(
    DB*         db,
    const char* name,
    const char* def
);
```

zGetSymdef

Return symdef entry or NULL

```
char* zGetSymdef(
    DB*         db,
    const char* name
);
```

zDeleteSymdef

Remove symdef entry

```
void zDeleteSymdef(
    DB*         db,
    const char* name
);
```

Delete Contents of Database

```

```

zDeleteContentsOfDataBase

Delete all objects (called from zCloseDataBase).

```
void zDeleteContentsOfDataBase(
    DB* db
);
```

zDeleteContentsOfModule

Delete all inst/net/netBundle in a Module.

```
void zDeleteContentsOfModule(
    DB*     db,
    Module* module
);
```

zDeleteContentsOfModulePreserveHierInsts

Like zDeleteContentsOfModule but preserves hierarchical instances.

```
void zDeleteContentsOfModulePreserveHierInsts(
    DB*     db,
    Module* module
);
```

zDeleteZombies

Delete those objects with the Zombie flag.

```
void zDeleteZombies(
    DB* db
);
```

zDeleteAllButFlat

Delete all expect top module and it's flat info.

```
void zDeleteAllButFlat(
    DB* db
);
```

zDeleteZombiesInModule

Delete those inst/net/netBundle with Zombie flag.

```
void zDeleteZombiesInModule(
    DB*     db,
    Module* module
);
```

zDeleteZombieModules

Delete those modules with the Zombie flag.

```
void zDeleteZombieModules(
    DB* db
);
```

zInvalidateAll

Is initially NULL but can be set to a callback functions to get called back before database objects go away.

```
extern void (*zInvalidateAll)(
    DB* db
);
```

zInvalidateZombies

Is initially NULL but can be set to a callback functions to get called back before database objects go away.

```
extern void (*zInvalidateZombies)(
    DB* db
);
```

zCellPointerIsEqual

Compare two cells pointer.

```
zBool zCellPointerIsEqual(
    Cell* cell1,
    Cell* cell2
);
```

zModulePointerIsEqual

Compare two module pointer.

```
zBool zModulePointerIsEqual(
    Module* module1,
    Module* module2
);
```

zNetPointerIsEqual

Compare two net pointer.

```
zBool zNetPointerIsEqual(
    Net* net1,
    Net* net2
);
```

zInstPointerIsEqual

Compare two inst pointer.

```
zBool zInstPointerIsEqual(
    Inst* inst1,
    Inst* inst2
);
```

zOppositeDir

return the opposite port/pin direction

```
PortFlags zOppositeDir(
    PortFlags dir
);
```

zDir2Str

return string for given direction flags.

```
const char* zDir2Str(
    PortFlags dir
);
```

zGetOriginalName

Get the original name of the given cell. Return NULL if the original name is not known.

```
char* zGetOriginalName(
    DB*   db,
    Cell* cell
);
```

zIdenticalInterfaceIgnoreMemberBrackets

Test if the interface of two given Cells is identical. If icase is zTrue, then the port names are compared using zStrcasecmp. Differences in '<','[' and '>', ']' are ignored.

```
zBool zIdenticalInterfaceIgnoreMemberBrackets(
    Cell* oldCell,
    Cell* newCell,
    zBool icase
);
```

zSearchPortEscapedNameIgnoreMemberBrackets

```
int zSearchPortEscapedNameIgnoreMemberBrackets(
    DB*,
    Cell*,
    const char* name,
    zBool       icase,
    zBool       errMsg
);
```

zSearchPortBusEscapedNameIgnoreMemberBrackets

```
int zSearchPortBusEscapedNameIgnoreMemberBrackets(
    DB*,
    Cell*,
    const char* name,
    zBool       icase,
    zBool       errMsg
);
```

The Database Memory Manager

Create and Open a Database

zNewDataBase

Create a new database.

```
#define zNewDataBase() \
        zNewDataBase_(BINFILE_VERSION_MAJOR, BINFILE_VERSION_MINOR)
```

zNewDataBase_

Do not use directly, use zNewDataBase instead.

```
DB* zNewDataBase_(
    int         major,
    int         minor
);
```

zOpenDataBase

open an existing database - binfile : name of the existing db file

```
#define zOpenDataBase(binfile) \
        zOpenDataBase_(binfile, zFalse, BINFILE_VERSION_MAJOR, \
                                        BINFILE_VERSION_MINOR)
```

zOpenDataBase_

Do not use directly, use a zOpenDataBase* function instead.

```
DB* zOpenDataBase_(
    const char* binfile,
    zBool       quick,
    int         major,
    int         minor
);
```

zCheckDataBase

Check the binfile for correct structure. Return zFalse (and set zLastErrorMsg) on error.

```
zBool zCheckDataBase(
    const char* binfile
);
```

zFileInfoDataBase

Print file header infos from given binfile into buffer.

```
zBool zFileInfoDataBase(
    const char* binfile,
    char*       buffer,
    int         sizeofbuffer
);
```

zParserBitsDataBase

Return parser bits from Binfile header.

```
int zParserBitsDataBase(
    const char* binfile
);
```

zCloseDataBase

Close database and free all memory. Return zFalse (and set zLastErrorMsg) on errors.

```
zBool zCloseDataBase(
    DB* db
);
```

zSaveDataBase

Write database to binfile. Return zFalse (and set zLastErrorMsg) on error.

```
zBool zSaveDataBase(
    DB*         db,
    const char* binfile
);
```

zSaveDataBaseCompact

Write database to compact binfile.

```
#define zSaveDataBaseCompact(db, binfile) zSaveDataBase(db, binfile)
```

Memory Allocation

zAllocDB

Allocate cleared memory.

```
void* zAllocDB(
    DB*    db,
    zULong size
);
```

zMallocDB

Allocate memory.

```
void* zMallocDB(
    DB*    db,
    zULong size
);
```

zFreeDB

Return memory chunk to free-list.

```
void  zFreeDB(
    DB*   db,
    void* chunk
);
```

zReallocDB

Reallocate larger memory chunk

```
void* zReallocDB(
    DB*    db,
    void*  chunk,
    zULong size
);
```

zReallocDynamicDB

Reallocate memory chunk with best size.

```
void* zReallocDynamicDB(
    DB*     db,
    zULong* sizePtr,
    zULong  incr_size,
    void*   chunk
);
```

zStrAllocDB

Allocate string memory.

```
char* zStrAllocDB(
    DB*         db,
    const char* name
);
```

zStrFreeDB

Not implemented.

```
#define zStrFreeDB(DB, STR) UNUSED(db) /* noop */
```

zStrFreeAllDB

Free all string memory.

```
void  zStrFreeAllDB(
    DB* db
);
```

zStrmemAllocLenDB

Allocate string memory with the given length.

```
char* zStrmemAllocLenDB(
    DB*             db,
    struct Strmem** strmemPtr,
    unsigned        len
);
```

zStrmemFreeAllDB

Free all string memory.

```
void  zStrmemFreeAllDB(
    DB*,
    struct Strmem**
);
```

zStrmemMemoryUsage

Return used string memory.

```
zULong zStrmemMemoryUsage(
    struct Strmem*,
    unsigned*      cnt
);
```

The Database Primitive Function Definition

Access Primitive Functions

Port Functions

Enumeration iof named port functions.

```
typedef enum PortFunc {
    PortFuncBAD,        /* Used to identify errors.                          */
    PortFuncSELECT,     /* Select    port (MUX, SELECTOR)                    */
    PortFuncSET,        /* Set       port (DFF, LATCH)                       */
    PortFuncRESET,      /* Reset     port (DFF, LATCH)                       */
    PortFuncCLOCK,      /* Clock     port (DFF)                              */
    PortFuncENABLE,     /* Enable    port (LATCH)                            */
    PortFuncDRAIN,      /* Drain     port (PMOS,NMOS)                        */
    PortFuncGATE,       /* Gate      port (PMOS,NMOS)                        */
    PortFuncSOURCE,     /* Source    port (PMOS,NMOS)                        */
    PortFuncBULK,       /* Bulk      port (PMOS,NMOS,NPN,PNP,DIODE,ZDIODE,   */
                        /*                 RES,CAP)                          */
    PortFuncCOLLECTOR,  /* Collector port (NPN,PNP)                          */
    PortFuncBASE,       /* Base      port (NPN,PNP)                          */
    PortFuncEMITTER,    /* Emitter   port (NPN,PNP)                          */
    PortFuncANODE,      /* Anode     port (DIODE,ZDIODE)                     */
    PortFuncCATHODE,    /* Cathode   port (DIODE,ZDIODE)                     */
    PortFuncPLUS,       /* Plus      port (CAP)                              */
    PortFuncMINUS,      /* Minus     port (CAP)                              */
    PortFuncONE,        /* First     port (RES)                              */
    PortFuncTWO,        /* Second    port (RES)                              */
} PortFunc;
```

CheckPortFunc

check or fix port direction flags. In full mode (mode=0), this procedure performs a full check of the cell interface (all ports and portBuses) concerning the port order, the direction flags, the PortFNeg flag and the bus widths.

In building-mode (mode=1 or 2), this function is assumed to be called after each Port (mode=1) or PortBus (mode=2) is completely added to the cell interface. If the last port/portBus has PortFUnknown direction, then the direction is FIXED (no error). However, if a direction is set, then it is checked and an error may be returned. In building-mode only some checks are performed.

The cell's PrimitiveFunc defines the function of the ports/portBuses by the port order. The docs at doc/api/tprim.html define details.

It returns zTrue in case of success or zFalse in case of an error (message in zLastErrorMsg).

```
zBool zCheckPortFunc(
    Cell* cell,
    int   mode
);
```

GetOrderNo

Return the order number of the given port function at the given cell. Supported port functions are defined by the PortFunc enumeration above.

```
int zGetOrderNo(
    Cell*    cell,
    PortFunc portFunc
);
```

GetFuncPortOrderNo

Return the order number of the given port function at the given primitive function. Supported port functions are defined by the PortFunc enumeration above.

```
int zGetFuncPortOrderNo(
    PrimitiveFunc func,
    PortFunc portFunc
);
```

Str2PortFunc

Return the PortFunc enum value for a given name.

```
PortFunc zStr2PortFunc(
    const char* name
);
```

GetFuncPort

Return the number of the port in the cell's portList for a given port order number.

```
int zGetFuncPort(
    Cell*  cell,
    int    orderNo,
    zBool* isBus
);
```

LookupPrimitiveFunc

name -→ PrimitiveFunc (or PrimFLAST on error).

```
PrimitiveFunc zLookupPrimitiveFunc(
    const char* function
);
```

PrintPrimitiveFunc

PrimitiveFunc -→ name.

```
const char* zPrintPrimitiveFunc(
    PrimitiveFunc func,
    CellFlags     flags
);
```

Access Primitive Types

IsTransistorDevice

Return true if cell is a transistor device.

```
zBool zIsTransistorDevice(
    Cell* cell
);
```

IsTransistor

Return true if cell is a transistor.

```
zBool zIsTransistor(
    Cell* cell
);
```

IsDeviceFunction

Return true if the given func is a device func.

```
zBool zIsDeviceFunction(
    PrimitiveFunc func
);
```

IsOperator

Return true if the given cell is an operator.

```
zBool zIsOperator(
    Cell* cell
);
```

CheckPortIsHidable

Returns zTrue if the given port is ok for the PortFHide flag.

```
zBool zCheckPortIsHidable(
    Cell* cell,
    int   portNo
);
```

OppositePort

Returns portno of opposite port, or -1 if not possible.

```
int zOppositePort(
    Cell* cell,
    int   portNo
);
```

PolarDevice

Return true if the given cell is a polar device.

```
zBool zPolarDevice(
    Cell* cell
);
```

zSetDisplayAttribute

Set the meta attribute format string how to display attributes at the given cell. If metaAttribute is NULL, then the internal default format string is used.

```
void zSetDisplayAttribute(
    DB*         db,
    Cell*       cell,
    const char* metaAttribute
);


typedef struct {
    const char* name;
    PortFlags   dir;
} PrimDescrPort;

typedef struct  {
    const char*        prefix;
    const char*        type;
    PrimDescrPort*     ports;
    enum PrimitiveFunc func;
    int                minPortCnt;
    int                maxPortCnt;
} PrimDescr;
```

zPrimPrefix2Func

Find function for given prefix (like MN, D).

```
enum PrimitiveFunc zPrimPrefix2Func(const char* prefix);
```

zPrimType2Decr

Find description for spice types (like NMOS, DIODE, NPN).

```
PrimDescr* zPrimType2Descr(const char* type);
```

zPrimFunc2Decr

Find description for functions (like PrimFNMOS etc.).

```
PrimDescr* zPrimFunc2Descr(enum PrimitiveFunc func);
```

zPrimModelName2Func

Guess model names (like nmos, nch) to primitive function.

```
enum PrimitiveFunc zPrimModelName2Func(const char* modelName);
```

The Database Validator

Validate Flags

Available Flags

Flags to control validate options.

```
/* namespace */
#define VALIDATE_NAMESPACE              (1U<<0)
#define VALIDATE_NAMESPACE_S            "namespace"
/* namespace and ignore case */
#define VALIDATE_ICASENAMESPACE         (1U<<1)
#define VALIDATE_ICASENAMESPACE_S       "icasenamespace"
/* spos */
#define VALIDATE_SPOS                   (1U<<2)
#define VALIDATE_SPOS_S                 "spos"
/* primFunction in mode9 */
#define VALIDATE_MODE9                  (1U<<3)
#define VALIDATE_MODE9_S                "mode9"
/* flat database */
#define VALIDATE_FLAT                   (1U<<4)
#define VALIDATE_FLAT_S                 "flat"
/* spos lineinfo */
#define VALIDATE_LINEPOS                (1U<<5)
#define VALIDATE_LINEPOS_S              "linepos"
/* cycle in hierarchy */
#define VALIDATE_TORDER                 (1U<<6)
#define VALIDATE_TORDER_S               "torder"
/* if visit flags are cleared */
#define VALIDATE_VISIT                  (1U<<7)
#define VALIDATE_VISIT_S                "visit"
/* if all flags are cleared */
#define VALIDATE_FLAGS                  (1U<<8)
#define VALIDATE_FLAGS_S                "flags"
/* net/bus values */
#define VALIDATE_VALUE                  (1U<<9)
#define VALIDATE_VALUE_S                "value"
/* that DB hiersep is unused */
#define VALIDATE_HIERSEP                (1U<<10)
```

```
#define VALIDATE_HIERSEP_S              "hiersep"
/* flat database incl. name matches */
#define VALIDATE_FLATPEDANTIC           (1U<<11)
#define VALIDATE_FLATPEDANTIC_S         "flatpedantic"
/* flat values at signals */
#define VALIDATE_FLATVALUE              (1U<<12)
#define VALIDATE_FLATVALUE_S            "flatvalue"
/* attribute format 'name=value' */
#define VALIDATE_ATTRIBUTES             (1U<<13)
#define VALIDATE_ATTRIBUTES_S           "attributes"
/* integrity of parasitic data */
#define VALIDATE_PARASITIC              (1U<<14)
#define VALIDATE_PARASITIC_S            "parasitic"
/* power/ground flags of nets */
#define VALIDATE_PG                     (1U<<15)
#define VALIDATE_PG_S                   "pg"
/* value nets only one pin. */
#define VALIDATE_SINGLEVALCONN          (1U<<16)
#define VALIDATE_SINGLEVALCONN_S        "singlevalconn"
/* for unused visible hiersep. */
#define VALIDATE_VISIBLEHIERSEP         (1U<<17)
#define VALIDATE_VISIBLEHIERSEP_S        "visiblehiersep"
/* for valid clock flags at cell. */
#define VALIDATE_CLOCK                  (1U<<18)
#define VALIDATE_CLOCK_S                 "clock"
/* for valid clock flags at cell/port*/
#define VALIDATE_CLOCKSTRICT            (1U<<19)
#define VALIDATE_CLOCKSTRICT_S          "clockstrict"
/* flags of bus members. */
#define VALIDATE_BUSMEMBERDIR           (1U<<20)
#define VALIDATE_BUSMEMBERDIR_S         "busmemberdir"
/* skill export namespace. */
#define VALIDATE_SKILLEXPORTNAMESPACE   (1U<<21)
#define VALIDATE_SKILLEXPORTNAMESPACE_S "skillexportnamespace"

/* keep going. */
#define VALIDATE_KEEPGOING              (1U<<22)
#define VALIDATE_KEEPGOING_S            "keepgoing"

/* attrList NULL pointer */
#define VALIDATE_ATTRLISTNULL           (1U<<23)
#define VALIDATE_ATTRLISTNULL_S         "atrrlistnull"

/* net bus wide connection */
#define VALIDATE_NETBUSWIDECONNECTION   (1U<<24)
#define VALIDATE_NETBUSWIDECONNECTION_S "netbuswideconnection"

/* defaults. */
#define VALIDATE_DEFAULTS               ( VALIDATE_NAMESPACE                    \
                                        | VALIDATE_SPOS                         \
                                        | VALIDATE_FLAT                         \
```

```
                                | VALIDATE_TORDER                   \
                                | VALIDATE_VALUE                   \
                                | VALIDATE_HIERSEP                 \
                                | VALIDATE_ATTRIBUTES              \
                                | VALIDATE_PARASITIC               \
                                | VALIDATE_PG                      \
                                | VALIDATE_CLOCK                   \
                                | VALIDATE_BUSMEMBERDIR            \
                                | VALIDATE_ATTRLISTNULL            \
                                | VALIDATE_NETBUSWIDECONNECTION    \
                                | VALIDATE_KEEPGOING               \
                                )
#define VALIDATE_DEFAULTS_S             "defaults"
```

Default Flags for Validation

Built-in defaults:

```
```

zValidateDB

Validate the given ZDB database.

```
zBool zValidateDB(
    DB* db,
    int flags
);
```

The Source Position API

Type Definitions for Spos

Spos Type

Definition of the type which the Spos belongs to. The order of the enumeration is used as a priority during pick operation: Higher numbers get prio over lower numbers.

```
enum SposType {
    SposInvalid,
    SposNext,
    SposInst,
    SposCell,
    SposNet,
    SposPin,
    SposPort,
    SposNetBus,
    SposPortBus,
    SposNull,
    SposVirtual
};
```

SposTypeFlags

Definition of the type flags use in bit flags.

```
typedef zUInt8 SposTypeFlags;
#define SposTypeUndefine   0x00
#define SposTypeFFile      0x01
#define SposTypeFLine      0x02
#define SposTypeFPrimitive 0x04
#define SposTypeFModule    0x08
#define SposTypeFPort      0x10
#define SposTypeFInst      0x20
#define SposTypeFPin       0x40
#define SposTypeFNet       0x80
```

Definition of a Spos

Each struct stores the index of the file in which this Spos lies in. The type indicates the type of DB object this spos belongs to. The special type SposNext is used to indicate, that there are more Spos in the list. Only the last entry points the DB object. Only begPos and length are stored, the endPos can be calculated as (begPos+length). Spos entries of type SposNext have an additional pointer to the last entry, for faster access to the corresponding DB object. Invalid Spos entries, which exist after a call to zSposInvalidate are marked as type SposInvalid.

```
#define SFILES_BITS 24
#define SPOS_MAX_SFILES (1L << SFILES_BITS)
typedef struct Spos {
    unsigned      sfileIdx:SFILES_BITS; /* index in db.sfiles (max: 1<<24)    */
    unsigned      BF_ENUM_type_:5;  /* SposType                               */
    unsigned      BF_ENUM_nt_:3;    /* nulltype if type==SposNull             */
    unsigned      length;           /* length of source file pos              */
    zFPos         begPos;           /* begin of source file position          */
    union {                         /* union ptr to next Spos or              */
        struct { struct Spos* next; struct Spos* last; } x;  /* ptr to next   */
                                                             /* and last spos */
                                                             /* of same obj   */
        struct { struct Spos* next; } invalid; /* nxt invalid/unused in bucket*/
        struct {Cell*   cell;                          }  c;  /* SposCell      */
        struct {Cell*   cell; int portNo;              }  p;  /* SposPort      */
        struct {Cell*   cell; int portBusNo;           } pb;  /* SposPortBus   */
        struct {Module* mod;  Inst*   inst; int pinNo;} pi;  /* SposPin       */
        struct {Module* mod;  Inst*   inst;            }  i;  /* SposInst      */
        struct {Module* mod;  Net*    net;             }  n;  /* SposNet       */
        struct {Module* mod;  NetBus* netBus;          } nb;  /* SposNetBus    */
        struct {VirtualObject* obj;                    }  v;  /* SposVirtual   */
    } u;
} Spos;
```

zSposGetType

Get the type of a spos.

```
#define zSposGetType(spos) \
    ((enum SposType)(spos)->BF_ENUM_type_)
```

zSposGetLastType

Get the type of the last spos entry.

```
#define zSposGetLastType(spos) \
    zSposGetType(zSposLAST(spos))
```

zSposGetBegPos

Get the beginning of spos.

```
#define zSposGetBegPos(spos) \
    ((spos)->begPos)
```

zSposGetLength

Get the length of spos.

```
#define zSposGetLength(spos) \
    ((spos)->length)
```

zSposGetEndPos

Get end of spos.

```
#define zSposGetEndPos(spos) \
    ((spos)->begPos + (spos)->length)
```

zSposGetSfile

Get the sfile pointer.

```
#define zSposGetSfile(db, spos) \
    ((db)->sfileList[(spos)->sfileIdx])
```

zSposSetType

Set the type of a spos.

```
#define zSposSetType(spos, sposTypeValue) \
    (spos)->BF_ENUM_type_ = (unsigned)(sposTypeValue)
```

zSposGetNullType

Get the OidNullType of a spos for a null Oid.

```
#define zSposGetNullType(spos) \
    ((enum OidNullType)(spos)->BF_ENUM_nt_)
```

zSposSetNullType

Set the OidNullType of a spos for a null Oid.

```
#define zSposSetNullType(spos, nullTypeValue) \
    (spos)->BF_ENUM_nt_ = (unsigned)(nullTypeValue)
```

Spos File Types

Define different Spos file types.

```
typedef enum {
    SposFileTUndef,
    SposFileTSpice2,
    SposFileTSpice3,
    SposFileTPSpice,
    SposFileTHSpice,
    SposFileTCalibre,
    SposFileTCdl,
    SposFileTSpectre,
    SposFileTSpectreLpe,
    SposFileTSpectrePex,
    SposFileTDspf,
    SposFileTEldo,
    SposFileTVerilog,
    SposFileTVhdl,
    SposFileTglVerilog,
    SposFileTEdif,
    SposFileTDef,
    SposFileTLef,
    SposFileTSpef,
    SposFileTLiberty,
    SposFileTEND
} SposFileType;
```

SposBucketRoot

Each bucket Root has a free list of invalid/unused spos entries. and a linked list of Spos buckets containing a fixed number of spos entries.

```
typedef struct SposBucketRoot {
    struct SposBucket* firstBucket;      /*ptr to first bucket */
    struct Spos*       firstInvalidSpos; /*ptr to first invalid/unused spos */
} SposBucketRoot;
```

Definition of the Sfile Structure

Each source file manages: - filepositions for the lines in SlineTabs - a list of buckets with Sposs which belong to a specific filepos range of fixed size - Spos which are too long and so belong to more than one range are kept in the largeBucket. The bucket ranges overlap, so small Spos, which lie at the border from one bucket range to the other, do not necessarily go into the largeBucket list.

```
typedef struct Sfile {
    struct SposBucketRoot* bucketRootList;  /* list of bucketRoots           */
    struct SposBucketRoot  largeBucketRoot; /* one bucketRoot for large sposs */
    struct SlineTab*    slineTabFirst;   /* linked list of SlineTabs         */
    struct SlineTab*    slineTabLast;    /* linked list of SlineTabs         */
    char*               fname;           /* absolute fname                   */
    char*               shortFname;      /* short fname                      */
    struct Spos*        spos;            /* start of spos without an object  */
    zFPos               fileSize;        /* size of file                     */
    zDate               modDate;         /* modification date of file        */
    zULong              refCnt;          /* no of Spos which ref sfile       */
    unsigned            slineLastOffset; /* lineno offset of last Tab        */
    unsigned            maxLineno;       /* max lineno                       */
    unsigned            maxColumn;       /* max line length                  */
    int                 ftype;           /* fname type from zConvertFileName */
                                         /* or -1 if noconvert               */
    zBool               slineHasGaps;    /* needs fill because of "gaps"     */
    zBool               notPopulated;    /* lineinfo not yet populated       */
    SposFileType        type;            /* type of source file              */
    unsigned            idx:SFILES_BITS; /* uniq index of file               */
} Sfile;
```

SlineTab

Filepositions of lines are managed in chunks for better memory usage/performance missing entries are preset to -1 the file position of each \n is entered in pos[].

```
typedef struct SlineTab {
    struct SlineTab *next;      /* linked list of other SlineTab    */
    unsigned         posCnt;    /* maximum number of file positions */
    zFPos            pos[1];    /* variable size                    */
} SlineTab;
```

zSposNewFile

Allocate a new Sfile, insert it into db→sfiles, and return it. In case of an error (e.g. too many SPOS files) NULL is returned and an error message is stored in the global error buffer. The given fname can be relative or absolute. Normally all fnames are stored absolute in the DB. Use noconvert=zTrue to store them as is.

```
Sfile* zSposNewFile(
    DB*          db,
    const char*  fname,
    zBool        noconvert,
    SposFileType type
);
```

zSposSearchFile

Search for file, return NULL if not found.

```
Sfile* zSposSearchFile(
    DB*         db,
    const char* fname,
    zBool       noconvert,
    zBool       setLastErrMsg
);
```

zSposMakeFile

Use zSposSearchFile to check whether file already exists. If not, create it with zSposNewFile.

```
Sfile* zSposMakeFile(
    DB*          db,
    const char*  fname,
    zBool        noconvert,
    SposFileType type
);
```

zSposGetFile

Get file by index, return NULL if invalid.

```
Sfile* zSposGetFile(
    DB*   db,
    int   idx,
    zBool setLastErrMsg
);
```

zSposSetBase

Set base name for zSposGetFilename() if noconvert=zTrue.

```
void zSposSetBase(
    DB*         db,
    const char* base
);
```

zSposGetFilename

Return Sfile.fname rel to current working dir in given buffer. Normally the buffer should be PATH_MAX long.

```
zBool zSposGetFilename(
    DB*      db,
    const    Sfile*,
    char*    buf,
    unsigned maxlen
);
```

zSposSetFilename

Set new filename.

```
zBool zSposSetFilename(
    DB*         db,
    Sfile*      sfile,
    const char* fname
);
```

zSposGetShortFilename

Return Sfile.shortfname rel.

```
const char* zSposGetShortFilename(
    DB*   db,
    const Sfile*
);
```

zSposSetShortFilename

Set short filename. Used only for display.

```
void zSposSetShortFilename(
    DB*         db,
    Sfile*      sfile,
    const char* fname
);
```

zSposXXXInsert

Insert new Spos into sfile, and link it with DB obj. Return zTrue if OK, else a warning message is in zLastErrorMsg.

```
zBool zSposCellInsert(
    DB*           db,
    Cell*         cell,
    Sfile*        sfile,
    zFPos         begPos,
```

```
    unsigned      length
);
zBool zSposPortInsert(
    DB*           db,
    Cell*         cell,
    int           port,
    Sfile*        sfile,
    zFPos         begPos,
    unsigned      length
);
zBool zSposPortBusInsert(
    DB*           db,
    Cell*         cell,
    int           portBusNo,
    Sfile*        sfile,
    zFPos         begPos,
    unsigned      length,
    zBool         members
);
zBool zSposInstInsert(
    DB*           db,
    Module*       module,
    Inst*         inst,
    Sfile*        sfile,
    zFPos         begPos,
    unsigned      length
);
zBool zSposPinInsert(
    DB*           db,
    Module*       module,
    Inst*         inst,
    int           pinNo,
    Sfile*        sfile,
    zFPos         begPos,
    unsigned      length
);
zBool zSposNetInsert(
    DB*           db,
    Module*       module,
    Net*          net,
    Sfile*        sfile,
    zFPos         begPos,
    unsigned      length
);
zBool zSposNetBusInsert(
    DB*           db,
    Module*       module,
    NetBus*       netBus,
    Sfile*        sfile,
    zFPos         begPos,
    unsigned      length,
```

```
    zBool         members
);
zBool zSposNullInsert(
    DB*           db,
    enum OidNullType nt,
    Sfile*        sfile,
    zFPos         begPos,
    unsigned      length
);
zBool zSposVirtualInsert(
    DB*            db,
    VirtualObject* obj,
    Sfile*         sfile,
    zFPos          begPos,
    unsigned       length
);
```

zSposNEXT

Get next spos or NULL if there are no more.

```
#define zSposNEXT(spos) \
    ((zSposGetType(spos) == SposNext) ? (spos)->u.x.next : NULL)
```

zSposLAST

Get last spos. Because these are implemented as macros they should not be used with arguments which have side effects.

```
#define zSposLAST(spos) \
    ((zSposGetType(spos) == SposNext) ? (spos)->u.x.last : (spos))
```

zSposXXXDelete

Delete all spos from given object. If there are no more spos for a file, the file is delete too. Module or Primitive delete all contained objects (ports, inst etc.) Return zTrue if OK, else a warning message is in zLastErrorMsg.

```
zBool zSposCellDelete(
    DB*   db,
    Cell* cell
);
zBool zSposPortDelete(
    DB*   db,
    Port* port
);
zBool zSposPortBusDelete(
    DB*      db,
    PortBus* portBus
);
zBool zSposInstDelete(
    DB*   db,
    Inst* inst
);
zBool zSposPinDelete(
    DB*   db,
    Inst* inst,
    int   pinNo
);
zBool zSposNetDelete(
    DB*  db,
    Net* net
);
zBool zSposNetBusDelete(
    DB*     db,
    NetBus* netBus
);
zBool zSposNullDelete(
    DB*              db,
    enum OidNullType nt,
    Sfile*           sfile
);
zBool zSposVirtualDelete(
    DB*            db,
    VirtualObject* obj
);
```

zSposXXXClone

Clone all Spos entries of an object. Clone list of spos entries - create a 1:1 copy, except if limit is set to non-zero, then only create up to limit spos entries.

```
zBool zSposCellClone(
    DB*,
    int*,
    Spos*,
    Cell*,
```

```
    int   limit
);
zBool zSposPortClone(
    DB*,
    int*,
    Spos*,
    Cell*,
    int,
    int   limit
);
zBool zSposPortBusClone(
    DB*,
    int*,
    Spos*,
    Cell*,
    int,
    int   limit
);
zBool zSposInstClone(
    DB*,
    int*,
    Spos*,
    Module*,
    Inst*,
    int   limit
);
zBool zSposPinClone(
    DB*     db,
    int*    sfileIdxMap,
    Spos*   src,
    Module* dmod,
    Inst*   dest,
    int     pinNo,
    int     limit
);
zBool zSposNetClone(
    DB*,
    int*,
    Spos*,
    Module*,
    Net*,
    int     limit
);
zBool zSposNetBusClone(
    DB*,
    int*,
    Spos*,
    Module*,
    NetBus*,
    int     limit
);
```

zSposUpdateCell

Update cell pointer, if needed

```
void zSposUpdateCell(
    Spos*,
    Cell*
);
```

zSposUpdatePinNo

Update pinNo pointer, if needed.

```
void zSposUpdatePinNo(
    Spos*,
    int
);
```

zSposPick

Returns the Spos of the object with the smallest filepos range with filePos in it. In case of nothing found, return NULL

```
Spos* zSposPick(
    DB*    db,
    Sfile* sfile,
    zFPos  filePos
);
```

zSposPickAcceptCB

Returns the Spos of the object with the smallest filepos range with filePos in it. In case of nothing found, return NULL For each possible result a callback is consulted. if it returns zFalse the result is ignored.

```
typedef enum {SposAcceptFalse, SposAcceptTrue, SposAcceptErr} SposAcceptRes;
typedef SposAcceptRes (*SposAcceptCB)(void* ctx, const char* cellName);

Spos* zSposPickWithAcceptCB(
    DB*    db,
    Sfile* sfile,
    zFPos  filePos,
    void* acceptCtx,
    SposAcceptCB acceptCB,
    zBool* okPtr
);
```

zSposPickList

Returns a list Spos* of the objects. In case of nothing found, return NULL

```
Spos** zSposPickList(
    DB*    db,
    Sfile* sfile,
    zFPos  filePos
);
```

zSposForeachRange

Find Spos in a given file/range and call callback foreach. Abort if callback return -1 (error) or 1 (break) if lastPos == -1 the last position of the file is used. continue loop if callback return 0 (ok). Return true if no error occurred. If uniq > 0, multiple oids with same type, beg/end are suppressed.

```
zBool zSposForeachRange(
    DB*      db,
    Sfile*   sfile,
    unsigned uniqCnt,
    zBool    sort,
    zFPos    firstPos,
    zFPos    lastPos,
    unsigned colBeg,
    unsigned colEnd,
    zFPos*   linePos,
    int      (*cb)(Spos*,void*),
    void*    info
);
```

zSposForeachRangeWithAcceptCB

Find Spos in a given file/range and call callback foreach. Abort if callback return -1 (error) or 1 (break) if lastPos == -1 the last position of the file is used. continue loop if callback return 0 (ok). Return true if no error occurred. If uniq > 0, multiple oids with same type, beg/end are suppressed. For each possible result a callback is consulted. if it returns zFalse the result is ignored.

```
zBool zSposForeachRangeWithAcceptCB(
    DB*      db,
    Sfile*   sfile,
    unsigned uniqCnt,
    zBool    sort,
    zFPos    firstPos,
    zFPos    lastPos,
    unsigned colBeg,
    unsigned colEnd,
    zFPos*   linePos,
    int      (*cb)(Spos*,void*),
    void*    info,
    SposAcceptCB acceptCB,
    void* acceptCtx
);
```

zSposInsertLine

Insert the file position for the lineno.

In case of an error, return NULL and store the error message in zLastErrorMsg. lineno : beginning at 1, should be up-counting. filepos : must be the position of the newline (\n) character.

If lineno is 0 the whole file will be scanned and all filepos will be determined. If the given lineno is not up-counting (is not exactly one above the previous zSposInsertLine call's lineno), then the missing linenos are computed on demand (in zSposSearchLine). Duplicate calls with identical (lineno, filepos) are ignored, but a duplicate lineno with a different filepos results in an error.

In case of an error, return NULL and store the error message in zLastErrorMsg.

```
zBool zSposInsertLine(
    DB*      db,
    Sfile*   sfile,
    unsigned lineno,
    zFPos    filepos
);
```

zSposInsertEOF

Define the file-size.

filesize : the file size (file position after the last char)

In case of an error, return NULL and store the error message in zLastErrorMsg.

Corner Cases: (1) "billy" -→ zSposInsertEOF (5) (2) "" -→ zSposInsertEOF (0) (3) "\n" -→ zSposInsertLine(1,0) zSposInsertEOF (1) (4) "billy\n" -→ zSposInsertLine(1,5) zSposInsertEOF (6) (5) "billy\nb" -→ zSposInsertLine(1,5) zSposInsertEOF (7) (6) "billy\nb\n" -→ zSposInsertLine(1,5) zSposInsertLine(2,7) zSposInsertEOF (8)

```
zBool zSposInsertEOF(
    DB*    db,
    Sfile* sfile,
    zFPos  filesize
);
```

zSposSearchLine

Search for lineno, begPos and endPos at a given filepos.

If zSposInsertLine() or zSposInsertEOF() calls were missing, then the file is opened and scanned to get the missing information. If that file-open fails, then return zFalse and store the error message in zLastErrorMsg.

If filepos is -1 the last line is searched.

If filepos is out of range (<0 || >= filesize) then, return zFalse and store the error message in zLastErrorMsg.

filepos : the file position between start-of-line and the newline char (included). lineno : return: the line number begPos : return: the position of start-of-line (first char in line). endPos : return: the position directly after the newline (\n) char (or the file size)

Corner Cases: (1) zSposSearchLine(0) -→ 1,0,5 zSposSearchLine(4) -→ 1,0,5 zSposSearchLine(5) -→ error (2) zSposSearchLine(0) -→ error (3) zSposSearchLine(0) -→ 1,0,1 (4) zSposSearchLine(0) -→ 1,0,6 zSposSearchLine(5) -→ 1,0,6 zSposSearchLine(6) -→ error (5) zSposSearchLine(0) -→ 1,0,6 zSposSearchLine(6) -→ 2,6,7 (6) zSposSearchLine(6) -→ 2,6,8 zSposSearchLine(7) -→ 2,6,8 zSposSearchLine(8) -→ error

```
zBool zSposSearchLine(
    DB*       db,
    Sfile*    sfile,
    zFPos     filepos,
    unsigned* lineno,
    zFPos*    begPos,
    zFPos*    endPos
);
```

zSposGetFilepos

Search for begPos and endPos for a given lineno

If zSposInsertLine() or zSposInsertEOF() calls were missing, then the file is opened and scanned to get the missing information. If that file-open fails, then return zFalse and store the error message in zLastErrorMsg.

if lineno == 0 then return the last line. if lineno is out of range (too big), then return zFalse and store the error message in zLastErrorMsg.

lineno : the line number, starting at 1 (or 0). begPos : return: the position of start-of-line (first char in line). endPos : return: the position directly after the newline (\n) char (or the file size)

Corner Cases: (1) file: "billy" zSposGetFilepos(0) -→ 0,5 - last line zSposGetFilepos(1) -→ 0,5 - line#1 zSposGetFilepos(2) -→ error (2) file: "" zSposGetFilepos(0) -→ error (3) file: "\n" zSposGetFilepos(0) -→ 0,1 - last line zSposGetFilepos(1) -→ 0,1 - line#1 zSposGetFilepos(2) -→ error (4) file: "billy\n" zSposGetFilepos(0) -→ 0,6 - last line zSposGetFilepos(1) -→ 0,6 - line#1 zSposGetFilepos(2) -→ error (5) file: "billy\nb" zSposGetFilepos(0) -→ 6,7 - last line zSposGetFilepos(1) -→ 0,6 - line#1 zSposGetFilepos(2) -→ 6,7 - line#2 (6) file: "billy\nb\n" zSposGetFilepos(0) -→ 6,8 - last line zSposGetFilepos(1) -→ 0,6 - line#1 zSposGetFilepos(2) -→ 6,8 - line#2 zSposGetFilepos(3) -→ error

```
zBool zSposGetFilepos(
    DB*      db,
    Sfile*   sfile,
    unsigned lineno,
    zFPos*   begPos,
    zFPos*   endPos
);
```

zSposDeleteAll

Delete all Spos data, free database memory.

```
void zSposDeleteAll(
    DB* db
);
```

zSposInvalidate

Flag Spos data of one object as SposInvalid.

```
void zSposInvalidate(
    DB*   db,
    Spos*
);
```

zSposGetOID

Get oid from given spos. Returns zFalse and set zLastErrorMsg if spos is SposInvalid.

```
zBool zSposGetOID(
    Spos*       spos,
    struct OID* oid
);
```

zSposValidate

Return false on error and report to zLastErrorMsg

```
zBool zSposValidate(
    DB*   db,
    zBool linePos
);
```

zSposMemoryUsage

Return total memory usage.

```
zULong zSposMemoryUsage(
    struct Sfile*,
    unsigned*     cnt
);
```

zSposUsage

Return number of pos and line entries.

```
void zSposUsage(
    struct  Sfile*,
    zULong* sposCnt,
    zULong* lineCnt
);
```

zSposFileType2Str

Return file type as a string.

```
const char* zSposFileType2Str(
    SposFileType type
);
```

zSposStr2FileType

Return file type as enum

```
SposFileType zSposStr2FileType(
    const char* str
);
```

zSposSfileIsEqual

Compare two sfile pointer.

```
zBool zSposSfileIsEqual(
    Sfile* sfile1,
    Sfile* sfile2
);
```

Object Identification (OID)

The Database Net/Pin/Inst/etc Identification Items

Type Definitions for OIDs

Path

Definition of the Path part of an OID These Paths are only created by malloc and are automatically freed (by zOID_unref) if refCount goes to 0.

```
typedef struct Path {           /* Object Identification                    */
    int         refCount;       /* How many OIDs are pointing to me         */
    int         pathLength;     /* Length of path                           */
    Module*     module;         /* The root module                          */
    Inst*       path[1];        /* The instance path                        */
} Path;
```

OID Type

Definition of the Object Identification type.

```
enum oidtypeE {
    OidBAD        =  0,
    OidModule     =  1,
    OidPrimitive  =  2,
    OidPort       =  3,
    OidPortBus    =  4,
    OidInst       =  5,
    OidPin        =  6,
    OidPinBus     =  7,
    OidNet        =  8,
    OidNetBus     =  9,
    OidSignal     = 10,
    OidNetSeg     = 11,
    OidNull       = 12,
    OidVirtual    = 13,
    OidLAST       = 14
};

typedef enum oidtypeE OidType;
```

OID Null Types

Definition of the OidNull object types.

```
typedef enum OidNullType {
    OidNullTNONE       =  0,
    OidNullTWhite      =  1,
    OidNullTGrey       =  2,
    OidNullTBlack      =  3,
    OidNullTComment    =  4,
    OidNullTUndefined  =  5,
    OidNullTColor      =  6,
    OidNullTERROR      =  7
} OidNullType;
```

OID Flags

Definition of OID flags

```
typedef unsigned char OidFlags;
#define OidFNone        0x00    /* No OID flag is set                          */
#define OidFPath        0x01    /* p.path is valid instead of p.module         */
#define OidFVisit       0x20    /* general purpose flag                        */
#define OidFFreeList    0x40    /* toid.c: This OID is in memory freelist      */
#define OidFMemToFree   0x80    /* toid.c: This OID is marked to get freed     */
```

OID Structure

Definition of the OID structure.

```
typedef struct OID {            /* Object Identification                    */
    OidType     type;
    OidFlags    flags;
    union {
        Module*     module;
        Primitive*  primitive;
        Path*       path;       /* if flags & OidFPath                      */
        struct OID* next;       /* if flags & OidFFreeList                  */
    } p;                        /* defines the parent                       */
    union {
        int         portNo;     /* also portBusNo                           */
        Inst*       inst;
        PinRef      pin;        /* also pinBus                              */
        Net*        net;        /* also signal                              */
        NetBus*     netBus;
        struct {
            Net*            net;
            unsigned short  pin0;
            unsigned short  pin1;
        }           netseg;
        OidNullType nullType;   /* for OidNull                              */
        VirtualObject* virtualObject;
    } o;                        /* defines the object inside the parent     */
} OID;
```

Access Functions to OIDs

zOID_getType

Get the type of 'oid'.

```
#define zOID_getType(oid) \
    (oid)->type
```

zOID_setType

Set the type of 'oid' to 'newType'.

```
#define zOID_setType(oid,newType) \
    (oid)->type = (newType)
```

zOID_EndOfPath

Get Cell at end of Path, that is the module or primitive referenced by the path's last instance.

```
Cell* zOID_EndOfPath(
    const OID* oid
);
```

zOID_samePath

return zTrue if both oids have same path.

```
zBool zOID_samePath(
    const OID* a,
    const OID* b
);
```

zOID_samePathMinus

return zTrue if both oids have same path, but ignore last element of a

```
zBool zOID_samePathMinus(
    const OID* a,
    const OID* b
);
```

zOID_ref

Increase the ref count of the path.

```
void zOID_ref(
    const OID*
);
```

zOID_unref

Decrease the ref count of the path and eventually free Path.

```
void zOID_unref(
    const OID*
);
```

Public Service Functions for Hash Tables with an OID Key

zOID_hash

```
unsigned zOID_hash(
    OID
);
```

zOID_cmp

```
int zOID_cmp(
    OID a,
    OID b
);
```

zOID_dictCmp

```
int zOID_dictCmp(
    OID a,
    OID b
);
```

zOID_IsEqual

```
zBool zOID_IsEqual(
    OID a,
    OID b
);
```

zOID_equal

```
int zOID_equal(
    OID a,
    OID b
);
```

zOID_cmpVP

```
int zOID_cmpVP(
    const void* objA,
    const void* objB
);
```

zOID_cmpP

```
int zOID_cmpP(
    const OID* a,
    const OID* b
);
```

zOID_dictCmpVP

```
int zOID_dictCmpVP(
    const void* objA,
    const void* objB
);
```

zOID_dictCmpP

```
int zOID_dictCmpP(
    const OID* a,
    const OID* b
);
```

zOID_equalP

```
int zOID_equalP(
    const OID* a,
    const OID* b
);
```

zOID_create

```
zBool zOID_create(
    DB*   db,
    int   argc,
    char* argv[],
    OID*  result,
    zBool icase
);
```

zOID_isBusMember

```
zBool zOID_isBusMember(
    const OID* oid,
    int*       bool_result
);
```

zOID_busOf

```
zBool zOID_busOf(
    const OID* oid,
    OID*       result
);
```

zOID_widthOf

```
zBool zOID_widthOf(
    const OID* oid,
    int*       result
);
```

zOID_hasRange

```
zBool zOID_hasRange(
    const OID* oid,
    int*       bool_result
);
```

zOID_rangeLsb

```
zBool zOID_rangeLsb(
    const OID* oid,
    int*       result
);
```

zOID_rangeMsb

```
zBool zOID_rangeMsb(
    const OID* oid,
    int*       result
);
```

zOID_isModule

```
zBool zOID_isModule(
    const OID* oid,
    int*       bool_result
);
```

zOID_isOperator

```
zBool zOID_isOperator(
    const OID* oid,
    int*       bool_result
);
```

zOID_isTop

```
zBool zOID_isTop(
    const OID* oid,
    int*       bool_result
);
```

zOID_isPgNet

```
zBool zOID_isPgNet(
    const OID* oid,
    int*       bool_result
);
```

zOID_moduleOf

```
zBool zOID_moduleOf(
    const OID* oid,
    OID*       result
);
```

zOID_primitiveOf

```
zBool zOID_primitiveOf(
    const OID* oid,
    OID*       result
);
```

zOID_down

```
zBool zOID_down(
    const OID* oid,
    OID*       result
);
```

zOID_up

```
zBool zOID_up(
    const OID* oid,
    OID*       result
);
```

zOID_refCount

```
zBool zOID_refCount(
    const OID* oid,
    int*       result
);
```

zOID_concat

```
zBool zOID_concat(
    const OID* inst,
    const OID* pin,
    OID*       result
);
```

zOID_changePin

Change pin no of a given pin/portOid.

```
zBool zOID_changePin(
    OID*  oid,
    int   pinNo
);
```

zOID_convertPin

Change pin/port of a pin/portOid. The resulting OID is stored where "result" points to. The result oid's refCount is NOT incremented - like for all OID-returning functions in this file, it is up to the caller to perform zOID_ref(result) and zOID_unref(result).

```
zBool zOID_convertPin(
    const OID*  oid,
    int         pinNo,
    OID*        result
);
```

zOID_convertToPin

Convert to pin oid. The resulting OID is stored where "result" points to. The result oid's refCount is NOT incremented - like for all OID-returning functions in this file, it is up to the caller to perform zOID_ref(result) and zOID_unref(result).

```
zBool zOID_convertToPin(
    const OID*  oid,
    Inst*       inst,
    int         pinNo,
    OID*        result
);
```

zOID_convertToNet

Convert to net oid. The resulting OID is stored where "result" points to. The result oid's refCount is NOT incremented - like for all OID-returning functions in this file, it is up to the caller to perform zOID_ref(result) and zOID_unref(result).

```
zBool zOID_convertToNet(
    const OID*  oid,
    Net*        net,
    OID*        result
);
```

zOID_convertToPort

Convert to port oid. The resulting OID is stored where "result" points to. The result oid's refCount is NOT incremented - like for all OID-returning functions in this file, it is up to the caller to perform zOID_ref(result) and zOID_unref(result).

```
zBool zOID_convertToPort(
    const OID*  oid,
    int         portNo,
    OID*        result
);
```

zOID_convertTo

Convert an OID. if pname == NULL pin -→ inst pinBus -→ inst signal -→ net netSeg -→ net port -→ module/primitive portBus -→ module/primitive

if pname is given pin ←- inst pinBus ←- inst port ←- module/primitive portBus ←- module/primitive

The resulting OID is stored where "result" points to. The OID's root module and path information is copied from the given oid. The result oid's refCount is NOT incremented - like for all OID-returning functions in this file, it is up to the caller to perform zOID_ref(result) and zOID_unref(result).

```
zBool zOID_convertTo(
    DB*         db,
    const OID*  oid,
    const char* rtype,
    const char* pname,
    OID*        result
);
```

zOID_convertable

Return what zOID_convertTo can do (with pname == NULL). The specialRes result is a 2nd possibility.

```
OidType zOID_convertable(
    const OID* oid,
    OidType*   specialRes
);
```

zOID_convertFromParasitic

Converts a parasitic OID to a normal OID.

```
zBool zOID_convertFromParasitic(
    DB*,
    const OID* oid,
    OID*
);
```

zOID_convertToParasitic

Converts a normal OID to a parasitic OID.

```
zBool zOID_convertToParasitic(
    DB*,
    const OID* oid,
    const OID*,
    OID*
);
```

zOID_hasParentInst

Checks if the inst/net oid has a path - that's equivalent to checking if the oid is tree-based (that means there is a parent one step up the design hierarchy).

```
zBool zOID_hasParentInst(
    const OID* oid,
    int*       bool_result
);
```

zOID_parentInst

Returns the parent inst (that means, the path is one entry shorter). Replace OID.o.inst by the previously last in path.

```
zBool zOID_parentInst(
    const OID* oid,
    OID*       result
);
```

zOID_hasParentMod

Checks if the oid has a containing module - this is almost always zTrue, except for non tree-based primitive or module OIDs.

```
zBool zOID_hasParentMod(
    const OID* oid,
    int*       bool_result
);
```

zOID_parentModule

Returns the parent module (that means, the type is set to OidModule). If however, the oid already is a module, then the path is cut by one entry.

```
zBool zOID_parentModule(
    const OID* oid,
    OID*       result
);
```

zOID_isConnected

Checks if the given pin or port OID connects to a net or not (unconnected).

```
zBool zOID_isConnected(
    const OID* oid,
    int*       bool_result
);
```

zOID_connectedNet2

Set netP to the connected net of the given pin or port OID.

```
zBool zOID_connectedNet2(
    const OID* oid,
    Net**       netP
);
```

zOID_connectedNet

Returns the OID for the net connected by the given pin or port OID.

```
zBool zOID_connectedNet(
    const OID* oid,
    OID*       result
);
```

zOID_createPath

If plen > 0 then allocate a Path and store a pointer to it in the given OID and fill the Path with the given "root" and "path" and set the OidFPath flag.

If plen == 0, then clear the OidFPath flag and store the given "root" into the given OID.

If path == NULL, then also allocate the Path, but leave it's array of Inst* uninitialized.

```
void zOID_createPath(
    OID*,
    Module* root,
    Inst**  path,
    int     plen
);
```

zOID_createReversePath

Like `zOID_createPath()` but the given path is in reverse order.

```
void zOID_createReversePath(
    OID*,
    Module* root,
    Inst**  path,
    int     plen
);
```

zOID_searchTreeBased

Searches a tree-OID that matches the given module-based OID - use hint as a starting point.

```
zBool zOID_searchTreeBased(
    DB*        db,
    const OID* oid,
    const OID* hint,
    OID*       result
);
```

zOID_createFromString

The given string 'str' is converted into 'oid'. The returned 'oid' will be of the given 'type'. The given 'hierSep' character is used to split 'str' in it's hierarchical path of instance names rooted at given module 'top' and an object name. If 'delim' is not '\0' it is used, instead of 'hierSep' for separating the object name. If 'escape' is not '\0' 'hierSep' and 'delim' characters following 'escape' will be added to the corresponding name. If 'topInstName' is not 'NULL' the first part of 'str' is ignored if it matches 'topInstName'. If 'icase' is zTrue instance name and object searching is case insensitive. If 'guess' is zTrue instance names which are not found directly will be searched by prepending an 'X'. If 'guess' is zTrue try replacing first 'X' with the following second character. The oid's refCount is NOT incremented - like for all OID-returning functions in this file, it is up to the caller to perform zOID_ref(oid) and zOID_unref(oid).

```
zBool zOID_createFromString(
    DB*         db,
    OID*        oid,
    OidType     type,
    Module*     top,
    const char* str,
    char        hierSep,
    char        delim,
    char        escape,
    const char* topInstName,
    ce_Bool     icase,
    ce_Bool     guess,
    ce_Bool     escapeVerilog,
    ce_Bool     autoPopulate
);
```

zOID_createFromStringList

```
zBool zOID_createFromStringList(
    DB*          db,
    OidType      type,
    const char** pathList,
    int          pathListLength,
    const char*  oname,
    ce_Bool      icase,
    ce_Bool      guessPath,
    ce_Bool      guessName,
    ce_Bool      autoPopulate,
    OID*         oid
);
```

zOID_createNetSeg

Create net seg oid from two pin/ports. The res's refCount is NOT incremented - like for all OID-returning functions in this file, it is up to the caller to perform zOID_ref(res) and zOID_unref(res).

```
zBool zOID_createNetSeg(
    OID* res,
    OID* oid1,
    OID* oid2
);
```

Access OID Information

Public functions that return a string

In case of an error, return NULL and store the error message in zLastErrorMsg.

zOID_printDirection

String from PortFlags or NULL to indicate an error.

```
char* zOID_printDirection(
    PortFlags dir
);
```

zOID_direction

set PortFlags of given dirP. return zFalse to indicate an error (zLastErrorMsg)

```
zBool zOID_direction(
    const OID* oid,
    PortFlags* dirP
);
```

zOID_directionOf

String from pin or port OID or NULL to indicate an error.

```
char* zOID_directionOf(
    const OID* oid
);
```

zOID_type

Get type string from OidType.

```
char* zOID_type(
    const OID* oid
);
```

zOID_oname

Get the object name.

```
char* zOID_oname(
    const OID* oid
);
```

zOID_pname

Get the pin name.

```
char* zOID_pname(
    const OID* oid
);
```

zOID_cname

Get the cell name.

```
char* zOID_cname(
    const OID* oid
);
```

zOID_netsegpname

Get net segment pin name.

```
char* zOID_netsegpname(
    const OID* oid,
    int        field /*0, 1, 2, 3*/
);
```

zOID_print

Print the given OID to a string.

```
char* zOID_print(
    DB          *db,
    const OID*   oid,       /* The OID to print.                */
    zBool        addType,   /* Add the object type.             */
    zBool        addWidth,  /* Add the width of a bus object.   */
    zBool        addRange,  /* Add the range of a bus object.   */
    zBool        typeTitle, /* Print the type as title.         */
    zBool        addRoot,   /* Add the root (toplevel) module.  */
    zBool        addPath,   /* Add the path to the object.      */
    zBool        addName,   /* Add the object type.             */
    zBool        tclName,   /* Add Tcl conform object names.    */
    zBool        nameAttr,  /* Use the @name attribute as the object name. */
    zBool        escapeVerilog, /* Escape identifiers for verilog.*/
    char         hiersep,   /* Hierarchy separator to use.      */
    char         delimiter, /* Pin delimiter character to use.  */
    char         joinchar,  /* Join all OID elements.           */
    DStr        *result     /* DStr to store the OID string     */
                            /* (result needs to be initialized). */
);
```

zOID_stringRepresentation

Create a Tcl-compatible string representation of the OID.

```
char* zOID_stringRepresentation(
    const OID*   oid,
    DStr* result
);
```

zOID_dump

dump oid as debug message. if msg is not NULL it is prepended as additional info.

```
void zOID_dump(
    const char*  msg,
    const OID*   oid
);
```

zOID_dumpList

dump oid list as debug message. if msg is not NULL it is prepended as additional info.

```
void zOID_dumpList(
    const char*  msg,
    const OID*   oidList
);
```

zOID_appendList

Append and ref an OID to an oidList.

```
void zOID_appendList(
    OID** oidList,
    OID*  oid
);
```

zOID_resetList

Unref all oids in a list and set the list length to 0.

```
void zOID_resetList(
    OID* oidList
);
```

zOID_freeList

unref all oids in a list and free array.

```
void zOID_freeList(
    OID* oidList
);
```

zOID_str2type

Get OidType from type string.

```
OidType zOID_str2type(
    const char* str,
    zBool       icase
);


OidType zOID_Str2Type(const char* str);
const char* zOID_Type2Str(OidType type);
void zOID_TypeUsageList(const char*** list);
```

zOID_nulltype2str

Get string from OidNullType.

```
const char* zOID_nulltype2str(
    OidNullType t
);
```

zOID_str2nulltype

Get OidNullType from nulltype string.

```
OidNullType zOID_str2nulltype(
    const char* str,
    zBool       errMsg
);
```

zOID_validate

Validate OID structure.

```
zBool zOID_validate(
    const OID* oid
);
```

zOID_setSchematicCache

```
zBool zOID_setSchematicCache(OID* oid, const char* schematicLayout);
```

zOID_getSchematicCache

```
const char* zOID_getSchematicCache(OID* oid);
```

zOID_clearSchematicCache

```
zBool zOID_clearSchematicCache(DB* db, OID* oid);
```

zOID_highlightGet

Get highlight color of the given OID.

```
zBool zOID_highlightGet(
    const OID* oid,
    int*       color,
    int        permanent
);
```

zOID_highlightSet

```
zBool zOID_highlightSet(
    const OID* oid,
    int        color,
    int        permanent
);
```

zOID_highlightDelete

Delete highlight color of the given OID.

```
zBool zOID_highlightDelete(
    const OID* oid,
    int        permanent
);
```

zOID_highlight2Get

get highlight in design, flat and parasitic.

```
zBool zOID_highlight2Get(
    DB*        db,
    const OID* oid,
    int*       color,
    int        permanent
);
```

zOID_highlight2Set

set highlight in design, flat and parasitic.

```
zBool zOID_highlight2Set(
    DB*        db,
    const OID* oid,
    int        color,
    int        permanent
);
```

zOID_highlight2Delete

delete highlight in design, flat and parasitic.

```
zBool zOID_highlight2Delete(
    DB*        db,
    const OID* oid,
    int        permanent
);
```

zOID_insertModule

Similar to zOID_down, but insert the new Instance at the given level.

```
zBool zOID_insertModule(
    const OID*,
    int        level,
    Inst*      newInst,
    OID*       res
);
```

zOID_removeModule

Counterpart to zOID_insertModule.

```
zBool zOID_removeModule(
    const OID*,
    int        level,
    OID*       res
);
```

zOID_htraverseAll

Similar to searchTreeBased but find all paths from given top to given target. For each path found call the callback function func with the path stored in stack.

```
struct htraverseAll {
    DB*     db;
    Cell*   target;
    Module* top;
    Inst*   stack[128];
    zBool   (*func)(struct htraverseAll*,int);
    void*   userData1;
    void*   userData2;
    int     userData3;
};
zBool zOID_htraverseAll(struct htraverseAll*, int stackTop);
```

zOID_isTopRoot

return true if OID is top based.

```
zBool zOID_isTopRoot(const OID* oid);
```

zOID_isParasitic

return true if OID is parasitic.

```
zBool zOID_isParasitic(const OID* oid);
```

zOID_rootModule

return root module of path (or NULL).

```
Module* zOID_rootModule(
    const OID* oid
);
```

zOID_setNull

set oid to null.

```
void zOID_setNull(
    OID* oid
);
```

zOID_setModulePort

set oid to module port.

```
void zOID_setModulePort(
    OID* oid,
    Module* mod,
    int portNo
);
```

zOID_setModule

set oid to module.

```
void zOID_setModule(
    OID* oid,
    Module* mod,
    Inst** path,
    int plen
);
```

zOID_setNet

set oid to net.

```
void zOID_setNet(
    OID* oid,
    Module* mod,
    Inst** path,
    int plen,
    Net* net
);
```

zOID_setNetFromNetBus

set oid to net.

```
void zOID_setNetFromNetBus(
    OID* oid,
    const OID* netBusOid,
    Net* net
);
```

zOID_setInstPin

set oid to inst pin.

```
void zOID_setInstPin(
    OID* oid,
    Module* mod,
    Inst** path,
    int plen,
    Inst* inst,
    int pinNo
);
```

zOID_setInst

set oid to inst.

```
void zOID_setInst(
    OID* oid,
    Module* mod,
    Inst** path,
    int plen,
    Inst* inst
);
```

zOID_setInstFromPin

set oid to inst from pin.

```
void zOID_setInstFromPin(
    OID* oid,
    const OID* pinOid
);
```

zOID_pinInstName

return name of inst.

```
const char* zOID_pinInstName(
    const OID* oid
);
```

zOID_pinInstPath

get the inst path string.

```
const char* zOID_pinInstPath(
    const OID* oid,
    DStr* buf,
    char delim
);
```

zOID_pinInst

return inst.

```
Inst* zOID_pinInst(
    const OID* oid
);
```

zOID_pinInstRef

return referenced cell of inst.

```
Cell* zOID_pinInstRef(
    const OID* oid
);
```

zOID_pinName

return name of pin.

```
const char* zOID_pinName(
    const OID* oid
);
```

zOID_pinNo

return number of pin.

```
int zOID_pinNo(
    const OID* oid
);
```

zOID_portName

return name of port.

```
const char* zOID_portName(
    const OID* oid
);
```

zOID_portDir

return dir of port.

```
PortFlags zOID_portDir(
    const OID* oid
);
```

zOID_setNetFromPin

set oid to net, get path from pinOid.

```
void zOID_setNetFromPin(
    OID* oid,
    const OID* pinOid,
    Net* net
);
```

The ZDB Flat View Manager

Data Types for the Flat Tree

Supported Node Types

```
typedef unsigned char FNodeType;
```

Supported Flat Flags

```
typedef unsigned char FNodeFlags;
#define FNodeFNone    0x00
#define FNodeFOrange  0x01
#define FNodeFCyan    0x02
#define FNodeFBlack   0x04
#define FNodeFWhite   0x08
#define FNodeFMagenta 0x40
#define FNodeFTarget  0x80

#define FNodeFVisited 0x10
#define FNodeFAll     0xff
```

Flat Attribute Structure

```
typedef struct FAttr {
    const char* name;
    const char* value;
    int         nlen;   /* length of name */
} FAttr;
```

zFlatFree

Delete complete Top Node incl. tree from module defined by the given OID. If OID == NULL then all flat views are deleted.

```
zBool zFlatFree(
    DB*        db,
    const OID* oid
);
```

zFlatFreeByModule

Delete complete Top Node incl. tree from module.

```
void zFlatFreeByModule(
    DB*     db,
    Module* mod
);
```

zFlatClearCache

Clear the flat tree cache.

```
void zFlatClearCache(
    DB* db
);
```

zFlatCompress

Compress complete flat tree of module defined by given OID.

```
zBool zFlatCompress(
    DB*        db,
    const OID* oid
);
```

zFlatValidate

Validate the flat tree.

```
zBool zFlatValidate(
    DB*        db,
    const OID* oid,
    zBool      pedantic
);
```

zFlatWrite

Process nodes using zFlatWriteCB to write using OIDs. Starting from module defined by the given OID. If OID == NULL then all flat views are written out. For each hicol, flags, attr which is set in the tree a function from the given callbacks is called.

```
struct zWriteInfo;

struct zFlatWriteCB {
    void (*writeHicol)(
        struct zWriteInfo* writeInfo,
        const char*        oid,
        HiCol*             hicol
    );
    void (*writeFlags)(
        struct zWriteInfo* writeInfo,
        const char*        oid,
        FNodeFlags         flags
    );
    void (*writeAttrs)(
        struct zWriteInfo* writeInfo,
        const char*        oid,
        FAttr*             attrs,
        int                c
    );
    void (*writeOOMRs)(
        struct zWriteInfo* writeInfo,
        const char*         pinOid,
        const char*         netOid
    );
    void (*writeId)(
        struct zWriteInfo* writeInfo,
        const char*        oid,
        zUInt64            id
    );
};

zBool zFlatWrite(
    DB*                 db,
    const OID*          oid,
    struct zWriteInfo*  writeInfo,
    struct zFlatWriteCB* callbacks
);
```

zFlatPreprocessTree

write hier tree to ascii file. create fnode→uniqNo mapping in populateInfo create module→hierNodes list mapping in populateInfo.

```
void zFlatPreprocessTree(
    DB* db,
    struct zWriteInfo* writeInfo
);
```

zFlatWriteTree

write hier trees to ascii file.

```
void zFlatWriteTree(
    DB* db,
    struct zWriteInfo* writeInfo
);
```

zFlatWriteModuleChildNodes

Write flat data for given module into ascii file.

```
struct FNode;
void zFlatWriteModuleChildNodes(
    DB* db,
    struct zWriteInfo* writeInfo,
    Module* mod
);
```

zFlatDump

Dump flat tree structures starting from module defined by the given OID into the given file pointer. If OID == NULL then all flat views are dumped.

```
zBool zFlatDump(
    DB*        db,
    const OID* oid,
    IOWriter*  wrt
);
```

zFlatDumpFile

Dump flat tree structures starting from module defined by the given OID into the given file. If OID == NULL then all flat views are dumped.

```
zBool zFlatDumpFile(
    DB*         db,
    const OID*  oid,
    const char* fname,
    IOType      type
);
```

zFlatDeleteZombies

Executes deleteZombie for each node in the flat tree.

```
zBool zFlatDeleteZombies(
    DB*     db,
    Module* topMod
);
```

FlatForeachCallback

The FlatForeachCallback (callback functions) are called for each visited object (the given "ctx" is passed to the callback function). The FlatForeachCallback expect to return 0 for ok and continue, 1 for break, 2 for return and -1 on error.

```
typedef int FlatForeachCallback(
     void*      ctx,
     const OID*
);
typedef int FlatForeachCallback2(
    void*       ctx,
    const OID*,
    HiCol*,
    const char* attrVal

);
```

zFlatForeachSignal

Traverse the design hierarchy tree top down and call the given callback function for each signal.

```
int zFlatForeachSignal(
    DB*                  db,
    const OID*           top,
    zBool                skipGlobal,
    zBool                skipConst,
    zBool                autoPopulate,
    FlatForeachCallback* foreachCB,
    void*                foreachCtx
);
```

zFlatForeachNet

Call the given callback function for each net of a signal.

```
int zFlatForeachNet(
    DB*                 db,
    const OID*          netOID,
    zBool               skipGlobal,
    zBool               skipConst,
    zBool               autoPopulate,
    FlatForeachCallback* foreachCB,
    void*               foreachCtx
);
```

zFlatForeachPin

Call the given callback function for each pin or port connected to the given signal.

```
int zFlatForeachPin(
    DB*                 db,
    const OID*          netOID,
    zBool               addHier,
    zBool               stopHier,
    zBool               skipConst,
    zBool               autoPopulate,
    FlatForeachCallback* foreachCB,
    void*               foreachCtx
);
```

zFlatForeachNetSeg

Call the given callback function for each net segment between the two given pin/port OIDs.

```
int zFlatForeachNetSeg(
    DB*                 db,
    const OID*          startOID,
    const OID*          endOID,
    zBool               skipConst,
    zBool               autoPopulate,
    FlatForeachCallback* foreachCB,
    void*               foreachCtx
);
```

zFlatForeachObj

Call the given callback function for each node in the flat tree.

```
int zFlatForeachObj(
    DB*                   db,
    const OID*            topOid,
    FlatForeachCallback2* foreachCB,
    void*                 foreachCtx,
    FNodeFlags            flagged,
    const char*           attrName,
    zBool                 hilight,
    zBool                 skipInvalid,
    zBool                 autoPopulate
);
```

zFlatForeachInst

Traverse the hierarchy tree rooted at given module OID and call the given callback function for each instance of a cell with the given flag set. If flag == 0, then traverse all instances of all Cells. Return same as foreachCB: 0 (ok), 1 (break), 2 (return), -1 (error). or -2 (usage error, msg in zLastErrorMsg) If once == zTrue, then report only one inst of each referenced cell.

```
int zFlatForeachInst(
    DB*                   db,
    OID                   oid,
    CellFlags             flag,
    zBool                 once,
    zBool                 autoPopulate,
    FlatForeachCallback*  foreachCB,
    void*                 foreachCtx
);
```

zFlatSignalOf

Get the signal of a given net.

```
zBool zFlatSignalOf(
    DB*        db,
    const OID* net,
    zBool      skipGlobal,
    zBool      skipConst,
    zBool      autoPopulate,
    OID*       result
);
```

zFlatCountSignal

Count the number of signals.

```
zLong zFlatCountSignal(
    DB*        db,
    const OID* top,
    zBool      autoPopulate
);
```

zFlatCountNet

Count the number of nets of a signal.

```
zLong zFlatCountNet(
    DB*        db,
    const OID* netOID,
    zBool      autoPopulate
);
```

zFlatCountPin

Count the number of pins and ports connected to the given signal.

```
zLong zFlatCountPin(
    DB*        db,
    const OID* netOID,
    zBool      addHier,
    zBool      stopHier,
    zBool      autoPopulate
);
```

zFlatCountNetSeg

Count the number of net segments between the two given pin/port OIDs.

```
zLong zFlatCountNetSeg(
    DB*        db,
    const OID* startOID,
    const OID* endOID,
    zBool      autoPopulate
);
```

zFlatAddAttrs

Add a new attribute to the given OID. If 'propagate' is true, all nets of signal are processed. For netbuses the first member is followed as long as there are 1-1 connection through the hierarchy.

```
zBool zFlatAddAttrs(
    DB*         db,
    const OID*  oid,
    const FAttr attr[],
    int         len,
    zBool       propagate
);
```

zFlatSetAttrs

Same as zFlatAddAttr but attributes with the same name are overwritten.

```
zBool zFlatSetAttrs(
    DB*         db,
    const OID*  oid,
    const FAttr attr[],
    int         len,
    zBool       propagate
);
```

zFlatGetAttrs

Get attributes into attr until a maximum of len elements.

```
int zFlatGetAttrs(
    DB*        db,
    const OID* oid,
    FAttr      attr[],
    int        len
);
```

zFlatGetAttrsSorted

Get attributes into attr until a maximum of len elements. Sorted by name.

```
int zFlatGetAttrsSorted(
    DB*        db,
    const OID* oid,
    FAttr      attr[],
    int        len
);
```

zFlatDelAttrs

Delete attribute at the given OID, if name is NULL delete all attribute of given OID.

```
zBool zFlatDelAttrs(
    DB*          db,
    const OID*   oid,
    const char** names
);
```

zFlatDelAttrsAll

Recursively traverses the whole tree and delete attributes with given names.

```
zBool zFlatDelAttrsAll(
    DB*          db,
    const OID*,
    const char** name
);
```

zFlatSearchInstAttrValue

Get flat inst attribute value.

```
const char* zFlatSearchInstAttrValue(
    DB*         db,
    Inst*       inst,
    const char* attrName,
    const char* topinfo,
    const char* path,
    zBool       exact
);
```

zFlatSearchPortAttrValue

Get flat port attribute value.

```
const char* zFlatSearchPortAttrValue(
    DB*         db,
    Port*       port,
    const char* attrName,
    const char* topinfo,
    const char* path,
    zBool       exact
);
```

zFlatSearchPortBusAttrValue

Get flat portBus attribute value.

```
const char* zFlatSearchPortBusAttrValue(
    DB*         db,
    PortBus*    portBus,
    const char* attrName,
    const char* topinfo,
    const char* path,
    zBool       exact
);
```

zFlatSearchPinAttrValue

Get flat pin attribute value.

```
const char* zFlatSearchPinAttrValue(
    DB*         db,
    Inst*       inst,
    int         pinNo,
    const char* attrName,
    const char* topinfo,
    const char* path,
    zBool       exact
);
```

zFlatSearchPinBusAttrValue

Get flat pinBus attribute value.

```
const char* zFlatSearchPinBusAttrValue(
    DB*         db,
    Inst*       inst,
    int         pinBusNo,
    const char* attrName,
    const char* topinfo,
    const char* path,
    zBool       exact
);
```

zFlatSearchSignalAttrValue

Get flat signal attribute value.

```
const char* zFlatSearchSignalAttrValue(
    DB*         db,
    Net*        signal,
    const char* attrName,
    const char* topinfo,
    const char* path,
    zBool       exact
);
```

zFlatSearchNetAttrValue

Get flat net attribute value.

```
const char* zFlatSearchNetAttrValue(
    DB*         db,
    Net*        net,
    const char* attrName,
    const char* topinfo,
    const char* path,
    zBool       exact
);
```

zFlatSearchNetBusAttrValue

Get flat netBus attribute value.

```
const char* zFlatSearchNetBusAttrValue(
    DB*         db,
    NetBus*     netBus,
    const char* attrName,
    const char* topinfo,
    const char* path,
    zBool       exact
);
```

zFlatSearchInstHilight

Get flat inst hilight info.

```
zBool zFlatSearchInstHilight(
    DB*         db,
    const char* topInfo,
    const char* path,
    Inst*       inst,
    int*        color,
    int*        flags
);
```

zFlatSearchPortHilight

Get flat port hilight info.

```
zBool zFlatSearchPortHilight(
    DB*         db,
    const char* topInfo,
    const char* path,
    Port*       port,
    int*        color
);
```

zFlatSearchPortBusHilight

Get flat portBus hilight info.

```
zBool zFlatSearchPortBusHilight(
    DB*         db,
    const char* topInfo,
    const char* path,
    PortBus*    portBus,
    int*        color
);
```

zFlatSearchPinHilight

Get flat pin hilight info.

```
zBool zFlatSearchPinHilight(
    DB*         db,
    const char* topInfo,
    const char* path,
    Inst*       inst,
    int         pinNo,
    int*        color
);
```

zFlatSearchPinBusHilight

Get flat pinBus hilight info.

```
zBool zFlatSearchPinBusHilight(
    DB*         db,
    const char* topInfo,
    const char* path,
    Inst*       inst,
    int         pinBusNo,
    int*        color
);
```

SegmentHi

Struct to store flat segment hilight.

```
struct SegmentHi {
    struct {
        Inst*       inst;
        const char* pinName;
        zBool       isPort;
    } con[2];
    int color;
};
```

zFlatSearchNetHilight

Get flat net hilight info.

```
zBool zFlatSearchNetHilight(
    DB*                 db,
    Module*             mod,
    const char*         topInfo,
    const char*         path,
    Net*                net,
    int*                color,
    struct SegmentHi** segmentArray
);
```

zFlatSearchNetBusHilight

Get flat netBus hilight info.

```
zBool zFlatSearchNetBusHilight(
    DB*            db,
    const char*    topInfo,
    const char*    path,
    NetBus*        netBus,
    int*           color
);
```

zFlatNodeMakeInst

Make a flat node for an instance.

```
struct FNode* zFlatNodeMakeInst(
    DB*           db,
    Module*       root,
    struct FNode* node,
    Inst*         inst
);
```

zFlatNodeMakeNet

Make a flat node for a net.

```
struct FNode* zFlatNodeMakeNet(
    DB*           db,
    Module*       root,
    struct FNode* node,
    Net*          net
);
```

zFlatNodeMakeSignal

Make a flat node for a signal.

```
struct FNode* zFlatNodeMakeSignal(
    DB*           db,
    Module*       root,
    struct FNode* node,
    Net*          net
);
```

zFlatNodeAddAttrs

Add attributes to a given node.

```
zBool zFlatNodeAddAttrs(
    DB*              db,
    struct FTopNode* topNode,
    struct FNode*    node,
    const FAttr      attr[],
    int              len
);
```

zFlatNodeSetAttrs

Set attributes at a given node.

```
zBool zFlatNodeSetAttrs(
    DB*              db,
    struct FTopNode* topNode,
    struct FNode*    node,
    const FAttr      attr[],
    int              len
);
```

zFlatNodeGetAttrs

Get attributes at a given node.

```
int zFlatNodeGetAttrs(
    DB*              db,
    struct FTopNode* topNode,
    struct FNode*    node,
    FAttr            attr[],
    int              len
);
```

zFlatNodeDelAttrs

Delete attributes at a given node.

```
zBool zFlatNodeDelAttrs(
    DB*              db,
    struct FTopNode* topNode,
    struct FNode*    node,
    const char**     names
);
```

zFlatGetAttrsMerged

Get merged flat and module based attributes at the given node.

```
int zFlatGetAttrsMerged(
    DB*        db,
    const OID* oid,
    FAttr      attr[],
    int        len
);
```

zFlatGetAttrsMergedSorted

Get merged flat and module based attributes at the given node, sorted by name.

```
int zFlatGetAttrsMergedSorted(
    DB*        db,
    const OID* oid,
    FAttr      attr[],
    int        len
);
```

zFlatNodeGetAttrValue

Get attribute value at a given node.

```
const char* zFlatNodeGetAttrValue(
    DB*           db,
    Module*       root,
    struct FNode* node,
    const char*   name
);
```

zFlatGetAttrValue

Get flat attribute value of the given OID.

```
const char* zFlatGetAttrValue(
    DB*         db,
    const OID*  oid,
    const char* name
);
```

zFlatGetAttrValueMerged

Get merged flat and module based attributes at the given OID.

```
const char* zFlatGetAttrValueMerged(
    DB*         db,
    const OID*  oid,
    const char* name
);
```

zFlatStr2Flag

Convert a string into a flat flag.

```
FNodeFlags zFlatStr2Flag(
    const char* str
);
```

zFlatFlag2Strs

Convert flags into strings. Return the number flags.

```
int zFlatFlag2Strs(
    FNodeFlags  flags,
    const char* strs[10]
);
```

zFlatGetFlags

Return a bit vector of flags.

```
FNodeFlags zFlatGetFlags(
    DB*        db,
    const OID* oid
);
```

zFlatGet5Flags

Like zFlatGetFlags above, but if called with a pin OID the search at inst+pin+pinBus+port+portBus and if called with oid=port then also search port+portBus.

```
FNodeFlags zFlatGet5Flags(
    DB*        db,
    const OID* oid
);

#ifdef COMPARE_CONE
FNodeFlags zFlatGet5FlagsX(
    DB*        db,
    const OID* oid,
    zBool*     fromPort
);
#endif
```

zFlatIsFlag

Set zTrue or zFalse depending on the flag state to the given 'isSet' zBool pointer and returns zTrue on success or zFalse on error (zLastErrorMsg is set).

```
zBool zFlatIsFlag(
    DB*        db,
    const OID* oid,
    FNodeFlags flag,
    zBool*     isSet
);
```

zFlatSetFlag

Store the old flag values through the "prevFlags" argument (if not NULL) and sets the given flags. Return zTrue on success or zFalse on error (zLastErrorMsg is set).

```
zBool zFlatSetFlag(
    DB*         db,
    const OID*  oid,
    FNodeFlags  flag,
    FNodeFlags* prevFlags
);
```

zFlatClrFlag

Like zFlatSetFlag above but also clear the given flag.

```
zBool zFlatClrFlag(
    DB*         db,
    const OID*  oid,
    FNodeFlags  flag,
    FNodeFlags* prevFlags
);
```

zFlatClrFlagAll

Like zFlatClrFlag but clear flags at all flat-tree nodes.

```
zBool zFlatClrFlagAll(
    DB*        db,
    const OID* oid,
    FNodeFlags flag
);
```

zFlatGetFlagOIDs

Return q list of all oids which have the given flag set.

```
zBool zFlatGetFlagOIDs(
    DB*        db,
    const OID* top,
    FNodeFlags flag,
    OID**      resultList
);
```

zFlatHilightSet

Hilight Function to set flat highlight information.

```
zBool zFlatHilightSet(
    DB*        db,
    const OID* oid,
    int        color,
    int        permanent
);
```

zFlatHilightGet

Hilight Function to get flat highlight information.

```
zBool zFlatHilightGet(
    DB*        db,
    const OID* oid,
    int*       color,
    int        permanent
);
```

zFlatHilightGetMerged

Hilight Function to get flat and module based highlight information.

```
zBool zFlatHilightGetMerged(
    DB*        db,
    const OID* oid,
    int*       color,
    int        permanent
);
```

zFlatHilightGetSignalOrNet

Get hicolor of signal or first associated net which has a color.

```
zBool zFlatHilightGetSignalOrNet(
    DB*        db,
    const OID* oid,
    int*       color,
    int        permanent
);
```

zFlatHilightDelAll

Hilight Function to delete flat highlight information.

```
zBool zFlatHilightDelAll(
    DB*        db,
    const OID* top,
    int        color,
    int        permanent
);
```

OOMPinRef

Struct to store flat connectivity (OOMR: Out-Of-Module References).

```
struct OOMPinRef {
    const char* hinstname;
    int         pinNo;
};
```

zFlatAddOOMR

Add an OOMR connection.

```
zBool zFlatAddOOMR(
    DB*        db,
    const OID* pin,
    const OID* net
);
```

zFlatDelOOMR

Del an OOMR connection.

```
zBool zFlatDelOOMR(
    DB*        db,
    const OID* pin
);
```

zFlatGetOOMR

Get OOMR connections.

```
zBool zFlatGetOOMR(
    DB*        db,
    const OID* oid,
    OID**      resultListPtr
);
```

FlatForeachOOMRCallback

The FlatForeachOOMRCallback (callback functions) are called for each oomr entry pair (the given "ctx" is passed to the callback function). The FlatForeachOOMRCallback expect to return 0 for ok and continue, 1 for break, 2 for return and -1 on error.

```
typedef int FlatForeachOOMRCallback(
     void*      ctx,
     const OID* pinOid,
     const OID* netOid
);
```

zFlatForeachOOMR

Iterate over all flat oomr entries.

```
int zFlatForeachOOMR(
    DB*                       db,
    const OID*                topMod,
    FlatForeachOOMRCallback* foreachCB,
    void*                     foreachCtx
);
```

zFlatGetOOMPinList

From a given hierarchical net return: (a) top module (b) list of connected instance pins (as a list of OOMPinRef) path is separated by db→internalHierSep. Return false if the hierarchical net has non stored pin connections.

```
zBool zFlatGetOOMPinList(
    DB*                db,
    const char*        topinfo,
    const char*        path,
    const Net*         net,
    Module**           topPtr,
    struct OOMPinRef** oomrListPtr,
    zBool*             allocated
);
```

zFlatGetOOMNet

From a given vdiHinst + pinNo return the hierarchical name to the connected net (db.hiersep separated string). Return NULL if the given pin is unconnected.

```
const char* zFlatGetOOMNet(
    DB*         db,
    const char* topinfo,
    const char* path,
    const Inst* inst,
    int         pinNo
);
```

zFlatMoveInsts

Update Data Tree after zOperHierStart/zOperHierAdd

```
int zFlatMoveInsts(
    DB*         db,
    Module*     top,
    const char* iname,
    Module*     from,
    Module*     to
);
```

zFlatUpdateAddHier

Update Data Tree after zOper*Hier has added/removed one level of Hierarchy.

```
zBool zFlatUpdateAddHier(
    DB*       db,
    Module*   parent,
    InstFlags movedIFlag,
    NetFlags  movedNFlag,
    NetFlags  crossedNFlag,
    Inst*     newInst
);
```

zFlatUpdateRmHier

```
zBool zFlatUpdateRmHier(
    DB*         db,
    Module*     parent,
    Inst*       rmInst,
    NetFlags    movedNFlag,
    NetFlags    crossedNFlag,
    const char* prefix
);
```

zFlatCreateHier

Update Data Tree after zOperCreateHier.

```
zBool zFlatCreateHier(
    DB*     db,
    Module* topMod,
    char    hsep
);
```

zFlatUpdateSortPorts

Update Data Tree after zOperPortSorts.

```
zBool zFlatUpdateSortPorts(
    DB*       db,
    Module*   top,
    CellFlags flags
);
```

zFlatTreeMemoryUsage

Return total flat tree memory usage.

```
zULong zFlatTreeMemoryUsage(
    struct FTopNode* top,
    unsigned*        cnt
);
```

zFlatTreeStrmemUsage

Return total string memory usage.

```
zULong zFlatTreeStrmemUsage(
    struct FTopNode* top,
    unsigned*        cnt
);
```

zFlatMapAttr

Map instance attributes in tree.

```
struct Symlib;
struct Cell2SpiceMapping;
zBool zFlatMapAttr(
    DB*                       db,
    struct Cell2SpiceMapping* mapTable
);
```

zFlatAddAttrName

Add attribute name used for packAttrs format.

```
void zFlatAddAttrName(
    DB*              db,
    struct FTopNode* top,
    const char*      name,
    int              nlen
);
```

zFlatNewTopNode

Create a top node starting at module root. Preallocate attrNameCnt attribute name entries.

```
struct FTopNode* zFlatNewTopNode(
    DB*     db,
    Module* root,
    int     attrNameCnt
);
```

zFlatMakeTopNode

Create a new top node starting at module root, if it no already exists.

```
struct FTopNode* zFlatMakeTopNode(
    DB*     db,
    Module* root,
    int     attrNameCnt
);
```

zFlatGetRootNode

Get root of flat data node from topNode. preallocate subListCnt sub node entries.

```
struct FNode* zFlatGetRootNode(
    DB*             db,
    struct FTopNode* top,
    int             subListCnt
);
```

zFlatNewInstNode

Create a new flat inst node below the given parent node. preallocate subListCnt sub node entries. The given name should match to be hierarchical object. If it does not match it can not be accessed later and will be complained by validating with 'pedantic'.

```
struct FNode* zFlatNewInstNode(
    DB*             db,
    struct FTopNode* top,
    struct FNode*    parent,
    const char*      name,
    int             subListCnt
);
```

zFlatMakeInstNode

Return exiting inst node or create a new flat inst node below the given parent node. The given

name should match to be hierarchical object. If it does not match it can not be accessed later and will be complained by validating with 'pedantic'.

```
struct FNode* zFlatMakeInstNode(
    DB*              db,
    struct FTopNode* top,
    struct FNode*    parent,
    const char*      name
);
```

zFlatMakeInstNodes

Create a new flat inst node path below the given parent node. All nodes on the given NULL terminated path array are created if the do not already exist. The given name should match to be hierarchical object. If it does not match it can not be accessed later and will be complained by validating with 'pedantic'.

```
struct FNode* zFlatMakeInstNodes(
    DB*              db,
    struct FTopNode* top,
    struct FNode*    parent,
    const char**     path
);
```

zFlatNewNetNode

Create a new flat net node below the given parent node. The given name should match to the hierarchical object. If it does not match it can not be accessed later and will be complained by validating with 'pedantic'.

```
struct FNode* zFlatNewNetNode(
    DB*              db,
    struct FTopNode* top,
    struct FNode*    parent,
    const char*      name
);
```

zFlatMakeNetNode

Search/create a flat net node below the given parent node. The given name should match to the hierarchical object. If it does not match it can not be accessed later and will be complained by validating with 'pedantic'.

```
struct FNode* zFlatMakeNetNode(
    DB*             db,
    struct FTopNode* top,
    struct FNode*    parent,
    const char*      name
);
```

zFlatNewNetOOMRNode

Create a new flat NetOOMR node below the given parent node. The given name should match to the hierarchical object. If it does not match it can not be accessed later and will be complained by validating with 'pedantic'.

```
struct FNode* zFlatNewNetOOMRNode(
    DB*             db,
    struct FTopNode* top,
    struct FNode*    parent,
    const char*      name
);
```

zFlatNewNetSegNode

Create a new flat NetSeg node below the given parent node. The given name should match to the hierarchical object. If it does not match it can not be accessed later and will be complained by validating with 'pedantic'.

```
struct FNode* zFlatNewNetSegNode(
    DB*             db,
    struct FTopNode* top,
    struct FNode*    parent,
    const char*      name,
    int              pin0,
    int              pin1
);
```

zFlatMakeNetSegNode

Search/create a flat NetSeg node below the given parent node. The given name should match to the hierarchical object. If it does not match it can not be accessed later and will be complained by validating with 'pedantic'.

```
struct FNode* zFlatMakeNetSegNode(
    DB*             db,
    struct FTopNode* top,
    struct FNode*    parent,
    const char*      name,
    int              pin0,
    int              pin1
);
```

zFlatNewPortNode

Create a new flat port node below the given parent node. The given name should match to the hierarchical object. If it does not match it can not be accessed later and will be complained by validating with 'pedantic'.

```
struct FNode* zFlatNewPortNode(
    DB*             db,
    struct FTopNode* top,
    struct FNode*    parent,
    const char*      name
);
```

zFlatNewPortBusNode

Create a new flat portBus node below the given parent node. The given name should match to the hierarchical object. If it does not match it can not be accessed later and will be complained by validating with 'pedantic'.

```
struct FNode* zFlatNewPortBusNode(
    DB*             db,
    struct FTopNode* top,
    struct FNode*    parent,
    const char*      name
);
```

zFlatNewPinNode

Create a new flat net node below the given parent node. The given name should match to the hierarchical object. If it does not match it can not be accessed later and will be complained by validating with 'pedantic'.

```
struct FNode* zFlatNewPinNode(
    DB*             db,
    struct FTopNode* top,
    struct FNode*    parent,
    const char*      name
);
```

zFlatNewPinBusNode

Create a new flat net node below the given parent node. The given name should match to the hierarchical object. If it does not match it can not be accessed later and will be complained by validating with 'pedantic'.

```
struct FNode* zFlatNewPinBusNode(
    DB*             db,
    struct FTopNode* top,
    struct FNode*    parent,
    const char*      name
);
```

zFlatNewNetIdNode

Create a new flat net id node below the given parent node. The given name should match to the hierarchical object. If it does not match it can not be accessed later and will be complained by validating with 'pedantic'.

```
struct FNode* zFlatNewNetIdNode(
    DB*             db,
    struct FTopNode* top,
    struct FNode*    parent,
    const char*      name
);
```

zFlatNewNetBusIdNode

Create a new flat net bus id node below the given parent node. The given name should match to the hierarchical object. If it does not match it can not be accessed later and will be complained by validating with 'pedantic'.

```
struct FNode* zFlatNewNetBusIdNode(
    DB*             db,
    struct FTopNode* top,
    struct FNode*    parent,
    const char*      name
);
```

zFlatNewSignalIdNode

Create a new flat signal id node below the given parent node. The given name should match to the hierarchical object. If it does not match it can not be accessed later and will be complained by validating with 'pedantic'.

```
struct FNode* zFlatNewSignalIdNode(
    DB*             db,
    struct FTopNode* top,
    struct FNode*    parent,
    const char*      name
);
```

zFlatNewNamedNode

Create a new flat node below the given parent node. Internal use only. The given name should match to the hierarchical object. If it does not match it can not be accessed later and will be complained by validating with 'pedantic'.

```
struct FNode* zFlatNewNamedNode(
    DB*             db,
    struct FTopNode* top,
    struct FNode*    parent,
    FNodeType        type,
    const char*      name,
    int              subListCnt
);
```

zFlatNewIndexedNode

Create new flat node below the given parent node. Internal use only. The given index should match the hierarchical object. If it does not match it can not be accessed later and will be complained by validating with 'pedantic'.

```
struct FNode* zFlatNewIndexedNode(
    DB*           db,
    struct FNode* parent,
    FNodeType     type,
    int           index
);
```

zFlatNewPinOOMRNode

Create new flat node below the given parent node. Internal use only. The given index should match the hierarchical object. If it does not match it can not be accessed later and will be complained by validating with 'pedantic'.

```
struct FNode* zFlatNewPinOOMRNode(
    DB*              db,
    struct FTopNode* top,
    struct FNode*    parent,
    int              index,
    const char*      hnetname
);
```

zFlatSetFlags

Set flag for the object represented by the flat node.

```
void zFlatSetFlags(
    struct FNode* node,
    FNodeFlags    flags
);
```

zFlatSetHicol

Set normal and permanent highlight color for the object represented by the flat node.

```
void zFlatSetHicol(
    struct FNode* node,
    int           norm,
    int           perm
);
```

zFlatSetPackedAttrs

Set attributes of given node in packedAttrs format. Internal use only.

```
void zFlatSetPackedAttrs(
    DB*              db,
    struct FTopNode* top,
    struct FNode*    node,
    const char*      packedAttrs
);
```

zFlatSetPackedAttrsLen

Set attributes of given node in packedAttrs format. For speed, if len of packedAttrs is already known Internal use only.

```
void zFlatSetPackedAttrsLen(
    DB*              db,
    struct FTopNode* top,
    struct FNode*    node,
    const char*      packedAttrs,
    int              len

);
```

zFlatNewPackedAttrNode

Add attributes of given node in packedAttrs format. Internal use only.

```
struct FNode* zFlatNewPackedAttrNode(
    DB*              db,
    struct FTopNode* top,
    struct FNode*    parent,
    const char*      packedAttrs
);
```

zFlatNewPackedAttrNodeLen

Add attributes of given node in packedAttrs format. Internal use only.

```
struct FNode* zFlatNewPackedAttrNodeLen(
    DB*              db,
    struct FTopNode* top,
    struct FNode*    parent,
    const char*      packedAttrs,
    int              len
);
```

zFlatSetNetSeg

Set net segment pin indices Internal use only.

```
void zFlatSetNetSeg(
    struct FNode* node,
    int           pin0,
    int           pin1
);
```

zFlatSetPinOOMR

Set oomr info for given node. Internal use only.

```
void zFlatSetPinOOMR(
    DB*              db,
    struct FTopNode* top,
    struct FNode*    node,
    const char*      hnetname
);
```

zFlatAddNetOOMR

Set oomr info for given node. Internal use only.

```
void zFlatAddNetOOMR(
    DB*              db,
    struct FTopNode* top,
    struct FNode*    node,
    const char*      hinstname,
    int              pinNo
);
```

zFlatNodeSetId

Set id for given node. Internal use only.

```
void zFlatNodeSetId(
    struct FNode*    node,
    zUInt64          id
);
```

zFlatClone

Clone flat info to another database 'dst'. If at least one of 'keepCell','keepInst','keepPos' is not 0 only flagged objects are cloned and invalid entries are ignored.

```
zBool zFlatClone(
    DB*          src,
    DB*          dst,
    CellFlags    keepCell,
    InstFlags    keepInst,
    NetFlags     keepNet,
    PortFlags    keepPort,
    const char** prefixPath,
    int          prefixPathLen
);
```

zFlatMerge

Merge flat info to another database 'dst'.

```
zBool zFlatMerge(
    DB*          src,
    DB*          dst,
    const char** prefixPath,
    int          prefixPathLen
);
```

zFlatSetId

Set a new id to the given OID.

```
zBool zFlatSetId(
    DB*         db,
    const OID*  oid,
    zUInt64     id
);
```

zFlatGetId

Get a id to the given OID.

```
zBool zFlatGetId(
    DB*         db,
    const OID*  oid,
    zUInt64*    id
);
```

The Symlib File Scanner

Create Symlib Entries in the Database

zSymlibCreate

Create a symlib file.

```
zBool zSymlibCreate(
    DB*         db,
    const char* filename
);
```

zSymlibScan

In normal mode, use symTable to lookup symbols (fast). In glob mode, loop over all cells and try to

sequential match sym entries (slow).

```
zBool zSymlibScan(
    DB*   db,
    zBool force
);
```

zSymlibCopy

Create primitives for each symbol. If withSpiceSymbols is zTrue additional primitives for spice mappings are created (if they do not contain pattern names with * and ?(.

```
zBool zSymlibCopy(
    DB*   db,
    zBool withSpiceSymbols
);
```

zSymlibCheck

Check the value of the given @symbol attribute.

```
zBool zSymlibCheck(
    DB*         db,
    Cell*       cell,
    const char* value,
    zBool       icase
);
```

zSymlibAdd

Parse whole file into db. Store header infos in Symlib struct. Symbols in symList, symio info in symioList, spice info in spiceList. If index mode store comments in commentList if there is no symlib allocate it.

```
zBool zSymlibAdd(
    DB*         db,
    const char* fname,
    zBool       withComments
);
```

zSymlibParseStr

Like zSymlibAdd but read the input from a string.

```
zBool zSymlibParseStr(
    DB*         db,
    const char* str,
    zBool       withComments
);
```

zSymlibParseDB

Look for @symbol attributes in DB at all cells and store them in Symlib struct. If the value of the Boolean parameter 'fromFunc' is true and no @symbol attribute was found at a cell but the cell has a function defined then also add a symbol entry for based on the function to the Symlib struct.

```
zBool zSymlibParseDB(
    DB*   db,
    zBool fromFunc
);
```

zSymlibClear

Delete symlib structure. Clear symdef table and all @sym and @symbol attributes, the @libName attribute used for skillexport will be removed too (slow, because all objects need to be checked).

```
zBool zSymlibClear(
    DB* db
);
```

zSymlibInfo

Get header infos out of first symlib. libName is returned in zLastErrorMsg Return zFalse if there is no symlib.

```
zBool zSymlibInfo(
    DB*      db,
    zUInt16* vers,
    zBool*   icase,
    zBool*   glob
);
```

zSymlibGitRevisions

List of revisions of all symlibs. need to be zArrayFree-d.

```
const char** zSymlibGitRevisions(
    DB*     db
);
```

zSymlibScanSymDef

```
zBool zSymlibScanSymDef(
    int    argc,
    char** argv,
    zBool  ic,
    int**  indexListPtr
);
```

zSymlibRemove

Remove all symlibs.

```
void zSymlibRemove(
    DB* db
);
```

zSymlibMapGet

Search spice mappings for the given cell.

```
struct Symlib;
struct Cell2SpiceMapping;
struct Sspice;
struct Sspice* zSymlibMapGet(
    struct Cell2SpiceMapping* mapTable,
    Cell*                     cell
);
```

zSymlibMapAttrFind

Find a given attr name in attrmapList.

```
const char* zSymlibMapAttrFind(
    struct Sspice* spice,
    const char*    name
);
```

zSymlibWrite

Write symlib file to ASCII format.

```
struct zWriteInfo;
void zSymlibWrite(
    struct zWriteInfo* writeInfo
);
```

zSymlibNew

Add symlib to DB.

```
struct Symlib* zSymlibNew(
    DB* db,
    const char* libname,
    double      version,
    const char* gitRevision,
    zBool       icase,
    zBool       glob,
    zBool       microsoft
);
```

zSymlibNewIO

```
struct Ssymio* zSymlibNewIO(
    DB*         db,
    struct Symlib* symlib,
    zFPos       start,
    const char* sname,
    char        type,
    const char* oname,
    const char* mname,
    const char* lname,
    const char* pname,
    const char* netset
);
```

zSymlibNewSpice

```
struct Sspice* zSymlibNewSpice(
    DB*         db,
    struct Symlib* symlib,
    const char* sname,
    const char* lname,
    const char* mname,
    const char* bulkattr,
    const char* bulktype,
    const char* prefix,
    zFPos       start,
    zBool       rename,
    zBool       icase
);
```

zSymlibNewBulk

```
void zSymlibNewSpiceBulk(
    DB*             db,
    struct Sspice* spice,
    const char*    bulkattr,
    const char*    bulktype
);
```

zSymlibNewAttrMap

```
void zSymlibNewAttrMap(
    DB*            db,
    struct Sspice* spice,
    const char*    mname,
    const char*    oname,
    const char*    type,
    zBool          constant
);
```

zSymlibNewComment

if symlib is NULL use last symlib. return zFalse if there is no symlib.

```
zBool zSymlibNewComment(
    DB*         db,
    struct Symlib* symlib,
    const char* comment
);
```

zSymlibNewSymbol

```
struct Ssym* zSymlibNewSymbol(
    DB*         db,
    struct Symlib* symlib,
    const char* sname,
    zFPos       start,
    int         argc
);
```

zSymlibFillHashTable

call after all symbols are created.

```
void zSymlibFillHashTable(
    DB* db,
    struct Symlib* symlib
);
```

zSymlibNewSymArg

```
void zSymlibNewSymArg(
    DB*          db,
    struct Symlib* symlib,
    struct Ssym* sym,
    int          i,
    const char*  arg
);
```

zSymlibNewSymDefinition

```
zBool zSymlibNewSymDefinition(
    DB* db,
    struct Symlib* symlib,
    struct Ssym* sym
);
```

zSymlibNewPortMap

```
void zSymlibNewPortMap(
    DB*            db,
    struct Sspice* spice,
    const char*    mname,
    const char*    oname
);
```

zSymlibNewPortType

```
void zSymlibNewPortType(
    DB*            db,
    struct Sspice* spice,
    const char*    oname,
    const char*    sigtype
);
```

zSymlibSetSymmapMode

```
void zSymlibSetSymmapMode(
    DB*            db,
    struct Sspice* spice,
    const char*    symmapMode
);
```

Cone Extraction

Types for the Cone Extraction

Extraction Direction

Direction of the Cone extraction.

```
enum ConeEdir {
    ConeEdirToOut = 0,
    ConeEdirToIn  = 1
};
```

Extraction Options

Target and extraction options.

```
#define ConeEFtargetIO        0x00001 /* Toplevel I/O ports                  */
#define ConeEFtargetDangle    0x00002 /* Dangling nets                       */
#define ConeEFtargetOpenPin   0x00004 /* Open pins                           */
#define ConeEFtargetOpenPort  0x00008 /* Open ports                          */
#define ConeEFtargetLoop      0x00010 /* Circuit loops                       */
#define ConeEFtargetPGNet     0x00020 /* Power/Ground nets (constant values) */
#define ConeEFtargetConstNet  0x00040 /* Constant value nets are target      */
#define ConeEFdontDive        0x00100 /* Don't leave hierarchy-level         */
#define ConeEFcheckArcs       0x00200 /* Consult zConeCalcArc()              */
#define ConeEFcacheArcs       0x00400 /* Cache arcs at Modules               */
#define ConeEFunknown2IO      0x00800 /* Pin-dir "unknown" is IO             */
#define ConeEFcoarse          0x01000 /* Coarse the result                   */
#define ConeEFreachable       0x02000 /* Only targets to resultList          */
#define ConeEFaddStartPin     0x04000 /* Add start pin to resultList         */
#define ConeEFemptyModAsPrim  0x08000 /* Treat empty modules as primitives   */
#define ConeEFflat            0x10000 /* Suppress hierarchical objs in result */
#define ConeEFcreateNetSeg    0x20000 /* Create net segments in result       */
#define ConeEFautoPopulate    0x40000 /* Populate missing module content     */
#define ConeEFprocessPGNets   0x80000 /* Continue tracing at PG nets         */
#define ConeEFfilterLogicalInvalid 0x100000 /* filter paths with constant    */
```

ResultPath

```
struct ResultPath {
    OID* oidList;       /* List of Segm of this path       */
    int  depth;         /* Number of logic levels          */
};

typedef int (*ConeCustomCB)(void* ctx, const OID*);
```

Cone Extraction Function

zConeExtract

Start the Cone Extraction.

```
zBool zConeExtract(
    const OID*          startList,              /* Extraction start list       */
    enum ConeEdir       direction,              /* Direction of the extraction */
    zBool               ignoreDir,              /* Ignore module port dir      */
    unsigned            flags,                  /* Extraction options          */
    CellFlags           targetFlaggedCell,      /* CellFxxx,... or 0           */
    InstFlags           targetFlaggedInst,      /* InstFxxx,... or 0           */
    PinFlags            targetFlaggedPin,       /* PortFxxx,... or 0           */
    PortFlags           targetFlaggedPort,      /* PortFxxx,... or 0           */
    NetFlags            targetFlaggedNet,       /* NetFxxx,...  or 0           */
    FNodeFlags          targetFlatFlagged,      /* FNodeFxxx,... or 0          */
    FNodeFlags          targetFlatNetNotFlagged,/*FNodeFxxx,... or 0          */
    const OID*          targetObj,              /* target object or NULL       */
    DB*                 db,                     /* DB root                     */
    const char**        targetCell[4],          /* 4 lists from -targetCell*   */
    const char**        excludeCell,            /* 1 list from -excludeCell    */
    CellFlags           excludeFlaggedCell,     /* CellFxxx,... or 0           */
    InstFlags           excludeFlaggedInst,     /* InstFxxx,... or 0           */
    PortFlags           excludeFlaggedPin,      /* PortFxxx,... or 0           */
    PortFlags           excludeFlaggedPort,     /* PortFxxx,... or 0           */
    NetFlags            excludeFlaggedNet,      /* NetFxxx,...  or 0           */
    FNodeFlags          excludeFlatFlagged,     /* FNodeFxxx,... or 0          */
    int                 limit,                  /* Limit levels, 0 = no limit  */
    int                 pathLimit,              /* Max # paths,  0 = no limit  */
    zBool               excludeFuncPort,        /* Exclude ports with special  */
                                                /* function (e.g. sel,clk)     */
    ConeCustomCB        customCB,               /* customize CB for pathList   */
    void*               customCtx,

    OID**               resultList,             /* RESULT: list of OIDs        */
    struct ResultPath** resultPathList          /* RESULT: list of ResultPaths */
);
```

zConeExtractShortest

Special Cone Extraction.

```
zBool zConeExtractShortest(
    const OID*          startList,              /* Extraction start list       */
    enum ConeEdir       direction,              /* Direction of the extraction */
    zBool               ignoreDir,              /* Ignore module port dir      */
    unsigned            flags,                  /* Extraction options          */
    CellFlags           targetFlaggedCell,      /* CellFxxx,... or 0           */
    InstFlags           targetFlaggedInst,      /* InstFxxx,... or 0           */
    PinFlags            targetFlaggedPin,       /* PortFxxx,... or 0           */
    PortFlags           targetFlaggedPort,      /* PortFxxx,... or 0           */
    NetFlags            targetFlaggedNet,       /* NetFxxx,...  or 0           */
    FNodeFlags          targetFlatFlagged,      /* FNodeFxxx,... or 0          */
    FNodeFlags          targetFlatNetNotFlagged,/*FNodeFxxx,... or 0          */
    const OID*          targetObj,              /* target object or NULL       */
    DB*                 db,                     /* DB root                     */
    const char**        targetCell[4],          /* 4 lists from -targetCell*   */
    const char**        excludeCell,            /* 1 list from -excludeCell    */
    CellFlags           excludeFlaggedCell,     /* CellFxxx,... or 0           */
    InstFlags           excludeFlaggedInst,     /* InstFxxx,... or 0           */
    PortFlags           excludeFlaggedPin,      /* PortFxxx,... or 0           */
    PortFlags           excludeFlaggedPort,     /* PortFxxx,... or 0           */
    NetFlags            excludeFlaggedNet,      /* NetFxxx,...  or 0           */
    FNodeFlags          excludeFlatFlagged,     /* FNodeFxxx,... or 0          */
    zBool               excludeFuncPort,        /* Exclude ports with special  */
                                                /* function (e.g. sel,clk)     */
    OID*                targetNets,             /* Special for shortestPath    */

    OID**               resultList,             /* RESULT: list of OIDs        */
    OID***              resultLists             /* RESULT: list of OIDs list   */
);
```

zConeToPG

Find paths to Power/Ground over tdevices starting at given pin.

```
zBool zConeToPG(
    DB*       db,
    PinRef*   startPin,
    zBool     opposite,
    int       depthLimit,
    int       connLimit,
    CellFlags excludeFlaggedCell,
    InstFlags excludeFlaggedInst,
    NetFlags  onlyFlaggedNet,
    PinRef**  resultList
);
```

Define Arcs

Set/get "Arcs" at Operators (Module with a defined function).

Arc Functions

zConepClearArc

Clear user defined arcs of all (cellName == NULL) or only the given cell.

```
zBool zConepClearArc(
    const char* cellname
);
```

zConepDefineArc

Add a user defined arc.

```
zBool zConepDefineArc(
    const char*  cellName,
    const char*  enterName,
    const char** leaveNameList
);
```

zConepInvertArc

Invert user defined arcs of all (cellName == NULL) or only the given cell.

```
zBool zConepInvertArc(
    const char* cellname
);
```

zConepUserdefArcCheck

Check user defined arcs and issue a warning if not all port arcs are defined.

```
void zConepUserdefArcCheck(
    DB* db
);
```

zConepUserdefArcSet

Create user defined arc list from arcTab entries for the given cell.

```
void zConepUserdefArcSet(
    DB*       db,
    Cell*     cell,
    CellFlags cflag,
    PortFlags pflag
);
```

zConepCalcArcList

Calculate arc list for the port at the given cell.

```
int* zConepCalcArcList(
    DB*   db,
    Cell* cell,
    int   port,
    zBool toIn
);
```

Find Objects

Type Definitions for the Find API

Callback Return Value

The return value of the find callback function. zFindCallbackResult_Continue if the search should continue, zFindCallbackResult_Break if the search should stop, zFindCallbackResult_Error if an error has occurred.

```
typedef enum zFindCallbackResult_ {
    zFindCallbackResult_Continue = 0,
    zFindCallbackResult_Break    = 1,
    zFindCallbackResult_Error    = 2
} zFindCallbackResult;
```

zFindCallbackResult

The zFindCallback callback function are called for each matching object. The given "context" is passed to the callback function.

The "oid" is referenced before the callback function is called, and is dereferenced afterwards.

```
typedef zFindCallbackResult (*zFindCallback)(void* context, const OID* oid);
```

zFindType

Specify the object type to find.

```
typedef enum zFindType_ {
    zFindType_None      = 0,
    zFindType_Inst      = (1 << 0),
    zFindType_Port      = (1 << 1),
    zFindType_PortBus   = (1 << 2),
    zFindType_Net       = (1 << 3),
    zFindType_NetBus    = (1 << 4),
    zFindType_Pin       = (1 << 5),
    zFindType_PinBus    = (1 << 6),
    zFindType_Module    = (1 << 7),
    zFindType_Primitive = (1 << 8),
    zFindType_Parasitic = (1 << 9),
    zFindType_Any       = (1 << 10) - 1
} zFindType;
```

zFindResultType

Specify the find result type.

```
typedef enum zFindResultType_ {
    zFindResultType_modbased  = (1 << 0),
    zFindResultType_treebased = (1 << 1),
    zFindResultType_both      = zFindResultType_modbased |
                                zFindResultType_treebased
} zFindResultType;
```

Find Functions

zFind_simple

Parameters: db the data base to work on type the type of objects to search, a bitwise combination of zFindType pattern the search pattern ignoreCase if zTrue, be case insensitive exact if zTrue, perform fixed-string matching, else perform wild-card-style pattern matching hiersep the hiersep character used in pattern; if '\0', try to guess the hiersep character from the pattern callback the callback function to call on each match callbackContext a user-specified context to hand over to the callback function

Return values: zTrue if no error occurred zFalse if there was an error; in this case, the actual error message can be found in zLastErrorMsg

```
zBool zFind_simple(
    DB*           db,
    int           type,
    const char*   pattern,
    zBool         ignoreCase,
    zBool         exact,
    char          hiersep,
    zBool         autoPopulate,
    zFindCallback callback,
    void*         callbackContext
);
```

zFind

Parameters: db the data base to work on type the type of objects to search, a bitwise combination of zFindType namePattern the object name search pattern namePatternIgnoreCase if true, namePattern is matched case insensitively. pathPattern the object path search pattern pathPatternIgnoreCase if zTrue, pathPattern is matched case insensitively. topPattern the object top search pattern topPatternIgnoreCase if zTrue, topPattern is matched case insensitively. cellPattern the cell search pattern; used when searching for instances (the instance must instantiate a matching cell), pins/pinBuses, ports/portBuses, nets/netBuses. cellPatternIgnoreCase if zTrue, cellPattern is matched case insensitively. exact if zTrue, perform fixed-string matching, else perform wild-card-style pattern matching hiersep the hiersep character used in pattern; if '\0', try to guess the hiersep character from the pattern dontMatchHiersep if zTrue, wild-cards in the pathPattern won't match any hiersep characters resultType type of result objects to report callback the callback function to call on each match callbackContext a user-specified context to hand over to the callback function

Return values: zTrue if no error occurred zFalse if there was an error; in this case, the actual error message can be found in zLastErrorMsg

```
zBool zFind(
    DB*           db,
    int           type,
    const char*   namePattern,
    zBool         namePatternIgnoreCase,
    const char*   pathPattern,
    zBool         pathPatternIgnoreCase,
    const char*   topPattern,
    zBool         topPatternIgnoreCase,
    const char*   cellPattern,
    zBool         cellPatternIgnoreCase,
    zBool         exact,
    char          hiersep,
    zBool         dontMatchHiersep,
    int           resultType,
    zBool         autoPopulate,
    zFindCallback callback,
    void*         callbackContext
);
```

Database Utilities

Subtract two OID Lists

zUtilsSubtract

Subtract oidList2 from oidList1. oidLists1: list of unref'd OIDs oidLists2: list of unref'd OIDs resultList: output list of ref'd OIDs

```
zBool zUtilsSubtract(
    DB*   db,
    OID** resultList,
    OID*  oidList1,
    OID*  oidList2
);
```

Create a Path from a String

zUtilsPathFromString

Split the given string into a list of instances and a string containing the last part. The names are separated by a hierSep. The last part is separated by the given delim (or hierSep if '\0'). If guess is true an 'X' is prepended to instance names for a second chance, if the original instance name is not found. The resulting Instance list is stored in pathPtr and the returned oname must be freed by calls to zFree.

```
zBool zUtilsPathFromString(
    DB*         db,
    Module*     top,
    OidType     type,
    const char* str,
    char        hierSep,
    char        delim,
    char        escape,
    ce_Bool     icase,
    ce_Bool     guess,
    ce_Bool     escapeVerilog,
    ce_Bool     autoPopulate,
    const char* topInstName,
    Module**    topModulePtr,
    Inst***     pathPtr,
    char**      onamePtr
);
```

zUtilsPathFromStringList

Split the given string into a list of instances and a string containing the last part. The names are separated by a hierSep. The last part is separated by the given delim (or hierSep if '\0'). If guess is true an 'X' is prepended to instance names for a second chance, if the original instance name is not found. The resulting Instance list is stored in pathPtr and the returned oname must be freed by calls to zFree.

```
zBool zUtilsPathFromStringList(
    DB*          db,
    Module**     topModule,
    const char** pathList,
    int          pathListLength,
    ce_Bool      icase,
    ce_Bool      guessStart,
    ce_Bool      autoPopulate,
    Inst***      pathPtr
);
```

zUtilsPath2Str

Concat path of a given oid to a string.

```
void zUtilsPath2Str(
    const OID*   oid,
    char         div,
    DStr* str
);
```

zUtilsCmpInstNameVal

Compare instance by name and value (e.g. for zQsort).

```
int zUtilsCmpInstNameVal(
    Inst** a,
    Inst** b
);
```

zUtilsCmpInstNameValVP

Compare instance by name and value (e.g. for zQsort).

```
int zUtilsCmpInstNameValVP(
    const void* objA,
    const void* objB
);
```

zUtilsFormatValLabel

Format value label using vdi data.

```
zBool zUtilsFormatValLabel(
    DB*                     db,
    const char*             topinfo,
    const char*             path,
    const char*             nlv,
    OidType                 otype,
    void*                   obj,
    int                     pinNo,
    char**                  attrList,
    char*                   val,
    unsigned                valSize,
    const char*             pseudoattr,
    const char*             pseudoval
);
```

zUtilsFormatOidValLabel

Format value label using oid.

```
zBool zUtilsFormatOidValLabel(
    DB*         db,
    const OID*  oid,
    char*       val,
    unsigned    valSize,
    const char* pseudoattr,
    const char* pseudoval,
    const char* prepend
);
```

zUtilsNlv2Oid

.

```
void zUtilsNlv2Oid(
    DB*         db,
    const char* nlvID,
    const char* nlvTopInfo,
    const OID*  currentModule,
    OID**       oidList
);
```

zUtilsOid2Nlv

.

```
void zUtilsOid2Nlv(
    DB*         db,
    const OID*  oid,
    const char* topInfo,
    char**      nlvID
);
```

The Database Operators

Generic Operators

zOperTSort

Sorts the given Cell-lists for traversals in a "defined before use" manner.

```
zBool zOperTSort(
    Cell** cellList,
    zBool  checkOnly
);
```

zOperMakeTop

Computes top modules if the database currently defines less than topCnt tops.

```
zBool zOperMakeTop(
    DB* db,
    int topCnt
);
```

zOperUseAllTopCandidates

Use all modules of the top candidates as top modules. This list is generated while guessing top modules. If there are no top candidates present do nothing.

```
zBool zOperUseAllTopCandidates(
    DB*   db
);
```

zOperGuessTopModule

Guess top module(s) in the given database. If all modules which are not instantiated are flagged as libcell, use all of them as top, if useLibcellFlag is set. If singleTop is set, use only the best module as top. In all other cases use an educated guess.

```
zBool zOperGuessTopModule(
    DB*   db,
    zBool useAllUnreferencedCells,
    zBool useLibcellFlag,
    zBool singleTop,
    zBool force,
    zBool restore
);
```

zOperSortTop

Divide the database's topList into real-tops and library cells. Sort each section alphabetically.

```
zBool zOperSortTop(
    DB* db
);
```

zOperDefTop

Defines the module as the new top (and deletes potential instances of it).

```
zBool zOperDefTop(
    DB*     db,
    Module* topmod
);
```

zOperTmpTop

Defines the module as the new tmp top. If topmod is NULL reset topList to save original tops.

```
zBool zOperTmpTop(
    DB*     db,
    Module* topmod
);
```

zOperSetTop

Define the module with the given name as the new top.

```
zBool zOperSetTop(
    DB*         db,
    const char* topModuleName,
    zBool       icase
);
```

zOperDeleteUnused

Deleted the not-instantiated modules (except the top modules).

```
zBool zOperDeleteUnused(
    DB* db
);
```

zOperZombieUnused

Flag cells with refCount == 0 as Zombies (except the top modules and libCells).

```
zBool zOperZombieUnused(
    DB* db
);
```

zOperValidatePG

Search DB for Power/Ground nets. Return zFalse if non.

```
zBool zOperValidatePG(
    DB* db
);
```

zOperMergeNet

Merges the two given nets.

```
zBool zOperMergeNet(
    DB*     db,
    Module* module,
    Net*    toNet,
    Net*    fromNet,
    zBool   spos,
    zBool   ignorePortConn
);
```

zOperMergeNetBus

Merges the two given netBuses.

```
zBool zOperMergeNetBus(
    DB*     db,
    Module* module,
    NetBus* to,
    NetBus* from,
    zBool   spos
);
```

zOperConnect

Connect a net to a pin.

```
zBool zOperConnect(
    DB*     db,
    Module* module,
    PinRef* ref,
    Net*    net
);
```

zOperDisconnect

Disconnect a pin.

```
Net* zOperDisconnect(
    DB*     db,
    Module* module,
    PinRef* ref
);
```

Device Related Operators

```

```

zOperBulk

Toggle the bulk visibility. mode: 0 = showwrong, 1 = all, 2 = none, 3 = nopg.

```
zBool zOperBulk(
    DB* db,
    int mode
);
```

zOperGroupMultiFinger

Group multifinger transistors.

```
zBool zOperGroupMultiFinger(
    DB*         db,
    const char* sep
);
```

zOperEvalParams

Evaluate parameters. Hspice has two modes controlled by ".option parhier" GLOBAL (the default) and LOCAL. We process the whole hierarchy and set not constant attributes in the flat tree. In 'global' mode parameter which are set once, are NOT overwritten during going down the hierarchy. In 'local' mode, parameter get overwritten by instance attribute expressions and are preset by module default attribute expressions. Recursive expression using the value of other parameters are allowed. Attribute which have names beginning with '@' or '$' are ignored.

```
zBool zOperEvalParams(
    DB*   db,
    zBool parhierLocal,
    zBool icase
);
```

zOperPowerAndDirSettings

Set power and directions flags from -node settings.

```
zBool zOperPowerAndDirSettings(
    DB*                 db,
    struct SpiceNodes*,
    zBool               icase
);
```

zOperGuessPower

Guess power and ground nodes. Set NetFInternalPotentialPower/NetFInternalPotentialGround based on connected bulk pins. if setFlag is true NetFPower/NetFGround are set too.

```
zBool zOperGuessPower(
    DB*   db,
    zBool setFlag
);
```

zOperGuessDir

Guess port dirs of modules.

```
zBool zOperGuessDir(
    DB* db
);
```

zOperPowerAndDirGuess

Guess power nodes and port directions.

```
zBool zOperPowerAndDirGuess(
    DB*   db,
    zBool icase,
    zBool avoidShorted,
    zBool evalVsrc2P,
    zBool evalVsrc2I,
    zBool addTopPorts,
    zBool force,
    int   pwrprop
);
```

zOperHidePowerPorts

Check each subckt port. If connected to power/ground/negpower or unconnected set hide flags.

```
zBool zOperHidePowerPorts(
    DB* db
);
```

zOperWeakFlow

Set weakflow instance flags for resistors and transistors. Check whether resistance is big enough. Check whether w/l is less than 1 (w-l is less than 0).

```
zBool zOperWeakFlow(
    DB*    db,
    zBool  hspice,
    double resLimit
);
```

Database Modification Operators

```

```

zOperCleanup

- If there were port flags modified, the pin flags need to be updated.
- If there were global nets which are not power/ground, make warning.
- Remove top (unreferenced) modules which contain only zombie insts.

```
void zOperCleanup(
    DB* db
);
```

zOperSinglize

Make the given instance the only one referring to its down-module (make inst→cellRef→refCount == 1).

```
zBool zOperSinglize(
    DB*      db,
    Inst*    inst,
    Module** reload
);
```

zOperSinglizePath

Make all instances in the given path refCount == 1.

```
zBool zOperSinglizePath(
    DB*      db,
    int      pathLen,
    Inst*    path[128],
    Module** reload
);
```

zOperSinglizeTree

Traverse all database hierarchy and make refCount == 1.

```
zBool zOperSinglizeTree(
    DB*     db,
    Module* mod
);
```

zUpdateSinglize

All three Singlize functions create duplicate Modules (clones with identical names). They call zUpdateSinglize to update OIDs that are affected. zUpdateSinglize is initially NULL, but can be set to a callback function - to get called directly after database objects have been singlized.

```
extern void (*zUpdateSinglize)(
    DB*,
    InstFlags
);
```

zOperFlatDesign

Flatten given top module

```
zBool zOperFlatDesign(
    DB*         db,
    struct OID* top,
    zBool       spos,
    zBool       keep
);
```

zOperCollectSignalData

Transport net values, pg-flags, spos and attributes to the corresponding signal.

```
typedef zUInt8 ColWhat;
#define ColPG       0x01
#define ColValue    0x02
#define ColSpos     0x04
#define ColAttr     0x08
#define ColHilight  0x10
#define ColFAttr    0x20
#define ColFHilight 0x40

zBool zOperCollectSignalData(
    DB*     db,
    Module* top,
    ColWhat what
);
```

zOperHiersep

Scan the database inst/net/netBus names and return separator character. The given "mode" is: 0 for "scan" 1 for "get" 2 for "wish" 3 for "set"

```
zBool zOperHiersep(
    DB*           db,
    int           mode,
    char*         result,
    unsigned char w
);
```

zOperUnused

Scan database and return characters which is not used in the design and by the hiersep. For each char in 'wish' a unused char is returned in the result.

```
zBool zOperUnused(
    DB*         db,
    const char* wish,
    char*       res
);
```

zOperRenameAllUniq

Rename all Modules, Primitives, Ports and PortBusses.

```
zBool zOperRenameAllUniq(
    DB*   db,
    zBool updateOIDs
);
```

zOperRenameDigit

Rename all nets and ports to avoid starting with a digit.

```
zBool zOperRenameDigit(
    DB*   db,
    zBool icase
);
```

zOperRename

Rename the object given as an OID.

```
zBool zOperRename(
    DB*         db,
    struct OID* oid,
    const char* newName,
    zBool       updateOIDs,
    zBool       checkName
);
```

zOperChangeCellRef

Change the cell referenced by an instance. Special case: if inst == NULL: only call zUpdateRename.

```
zBool zOperChangeCellRef(
    DB*     db,
    Module* mod,
    Inst*   inst,
    Cell*   newCellRef,
    zBool   updateOIDs
);
```

zOperChangeSimilar

Change cellRef of an instance with similar interface. Connectivity is copied to a cloned instance by using port names. Scalar port to portbus with same name is possible. The old instance is flagged zombie and needs to be removed by calling zDeleteZombies.

Modify Hierarchy

```

```

zOperAddHier

Move some instances into a new module (level of hierarchy). Create a new module "mname" and one instance "iname" of it. Place the instance "iname" in the given module "parent" and: + Move all

instances "instList" from "parent" to the new module. + Move all connected local nets from "parent" to the new module. + Split all connected crossing nets (the original one stays at the outside and a new one is for the inside). + if useFirstInst is true, all nets connected to the first inst of instList are forced as interface nets, port names of this instance are used for new ports. + if pwrGndToo is true, all power gnd nets create interface ports/nets. Update all OIDs. Update flat-tree data structure. If "prefix" is not NULL, then remove prefix from instance, net and netBus names when they are moved or created inside the new module.

```
Inst* zOperAddHier(
    DB*         db,
    Module*     parent,
    const char* mname,
    const char* iname,
    Inst**      instList,
    const char* prefix,
    zBool       useFirstInst,
    zBool       pwrGndToo
);
```

zOperRmHier

Remove one level or hierarchy. The given "rinst" (instance to remove) refers to "rmod" (module to remove), and rinst must be the only instance of rmod (no multiple instances).

Do this: + Move all instances from "rmod" to "parent". + Move all rmod's local inside nets from "rmod" to "parent". + Merge connectivity of the crossing nets. + Remove "rmod" and "rinst". When moving the instances and local nets, then we may need to prefix their names to avoid name clashes. If so, then return the prefix string. Update all OIDs. Update flat-tree data structure.

```
zBool zOperRmHier(
    DB*         db,
    Module*     parent,
    Inst*       rinst,
    char        prefix[64],
    const char* given_prefix,
    zBool       delZombies
);
```

zUpdateAddHier

UpdateRmHier and zUpdateRename are initially NULL, but they can be set to callback functions - to get called directly after database objects have been moved.

```
extern void (*zUpdateAddHier)(
    DB*,
    Module*,
    InstFlags,
    NetFlags,
    NetFlags,
    Inst*
);
extern void (*zUpdateRmHier)(
    DB*,
    Module*,
    Inst*,
    NetFlags,
    NetFlags
);
extern int  (*zUpdateRename)(
    DB*
);
```

zOperHierStart

Start a set of HierAdd calls.

```
zBool zOperHierStart(
    DB*     db,
    Module* top,
    int     fold
);
```

zOperHierAdd

Add an extra level of hierarchy.

```
Inst* zOperHierAdd(
    DB*          db,
    const char*  cname,
    const char*  iname,
    const char** iList,
    int          bCount,
    const char** oList,
    const char** sList,
    const char** memList,
    zBool        oneg,
    int          hsep,
    const char*  foldkey,
    const char*  base,
    zBool        doGuessPortDir,
    zBool        createSupplyPorts
);
```

zOperHierFinish

Finish and cleanup the set of zOperHierAdd calls.

```
zBool zOperHierFinish(
    DB* db
);
```

zOperHierFoldCount

Return fold-table size (for debugging only).

```
int zOperHierFoldCount(
    void
);
```

zOperHierPathCount

Return flat attr path count (for debugging only).

```
int zOperHierPathCount(
    void
);
```

zOperHierFoldDupl

Return no of dupl entries in fold-table for given key.

```
int zOperHierFoldDupl(
    const char*
);
```

zOperHierFoldBase

Return no of base entries in base-table for given base.

```
int zOperHierFoldBase(
    const char*
);
```

zOperHierFoldList

Return list of modules inf fold-table (caller must call zArrayFree). Works outside of Start/Finish.

```
Module** zOperHierFoldList(
    DB* db
);
```

zInvalidateFlagged

Initially NULL, but it can be set to a callback functions - to get called in zOperHierFinish to cut off all foreign pointers to visited inst and nets.

```
extern void (*zInvalidateFlagged)(
    DB*,
    CellFlags,
    InstFlags,
    NetFlags,
    PortFlags,
    VirtualFlags
);
```

zOperCreateHier

Completely create new levels of hierarchy by pathsep If top == NULL loop over all tops.

```
zBool zOperCreateHier(
    DB*         db,
    Module*     top,
    char        pathsep,
    const char* prefix,
    zBool       doGuessPortDir,
    zBool       createSupplyPorts
);
```

Netlist Reduction

```

```

zOperCollectModuleParam

Find all instance attributes of each module.

```
void zOperCollectModuleParam(
    DB* db
);
```

zOperGuessPortBus

Guess port buses.

```
zBool zOperGuessPortBus(
    DB*         db,
    Cell*       cell,
    const char* open,
    const char* close,
    zBool       down
);
```

zOperGuessNetBus

Guess net buses

```
zBool zOperGuessNetBus(
    DB*         db,
    Module*     mod,
    const char* open,
    const char* close,
    zBool       down
);
```

zOperGuessInstArray

Guess instance arrays.

```
zBool zOperGuessInstArray(
    DB*         db,
    Module*     module,
    const char* startDelim,
    const char* endDelim
);
```

zOperSortPorts

Sort ports in bus sequence, update pin connectivity.

```
zBool zOperSortPorts(
    DB*   db,
    zBool down
);
```

zOperGuessBusses

Wrapper for the above **GuessBus** operators.

```
zBool zOperGuessBusses(
    DB*         db,
    const char* startDelim,
    const char* endDelim,
    zBool       down
);
```

zOperVerilogBusses

Create Verilog conform buses.

```
zBool zOperVerilogBusses(
    DB*   db,
    Cell* cell
);
```

zOperExpand

Expand subckts and replace subckt instances by their contents and delete expanded instances and modules.

```
zBool zOperExpand(
    DB*          db,
    zBool        autoExpand,
    zBool        autoExpand0,
    const char** expands,
    zBool        icase
);
```

zOperSubckt2Dev

Set primitive function for matching subckts.

```
zBool zOperSubckt2Dev(
    DB*               db,
    struct StrTriple* subckt2dev,
    zBool             icase
);
```

zOperShortRes

Short resistors.

```
zBool zOperShortRes(
    DB*         db,
    const char* limitStr,
    zBool       hspice
);
```

zOperMergeParallelInst

Merge parallel connected instances.

```
zBool zOperMergeParallelInst(
    DB*                           db,
    Module*                       mod,
    zBool                         hspice,
    zBool                         parasitic,
    zBool                         icase,
    struct MergeParallelInstEntry* entries

);
```

zOperMergeParallelCap

Merge parallel connected capacitors.

```
zBool zOperMergeParallelCap(
    DB*     db,
    Module* mod,
    zBool   hspice,
    zBool   para
);
```

zOperMergeParallelRes

Merge parallel connected resistors.

```
zBool zOperMergeParallelRes(
    DB*     db,
    Module* mod,
    zBool   hspice,
    zBool   parasitic
);
```

zOperMergeParallelDiode

Merge parallel connected diodes.

```
zBool zOperMergeParallelDiode(
    DB*     db,
    Module* mod,
    zBool   hspice,
    zBool   parasitic
);
```

zOperMergeSerialRes

Merge serial connected resistors.

```
zBool zOperMergeSerialRes(
    DB*     db,
    Module* mod,
    zBool   hspice,
    zBool   parasitic
);
```

zOperMergeSerialCap

Merge serial connected capacitors.

```
zBool zOperMergeSerialCap(
    DB*     db,
    Module* mod,
    zBool   hspice,
    zBool   parasitic
);
```

zOperRemoveMOS

Remove mos.

```
zBool zOperRemoveMOS(
    DB*   db,
    zBool useless
);
```

zOperRemoveUseless

Remove useless.

```
zBool zOperRemoveUseless(
    DB* db
);
```

zOperRemoveRes

Remove res which are connected to only 1 net.

```
zBool zOperRemoveRes(
    DB* db
);
```

zOperRemoveCap

Remove cap which are connected to only 1 net.

```
zBool zOperRemoveCap(
    DB* db
);
```

zOperMergeParallel

Merge parallel connected transistors.

```
zBool zOperMergeParallel(
    DB*         db,
    Module*     mod,
    zBool       hspice,
    zBool       parasitic,
    const char* multi,
    double      equality

);
```

zOperMergeSerial

Merge serial connected transistors.

```
zBool zOperMergeSerial(
    DB* db
);
```

zOperMergeRams

Merge read/write ports into RAM instances.

```
zBool zOperMergeRams(
    DB*     db,
    Module* mod
);
```

zOperRemoveEmptyModule

Remove empty modules.

```
zBool zOperRemoveEmptyModule(
    DB*   db,
    zBool ckDanglingNets,
    zBool ckZombies
);
```

zOperDeletePort

Delete ports from a cell.

```
zBool zOperDeletePort(
    DB*          db,
    Cell*        cell,
    const char** portNames
);
```

zOperRtlSchem

Meaningful combination of the following operpp than can be used for RTL schematic visualization.

```
zBool zOperRtlSchem(
    DB*   db,
    int   level,
    zBool preserveAssign
);
```

zOperRemoveBuf

Merge the nets connected to BUF and WIDE_BUF instances. Remove the BUF instance.

```
zBool zOperRemoveBuf(
    DB*      db,
    Module*  module,
    unsigned opts
);
```

zOperRemoveInv

Remove inverter connected to power/ground.

```
zBool zOperRemoveInv(
    DB*      db,
    Module*  module,
    unsigned opts
);
```

zOperCreateConst

Replace power/ground nets with a constant net.

```
zBool zOperCreateConst(
    DB*      db,
    Module*  module,
    unsigned opts
);
```

zOperChain

Replace MUX chains by PRIO_SELECTOR instances. Replace boolean chains by reduced functions.

```
zBool zOperChain(
    DB*      db,
    Module*  module,
    unsigned opts
);
```

zOperGuessWide

Combine scalar instances to wide instances if multiple outputs connect to same bus pin

```
zBool zOperGuessWide(
    DB*      db,
    Module*  module,
    unsigned opts
);
```

zOperReducePins

Create extra hierarchy for Instances which have same nets connected to multiple pins.

```
zBool zOperReducePins(
    DB*      db,
    Module*  module,
    unsigned opts
);
```

zOperRemoveDangle

Remove unconnected nets, and empty netbuses.

```
zBool zOperRemoveDangle(
    DB*      db,
    Module*  module,
    unsigned opts
);
```

zOperBubbles

Replace inverters by bubbles (or change function)

```
zBool zOperBubbles(
    DB*      db,
    Module*  module,
    unsigned opts
);
```

zOperNegEdge

Remove all inverter directly connected to a clk, set or rst pin of a Flip-Flop and add a bubble to the pin.

```
zBool zOperNegEdge(
    DB*      db,
    Module*  module,
    unsigned opts
);
```

zOperInstArray

Find wide block by using instname.

```
zBool zOperInstArray(
    DB*      db,
    Module*  module,
    unsigned opts
);
```

zOperGuessWideBus

ind bundle connecting to same driver.

```
zBool zOperGuessWideBus(
    DB*      db,
    Module*  module,
    unsigned opts
);
```

zOperRemoveUnused

emove unreferenced cells.

```
zBool zOperRemoveUnused(
    DB*      db,
    Module*  module,
    unsigned opts
);
```

zOperConnectByName

Mark net(s) as not to be routed.

```
zBool zOperConnectByName(
    DB*         db,
    const char* netNamePattern,
    zBool       exact,
    zBool       caseSensitive
);
```

Parasitic Operators

zOperAnalyzeCoupling

Analyze coupling connections in parasitic modules.

```
zBool zOperAnalyzeCoupling(
    DB*   db,
    zBool icase
);
```

zOperParaRmHier

Remove parasitic hierarchy.

```
zBool zOperParaRmHier(
    DB*         db,
    Module*     mod,
    Inst*       inst,
    const char* prefix,
    char        div,
    zBool       delZombies
);
```

zOperParaRename

Rename instances of parasitic module.

```
zBool zOperParaRename(
    DB*     db,
    Module* mod
);
```

zOperParaPex

Rename ports of parasitic modules using !PARAINST and !PARAPORTNO. Because the real port names are only known **after** subckt2dev macro guessing was done. In the pex file the subckt names are only numeric node numbers.

```
zBool zOperParaPex(
    DB* db,
    Module* mod
);
```

zOperParaMinMax

Calculate min/max cap/res/ind values.

```
zBool zOperParaMinMax(
    DB*     db,
    Module* mod
);
```

zOperParaNetCap

Calculate netcap values.

```
zBool zOperParaNetCap(
    DB*     db,
    Module* mod
);
```

zOperParaFinish

cleanup flags.

```
zBool zOperParaFinish(
    DB*   db,
    zBool deleteUnmatched
);
```

zOperParaInline

Create inline parasitics in design.

```
zBool zOperParaInline(
    DB*   db,
    zBool inlineMod,
    const char* modPrefix
);
```

zOperParaRollback

Try to cleanup after interrupt db.

```
zBool zOperParaRollback(
    DB*   db,
    zBool delZombies
);
```

zOperCreatePreplace

Create preplace info from @X,@Y.

```
zBool zOperCreatePreplace(
    DB* db
);
```

zOperLayoutComments

add mapped layout attributes to attrList values of "@layer" attribute get mapped via mapFunc (if != NULL).

```
void zOperLayoutComments(
    DB* db,
    char*** attrListPtr,
    const char* comment,
    const char* (*layerValueMap)(int)
);
```

zOperReportParallelInst

Report parallel connected instances.

```
struct ReportParallelInstResult {
    Module* mod;
    Inst**   instList;
};

zBool zOperReportParallelInst(
    DB*                             db,
    Module*                         mod,
    zBool                           hspice,
    zBool                           parasitic,
    zBool                           icase,
    struct MergeParallelInstEntry* entries,
    struct ReportParallelInstResult** reportResultListPtr
);
```

zOperMergeSerialParallel

Merge serial and parallel connected transistors.

```
zBool zOperMergeSerialParallel(
    DB*         db,
    Module*     mod,
    zBool       hspice,
    const char* multi,
    double      equality
);
```

zOperRmHierDynamicPrefix

Remove one level or hierarchy. The given "rinst" (instance to remove) refers to "rmod" (module to remove), and rinst must be the only instance of rmod (no multiple instances).

Do this: + Move all instances from "rmod" to "parent". + Move all rmod's local inside nets from "rmod" to "parent". + Merge connectivity of the crossing nets. + Remove "rmod" and "rinst". When moving the instances and local nets, either the given_prefix, or an automatically created unique prefix (derived from "rinst"'s name) is prepended to avoid name clashes. The used prefix is returned in the allocated string "*used_prefix", which the caller must free with "zFree(*used_prefix)". Update all OIDs. Update flat-tree data structure.

```
zBool zOperRmHierDynamicPrefix(
    DB*         db,
    Module*     parent,
    Inst*       rinst,
    zBool       delZombies,
    const char* given_prefix,
    char**      used_prefix
);
```

zOperFlattenSubtree

Flatten the complete subtree rooted at the given instance.

```
zBool zOperFlattenSubtree(
    DB*         db,
    struct OID* rootInstance
);
```

Database Reduction Operators

Reduce Resistance

zOperReduceRes

Reduce resistance network if doMesh is true, n-star to mesh transformation is done. if doModify is true, the given modules gets modified. if intoModule is given, the reduced network is created in given module. if portNoList == NULL (or cnt==0) all nets connected to a port get not reduced. if portNoList != NULL only nets connected to listed port get not reduced. if resList != NULL, portNoList must even number of ports, for each pair the network get reduce to one resistor the resistance are returned in resList, which must have space for cnt/2 doubles.

```
zBool zOperReduceRes(
    DB*     db,
    Module* mod,
    int*    portNoList,
    int     cnt,
    zBool   doMesh,
    zBool   doModify,
    Module* intoModule,
    double* resList
);
```

The Database Calculations

Calculate Resistance

zCalcRes

Calculate the resistance of a network between two ports.

```
zBool zCalcRes(
    DB*     db,
    Module* module,
    int     portNo1,
    int     portNo2,
    long    loopLimit,
    double  minChange,
    double* res
);
```

zCalcResList

Calculate resistances between port pairs.

```
zBool zCalcResList(
    DB*     db,
    Module* mod,
    int     len,
    int*    portNoPairList,
    long    loopLimit,
    double  minChange,
    double* resList
);
```

Convert Values

ce_Radix Enumeration

Enumeration to define all supported radix types.

```
typedef enum {
    ce_UnknownRadix = 0,
    ce_Bin = 2,
    ce_Oct = 8,
    ce_Dec = 10,
    ce_Hex = 16
} ce_Radix;
```

ce_ConvertStringToRadix

Get ce_Radix enum type for bin|oct|dec|hex|? (case insensitive) string.

```
ce_Radix ce_ConvertStringToRadix(const char* inputValue);
```

ce_ConvertRadixToString

Return the string representation for a ce_Radix enum type.

```
const char* ce_ConvertRadixToString(ce_Radix radix);
```

ce_ConvertRadixUsageList

Add all ce_Radix values except ce_UnknownRadix to the list.

```
void ce_ConvertRadixUsageList(const char*** list);
```

Spice Conversions

ce_ConvertSpiceUnitToDouble

Get the scale for a given string like 'K' (→ 1e3) or 'p' (→ 1e-12). Set hspice to ce_True to enable the usage of 'a' (atto → 1e-18). Return ce_False and set ce_ErrorLastMsg on errors.

```
ce_Bool ce_ConvertSpiceUnitToDouble(const char* inputValue, ce_Bool hspice,
                                    double* result);
```

ce_ConvertSpiceValueToDouble

Convert a Spice value into a double. Set hspice to ce_True to enable the usage of 'a' (atto → 1e-18). Return ce_False and set ce_ErrorLastMsg on errors.

```
ce_Bool ce_ConvertSpiceValueToDouble(const char* inputValue, ce_Bool hspice,
                                     double* result);
```

ce_ConvertDoubleToSpice

Convert a double into Spice string representation. Set hspice to ce_True to enable the usage of 'a' (atto → 1e-18). Result needs to be initialized by the caller. Return ce_False and set ce_ErrorLastMsg on errors.

```
ce_Bool ce_ConvertDoubleToSpice(double inputValue, ce_Bool hspice,
                                ce_DString* result);
```

Binary Conversions

ce_ConvertFromBinary

Convert binary bits to one string value with given radix. Set radix to the desired radix of result. Set addPrefix to ce_True to add a preceding 'O' if radix is ce_Oct or "0x" if radix is ce_Hex. Set

addVerilogPrefix to add a preceding "len'(b|o|d|h)" in Verilog style to the result, depending on the given radix. Setting both addPrefix and addVerilogPrefix to ce_True is not allowed! Set noOfBits to add preceding zeros to the result (ignored if noOfBits is too small). Result needs to be initialized by the caller. Return ce_False and set ce_ErrorLastMsg on errors.

```
ce_Bool ce_ConvertFromBinary(const char* inputValue, ce_Radix radix,
                             ce_Bool addPrefix, ce_Bool addVerilogPrefix,
                             unsigned noOfBits, ce_DString* result);
```

ce_ConvertToBinary

Convert a string with optional prefix to binary bits. Set radix to the radix of the inputValue. Set hasPrefix if the inputValue starts with '0' (octal) or "0x" (hexadecimal). Set addVerilogPrefix to add a preceding "len'(b|o|d|h)" in Verilog style to the result, depending on the given radix. Set noOfBits to add preceding zeros to the result (ignored if noOfBits is too small). Result needs to be initialized by the caller. Return ce_False and set ce_ErrorLastMsg on errors.

```
ce_Bool ce_ConvertToBinary(const char* inputValue, ce_Radix radix,
                           ce_Bool hasPrefix, ce_Bool addVerilogPrefix,
                           unsigned noOfBits, ce_DString* result);
```

Verilog Conversions

ce_ConvertFromVerilogValue

Format: [0-9_]* or [0-9_]\'[bodh][0-9a-fxz]. Convert a Verilog value into ce_BigInt. Result needs to be initialized by the caller. Return ce_False and set ce_ErrorLastMsg on errors.

```
ce_Bool ce_ConvertFromVerilogValue(const char* inputValue, ce_BigInt* result);
```

ce_ConvertToVerilogValue

Convert a ce_BigInt number in to a ce_DString in Verilog format: noOfBits\'[bodh][0-9a-f]. Set radix to the desired radix of result. Set noOfBits to add preceding zeros to the result (ignored if noOfBits is too small). Set sepDistance to x > 0 to include underscore separators every x digits or to 0 to disable underscore separators. Result needs to be initialized by the caller. Return ce_False and set ce_ErrorLastMsg on errors.

```
ce_Bool ce_ConvertToVerilogValue(ce_BigInt* inputValue, ce_Radix radix,
                                 unsigned noOfBits, unsigned sepDistance,
                                 ce_DString* result);
```

ce_ConvertChangeVerilogRadix

Convert a Verilog number in format ([0-9]\'[bodh][0-9a-f] or [0-9a-f]) to a ce_DString in Verilog format ([0-9]\'[bodh][0-9a-f]) with different radix. Set radix to the desired radix of result. Result

needs to be initialized by the caller. Return ce_False, set ce_ErrorLastMsg and copy inputValue to result on errors.

```
ce_Bool ce_ConvertChangeVerilogRadix(ce_DString* inputValue, ce_Radix radix,
                                     ce_DString* result);
```

Format Value

ce_ConvertAddThousandsSeparator

Format value with separators. Result needs to be initialized by the caller. Return ce_False and set ce_ErrorLastMsg on errors.

```
ce_Bool ce_ConvertAddThousandsSeparator(ce_Long inputValue, const char* sep,
                                        ce_DString* result);
```

Clone and Merge

Clone Database Objects

zClonePrimitive

Clone a primitive.

```
Primitive* zClonePrimitive(
    DB*          db,
    Cell*        orig,
    const char*  rename,
    zBool        bitblasted,
    PortFlags    skipFlaggedPort,
    zBool        spos
);
```

zCloneModule

Clone the complete module.

```
Module* zCloneModule(
    DB*          db,
    Module*      orig,
    const char*  rename,
    zBool        bitblasted,
    PortFlags    skipFlaggedPort,
    InstFlags    skipFlaggedInst,
    zBool        skipContent,
    zBool        spos
);
```

zCloneInst

Clone an instance.

```
Inst* zCloneInst(
    DB*         db,
    Module*     dest,
    Inst*       orig,
    const char* rename,
    zBool       spos
);
```

zCloneNet

Clone a net.

```
Net* zCloneNet(
    DB*         db,
    Module*     dest,
    Net*        orig,
    const char* rename,
    zBool       spos
);
```

zCloneAttrList

Clone an attribute list.

```
char** zCloneAttrList(
    DB*    db,
    char** attrList
);
```

Clone a Database

zCloneDB

Clone all flagged objects of the given database and either return the cloned database or clone into an existing database.

```
DB* zCloneDB(
    DB*          src,
    DB*          into,
    CellFlags    keepCell,
    InstFlags    keepInst,
    NetFlags     keepNet,
    PortFlags    keepPort,
    PinFlags     keepPin,
    zBool        spos
);
```

Merge two Databases

zMergeDB

Clone cells from one db into another. if resolveWithRename is true already existing cells get renamed. already existing instances still reference the renamed cell. if resolveWithRename is false only the contents of modules is cloned if the existing module is empty.

```
zBool zMergeDB(
    DB*   src,
    DB*   dst,
    zBool spos,
    zBool resolveWithRename
);
```

zClonePrimitiveNoUniq

Clone a primitive do not find an uniq name.

```
Primitive* zClonePrimitiveNoUniq(
    DB*          db,
    Cell*        orig,
    const char*  rename,
    zBool        bitblasted,
    PortFlags    skipFlaggedPort,
    zBool        spos
);
```

zCloneDBEx

Similar to zCloneDB, but add the option to create modules for all primitives in the source database.

```
DB* zCloneDBEx(
    DB*       src,
    DB*       into,
    CellFlags keepCell,
    InstFlags keepInst,
    NetFlags  keepNet,
    PortFlags keepPort,
    PinFlags  keepPin,
    zBool     primAsModule,
    zBool     spos);
```

Clock Domain Analyzer Functions

The CDC Object

zCdcNew

Create a new CDC object and calculate the clkPinOidList, the clkDomainList with clkSrc and clkPinOidList as well as the reducedClkPinIdxList of all clkDomains.

```
struct zCdc* zCdcNew(
    DB*     db,
    Module* top,
    zBool   compr,
    zBool   skipUndriven,
    int     skipLess
);
```

zCdcFree

Cleanup the CDC object.

```
void zCdcFree(
    struct zCdc* cdc
);
```

zCdcGetDB

Get the associated database.

```
DB* zCdcGetDB(
    struct zCdc* cdc
);
```

zCdcSetHook

Set hook for the Tcl command.

```
void zCdcSetHook(
    struct zCdc* cdc,
    void*        hook
);
```

zCdcGetHook

Return the hook for the Tcl command.

```
void* zCdcGetHook(
    struct zCdc* cdc
);
```

Clock Domains

zCdcGetDomainCnt

Get the number of clock domains.

```
int zCdcGetDomainCnt(
    struct zCdc* cdc
);
```

zCdcGetDomainList

List of domain indices, sorted by the number of contained clkPins.

```
int* zCdcGetDomainList(
    struct zCdc* cdc,
    zBool        incr
);
```

zCdcGetSrc

return clkSrc of given Domain, or NULL

```
OID* zCdcGetSrc(
    struct zCdc* cdc,
    int          clkDomainIdx
);
```

zCdcGetAllSources

Return list of all clock source OIDs, or NULL.

```
OID* zCdcGetAllSources(
    struct zCdc* cdc,
    int          clkDomainIdx
);
```

zCdcGetClkPinList

Return clkPinOidList of given Domain.

```
OID* zCdcGetClkPinList(
    struct zCdc* cdc,
    int          clkDomainIdx
);
```

zCdcGetClkPinCount

Return physical clock pin count of the given Domain.

```
zLong zCdcGetClkPinCount(
    struct zCdc* cdc,
    int          clkDomainIdx
);
```

zCdcGetReducedClkPinList

Return reduced clkPinOidList of given Domain.

```
OID* zCdcGetReducedClkPinList(
    struct zCdc* cdc,
    int          clkDomainIdx
);
```

zCdcGetTreeList

Get OIDs of the clock domain for the given domain index; include control logic if controlLogic==1.

```
OID* zCdcGetTreeList(
    struct zCdc* cdc,
    int          clkDomainIdx,
    zBool        cache,
    zBool        controlLogic
);
```

zCdcGetReducedTreeList

Get the reduced OID list of the clock domain for the given domain index; include control logic if controlLogic==1.

```
OID* zCdcGetReducedTreeList(
    struct zCdc* cdc,
    int          clkDomainIdx,
    zBool        cache,
    zBool        controlLogic
);
```

Clock Domain Crossing

zCdcGetTreeCrossCnt

Get the number of domain crossings.

```
int zCdcGetTreeCrossCnt(
    struct zCdc* cdc
);
```

zCdcGetTreeCrossList

Get domain cross indices sorted by the number of contained src|clkPins.

```
int* zCdcGetTreeCrossList(
    struct zCdc* cdc,
    int          mode
);
```

zCdcGetTreeCrossSrcList

Return clkSrcOidList of given domain crossing.

```
OID* zCdcGetTreeCrossSrcList(
    struct zCdc* cdc,
    int          treeCrossIdx
);
```

zCdcGetTreeCrossClkPinList

Return clkPinOidList of given domain crossing.

```
OID* zCdcGetTreeCrossClkPinList(
    struct zCdc* cdc,
    int          treeCrossIdx
);
```

zCdcGetTreeCrossCone

Return the cone for the given domain crossing.

```
OID* zCdcGetTreeCrossCone(
    struct zCdc* cdc,
    int          treeCrossIdx
);
```

zCdcGetClkPinDomains

Return domainIdxList of the given clkPinOid.

```
int* zCdcGetClkPinDomains(
    struct zCdc* cdc,
    OID*         clkPinOid
);
```

zCdcCalcDomainCross

Calculate the domain crossings and return the number of crossings.

```
int zCdcCalcDomainCross(
    struct zCdc* cdc,
    zBool        removeDRS,
    int*         domainIdxList
);
```

zCdcGetDomainCrossCnt

Get the number of domain crossings.

```
int zCdcGetDomainCrossCnt(
    struct zCdc* cdc
);
```

zCdcGetDomainCrossList

Get the domain cross indices sorted by the number of contained clkpins.

```
int* zCdcGetDomainCrossList(
    struct zCdc* cdc,
    zBool        incr
);
```

zCdcGetDomainCrossDomainPair

Fill lists with source and target domain indices. Return zFalse for wrong domain cross idx.

```
zBool zCdcGetDomainCrossDomainPair(
    struct zCdc* cdc,
    int          domainCrossIdx,
    int*         srcDomainIdx,
    int*         trgDomainIdx
);
```

zCdcGetDomainCrossTrgList

Return OID list for the given domain crossing.

```
OID* zCdcGetDomainCrossTrgList(
    struct zCdc* cdc,
    int          domainCrossIdx
);
```

zCdcGetDomainCrossCone

Return cone for the given domain crossing.

```
OID* zCdcGetDomainCrossCone(
    struct zCdc* cdc,
    int          domainCrossIdx
);
```

Database Report Functions

Count Object

zReportInstCount

Traverses the database hierarchy tree rooted at the given top and counts the number of instances of each primitive and each module.

```
struct InstCount {
    Cell*    cell;
    zUInt64  count;
};
zBool zReportInstCount(
    DB*                db,
    Module*            top,
    struct InstCount** resultListPtr
);
```

zReportNetCount

Traverses the database hierarchy tree rooted at the given "top" and count the number of nets.

zReportportCount

Traverses the database hierarchy tree rooted at the given "top" and count the number of ports.

```
zBool zReportPortCount(
    DB*       db,
    Module*   top,
    zUInt64*  resultCount,
    zBool     nofunc
);
```

ReportType

List object types that can be reported by the report functions below.

```
typedef zUInt16 ReportType;
#define ReportNone      0x0000
#define ReportCell      0x0001
#define ReportModInst   0x0002
#define ReportPrimInst  0x0004
#define ReportInst      (ReportModInst|ReportPrimInst)
#define ReportPin       0x0008
#define ReportNet       0x0010
#define ReportNetBus    0x0020
#define ReportPort      0x0040
#define ReportPortBus   0x0080
#define ReportParasitic 0x0100
#define ReportAll       0x01FF
```

zReportObjCount

Traverses the database hierarchy tree rooted at the given "top" and count the number of objs.

```
zBool zReportObjCount(
    DB*             db,
    Module*         top,
    zUInt64*        resultCount,
    zBool           nofunc,
    ReportType      type
);
```

zReportHierObjCount

Loop over all modules in the database and count the total number of objects.

```
zUInt64 zReportHierObjCount(
    DB*         db,
    zBool       nofunc,
    ReportType type
);
```

Design Histogram

zReportHistogram

Report statistic (histogram) data for the given database.

```
struct zHistogram {
    zUInt64 cellCount;
    zUInt64 cellPlacement;
    zUInt64 deviceCount;
    zUInt64 devicePlacement;
    zUInt64 netCount;
    zUInt64 cellPinCount;
    zUInt64 devicePinCount;
    zUInt64 pinCount;
    zUInt64 portCount;
};
zBool zReportHistogram(
    DB*                 db,
    struct zHistogram** histogramListPtr
);
```

Design Statistics

zReportDesignStatistics

Report design statistics for the given database into a file.

```
zBool zReportDesignStatistics(
    DB*         db,
    const char* filename
);
```

Database Tools

Electrical Rule Checks

zFloatingNodes

The zFloatingNodes function loops over all modules and count the number of pins for each net. Power/Ground nets are skipped. All nets with exactly one pin are stored in the resultList.

```
void zFloatingNodes(
    DB*   db,
    OID** resultList
);
```

zFlatfloatNodes

The zFlatfloatNodes function loops over all signals and count the number of pins for each signal. Power/Ground nets are skipped. All signals with exactly one pin are stored in the resultList.

```
void zFlatfloatNodes(
    DB*   db,
    OID** resultList
);
```

zHeavyNodes

The zHeavyNodes function loops over all signals and count the number of pins for each signal. Power/Ground nets are skipped. The 10 biggest signals are stored in the resultList.

```
void zHeavyNodes(
    DB*   db,
    OID** resultList
);
```

zHeavyCR

The zHeavyCR function loops over all C or R and checks each value (capacitance or resistance) and collects 10 objects with the biggest values. The 10 biggest Cs and 10 biggest Rs are stored in the resultList.

```
void zHeavyCR(
    DB*   db,
    OID** resultList
);
```

zCCoupling

The zCCoupling function loops over all Cs and checks if they connect neither to power nor to ground that means they couple two data wires. The coupling Cs are stored in the resultList.

```
void zCCoupling(
    DB*   db,
    OID** resultList
);
```

zWrongBulk

The zWrongBulk function loops over all PMOS and NMOS transistors and checks if their bulks are connected to power and ground respectively. If not, then the transistor is stored in the resultList.

```
void zWrongBulk(
    DB*   db,
    OID** resultList
);
```

zMultiDriver

The zMultiDriver function loops over all signals and checks if a signal has more than one driver. The signals with more than one driver pin are stored in the resultList.

```
void zMultiDriver(
    DB*   db,
    OID** resultList
);
```

zZeroDriver

The zZeroDriver function loops over all signals and checks if a signal has no driver. The signals with no driver pin are stored in the resultList.

```
void zZeroDriver(
    DB*   db,
    OID** resultList
);
```

zOpenGate

The zOpenGate function loops over all transistors to find open/unconnected gate pins.

```
void zOpenGate(
    DB*   db,
    OID** resultList
);
```

Recreate Hierarchy

zCreateHier

The zCreateHier function loops over a flat design and recreates the hierarchy based on the instance names.

```
int zCreateHier(
    DB*         db,
    const char* hiersep
);
```

Analyze the Database

zFindGuessedSupplyNets

This function reports all nets that have been guessed by zOperGuessPower as a supply net.

```
void zFindGuessedSupplyNets(
    DB*   db,
    OID** resultList
);
```

Blocklevel Functions

The Blocklevel Object

zBlocklevelNew

```
zBlocklevel* zBlocklevelNew(DB* db, DB* blockDb);
```

zBlocklevelCreate

```
Module* zBlocklevelCreate(zBlocklevel *blocklevel, Module *module);
```

zBlocklevelExistsByPath

```
zBool zBlocklevelExistsByPath(zBlocklevel *blocklevel,
            const char** path, int plen);
```

zBlocklevelCreateByPath

```
zBool zBlocklevelCreateByPath(zBlocklevel *blocklevel,
            const char** path, int plen, struct OID* blocklevelModOid);
```

zBlocklevelBuildTree

```
zBool zBlocklevelBuildTree(zBlocklevel *blocklevel);
```

zBlocklevelFree

```
void zBlocklevelFree(zBlocklevel *blocklevel);
```

The Blocklevel View

Dump the Database

Internal Database Dump

Dump Options

The options can be specified using the name of the DUMPF_* macro.

```
typedef ce_UInt64 zDumpFlags;

/* No dump flags set */
#define DUMPF_NO_OPTIONS                        0x0000000000000000
#define DUMPS_NO_OPTIONS                        "NoFlags"

/* Sort all primitives */
#define DUMPF_SORT_PRIMS                        0x0000000000000001
#define DUMPS_SORT_PRIMS                        "SortPrimitives"

/* Sort all modules */
#define DUMPF_SORT_MODS                         0x0000000000000002
#define DUMPS_SORT_MODS                         "SortModules"

/* Sort all files */
#define DUMPF_SORT_FILES                        0x0000000000000004
#define DUMPS_SORT_FILES                        "SortFiles"

/* Sort all pins */
#define DUMPF_SORT_PINS                         0x0000000000000008
#define DUMPS_SORT_PINS                         "SortPins"

/* Sort all instances */
#define DUMPF_SORT_INSTS                        0x0000000000000010
#define DUMPS_SORT_INSTS                        "SortInstances"

/* Sort all nets */
#define DUMPF_SORT_NETS                         0x0000000000000020
#define DUMPS_SORT_NETS                         "SortNets"

/* Sort all ports */
#define DUMPF_SORT_PORTS                        0x0000000000000040
#define DUMPS_SORT_PORTS                        "SortPorts"

/* Sort all sub-ports */
#define DUMPF_SORT_SUBPORTS                     0x0000000000000080
#define DUMPS_SORT_SUBPORTS                     "SortSubPorts"

/* Dump ports by order number */
#define DUMPF_DUMP_PORT_ORDER                   0x0000000000000100
#define DUMPS_DUMP_PORT_ORDER                   "DumpPortOrder"

/* Dump flags by name */
#define DUMPF_DUMP_FLAGS                        0x0000000000000200
#define DUMPS_DUMP_FLAGS                        "DumpFlags"

/* Dump primFunction */
```

```
#define DUMPF_DUMP_FUNC                 0x0000000000000400
#define DUMPS_DUMP_FUNC                 "DumpFunc"

/* Dump spos information */
#define DUMPF_DUMP_SPOS                 0x0000000000001000
#define DUMPS_DUMP_SPOS                 "DumpSpos"

/* Dump attributes */
#define DUMPF_DUMP_ATTR                 0x0000000000002000
#define DUMPS_DUMP_ATTR                 "DumpAttr"

/* Prefix module/primitives names with uniq index */
#define DUMPF_DUMP_INDEX                0x0000000000004000
#define DUMPS_DUMP_INDEX                "DumpIndex"

 * Special "comparable" dump:
 * - the interface of Primitives and Operators (if a function is given)
 *   is printed without cellname and without portname (but port-order, like
 *   with the DumpPortOrder option);  portnames are also suppressed for
 *   Blackbox Modules (CellFUndefined).
 * - empty Modules and Operators are always printed as Primitives
 *   (without contents - see dumpContents.)
 * - the name of power/ground nets is printed as "0" or "1".
 * - unnamed instances ("i%d", "i_%d" or " I%d") is replaced by "-".
 * - map REDUCE primitive functions to their "normal" counterparts by ignoring
 *   the PortBus.
 * - map AND with only one input to a BUF.
#define DUMPF_DUMP_COMPARABLE           0x0000000000008000UL
#define DUMPS_DUMP_COMPARABLE           "Comparable"

/* Sort spos */
#define DUMPF_SORT_SPOS                 0x0000000000010000UL
#define DUMPS_SORT_SPOS                 "SortSpos"

/* Skip hidden ports */
#define DUMPF_NO_HIDE_PORT              0x0000000000020000UL
#define DUMPS_NO_HIDE_PORT              "NoHidePort"

/* Sort all net buses */
#define DUMPF_SORT_NET_BUSES            0x0000000000040000UL
#define DUMPS_SORT_NET_BUSES            "SortNetBuses"

/* Dump unconnected pins */
#define DUMPF_UNCONNECTED_PIN           0x0000000000080000UL
#define DUMPS_UNCONNECTED_PIN           "UnconnectedPin"

/* Skip contents of modules */
#define DUMPF_NO_CONTENT                0x0000000000100000UL
#define DUMPS_NO_CONTENT                "NoContent"

/* Sort attr */
```

```
#define DUMPF_SORT_ATTR                         0x0000000000200000UL
#define DUMPS_SORT_ATTR                         "SortAttr"

/* Skip primitives */
#define DUMPF_NO_PRIM                           0x0000000000400000UL
#define DUMPS_NO_PRIM                           "NoPrim"

/* Dump bus range */
#define DUMPF_DUMP_RANGE                        0x0000000000800000UL
#define DUMPS_DUMP_RANGE                        "DumpRange"

/* Dump filetype of spos files */
#define DUMPF_DUMP_SPOS_FILE_TYPE               0x0000000001000000UL
#define DUMPS_DUMP_SPOS_FILE_TYPE               "DumpSposFileType"

/* Not dump parasitic content */
#define DUMPF_NO_PARASITIC_CONTENT              0x0000000002000000UL
#define DUMPS_NO_PARASITIC_CONTENT              "NoParaContent"

/* Not dump root attr */
#define DUMPF_NO_ROOT_ATTR                      0x0000000004000000UL
#define DUMPS_NO_ROOT_ATTR                      "NoRootAttr"

/* Normalize long float attr */
#define DUMPF_NORM_FLOAT_ATTR                   0x0000000008000000UL
#define DUMPS_NORM_FLOAT_ATTR                   "NormalizeFloatAttr"

/* Dump only module names */
#define DUMPF_MODULE_ONLY                       0x0000000010000000UL
#define DUMPS_MODULE_ONLY                       "ModuleOnly"

/* Special case dump flat attr tree */
#define DUMPF_DUMP_FLAT_ATTR                    0x0000000020000000UL
#define DUMPS_DUMP_FLAT_ATTR                    "DumpFAttr"

/* Unify the names of auto generated primitives */
#define DUMPF_UNIFY_AUTOGEN_PRIM_NAMES          0x0000000080000000UL
#define DUMPS_UNIFY_AUTOGEN_PRIM_NAMES          "UnifyAutogenPrimNames"

/* Remove nets which are only connected to one port/pin */
#define DUMPF_REMOVE_DANGLING_NETS              0x0000000100000000UL
#define DUMPS_REMOVE_DANGLING_NETS              "RemoveDanglingNets"

/* Sort all elements */
#define DUMPF_SORT                                                  ( \
                                                DUMPF_SORT_SPOS       | \
                                                DUMPF_NO_HIDE_PORT    | \
                                                DUMPF_SORT_NET_BUSES  | \
                                                DUMPF_SORT_PORTS      | \
                                                DUMPF_SORT_NETS       | \
                                                DUMPF_SORT_INSTS      | \
```

```
                                        DUMPF_SORT_PRIMS    | \
                                        DUMPF_SORT_MODS     | \
                                        DUMPF_SORT_PINS     | \
                                        DUMPF_SORT_ATTR     | \
                                        DUMPF_SORT_FILES    )
#define DUMPS_SORT                      "Sort"

/* Built-in default dump flags */
#define DUMPF_DEFAULT                                       ( \
                                        DUMPF_SORT          | \
                                        DUMPF_DUMP_FLAGS    | \
                                        DUMPF_DUMP_FUNC     | \
                                        DUMPF_DUMP_SPOS     | \
                                        DUMPF_DUMP_ATTR     )
#define DUMPS_DEFAULT                   "Default"
```

zDumpDatabase

Dump the contents of the database to the a file with the given name. The output file can be compressed using the IOType option. Return false and set the last error message if something went wrong. If is cellPattern is not NULL, only cells matching the given pattern are dumped. The dump details can be controlled using the dump flags.

```
ce_Bool zDumpDatabase(
    DB*         db,
    const char *filename,
    ce_IOType   ioType,
    const char *cellPattern,
    zDumpFlags  flags
);
```

zDump

Map the legacy function name.

```
#define zDump(db, fname, type, cellPattern, flags) \
    zDumpDatabase(db, fname, type, cellPattern, flags)
```

zDumpDatabase_OptionString

Dump the contents of the database to the a file with the given name. The output file can be compressed using the IOType option. Return false and set the last error message if something went wrong. If is cellPattern is not NULL, only cells matching the given pattern are dumped. The dump details can be controlled using the stringFlags: dump flag names separated by the pipe character, e.g. DumpFlags|DumpAttr.

```
ce_Bool zDumpDatabase_OptionString(
    DB*         db,
    const char *filename,
    ce_IOType   ioType,
    const char *cellPattern,
    const char *stringFlags
);
```

zDumpOpt

Map the legacy function name.

```
#define zDumpOpt(db, fname, type, cellPattern, strflags) \
    zDumpDatabase_OptionString(db, fname, type, cellPattern, strflags)
```

zDumpDatabase_OptionArray

Dump the contents of the database to the a file with the given name. The output file can be compressed using the compress option. Return false and set the last error message if something went wrong. If is cellPattern is not NULL, only cells matching the given pattern are dumped. The dump details can be controlled using the flagNames string array: A list of flag names, e.g. [0] = DumpFlags, [1] = DumpAttr.

```
ce_Bool zDumpDatabase_OptionArray(
    DB          *db,
    ce_IOType    ioType,
    const char  *cellPattern,
    const char **flagNames,
    const char  *filename
);
```

Database Writer

Write zdb data as either Tcl or a standard netlist format like Verilog or Spice.

Write Standard Netlist Formats

zWriteTcl

Write the db as Tcl '$db load ...' commands to a file named fname.

```
zBool zWriteTcl(
    DB*         db,
    const char* fname,
    IOType      type,
    zBool       doTSort,
    zBool       doSpos,
    zBool       doAttrs,
    zBool       doFlags,
    zBool       doFlat,
    zBool       doMangle,
    zBool       doSposIdx,
    zBool       doTableAlloc,
    zBool       doContent,
    int         comments,
    const char* cellPattern
);
```

zWriteVerilog

Write the database as a Verilog file.

```
zBool zWriteVerilog(
    DB*         db,
    const char* fname,
    IOType      type,
    zBool       named,
    int         comments,
    zBool       objcomments,
    zBool       preserveNet,
    PortFlags   ignorePort,
    CellFlags   ignoreCell,
    CellFlags   ignoreRCell,
    InstFlags   ignoreInst,
    zBool       ignoreAutoGen,
    zBool       implementFunction,
    zBool       originalName,
    zBool       writeCellDefine
);
```

zWriteSpice

Write the db as a Spice file.

```
zBool zWriteSpice(
    DB*         db,
    const char* fname,
    IOType      type,
    int         comments,
    const char* valueNetNames[4],
    zBool       deviceAsSubckt,
    zBool       subcktx,
    zBool       noModel,
    zBool       addDotEnd,
    zBool       originalName
);
```

zWriteSpef

Write the db as a Spef file.

```
zBool zWriteSpef(
    DB*          db,
    const char*  fname,
    IOType       type,
    int          comments,
    zBool        nameMap,
    zBool        ignoreEmpty,
    Module*      top,
    const char*  design,
    const char*  program,
    const char*  version,
    const char*  vendor,
    const char*  date,
    const char*  designflow,
    const char*  divider,
    const char*  delimiter,
    const char*  busdelimiter,
    const char*  timescale,
    const char*  timeunit,
    const char*  capscale,
    const char*  capunit,
    const char*  resscale,
    const char*  resunit,
    const char*  indscale,
    const char*  indunit,
    const char** powernets,
    const char** gndnets,
    const char** onlys,
    zBool        renameLocals,
    zBool        noEsc
);
```

zWriteDspf

Write the db as a DSPF file.

```
zBool zWriteDspf(
    DB*          db,
    const char*  fname,
    IOType       type,
    int          comments,
    zBool        ignoreEmpty,
    Module*      top,
    const char*  design,
    const char*  program,
    const char*  version,
    const char*  vendor,
    const char*  date,
    const char*  divider,
    const char*  delimiter,
    const char*  busdelimiter,
    const char** gndnets,
    const char** onlys,
    zBool        noEsc
);
```

Generic Write Interface

zWriteInfo

Write info struct.

```
struct zWriteFunctions;
struct zWriteInfo {
    IOWriter*               wrt;
    const char*             fname;
    DB*                     db;
    IOType                  type;
    zBool                   doSpos;
    zBool                   doAttrs;
    zBool                   doFlags;
    zBool                   doFlat;
    zBool                   doMangle;
    zBool                   doStopCell;
    zBool                   ok;
    zBool                   dbgRecord;
    int                     doComments;
    int                     mangleId;
    char                    unused[4];
    struct zWriteFunctions* functions;

    struct zStrTab          mangleMap;
    struct StrSpace         strspace;
    char                    buf[100];
};
```

zWriteMangleName

Mangle object names.

```
char* zWriteMangleName(struct zWriteInfo* writeInfo, const char* name);
```

zWriteGzPutS

Write a string to the output file.

```
void zWriteGzPutS(
    struct zWriteInfo*,
    const char*
);
```

zWriteGzPutC

Write a char to the output file.

```
void zWriteGzPutC(
    struct zWriteInfo*,
    char
);
```

zWriteGzPutLen

Write a string to the output file.

```
void zWriteGzPutLen(
    struct zWriteInfo*,
    int                 len,
    const char*
);
```

zWriteGzPutI

Write a decimal number to the output file.

```
void zWriteGzPutI(
    struct zWriteInfo* writeInfo,
    int                 i
);
```

zWriteGzPutL

Write a decimal number to the output file.

```
void zWriteGzPutL(
    struct zWriteInfo* writeInfo,
    zLong               i
);
```

zWriteGzPutU

Write a decimal number to the output file.

```
void zWriteGzPutU(
    struct zWriteInfo* writeInfo,
    unsigned            i
);
```

zWriteGzPutUL

Write a decimal number to the output file.

```
void zWriteGzPutUL(
    struct zWriteInfo* writeInfo,
    zULong             i
);
```

zWriteGzPutULL

Write a decimal number to the output file.

```
void zWriteGzPutULL(
    struct zWriteInfo* writeInfo,
    zUInt64             i
);
```

zWriteVHDL

Write the db as VHDL netlist.

```
zBool zWriteVHDL(
    DB*         db,
    const char* fname,
    IOType      type,
    int         comments,
    zBool       implementFunction
);
```

zWriteLiberty

Write the db as Liberty library.

```
zBool zWriteLiberty(
    DB*         db,
    const char* fname,
    IOType      type,
    const char* libraryName,
    zBool       allCells,
    int         generatedDataRepeat
);
```

zWriteGzPutD

Write a decimal number to the output file.

```
void zWriteGzPutD(
    struct zWriteInfo* writeInfo,
    double             d
);
```

zWriteGzPutX

Write a decimal number to the output file.

```
void zWriteGzPutX(
    struct zWriteInfo* writeInfo,
    int                i
);
```

zWriteBinR

Write a record marker to the output file.

```
void zWriteBinR(
    struct zWriteInfo*,
    char
);
```

zWriteBinC

Write a char to the output file.

```
void zWriteBinC(
    struct zWriteInfo*,
    char
);
```

zWriteBinI

Write int to the output file.

```
void zWriteBinI(
    struct zWriteInfo*,
    int
);
```

zWriteBinU

Write unsigned to the output file.

```
void zWriteBinU(
    struct zWriteInfo*,
    unsigned
);
```

zWriteBinL

Write long to the output file.

```
void zWriteBinL(
    struct zWriteInfo*,
    long
);
```

zWriteBinUL

Write unsigned long to the output file.

```
void zWriteBinUL(
    struct zWriteInfo*,
    zULong
);
```

zWriteBinLL

Write long long to the output file.

```
void zWriteBinLL(
    struct zWriteInfo*,
    long long
);
```

zWriteBinULL

Write long long to the output file.

```
void zWriteBinULL(
    struct zWriteInfo*,
    unsigned long long
);
```

zWriteBinS

Write a string to the output file.

```
void zWriteBinS(
    struct zWriteInfo*,
    const char*
);
```

zWriteBinSLen

Write a beginning of string to the output file.

```
void zWriteBinSLen(
    struct zWriteInfo*,
    int len,
    const char*
);
```

zWriteSposComment

Write comment with spos to the corresponding module.

```
void zWriteSposComment(
    struct zWriteInfo* writeInfo,
    struct Spos*       spos,
    const char*        commentChar
);
```

Write Functions

Function pointer that are called by zWrite to output objects.

```
struct zWriteFunctions {
    void (*writeHeader)            (struct zWriteInfo*);
    void (*writeFooter)            (struct zWriteInfo*);
    void (*writeDatabaseAttribute) (struct zWriteInfo*, char* attr);
    void (*writeSposFile)          (struct zWriteInfo*, Sfile* sfile);
    void (*writeSposLineInfo)      (struct zWriteInfo*, Sfile* sfile);
    void (*writePrimitiveBegin)    (struct zWriteInfo*, Primitive* prim);
    void (*writePrimitiveEnd)      (struct zWriteInfo*, Primitive* prim);
    void (*writeModuleBegin)       (struct zWriteInfo*, Module* mod,
                                                        zBool ignoreContent);
    void (*writeModuleEnd)         (struct zWriteInfo*, Module* mod);
    void (*writeInterfaceBegin)    (struct zWriteInfo*);
    void (*writePort)              (struct zWriteInfo*, Cell* cell, Port* p);
    void (*writePortBus)           (struct zWriteInfo*, Cell* cell, PortBus* p);
    void (*writeInterfaceEnd)      (struct zWriteInfo*);
    void (*writeInstListBegin)     (struct zWriteInfo*);
    void (*writeInst)              (struct zWriteInfo*, Module* mod, Inst* i);
    void (*writeInstListEnd)       (struct zWriteInfo*);
    void (*writeNetListBegin)      (struct zWriteInfo*);
    void (*writeNetBegin)          (struct zWriteInfo*, Module* mod, Net* n);
    void (*writePinRef)            (struct zWriteInfo*, Module* mod, PinRef* p);
    void (*writeNetEnd)            (struct zWriteInfo*, Module* mod, Net* n);
    void (*writeNetListEnd)        (struct zWriteInfo*);
    void (*writeNetBusListBegin)   (struct zWriteInfo*);
    void (*writeNetBus)            (struct zWriteInfo*, Module* mod, NetBus* n);
    void (*writeNetBusListEnd)     (struct zWriteInfo*);
    void (*writeFlatHicol)         (struct zWriteInfo*, const char* oid,
                                                        HiCol* hicol);
    void (*writeFlatFlags)         (struct zWriteInfo*, const char* oid,
                                                        FNodeFlags flags);
    void (*writeFlatAttrs)         (struct zWriteInfo*, const char* oid,
                                                        FAttr* attrs, int);
    void (*writeFlatOOMRs)         (struct zWriteInfo*, const char* pinOid,
                                                        const char* netOid);
    void (*writeFlatId)            (struct zWriteInfo*,const char* oid,zUInt64);
    void (*writeVirtualObject)     (struct zWriteInfo*, VirtualObject* obj);
    void (*writeTmpTop)            (struct zWriteInfo*, Module* tmpTop);
    void (*writeFlatHierTree)      (struct zWriteInfo*);
    void (*writeModuleContent)     (struct zWriteInfo*, Module* mod);
    void (*writeModuleFlat2)       (struct zWriteInfo*, Module* mod);
    void (*writeEof)               (struct zWriteInfo*);
};
```

zWrite

Generic structural ZDB output interface.

```
zBool zWrite(
    struct zWriteInfo*      writeInfo,
    DB*                     db,
    const char*             fname,
    IOType                  type,
    zBool                   doTSort,
    zBool                   doSpos,
    zBool                   doAttrs,
    zBool                   doFlags,
    zBool                   doFlat,
    zBool                   doMangle,
    zBool                   doStopCell,
    int                     doComments,
    const char*             cellPattern,
    struct zWriteFunctions* functions
);
```

Write and Read ASCII Format

ASCII Tokens

Tokens to identify database object.

```
#define AsciiMagicNum (0x12345678)

enum AsciiToken {
    AsciiTNull          = '\0',
    AsciiTRec           = '\n',
    AsciiTEsc           = 0x1B,
    AsciiTEof           = 0x03,

    AsciiTMagic         = '*',
    AsciiTVersion       = 'A',
    AsciiTPlatform      = '+',
    AsciiTAttr          = 'B',
    AsciiTSfile         = 'C',
    AsciiTSline         = 'D',
    AsciiTSpos          = 'E',
    AsciiTNullType      = 'F',
    AsciiTPort          = 'G',
    AsciiTPbus          = 'H',
    AsciiTPrim          = 'I',
    AsciiTMod           = 'J',
    AsciiTTopMod        = 'K',
    AsciiTPara          = 'L',
    AsciiTInst          = 'M',
    AsciiTInstInfo      = 'N',
    AsciiTPin           = 'O',
    AsciiTNet           = 'P',
```

```
    AsciiTNetInfo       = 'Q',
    AsciiTPinRef        = 'R',
    AsciiTPortRef       = 'S',
    AsciiTNbus          = 'T',
    AsciiTNbusMem       = 'U',
    AsciiTHiCol         = 'V',
    AsciiTVirtualObject = 'W',
    AsciiTPinInternal   = 'X',

    AsciiTFlatTopNode   = 'a',
    AsciiTFlatAttrName  = 'b',
    AsciiTFlatNamNode   = 'c',
    AsciiTFlatIdxNode   = 'd',
    AsciiTFlatAttr      = 'e',

    AsciiTFlatNetSeg    = 'h',
    AsciiTFlatPinOOMR   = 'i',
    AsciiTFlatNetOOMR   = 'j',

    AsciiTSymlibHeader  = 'k',
    AsciiTSymlibIO      = 'l',
    AsciiTSymlibSpice   = 'm',
    AsciiTSymlibAmap    = 'n',
    AsciiTSymlibSym     = 'o',
    AsciiTSymlibComment = 'p',
    AsciiTSymdef        = 'q',

    AsciiTTmpTop        = 'r',

    AsciiTHeader        = 't',
    AsciiTInstPara      = 'v',
    AsciiTCellCnt       = 'w',

    AsciiTParaModPos    = 'x',
    AsciiTModPos        = 'y',

    AsciiTSeekEnd       = 'z',

    AsciiTReloadPara    = '1',
    AsciiTReload        = '2',
    AsciiTFlatModPos    = '3',
    AsciiTSfilePos      = '4',

    AsciiTFlatHierNode     = '5',
    AsciiTFlatHierInstNode = '6',
    AsciiTFlatPrimInstNode = '7',
    AsciiTFlatChildEnd     = '8',

    AsciiTFlatIdNode       = '9'
};
```

zWriteAscii

Write the database as an ASCII file.

```
zBool zWriteAscii(
    DB*          db,
    const char*  fname,
    IOType       type
);
```

zWriteAsciiFlatNodeInfoAdd

add fnode to uniq number mapping. Internal use.

```
void zWriteAsciiFlatNodeInfoAdd(
    struct zWriteInfo* writeInfo,
    struct FNode* fnode
);
```

zWriteAsciiFlatModInfoGet

get fnode to uniq number mapping. Internal use.

```
zULong zWriteAsciiFlatNodeInfoGet(
    struct zWriteInfo* writeInfo,
    struct FNode* fnode
);
```

zWriteAsciiFlatModInfoAdd

add module to fnode number mapping. Internal use.

```
void zWriteAsciiFlatModInfoAdd(
    struct zWriteInfo* writeInfo,
    Module* mod,
    struct FTopNode* topNode,
    struct FNode* n
);
```

zWriteAsciiFlatModInfoGet

get module to fnode number mapping. Internal use.

```
struct FlatModInfo* zWriteAsciiFlatModInfoGet(
    struct zWriteInfo* writeInfo,
    Module* mod
);
```

zReadAscii

Read an ASCII file and fill a database.

```
typedef struct {
    int major;
    int minor;
    int parserBits;
} zAsciiHead;


zBool zReadAscii(
    DB*         db,
    const char* fname,
    IOType      type,
    zAsciiHead* head,
    zBool       quick
);


zBool zReadAscii_autoPopulateModule(
    DB*         db,
    Module*     mod
);

zBool zReadAscii_autoPopulateModuleOID(
    DB*         db,
    const OID*  oid
);
```

zReadAscii_PopulateModule

Populate the database with the contents of the given module.

```
zBool zReadAscii_PopulateModule(
    DB*         db,
    const char* moduleName,
    zBool       parasitic,
    zBool       withFlat
);
```

zReadAscii_PopulateFlat

Populate the database with the flat data of the given module.

```
zBool zReadAscii_PopulateFlat(
    DB*         db,
    const char* moduleName
);
```

zReadAscii_PopulateSourceFileInfo

Incrementally fill the source file information in the database for the source file with the given index.

```
zBool zReadAscii_PopulateSourceFileInfo(
    DB* db,
    int sFileIdx
);
```

zTestAscii

Internal use.

```
void zTestAscii(
    DB* db
);
```

Interface between Tcl and ZDB

Get the Database by Name

zT_GetDbFromName

Get DB* struct for a db tcl cmd name.

```
DB* zT_GetDbFromName(
    Tcl_Interp* interp,
    CONST char* name
);
```

Work with OIDs

zT_SetOIDObj

The Tcl_Obj is made an OID-type by setting its internal representation.

```
void zT_SetOIDObj(
    DB*        db,
    Tcl_Obj*   theObj,
    const OID* source
);
```

zT_GetOIDFromObj

Get the OID from the given Tcl_Obj.

```
int  zT_GetOIDFromObj(
    DB*            db,
    Tcl_Interp*    interp,
    Tcl_Obj*,
    OID*           resultPtr
);
```

zT_GetOIDListFromObj

Get an OID list from the given Tcl_Obj.

```
int zT_GetOIDListFromObj(
    DB*            db,
    Tcl_Interp*    interp,
    Tcl_Obj* CONST,
    OID**
);
```

zT_GetUnsharedOID

Make *varP point to an unshared (i.e. writable) OID-type Tcl_Obj and return the OID pointer.

```
OID* zT_GetUnsharedOID(
    DB*         db,
    Tcl_Interp* interp,
    Tcl_Obj**   varP,
    const OID*
);
```

Set the Tcl Result

zT_SetOIDResult

Set an OID as the Tcl object result.

```
void zT_SetOIDResult(
    DB*         db,
    Tcl_Interp* interp,
    const OID*  result
);
```

zT_GetObjFromOID

Get the Tcl object from an OID.

```
Tcl_Obj* zT_GetObjFromOID(
    DB*        db,
    const OID* oid
);
```

zT_ErrorSetResult

Set an Tcl error result.

```
void zT_ErrorSetResult(
    Tcl_Interp* interp,
    const char*,
    Tcl_Obj*,
    const char*
);
```

The OEM Specific Header Files

FlexNet License Check for OEM Customers

Functions to Interact with the License Sub-System

zLicenseCheckoutFeature

Check out a the given FlexNet feature. In case of an error the error message can be retrieved with zGetError().

```
zBool zLicenseCheckoutFeature(
    const char* featureName
);
```

zLicenseSetParserBit

Set parser bit to identify the creator of a database.

```
zBool zLicenseSetParserBit(
    DB*         zdb,
    const char* featureName
);
```

Tutorials

Command Line Tools Tutorial

This tutorial demonstrates the usage of the command line tools. For more information, please consult to the Command Line Arguments documentation for each tool in the `doc/parser` directory.

Overview

The use of the command line tools allows many combinations of different input descriptions for library cells and their graphical representations. This tutorial shows the following examples:

VHDL library example	Uses **vhdl2vdb** to create a precompiled library and **rtl2zdb** to use this library in a VHDL design.
edif2zdb example	Uses **edif2zdb** to generate a ZDB Binfile from an EDIF design.
Synopsys Liberty example	Uses **liberty2zdb** to create library cells from Synopsys Liberty file format and **verilog2zdb** to parse a Verilog design.
sym2zdb and zdb2sym example	Shows the usage of **sym2zdb** and **zdb2sym**.

VHDL Library Example

When processing a VHDL design the libraries normally are precompiled in the installation tree. If user defined libraries are needed, they can be created with the **vhdl2vdb** tool.
As an example we compile the `std_logic_1164.vhd` into a local library and use it for a small example design `delta0.vhd` parsed with **rtl2zdb**.

```
vhdl2vdb -vhdl93 -library ieee -unit std_logic_1164 \
         -vhdlLibPath ./mylibs vhdl_packages/vhdl93/std_1164.vhd
rtl2zdb  -vhdlLibPath ./mylibs demo/rtl/delta0/delta0.vhd -o design.zdb
rtlvisionpro -binfile design.zdb
```

Edif2zdb Example

In this example we create a ZDB Binfile from an EDIF design, using **edif2zdb**. The generated ZDB Binfile `design.zdb` is then used for display in RTLvision PRO.

```
edif2zdb -o design.zdb -symlib demo/edif/generic.sym demo/edif/gl85.edf
rtlvisionpro -binfile design.zdb
```

Synopsys Liberty File Example

In this example we create a library `lib.zdb` from the Synopsys .lib file `example.lib` with the help of **liberty2zdb**. It is referenced in a Verilog design `example.v` parsed with **verilog2zdb**. The generated

design output file `design.zdb` is then used for display in RTLvision PRO.

```
liberty2zdb demo/api/cust4/example.lib -o lib.zdb
verilog2zdb -binlib lib.zdb demo/api/cust4/example.v -o design.zdb
rtlvisionpro -binfile design.zdb
```

Sym2zdb and Zdb2sym Example

In this example we create a library with cells (and symbol graphics) by using **sym2zdb**. The generated binary library file `lib.zdb` can then be used either with RTLvision PRO or a batch parser to add a symbol shape to a primitive or module defined in the input file.

```
sym2zdb -o lib.zdb symutils/symlib/generic.sym
```

The next example shows the extraction of symbol graphics from an existing binary database file. Here the `lib.zdb`, which was created in the code above, is used to create the symlib file `lib.sym`.

```
zdb2sym -o lib.sym lib.zdb
```

Mixing Verilog and VHDL

RTLvision PRO allows for mixing Verilog and VHDL design files.

All used example files can be found in the `demo/rtl/mixed` folder in the RTLvision PRO directory.

Instantiating a VHDL Entity in a Verilog Module

A VHDL entity can be instantiated like any other verilog module. However, since a VHDL entity can have multiple architectures, the required architecture name can be specified as `entity_name(arch_name)`. If no architecture is specified, the last defined architecture is chosen.

Example

A VHDL full adder with two architectures (`structural` and `dataflow`):

```
-- adder.vhdl

entity full_adder is
    port(a, b, cin: in  bit;
         sum, cout: out bit);
end full_adder;

architecture structural of full_adder is
component half_adder
    port(a, b:      in  bit;
         sum, cout: out bit);
end component;

signal u0_cout, u0_sum, u1_cout: bit;

begin
    u0: half_adder port map (a => a,      b => b,   sum => u0_sum, cout => u0_cout);
    u1: half_adder port map (a => u0_sum, b => cin, sum => sum,    cout => u1_cout);

    cout <= u0_cout or u1_cout;
end structural;

architecture dataflow of full_adder is
begin
  sum <=  ((not cin) and (not a) and      b ) or
          ((not cin) and      a  and (not b)) or
          (     cin  and (not a) and (not b)) or
          (     cin  and      a  and      b );

  cout <= ((not cin) and      a  and      b ) or
          (     cin  and (not a) and      b ) or
          (     cin  and      a  and (not b)) or
          (     cin  and      b  and      b );
end dataflow;
```

A Verilog top module instantiating three full adders: the first instantiation explicitly selects the `structural` architecture (please note the backslash-escaped module name `\full_adder(structural)` to avoid Verilog syntax errors), the second instantiation explicitly selects the `dataflow` architecture, and the third instantiation does not specify any architecture, but gets the `dataflow` variant, because it is declared last in the VHDL file.

```
// top.v

module top (a, b, cin, sum0, cout0, sum1, cout1, sum2, cout2);
    input a, b, cin;
    output sum0, cout0;
    output sum1, cout1;
    output sum2, cout2;

    // explicitly select the 'structural' architecture
    \full_adder(structural)  adder_explicitly_struct (a, b, cin, sum0, cout0);

    // explicitly select the 'dataflow' architecture
    \full_adder(dataflow)  adder_explicitly_dataflow (a, b, cin, sum1, cout1);

    // just select the last defined architecture
    full_adder adder_last_defined (a, b, cin, sum2, cout2);
endmodule
```

The example can be loaded into RTLvision PRO with

```
rtlvisionpro -vhdl2008 demo/rtl/mixed/adder.vhdl -sysverilog demo/rtl/mixed/top.v
```

Instantiating a Verilog Module in a VHDL Architecture

A Verilog module can be instantiated either by a direct entity instantiation or through a component or configuration, just like a VHDL entity.

Example

A full adder in Verilog:

```
// adder.v

module full_adder (a, b, cin, sum, cout);
    input a, b, cin;
    output sum, cout;

    assign sum = a ^ b ^ cin;
    assign cout = ((a ^ b) & cin) | (a & b);
endmodule
```

A VHDL entity instantiating two Verilog full adders: the first instantiation is a direct instantiation that uses the full Verilog module name including its library (`work.full_adder`), for the second instantiation a new component (`component full_adder`) is defined in the architecture.

```
-- top.vhdl

entity top is
    port(a, b, cin: in  bit;
         sum0, cout0: out bit;
         sum1, cout1: out bit);
end top;

architecture arch of top is
    -- New component
    component full_adder is
        port(a, b, cin: in  bit;
             sum, cout: out bit);
    end component full_adder;
begin
    -- Direct entity instantiation
    a1 : entity work.full_adder
        port map (a => a, b => b, cin => cin, sum => sum0, cout => cout0);

    -- Using component instantiation
    a2 : full_adder
        port map (a => a, b => b, cin => cin, sum => sum1, cout => cout1);
end arch;
```

The example can be loaded into RTLvision PRO with

```
rtlvisionpro -sysverilog demo/rtl/mixed/adder.v -vhdl2008 demo/rtl/mixed/top.vhdl
```

Clock Domain Analyzer Tutorial

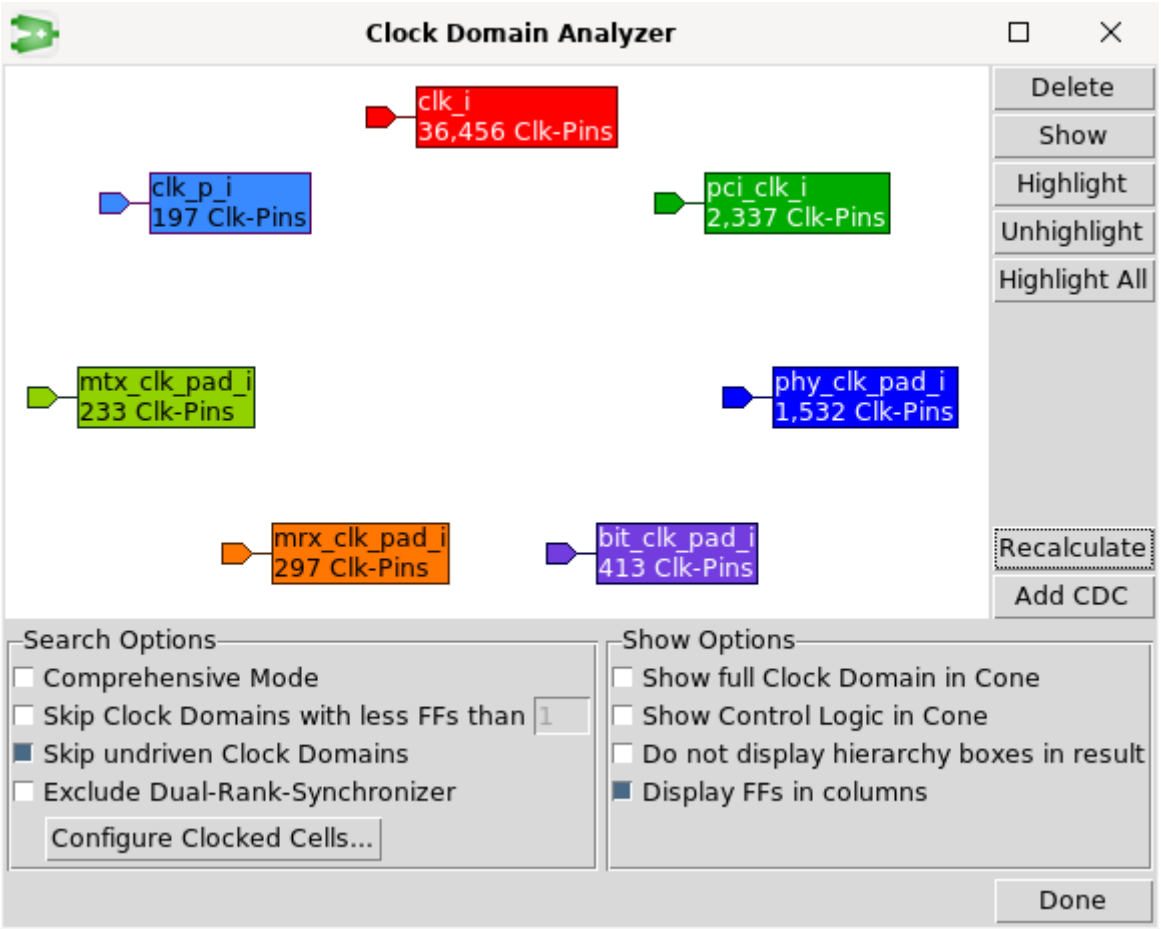
This tutorial demonstrates the usage of the Clock Domain Analyzer window by running an example from the `demo/rtl/` directory. For more general information please consult the Quickstart guide or the Reference Manual.

The following steps are based on the design stored in `demo/rtl/wb_sys/`. The design files are listed in the predefined fileset `demo/rtl/wb_sys/conmax_top.f`. If you have trouble loading this fileset please consult the Quickstart guide or the Open Files manual first.

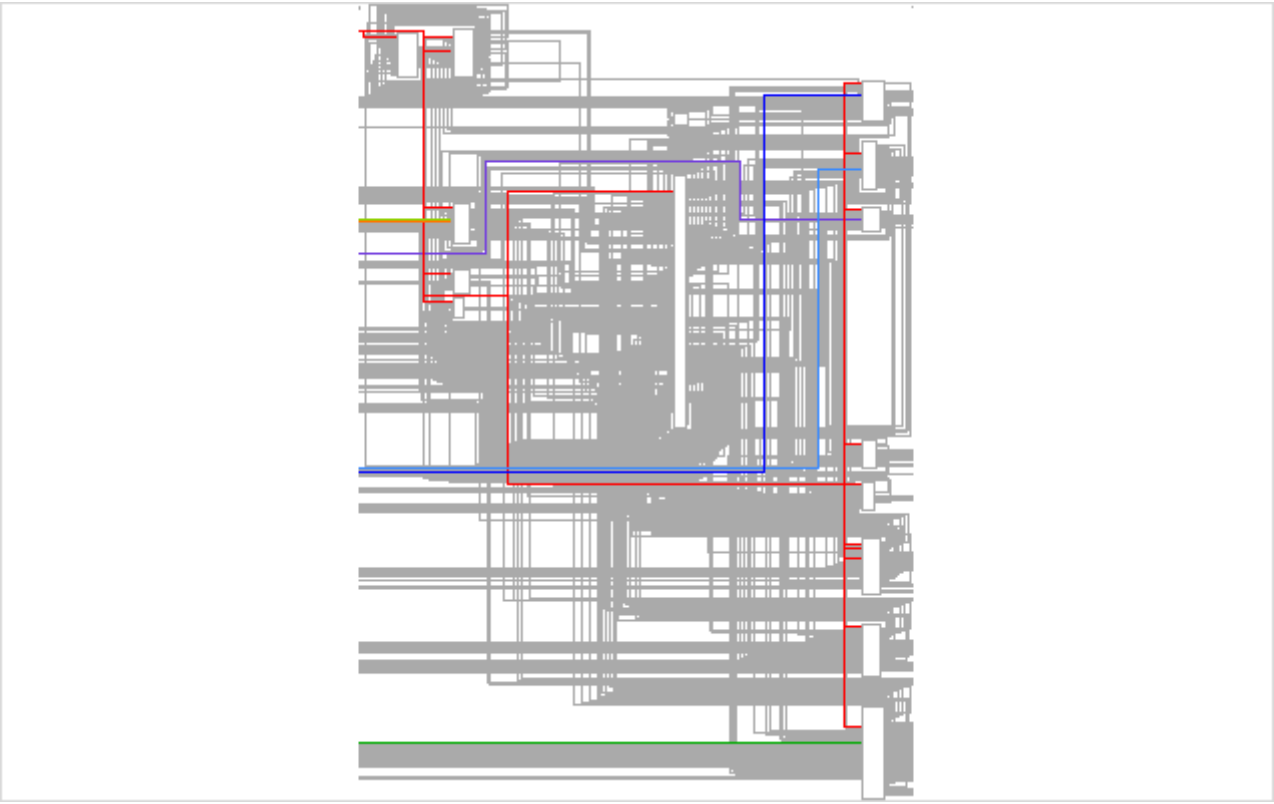
Clock Domain

After loading the fileset `conmax_top.f` from the directory `demo/rtl/wb_sys/` open the Clock Domain Analyzer window from the Tools main menu. Since the clock domain algorithms rely on correct clocked cells information, please revise the Clocked Cells settings. If the clocked cells settings are correct then you can see the calculated clock domain information in a graphical way. Depending on the design size this can take some time. Each clock domain is represented by a box with the total number of connections to clocked cells inside.

Use the "Highlight All" button to highlight each clock domain in a different color.



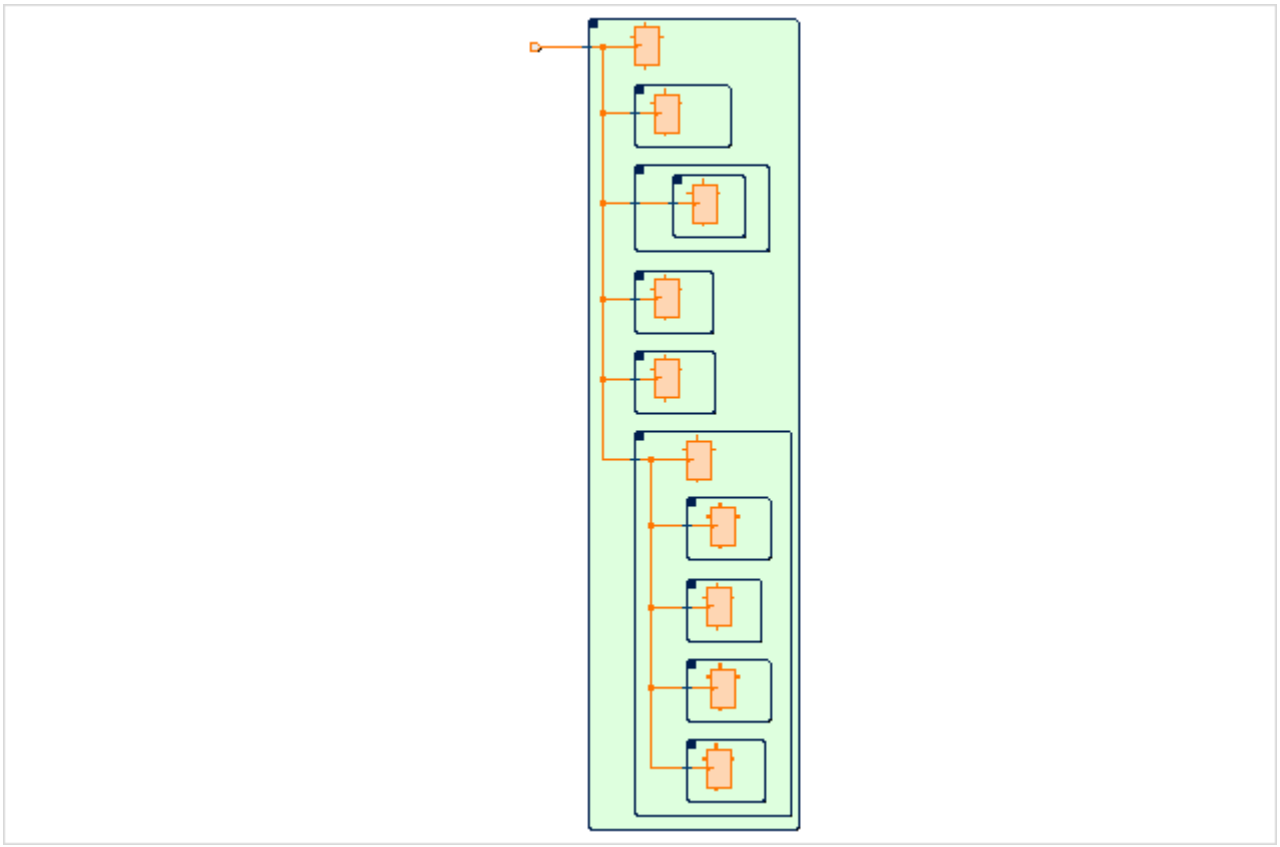
If you go to the Schematic window and turn on the greymode then it is very easy to identify the clock signals.



You can double click on the graphical representation of a clock domain to load and display the clock

domain in the Cone window. The following image shows the top part of the displayed clock domain.

In each hierarchy level only one representative of a clocked cell is displayed and the total count of clocked cells in each hierarchy level is annotated at this cell.

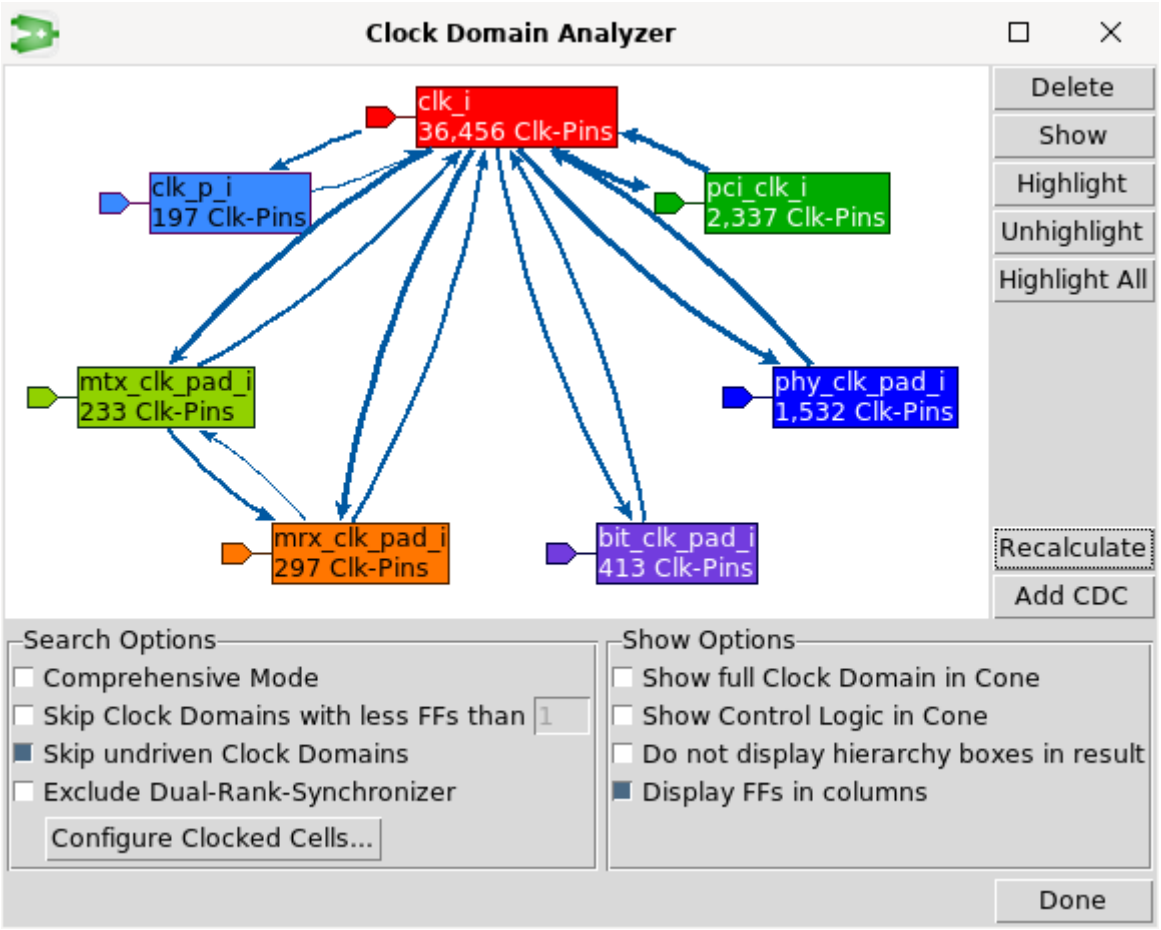


Clock Domain Crossings

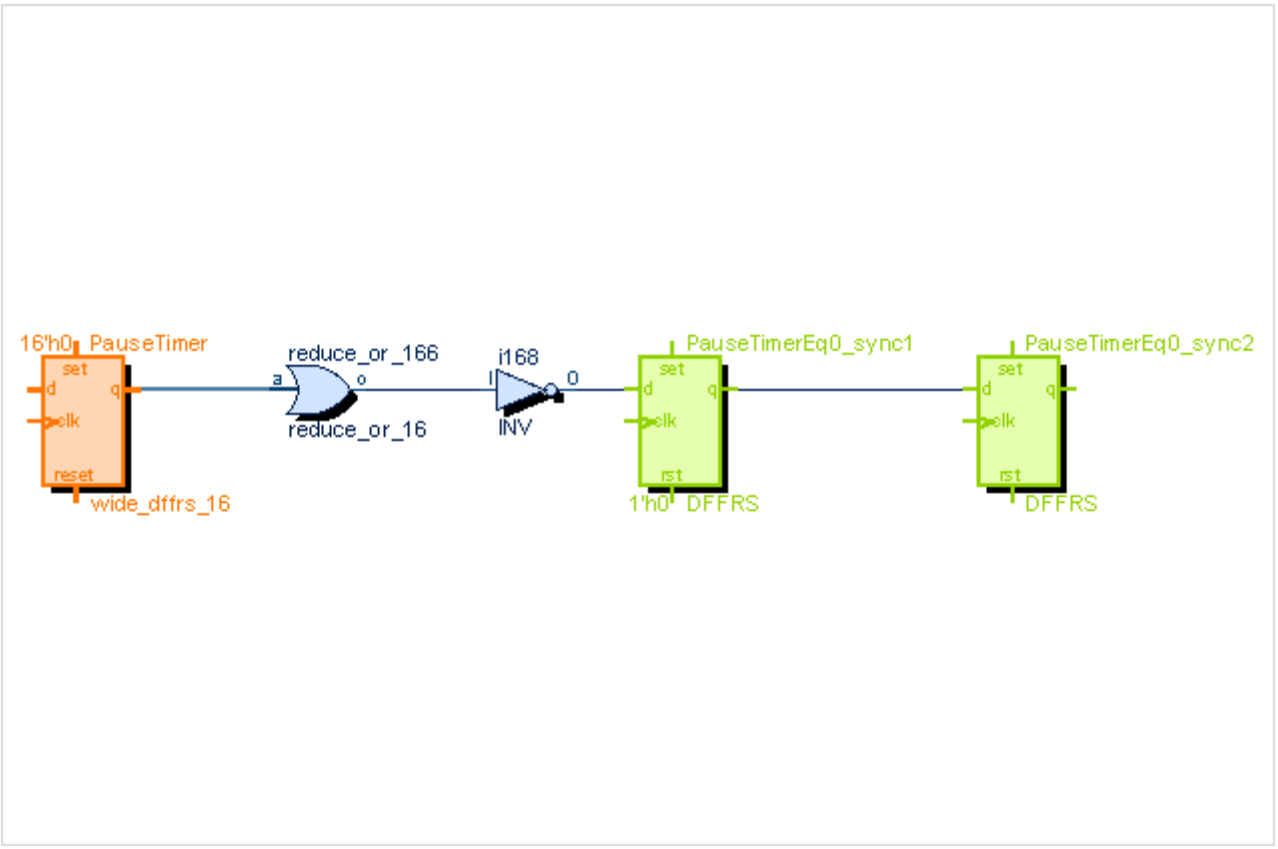
A click on the "Add CDC" button allows you to find all crossings between the various clock domains.

You can select two or more clock domains to limit the number of source and target domains. Multiple objects can be selected by holding down the `Ctrl` key.

The result will look like the following image.

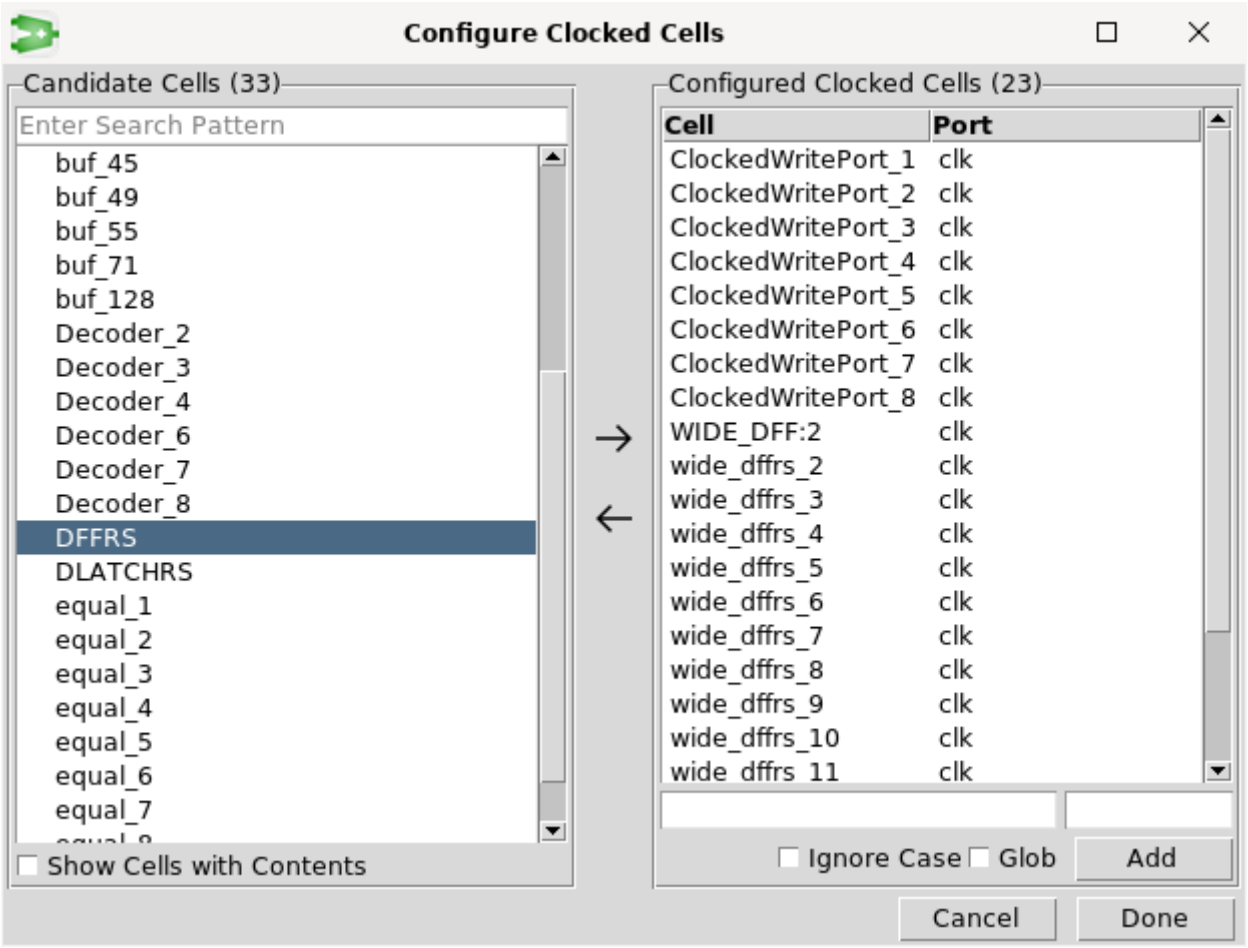


The transitions between the domains are indicated by arrows. An arrow with a thin line means there is only a little bit of logic between the two domains. An arrow with a thick line indicates that there is a lot of logic between the two domains.



The nets connecting the logic between two domains can also be highlighted. For this purpose, double click on the transition arrow of interest. This will load and show the logic in the Cone window. This transition arrow will be shown as a dotted line afterwards. In the image above the transition between 'mrx_clk_pad_i domain' and 'mtx_clk_pad_i domain' is highlighted in purple. Now you can double click on the output pin 'q' of the light green highlighted Flip-Flop named 'PauseTimerEq0_sync1' on the right side to see that this transition is synchronized by a second Flip-Flop of the same color. The contents of your Cone window will look like the above image.

Clocked Cells



This section explains how to manually specify or add additional clocked cells. When reading RTL, the parser knows which cells are clocked and automatically adds them to the list of clocked cells. You can use this dialog to specify additional cells as clocked cells.

If you read a gatelevel netlist then the parser does not know anything about clocked cells. In that case you have to use this dialog to specify all clocked cells and the corresponding clocked ports in order to use the Clock Domain Analyzer window.

Symlib Tutorial

Create and use your own symbol library.

You can provide the RTLvision PRO tool a symbol library. This symbol library is an ASCII file that contains symbol shape definitions for cells used in the loaded design.

Map to Built-In Symbol Shapes

The following 'symbol' lines map the name patterns given in the second column to built-in logic gate symbols:

```
symbol and*    * AND   permute all
symbol buf*    * BUF
symbol nand*   * NAND  permute all
symbol nor*    * NOR   permute all
symbol not*    * INV
symbol or*     * OR    permute all
symbol xnor*   * XNOR  permute all
symbol xor*    * XOR   permute all
```

A mapping to the built-in multiplexer symbol looks like this:

```
symbol MUX*    * MUX   port SEL in.top
```

A mapping to a latch or dff symbol is expected to have 1 or 2 inputs, one clocked input, 1 or 2 outputs and maximal 1 top or bottom pin. The input marked as clk is always placed at the second input pin position. The inputs marked as top or bot are placed on the top respectively on the bottom of the symbol (control pins).

```
symbol DFF*    * LATCH port CLK in.clk port SET in.top port RST in.bot
```

This is a mapping to a generic box:

```
symbol other*  * GEN   port C in.clk port SEL in.top
```

You can also specify a Boolean equation as the symbol type:

```
symbol xor     * BOOL Y a^b^c   permute all
symbol xa1     * BOOL Y (a^b)*c permute all
```

These symbols are for AND-OR, OR-AND, AND-XOR, OR-XOR, XOR-AND and XOR-OR combi gates. They are expected to have one output and n input pins. A symbol shape for the gate depends on the specified type; "type" must be a sequence of digits > 0 - each defines the number of inputs for a first-level sub-gate. The order of the input pins is important - e.g. for "AO(3211)", the first 3 input pins go to the first AND sub-gate, and the next 2 input pins go to the second AND sub-gate and the remaining 2 input pins go directly to the OR sub-gate. The sum of the digits must be equal to the number of input pins. Both input and output pins support the .neg pin option to create bubbled pins.

```
AO(type)
OA(type)
AX(type)
OX(type)
XA(type)
XO(type)
```

Convert Symbols Coming from Other Tools

The RTLvision PRO download package contains symbol conversion utilities, a symbol viewer as well as a symbol editor and documentation of the symlib format.

There are two conversion tools:

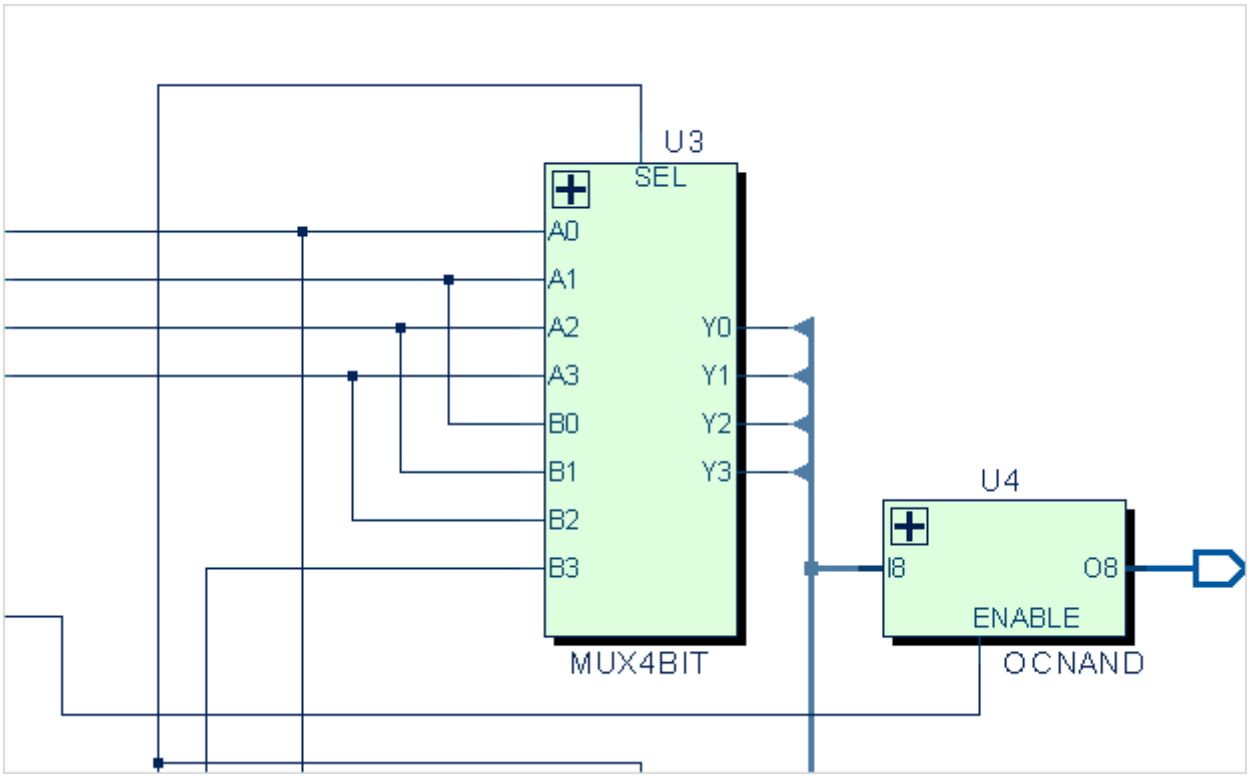
- Slibconv: a converter program that converts symbol shapes from the Synopsys symbol library format (slib file) to the Symlib Data Format.
- Cadence2symlib: a exporter program that exports symbol shapes from Cadence to the Symlib Data Format.

The result from each of this conversion utilities is a symlib file. You can use the symedit tool to view the result of the conversion process.

Use symedit to either start from scratch creating symbols or use it to customize the result of one of the conversion utilities.

To overwrite a box with a nice symbol shape the name of the symbol must match the name of the module. Here is an example of the gl85 design which can be found in the demo directory.

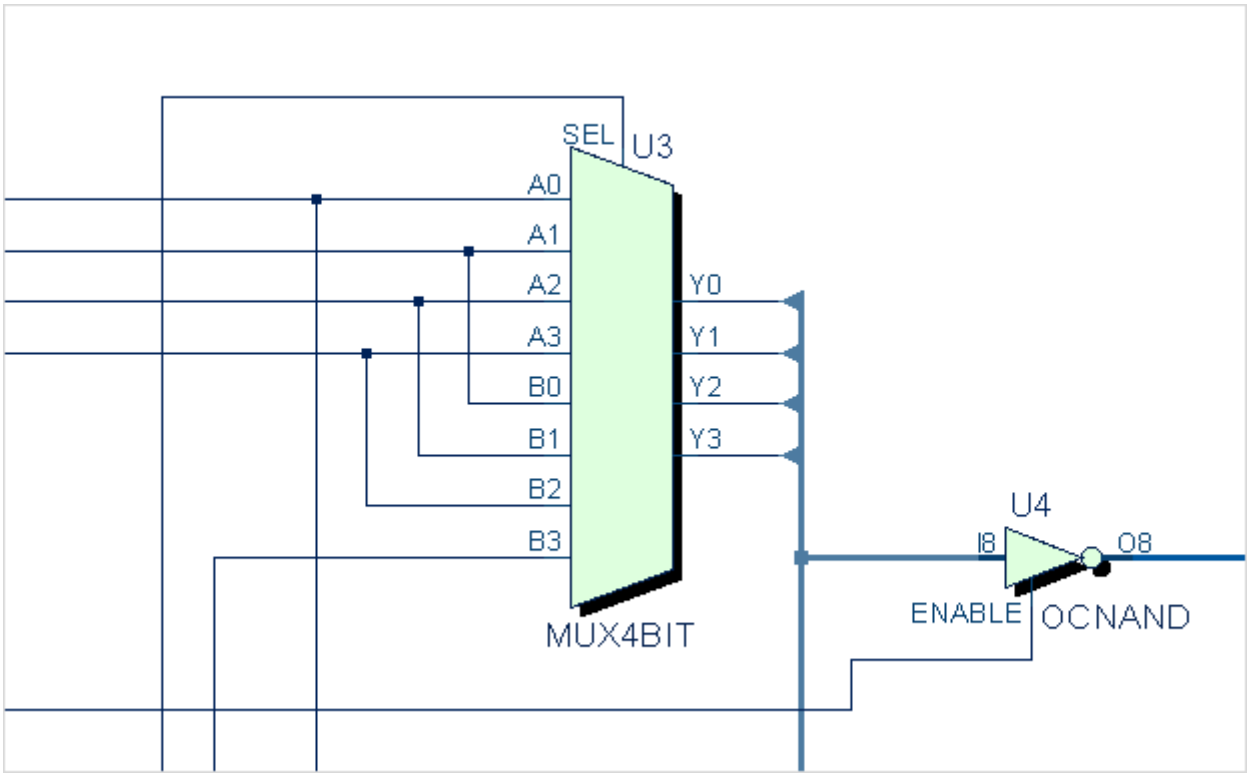
In the Schem window you can see a box for the module MUX4BIT:



Now you can provide a symbol library file to RTLvision PRO in the file open dialog or on the command line. Here is an excerpt of a symlib file:

```
symbol MUX4BIT * DEF \
    port A0 left -loc -12 -90 0 -90 \
    pinattrdsp @name -lr -2 -92 10 \
    port A1 left -loc -12 -80 0 -80 \
    pinattrdsp @name -lr -2 -82 10 \
    ...
    port Y0 right -loc 52 -70 40 -70 \
    pinattrdsp @name -ll 42 -72 10 \
    ...
    permute {([A0,A1,A2,A3],[B0,B1,B2,B3])} \
    attrdsp @name -ll 20 -110 10 \
    attrdsp @cell -uc 20 -7 10 \
    fpath 0 -10 0 -100 40 -80 40 -30 0 -10
```

Now you can see the symbol shape in the Schem window.



The API Tutorial

Introduction

This Tutorial explains how to create a design rule checker using the API. It also shows how to customize the Popup menu and add custom methods. After you have read and understood this tutorial you should be able to implement simple extensions to the tool by yourself.

How to Run an Userware Script

There are three possibilities to execute an Userware script:

1. Load a file choosing **File › Load Userware** from the menu or by entering the Tcl command `source <scriptname>` in Console window.
2. Start RTLvision PRO with the commandline option `-userware <scriptname>`. If you use this method, make sure that you register your script using the `gui database registerChangedCallback` API command.
3. Extend the GUI and invoke your Userware script on demand, e.g. from the main menu or popup menu.

The first four examples below support the methods (1) and (2). The last example also supports method (3).

The example scripts are stored in the `demo/api/` folder. To run them, please read the `@example` section in the header comment of the corresponding script.

Overview

Find Heavy Nodes

As an introductory example we describe a simple script that finds "heavy nodes", i.e. nets with a large number of connected pins.

This example script loops over all nets in each module and counts the number of pins for each net. Power and ground nets are skipped, as well as nets with constant values. The 10 biggest nets are stored into the Mem window.

```
proc heavyNodes {db} {
```

Since this script may be evaluated every time the database has changed (see below), we need to handle the case where the database has just been closed. In this case we do nothing and return:

```
    if {$db == ""} {    ;# if database changed to null-string
        return
    }
```

We need a list that stores the result. The elements will be ('value', OID) pairs, where 'value' is the number of pins the net connects to and OID is the unique database name of this net. We initialize the list with no elements:

```
    set topList {}              ;# top-ten list
```

Now we loop over all modules in the database using the database API command foreach. Inside this loop, we start a nested loop over all nets of this module, skipping power, ground, negpower and constant value nets:

```
    $db foreach module mod {
        $db foreach net $mod net {
            if {[$db isPgNet $net] || ([$db value $net] != "")} {
                continue
            }
```

Count the number of connected pins for the current net using another nested database loop:

```
            set cnt 0
            $db foreach pin $net p {
                incr cnt
            }
```

Insert this net into top-ten list `topList`. We're using a small helper procedure `addTopTen` (see below) for managing the ('value', OID) pairs. The `addTopTen` procedure returns the resulting list that we store in the same variable `topList`:

```
            set topList [addTopTen $topList $cnt $net]
        }
    }
```

The result is stored in the Mem window. Therefore we clear and activate the Mem window before we add anything into it:

```
    gui mem clear
    gui window show Mem
    gui window show Schem
```

Before we can store the result in the Mem window we must convert our result list `topList` into a valid OID list, here called `res`. We can achieve this using a small helper procedure `getTopTenA` (see below) that extracts the OIDs from `topList` and returns a new list that we store in variable `res`:

```
    set res [getTopTenA $db $topList]
    gui mem append $res
}
```

The addTopTen procedure inserts the given ('value', OID) pair into a given sorted list with max length of 10. The result is an updated top-ten list in descending order. The code is pure Tcl and doesn't use any API commands:

```
proc addTopTen {list value oid} {
    set entry [list $value $oid]
    set i 0
    foreach e $list {
        if {$value > [lindex $e 0]} {
            # insert entry before $e and limit list length to max 10.
            set list [linsert $list $i $entry]
            return [lrange $list 0 9]
        }
        incr i
    }
    if {$i < 10} {
        # we didn't insert, but the list length is < 10 - so we append
        lappend list $entry
    }
    return $list
}
```

The getTopTenA procedure returns a list of OIDs from the given top-ten list. In addition, the count fields are added as "count" attribute to the nets and the tooltipsWithAttrs flag is set (to display attributes in tooltips). The result is a new list that contains only OIDs:

```
proc getTopTenA {db list} {
    set res {}
    foreach n $list {
        set cnt [lindex $n 0]
        set net [lindex $n 1]
        $db attr $net set count=$cnt
        lappend res $net
    }
    gui settings set "tooltipsWithAttrs" 1
    return $res
}
```

The last part of the script is the entry point for the Userware code. We use gui database runOrRegisterChangedCallback to either immediately run the registered heavyNodes procedure if we have a database, or otherwise register the procedure to be executed after the database is available.

```
gui database runOrRegisterChangedCallback heavyNodes
```

Since the number of connected pins is counted inside a module only, the result of the above script is of little practical use. In the real world a net does not stop at a module boundary. A net can be interconnected with many other nets through multiple levels of hierarchy. To reflect this situation we have to traverse the net through the design hierarchy tree. Such an example is discussed in the next chapter.

Find Heavy Nodes (Design Tree Based)

Next we explain the Userware script `demo/api/heavyNodes.tcl` (design tree based) in the demo/api folder. This example is very similar to the previous one, except that it traverses the database in an instance-tree oriented way (rather than looping over all modules). It searches for signals, i. e. nets that are connected across module boundaries. Both scripts have many code fragments in common - in particular the procedures getTopTen and addTopTen. Only the different parts are explained in detail.

The heavyNodes procedure is the main procedure of our script. It loops over all signals in the design, starting in the top modules and counts the number of pins for each signal. Signals connected to power or ground are skipped, as well as signals with constant values. The 10 biggest signals are stored into the Mem window.

```
proc heavyNodes {db} {
```

Since this script may be evaluated every time the database has changed (see below), we need to handle the case where the database has just been closed. In this case we do nothing and return:

```
    if {$db == ""} {   ;# Return if the database is empty.
        return
    }
```

We need a list that stores the result. The elements will be ('value', OID) pairs, where 'value' is be the number of pins the signal connects to and OID is the unique database name of this signal. We initialize the list with no elements:

```
    set topList {} ;# top-ten list
```

Now we loop over all top modules in the database using the database API command foreach. For each top module found, we loop over all signals contained therein:

```
    $db foreach top mod {
        $db flat foreach signal $mod sig {
```

We skip all signals that carry power, ground or negpower:

```
            # skip power, ground and negpower nets
            if {[$db isPgNet $sig]} {
                continue
            }
```

We also skip all signals that contain nets with a constant value.

```
            set skip 0
            $db flat foreach net $sig net {
                if {[$db value $net] != ""} {
                    set skip 1
                    break
                }
            }
            if {$skip} {
                continue
            }
```

Now, again using an API foreach command, we count the number of pins that are connected to this signal:

```
            # count the number of connected pins
            set cnt 0
            $db flat foreach pin $sig p {
                incr cnt
            }
```

As the last action in this loop we insert this signal together with the pin count into top-ten list:

```
            set topList [addTopTenHeavyNodes $topList $cnt $sig]
        }
    }
```

Now, after having looped through all top modules, we display the result in the same way as in the example before. The procedures addTopTenHeavyNodes and getTopTenA are identical.

The script's entry point is also the same. The command

```
gui database runOrRegisterChangedCallback heavyNodes
```

is used to immediately run heavyNodes if we have a database, or otherwise register the proc to be executed after the database is available.

Find Heavy CAP or RES

Here, we explain the Userware script `demo/api/heavyCR.tcl` in the demo/api folder.

This example searches the current database for the capacitors and resistors with the highest values.

The `heavyCR` procedure is the main procedure of our script. It loops over all capacitors (primitive function CAP) and resistors (primitive function RES) and checks each value (capacitance or resistance) keeping the highest ten values of each in two lists. The lists are then combined and stored in the Mem window.

Since this script may be evaluated every time the database has changed (see below), we need to handle the case where the database has just been closed. In this case we do nothing and return:

```
proc heavyCR {db} {
    if {$db == ""} {       ;# if database changed to null-string
        return
    }
```

We need two lists storing the highest C and R values and the corresponding OIDs. The elements will be ('value', OID) pairs. 'value' will be capacitance or resistance values and OID is the unique database name of the C or R found. We initialize them with no elements:

```
    set CAPList {}              ;# top-ten list
    set RESList {}              ;# top-ten list
```

Now we loop over all modules in the database using the database API command foreach. Inside this loop, we start a nested loop over all instances of this module skipping all instances of modules:

```
    $db foreach module mod {
        $db foreach inst $mod inst {
            if {[$db isModule $inst]} {
                continue
            }
```

Get the primitive type of the instance (i.e. "CAP" for capacitor or "RES" for resistor):

```
            set func [$db primFuncOf $inst]
```

The Spice parser creates an attribute "R" or "C" which contains the value it has parsed (capacitance or resistance). We query the database using the API function "$db attr $oid getValue" to retrieve the value we are interested in. Since the values may not be in base units, we have a small procedure `map_spice_unit` (see below) which translates the parsed values into base units.

If the function is a capacitor, then we add its value to the top ten capacitor list. We do this with the small helper procedure `addTopTen` (introduced above) for managing the ('value', OID) pairs:

```
            if {$func == "CAP"} {
                set value [$db attr $inst getValue "C"]
                if {$value == ""} {
                    continue
                }
                set value [map_spice_unit $value]
                if {$value == ""} {
                    continue
                }
                set CAPList [addTopTen $CAPList $value $inst]
            }
```

The same approach is followed for the resistor values:

```
            if {$func == "RES"} {
                set value [$db attr $inst getValue "R"]
                if {$value == ""} {
                    continue
                }
                set value [map_spice_unit $value]
                if {$value == ""} {
                    continue
                }
                set RESList [addTopTen $RESList $value $inst]
            }
        }
    }
```

We clear and activate the Mem window as in the example above:

```
    # Clear and activate the Mem window
    gui mem clear
    gui window show Mem
    gui window show Schem
```

Now we create a new list `res` by concatenating the two result lists using a small helper procedure `getTopTen` (see below). Actually, we create a new list with the OIDs from `$CAPList` then a separator line and then the OIDs from `$RESList`.

```
    set res [concat [getTopTen $CAPList] ------ [getTopTen $RESList]]
```

The OID list `res` can finally be stored into the Mem window:

```
    gui mem append $res
```

To visually verify the result shown in Mem window, we turn on the display of attributes in tooltips:

```
    gui settings set "tooltipsWithAttrs" 1
}
```

The `map_spice_unit` procedure maps spice values like 0.5n or 12.8k to the base unit or returns "" if the value is not ok. The code is pure Tcl and doesn't use any API commands:

```
proc map_spice_unit {val} {
    if {[scan $val "%g%s" v unit] == 2} {
        set u 1
        switch -glob -- [string tolower $unit] {
            t*          {set u 1e12}
            g*          {set u 1e9}
            meg*        {set u 1e6}
            k*          {set u 1e3}
            mil*        {set u 25.4e-6}
            m*          {set u 1e-3}
            u*          {set u 1e-6}
            n*          {set u 1e-9}
            p*          {set u 1e-12}
            f*          {set u 1e-15}
        }
        return [expr $v * $u]
    } else {
        if {[string is double $val]} {
            return $val
        }
    }
    return ""   ;# bad
}
```

The `getTopTen` procedure returns a list of OIDs from the given top-ten list. The code is pure Tcl and doesn't use any API commands:

```
proc getTopTen {list} {
    set res {}
    foreach n $list {
        lappend res [lindex $n 1]
    }
    return $res
}
```

The entry point of our Userware code is almost identical to the previous example, except that the procedure name is `heavyCR`.

Find Heavy CAP or RES (Design Tree Based)

Here, we explain the Userware script `demo/api/heavyCRt.tcl` in the demo/api folder.

This example is very similar to the previous one, except that it traverses the database in an instance-tree oriented way (rather than looping over all modules). Both have many code fragments in common - in particular the procedures `getTopTen`, `addTopTen` and `map_spice_unit`. Only the different parts are explained in details.

The `heavyCRt` procedure is the main procedure of our script. It traverses the design hierarchy tree for all Cs and Rs and checks each value (capacitance or resistance) and collects the 10 objects with the highest values. The 10 highest Cs and 10 highest Rs are stored into the Mem window.

Since this script may be evaluated every time the database has changed (see below), we need to handle the case where the database has just been closed. In this case we do nothing and return:

```
proc heavyCRt {db} {
    if {$db == ""} {       ;# if database changed to null-string
        return
    }
```

The two result lists we need must be processed in other procedures, too. Therefore we need to declare them `global`:

```
    global CAPList RESList

    set CAPList {}              ;# top-ten list
    set RESList {}              ;# top-ten list
```

Now we loop over all top modules in the database using the database API command foreach. For each top module found, we call the recursive procedure `traverseCRt` (see below) that examines the Cs and Rs:

```
    $db foreach top mod {
        traverseCRt $db $mod
    }
```

We need to clear the 'yellow' flag that was previously set in `traverseCRt` during recursive design hierarchy traversal:

```
    $db foreach module mod {
        $db flag $mod clear yellow
    }
```

As in the example above, we clear and activate the Mem window. Here we also store the result in the Mem window and activate the option to display attributes in the tooltips:

```
    # Clear and activate the Mem window
    gui mem clear
    gui window show Mem
    gui window show Schem

    # Store the result into the Mem window.
    set res [concat [getTopTen $CAPList] ------ [getTopTen $RESList]]
    gui mem append $res

    # Display attributes in tooltips
    gui settings set "tooltipsWithAttrs" 1
}
```

The `traverseCRt` procedure actually traverses the design hierarchy tree by recursively calling itself for each down-module. It collects the Rs and Cs in two global result lists `CAPList` and `RESList`:

```
proc traverseCRt {db mod} {
    global CAPList RESList

    $db foreach inst $mod inst {
```

If the instance ($inst) is non-primitive (i.e. it has a down-module), we dive down into its down-module and call this procedure recursively; but only if the "yellow" flag is not set for this module. We use the "yellow" flag to indicate that a module has been visited before. This way we can avoid traversing the same module multiple times (speed-up).

```
        if {[$db isModule $inst]} {
            set down [$db moduleOf $inst]
            if {![$db flag $down is yellow]} {
                traverseCRt $db $down
                $db flag $down set yellow
            }
            continue
        }

        # get primitive function
        set func [$db primFuncOf $inst]

        # check attribute R and C (the Spice parser stores values there).
        # and insert this C or R into top-ten list

        if {$func == "CAP"} {
            set value [$db attr $inst getValue "C"]
            if {$value == ""} {
                continue
            }
            set value [map_spice_unit $value]
            if {$value == ""} {
                continue
            }
            set CAPList [addTopTen $CAPList $value $inst]
        }
        if {$func == "RES"} {
            set value [$db attr $inst getValue "R"]
            if {$value == ""} {
                continue
            }
            set value [map_spice_unit $value]
            if {$value == ""} {
                continue
            }
            set RESList [addTopTen $RESList $value $inst]
        }
    }
}
```

The helper procedure `map_spice_unit` is used as in the example above.

The entry point of our Userware code is almost identical to the first example, except that the procedure name is `heavyCRt`.

Find Floating Nodes

Here, we explain the userware script `demo/api/floatingNodes.tcl` in the demo/api folder.

This example loops over all nets to find floating nodes. The `floatingNodes` procedure loops over all

nets and counts the number of pins for each net. Power and ground nets are skipped. All nets with exactly one pin are stored into the Mem window.

Since this script may be evaluated every time the database has changed (see below), we need to handle the case where the database has just been closed. In this case we do nothing and return:

```
proc floatingNodes {db} {
    if {$db == ""} {       ;# if database changed to null-string
        return
    }

    set resList {}              ;# result List
```

Now we loop over all modules in the database using the database API command foreach. Inside this loop, we start a nested loop over all nets of this module, skipping supply and constant nets:

```
    $db foreach module mod {
        $db foreach net $mod net {
            # skip power, ground, negpower and constant nets
            if {[$db isPgNet $net] || ([$db value $net] != "")} {
                continue
            }
```

Count the number of connected pins and show the overall result in the Mem window:

```
            set cnt 0
            $db foreach pin $net p {
                incr cnt
                if {$cnt > 1} {
                    # no need to count more
                    break
                }
            }
            if {$cnt == 1} {
                lappend resList $net
            }
        }
    }

    gui mem clear
    gui window show Mem
    gui window show Schem
    gui mem append $resList
}
```

The entry point of our Userware code is almost identical to the first example, except that the procedure name is `floatingNodes`.

Customize the Popup Menu

Here, we explain the userware script `demo/api/customPopup.tcl` in the demo/api folder.

This example adds two entries to the Userware submenu of the Popup Menu. The first entry "Show Net" invokes the procedure `ShowNet` to load a net and all connected objects to the Cone window. The second entry "Show Signal" invokes the procedure `ShowNet` in signal mode to load a signal and all connected objects to the Cone window:

```
proc ShowNet {signalMode oidList} {
```

First check if the database or the oidList is not empty and if the object type is a net.

```
    ##
    # Return if the database is empty.
    #
    set db [gui database get]
    if {$db == ""} {
        return
    }

    ##
    # We expect one net object in $oidList.
    #
    if {[llength $oidList] != 1} {
        return
    }
    set net [lindex $oidList 0]
    if {[$db oid type $net] != "net"} {
        return
    }
```

Decide if the procedure ShowNet is running in signal mode or not. If signal mode is on, then loop over all nets belonging to the signal and collect a list of pins. In the other case loop over all pins of $net and collect a list of pins.

```
    if {$signalMode} {
       ##
       # Collect all pins at all nets of the given signal.
       #
       $db flat foreach net $net n {
          $db foreach pin $n pin {
              lappend pinList $pin
           }
       }
    } else {
       ##
       # Collect all pins at this net.
       #
       $db foreach pin $net pin {
           lappend pinList $pin
       }
    }
}
```

Load $pinList in the Cone window of and activate the Tree and Cone tab, also zoom to fullfit.

```
    gui cone load $pinList
    gui window show Tree
    gui window show Cone
    gui cone zoom fullfit
```

In case you need to remove existing custom entries in the Popup menu add

```
gui popup reset
```

to the script.

Append the new entry "Show Net" at the end of the Popup menu.

```
gui popup append "Show Net"    [list ShowNet 0]
```

Append the new entry "Show Signal" at the end of the Popup menu.

```
gui popup append "Show Signal" [list ShowNet 1]
```

GUI Customization Tutorial

This tutorial explains how to perform common GUI customization tasks using the GUI API, such as extending the main menu, extending the popup menu, and adding your own custom widgets.

The tutorial comes with a number of example Tcl files in `demo/api/guiCustomization`. You can load these Tcl files as usual Userware scripts, e.g. at application startup using RTLvision PRO's command-line option `-userware demo/api/guiCustomization/script-name.tcl`, from the **File › Load Userware** menu, or with the `source demo/api/guiCustomization/script-name.tcl` command from the GUI's Console window.

Extending the Main Menu

A new main menu entry (along with an empty sub-menu) can be added by using the `gui menu mainMenu $name -position $position` command.

`$name` is the display text of the menu entry in the main menu - if you add an underscore character `_` to the `$name`, the following character will be the menu entry's keyboard shortcut in combination with the `Alt` key.

The optional `$position` argument is either a number denoting the absolute insert position of the new menu entry into the main menu, or `end` (the default), which puts the new menu entry right of all existing menu entries.

NOTE	The `Help` menu will always stay at the rightmost position.

Example

```
gui menu mainMenu _Custom -position 5
```

This adds a new menu entry names `Custom` to the fifth position in the main menu, and assigns the keyboard shortcut `Alt` + `C`.

The new main menu entry can now be populated by:

```
gui menu command $path $command
```

which adds a simple menu item with an associated command,

```
gui menu checkbutton $path $command $variableName
```

which adds a checkable menu item where the check state is stored in the given variable,

```
gui menu radiobutton $path $command $variableName $variableValue
```

which adds a radio-button style menu item,

```
gui menu separator $path
```

which adds a menu separator and

```
gui menu submenu $path
```

which adds a menu item with an empty sub-menu.

The $path argument of these commands defines the menu hierarchy into which to insert the new menu item, where the last item of $path is the name of the new menu item - again, you can use an underscore character (_) to define the keyboard shortcut.

Example

The following examples use as $command procedures that are defined in the demo/api/guiCustomization/customizeMenu.tcl script.

```
gui menu command              \
    {{Custom} {_Into Memory}} \
    {CustomizeMenu:intoMemory}
```

This adds a new menu item with the name Into Memory and with the keyboard shortcut Alt + I into the sub-menu of the Custom main menu entry and associates it with the command CustomizeMenu:intoMemory (this command is run whenever the menu item is clicked).

```
gui menu submenu \
    {{Custom} {_Rotate}}
```

This adds a new menu item with the name Rotate and the keyboard shortcut Alt + R and an empty sub-menu into the sub-menu of the Custom main menu entry.

```
gui menu command              \
    {{Custom} {Rotate} {_0}} \
    {CustomizeMenu:rotate R0}
```

This adds a new menu item with the name 0 and the keyboard shortcut Alt + 0 into the Custom / Rotate sub-menu, and associates the command CustomizeMenu:rotate R0 with it.

```
gui menu checkbutton             \
    {{Custom} {Show _Net Names}} \
    {gui settings changed}       \
    [gui settings variable "nlv:shownetname"]
```

This adds a checkable menu item named Show Net Names and with the keyboard shortcut Alt + N into the sub-menu of the Custom main menu entry. The check state is reflected in the nlv:shownetname settings variable (this is a global configuration option - normally set via the Preferences dialog - that

specifies if the names of nets should be shown in the Schematic view). Every time the menu item is clicked, the value of `nlv:shownetname` setting is flipped and the command `gui settings changed` is executed.

You can also perform context sensitive modifications to menu items by using the `gui menu customizeEntry $path $command` command. The `$command` is executed right after opening the corresponding menu and creating the menu item indicated by `$path`. `$command` is called with an `$menuId` parameter which is the current Tk menu widget; by definition, the menu entry indicated by `$path` is the last item of the Tk menu, so it can be referenced by the index `end` when calling `$menuId` sub-commands within `$command`.

Example

```
proc CustomizeMenu:enableIfAnInstanceIsSelected {menuId} {
    set hasInstSelection 0

    set db [gui get database]
    if {$db != {}} {
        foreach oid [gui selection get] {
            if {[$db oid type $oid] eq "inst"} {
                set hasInstSelection 1
                break
            }
        }
    }

    $menuId entryconfigure end \
        -state [expr {$hasInstSelection ? "normal" : "disabled"}]
}

# Only enable the "Custom / Rotate" sub-menu if at least one object of
# type "Inst" is selected:
gui menu customizeEntry \
    {{Custom} {Rotate}} \
    CustomizeMenu:enableIfAnInstanceIsSelected
```

The full example can be found in `demo/api/guiCustomization/customizeMenu.tcl`.

Extending the Toolbar

New toolbar buttons can be added with the `gui toolbar addButton` function.

Example

```
set window          .
set name            toggleNetNames
set imageSetName    show-net
set imageSetPattern [file join [file dirname [info script]] show-net-SIZE.png]
set position        10
set command         {CustomizeToolbar:toggleNetNames}
set tooltip         "Toggle the display of net names in the schematic view"

##
# Load an imageset. Since the file pattern is ".../show-net-SIZE.png", this will
# load 16, 24, 32, 48, 64 pixel versions from ".../show-net-16.png",
# ".../show-net-24.png", etc.
#
gui imageset load $imageSetName $imageSetPattern

gui toolbar addButton   \
    $window             \
    $name               \
    $imageSetName       \
    -command  $command  \
    -tooltip  $tooltip  \
    -position $position

proc CustomizeToolbar:toggleNetNames {} {
    gui settings set "nlv:shownetname" [expr {![gui settings get "nlv:shownetname"]}]
    gui settings changed
}
```

The full example can be found in `demo/api/guiCustomization/customizeToolbar.tcl`.

Extending the Popup Menu

New popup menu entries can be added with `gui popup append $label $command ?-menuname $menuname?`, where `$label` is the intended menu label, `$command` is the command to be associated with the popup menu entry (`$command` is appended with the list of currently selected OIDs before being invoked), and `$menuname` is the name of the sub-menu to create the new item in (if `-menuname $menuname` is not specified, the menu item is created in a sub-menu called `Userware`).

Example

```
gui popup append       \
    -menuname "Custom" \
    "Toggle Net Names" \
    {CustomizePopup:toggleNetNames}
gui popup append       \
    -menuname "Custom" \
    "Rotate +90"       \
    {CustomizePopup:rotate +R90}

proc CustomizePopup:toggleNetNames {oids} {
    gui settings set "nlv:shownetname" [expr {![gui settings get "nlv:shownetname"]}]
    gui settings changed
}

proc CustomizePopup:rotate {mode oids} {
    set db [gui database get]
    foreach oid $oids {
        if {[$db oid type $oid] eq "inst"} {
            $db orient $oid $mode
        }
    }
    gui database modified
}
```

This example creates two popup menu items `Custom / Toggle Net Names` and `Custom / Rotate 90`.

In order to customize the new popup menu entries (e.g. enable them only under specific conditions), `gui popup customize $command` can be used. This executes `$command` (`$command` is appended with the Tk menu command corresponding to the popup menu and the list of currently selected OIDs before being executed) every time the popup menu is invoked, right after it is filled with all entries.

Example

```
gui popup customize {CustomizePopup:customize}

proc CustomizePopup:containsInst {oids} {
    set db [gui database get]
    if {$db != {}} {
        foreach oid $oids {
            if {[$db oid type $oid] eq "inst"} {
                return 1
            }
        }
    }
    return 0
}

proc CustomizePopup:customize {menu oids} {
    # Always enable our own "Custom" popup menu item.
    $menu entryconfigure "Custom" -state normal

    # Get the "Custom" sub-menu.
    set customMenu [$menu entrycget "Custom" -menu]

    # Always enable "Custom / Toggle Net Names".
    $customMenu entryconfigure "Toggle Net Names" \
        -state normal

    # Only enable "Custom / Rotate 90" if at least one instance is
    # selected.
    set hasInst [CustomizePopup:containsInst $oids]
    $customMenu entryconfigure "Rotate 90" \
        -state [expr {$hasInst ? "normal" : "disabled"}]
}
```

The full example can be found in `demo/api/guiCustomization/customizePopup.tcl`.

Adding Custom Widgets

RTLvision PRO has a dedicated area where custom widgets can be shown - the bottom tabs that also contain the Search window, the Infobox, etc.

Users can add their own widgets into this area using the `gui window insertCustomWidget $name` command, where `$name` is the label displayed on the newly added tab. `gui window insertCustomWidget` internally creates a new `ttk::frame` and return's its widget path. This `ttk::frame` is fully customizable by the user (e.g. by adding child widgets, etc.).

The custom widget (and the corresponding tab) can be removed with `gui window removeCustomWidget $name`. The optional parameter `-pluginNamespace` can be used to call the `Finit` procedure in the given namespace to unload and therewith disable the plugin when the custom widget is destroyed (e.g. when the user clicks the 'x' in the widget's tab).

Example

```
set name "My Custom Widget (in the bottom tab area)"
set customWidget [gui window insertCustomWidget $name]
ttk::label  $customWidget.label     \
    -text "This is a custom widget."
ttk::button $customWidget.btnRemove \
    -text "Remove"                  \
    -command [list gui window removeCustomWidget $name]
pack $customWidget.label
pack $customWidget.btnRemove
```

This creates a custom widget and inserts it into the bottom tabs as `My Custom Widget`. The custom widget contains a label and a button that removes the widget by calling `gui window removeCustomWidget`.

Instead of the "bottom tab" area, a custom widget can also be inserted into an arbitrary Tab window by using the `-tabwindow $tab` option:

Example

```
set name2 "My Custom Widget (in an arbitrary tab)"
# get the Tab window where the "Schem" window lives in
set tab [gui window getTab "Schem"]
set customWidget2 [gui window insertCustomWidget -tabwindow $tab $name2]
ttk::label  $customWidget2.label \
    -text "This is a custom widget in the default Tab window."
pack $customWidget2.label
```

The full example can be found in `demo/api/guiCustomization/customWidget.tcl`.

Changing the GUI Layout

This example adds a new Cone window to the right of the default/main Schem window using the high-level API function `gui window split …`:

```
# get the main Schem window
set schem [gui window defaultClassWindow Schem]

# create an empty Tab window right of the Schem window
set tab [gui window split $schem right]

# insert a new Cone window into the Tab window
gui window new -tabwindow $tab "Cone"
```

This example adds a row with new Schem, Cone and Source windows using lower level API functions:

```
##
# Get the main vertical pane window of the main window.
#
set mainVertical [gui window getMainVerticalPane .]

##
# Insert a new horizontal pane window into the main vertical pane.
# It is added bottom-most into the vertical pane, but above the Console and
# Messages windows.
#
set horizontal [gui window createHorizontalPane $mainVertical]

##
# Create and insert three vertical Pane windows into the horizontal Pane.
#
set vertical1 [gui window createVerticalPane $horizontal]
set vertical2 [gui window createVerticalPane $horizontal]
set vertical3 [gui window createVerticalPane $horizontal]

##
# Insert a Tab window into each of the vertical Pane windows.
#
set tab1 [gui window createTab $vertical1]
set tab2 [gui window createTab $vertical2]
set tab3 [gui window createTab $vertical3]

##
# Space the three vertical Pane windows evenly by setting the horizontal
# pane's sash positions to 1/3 and 2/3.
#
gui window setPaneSashes $horizontal {0.333 0.666}

##
# Insert new Schem, Cone, Source windows into the three Tab windows.
#
gui window new -tabwindow $tab1 "Schem"
gui window new -tabwindow $tab2 "Cone"
gui window new -tabwindow $tab3 "Source"
```

This example adds a new top-level window with side-by-side Schem and Cone windows at the top and a full-width Source window at the bottom.

```
##
# Create a new top-level window (this also creates and inserts the
# obligatory vertical pane).
#
set top [gui window createToplevel]

##
```

```
# Show it.
#
gui window modelessDialog \
    $top \
    "My Custom Top-Level Window" \
    -place "CENTER" \
    -size {800 600} \
    -onTop \
    -closeCallback "destroy $top"

##
# Get the vertical pane.
#
set vertical [gui window getMainVerticalPane $top]

##
# Create two horizontal panes.
#
set horizontal1 [gui window createHorizontalPane $vertical]
set horizontal2 [gui window createHorizontalPane $vertical]

##
# Space the horizontal panes evenly (set the vertical pane's sash
# position to 1/2).
#
gui window setPaneSashes $vertical {0.5}

##
# Create some vertical Pane windows.
#
set vertical1_1 [gui window createVerticalPane $horizontal1]
set vertical1_2 [gui window createVerticalPane $horizontal1]
set vertical2   [gui window createVerticalPane $horizontal2]

##
# Space the vertical pane windows in the first horizontal Pane evenly by
# setting its sash position to 1/2.
#
gui window setPaneSashes $horizontal1 {0.5}

##
# Create a Tab window in each vertical Pane.
#
set tab1_1 [gui window createTab $vertical1_1]
set tab1_2 [gui window createTab $vertical1_2]
set tab2   [gui window createTab $vertical2]


##
# Insert Schem, Cone, Source windows into the Tab windows.
#
```

```
gui window new -tabwindow $tab1_1 "Schem"
gui window new -tabwindow $tab1_2 "Cone"
gui window new -tabwindow $tab2 "Source"
```

The full example can be found in `demo/api/guiCustomization/customizeLayout.tcl`.

Glossary (Terms & Definitions)

This Glossary defines several terms (words and phrases) that are used within this documentation and on the GUI (graphical user interface) of RTLvision PRO.

Overview

General Terms

Bookmark

A "Bookmark" is used to save the state of a window. It can be restored later.

EDIF

EDIF stands for Electronic Design Interchange Format, and has been predominantly used as a neutral format in which to store electronic netlists and schematics.

FlexNet

Software package from Flexera Software (former Acresso, Macrovision and Globetrotter) that verifies license keys. RTLvision PRO requires a FlexNet license key to run.

Floating License

A FlexNet protected license that is automatically shared between computers in a local network.

Node-Locked License

A FlexNet protected license that is bound to a certain computer.

Project

A "Project" file includes all design related settings used to load the input files. The project file has the extension `.vpj`.

RTLvision PRO

Name of the software tool. It reads Verilog and VHDL files, generates and displays schematic pictures for debugging and documentation.

Snapshot

A "Snapshot" is used to save the state of the Cone window to a file. It can be restored later — or sent to a colleague.

Verilog

Verilog is a hardware description language, used for the design of ASICs and FPGAs in order to make digital circuits.

VHDL

VHDL is a hardware description language, used for the design of ASICs and FPGAs in order to make digital circuits.

Workspace

A "Workspace" defines all current settings of the GUI (including all check boxes, colors, etc.). The workspace file has the extension `.ws`.

GUI Terms

Cone Window

The Cone window displays schematic fragments that allow for incremental navigation using double clicks.

Console Window

The Console window displays warnings and error messages and can be used to interactively enter API commands.

Context Menu

A Context Menu is popped up if the right mouse button is pressed (but not moved) over a Database Object.

Drag & Drop

Drag & Drop is done by pressing the left (or right) mouse button over a Database Object and then moving it to another window (Schem, Cone, Source, Mem) or to the Read Spice dialog's advanced buttons (for defining Spice Additives).

Highlight

A mechanism to highlight certain Database Objects (instance, net, pin) simultaneously in all views (in all windows).

Mem Window

The Mem window can be used as a notepad (memory) to temporarily store Database Objects.

Read RTL Dialog

A dialog window to define Verilog files and libraries, VHDL files and libraries. The dialog's **[Read]** button starts the Verilog or VHDL parser.

Read Verilog Dialog

A dialog window to define Verilog files and libraries. The dialog's **[Read]** button starts the Verilog parser.

Read EDIF Dialog

A dialog window to select the EDIF file. The dialog's **[Read]** button starts the EDIF parser.

Schem Window

The Schem window displays the schematic pages of one Module.

Search Window

The Search window is used to query the Database. Search results are displayed in a list of Database Objects and can be used for highlighting and other functions.

Selection Label

The Selection Label displays the most recently selected Database Objects.

Source Window

The Source window displays the source file of the loaded design.

Tooltip

A label that is displayed under the mouse cursor on appropriate objects after a small delay. Tooltips usually give additional context sensitive information or hints about the object under the mouse cursor.

Tree Window

The Tree window displays the design hierarchy tree.

Pane Window

The Pane window is a container for a Tab group.

Database Terms

Attributes

Each Database Object can store attribute strings in a `name=value` syntax (these attributes are module-based in contrast to Flat Attributes).

Binfile

ZDB binary file representation of the database memory.

Database

A database stores the full design: the top Module(s) plus all sub modules.

Database Object

A Database object is one of: Module, Primitive, Port, PortBus, Instance, Pin, PinBus, Net or NetBus.

Flat Attributes

Each Database Top Module can store instance-tree-based attributes - called "Flat Attributes" (in parallel to module-based Attributes). They also follow the `name=value` syntax.

Meta Attributes

The "Meta Attributes" control what Attributes and Flat Attributes are displayed in the Schem and Cone windows. They are called `@nlv` or `@nlv:..` and are attached to a Module or Primitive or to the Database root.

Module

A Database Module represents a Spice sub-circuit or the top-level circuit. The Tree window displays the tree of Modules.

OID

Each Database Object can be referred to by an OID (Object Identification). OIDs can store the path through the hierarchy tree to a certain instance.

Primitive

A Database Primitive represents a cell with a defined function. For a digital design this is, e.g. a Boolean gate like AND, NAND or INV.

Appendix A: Copyright Notes and Third-Party Software Components

Third-Party Licenses

RTLvision PRO includes or uses the following third-party software components in accordance with the respective license terms:

Bison Parser

- Version: 3.8.2
- Source: https://www.gnu.org/software/bison/
- License type: GNU General Public License with Bison exception

```
Copyright (C) 1984, 1989-1990, 2000-2015, 2018 Free Software Foundation, Inc.

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.

As a special exception, you may create a larger work that contains
part or all of the Bison parser skeleton and distribute that work
under terms of your choice, so long as that work isn't itself a
parser generator using the skeleton or a modified version thereof
as a parser skeleton.  Alternatively, if you modify or redistribute
the parser skeleton itself, you may (at your option) remove this
special exception, which will cause the skeleton and the resulting
Bison output files to be licensed under the GNU General Public
License without this special exception.

This special exception was added by the Free Software Foundation in
version 2.2 of Bison.
```

Flex

- Version: 2.6.4

- Source: https://github.com/westes/flex
- License type: BSD

```
Flex carries the copyright used for BSD software, slightly modified
because it originated at the Lawrence Berkeley (not Livermore!) Laboratory,
which operates under a contract with the Department of Energy:

Copyright (c) 2001, 2002, 2003, 2004, 2005, 2006, 2007 The Flex Project.

Copyright (c) 1990, 1997 The Regents of the University of California.
All rights reserved.

This code is derived from software contributed to Berkeley by
Vern Paxson.

The United States Government has rights in this work pursuant
to contract no. DE-AC03-76SF00098 between the United States
Department of Energy and the University of California.

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

1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.

Neither the name of the University nor the names of its contributors
may be used to endorse or promote products derived from this software
without specific prior written permission.

THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE.

This basically says "do whatever you please with this software except
remove this notice or take advantage of the University's (or the flex
authors') name".

Note that the "flex.skl" scanner skeleton carries no copyright notice.
You are free to do whatever you please with scanners generated using flex;
for them, you are not even bound by the above copyright.
```

Graphviz

- Version: 2.40.1

- Source: https://www.graphviz.org/
- License type: EPL

```
Eclipse Public License - v 1.0

   THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS ECLIPSE
   PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF
   THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.

   1. DEFINITIONS

   "Contribution" means:

   a) in the case of the initial Contributor, the initial code and
   documentation distributed under this Agreement, and

   b) in the case of each subsequent Contributor:

   i) changes to the Program, and

   ii) additions to the Program;

   where such changes and/or additions to the Program originate from and
   are distributed by that particular Contributor. A Contribution
   'originates' from a Contributor if it was added to the Program by such
   Contributor itself or anyone acting on such Contributor's behalf.
   Contributions do not include additions to the Program which: (i) are
   separate modules of software distributed in conjunction with the
   Program under their own license agreement, and (ii) are not derivative
   works of the Program.

   "Contributor" means any person or entity that distributes the Program.

   "Licensed Patents" mean patent claims licensable by a Contributor which
   are necessarily infringed by the use or sale of its Contribution alone
   or when combined with the Program.

   "Program" means the Contributions distributed in accordance with this
   Agreement.

   "Recipient" means anyone who receives the Program under this Agreement,
   including all Contributors.

   2. GRANT OF RIGHTS

   a) Subject to the terms of this Agreement, each Contributor hereby
   grants Recipient a non-exclusive, worldwide, royalty-free copyright
   license to reproduce, prepare derivative works of, publicly display,
   publicly perform, distribute and sublicense the Contribution of such
   Contributor, if any, and such derivative works, in source code and
```

```
object code form.

b) Subject to the terms of this Agreement, each Contributor hereby
grants Recipient a non-exclusive, worldwide, royalty-free patent
license under Licensed Patents to make, use, sell, offer to sell,
import and otherwise transfer the Contribution of such Contributor, if
any, in source code and object code form. This patent license shall
apply to the combination of the Contribution and the Program if, at the
time the Contribution is added by the Contributor, such addition of the
Contribution causes such combination to be covered by the Licensed
Patents. The patent license shall not apply to any other combinations
which include the Contribution. No hardware per se is licensed
hereunder.

c) Recipient understands that although each Contributor grants the
licenses to its Contributions set forth herein, no assurances are
provided by any Contributor that the Program does not infringe the
patent or other intellectual property rights of any other entity. Each
Contributor disclaims any liability to Recipient for claims brought by
any other entity based on infringement of intellectual property rights
or otherwise. As a condition to exercising the rights and licenses
granted hereunder, each Recipient hereby assumes sole responsibility to
secure any other intellectual property rights needed, if any. For
example, if a third party patent license is required to allow Recipient
to distribute the Program, it is Recipient's responsibility to acquire
that license before distributing the Program.

d) Each Contributor represents that to its knowledge it has sufficient
copyright rights in its Contribution, if any, to grant the copyright
license set forth in this Agreement.

3. REQUIREMENTS

A Contributor may choose to distribute the Program in object code form
under its own license agreement, provided that:

a) it complies with the terms and conditions of this Agreement; and

b) its license agreement:

i) effectively disclaims on behalf of all Contributors all warranties
and conditions, express and implied, including warranties or conditions
of title and non-infringement, and implied warranties or conditions of
merchantability and fitness for a particular purpose;

ii) effectively excludes on behalf of all Contributors all liability
for damages, including direct, indirect, special, incidental and
consequential damages, such as lost profits;

iii) states that any provisions which differ from this Agreement are
offered by that Contributor alone and not by any other party; and
```

```
iv) states that source code for the Program is available from such
Contributor, and informs licensees how to obtain it in a reasonable
manner on or through a medium customarily used for software exchange.

When the Program is made available in source code form:

a) it must be made available under this Agreement; and

b) a copy of this Agreement must be included with each copy of the
Program.

Contributors may not remove or alter any copyright notices contained
within the Program.

Each Contributor must identify itself as the originator of its
Contribution, if any, in a manner that reasonably allows subsequent
Recipients to identify the originator of the Contribution.

4. COMMERCIAL DISTRIBUTION

Commercial distributors of software may accept certain responsibilities
with respect to end users, business partners and the like. While this
license is intended to facilitate the commercial use of the Program,
the Contributor who includes the Program in a commercial product
offering should do so in a manner which does not create potential
liability for other Contributors. Therefore, if a Contributor includes
the Program in a commercial product offering, such Contributor
("Commercial Contributor") hereby agrees to defend and indemnify every
other Contributor ("Indemnified Contributor") against any losses,
damages and costs (collectively "Losses") arising from claims, lawsuits
and other legal actions brought by a third party against the
Indemnified Contributor to the extent caused by the acts or omissions
of such Commercial Contributor in connection with its distribution of
the Program in a commercial product offering. The obligations in this
section do not apply to any claims or Losses relating to any actual or
alleged intellectual property infringement. In order to qualify, an
Indemnified Contributor must: a) promptly notify the Commercial
Contributor in writing of such claim, and b) allow the Commercial
Contributor to control, and cooperate with the Commercial Contributor
in, the defense and any related settlement negotiations. The
Indemnified Contributor may participate in any such claim at its own
expense.

For example, a Contributor might include the Program in a commercial
product offering, Product X. That Contributor is then a Commercial
Contributor. If that Commercial Contributor then makes performance
claims, or offers warranties related to Product X, those performance
claims and warranties are such Commercial Contributor's responsibility
alone. Under this section, the Commercial Contributor would have to
defend claims against the other Contributors related to those
```

```
performance claims and warranties, and if a court requires any other
Contributor to pay any damages as a result, the Commercial Contributor
must pay those damages.

5. NO WARRANTY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS
PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY
WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR
FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible
for determining the appropriateness of using and distributing the
Program and assumes all risks associated with its exercise of rights
under this Agreement , including but not limited to the risks and costs
of program errors, compliance with applicable laws, damage to or loss
of data, programs or equipment, and unavailability or interruption of
operations.

6. DISCLAIMER OF LIABILITY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPIENT NOR
ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING
WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR
DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED
HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. GENERAL

If any provision of this Agreement is invalid or unenforceable under
applicable law, it shall not affect the validity or enforceability of
the remainder of the terms of this Agreement, and without further
action by the parties hereto, such provision shall be reformed to the
minimum extent necessary to make such provision valid and enforceable.

If Recipient institutes patent litigation against any entity (including
a cross-claim or counterclaim in a lawsuit) alleging that the Program
itself (excluding combinations of the Program with other software or
hardware) infringes such Recipient's patent(s), then such Recipient's
rights granted under Section 2(b) shall terminate as of the date such
litigation is filed.

All Recipient's rights under this Agreement shall terminate if it fails
to comply with any of the material terms or conditions of this
Agreement and does not cure such failure in a reasonable period of time
after becoming aware of such noncompliance. If all Recipient's rights
under this Agreement terminate, Recipient agrees to cease use and
distribution of the Program as soon as reasonably practicable. However,
Recipient's obligations under this Agreement and any licenses granted
```

```
by Recipient relating to the Program shall continue and survive.

Everyone is permitted to copy and distribute copies of this Agreement,
but in order to avoid inconsistency the Agreement is copyrighted and
may only be modified in the following manner. The Agreement Steward
reserves the right to publish new versions (including revisions) of
this Agreement from time to time. No one other than the Agreement
Steward has the right to modify this Agreement. The Eclipse Foundation
is the initial Agreement Steward. The Eclipse Foundation may assign the
responsibility to serve as the Agreement Steward to a suitable separate
entity. Each new version of the Agreement will be given a
distinguishing version number. The Program (including Contributions)
may always be distributed subject to the version of the Agreement under
which it was received. In addition, after a new version of the
Agreement is published, Contributor may elect to distribute the Program
(including its Contributions) under the new version. Except as
expressly stated in Sections 2(a) and 2(b) above, Recipient receives no
rights or licenses to the intellectual property of any Contributor
under this Agreement, whether expressly, by implication, estoppel or
otherwise. All rights in the Program not expressly granted under this
Agreement are reserved.

This Agreement is governed by the laws of the State of New York and the
intellectual property laws of the United States of America. No party to
this Agreement will bring a legal action under this Agreement more than
one year after the cause of action arose. Each party waives its rights
to a jury trial in any resulting litigation.
```

LEF/DEF Parser

- Version: 5.8
- Source: https://si2.org/oa-tools-utils-libs/
- License type: Apache License

```
Copyright (c) 2012 - 2017, Cadence Design Systems

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

Liberty Parser

- Version: 2.6
- Source: https://www.opensourceliberty.org/
- License type: SYNOPSYS Open Source License Version 1.0

```
Liberty License Agreement

THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS SYNOPSYS OPEN SOURCE
LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM
CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.

1. DEFINITIONS
"Contribution" means:
     a) in the case of Synopsys, Inc., ("Synopsys"), the Original Program, and
     b) in the case of each Contributor,
          i) changes to the Program, and
          ii) additions to the Program;
where such changes and/or additions to the Program originate from and are distributed
by that particular Contributor. A Contribution 'originates' from a Contributor if it
was made by such Contributor itself or anyone acting on such Contributor's behalf.
Contributions do not include additions to the Program which: (i) are separate modules
of software distributed in conjunction with the Program under their own license
agreement, and (ii) are not derivative works of the Program.

"Contributor" means Synopsys and any other entity that distributes the Program.

"Licensed Patents" mean patent claims licensable by a Contributor that are necessarily
infringed by the use or sale of its Contribution alone or when combined with the
Program. A Licensed Patent only includes claims directed towards a data structure,
algorithm, technique or method that is directly implemented by the Contribution alone
or by the Contribution combined with the Program existing at the time the Contribution
is made.

"Original Program" means the original version of the Liberty TM format specification
and reference implementation accompanying this Agreement as released by Synopsys,
including source code, object code and documentation, if any.

"Program" means the Original Program and other Contributions.

"Recipient" means anyone who receives the Program under this Agreement, including all
Contributors.

2. GRANT OF RIGHTS
a) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a
non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare
derivative works of, publicly display, publicly perform, distribute and sublicense the
Contribution of such Contributor, if any, and such derivative works, in source code
and object code form. This license does not provide Recipient a license to reproduce,
```

prepare derivative works of, publicly display, publicly perform, distribute or sublicense other copyrighted works of Contributors.

b) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free patent license under that Contributor's Licensed Patents to make, use, sell, offer to sell, import and otherwise transfer the Contribution of such Contributor, if any, in source code and object code form. This patent license shall apply to the combination of the Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition of the Contribution causes such combination to be covered by the Licensed Patents. The scope of a Contributor's Licensed Patents shall be limited solely to the extent of that Contributor's Contribution. The patent license shall not apply to any other combinations that include the Contribution.

c) Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, no assurances are provided by any Contributor that the Program does not infringe the patent or other intellectual property rights of any other entity. Each Contributor disclaims any liability to Recipient for claims brought by any other entity based on infringement of intellectual property rights or otherwise. As a condition to exercising the rights and licenses granted hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual property rights needed, if any. For example, if a third party patent license is required to allow Recipient to distribute the Program, it is Recipient's responsibility to acquire that license before distributing the Program.

d) Each Contributor represents that to its knowledge it has sufficient copyright rights in its Contribution, if any, to grant the copyright license set forth in this Agreement.

3. REQUIREMENTS
A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that:

a) it complies with the terms and conditions of this Agreement; and
b) its license agreement:

     i) effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;

     ii) effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits;

     iii) states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and

     iv) states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.

When the Program is made available in source code form:

     a) it must be made available under this Agreement; and
     b) a copy of this Agreement must be included with each copy of the Program.

Each Contributor must include the following in a conspicuous location in the Program:

Copyright © [insert date here], Synopsys, Inc. and others. All Rights reserved.

In addition, each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.

4. COMMERCIAL DISTRIBUTION
Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and indemnify every other Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified Contributor must: a) promptly notify the Commercial Contributor in writing of such claim, and b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense.

For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is then a Commercial Contributor. If that Commercial Contributor then makes performance claims, or offers warranties related to Product X, those performance claims and warranties are such Commercial Contributor's responsibility alone. Under this section, the Commercial Contributor would have to defend claims against the other Contributors related to those performance claims and warranties, and if a court requires any other Contributor to pay any damages as a result, the Commercial Contributor must pay those damages.

5. NO WARRANTY
EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement, including but not limited to the risks and costs of program errors,

compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

6. DISCLAIMER OF LIABILITY
EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. GENERAL
If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

If Recipient institutes patent litigation against a Contributor with respect to a patent applicable to software (including a cross-claim or counterclaim in a lawsuit), then any patent licenses granted by that Contributor to such Recipient under this Agreement shall terminate as of the date such litigation is filed. In addition, If Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s), then such Recipient's rights granted under Section 2(b) shall terminate as of the date such litigation is filed.

All Recipient's rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient's rights under this Agreement terminate, Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable. However, Recipient's obligations under this Agreement and any licenses granted by Recipient relating to the Program shall continue and survive.

Synopsys may publish new versions (including revisions) of this Agreement from time to time. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to distribute the Program (including its Contributions) under the new version. No one other than Synopsys has the right to modify this Agreement. Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved.

This Agreement is governed by the laws of California, without reference to conflict of laws principles. Each party waives its rights to a jury trial in any resulting

```
litigation.
```

LZ4

- Version: 1.9.4
- Source: https://lz4.org/
- License type: lz4
  BSD 2-Clause License (http://www.opensource.org/licenses/bsd-license.php)

```
LZ4 Library
Copyright (c) 2011-2016, Yann Collet
All rights reserved.

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

* Redistributions of source code must retain the above copyright notice, this
  list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this
  list of conditions and the following disclaimer in the documentation and/or
  other materials provided with the distribution.

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

PNPOLY - Point Inclusion in Polygon Test

- Source: https://wrf.ecse.rpi.edu//Research/Short_Notes/pnpoly.html
- License type: Public Domain

```
Copyright (c) 1970-2003, Wm. Randolph Franklin

Permission is hereby granted, free of charge, to any person obtaining a copy of this
software and associated documentation files (the "Software"), to deal in the Software
without restriction, including without limitation the rights to use, copy, modify,
merge, publish, distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject to the following
conditions:

    Redistributions of source code must retain the above copyright notice, this list
of conditions and the following disclaimers.
    Redistributions in binary form must reproduce the above copyright notice in the
documentation and/or other materials provided with the distribution.
    The name of W. Randolph Franklin may not be used to endorse or promote products
derived from this Software without specific prior written permission.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF
CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE
OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

String Functions (strcmp et al)

- License type: BSD

```
Copyright (c) 1992, 1993
        The Regents of the University of California.  All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software
   must display the following acknowledgement:
        This product includes software developed by the University of
        California, Berkeley and its contributors.
4. Neither the name of the University nor the names of its contributors
   may be used to endorse or promote products derived from this software
   without specific prior written permission.

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

Tcl/Tk

- Version: 8.6.14
- Source: https://www.tcl.tk/software/tcltk/
- License type: Custom

```
This software is copyrighted by the Regents of the University of
California, Sun Microsystems, Inc., Scriptics Corporation, ActiveState
Corporation and other parties.  The following terms apply to all files
associated with the software unless explicitly disclaimed in
individual files.

The authors hereby grant permission to use, copy, modify, distribute,
and license this software and its documentation for any purpose, provided
that existing copyright notices are retained in all copies and that this
notice is included verbatim in any distributions. No written agreement,
license, or royalty fee is required for any of the authorized uses.
Modifications to this software may be copyrighted by their authors
and need not follow the licensing terms described here, provided that
the new terms are clearly indicated on the first page of each file where
they apply.

IN NO EVENT SHALL THE AUTHORS OR DISTRIBUTORS BE LIABLE TO ANY PARTY
FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES
ARISING OUT OF THE USE OF THIS SOFTWARE, ITS DOCUMENTATION, OR ANY
DERIVATIVES THEREOF, EVEN IF THE AUTHORS HAVE BEEN ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.

THE AUTHORS AND DISTRIBUTORS SPECIFICALLY DISCLAIM ANY WARRANTIES,
INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT.  THIS SOFTWARE
IS PROVIDED ON AN "AS IS" BASIS, AND THE AUTHORS AND DISTRIBUTORS HAVE
NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR
MODIFICATIONS.

GOVERNMENT USE: If you are acquiring this software on behalf of the
U.S. government, the Government shall have only "Restricted Rights"
in the software and related documentation as defined in the Federal
Acquisition Regulations (FARs) in Clause 52.227.19 (c) (2).  If you
are acquiring the software on behalf of the Department of Defense, the
software shall be classified as "Commercial Computer Software" and the
Government shall have only "Restricted Rights" as defined in Clause
252.227-7014 (b) (3) of DFARs.  Notwithstanding the foregoing, the
authors grant the U.S. Government and others acting in its behalf
permission to use and distribute the software in accordance with the
terms specified in this license.
```

Tcllib

- Version: 2.0
- Source: https://www.tcl.tk/software/tcllib/
- License type: Custom

```
This software is copyrighted by Ajuba Solutions and other parties.
The following terms apply to all files associated with the software
unless explicitly disclaimed in individual files.

The authors hereby grant permission to use, copy, modify, distribute,
and license this software and its documentation for any purpose,
provided that existing copyright notices are retained in all copies
and that this notice is included verbatim in any distributions. No
written agreement, license, or royalty fee is required for any of the
authorized uses.  Modifications to this software may be copyrighted by
their authors and need not follow the licensing terms described here,
provided that the new terms are clearly indicated on the first page of
each file where they apply.

IN NO EVENT SHALL THE AUTHORS OR DISTRIBUTORS BE LIABLE TO ANY PARTY
FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES
ARISING OUT OF THE USE OF THIS SOFTWARE, ITS DOCUMENTATION, OR ANY
DERIVATIVES THEREOF, EVEN IF THE AUTHORS HAVE BEEN ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.

THE AUTHORS AND DISTRIBUTORS SPECIFICALLY DISCLAIM ANY WARRANTIES,
INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND
NON-INFRINGEMENT.  THIS SOFTWARE IS PROVIDED ON AN "AS IS" BASIS, AND
THE AUTHORS AND DISTRIBUTORS HAVE NO OBLIGATION TO PROVIDE
MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

GOVERNMENT USE: If you are acquiring this software on behalf of the
U.S. government, the Government shall have only "Restricted Rights" in
the software and related documentation as defined in the Federal
Acquisition Regulations (FARs) in Clause 52.227.19 (c) (2).  If you
are acquiring the software on behalf of the Department of Defense, the
software shall be classified as "Commercial Computer Software" and the
Government shall have only "Restricted Rights" as defined in Clause
252.227-7013 (c) (1) of DFARs.  Notwithstanding the foregoing, the
authors grant the U.S. Government and others acting in its behalf
permission to use and distribute the software in accordance with the
terms specified in this license.
```

TkDNDlib

- Version: 2.9.3
- Source: https://github.com/petasis/tkdnd
- License type: Custom

```
This software is copyrighted by:
George Petasis, National Centre for Scientific Research "Demokritos",
Aghia Paraskevi, Athens, Greece.
e-mail: petasis@iit.demokritos.gr

The following terms apply to all files associated
with the software unless explicitly disclaimed in individual files.

The authors hereby grant permission to use, copy, modify, distribute,
and license this software and its documentation for any purpose, provided
that existing copyright notices are retained in all copies and that this
notice is included verbatim in any distributions. No written agreement,
license, or royalty fee is required for any of the authorized uses.
Modifications to this software may be copyrighted by their authors
and need not follow the licensing terms described here, provided that
the new terms are clearly indicated on the first page of each file where
they apply.

IN NO EVENT SHALL THE AUTHORS OR DISTRIBUTORS BE LIABLE TO ANY PARTY
FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES
ARISING OUT OF THE USE OF THIS SOFTWARE, ITS DOCUMENTATION, OR ANY
DERIVATIVES THEREOF, EVEN IF THE AUTHORS HAVE BEEN ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.

THE AUTHORS AND DISTRIBUTORS SPECIFICALLY DISCLAIM ANY WARRANTIES,
INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT.  THIS SOFTWARE
IS PROVIDED ON AN "AS IS" BASIS, AND THE AUTHORS AND DISTRIBUTORS HAVE
NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR
MODIFICATIONS.
```

Tklib

- Version: 0.8
- Source: https://www.tcl.tk/software/tklib/
- License type: Custom

```
This software is copyrighted by Ajuba Solutions and other parties.
The following terms apply to all files associated with the software unless
explicitly disclaimed in individual files.

The authors hereby grant permission to use, copy, modify, distribute,
and license this software and its documentation for any purpose, provided
that existing copyright notices are retained in all copies and that this
notice is included verbatim in any distributions. No written agreement,
license, or royalty fee is required for any of the authorized uses.
Modifications to this software may be copyrighted by their authors
and need not follow the licensing terms described here, provided that
the new terms are clearly indicated on the first page of each file where
they apply.

IN NO EVENT SHALL THE AUTHORS OR DISTRIBUTORS BE LIABLE TO ANY PARTY
FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES
ARISING OUT OF THE USE OF THIS SOFTWARE, ITS DOCUMENTATION, OR ANY
DERIVATIVES THEREOF, EVEN IF THE AUTHORS HAVE BEEN ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.

THE AUTHORS AND DISTRIBUTORS SPECIFICALLY DISCLAIM ANY WARRANTIES,
INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT.  THIS SOFTWARE
IS PROVIDED ON AN "AS IS" BASIS, AND THE AUTHORS AND DISTRIBUTORS HAVE
NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR
MODIFICATIONS.

GOVERNMENT USE: If you are acquiring this software on behalf of the
U.S. government, the Government shall have only "Restricted Rights"
in the software and related documentation as defined in the Federal
Acquisition Regulations (FARs) in Clause 52.227.19 (c) (2).  If you
are acquiring the software on behalf of the Department of Defense, the
software shall be classified as "Commercial Computer Software" and the
Government shall have only "Restricted Rights" as defined in Clause
252.227-7013 (c) (1) of DFARs.  Notwithstanding the foregoing, the
authors grant the U.S. Government and others acting in its behalf
permission to use and distribute the software in accordance with the
terms specified in this license.
```

ZLib

- Version: 1.3.1
- Source: https://www.zlib.net/
- License type: zlib

```
(C) 1995-2017 Jean-loup Gailly and Mark Adler

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

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

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

 Jean-loup Gailly        Mark Adler
 jloup@gzip.org          madler@alumni.caltech.edu
```

Except where otherwise noted in the source code (e.g. the files hash.c, list.c and the trio files, which are covered by a similar licence but with different Copyright notices) all the files are:

```
Copyright (C) 1998-2012 Daniel Veillard.  All Rights Reserved.
```

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is fur- nished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FIT- NESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Bundled Demo Files

RTLvision PRO includes demo HDL designs and an Open Source design kit with the following third party licenses.

SOLDERPAD HARDWARE LICENSE

- License type: The Solderpad Hardware License is closely based on the Apache 2.0 software license

```
SOLDERPAD HARDWARE LICENSE version 0.51

This license is based closely on the Apache License Version 2.0, but is not
approved or endorsed by the Apache Foundation. A copy of the non-modified
Apache License 2.0 can be found at http://www.apache.org/licenses/LICENSE-2.0.

As this license is not currently OSI or FSF approved, the Licensor permits any
Work licensed under this License, at the option of the Licensee, to be treated
as licensed under the Apache License Version 2.0 (which is so approved).

This License is licensed under the terms of this License and in particular
clause 7 below (Disclaimer of Warranties) applies in relation to its use.

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and
distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the Rights owner or entity authorized by the Rights owner
that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities
that control, are controlled by, or are under common control with that entity.
For the purposes of this definition, "control" means (i) the power, direct or
indirect, to cause the direction or management of such entity, whether by
contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the
outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising
permissions granted by this License.

"Rights" means copyright and any similar right including design right (whether
registered or unregistered), semiconductor topography (mask) rights and
database rights (but excluding Patents and Trademarks).

"Source" form shall mean the preferred form for making modifications, including
but not limited to source code, net lists, board layouts, CAD files,
documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or
translation of a Source form, including but not limited to compiled object
code, generated documentation, the instantiation of a hardware design and
conversions to other media types, including intermediate forms such as
bytecodes, FPGA bitstreams, artwork and semiconductor topographies (mask
```

```
works).

"Work" shall mean the work of authorship, whether in Source form or other
Object form, made available under the License, as indicated by a Rights notice
that is included in or attached to the work (an example is provided in the
Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that
is based on (or derived from) the Work and for which the editorial revisions,
annotations, elaborations, or other modifications represent, as a whole, an
original work of authorship. For the purposes of this License, Derivative Works
shall not include works that remain separable from, or merely link (or bind by
name) or physically connect to or interoperate with the interfaces of, the Work
and Derivative Works thereof.

"Contribution" shall mean any design or work of authorship, including the
original version of the Work and any modifications or additions to that Work or
Derivative Works thereof, that is intentionally submitted to Licensor for
inclusion in the Work by the Rights owner or by an individual or Legal Entity
authorized to submit on behalf of the Rights owner. For the purposes of this
definition, "submitted" means any form of electronic, verbal, or written
communication sent to the Licensor or its representatives, including but not
limited to communication on electronic mailing lists, source code control
systems, and issue tracking systems that are managed by, or on behalf of, the
Licensor for the purpose of discussing and improving the Work, but excluding
communication that is conspicuously marked or otherwise designated in writing
by the Rights owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf
of whom a Contribution has been received by Licensor and subsequently
incorporated within the Work.

2. Grant of License. Subject to the terms and conditions of this License, each
Contributor hereby grants to You a perpetual, worldwide, non-exclusive,
no-charge, royalty-free, irrevocable license under the Rights to reproduce,
prepare Derivative Works of, publicly display, publicly perform, sublicense,
and distribute the Work and such Derivative Works in Source or Object form and
do anything in relation to the Work as if the Rights did not exist.

3. Grant of Patent License. Subject to the terms and conditions of this
License, each Contributor hereby grants to You a perpetual, worldwide,
non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this
section) patent license to make, have made, use, offer to sell, sell, import,
and otherwise transfer the Work, where such license applies only to those
patent claims licensable by such Contributor that are necessarily infringed by
their Contribution(s) alone or by combination of their Contribution(s) with the
Work to which such Contribution(s) was submitted. If You institute patent
litigation against any entity (including a cross-claim or counterclaim in a
lawsuit) alleging that the Work or a Contribution incorporated within the Work
constitutes direct or contributory patent infringement, then any patent
licenses granted to You under this License for that Work shall terminate as of
```

the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

    You must give any other recipients of the Work or Derivative Works a copy of this License; and

    You must cause any modified files to carry prominent notices stating that You changed the files; and

    You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

    If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License. You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY

```
KIND, either express or implied, including, without limitation, any warranties
or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
PARTICULAR PURPOSE. You are solely responsible for determining the
appropriateness of using or redistributing the Work and assume any risks
associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in
tort (including negligence), contract, or otherwise, unless required by
applicable law (such as deliberate and grossly negligent acts) or agreed to in
writing, shall any Contributor be liable to You for damages, including any
direct, indirect, special, incidental, or consequential damages of any
character arising as a result of this License or out of the use or inability to
use the Work (including but not limited to damages for loss of goodwill, work
stoppage, computer failure or malfunction, or any and all other commercial
damages or losses), even if such Contributor has been advised of the
possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or
Derivative Works thereof, You may choose to offer, and charge a fee for,
acceptance of support, warranty, indemnity, or other liability obligations
and/or rights consistent with this License. However, in accepting such
obligations, You may act only on Your own behalf and on Your sole
responsibility, not on behalf of any other Contributor, and only if You agree
to indemnify, defend, and hold each Contributor harmless for any liability
incurred by, or claims asserted against, such Contributor by reason of your
accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
```

SkyWater Open Source PDK

- License type: Apache 2.0 license

```
Copyright 2020 SkyWater PDK Authors

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

Appendix B: Changelog

This is the list of changes for each release. The most recent changes are at the top.

RTLvision PRO 2024

This is a major release, the following features were fixed and/or added:

- Add support to add a vector to a group in the Wave window.
- Create group at the current location in the Wave window.
- Add the GUI API command `gui wave registerTreeStateChangedCallback` to register a callback that is evaluated every time before the state of the loaded signal tree in the Wave window will change.
- Add the GUI API command `gui wave signalState` to get the open/close state of a signal.
- Add the GUI API command `gui wave registerSelectionChangedCallback` to register a callback that is evaluated every time the selection of the loaded signal list in the Wave window has changed.
- Speed-up annotating Wave values in the Schem and Cone window.
- Add the GUI API command `gui window registerNameChangedCallback` to register a callback that is evaluated every time when the name of a window has changed.
- Add the GUI API command `gui window registerMoveCallback` to register a callback that is evaluated every time when a window is moved to another pane.
- Add new option to show the original RTL name without parameters in the tree.

In addition, the following features were fixed and/or added:

- Enhance the `gui cone load` and `gui cone append` functions and add the option `-foldModules` to fold all loaded module instances.

RTLvision PRO 2023.1.7

This is a maintenance release, the following features were fixed and/or added:

- Preserve the displayed window name while detaching a window with a custom name.
- Speed-up the GUI API command `gui extract setData`.

RTLvision PRO 2023.1.6

This is a maintenance release, the following features were fixed and/or added:

- Display the range of bus objects in the Tree window.
- Show bus members in the Tree window according to the defined range direction.
- Fix a crash in the Search window.
- Enhance `slibconv` to guess the port direction of converted slib symbol files.

RTLvision PRO 2023.1.5

This is a maintenance release, the following features were fixed and/or added:

- Enhance the GUI API command `gui busy` to avoid "recursive call" errors.
- Add support to set an attribute format string at virtual objects using the @nlv:virtual meta attribute.
- Fix the GUI API commands `gui schem contents` and `gui cone contents` filtering for pin, pinBus and netBus object types.
- Enhance the LEF/DEF reader to store more attributes at each generated database object.

RTLvision PRO 2023.1.4

This is a maintenance release, the following features were fixed and/or added:

- Fix updating the object attributes of the Infobox window.
- Fix scanning the display attribute format string in the Select Attributes dialog.
- Fix the GUI API command `gui busy false` for new toplevel windows created while in busy mode.
- Includes the Verific February 2024 Software Release.

RTLvision PRO 2023.1.3

This is a maintenance release, the following features were fixed and/or added:

- Fix moving a group in the Wave window.
- Add the gui setting 'infobox:hideFlags' to hide the Flags pane of the Infobox window.
- Add the GUI API command `gui busy` to set the busy state of the GUI.
- Increase the size limit for display labels in the Schem and Cone window.

RTLvision PRO 2023.1.2

This is a maintenance release, the following features were fixed and/or added:

- Goto sets selection in Source window.
- Includes the Verific January 2024 Software Release.

RTLvision PRO 2023.1.1

This is a maintenance release, the following features were fixed and/or added:

- Fix the result list format of the GUI API command `gui wave getSignals`.
- Keep the entire Wave window state when moving the Wave window.

- Fix tracing a bus bit in the Cone window.

RTLvision PRO 2023.1

This is a major release, the following features were fixed and/or added:

- Speed-up the schematic generation for circuits with very wide net buses.
- Speed-up scrolling in the Wave window.

In addition, the following features were fixed and/or added:

- Add the command line option -wdbTop with better top guessing which replaces the deprecated -vcdTop option.
- Extend the Waveform Database and add support to store direction information at a variable.
- Enhance the GUI API command `gui wave` and add the new sub-command getSignals to return the list of signals loaded into the Wave window.
- Enhance the GUI API command `gui wave` and add the new sub-commands setNameMarker and clearNameMarker to set or remove a marker at the signal name in the Wave window.
- Add the GUI API command `gui wave setHighlightTimes` to provide highlight information for a list of objects at once.
- Add the GUI API command `gui wave registerAddSignalCallback` to register a callback that is evaluated every time before new signals are added to the Wave window.
- Add the possibility to address netBus members in the Wave window.
- Detaching a Wave window preserves the loaded signals.
- Fix crash in the WDB reader function oid2varid.
- Includes the Verific December 2023 Software Release.
- Enhance the simple mode of the Search window and try to find an exact match first before matching the given glob style pattern.
- Enhance the GUI API and add the new commands `gui tooltip registerCallback` and `gui tooltip removeCallback` to set and remove a callback procedure to display additional text in the tooltip window.
- Fix the GUI API command `gui settings changed` to not alter customizations in the Highlight menu.
- Add the GUI API commands `gui schem registerChangedCallback` and `gui schem removeChangedCallback` to register a callback procedure that is called if the contents of the Schem window has changed.
- Enhance the GUI API command `gui popup` and add the new sub-commands similar to `gui menu`.
- Enhance the `-fixed` option of the GUI API commands `gui window new` and `gui window insertCustomWidget` to also disable renaming of the tab.
- Enhance the GUI API commands `gui tab setAvailableClasses` and `gui tab getAvailableClasses` add the option `-tabwindow` to set the available classes only in the given Tab window.

- Enhance the GUI API command `gui tab setAvailableClasses` to not show the plus menu if the given list of available classes is empty.

RTLvision PRO 2023.0.2

This is a maintenance release, the following features were fixed and/or added:

- Includes the Verific November 2023 Software Release.
- Nets with the `hide` flag are also hidden in the Tree window.

RTLvision PRO 2023.0.1

This is a maintenance release, the following features were fixed and/or added:

- Fix a crash showing a search result if the database was opened in `quick mode`.
- Add the `-autoPopulate` option to the database API commands `get_inst` and `get_net` to automatically populate modules if the database was opened in `quick mode`.
- Enhance the GUI API command `gui window insertCustomWidget` and add the option -fixed to disable the close button.
- Fix the database command oid createFromString for cases where the object name includes the given hierarchy separator character.
- Fix the Cone Extraction API option "-emptyModAsPrim".

RTLvision PRO 2023

This is a major release, the following features were fixed and/or added:

- Add new Assertion window listing all assertions in the design.
- Enhance the GUI API command `gui menu` and add support to insert new menu items at a given position.

In addition, the following features were fixed and/or added:

- Add the command `gui dnd registerCallback` to the GUI API.
- Enhance the GUI API command `gui window new` and add the option -fixed to disable the close button.
- Enhance the GUI API command `gui wave` and add the new sub-commands to convert OIDs into VarIds and vice versa.
- Add the option to auto populate the Tree if a binfile was opened in `quick mode`.
- Fix the horizontal scrollbar of the Tree window.
- Enhance the open waveform database dialog and add the option to open the Wave window after reading the input file.
- Includes the Verific September 2023 Software Release.

- New database API command $db get_load to get the load of a net or signal.

RTLvision PRO 2022.3.3

This is a maintenance release, the following features were fixed and/or added:

- All GUI components use the @name attribute value as the displayed object name.
- Add the command `gui wave getTimeRange` to the GUI API.
- Fix update in the Tree after the database was populated.
- Enhance the database API command oid print and add support to use the @name attribute as the displayed object name.

RTLvision PRO 2022.3.2

This is a maintenance release, the following features were fixed and/or added:

- Includes the Verific August 2023 Software Release.
- Enhance support for attributes to control the styling of schematic objects.

RTLvision PRO 2022.3.1

This is a maintenance release, the following features were fixed and/or added:

- Includes the Verific July 2023 Software Release.

RTLvision PRO 2022.3

This is a major release, the following features were fixed and/or added:

- This is the first RTLvision PRO release to support the Altair Units license model.
- Enhance the symbol shape for hierarchical instances and allow pins to be placed at the top and bottom.

In addition, the following features were fixed and/or added:

- Enhance storing a symbol library in the database.
- Includes the Verific June 2023 Software Release.
- Embedded macros can be processed with the "Strict Language Checking" option disabled (command line: -pedantic off).
- Fix syntax error in the header section reported by the VCD reader.

RTLvision PRO 7.2.12

This is a maintenance release, the following features were fixed and/or added:

- Make the progress bar local to the RTLvision PRO window.
- Speed-up drawing and updating the schematic in the Schem and Cone windows.

RTLvision PRO 7.2.11

This is a maintenance release, the following features were fixed and/or added:

- Speed-up the Search function.

RTLvision PRO 7.2.10

This is a maintenance release, the following features were fixed and/or added:

- Includes the Verific April 2023 Software Release.
- Extend the GUI API and add `gui wave saveSignals` to save all signals loaded in the Wave window to a file.
- Extend the GUI API and add `gui wave loadSignals` to load signals into the Wave window from a file.
- Extend the GUI API and add `gui window setLabel` to set the display label of the tab for a given window.

RTLvision PRO 7.2.9

This is a maintenance release, the following features were fixed and/or added:

- Enhance the $db write tcl command to support writing custom symbol shapes for I/O ports and P/G stubs.
- Enhance the cadence2symlib.il script for guessing netSet type bulk name attributes.
- Includes the Verific March 2023 Software Release.
- Fix a crash rendering the symbol from the Boolean equations of a multi output function cell.
- Extent the Popup menu of the Wave window and add the option to copy the value of the selected signal.
- Fix the **Group › Rename** and **Group › Remove** entries in the Popup menu of the Wave window.
- Fix application error while moving a custom group in the Wave window.
- Fix selecting multiple signals in the signal selection of the Wave window.

RTLvision PRO 7.2.8

This is a maintenance release, the following features were fixed and/or added:

- Fix a crash in the spos database API, when adding a source position to a file that has been deleted from the database.

RTLvision PRO 7.2.7

This is a maintenance release, the following features were fixed and/or added:

- Increase the number of supported source files in the spos database.
- Fix displaying the object attributes after opening the Infobox.
- Enhance support to display self transitions in the Wave window also for X and Z values.

RTLvision PRO 7.2.6

This is a maintenance release, the following features were fixed and/or added:

- Includes the Verific February 2023 Software Release.
- Add support to toggle the nethide mode also in the Cone window.
- Fix displaying custom symbol shapes for P/G netstubs and I/O ports in the Infobox.
- Add support to display self transitions in the Wave window.

RTLvision PRO 7.2.5

This is a maintenance release, the following features were fixed and/or added:

- Enhance the cadence2symlib.il script and improve the access direction guessing to avoid overlapping wires.
- Add a horizontal scrollbar to the Tree window.
- The Verilog netlist parser uses cells created by the Liberty reader for blackbox instances.
- Add the new command line option `-skipMuxDetection` to the Liberty reader.
- Avoid a bad terminal order in the symbol shape created by the Liberty reader.

RTLvision PRO 7.2.4

This is a maintenance release, the following features were fixed and/or added:

- Includes the Verific December 2022 Software Release.
- Add support to display graphical comments for net and netBus objects in the Schem and Cone window.
- Fix a crash if an incompatible binfile is opened in quick mode.
- Enhance the database clone API to ignore spos errors and continue cloning the structure.
- Increase the maximum number of supported source files.
- Enhance the "$db cone" API command and add the option "-filterLogicalInvalid" to ignore logically invalid input paths at gates with constants.

RTLvision PRO 7.2.3

This is a maintenance release, the following features were fixed and/or added:

- Includes the Verific November 2022 Software Release.
- Add the new command line option -localincdir to specify an include directory only valid for the next file on the command line.
- Add the new command line option -localdefine to specify a macro only valid for the next file on the command line.
- Windows binaries are now signed.

RTLvision PRO 7.2.2

This is a maintenance release, the following features were fixed and/or added:

- The Search window can automatically populate a binfile opened in quick mode.
- Remove Hierarchy singlizes only the selected module if Signal Mode is turned off.

RTLvision PRO 7.2.1

This is a maintenance release, the following features were fixed and/or added:

- Includes the Verific October 2022 Software Release.
- Add instance orientation and port placement information to the schematic cache.
- Reduce the memory consumption of the fast flat.

RTLvision PRO 7.2.0

This is a major release, the following features were fixed and/or added:

- Add an alternative, more colorful icon style which can be enabled via the Preferences dialog.
- Speed-up opening a binfile.
- Add flat ID support for nets, netBuses and signals including C- and Tcl-APIs.

In addition, the following features were fixed and/or added:

- Includes the Verific September 2022 Software Release.
- Add command line option to load project file via `-project`.
- Make mouse/mouse wheel bindings configurable via the Preferences dialog.
- The display of database object and GUI help tooltips can be configured individually in the Preferences dialog.
- Add popup menu to the Source window's "Source file" combobox to copy the file path.
- Remove obsolete Tcl-API functions related to "mmap", e.g. `zdb hasmmap`, `$db mmap ⋯`, etc.

- Remove obsolete C-API functions related to "mmap", e.g. `zPreloadDataBase`, `zGetDatabaseUsedSize`, etc.
- Enhance the WDB API to support adding scopes and signals to a loaded waveform database.
- Enhance the progress bar of the Cone Extract dialog.
- Add the API command `gui window setToplevelTitle ⋯` to explicitly set the title of top-level windows.
- Add the API commands `gui tab getAvailableClasses` and `gui tab setAvailableClasses ⋯` to get/set the list of window classes that can be created using the Tab window's "+" button.
- Enhance the Save Schematic as Image dialog and add the option to copy a PNG image to the clipboard.
- Change the progress dialog to a top-level window, so that it appears above all other windows.

RTLvision PRO 7.1.9

This is a maintenance release, the following features were fixed and/or added:

- Includes the Verific July 2022 Software Release.

RTLvision PRO 7.1.8

This is a maintenance release, the following features were fixed and/or added:

- Includes the Verific June 2022 Software Release.
- Fix EDIF parser for sub-ports of portBundle instances in joins.

RTLvision PRO 7.1.7

This is a maintenance release, the following features were fixed and/or added:

- Includes the Verific May 2022 Software Release.

RTLvision PRO 7.1.6

This is a maintenance release, the following features were fixed and/or added:

- Includes the Verific April 2022 Software Release.
- Add new settings to overwrite the time marker and cursor label of the Wave window using the `gui settings` API.
- Line wrapping in the tooltips' source previews.
- Wrap & scroll long title labels in the Source window's Action Bar.
- Add API function `gui window isStatusPaneVisible`.
- Add API function `gui window getPaneSashes`.

- Add API functions `gui window getState` and `gui window setState`.
- Display user defined highlight colors in the color selection of the toolbar and main menu.

RTLvision PRO 7.1.5

This is a maintenance release, the following features were fixed and/or added:

- Includes the Verific March 2022 Software Release.
- Add popup menu to the Source window's "Source file" combobox to copy the file path.
- Enhance error handling for non existing files in the Liberty parser.
- The `gui wave show` command no longer apply visual feedback for the added signals.
- Also restore symbol libraries with the `Restore Settings` option in the File Open dialog.
- Enhance the `symmap` mapping for symbol libraries and add support to define a port name mapping.
- Enhance the cone extraction API and add the option to continue tracing the cone through power/ground nets.
- Fix opening the 'Report Instance Count' dialog if no module is displayed in the Schem window.
- Enhance the "Expand Nets" feature of the Cone window and highlight all added objects with the goto color.

RTLvision PRO 7.1.4

This is a maintenance release, the following features were fixed and/or added:

- Includes the Verific January 2022 Software Release.

RTLvision PRO 7.1.3

This is a maintenance release, the following features were fixed and/or added:

- Includes the Verific December 2021 Software Release.
- Add 'ZoomChanged' callbacks for Schem and Cone windows.

RTLvision PRO 7.1.2

This is a maintenance release, the following features were fixed and/or added:

- Includes the Verific November 2021 Software Release.
- Add support to drag results from the inline search result window in the Schem and Cone window.
- Sort the result list displayed in the inline search result window in the Schem and Cone window.
- Clear the inline search result list of the Schem and Cone window when loading a new design.

- Fix the display order of the Plugins list in the Plugins dialog.
- Fix the option to Singlize the parent module in the "Create Hierarchy" dialog.
- Restrict effect of double clicking items in the Tree window to the current top-level window.
- Add support for permanent zoom independent graphical marks in the Schem and Cone window.

RTLvision PRO 7.1.1

This is a maintenance release, the following features were fixed and/or added:

- Remove the option to find objects in the Cone from the Search window.
- Adjust the height of the inline search results window in the Schem and Cone window.
- Fix a potential crash while selecting an autobundle in the Schem or Cone window.

RTLvision PRO 7.1.0

This is a major release, the following features were fixed and/or added:

- Use a new compressed database binfile format.
- Add the option to open a binfile in quick mode.
- Add the capability to search the content loaded to the Schem and Cone window.
- Enhance the jump to prev/next value change in the Wave window to stop at a given value.
- Enhance the Infobox: Add filter to display chosen range of buses.
- Make all GUI fonts configurable via the Preferences dialog.
- Plugins that create a tabbed window can be disabled by closing the tab.
- Add support for VHDL 2019.

In addition, the following features were fixed and/or added:

- Indicate additional clock sources in the Clock Domain Analyzer.
- Add option to $db write to disable writing `celldefine and `endcelldefine.
- Remove the obsolete parameter -readonly from the zdb open command.
- Add bin directory to provide platform independent access to all tools.
- Add the option to automatically highlight objects added to the Cone window.
- Add target Cone window submenu to all Cone popup menu entries.
- Add keyboard shortcuts kbd:\[Ctrl-z\] and kbd:\[Ctrl-y\] for history navigation in the Schem, Cone and Source window.
- Format keyboard sequences in menus and the documentation more naturally.
- Fix placement of the Magnify window.
- Includes the Verific October 2021 Software Release.

NOTE	Version 7.0 of RTLvision PRO introduces a number of user interface changes. We are summarizing them in a short demo video. You may want to watch the video to get familiar with the changes.

RTLvision PRO 7.0.18

This is a maintenance release, the following features were fixed and/or added:

- Includes the Verific September 2021 Software Release.
- Fix scrolling in the Console window.
- Remove escaping from multi-dimensional port names created by the Verilog netlist parser.
- A Search in the Cone window can be done on any object type.
- Add progress updates to `$db oper singlizeTree ⋯`.
- Fix symbol mapping.

RTLvision PRO 7.0.17

This is a maintenance release, the following features were fixed and/or added:

- Fix SPOS for ZDBs loaded from SDBL files via the `-sdbl` command line option.
- Fix setting port directions from a symbol library at top level modules.

RTLvision PRO 7.0.16

This is a maintenance release, the following features were fixed and/or added:

- Fix progress bar of the Liberty parser.
- Allow for more flexible bus member name syntax.
- Fix output image dimensions in the Save Schematic as Image dialog.
- Add `$db oper flattenSubtree` / `zOperFlattenSubtree`.

RTLvision PRO 7.0.15

This is a maintenance release, the following features were fixed and/or added:

- Includes the Verific August 2021 Software Release.
- Correct the progress bar for saving a ZDB binfile.
- Inline error messages in the Cone Extraction dialog.
- Enhance the cadence2symlib.il script to avoid bad parameter type cast for cyclic types.
- Enhance the cadence2symlib.il script and avoid expressions for the default value in the model name mapping field.

RTLvision PRO 7.0.14

This is a maintenance release, the following features were fixed and/or added:

- Fix the `gui wave registerMarkerChangedCallback` to append the time value including the unit.

RTLvision PRO 7.0.13

This is a maintenance release, the following features were fixed and/or added:

- Includes the Verific July 2021 Software Release.

RTLvision PRO 7.0.12

This is a maintenance release, the following features were fixed and/or added:

- Add a number of new `gui window ⋯` sub-commands: `gui window createToplevel`, `gui window getMainVerticalPane`, `gui window createHorizontalPane`, `gui window createTab`, `gui window setPaneSashes`, `gui window maximize`, `gui window unmaximize`, `gui window isMaximized`, `gui window setGeometry`, `gui window hide`, `gui window unhide`.
- Add a `-tabwindow $w` option to `gui window insertCustomWidget ⋯` to place the custom widget in an arbitrary Tab window.
- Add a section about customizing the GUI layout to the GUI Customization Tutorial.
- Add fix the evaluation order of "Window Created" callbacks (set with `gui window registerCreatedCallback`).
- Extend the GUI API and add `gui attribute registerGetCallback` and `gui attribute removeGetCallback` to register and remove a callback to get attributes from an external source to be displayed in OID Tooltips and in the attributes list of the Infobox.
- Fix crash in the Verilog netlist parser after a syntax error in the instance connectivity.
- Allow for arbitrarily sized prefixes when using `$db oper rmhier ⋯`.

RTLvision PRO 7.0.11

This is a maintenance release, the following features were fixed and/or added:

- Includes the Verific June 2021 Software Release.
- Fix crash when calling `$db find` with an empty `-path`.

RTLvision PRO 7.0.10

This is a maintenance release, the following features were fixed and/or added:

- Clear placeholder text of filter text fields before pasting with the middle mouse button.

RTLvision PRO 7.0.9

This is a maintenance release, the following features were fixed and/or added:

- Includes the Verific May 2021 Software Release.
- Fix restoring of Bookmarks via `gui bookmark set ⋯`.
- Fix adding Bookmarks using the keyboard shortcut.
- Add `gui cone registerChangedCallback` and `gui cone removeChangedCallback`.
- Fix the `gui schem fold/unfold` API commands for unsupported OIDs.
- Add new command line option `-pedantic` to the Verilog netlist parser to control suppression of warnings for unsupported behavioral Verilog syntax and to degrade warnings from Verilog libraries.
- Add `gui wave registerTimeRangeChangedCallback` and `gui wave removeTimeRangeChangedCallback`.

RTLvision PRO 7.0.8

This is a maintenance release, the following features were fixed and/or added:

- Add `gui imageset load` to load a set of images suitable for the toolbar.
- Fix image loading in `gui toolbar addButton` and `gui toolbar addCheckButton`.
- Add `gui toolbar children` to get the children of a specified toolbar.
- Delay emitting the "window created" event until the corresponding window is fully created.
- Add `gui settings save ⋯` and `gui settings load ⋯`.

RTLvision PRO 7.0.7

This is a maintenance release, the following features were fixed and/or added:

- Includes the Verific April 2021 Software Release.
- Fix setting the outline color for instances.
- Add the option to set the outline color for primitive and module instances separately.
- Add `gui window registerCurrentChangedCallback` and `gui window removeCurrentChangedCallback`.
- Add `gui window registerCreatedCallback` and `gui window removeCreatedCallback`.
- Add `gui window registerDestroyedCallback` and `gui window removeDestroyedCallback`.
- More robust GUI layout loading if RTLvision PRO is started in iconified mode (`-iconify`).
- Add popup menu option to clone windows.
- Don’t allow for detaching default windows.
- Speed-up detaching, attaching and cloning of Schem windows.
- Add option to show the selected module in an existing or new Schem window to the Tree’s popup menu.

- Fix "integer value too large" error in the Waveform viewer.
- Restore the state of the Wave window after loading a new design database.
- Keep the content of all Schem windows stable when toggling Blocklevel mode.

RTLvision PRO 7.0.6

This is a maintenance release, the following features were fixed and/or added:

- Includes the Verific March 2021 Software Release.
- Fix the `$db setPrimitive` function.

RTLvision PRO 7.0.5

This is a maintenance release, the following features were fixed and/or added:

- If the restored GUI layout is missing any standard windows (Schem, Cone, Source, Tree, Mem), force a reset to the default layout.
- Fix disabling a Plugin to avoid a Tcl error while quitting the RTLvision PRO GUI.
- Enhance the Verilog netlist parser to support built-in gate names as escaped identifiers.

RTLvision PRO 7.0.4

This is a maintenance release, the following features were fixed and/or added:

- Includes the Verific February 2021 (version b) Software Release.
- Add the command line option `-topLibrary` to specify the library containing the top-level design.
- Speed-up loading instances with a large number of pins into the Cone window.
- Fix the "-shortestPath" option of the "$db cone" API command for pin and port targets.
- Add documentation for the `$db htree` command.

RTLvision PRO 7.0.3

This is a maintenance release, the following features were fixed and/or added:

- Includes the Verific February 2021 Software Release.
- Fix matching signal from a waveform database to the loaded netlist database.
- Add the option to keep the current state of the Wave window when reloading the previous waveform database file.
- Include cursors to Wave window bookmarks.
- Add support for the Verilog 2001 Attribute syntax to the Verilog netlist parser.
- Restore default window contents after loading a new GUI layout.

RTLvision PRO 7.0.2

This is a maintenance release, the following features were fixed and/or added:

- Includes the Verific January 2021 Software Release.
- Enhance matching signal from a waveform database to the loaded netlist database.
- The `registerMarkerChangedCallback` callback is executed after all internal callbacks.
- Extend the GUI API with commands to register and remove callbacks to be executed when the values displayed in the Wave window are updated.
- Avoid Tcl error when using the "Comprehensive Mode" of the "Clock Domain Analyzer" with many domains.
- Fix the "Expand Nets" feature of the Cone window for dashed netBus objects.
- Options specified on the command line can overwrite settings restored from the automatically saved workspace.

RTLvision PRO 7.0.1

This is a maintenance release, the following features were fixed and/or added:

- Includes the Verific December 2020 Software Release.
- Display module parameters in the Tree view and in tooltips.
- Fix modeless dialog windows to stay on top of the main GUI.
- Enhance the "Do Not Display Hierarchy Boxes" feature of the Cone window and use the hiersep character to separate the hierarchy of the flat object name.
- Fix the "Expand Nets" feature of the Cone window in combination with the "Do Not Display Hierarchy Boxes" feature.
- Fix help button in the <manual#Photo, Save Schematic as Image>> dialog.
- Enhance the Export, Photo and Print dialogs.
- Extend the GUI API with commands to register and remove callbacks to be executed when the time marker in the Wave window changes.
- Find API: also apply the `-cellref` parameter when searching for pins/ports/nets.
- Search window: also apply "Cell Pattern" of the advanced options when searching for pins/ports/nets.

RTLvision PRO 7.0.0

This is a major release, the following features were fixed and/or added:

- Update the GUI architecture to allow vertical and horizontal splitting of the panes with an arbitrary number of tabs.
- The last GUI layout is automatically restored at startup.

- All global settings are automatically saved as a Workspace file and restored at startup.
- All design specific settings can be saved as a Project file.
- Use the left mouse button as the default for Drag & Drop.
- New Connectivity Browser showing the detailed connectivity of a net.
- Enhance the look and feel of the Statusbar.
- Messages are now displayed in the new Message View window.
- The Tree view is no longer coupled to the Schem window.
- Each Schem window displays the path to the current module in a breadcrumb style tree.
- Enhance the Export, Photo and Print dialogs.
- Decouple loading a waveform database from the Wave window.
- Introduce a new GUI API.

In addition, the following features were fixed and/or added:

- Add new option to $db write command which exports cells in the liberty format.
- Enhance the Export Netlist dialog: add option to write corresponding liberty.
- Use the system browser as the default help viewer.
- Add option in Tree Popup Menu to show all available top modules.
- Rename the file based Cone Bookmark feature to Snapshot.
- Add new option in the Preferences dialog to show an instance based view in the Tree .
- Fix the -y option of the Verilog netlist parser to read only the used library files.
- Enhance port naming for blackboxes of the Verilog netlist parser.
- The default compile mode for all Verilog files is MFCU.
- Disable the relaxed language checking mode of the RTL parser by default.
- Add the option to toggle the relaxed language checking to the "Read RTL" dialog.
- Unify and enhance the way how a parser resolves duplicate cell definitions (e.g. after merging two databases).
- Mouse click keyboard modifier to append objects to the selection in the Schem and Cone window is now the `control` key.
- Extend the GUI API and add commands to work with the global settings.

NOTE	The FlexNet package of this major release was upgrade to version 11.16.3. It is required to update the vendor daemon (`dconcept`) on the license server. The vendor daemon is backward compatible and can serve older versions of RTLvision PRO.

RTLvision PRO 6.12.26

This is a maintenance release, the following features were fixed and/or added:

- Includes the Verific November 2020 Software Release.
- Fix matching multi dimensional buses from VCD to the loaded design in the Wave window. Wave
- Disable the "Create Hierarchy" Popup menu if instances with a different path are selected.
- Preserve net values when creating artificial hierarchy.
- Fix invoking an external editor on Windows (do not force adding the .exe extension).

RTLvision PRO 6.12.25

This is a maintenance release, the following features were fixed and/or added:

- Fix bad name of first netbus member in the Infobox
- Enhance cadence2symlib.il script for batch usage.
- Fix bad placement of menus and dialogs on displays that span multiple screens with a different resolution.

RTLvision PRO 6.12.24

This is a maintenance release, the following features were fixed and/or added:

- The Liberty parser uses the -spos command line option to toggle the creation of file and line attributes.
- Fix displaying modules marked as encrypted.

RTLvision PRO 6.12.23

This is a maintenance release, the following features were fixed and/or added:

- Includes the Verific October 2020 Software Release.
- Modules with the attribute "@encrypted=1" display a pad lock icon in the schematic view.
- Objects of any type can be dropped to the Tree window to open the module containing the dropped object.
- Delay loading object details in the Infobox to avoid double click bug.
- Extend the GUI API and add Gui:AddToolbarCheckbutton to add a checkbutton to a toolbar.
- Extend the GUI API and add Gui:AddToolbarItem to add a custom item to a toolbar.
- Enhance the $db write verilog and $db write spice and add comment with source information.
- New Liberty parser option -storeGroup to specify a group name pattern for groups to be store as attributes in the created ZDB binlib.

RTLvision PRO 6.12.22

This is a maintenance release, the following features were fixed and/or added:

RTLvision PRO 6.12.21

This is a maintenance release, the following features were fixed and/or added:

- Includes the Verific September 2020 Software Release.
- The Verilog netlist parser sets the wire flag at all nets of the type wire.
- The Verilog netlist parser no longer creates an artificial level of hierarchy for arrayed instances.

RTLvision PRO 6.12.20

This is a maintenance release, the following features were fixed and/or added:

- The cadence2symlib.il script continues with the symbol conversion if a cell has no symbol view.
- Add support for arrayed blackbox instances to the Verilog netlist parser.
- Avoid syntax error reported by the Verilog netlist parser reading some System Verilog port declarations.
- Better handling of parameterized modules in the Source window.
- Only update the Infobox if a new object has been selected.

RTLvision PRO 6.12.19

This is a maintenance release, the following features were fixed and/or added:

- Includes the Verific August 2020 Software Release.
- Extend the GUI API and add Gui:SchemUnfold to expand a hierarchical instance in the Schem window.
- Extend the GUI API and add Gui:SchemFold to collapse a hierarchical instance in the Schem window.
- Extend the GUI API and add Gui:SchemIsFolded to check the fold state of a hierarchical instance in the Schem window.

RTLvision PRO 6.12.18

This is a maintenance release, the following features were fixed and/or added:

- Fix bad colors (white text on white background) when the desktop environment is set to a dark color scheme.
- Fix an error in the Schem window creating an artificial level of hierarchy after removing hierarchy.
- Modules with a custom symbol shape can be unfolded in the Schem window.

RTLvision PRO 6.12.17

This is a maintenance release, the following features were fixed and/or added:

- Display the instances of all parameterized modules in the Source window's action bar.
- Use meaningful default values for reading slib libraries without a configuration file.
- Add new command line option -forceSymlib to overwrite existing symbols.
- Avoid unexpected horizontal scrolling in the Wave window while scrolling vertically.
- Always show the expand or collapse icon of a vector signal in the Wave window.
- Extend the GUI API and add Gui:WaveShowMembers to expand or collapse the member view of a netBus in the Wave window.

RTLvision PRO 6.12.16

This is a maintenance release, the following features were fixed and/or added:

- Includes the Verific July 2020 Software Release.
- Fix wrong connection count in the Infobox.
- Avoid bad optimization in the Infobox.
- Avoid warning "Object does not match loaded design" in the Infobox.
- Fix crash/assertion in the Infobox.
- Allow for custom actions when clicking on display graphical object comments.
- Add missing cone arcs for the `WIDE_PRIO_SELECTOR` and `WIDE_CASE_SELECT_BOX` primitives.
- Add a "Top Pattern" option to the advanced Search window to restrict the search to a specific top module.
- Speed up the Search window by searching for direct, exact matches first.
- Print a meaningful error message when the X11 display cannot be opened.
- Clicking the "Hierarchy.Signal Name" label in the Wave window now switches the hierarchy path visibility of signal names.
- Add the option `-noError` to the `zdb source` command to ignore errors in the sourced Tcl script.

RTLvision PRO 6.12.15

This is a maintenance release, the following features were fixed and/or added:

- Fix invoking the Obfuscate Plugin interactively.
- Fix bad initial filename in the "Save Cone As ..." file selection dialog.
- Fix "load inst" database API command to accept the deprecated pin flag "neg".
- Fix search across files being stuck at the current file.
- Add bus-width markers.

RTLvision PRO 6.12.14

This is a maintenance release, the following features were fixed and/or added:

- Fix segfault in the Infobox converting a primitive pin to a port.
- Enhance the "Guess Instance Array" feature to select either primitive or hierarchical instances.

RTLvision PRO 6.12.13

This is a maintenance release, the following features were fixed and/or added:

- Enhance instance names generated by the "Guess Instance Array" function.
- Enhance instance names generated by the "Compact Schematic" function.
- Avoid "integer value too large to represent" error while displaying a source file (only on some displays).
- Fix the "Derive Window Title from the Toplevel Module" option if a binfile is loaded in RTLvision PRO.
- New flat view API command flatoomr foreach to loop over all out of module references (OOMRs).

RTLvision PRO 6.12.12

This is a maintenance release, the following features were fixed and/or added:

- Fix the RTL parser option to write a single Verilog file of the preprocessed input files.
- Fix error when parsing SystemVerilog files into multiply libraries in MFCU mode.
- Fix open legacy ZDB binfiles.
- Properly escape flat attributes when using $db write tcl -flat.

RTLvision PRO 6.12.11

This is a maintenance release, the following features were fixed and/or added:

- Add the option to write a single Verilog file of the preprocessed input files to the RTL parser.
- Increase the limit for elaborating for loops with non-constant conditions.
- Relax the RTL parser for unknown system functions.
- Fix Verilog cross module reference creation.
- Fix importing a fileset in the File Open dialog with undefined file types.
- Enhance support for parsing System Verilog assertions.

RTLvision PRO 6.12.10

This is a maintenance release, the following features were fixed and/or added:

- The quick parse mode of the Liberty reader is now the default.
- Add option to the "Diff" plugin to consider primitives with the same function to be equivalent.

RTLvision PRO 6.12.9

This is a maintenance release, the following features were fixed and/or added:

- Enable hierarchy-tree based elaboration to allow for cross module references.
- Fix adding multiple symbol libraries with a different case sensitivity option.
- Enhance the `tab` completion in the Console window to support namespaces.
- The visible hierarchy separator can be any character.

RTLvision PRO 6.12.8

This is a maintenance release, the following features were fixed and/or added:

- The "load inst" database API command accepts deprecated pin flags for compatibility with legacy Tcl dumps of a ZDB database.
- More robust handling of RTL operators with inconsistent interfaces.
- Force update of net attribute display on wires if attribute values have changed.

RTLvision PRO 6.12.7

This is a maintenance release, the following features were fixed and/or added:

- Fix the "Goto Target" command in the result list in the Cone Extraction dialog.
- Liberty files read through the GUI no longer set the primitive function.
- Fix error handling of the Liberty parser.
- Enhance displaying net attributes at the wire.
- Opening a new Workspace file no longer overwrites selected symbol libraries.
- Extend the GUI API and add Gui:BookmarkGetAll and Gui:BookmarkSetAll to get and set the bookmarks of a window.

RTLvision PRO 6.12.6

This is a maintenance release, the following features were fixed and/or added:

- Fix creating hierarchy using the `$db oper hierAdd ⋯`.
- Fix selection handling in the Schem and Cone window.

RTLvision PRO 6.12.5

This is a maintenance release, the following features were fixed and/or added:

- Fix segmentation fault in the Liberty parser if a function does not match the cell pins.
- Extend the GUI API and add Gui:WindowToPhoto to save a window as displayed on the screen as a PNG photo image.
- Add support to save the Clock Domain Analyzer results as a PNG image.
- Fix reading repeated definitions in a VCD file header.
- Fix error in the "Create Overview" plugin that prevented it from working if a tab other than the Schem tab was selected.

RTLvision PRO 6.12.4

This is a maintenance release, the following features were fixed and/or added:

- Avoid error in toggling the nethide mode after creating an artificial hierarchy.
- Extend the GUI API and add Gui:ClearSchematicCache to clear the internal schematic cache.
- Avoid creation of duplicate module attributes when using Verilog-style attributes.
- Fix the goto command for partly visible objects in the Schem and Cone window.
- Add the option to filter the list of Plugins displayed in the Plugin dialog by tags added to the Plugin script header.
- Speed up filling the Infobox by avoiding unnecessary communication with the license server when creating a new database.
- Changed the spos API command lineno and add support for getting the line number of the last byte (filesize).
- Enhance the documentation of the shipped Userware scripts.

RTLvision PRO 6.12.3

This is a maintenance release, the following features were fixed and/or added:

- Add new command line option -userwareEval to pass a Tcl script as a string to be evaluated.
- Extend the GUI API and add Gui:ShowReadDialog to show the read file dialog.
- Fix processing the command line option -userwareArgs if specified multiple times.
- Fix pasting text into the Console window (Windows platform only).

RTLvision PRO 6.12.2

This is a maintenance release, the following features were fixed and/or added:

- Re-calculate the contents of an open Clock Domain Analyzer dialog when loading a new design.

- Avoid errors in the Clock Domain Analyzer when the design hierarchy is modified via `$db oper addHier/rmHier ⋯`.
- Avoid endless flickering of scrollbars in corner cases.
- Fix keyboard shortcuts Control-PgUp and Control-PgDn in the Source window.
- The RTL parser uses separate namespace for automatically generated primitives (created by elaboration/synthesis) to avoid collision with user defined cells.

RTLvision PRO 6.12.1

This is a maintenance release, the following features were fixed and/or added:

- Avoid an empty Schem window after selecting a custom symbol shape.
- Fix splitting of netBuses when creating hierarchies via `$db oper addHier ⋯` in some special cases.
- Add the command line option `-dontElaborate` to skip the elaboration of specific modules; also extend the File Open dialog with a corresponding input field.

RTLvision PRO 6.12.0

This is a major release, the following features were fixed and/or added:

- The default Verilog file type is now System Verilog.
- Extend the database command $db write and add the option to save the contents of the database as a VHDL netlist.
- Add the option to display graphical object comments in the Schem and Cone window.
- The cone extraction no longer consumes stack memory.

In addition, the following features were fixed and/or added:

- Fix segfault in the Infobox.
- Enhance the setPrimitive API command and distinguish libcells and cells surrounded by the Verilog macros `` `celldefine `` and `` `endcelldefine ``.
- Add the option to treat cell surrounded by the Verilog macros `` `celldefine `` and `` `endcelldefine `` as primitives to the Preferences dialog.
- Change the zoom gesture in the Wave window to use the same mouse button as the zoom gesture in the Schem window.
- Multiple opened Wave windows are now fully independent from each other.
- Unify and enhance the way how a parser resolves duplicate cell definitions.
- Enhance guessing instance arrays of hierarchical modules.
- Expanding the Cone in Signal Mode no longer stops at inout ports.

RTLvision PRO 6.11.6

This is a maintenance release, the following features were fixed and/or added:

- Do not display the tree expansion icon if there are no items below.
- Add all files read by the RTL parser to the database.

RTLvision PRO 6.11.5

This is a maintenance release, the following features were fixed and/or added:

- Fix bug in rotation via popup menu: displayed and stored result is now consistent.
- Clear the current history if the "Remember History" feature of the Schem window is disabled in the Preferences dialog.
- Clear the current history if the "Remember History" feature of the Source window is disabled in the Preferences dialog.

RTLvision PRO 6.11.4

This is a maintenance release, the following features were fixed and/or added:

- Show an error dialog box on Windows if RTLvision PRO is invoked with wrong command line options.
- Fix displaying the tooltips in the Infobox.
- The checkbutton to "Create Named Connectivity"" in the Save Cone as Verilog dialog was missing.
- Fix missing grey color for signals not matching the loaded design in the Wave window.
- Fix missing highlight colors in the signal selection of the Wave window.
- Fix missing highlight colors in the Tree window.
- Clear the current history if the "Remember History" feature of the Cone window is disabled in the Preferences dialog.
- Fix overwriting the displayed net name with an @name attribute.

RTLvision PRO 6.11.3

This is a maintenance release, the following features were fixed and/or added:

- Avoid an error in the Schem window if a colon is used as the hierarchy separator.
- Enhance the Goto command and remove all existing goto highlights are deleted before calling the next Goto command.
- Enhance the "Guess Inst Arrays" feature to also group instance based on their names ignoring the connectivity.

RTLvision PRO 6.11.2

This is a maintenance release, the following features were fixed and/or added:

- Extend the GUI API and add the command Gui:GetLastSelection to get the most recently selected list of objects across all Windows of all Visualizers.
- Extend the GUI API and add the command Gui:WaveCustomizeValueColor to set a custom color for a specific signal and value in the Waveform window.
- The value changes in the Waveform window are drawn slightly thicker.
- Add a new GUI Customization Tutorial.

RTLvision PRO 6.11.1

This is a maintenance release, the following features were fixed and/or added:

- Correct error message reported by the license sub-system in case of an error.
- Fix several crashes if objects are displayed in the Infobox.
- Add a popup menu to the detail view of the Infobox to crossprobe a selected object to other views or load it to the Cone window.
- Add a double click binding to the detail view of the Infobox to crossprobe a displayed object to other views.

RTLvision PRO 6.11.0

This is a major release, the following features were fixed and/or added:

- Add the option to split the right side of a Visualizer.
- Extend the GUI API and add the command Gui:SplitVisualizer to split the right side of a Visualizer.
- Add the option to decorate net objects with additional logical direction indicator icons.
- Drop support for the Solaris platform.
- Drop support for the 32 Bit version of the Linux and Windows platform.

In addition, the following features were fixed and/or added:

- Add a new Cone toolbar button to expand all partially loaded nets and netBuses.
- The Clock Domain Analyzer displays the physical number of connected clock pins.
- Add the command line option `-ignoreUnit` to skip the elaboration of specified VHDL unit; also extend the File Open dialog with a corresponding input field.
- Fix the GUI API command Gui:ShowSearch>> if the Search tab is already created but not raised.
- Fix restoring the Infobox state from a Workspace file.
- Always show the top level schematic after startup in fullfit mode.

- Speed-up the GUI startup time if the network connection to the license server has a high latency.

NOTE	RTLvision PRO 6.10 is the last release that will support the Solaris operating system. Concept Engineering will continue to support Windows and Linux operating systems.

NOTE	RTLvision PRO 6.10 is the last release that will provide a 32 bit version for the Windows and Linux operating systems. Concept Engineering will continue to provide a 64 bit version.

RTLvision PRO 6.10.9

This is a maintenance release, the following features were fixed and/or added:

- Fix displaying the default attributes for MOS devices.
- Honor the special value 0 of the Big Module Limit when inline-expanding instances.

RTLvision PRO 6.10.8

This is a maintenance release, the following features were fixed and/or added:

- Avoid Tcl error using the Popup menu entries "Cone/Load Cone" and "Cone→Append Cone".
- Add missing new line character to the default format string for displaying M factors.

RTLvision PRO 6.10.7

This is a maintenance release, the following features were fixed and/or added:

- The -help output is always printed on stdout.
- Enhance the GUI API commands Gui:LoadCone and Gui:AppendCone and add the option to highlight the loaded objects using the goto color.
- The objects added to the Cone window using the "Extract To" feature are highlighted in the goto color.

RTLvision PRO 6.10.6

This is a maintenance release, the following features were fixed and/or added:

- Clarify UI elements in the Cone Extract dialog.
- Add the option to specify the initial directory for all file dialogs in the GUI.

RTLvision PRO 6.10.5

This is a maintenance release, the following features were fixed and/or added:

- Allow for drag & drop of text/objects from other applications to the Search window.
- Fix saving/loading "Last Recently Used" entries to/from Workspace files.
- Add support for instance and cell attributes to the Verilog netlist parser.

RTLvision PRO 6.10.4

This is a maintenance release, the following features were fixed and/or added:

RTLvision PRO 6.10.3

This is a maintenance release, the following features were fixed and/or added:

- Do not show source code preview in the tooltip for large files.
- Assign a minimum size to each paned window to avoid zero size panes.
- Avoid flickering when additional paned windows are shown or hidden.

RTLvision PRO 6.10.2

This is a maintenance release, the following features were fixed and/or added:

- Fix expansion of environment variables in Verilog options.
- Avoid Tcl error while scrolling in the PrimeTime Plugin.
- Fix "Last Recently Used" entries.
- Add new Userware example "viewLiberty.tcl" to load an instance of each Liberty cell into the Schem window.

RTLvision PRO 6.10.1

This is a maintenance release, the following features were fixed and/or added:

- Opening a settings file from the File menu invokes the parser to open the design data.
- The sym2zdb command creates additional cells for each "spice" mapping.

RTLvision PRO 6.10.0

This is a major release, the following features were fixed and/or added:

- Extend the GUI API and add the command Gui:AddToolbarButton to add custom toolbar buttons.
- Extend the GUI API and add the command Gui:RemoveToolbarButton to remove a toolbar button.
- The Infobox is now displayed in the bottom tab.
- Enhanced the detail view of the Infobox to display graphical information of the selected object.

- Enhance displaying net names at wires.

In addition, the following features were fixed and/or added:

- A settings file that contains command line options can be opened from the File menu.
- Add the option to connect nets by name to the File Open dialog.
- Add the option to reset the settings of the File Open dialog.
- Enhance the write verilog command and add the option to add function implementation that can be used to run simulation.
- Set the default print mode to. Color.
- Add new database operator mergeParallelInst to merge parallel connected instances.
- Enhance the "$db oid" command and add "access functions" for the pins of a netSeg "OID"
- Add the option to create hierarchy for function/procedure calls (via "Add Hierarchy for Function Calls" in the File Open dialog, or via `-funcHier on` from the command line).
- Add the command line option `-connectByName <pattern>` to connect matching nets by name.
- History navigation in the Schem window now includes moved objects.
- A selection in the Tree window updates the Last Selection label.

RTLvision PRO 6.9.12

This is a maintenance release, the following features were fixed and/or added:

- Fix the GUI API commands to extend the main menu for menu items with an underscore character in the label.

RTLvision PRO 6.9.11

This is a maintenance release, the following features were fixed and/or added:

- Fix and enhance the "Create Overview" plugin.
- Add the option to select the source for the window title.
- Avoid latency issues in the Console window by buffering messages.
- Fix the advanced Search when the path pattern contains wildcards and the name pattern doesn’t.
- Fix removing Trace Through Cells from the Preferences dialog.
- Enhance drawing the Wave window with enabled font scaling.

RTLvision PRO 6.9.10

This is a maintenance release, the following features were fixed and/or added:

- Ignore preceding white space in object names written to the netlist created by the "$db write

spice" command.

- Enhance the "$db write spice" command for constant assigns.
- Fix truncation of the displayed design name in the title bar.
- Fix missing design name after setting a new top module.
- Fix memory leaks when using `$db cone` with the `-shortestPath` option.

RTLvision PRO 6.9.9

This is a maintenance release, the following features were fixed and/or added:

- Adjust row height of the Tree window when using large fonts.
- Adding a net with no driver to the Cone window now displays the net connected to any pin.
- Enhance design name display in the title bar.
- Take the `-L` option into account when instantiating VHDL entities from Verilog.

RTLvision PRO 6.9.8

This is a maintenance release, the following features were fixed and/or added:

- The Verilog netlist parser continues reading a module after a parameter with a constant real value.
- Enhance support for nested macros in Verilog netlists.

RTLvision PRO 6.9.7

This is a maintenance release, the following features were fixed and/or added:

- Ignore trailing white space in object names written to the netlist created by the "$db write spice" command.
- Don't ignore bus range information when splitting netBuses via `$db oper addhier ⋯`.
- Fix error messages when calling Tcl parser commands with invalid options.
- Do not hide pins of instances with a symbol assigned in the Cone window.

RTLvision PRO 6.9.6

This is a maintenance release, the following features were fixed and/or added:

- Fix connectivity issue in `zOperChangeSimilar` if the new cell is missing a portBus.
- Add a new option `-similar` to the `$db oper changecellref` command.
- Validate the command line options specified in a file using the -argsFromFile option.

RTLvision PRO 6.9.5

This is a maintenance release, the following features were fixed and/or added:

- Enhance the "$db write spice" command and add the option -nomodel to not add .model statements for models or empty sub-circuits for macro models in the generated netlist.
- Fix bad netlist created by the "negedge" database operator.
- Fix segmentation fault in the "negedge" database operator.
- Fix segmentation fault in the "cloneDB" database command.
- Fix the "cloneDB" database command to also copy netbus range information.

RTLvision PRO 6.9.4

This is a maintenance release, the following features were fixed and/or added:

- Fix displaying a Source file not entered into the Spos database (e.g. by clicking on an error message in the Console window).
- Fix history navigation in the Schem window for schematics with multiple pages.
- New database operator "negedge" to remove all inverter directly connected to a clock, set or reset pin of a Flip-Flop and add a bubble to the pin.
- The create compact RTL schematic option removes all inverter directly connected to a clock, set or reset pin of a Flip-Flop and add a bubble to the pin.
- Allow for changing the file type and library of multiple files at once in the Open dialog.

RTLvision PRO 6.9.3

This is a maintenance release, the following features were fixed and/or added:

- Avoid Tcl error while saving the schematic of the design hierarchy as bitmap images if there are unused modules in the database.

RTLvision PRO 6.9.2

This is a maintenance release, the following features were fixed and/or added:

- Enhance the Cone extraction API and add the option to specify a custom callback to process the result list.
- Fix default VHDL library path discovery when running the "rtl2zdb" executable without an absolute path.
- Do not error out if a VHDL file is referring to a non-existing VHDL library.
- Add support for the Verilog pragma "map_to_module".

RTLvision PRO 6.9.1

This is a maintenance release, the following features were fixed and/or added:

- Avoid errors when scanning for plugins in unreadable directories.
- Avoid error when opening connectivity lens within unfolded instance.
- Add a configuration option for the used help browser.
- Automatically synchronize bookmarks from multiple visualizers.
- Re-add the slibconv symbol conversion utility.
- Add the option to display the waveform of a signal multiple times in the Wave window.
- Speed-up loading a design with many user defined primitives.
- Avoid syntax error in the Liberty parser with tight colons.

RTLvision PRO 6.9.0

This is a major release, the following features were fixed and/or added:

- Redesign the Search window which can now be opened from the toolbar and is shown in a bottom tab.
- Improve the performance of the Search window.
- New Blocklevel View that provides an abstract view of the original design.
- Add zoom-in and zoom-out buttons to the toolbar of the Schem and Cone window.
- Add support for setting additional cursor in the Wave window.
- Add new command line option -userwareArgs to pass a list of options to the specified Userware script.
- Add new command line option -L to specify the search order for Verilog libraries.

In addition, the following features were fixed and/or added:

- Adapt output of EDIF export optimized for OrCAD to support OrCAD Capture versions 15.7 and greater.
- Speed-up the API function busOf to get the netBus object for a given member net.
- Remove deprecated commands `zdb license`, `zdb message`, `zdb msg`, `zdb os`, and `zdb progress`. Use `zlicense`, `zmsg`, `zos`, and `zprogress` instead.
- Add "Extract to Driver/Load" button to the Cone window's toolbar.
- Enhance the "$db flag -db" API command and add the option to process only the given object type.

RTLvision PRO 6.8.12

This is a maintenance release, the following features were fixed and/or added:

- Enhance the GUI API command Gui:AddMenuSeparator and add the option to specify a separator ID.
- Extend the GUI API and add the command Gui:RemoveMenuSeparator to remove a menu separator.
- Extend the GUI API and add the command Gui:AddMainMenuItem to add a new main menu item.
- Fix using Liberty cells with non Verilog conform bundle member names.

RTLvision PRO 6.8.11

This is a maintenance release, the following features were fixed and/or added:

- Fix restore bookmarks in the Waveform viewer containing user defined groups.

RTLvision PRO 6.8.10

This is a maintenance release, the following features were fixed and/or added:

- Fix crash in "$db clone" and "$db merge" database API commands.
- Do not show error messages while typing into the time fields of the Waveform viewer window.

RTLvision PRO 6.8.9

This is a maintenance release, the following features were fixed and/or added:

- Improve performance of the Search window.
- Improve performance of the Tree window.
- Fix computation of short file names for display in the Source window.
- Improve performance of "Save Schematic as Image" function the Windows.

RTLvision PRO 6.8.8

This is a maintenance release, the following features were fixed and/or added:

- Fix Schem, Cone and Source tab navigation using the keyboard binding.

RTLvision PRO 6.8.7

This is a maintenance release, the following features were fixed and/or added:

- Enhance the "$db cone" API command and add the options "-flat" to exclude hierarchical pins in the result.
- Fix pointer adjustment error when opening a binfile containing a symlib.

RTLvision PRO 6.8.6

This is a maintenance release, the following features were fixed and/or added:

- Fix the Gui:RemovePopupEntry API command.
- Fix the 'Cone > Load Cone to I/O' entry in the Popup menu (cone was appended instead of loaded).
- Add the option to filter the list of Plugins displayed in the Plugin dialog and show only active or inactive Plugins.
- Fix incremental navigation in the Cone window when tracing single bit nets through buses at hierarchy boundaries.
- Extend the GUI API and add Gui:AddWaveAlias and Gui:WaveScopeUpdate to define an alias for an OID in a different scope.

RTLvision PRO 6.8.5

This is a maintenance release, the following features were fixed and/or added:

- Fix progress bar updates when opening big binfiles.
- Fix editing display attributes via the Select Attributes dialog.
- Add conditional expressions to meta attributes.
- Add support for port expressions to the Verilog netlist reader.
- Fix displaying value changes at time 0 in the Waveform viewer.
- The "create hierarchy" function can be called with an empty string to guess the hierarchy separator.

RTLvision PRO 6.8.4

This is a maintenance release, the following features were fixed and/or added:

- Add regexp/glob/exact options to the Find in File feature of the Source window.
- Fix Copy & Paste problem on X11.

RTLvision PRO 6.8.3

This is a maintenance release, the following features were fixed and/or added:

- Enhance the initial file name of the save file dialog.
- Fix displaying a highlighted time range in the Wave window.
- Speed-up the database operator $db oper deletePort ....
- Add the command line option -iconify to start the main window iconified.
- Add missing symutils directory to the release package.

- Fix creating the menu entry to access the symbol library export provided by the cadence2symlib.il script.

RTLvision PRO 6.8.2

This is a maintenance release, the following features were fixed and/or added:

- Display objects loaded by the Clock Domain Analyzer dialog in fullfit mode.
- Fix creation of SVA_POSEDGE primitives.
- Add the option to close the currently loaded design database to the File menu.

RTLvision PRO 6.8.1

This is a maintenance release, the following features were fixed and/or added:

- The Flat View API command flatattr delete now supports removing only one named attribute.
- Fix loading Recent Userware scripts from the File menu.
- Fix log file creation at start-up.

RTLvision PRO 6.8.0

This is a major release, the following features were fixed and/or added:

- New Plugin dialog for easy access to Userware scripts to customize and extend the functionality of RTLvision PRO.
- RTLvision PRO uses a global selection independent from the current Visualizer.
- Extend the GUI API and add Gui:AddSubMenu, Gui:AddMenuCommand, Gui:AddMenuCheckbutton, Gui:AddMenuRadiobutton, Gui:AddMenuSeparator, Gui:RemoveMenuEntry, and Gui:CustomizeMenuEntry to extend and customize the RTLvision PRO menu.
- Extend the GUI API and add Gui:RemovePopupEntry and Gui:RemoveCustomizePopup to remove custom Popup menu entries.
- Update the lef2zdb parser to support LEF version 5.8.
- Update the def2zdb parser to support DEF version 5.8.
- New Liberty parser option -setPrimFunc to set a primitive function at the created cell.
- Incompatible GUI API change: remove the "visu" parameter form the Gui:RegisterSelectionCallback and Gui:RemoveRegisteredSelectionCallback functions.
- Extend the GUI API and add Gui:IsLoadedAsAPlugin, Gui:AddPluginConfig and Gui:GetPluginConfigValue to develop custom plugins.
- Add support for 64-bit Windows.

In addition, the following features were fixed and/or added:

- Enhance the GUI API command Gui:AddMainMenuButton and add the option to specify the position of the menu label in the main menu.
- The Liberty parser recognizes more XOR and MUX functions from given Boolean equation.
- Fix the Find in File feature of the Source window if searching across multiple files.

RTLvision PRO 6.7.9

This is a maintenance release, the following features were fixed and/or added:

- Fix reading Liberty files with Windows line endings.
- Add new Popup menu command "Goto and Zoom" to goto the selected object in all views and zoom to the selected object in the Schem and Cone window.

RTLvision PRO 6.7.8

This is a maintenance release, the following features were fixed and/or added:

- Fix crash in the Verilog netlist parser for escaped module names.
- Fix buffer overflow in the Verilog netlist parser for large concatenations.
- The Verilog netlist parser no longer adds an extra space to an escaped instance name with bit subscript characters.
- Show all signals in the signal selection of the Wave window.

RTLvision PRO 6.7.7

This is a maintenance release, the following features were fixed and/or added:

- Fix value change navigation in the Wave window after removing a signal.
- Fix jump to previous value change in the Wave window if there is only one value change.
- Add an option to skip control logic when displaying a clock domain.
- Highlight objects added to the Cone window when performing the more operation on netbusses in signal mode.

RTLvision PRO 6.7.6

This is a maintenance release, the following features were fixed and/or added:

- Exporting Verilog from the Cone window preserves net values.
- Add the command line option "-storePlacement" to the DEF parser to control the storage of placement information as attributes (default is off).
- Enable the command line option "-top" also for the DEF parser.
- Fix error when opening the "Select Attributes dialog" multiple times.

- Fix removing portbuses with zOperDeletePort / $db oper deletePort.
- Strengthen zOperRemoveMOS / $db oper removeMOS to remove transistors with all pins shortened; improve documentation.

RTLvision PRO 6.7.5

This is a maintenance release, the following features were fixed and/or added:

- Fix creation of unique names in the post process operator API commands.

RTLvision PRO 6.7.4

This is a maintenance release, the following features were fixed and/or added:

- Keep Cone history even when the cone is cleared or new objects are loaded into the cone.
- Add bus member navigation for non Verilog conform buses to the Action Bar in the Source window.
- The Verilog netlist parser now creates the connectivity for escaped pin names with bit subscript characters.

RTLvision PRO 6.7.3

This is a maintenance release, the following features were fixed and/or added:

- Add the option to disable net hiding in the Schem window using a double-click.
- Add the option to always show the Find in File toolbar in the Source window.
- The visibility of the Action Bar in the Source window is updated after changing the corresponding Preferences value or calling Gui:PersistChanged.
- Increase the precision for values coming from the Liberty parser.

RTLvision PRO 6.7.2

This is a maintenance release, the following features were fixed and/or added:

- Fix licensing issue using the C-API.

RTLvision PRO 6.7.1

This is a maintenance release, the following features were fixed and/or added:

- Extend the GUI API and add Gui:RegisterHighlightChanged to register a customer specific callback that is called every time the highlight in the GUI changes.
- Enhance the database operator "$db oper mergeRams" and add all RAM net attributes to the created RAM instance.

- Enhance RAM detection of the RTL parser.
- The RTL parser now returns with an error if no input file is given.
- The RTL parser now returns with an error if no netlist was created.
- The Verilog netlist parser now returns with an error if no input file is given.

RTLvision PRO 6.7.0

This is a major release, the following features were fixed and/or added:

- Extend the GUI API and add Gui:WaveSetCursor to set named cursors in the Wave window.
- Display bus member attributes in the Infobox.

In addition, the following features were fixed and/or added:

- Only show signals with a value change in the signal selection of the Wave window.
- Only show the hierarchy tree for a scope that contain signals with a value change in the Wave window.
- Details of each result path displayed in the Cone Extract dialog can be shown.
- Display the number of blackbox modules in the Report Instance Count dialog.
- The Schem window maintains a global bookmark list instead of individual lists for each module.
- Bookmarks in the Cone window can be recalled after clearing the Cone view.
- Fix Drag & Drop and bookmarks in the Cone window when working with a multi-top design.
- Fix calling bookmarks that belong to a different top module in the Schem window.
- New database operator "$db oper mergeRams" to merge ReadPort and WritePort instances to one RAM instance.
- Fix API command "$db oid print" for pin OIDs.

RTLvision PRO 6.6.7

This is a maintenance release, the following features were fixed and/or added:

- Update values displayed in the Schem and Cone window if the time marker in the Wave window is removed.
- Provide visual feedback to the restored bookmark in the Source window.
- Avoid error in Schem history navigation.

RTLvision PRO 6.6.6

This is a maintenance release, the following features were fixed and/or added:

- Fix saving the schematic layout in a ZDB binfile.
- Fix the GUI API function Gui:WaveSetLabel if the time is in human readable format.

- Add keyboard shortcut Ctrl-f in the Source window to find text in a file.

RTLvision PRO 6.6.5

This is a maintenance release, the following features were fixed and/or added:

- Speed-up the Cone extraction Tcl API command.
- Extend the GUI API and add Gui:RegisterAddCone to register a callback procedure that is called after objects have been added to the Cone window.
- Extend the GUI API and add Gui:RemoveRegisteredAddCone to remove a previously register add Cone callback procedure.

RTLvision PRO 6.6.4

This is a maintenance release, the following features were fixed and/or added:

- Enhance the Command Kit API and add print_oid_as_slash_path to print an OID as a slash separated name.
- Enhance the Command Kit API and add print_oid_as_spef to print an OID as a SPEF name.
- The Command Kit API function get_top_design no longer throws an error if no top module could be found.
- Fix the Command Kit API function get_top_design guessing the design top if multiple tops are present.
- Add the option to select the Verilog compile mode to the Open dialog.
- Enable the command line option -compileMode for RTLvision PRO.
- Fix the use of permanent highlight colors in the Schem and Cone window.

RTLvision PRO 6.6.3

This is a maintenance release, the following features were fixed and/or added:

- The contents of RTL Operators is no longer elaborated by default.
- Speed-up reading Verilog files with a huge number of defparams.
- Extend the GUI API and add Gui:GetCustomWidgetPath to get the widget path to a custom widget.
- Add missing header file "include/zdb/zprimitive.h".
- Add missing object to the zdb link library.
- Enhance the database API command oid print.
- Enhance the Command Kit API function get_top_design to support multiple design tops.
- Speed-up the guess inst array post-process operator.
- Keep objects selected while using the history buttons in the Schem and Cone window.

RTLvision PRO 6.6.2

This is a maintenance release, the following features were fixed and/or added:

- Fix changing the name of the logfile in the Preferences dialog.
- Fix the create hierarchy feature of the Verilog netlist reader.
- Enhance the guess bus option and support escaped Verilog identifier.
- Liberty files are added to the spos database.
- The Liberty parser adds the attributes LIBERTY_FILE and LIBERTY_LINE to cell, port and portBus objects.
- The Verilog RTL parser does a more relaxed language checking for Verilog files with undefined macros.
- The Verilog RTL parser no longer ignores the entire module it there is a statement with a syntax error.
- Fix Source view navigation towards the end of the file.

RTLvision PRO 6.6.1

This is a maintenance release, the following features were fixed and/or added:

- Add the netlist option "Remove Buffer" to the Open dialog.
- Add the netlist option "Reduce Chain of Inverter" to the Open dialog.
- Add the new netlist related command line option -removeBuffer to remove all buffer instances.
- Add the new netlist related command line option -reduceInvChain to reduce a chain of inverter instances.
- Add the option to zoom to the selected object(s) in the Schem and Cone window.
- Add the option to propagate a port attribute to the pin.

RTLvision PRO 6.6.0

This is a major release, the following features were fixed and/or added:

- The Signal Mode is now a global RTLvision PRO setting accessible from the Toolbar.
- Add the option to control the elaboration of operators.

In addition, the following features were fixed and/or added:

- In the Preferences dialog trace through cells can be entered using a glob style pattern.

RTLvision PRO 6.5.5

This is a maintenance release, the following features were fixed and/or added:

- Enhance processing of filesets.
- Add command line option '-F' to read filesets containing relative file names.
- User definable file extensions for verilog dialects via new command line options +verilog1995ext+, +verilog2001ext+, and +systemverilogext+.
- Fix default VHDL library path if the installation directory contains a blank.
- Avoid error displaying decimal numbers > 64 bit in the Wave window.
- Enhance processing of command line options.

RTLvision PRO 6.5.4

This is a maintenance release, the following features were fixed and/or added:

- Fix error message opening an invalid binfile.

RTLvision PRO 6.5.3

This is a maintenance release, the following features were fixed and/or added:

- Enhance the Verilog netlist reader and add support for named parameter assignment.
- Fix short file names in the Source window.
- Make sure that line marks are always visible in the Source window.

RTLvision PRO 6.5.2

This is a maintenance release, the following features were fixed and/or added:

- Extend the Color tab of the Preferences dialog and add the option to configure all colors used in the Source window.
- Fix select symbols for custom symbol shapes.
- Fix the display of constant values in inline expanded modules.
- Display short file names in the Source window.

RTLvision PRO 6.5.1

This is a maintenance release, the following features were fixed and/or added:

- The VCD reader treats a syntax error in the header section as a warning.
- Add support to match flat VCD names to a hierarchical design.
- Add the appropriate file extension for a schematic saved as an image file.
- Colors are preserved saving a schematic in SVG format.
- Add the option to select the orientation of the created image to the Save Schematic as Image dialog.

RTLvision PRO 6.5.0

This is a major release, the following features were fixed and/or added:

- Add support to expand the contents of modules inline in the current module.
- Extend the GUI API and add Gui:ConeCustomMoreAction to register a custom procedure for the more command (double click) in the Cone window.
- The displayed time range in the Wave window can be set using the "From" and "To" fields.

In addition, the following features were fixed and/or added:

- Optimize the wiring in the Cone window after folding a hierarchical region.
- Display range information at bus objects.
- Add a x-scrollbar to the list of displayed signal names in the Wave window.
- Fix version check.

RTLvision PRO 6.4.5

This is a maintenance release, the following features were fixed and/or added:

- Fix reading a fileset on the RTLvision PRO command line as SystemVerilog.
- Object highlights are now visible in the Connectivity Lens window.
- Extend the GUI API and add Gui:ConeIsLoaded to check if one of the given OID in oidList is loaded to the Cone window.

RTLvision PRO 6.4.4

This is a maintenance release, the following features were fixed and/or added:

- The Liberty reader ignores internal pins.
- Enhance and update documentation.
- Add command line option "-defParam" to overwrite VHDL generics.

RTLvision PRO 6.4.3

This is a maintenance release, the following features were fixed and/or added:

- Optimize generation of the Source window’s hierarchical context.
- Fix links to documentation files and links within the documentation.
- Enhance the GUI API command Gui:DoubleClick and add support to remove the custom binding.
- Extend the GUI API and add Gui:ConeCustomFoldAction to register a custom procedure for the fold button at hierarchical instances in the Cone window.
- Extend the GUI API and add Gui:ConeCustomUnfoldAction to register a custom procedure for

the unfold button at hierarchical instances in the Cone window.

RTLvision PRO 6.4.2

This is a maintenance release, the following features were fixed and/or added:

- Disable "Beautify" entry in Popup menu if all sub-items are disabled.
- "Global" bookmarks in the Source view window.
- Fix increasing memory usage in the Source view window.
- Fix the Export Netlist option to create a corresponding symbol library.
- Fix Save Schematic as Image in PNG format.

RTLvision PRO 6.4.1

This is a maintenance release, the following features were fixed and/or added:

- Fix issue displaying source preview in tooltips.

RTLvision PRO 6.4.0

This is a major release, the following features were fixed and/or added:

- Add support for reading Verilog AMS files.
- Enhance navigation in the Source view window: introduce an "Action Bar" for object specific navigation.
- Add the option to select the font for the Source view window.
- Display source preview in tooltips.
- Add hierarchical context to the Source view window.
- Text section in the Source view window is now bind to shift + left mouse button.
- New database API command $db get_driver to get the driver of a net or signal.
- New flat view API command flat foreach flagged to loop over all objects with the given flag name set.
- New flat view API command flat foreach attr to loop over all objects with the given attribute name set.
- Extend the GUI API and add Gui:ShowCustomWidget to toggle the visibility of a created custom widget without destroying it.
- A new Visualizer can only be created as a toplevel window. A second inline Visualizer can be toggled from the Window menu or toolbar.
- Extend the GUI API and add Gui:ShowSecondVisualizer to toggle the visibility of the second Visualizer.
- Extend the GUI API and add Gui:GetSecondVisualizer to get the widget path to the second

Visualizer.

- Remove the S/C tab from the Visualizer.
- The Drag & Drop mouse button can be configured from right to left.
- Add the command line option "-sym2zdb" to preload cells from a given Symbol library.
- Drop support for the Solaris x86_64 platform.

In addition, the following features were fixed and/or added:

- Fix data for Source window of RTL files without a newline character in the last line.
- Add the command line option -version to RTLvision PRO.
- The behavior and options of the GUI API function Gui:InsertCustomWidget has changed.
- The Verilog netlist parser no longer creates wrong connectivity for escaped net names with bit subscript characters matching an existing bus member.
- The Verilog netlist parser no longer accepts reserved keywords as an instance name.
- Add horizontal scrolling to the Infobox.
- A double-click on a module in the Tree window also shows the module in the Source window.
- A double-click on a module in the Tree window no longer activates the Schem tab.
- A Goto operation may change the context in the Schem window.
- A binlib file created with the sym2zdb tool no longer requires a license feature.
- Fix searching across files in the Source window.
- Optimize mouse wheel scrolling and panning in the Source view window.
- Fix "text search across files" in the Source view window.
- Enhance performance of "text search" in the Source view window.
- Fix displaying TAB characters in the Source view window.
- Fix using the "Native Browser" for viewing the documentation on Linux.

RTLvision PRO 6.3.4

This is a maintenance release, the following features were fixed and/or added:

- Draw ports and portBuses with an unknown direction in grey color.
- Improve direction of ports created by the option to recreate hierarchy.
- Change the default behavior of the "zdb open" API command to open the specified binfile in readonly mode. To open a binfile writable the option -writable can be used.
- Starting the GUI with the command line option -help no longer require a license.
- Extend the GUI API and add Gui:WaveSetTimeRange to set the visible time range of the Wave window.
- All Wave window related GUI API commands accept a human readable time string.
- Wave window enhancements.

RTLvision PRO 6.3.3

This is a maintenance release, the following features were fixed and/or added:

- Fix reading multiple DEF files.
- Attribute values storing Y positions coming from a LEF or DEF file are no longer flipped.
- Add support for connecting special nets using pattern match to the DEF parser.
- Add support for fullscreen mode (new command line option "-fullscreen", new menu entry "Full Screen" in Window menu, new key binding "F11").
- Extend the GUI API and add Gui:FullScreen to toggle fullscreen mode.

RTLvision PRO 6.3.2

This is a maintenance release, the following features were fixed and/or added:

- Wave window enhancements.
- Add the command line option -createHier to the DEF parser to control hierarchy creation.
- Bind the escape key to cancel mouse stroke events in schematic windows.

RTLvision PRO 6.3.1

This is a maintenance release, the following features were fixed and/or added:

- Enhance icon and application name displayed by the window manager.
- Enhance the "operator" API and add "$db oper collectSignalData" to move data from all nets of a signal to the signal.
- No longer clear the Cone window if only display properties have been changed in the Preferences dialog.

RTLvision PRO 6.3.0

This is a major release, the following features were fixed and/or added:

- Add new Verilog command line option -globalInclude to define a global Verilog include file.
- Extend the GUI API and add Gui:WaveSetLabel to set a text label at the given oid and time in the Wave window.
- Enhance the "$db cone" API and add the option -shortestPath to quickly return the shortest path from a given start object to one of the specified targets.

In addition, the following features were fixed and/or added:

- Fix startup script to support quoted command line options.
- Avoid error while scrolling a zipped RTL file in the Source window.

- Fix reading escaped identifiers in a VCD file.
- Fix the EDIF parser to treat empty cells as primitives.
- Fix the EDIF parser to respect the "-spos off" command line option.
- Remove hard coded limitation while loading the driver of a net in the Cone window. This limit is now derived from the Big Module Limit.

RTLvision PRO 6.2.5

This is a maintenance release, the following features were fixed and/or added:

- Extend the GUI API and add Gui:GetCurrentWindow to get the name of the window with the input focus.
- Enhance the flat foreach pin command and add support for global nets.
- The API command isPgNet now returns a Boolean value.

RTLvision PRO 6.2.4

This is a maintenance release, the following features were fixed and/or added:

- Avoid error if Drag & Drop is started with a keyboard modifier.
- Speed-up "$db write spice" for large databases.
- Graphical marks are no longer displayed at the wire if Net Attribute at Wire" is disabled.
- User defined Graphical marks no longer interfere with internally used marks.
- Enhance the Graphical marks API and add support to set an individual background color and ratio for each mark.
- Enhance the Graphical marks API and add support to combine two (or more) marks into more complex shapes.
- Avoid error while starting the zdbsh binary.
- Fix selection of objects in the Tree window.
- Fix display of bus member signals in the Wave window.
- Fix selection of objects in the signal selection of the Wave window.
- Keep line marks in the Source window visible until the next mouse click.
- Fix highlighting of a bus with enabled "Signal Mode".

RTLvision PRO 6.2.3

This is a maintenance release, the following features were fixed and/or added:

- Avoid error using keyboard shortcuts for Popup menu commands.
- Enhance the cadence2symlib.il script to create filled paths to support symbol background colors.

RTLvision PRO 6.2.2

This is a maintenance release, the following features were fixed and/or added:

- Fix "flat signalOf" and "flat foreach" at complex connections.
- Fix segmentation fault in the RTL parser for "power" operators.

RTLvision PRO 6.2.1

This is a maintenance release, the following features were fixed and/or added:

- Parser options enabled on the command line were not visible in the corresponding parser dialog.
- Fix turning off Source file references for the RTL Parser.
- Update the Schem and Cone window if the corresponding option have been changed in the Preferences dialog.

RTLvision PRO 6.2.0

This is a major release, the following features were fixed and/or added:

- Add support to search for objects in the Cone window.
- Add support for Bookmarks also to the Wave window.
- Add support for moving signal in the Wave window.
- Add support to create groups of signals in the Wave window.

In addition, the following features were fixed and/or added:

- A binfile saved from the GUI now contains the context of the Source window.
- A binfile saved from the GUI now contains the context of the Wave window.
- Fix displaying logic values annotated from the Wave window in the Source window.
- Try to match OIDs in the Wave window also case insensitive.
- Do not show the logic value if a transition icon is annotated from the Wave window.
- Fix displaying netBus objects with null members in the Wave window.
- Fix missing top scope in the tree of the Wave window.
- Fix invalid database caused by the Verilog netlist parser option -createHier.
- Add support for bus indices greater 32767 to the Verilog netlist parser.
- Fix segmentation fault in the Liberty parser for strings greater than 1500 characters.
- Fix back annotation of SDF values.
- Avoid error while navigating through multiple schematic pages using the off-page connector.
- Add the option -populate to the "zdb open" command to load a binfile into physical memory.

- Fix missing support for the command line option -geometry.
- Extend the GUI API and add Gui:AddBookmark to add a bookmark in the Wave, Schem, Cone or Source window.
- Extend the GUI API and add Gui:SelectBookmark to select a bookmark in the Wave, Schem, Cone or Source window.
- Extend the GUI API and add Gui:RenameBookmark to rename a bookmark in the Wave, Schem, Cone or Source window.
- Extend the GUI API and add Gui:DeleteBookmark to delete a bookmark in the Wave, Schem, Cone or Source window.
- Remove the deprecated API commands "$db cdc calcFFload", "$db cdc getFFpinList" and "$db cdc treeCalc".

RTLvision PRO 6.1.0

This is a major release, the following features were fixed and/or added:

- Add support saving the schematic as a PNG or SVG image.
- New licensing model:
  - The checked out master feature enables the use of all parser through Tcl API commands.
  - The option "-wait_for_license" is no longer needed for opening a binfile using the "zdb open" command.
  - The option "-wait_for_license" is no longer needed for starting a parser using a Tcl command.
- Rename the license API command "zdb license" to "zlicense".
- Rename the system call API command "zdb os" to "zos".
- Rename the message API command "zdb message" to "zmsg".
- Rename the progress API command "zdb progress" to "zprogress".
- Remove the "zfork" API command.
- Increase the number of possible highlight colors.

In addition, the following features were fixed and/or added:

- Enhance the Wave window and add the option to create vectors for variables with the same basename and a consecutive bit subscript.
- Logic values annotated from the Wave window no longer interfere with custom attributes.
- Fix displaying transitions annotated from the Wave window.
- Add new command line option -semanticCheck to the Liberty parser to perform additional semantic checks reported as warnings. The option -pedantic change the severity of the semantic check messages into error.
- Enhance the "Save Schematic as Image" dialog.

- Add the option to use the goto color for additional selection feedback.
- Enhance displaying net attributes at the wire.
- Fix saving the "Instance Count" report to a text file if the displayed number of objects contains a decimal mark.
- Enhance the "$db coneToPG" API command and add the options "-only", "-excludeFlaggedCell" and "-excludeFlaggedInst".
- Enhance beautifying the schematic and add support to select a built-in symbol shape also for cells with bus ports.
- Enhance beautifying the schematic and add support to select a built-in MUX symbol.
- The pin list of the Navigate Net/Signal dialog is sorted.

RTLvision PRO 6.0.13

This is a maintenance release, the following features were fixed and/or added:

- Fix missing progress bar for the RTLvision PRO option -wait_for_license.
- Improve visual feedback while moving instances.
- Always hide all unconnected pins at module instances in the Cone window.
- Enhance the "$db write spice" API command and add the option "-noEnd" to not add an ".END" statement at the end of the created Spice file.
- New API command "$db count" to count objects.
- Enhance the "operator" API and add "$db oper deletePort" to remove ports or portBuses from a cell.

RTLvision PRO 6.0.12

This is a maintenance release, the following features were fixed and/or added:

- Fix reading a fileset as a Verilog netlist.
- Fix displaying netBus values from the Wave at the connected pinBus.

RTLvision PRO 6.0.11

This is a maintenance release, the following features were fixed and/or added:

- Allow multiple name/value pairs in the +define+ command line options.
- The RTL parser displays a Verilog port expression as the port name.
- Fix the calculation of crossing between selected clock domains only.
- Propagate values annotated from the Wave window through the hierarchy.
- Use graphical marks for transitions annotated from the Wave window.
- Add the option to configure the attribute color for values annotated from the Wave window.

- Fix display of port and net attributes in the Source window.
- Fix mapping of permanent highlight colors in the Schem and Cone window.
- Fix the database API command "flag -db" for netBus and module objects.

RTLvision PRO 6.0.10

This is a maintenance release, the following features were fixed and/or added:

- Fix missing error message if a license checkout failed.
- Enhance the highlight and flat highlight API to support the option "-both" for getting either the normal or permanent highlight color value.
- The Source, Tree, Mem and Search window now also show permanent highlight colors.
- Avoid wrong VCD parser syntax error.

RTLvision PRO 6.0.9

This is a maintenance release, the following features were fixed and/or added:

- Load Userware code before opening a binfile.
- Fix crash in create hierarchy with an empty string as the hierarchy separator character.
- Fix adding values to the Verilog Options tab of the Open Files dialog.

RTLvision PRO 6.0.8

This is a maintenance release, the following features were fixed and/or added:

- Fix wrong syntax error for the "criticality" keyword reported by the EDIF parser.
- Fix EDIF parser crash.
- Fix adding symbol libraries in the Open file dialog.
- Fix moving the viewport in the Minimap window.
- Extend the GUI API command Gui:SaveConeAs and add the option to save the contents a ZDB binfile.
- Extend the GUI API and add Gui:ZoomTo to zoom the Schem or Cone window to the given OID list.
- Extend the "$db report" API and add the command portCount to get the number of ports.

RTLvision PRO 6.0.7

This is a maintenance release, the following features were fixed and/or added:

- Fix the "vhdl2zdb" binary and "zvdb" Tcl command for compiling multiple units into one library.

- Extend the GUI API and add Gui:RegisterClearCone to register a callback procedure that is called before the Cone window will be cleared.
- Extend the GUI API and add Gui:RemoveRegisteredClearCone to remove a previously register clear Cone callback procedure.
- The GUI API command Gui:ClearCone now accepts "all" to clear the Cone window of all existing Visualizers.
- Extent the Flat View API and add the helper functions "$db flat count ..." to get the number of loops for the "$db flat foreach ..." commands.
- The "Big Module Limit" can be set to -1 to disable the generation of any new schematic.
- Fix the missing result of the API command "$db get_ports".

RTLvision PRO 6.0.6

This is a maintenance release, the following features were fixed and/or added:

- Add new "$db oper setDirection" command to set the direction of a port.
- Extend the GUI API and add Gui:ShowClockTreeAnalyzer to toggle the visibility of the Clock Domain Analyzer dialog window.
- Extend the GUI API and add Gui:HighlightClockDomain to highlight a clock domain.
- Extend the GUI API and add Gui:UnhighlightClockDomain to unhighlight a clock domain.
- Fix the Cone Extraction API option "-ignoreDir" for target pins.
- Enhance the operator command $db oper rename and add the option "-checkName" to check if the rename would create a name clash.
- Fix support for the FlexNet TIMEOUT option to automatically return an inactive license feature.
- Fix crash in "$db oid print" if the result is an empty string.

RTLvision PRO 6.0.5

This is a maintenance release, the following features were fixed and/or added:

- The environment variable CE_TCL_INIT_SCRIPT can point to a Tcl script that is sourced at startup.
- The symbol utilities package 'Symutils' is now part of the RTLvision PRO release.
- The ZDB API function "$db parentModule" now throws an error if called with a primitive port.
- Fix the ZDB API function "$db oid convertTo" for converting a module or primitive OID into a portBus OID.

RTLvision PRO 6.0.4

This is a maintenance release, the following features were fixed and/or added:

- Speed-up subsequent calls to "$db load module".

RTLvision PRO 6.0.3

This is a maintenance release, the following features were fixed and/or added:

- Extend the GUI API and add Gui:RegisterSelectionCallback to register a custom selection callback.
- Extend the GUI API and add Gui:RemoveRegisteredSelectionCallback to remove a registered selection callback.
- Also implicit license checkouts (e.g. like zdb open) will respect the value of the environment variable + `CONCEPT_WAIT_FOR_LICENSE`.
- Fix missing error message if a binfile specified on the command line of RTLvision PRO could not be opened.
- The zdb info $binfile command shows information about the parsers used to create the given binfile.
- Add new zdb parserbits $binfile command to get a list of all parsers used to create the given binfile.

RTLvision PRO 6.0.2

This is a maintenance release, the following features were fixed and/or added:

- The "Save Settings" and "Restore Settings" buttons of the Open dialog were mixed up.
- Fix consecutive license checkouts from the API after a failed checkout.
- Fix the RTLvision PRO option -wait_for_license if specified with a value of 0.
- If the environment variable CONCEPT_WAIT_FOR_LICENSE is set then all license checkouts will respect the value of this variable.
- Enable the option -wait_for_license for the zdb open API command.
- The FlexNet licensing software is part of the RTLvision PRO download package.

RTLvision PRO 6.0.1

This is a maintenance release, the following features were fixed and/or added:

- Fix crash in Liberty parser with bus_naming_style.
- The Search window uses a determinate progress bar.
- Fix $db flag $oid clear called without a flag name to clear all flags at the given object.
- Minor GUI API change: the Gui:Goto command no longer accepts the value 'all' for the option window name. Omitting the window option will affect all Visualizer components.

RTLvision PRO 6.0.0

This is a major release, the following features were fixed and/or added:

- New unified file Open dialog to read in files of any type.
- Dynamically elide long object names in the Schem and Cone window.
- Add history buttons to the Schem window.
- New keyboard shortcuts Control-PgUp and Control-PgDn to switch between the Schem, Cone,S/C and Source tab of a Visualizer.
- Hierarchical instances in the Cone window with contents now display a fold/unfold button in the top left corner.

In addition, the following features were fixed and/or added:

- The Connectivity Lens now displays all connections independent from the pin direction.
- Add support for scrolling with the middle mouse button in the Wave window.
- Enhance support for bus members in the Wave window.
- Add support to drop module based OIDs (e.g. from the Source window) to the Wave window.
- Modules created by the "Add Hierarchy for Processes" (-procHier) option compiled into a library not named "work" are no longer flagged as library cells.
- Add new command line option -createUniqConsts to the Verilog parser to create unique nets for each constant pin connection.
- Add new command line option -createUniqConsts to the RTL parser to create unique nets for each constant pin connection.
- Fix reading Edif or RTL files if the environment variable FLEXLM_DIAGNOSTICS is set.
- Enhance the database API and add "$db reloadParasitic" to reload a parasitic module.
- Enhance the database API and add "$db foreach couplingPort" to loop over all coupling connections and "$db foreach couplingInst" to loop over all coupling instances of a parasitic module.
- Fix the database API command "flag -db" for pins.
- Add support for sourcing gzipped Tcl files using the "zdb source" command.
- Add the option to the Preferences dialog to toggle the drop shadow effect on instances in the Schem and Cone window.
- Selected instance and port objects in the Cone window now display a toolbar to access the delete function.
- Extend the GUI API and add Gui:AddConeBookmark to add a bookmark in the Cone window.
- Extend the GUI API and add Gui:ShowConeBookmark to show a previously added bookmark in the Cone window.
- Extend the GUI API and add Gui:DeleteConeBookmark to delete a bookmark in the Cone window.
- Extend the GUI API and add Gui:SaveConeBookmark to save a bookmark file in the Cone window.
- Extend the GUI API and add Gui:OpenConeBookmark to restore a bookmark file in the Cone window.

- Extend the GUI API and add Gui:ShowConsole to toggle the visibility of the Console window.
- The number of items below a design info node in the Tree window can be configured.
- Enhance the $db clone API function and add support to skip flagged instances in the module to be cloned.
- Fix "too many colors" error while saving a Save Schematic as Image on Windows.
- Fix displaying net and netBus attributes at the wire.
- The ndl2zdb executable is no longer part of the package. Now the source code to compile ndl2zdb is shipped in the demo/api/cust23 directory.
- Fix the rename operator for parasitic modules.
- Rename the Cone Extraction API option "-diveEmpty" to "-emptyModAsPrim".
- Show direction for pins in the Infobox.
- New cloneDB command to clone a database.
- The cloneDB command also supports cloning into an existing database.
- All *2zdb batch parser now support the command line option -binlib multiple times.
- Long lines are wrapped in the Console window.
- Avoid error while typing in the Console window.
- Add errorInfo to error messages displayed in the Console window.
- Add support for Flexid-9 (Dongle) license server hostids.
- Drop support for RHEL 4 and older.
- Upgrade FlexNet license mechanism to flexnet-11.12.
- Changed cadence2symlib.il to support portBus and add inline description.
- Add new symlib keyword "symmap" to support better module symbol mapping.
- Add support for net targets to the -reachable option of the cone extraction API.
- Added -delzombies the operator $db oper rmhier.
- Extend the database command $db write verilog and add the option "-ignoreautogen" to skip all auto generated cells.
- Changed the options of the "$db write" command for "tcl", "spice", "verilog", "dspf" and "spef".
- Add option "-into" to tcl parser commands.

RTLvision PRO 5.10.18

This is a maintenance release, the following features were fixed and/or added:

- The port names displayed in the Schem and Cone window can be overwritten using the "@name" attribute.
- Fix Load Module to Cone for top modules with "do not display hierarchy boxes" enabled.
- Speed-up the "Find in File" function of the Source window.

- Enhance showing the result of the "Find in File" function of the Source window and highlight the matched text.
- Changed cone search to follow user defined arcs independent from pin direction.

RTLvision PRO 5.10.17

This is a maintenance release, the following features were fixed and/or added:

- The "ESC" key can be used to close the Connectivity Lens in the Schem and Cone window.
- Fix redraw issue after scrolling in the Schem and Cone window with an open Connectivity Lens.
- Enhance the GUI API command Gui:NewVisualizer and add the options -width and -height to specify the width and height of the new Visualizer.
- Avoid error while displaying the tooltip for a module based pinBus object.
- Do not show the bus width twice at the textual representation of a bus object.

RTLvision PRO 5.10.16

This is a maintenance release, the following features were fixed and/or added:

- Avoid syntax error reported by the Verilog netlist parser (verilog2zdb) while reading Verilog 2001 attributes.

RTLvision PRO 5.10.15

This is a maintenance release, the following features were fixed and/or added:

- The GUI API procedure Gui:DataBaseModified keep objects selected in the Schem and Cone window.
- Extend the GUI API and add Gui:RegisterDesignReady to register callbacks to be executed after all design files have been read.
- Extend the GUI API and add Gui:GetAllVisualizers to get a list of all active Visualizers.
- Extend the GUI API and add Gui:RemoveRegisteredDesignReady to remove a previously registered callback.

RTLvision PRO 5.10.14

This is a maintenance release, the following features were fixed and/or added:

- Extend the GUI API and add Gui:GotoSourceLine to goto the given file and line in the Source window.
- Fix the option to preserve assign statements in combination with the schematic compaction.

RTLvision PRO 5.10.13

This is a maintenance release, the following features were fixed and/or added:

- Fix foreach loop over all opened databases to allow closing the current database.
- Add new command line option -compileMode to the rtl2zdb parser to force compilation to either single file unit or multi file unit mode.

RTLvision PRO 5.10.12

This is a maintenance release, the following features were fixed and/or added:

- Fix the OID API command "$db oid convertTo pinBus".
- Enhance the database API command "flag -db set" to set a named flag at all database objects.
- Enhance the database API setPrimitive to set modules flagged with the new leafcell flag as primitives.
- Enhance the database API and add the command info binfile to get the name of the binfile associated with the database.
- Fix the error handling for the "flatflag $oid is" API command.
- Set the value of tcl_interactive to true for Userware scripts sourced from the Console window.
- Do not overwrite the errorInfo value in the Console window.

RTLvision PRO 5.10.11

This is a maintenance release, the following features were fixed and/or added:

- Do not flag Verilog transfer gate primitives as transistor devices.

RTLvision PRO 5.10.10

This is a maintenance release, the following features were fixed and/or added:

- Enhance the "$db write -verilog" API command to change priority for needed name changes.
- Count hidden NC pins in an extra list in Cone Export dialog.
- Enhance the Cone Extraction API and add the option "-ignoreDir" to ignore the module port direction while searching the cone result.
- Enhance the database API to examine the connectivity and add the "foreach pinCon" and "foreach portCon" to get only pins or ports connected to the given net object.
- Display the direction of pin and port objects in the tooltip label.
- Enhance constant value support at signals. Now the signal defines a value if one interconnected net has a constant value.
- Enhance the database API command flag to clear a named flag at all database objects.

- Extend the database API and add the command oid concat to convert a hierarchical instance OID to a tree base pin oid by appending a module based pin OID or a relative path to a pin.
- Extend the database API and add the command oid exists to check if an OID exists in the database.
- Extend the Operator API and add the command oper connect to connect a pin or port to a net.
- The default instance name displayed in the Schem and Cone window can be overwritten using the "@name" attribute.
- Fix reading files from a fileset as SystemVerilog.

RTLvision PRO 5.10.9

This is a maintenance release, the following features were fixed and/or added:

- Add support for dropping signal OIDs into the Wave window.
- Add support for dragging objects out of the Connectivity Lens.
- Fix the Popup menu on objects inside the Connectivity Lens window.

RTLvision PRO 5.10.8

This is a maintenance release, the following features were fixed and/or added:

- Add new Connectivity Lens to the Schem and Cone window to show detailed connectivity information for pinBus and portBus objects.
- Fix error handling in evaluated Userware scripts.
- Fix page splitting in the Cone window.
- Fix displaying net and netBus attributes at the wire.
- Fix toggling the display of port and portBus names.

RTLvision PRO 5.10.7

This is a maintenance release, the following features were fixed and/or added:

- New Tcl API service function "zdb formatvalue" to format values from or to Spice notation, for Verilog notation or to convert from binary notation into decimal, octal or hexadecimal.
- New Tcl spos API function to check if a file exists in the spos database.
- New database API service function getFuncPort to get a port by a given function.
- Add support for constant values at signals.
- Enhance the $db clone API function and add support to skip flagged portBuses.
- The $db clone API function now clones all portBus range information.
- Avoid error on Windows while merge new files with the currently loaded database.
- Enhance "Save Cone as Verilog" and add the option to create unconnected module ports.

- Enhance "Save Cone as Verilog" and add the option to create ports for all not connected pins.
- Enhance "Save Cone as Verilog" and add the option to create all top level I/O ports.
- Do not create empty subckts for devices in the output created by the "$db write -spice" command.

RTLvision PRO 5.10.6

This is a maintenance release, the following features were fixed and/or added:

- New database API service function isPgNet to test if a net or signal is flagged as power or ground.

RTLvision PRO 5.10.5

This is a maintenance release, the following features were fixed and/or added:

RTLvision PRO 5.10.4

This is a maintenance release, the following features were fixed and/or added:

- Extend the GUI API and add Gui:GetDesignTitle to get the design title displayed in the title bar.

RTLvision PRO 5.10.3

This is a maintenance release, the following features were fixed and/or added:

- The command line option -netlistPattern can be specified multiple times.
- Enhance the GUI API command Gui:NewVisualizer and add the option -topmodule to specify the name of the visible top module.
- Extend the GUI API and add Gui:SetTopModule to set the visible top module.
- Enhance the "$db write -verilog" API command and add the option to write object comments.
- Avoid an error while opening a binfile containing a source file that no longer exists.
- Fix the option to turn off the visibility of hierarchy boxes in the Cone window.

RTLvision PRO 5.10.2

This is a maintenance release, the following features were fixed and/or added:

- The command line option -f can be specified multiple times to the rtl2zdb and verilog2zdb parser.

RTLvision PRO 5.10.1

This is a maintenance release, the following features were fixed and/or added:

- Speed-up schematic generation for complex circuits.
- Fix the Popup menu action "Load/Append Cone" invoked from the Source window on module or primitive objects.
- Fix Segfault in RTL parser "rtl2zdb" option -compact, if there are mux chains with only partial connected bus pins.
- Fix reading EDIF files given on the command line.

RTLvision PRO 5.10.0

This is a major release, the following features were fixed and/or added:

RTLvision PRO 5.9.9

This is a maintenance release, the following features were fixed and/or added:

- Extend the GUI API and add Gui:ExportPhoto to save an image of the schematic.
- Extend the GUI API and add Gui:ExportPdf to save a PDF file of the schematic.
- Enhance the GUI API command Gui:GetDataBase and change the visualizer argument to be optional.
- Fix support for TCLLIBPATH environment variable in zdbsh.
- Add parasitic specific Userware examples.

RTLvision PRO 5.9.8

This is a maintenance release, the following features were fixed and/or added:

- New database API function valid to check for valid database command.
- New database API function foreach to loop over all opened databases.
- New options in API command $db oid print to get better control over the returned string.
- Extend the database API command flag by a toggle variant.
- Fixed -incdir and -define options if used multiple times.
- Fix reading EDIF files through the "Read EDIF" dialog.

RTLvision PRO 5.9.7

This is a maintenance release, the following features were fixed and/or added:

- zdbsh now supports TCLLIBPATH environment variable to search tcl packages.
- Fix verilog2zdb to support bigger constants.

RTLvision PRO 5.9.6

This is a maintenance release, the following features were fixed and/or added:

- Do not flag cells restored from a VDB (VHDL file compiled with vhdl2vdb) as a library cell if source code reference generation was enabled.
- Change non-fatal SDF parser error into a warning.
- Fix startup of the zdbsh.exe (part of the ZDB API package) on Windows.
- Add the "tcltest" package to the GUI and zdbsh.
- The zdbsh binary reads a file name .zdbshrc on Unix and zdbshrc.tcl on Windows located in the home directory at startup if the shell is started in interactive mode.
- New Tcl API command "zutil signal" to install a custom signal handler.

RTLvision PRO 5.9.5

This is a maintenance release, the following features were fixed and/or added:

- Add the new command line option -maxErrCnt to the RTL parser to control the number of errors before the parser stops with "too many errors".
- The VHDL RTL parser now does a more relaxed language checking and report a warning for the non fatal error "... is not declared". This can be disabled by the command line option -pedantic set to on.
- Enable the command line option -spos for the vhdl2vdb binary.
- Fix the "Export Verilog Netlist" feature and therewith the "$db write -verilog" API command: no longer create wrong connectivity for netBus members connected to a scalar port with bus syntax.
- Issue a warning for command line options that are allowed only once.
- Fix schematic compaction for guessed instance arrays with serial interconnection.

RTLvision PRO 5.9.4

This is a maintenance release, the following features were fixed and/or added:

- Enhance the Infobox and show all pins and connected nets of a selected instance.
- Fix "full mode" of the schematic compaction for instances with either unconnected pins or pins connected to a constant value.

RTLvision PRO 5.9.3

This is a maintenance release, the following features were fixed and/or added:

- The "$db write -verilog" API command no longer supports writing only flagged instances and cells. Instead ports, instances and cells can be flagged to be ignored.

- Enhance the command $db clone and add the option "-skipflaggedport flag" to not clone ports flagged with the given flag.
- Fix wrong source code references for RTL Verilog files specified with the -v option or found in a directory specified with the -y option.
- Add -skipLeadX option to vcdcompile, to ignore vcd X-values at the beginning.

RTLvision PRO 5.9.2

This is a maintenance release, the following features were fixed and/or added:

- Fix recognition of arrayed instance names in schematic compaction for instances generated by the RTL parser.
- Add support for "bus" type to the Liberty parser liberty2zdb.
- Fix support for "bundle" type in the Liberty parser liberty2zdb.
- Add the new command line option -createBus to create Verilog conform buses from single bit ports with consecutive numbering.
- Fix the "Export Verilog Netlist" feature and therewith the "$db write -verilog" API command: no longer create named connectivity in module interfaces.

RTLvision PRO 5.9.1

This is a maintenance release, the following features were fixed and/or added:

- New database API function isEmpty to check if a module is empty.
- Extend the GUI API and add Gui:FileDialog to show an "Open File", "Save File" or "Choose Directory" dialog.
- Use bus syntax for generated port names in schematic compaction.
- Unneeded bitsubscripts removed from generated port names in schematic compaction.
- Added recognition of arrayed instance names in schematic compaction.
- Changed port sort sequence in -guessBus option.
- Fix the "Trace back to X" feature of the Wave window.
- Add the option to overwrite Verilog parameters using the "-define" option to the Verilog RTL parser.

RTLvision PRO 5.9.0

This is a major release, the following features were fixed and/or added:

- Revise the Cone Extract dialog.
- Enhance the Clock Domain Analyzer. Calculate all clock domains before showing the dialog window.
- Simplify the Clock Domain Analyzer dialog.

- Add new keyboard shortcuts to the Wave window to jump to the previous ("p") or next ("n") value change as well as for zoom fit ("f"), zoom in ("i") and zoom out ("o").
- The Verilog RTL parser now does a more relaxed language checking and report warnings for many non fatal errors. This can be disabled by the new command line option -pedantic set to on.
- Support gzipped VCD files.
- Rename the color schemes to "Light", "Dark" and "Custom". The default color scheme is now "Light".

In addition, the following features were fixed and/or added:

- Remove the option to "Fold Library Cells" from the Clock Domain Analyzer window and Cone Extract dialog. This feature can be controlled using the Primitive Level settings in the Preferences dialog.
- Remove the option to "Don't dive Known Function" from the Cone Extract dialog. This feature already was always on to speed-up the cone extraction.
- Fix the Cone Extraction algorithm for constant value targets.
- Add new Clock Domain Analyzer API.
- Fix setting the time marker in the Wave window.
- The "create compact RTL schematic" option no longer groups clocked elements connected to different clock nets.
- The VCD reader now supports negative indices and escaped Verilog names.
- Add the option to configure Clocked Cells to the Tools menu.
- The RTL parser "rtl2zdb" reads unknown files as Verilog-2001.
- Enhance the Liberty reader "liberty2zdb" and add the option -skipSupplyPorts to not create supply ports (pg_pin group).
- Extend the Popup menu of the Wave window. Add options to jump to the previous or next value change.
- Add zoom operations to the Popup menu of the Wave window.
- Fix the cone extraction algorithm tracing through modules with input ports directly connected to output ports.
- Fix crash in Schem and Cone window for cases with more than 1000 parallel capacitors.
- Avoid error in Cone window if the option to not show hierarchy boxes is on and a loaded net has more than one representative.
- Enable the option -wait_for_license to wait the given amount of seconds for the next free license for all batch tools.
- The time to wait for a license can be defined using the environment variable CONCEPT_WAIT_FOR_LICENSE.
- New API function isOperator to check if an instance refers to an operator.
- Speedup the Source window if the option "Show Attribute" is disabled.

- Automatically disable the option "Show Attribute" for large netlist files.
- Avoid color flickering in the Source window: no longer change the background color while the mouse moves over a known object. Only change the background color if the object is selected by a mouse click.
- Extend the GUI API and add Gui:NewParasiticWindow and Gui:HideParasiticWindow to show/hide a new toplevel Parasitic window.
- Add an optional argument to all Parasitic window related GUI API procedures to work on one of the created toplevel windows.
- Change the goto color of the "Dark" color scheme to a different color than the first highlight color.
- Add support for signal OIDs to the flathilight API.
- Show signal highlights in the Memory window.
- Avoid error in the Cone window if double-clicked on an ambiguous connected portBus.
- Changed API for User Defined Arcs to block ports and allow inverted arcs.

RTLvision PRO 5.8.3

This is a maintenance release, the following features were fixed and/or added:

- Add support for custom attributes to the Liberty parser liberty2zdb.
- Fix wrong mouse wheel scroll behavior in the Wave window.
- Avoid the error "font "" doesn't exist" in the Wave window.
- New Operator API function chain -inv to replace chains of INV primitives by either one (odd number of INVs in the chain) INV or no INV (even number of INVs in the chain).
- New database API function hiersep to set the hierarchy separator.
- Fix history buttons in the Cone window.
- Restore bookmarks in the Cone window from a binary database.
- Clear list of bookmarks in the Schem and Cone window after the database has changed.

RTLvision PRO 5.8.2

This is a maintenance release, the following features were fixed and/or added:

- Add support for "bundle" type to the Liberty parser liberty2zdb.
- Sort the signals loaded to the Wave window using the "Get Signals from Cone" button.
- No longer show the "Bus Dialog" if an autobundle would be loaded in the Cone window.
- Add the option to the "Bus Dialog" of the Cone window to load multiple or all bus members.
- Avoid error if a partly loaded autobundle is selected in the Cone window.
- Do not show the Console window automatically in case of an user interrupt message.

RTLvision PRO 5.8.1

This is a maintenance release, the following features were fixed and/or added:

- Do not show net names at the connected pin if the name if automatically generated by a parser.
- The "lib2zdb" tool is no longer released with RTLvision PRO. Please use "liberty2zdb" instead.

RTLvision PRO 5.8.0

This is a major release, the following features were fixed and/or added:

- New option to show the Console window in case of an error message.
- Extend the GUI API and add Gui:GetConeNpages to get the number of schematic pages in the Cone window.
- Support horizontal scrolling in the Wave window with mouse wheel and the shift key.
- Support zooming in the Wave window with mouse wheel and the control key.
- After opening a new VCD file keep the signals loaded in the Wave window if they still match the new VCD file.
- Avoid wrong error "file too small" in the Wave window for large VCD files.
- Do not show the "go up" stroke in the Schem window if the current module has no parent.
- Unconnected hierarchical pins with a constant value can be hidden in the Cone window.
- A binfile saved from the GUI now contains the displayed design title.
- Avoid Tcl error while saving a Binfile but the schematic is not generated yet.
- Read the value of the command line option "-workspace" even if a binary database is given by the "-binfile" option.
- The result of the built-in ERC tools is now loaded to the Memory window.

RTLvision PRO 5.7.1

This is a maintenance release, the following features were fixed and/or added:

- New API for User Defined Arcs in cone extraction.
- Add the option to highlight all clock domains in different colors to the Clock Domain Analyzer window.
- Add the new command line option -netlistPattern to specify a pattern which input files are Verilog netlists.
- No longer strip all symbols from the zdb library needed for using the C-level API.

RTLvision PRO 5.7.0

This is a major release, the following features were fixed and/or added:

- All parser can recognize gzipped file automagically. Therefore the "-gunzip" option becomes obsolete.
- Add the option to guess instance arrays based on instance names to the Verilog parser.
- Add the option to show net names at the connected pins.
- Add the option to the "Export Netlist" feature to write a gzipped output file.
- Add the option "-gzip" to the "$db write -tcl|-verilog|-spice" command to write a gzipped output file.
- Avoid Error is the corresponding symlib of an exported netlist could not be generated, e.g. no cell has a function defined.

RTLvision PRO 5.6.2

This is a maintenance release, the following features were fixed and/or added:

- Extend the GUI API and add Gui:GetParserArgs to get a list of command line arguments that can be used to start a parser directly, e.g. from an Userware script.
- Clear the result list of the Search window after the database has changed.
- Show a list of all loaded cell names in the advanced mode of the Search window.
- Avoid memory segmentation of the internal memory on Windows to support loading larger files without an out of memory error.
- In the delivered pre-compiled VHDL libraries (VDBs) the path to the source files is now relative.
- Fix the visibility of the last line in the Source window.
- Issue better error message when using a range with a non-array (verilog2zdb).
- Better bus support for arrayed instances (verilog2zdb).

RTLvision PRO 5.6.1

This is a maintenance release, the following features were fixed and/or added:

- Do not save the schematic cache at the module using the "@schematic" attribute if the number of instances and nets in the module exceeds the limit of 16 million objects.
- Fix wrong port direction of operator output ports created by the "$db write -verilog" command.
- Add the option to the "Export Netlist" feature to write a symbol library matching the cells in the created netlist.
- Extent the $db symlib create command by the option to create symbols based on the primitive functions of the loaded database cells.
- Create additional netbusses for interfaces bus nets of a module with a known function.
- Add the option to guess buses to the Read Verilog dialog.
- Enable the command line option -guessBus also or the verilog2zdb executable.
- The values of Boolean command line options were not visible in the "Read" dialogs.

- Fix "Goto Line" in the Source window.
- The recreate hierarchy option created an invalid database.
- Guess the input file type also for gzipped files.
- No longer optimize the RTL schematic for "+" and "-" operation with constant values.

RTLvision PRO 5.6.0

This is a major release, the following features were fixed and/or added:

- For windows a setup.exe is now available to install the software.
- Fix the naming of portBus members at the hierarchical instances created by the "Add Hierarchy for Processes" (-procHier) option.
- Fix a crash in the vhdl2vdb executable if a not existing or not readable file is given.
- Add the command line option -argsFromFile to all *Vision tools to read all command line arguments from a file.
- Avoid freezing the GUI on Windows if the Source window tries to add a hyperlink to a non existing include file.
- Fix the operator $db oper rmhier for rare cases where an input port directly connects an output port.
- Changed spos API command foreachrange -uniq option to handle large designs better.

RTLvision PRO 5.5.3

This is a maintenance release, the following features were fixed and/or added:

- Fix for using same file name for reading a binlib and saving a binfile at the same time.
- Add the option -argsFromFile to the *2zdb parser to read all command line arguments from a file.
- Expand environment variables in the file specified with the option -argsFromFile.
- Add the option -wait_for_license to wait the given amount of seconds for the next free license.
- Limit the length of the displayed instance name in the Schem and Cone window to 32 characters.
- Avoid Tcl error while displaying the tooltip for a NULL OID.
- Fix missing cells in the list of trace through candidates.
- Enhance trace through to ignore supply ports.
- Add configuration options for the Tree window to the Preferences dialog.
- Fix the "Get Signals from Cone" button in the Wave window.
- Avoid errors in the cone extract dialog if the database changes while the dialog stays open.
- Add support for string parameters in the Verilog netlist parser (verilog2zdb).

RTLvision PRO 5.5.2

This is a maintenance release, the following features were fixed and/or added:

- A binfile saved from the GUI now contains the context of the Schem and Cone window.
- Fix Y scrolling in the Source window if attributes are displayed.
- Avoid Tcl error in the cone extract dialog if the extraction result list is empty.
- Fix crash in the VCD reader (optimization for timesteps that are a multiple of 2^16 failed).

RTLvision PRO 5.5.1

This is a maintenance release, the following features were fixed and/or added:

- Fix a crash in the "$db write -verilog" command.
- Fix a crash in the "Save Cone as Verilog" function.
- Extend the GUI API procedure Gui:GetSelection to get the selection of the Wave window.
- Fix spos line entries for RTL files included multiple times.
- Fix missing clock flags while creating compact RTL schematics.

RTLvision PRO 5.5.0

This is a major release, the following features were fixed and/or added:

- Extend the GUI API and add Gui:ToggleGreymode to toggle the greymode in either the Schem or Cone window.
- The autohide unconnected pins feature in the Cone window is now enabled per default.
- Add a checkbutton to the toolbar of the Cone window to control the autohide unconnected pins feature.
- Fix a performance issue of the RTL parser for non synthesizable RAM structures.
- The RTL parser always instantiate a subtractor operator for a subtraction in the code.
- The RTL parser always prevent bit-wise adder minimization for adders with (at least one) non-constant input.
- Add support for the command line option -library to the rtl2zdb executable to compile input files into different libraries.
- Remove the command line option "-indexsymlib" because symbol libraries no longer need this index.
- Add hyperlinks in the Source window to directly jump to include files.
- Avoid a Tcl error in the Wave window if "Trace back to X" is invoked from the list of loaded signals.
- Extend the load API by *-sposn*, *-pinsposn* to store file names as is.
- Add support for setting the spos fnamebase.

- Fix a performance issue in the Tree window.

RTLvision PRO 5.4.6

This is a maintenance release, the following features were fixed and/or added:

- Fix the selection of OIDs in the Source window for very long lines.
- Enhance DSPF support. Instances in the "Instance Section" not necessarily need to start with "X".
- Fix the GUI API function Gui:InsertCustomWidget: the given height was not always applied to the created frame.
- Fix the GUI API function Gui:AttributeChanged: port attributes are not updated at hierPins in the Cone window.
- Avoid an error in the Schem window if "Remove Hierarchy" is executed on a hierarchical instance with an object in the path that is instantiated multiple times.
- Extend spos API command foreachrange by the option *-sort* which sorts result by increasing begin positions.
- New configuration option to toggle the display of bus ranges at pinBus and portBus objects in the Schem and Cone window.
- Fix display of constant values in the Schem and Cone window.
- Add a warning message to the VHDL parser for missing generic values.
- Attribute labels in the Source window now appear in the correct order.

RTLvision PRO 5.4.5

This is a maintenance release, the following features were fixed and/or added:

- Remove the "Add" entry from the Bookmark context menu of the Cone window.
- Extend the GUI API and add Gui:LoadModuleToCone to load a module including the contents to the Cone window.
- Add "Export Netlist" to the Tools menu to export the loaded design either as a Spice or Verilog netlist.
- Fix a crash in the "$db write -spice" command.
- Fix bad Spice syntax generated by the "$db write -spice" command.
- Fix wrong connectivity in the output file created by the "$db write -spice" command.
- The commands "$db write -spice" and "$db write -verilog" no longer modify the database.
- Avoid Tcl error in the Preferences dialog.
- Extend the GUI API procedures WaveHighlightTime>>nd WaveClearHighlightTime>>nd add the option to specify a list of signals.
- Enhance pre-processor error message of the Verilog netlist parser (verilog2zdb).

RTLvision PRO 5.4.4

This is a maintenance release, the following features were fixed and/or added:

- The Verilog Netlist Parser (verilog2zdb) no longer creates duplicate portBus names if the same net connects to different pins of a blackbox.
- Redraw the Wave window after the GUI API procedure Gui:WaveClearHighlightTime was called.
- Speed-up the signal selection widget in the Wave window.
- Fix a crash in the RTL parser.
- Fix a crash in the Schem window while calculating the wiring.

RTLvision PRO 5.4.3

This is a maintenance release, the following features were fixed and/or added:

- Fix renaming bookmarks in the Schem window.
- Support persistent bookmarks (stored in the binfile) in the Schem and Cone window.
- The RTL parser now add parameters as module attributes.
- Module highlights are now also visible in the Tree window.
- The GUI API function Gui:PersistChanged now updates the visibility of all windows that can be controlled by a Persist variable.
- The GUI API function Gui:SetCurrentModule now accepts an empty OID to clear the Schem window and reset the Tree window.
- The name of a given binfile is no longer saved to the workspace file.
- The GUI API function Gui:Quit now accepts an optional return code as the exit status.
- New Userware example (demo/api/Diff/diff.tcl) to show the differences of two revisions of the same netlist.

RTLvision PRO 5.4.2

This is a maintenance release, the following features were fixed and/or added:

- Fix wrong syntax error reported by the SDF parser.
- Filter duplicate error messages of the SDF parser.
- Add the command line option -sdfTop to specify the optional name of the top level instance.
- Fix drawing issue of a value change in the Wave window on the Windows platform.
- Enhance the database command oid createFromString and add the option "-topInstName" to specify the name of the top level instance in the given path.

RTLvision PRO 5.4.1

This is a maintenance release, the following features were fixed and/or added:

- In some cases no source code reference data was added for Verilog include files.
- Fix the display order of netBus members in the Wave window.
- Add the option to the Wave window to limit the time read from a VCD file.
- Extend the popup menu of the Wave window by the "Goto Time" entry to jump to a specific time step.
- Fix drawing issues in the Wave window.
- The Verilog Netlist Parser (verilog2zdb) creates labels for constants showing the value.
- Add the command line option -vcdTop again to specify the optional name of the design under test.
- The Verilog Netlist Parser (verilog2zdb) now supports "X" and "Z" in constants.
- New Verilog parser option to recreate hierarchy from flat instance names.
- Enhance the Cone Extraction API and add the option "-diveEmpty" to dive into empty modules while running the Cone extraction.
- Highlights are now also visible in the Wave window.
- Fix the display of pinBus and portBus ranges in the Schem and Cone window.
- Fix a MSB, LSB issue in displaying constant values in the Schem window.
- Fix the display of constant values in the Cone window.
- The database command "$db write -verilog" can now handle constant nets.
- Fix wrong syntax created by the "$db write -spice" command.
- The Verilog RTL parser no longer stops the elaboration if an external net (out of module reference) is detected. Now a warnings is shown and the rest of the module will be elaborated.
- In the Schem window bookmarks can be added.
- Add the possibility to rename Schem, Cone and Source bookmarks.
- Extend the GUI API and add WaveHighlightTime>>nd WaveClearHighlightTime>>o highlight time ranges in the Wave window.
- All device types optionally support additional input ports after the last required port.
- Avoid long labels at bus rippers.
- Fix the "Internal error - Pin with no up/down/left/right direction" while generating a multi page device schematic.

RTLvision PRO 5.4.0

This is a major release, the following features were fixed and/or added:

- Add the option to create compact RTL schematics.

- If a port or portBus OID is dropped to the Wave window then the waveform of the connected net or netBus is shown.
- Remove the obsolete command line option -vcdTop.
- Add a button to the Wave window to zoom fit.
- Add the option to select signals from the opened VCD file without a loaded design.
- Display signal names in the Wave window with the full path name.
- The Clock Domain Analyzer window now shows the name of the clock source.
- Extend the GUI API and add AddSourceBookmark>>nd DeleteSourceBookmark>>o add/remove bookmarks to the Source window.
- Extend the GUI API and add NewWaveWindow>>nd HideWaveWindow>>o show/hide a new toplevel waveform window.
- Add an optional argument to all Waveform window related GUI API procedures to work on one of the created toplevel windows.
- Remove the database API net flag visit4.
- Enhance the database net and netBus objects to support constant values.
- Primitives with the function "X" and "Z" are no longer supported. The corresponding value can be added as a constant net value.
- New database API command "$db foreach operator" and "$db foreach primInst" to loop over all operators and primitive instances in a module.
- Show registers and operators for each module in the Tree window.
- Enhance the operator $db oper bulk by the new option "nopg" to show only bulk pins not connected to a power or ground node regardless of the device function.
- New sub command "remove" added to the $db symlib command.
- Fix the database command "$db write -verilog": do not write out primitives, add `celldefine to the created module.
- All cells with the "libcell" flag are treated as primitives.
- The progress is now moving while reading many RTL files.
- Add a toolbar to the Schem and Cone window.

RTLvision PRO 5.3.10

This is a maintenance release, the following features were fixed and/or added:

- The SDF parser no longer stops if a non fatal error occurs.
- Start the SDF parser if a SDF file was specified on the command line.
- The cone extraction treats each pin or port with an unknown direction (black box) as a bidirectional pin or port.
- The check mark unicode character to indicate the current module in the Tree window could not be display on Windows XP.

- Fix displaced line numbers in the Source window on Windows.

RTLvision PRO 5.3.9

This is a maintenance release, the following features were fixed and/or added:

- Fix the drop of objects from the Source window to the Schem and Cone window.
- A goto command on a blackbox instance now highlights this instance in the Source window using the goto color.
- The database API command tdevice now accepts a primitive OID.
- Add the possibility to specify devices as trace through cells.

RTLvision PRO 5.3.8

This is a maintenance release, the following features were fixed and/or added:

- New zdb API example zdbDump.c that dumps the contents of a binary database as an ASCII text file.
- Fix the database command $db write -verilog for primitives with buses.
- Disabled page splitting is now also used by the Export EDIF, Export Skill and print design hierarchy tree functions.
- Fix a problem with Copy & Paste to other X application on Unix platforms.
- Fix the file completion in the Console window for cases were the entered text starts with the tilde character.
- Fix the drop of module based OIDs to the Schem and Cone window.
- Enhance Export EDIF optimized for Tanner (S-Edit): add support for complex overlay and bus connections.

RTLvision PRO 5.3.7

This is a maintenance release, the following features were fixed and/or added:

- Fix endless loop in the PDF printer with very long labels in the Schem or Cone window.
- New Operator API function reducePins to collapse pins that have the same net connected to multiple pins.
- The highlight of the Clock Domain Analyzer window is now in sync with the rest of the GUI.

RTLvision PRO 5.3.6

This is a maintenance release, the following features were fixed and/or added:

- Limit the length of the displayed cell name in the Schem and Cone window to 32 characters.
- Enhance the Operator API function guessWide by the options "-equal", "-mux", "-prio", "-dlatch"

and "-repeat" to control the guessed types.

- Enhance the Operator API function createConst by the option "-bool" to remove AND/OR primitives with constant 0/1 at one input pin.
- Avoid a Tcl error in the Source window on Windows if a new file is selected from the list of files.
- The Read RTL dialog now remembers the last selected tab.

RTLvision PRO 5.3.5

This is a maintenance release, the following features were fixed and/or added:

- Fix the drop of modules from the Source to the Cone window.
- The Liberty reader "liberty2zdb" no longer creates unknown port direction for missing direction.
- A cone extraction from the context menu to top level I/O ports no longer stops at clocked cells.
- Fix the database command $db write -verilog if a cell contains a port with an unknown direction.
- New Operator API function removeInv to remove all INV instances if the input connects to a power or ground net.
- New Operator API function mergeNetBus to merge two netBuses.
- New Operator API function createConst to replace power ground nets with a "constant" net stub.
- New Operator API function guessWide to combine multiple scalar primitives into one wide primitive.
- New Operator API function chain -mux to replace chains of MUX/WIDE_MUX primitives by PRIO_SELECTOR/WIDE_PRIO_SELECTOR instances.
- New Operator API function chain -bool to replace chains of boolean (OR/AND) primitives by reduced boolean (REDUCE_OR/REDUCE_AND) instances.
- Guess the displayed ripper index from the subnet name.
- The database command $db clone can now create a bitblasted clone of an object.

RTLvision PRO 5.3.4

This is a maintenance release, the following features were fixed and/or added:

- Fix crash in Verilog parser if the creation of source code references is turned off.
- Fix "jump to previous bookmark" in the Source window starting from a line behind the last bookmark.
- The attribute display in the Source window could not be turned off.
- The database command $db clone can now create a clone of a primitive object.
- New Operator API function removeBuf to remove all BUF and WIDE_BUF instances and merge the connected nets.

RTLvision PRO 5.3.3

This is a maintenance release, the following features were fixed and/or added:

- If an instance OID is dropped to the Wave window then the waveform of all nets connected to the instance are shown.
- Fix startup error if the license feature gv-wave is available but cannot be checked out.
- Signals loaded to the Wave window are displayed with the full path name.
- Sub net index numbers are display at netBus ripper.
- Add the option to show the members of a bus object in the tooltip.
- Fix bus handling in Export EDIF optimized for Tanner (S-Edit).

RTLvision PRO 5.3.2

This is a maintenance release, the following features were fixed and/or added:

- The database command $db write -verilog can now write primitives with buses.
- Fix the direction of cone extraction to flip-flops and top level I/O ports in the Cone entry of the context menu.
- New database API command "$db foreach clocked" to loop over all clocked cells in a module.
- In the Source window bookmarks can be added.
- Use smaller icons in the action column of the Source window.
- Use new icons in the Tree window to indicate the current module and that a module is instantiated multiple times.

RTLvision PRO 5.3.1

This is a maintenance release, the following features were fixed and/or added:

- Avoid error if the mouse enters an OID with a next spos in the Source window.
- Support pinBus and portBus OID type for the option to extract the cone to flip-flops and top level I/O ports in the "Cone" entry of the context menu.
- Built-in symbol shape of a PRIO_SELECTOR changed.

RTLvision PRO 5.3.0

This is a major release, the following features were fixed and/or added:

- The schematic page sizes are now labeled "S", "M", "L", "XL" and "XXL".
- Increase the default page size of the schematic from "S" to "M".
- Add the option to export hierarchical EDIF.
- Add the option to export EDIF optimized for Tanner (S-Edit).

- Add the option to disable page splitting in the Schem window.
- Highlights are now also visible in the Tree window.
- Add the option to extract the cone to flip-flops and top level I/O ports to the "Cone" entry of the context menu.
- Highlighted instance objects in the Schem and Cone window now display a highlight background color.
- The Verilog RTL parser now does a more relaxed language checking and report warnings for many non fatal errors.
- The option "-minRamSize" is enabled per default with a value of 4096.
- Avoid a Tcl error in the "Load Cone" function of the Navigate Signal dialog if a module based OID was selected.
- Fixed guessPortBus and guessNetBus to produce buses containing only one member.
- Do not update the Magnify window if the schematic is moved with the middle mouse button.
- Update the Magnify window if the zoom factor of the schematic has changed.
- Extend spos API by the new command maxcolumn to get the length of the longest line of the given file.

RTLvision PRO 5.2.4

This is a maintenance release, the following features were fixed and/or added:

- New API function isTop to check if a module is a top module.
- New API function spos add to add spos information to an OID.
- Enhance the load net API command and add the option -pinspos to specify a source position for a pin.
- Fix the "Fold Library Cells" option of the Clock Domain Analyzer window.
- Avoid a Tcl error in the Source window if an object that contains special characters in its name is selected.
- New checkbutton in Preferences dialog to disable pin permutation.
- New command line option "-minRamSize" and new entry field to change the limit for the generation of multiport ram.

RTLvision PRO 5.2.3

This is a maintenance release, the following features were fixed and/or added:

- Blackbox modules are displayed in the color defined for greymode in the Tree window.
- The nodes in the Tree for quick access to ports, nets and primitive instances of each hierarchical module now support Drag & Drop.
- Avoid error if Cone specific entries in the Tools menu are invoked.

- On Windows a binary database can now be saved with the same name as the currently loaded binfile.
- The Search window ignored the selected result type for the mode "Hier".
- Extend the GUI API and add Gui:ConeFold, Gui:ConeUnfold and Gui:ConeIsFolded to set, remove and check the fold flag of modules in the Cone window.

RTLvision PRO 5.2.2

This is a maintenance release, the following features were fixed and/or added:

- Fix startup error if license feature gv-wave is not available.
- Allow multiple search paths for vhdl libraries with -vhdlLibPath.

RTLvision PRO 5.2.1

This is a maintenance release, the following features were fixed and/or added:

- Extend the GUI API and add procedures to control the Wave window.
- Remove the T/S (Tree & Search) and the All tab from the Visualizer. Now more individual views can be added from the Window menu.
- The Display Pin Names setting was not working.
- The database command $db write -tcl does not produce code to create the $db database anymore. This has to be done manually before sourcing the created file.
- Fix Save Cone As ... hierarchical netlist if a lot of objects are loaded to the Cone window.
- Fix Save Cone As ... flat netlist if two instances of the same module (that contains a netBus) are loaded to the Cone window.
- Fix Save Cone As ... if a netBus that contains a gap is loaded to the Cone window.
- Fix environment variable support in a Verilog fileset.

RTLvision PRO 5.2.0

This is a major release, the following features were fixed and/or added:

- Add support for reading VCD files and display a waveform.
- Add support to display top/bottom ports in the Schem and Cone window.
- Extend the Beautify menu by the entry "Position" to define the location of a port to top, bottom, left or right.
- Add the database API port flags top, bottom, left and right to place a port at the specified location.
- Enhance fullfit zoom in the Schem and Cone window. The new zoom-lock mode automatically adjusts the zoom factor whenever the window size changes.
- Enhance the Tree to allow quick access to ports, nets and primitive instances of each

hierarchical module.

- Increase the performance of the Source window if there are a lot of objects at the same position (e.g. very wide buses).
- Extend the GUI API functions Gui:Print and Gui:Log by the optional arguments "type", "filename" and "line".
- Extend the GUI API function Gui:PrintError by the optional arguments "filename" and "line".
- New GUI API function Gui:PrintWithCallback to add a custom binding to the text in the Console window.
- Extend the database API command load inst: add the new option -orient to specify the orientation of the instance.
- New database API command orient to get/set the orientation of an instance.
- Extend the database API command flag: add new cell flag "feedthru" to mark all instances of the cell as a feedthru component (used in the Cone window).
- New Operator API functions guessPortBus, guessNetBus and guessInstArray to guess buses.
- Fixed Database API function oid createFromString for cases if the pin delimiter is specified and is the same character as the hierarchy separator.
- The Minimap window can be moved inside the Schem window.
- New Magnify window to provide a detailed view of the schematic under the mouse.
- Change the number of highlight colors to 14 and add 14 additional permanent highlight colors.
- Extend the database API commands $db hilight and $db flathilight by the option "-permanent" to define permanent highlight colors.

RTLvision PRO 5.1.3

This is a maintenance release, the following features were fixed and/or added:

- The API command $db flatattr propagateValue now overwrites an existing "value" attribute.
- Avoid a Tcl error in the Cone window for a double click on pinBus objects with an ambiguous bus connection.
- The statusbar will be cleared 20 seconds after the last message was displayed.
- Avoid a Tcl error in the Cone window if a restored bookmark file is validated against the loaded database.
- The report created by the "Save as Text" feature of the Clock Domain Analyzer window contained the wrong number of flip-flops.
- Fix reading an EDIF file if the file extension is not "edif" or "edf".
- Add capability to analyze SystemVerilog language subset 2005 only (by ignoring 2009 keywords).
- Avoid an error for net segment objects displayed in the Infobox.

RTLvision PRO 5.1.2

This is a maintenance release, the following features were fixed and/or added:

- If signal mode is turned on then a *more* operation in the Cone window will transparently go through hierarchy borders.
  In this mode the newly loaded objects will be temporarily colored using the goto color.
- Enable Ctrl double-click in the Cone window on port objects to get a dialog to select the component to be loaded.
- Fix syntax highlighting in the Source window for gzipped files.
- Very long lines are display correct in the Source window while scrolling.
- Start verilog elaboration at top modules.
- The result of a clock domain search in the Clock Domain Analyzer window is now sorted by the total number of flip-flops.
- Enhance the Cone Extraction API and add the option to stop at all power/ground nets.
- Enhance the Cone Extraction dialog and add the option to stop at all power/ground nets.

RTLvision PRO 5.1.1

This is a maintenance release, the following features were fixed and/or added:

- The option "Do not display hierarchy boxes in result" of a CDC search in the Clock Domain Analyzer window was not working correctly.
- Enhance the primetime.tcl Userware to support entries spread over multiple lines.
- Fix the primetime.tcl Userware: add the correct attribute name to the zdb.
- A restored bookmark file can be validated against the loaded database.

RTLvision PRO 5.1.0

This is a major release, the following features were fixed and/or added:

- Update the appearance of the GUI.
- The Search window caches the last search strings and provides autocompletion for repeated queries. A selection menu provides fast access to the cached search queries.
- Show the Visualizer selection in the Print, Save Schematic as Image, Export EDIF and Export Skill dialogs only if there is at least one additional Visualizer visible.
- Combine the two dialogs Read Verilog and Read Edif into one single dialog window.
- The option "Display FFs in columns" to display the result of a CDC search in the Clock Domain Analyzer window was not working correctly.
- Fix merging design files with a binary library that contains blackbox cells.
- The configuration options from the "Visu" tab of the Preferences dialog moved to the "Misc"

Misc tab.

- Comments can be added to all objects.
- Add the possibility to specify trace through cells for incremental navigation in the Cone window.
- Enhance the flat foreach pin loop by the option -addHier to process hierarchical pins and ports.
- Add support to drop objects on a tab of the Visualizer.
- Add more shapes to the graphical marks.
- New GUI API function Gui:DoubleClick to register a customer specific double click binding.
- The behavior of the GUI API function Gui:RegisterDataBaseChanged has changed. Now the registered procedure is called before the schematic is rendered. If the registered Userware procedure needs to access the Schem window then the GUI API procedure Gui:SetCurrentModule needs to be called before.
- Swap the key binding for the page navigation in the Schem and Cone window. The "PgUp" key jumps to the next schematic page and the "PgDn" key to the previous.
- The pane tab elements divided by splitters are no longer covered while moving the splitters.
- Fix the command and file completion in the Console window for cases were a file or directory with the name of a matched command exists.
- Fix the command and file completion in the Console window on Windows for cases were a the entered text contains illegal characters for a file or directory.
- The RTL parser no longer renames long cell names (e.g. parameterized cells).
- Verilog and VHDL parser use ReadPort/WritePort operators and special RamNet to represent multi-port RAM behavior for smaller netlists.
- Blackbox instances are displayed in the color defined for greymode in the Schem, Cone and Source window.
- Support horizontal scrolling in Schem, Cone and Minimap window with mouse wheel and the shift key.
- Fix the GUI API procedure Gui:CustomizePopup: the wrong popup menu path was added to the callback.
- Increase default value for the "Big Module Limit" to 45000 objects.

RTLvision PRO 5.0.0

This is a major release, the following features were fixed and/or added:

- Add an action column to the Source window for easier navigation.
- Enhance the Source window and add the possibility to display pin attributes.
- Add support to read gzipped Verilog and VHDL files.
- Added vhdl2000 and vhdl2008 dialect support.
- Add highlight information to the bookmark file created with the context menu entry "Bookmark->Save As".

- Improve the search in file function of the Source window: a search starts from the currently displayed line, add the possibility to search backward and allow the search across multiple files.
- Extend spos API by the new command mtime to get the modification time of the given file.
- Add the option to load all nets hidden in the Big Module Limit dialog.
- Add the option to save the contents of the Report Instance Count dialog to a text file.
- Extend the Save Cone As Verilog dialog to create a Verilog netlist using named connectivity.
- Enhance the database command $db write and add the option to save the contents of the database as a Verilog netlist using named connectivity.
- Enhance the GUI API function Gui:SaveConeAs Verilog using named connectivity.

RTLvision PRO 4.7.4

This is a maintenance release, the following features were fixed and/or added:

RTLvision PRO 4.7.3

This is a maintenance release, the following features were fixed and/or added:

- The "liberty2zdb" executable now suppresses duplicate messages about unhandled group types.
- Speedup the Liberty reader "liberty2zdb" and reduce the peak memory usage.
- The option "-boolSym" of the Liberty reader "liberty2zdb" now accepts on/off values.
- The "oid createFromString" function can now create port, portBus and netBus OID types.
- Fix return code handling in flat foreach loops.
- New Userware example printPDF.tcl to print a schematic as a PDF file.

RTLvision PRO 4.7.2

This is a maintenance release, the following features were fixed and/or added:

- The "liberty2zdb" executable now uses the port direction named in the Liberty file for the top/bottom power and ground pins of the created symbol.
- The "liberty2zdb" executable could not create a symbol from a Liberty cell with one or more "pg_pin" with an output direction.
- New Userware example saveHilight.tcl to save all currently set highlights into a file.
- Avoid errors for rare cases were NULL OIDs can be selected in the GUI.

RTLvision PRO 4.7.1

This is a maintenance release, the following features were fixed and/or added:

- Stop displaying line numbers in the Source window if the end of the file has been reached.

- Fix crash in "flatattr delete" if there are flags set at netsegments.
- On Windows the Source window used a different line spacing for displaying normal text and comments. This resulted in shifted line numbers.
- On Windows the scrollbar of the Source window did not adjust to the cached scroll position when navigating through the design files.
- Navigate to the previous or next spos in the Source window did not highlight the line containing the object.
- In the Source window the colors of marked keywords are now visible through the selection.
- Avoid Tcl Error if "Signal Mode" is turned on and "Navigate Signal" on a module based OID is selected from the Source window.

RTLvision PRO 4.7.0

This is a major release, the following features were fixed and/or added:

- Enhance the database to store source positions for pin objects.
- The RTL parser always creates spos data for the Source window.
- The RTL parser creates source positions for pins.
- The Verilog parser creates source positions for pins.
- The Edif parser creates source positions for pins.
- The Edif parser no longer stops creating pin connections of a net if one connection could not be completed.
- Fix the displayed result of a CDC search in the Clock Domain Analyzer window.
- Fix crash in spos database for cases where Verilog library files are read multiple times.
- Add more options to navigate through a design in the Source window.
- Fix the file completion in the Console window if a directory or a file name contains a blank.
- Added object type as priority for "spos pick" (bus type objects have a higher priority).
- Allow "end" for spos lineno.
- Scrolling in the Source window is smoother, the scrollbar is no longer wobbling.
- Significantly increase the performance of the Source window if the syntax highlighting feature is enabled.

RTLvision PRO 4.6.5

This is a maintenance release, the following features were fixed and/or added:

- Display all possible matches of the command and file completion in the Console window.

RTLvision PRO 4.6.4

This is a maintenance release, the following features were fixed and/or added:

- The Liberty parser "liberty2zdb" now adds top and bottom ports to the symbol for the power and ground pins of a recognized function.
- Significantly increase the performance of the Source window if the syntax highlighting feature is enabled.
- Fix the "Show Frame" option in the Print dialog.
- Simplify the Print dialog, reduce the number of options.

RTLvision PRO 4.6.3

This is a maintenance release, the following features were fixed and/or added:

RTLvision PRO 4.6.2

This is a maintenance release, the following features were fixed and/or added:

- Distribute all possible matches shown by the command and file completion in the Console window to multiple output lines.
- Fix flat foreach instOfCell: continue traversing the hierarchy below a flagged cell.

RTLvision PRO 4.6.1

This is a maintenance release, the following features were fixed and/or added:

- New GUI API function Gui:Log to print a log message to the log file.
- Add all commands typed and executed in the Console window to the log file.
- Fix command and file completion in the Console window if the partly entered command or file starts with a dash character.
- Fix command and file completion in the Console window if a custom prompt is not ending with a blank (the blank is added automatically).
- Enable syntax highlighting of Verilog keywords in the Source window for files with the extension ".vlib", ".sv" and ".h".
- Elaborate verilog before VHDL in mixed language designs.

RTLvision PRO 4.6.0

This is a major release, the following features were fixed and/or added:

- Add support to print schematics as PDF.
- Add command and file completion in the Console window using the Tab key.

- The displayed prompt in the Console window can be customized using the Persist(prompt) variable.
- Add support to format attributes using a custom foreground and background color.
- Enable support for variable slices in the VHDL parser.
- Fix the displayed number of flip-flops belonging to a clock domain in the Clock Domain Analyzer window.
- Enhance path extraction: if the value of the -pathLimit option is 0 then up to 65535 paths are returned.
- Update Liberty parser "liberty2zdb" to support the latest Liberty format enhancements.
- A displayed instance attributes no longer overwrites the cell name in the schematic.
- Fix "get clipboard" function used by all paste operations.

RTLvision PRO 4.5.2

This is a maintenance release, the following features were fixed and/or added:

- Fix issues in highlighting keywords and comments in the Source window and beautify the displayed text.

RTLvision PRO 4.5.1

This is a maintenance release, the following features were fixed and/or added:

- Improve Verilog Parser error messages for syntax errors.
- Extend the Edif parser: if a cell specifies a userData property with the name primfunc then the value is used to specify the primitive function of this cell.
- Expand environment variables and home directories in symlib file names.

RTLvision PRO 4.5.0

This is a major release, the following features were fixed and/or added:

- Extend the database command $db write and add the option to save the contents of the database as a Verilog netlist.
- Add more common keyboard shortcuts.
- Bind a mouse double click in the Schem window to hide/unhide nets.
- The workspace file now stores the size and the position of the application window.
- Avoid Tcl error in the Source window function search in file if the displayed file was changed while performing the search.
- Extend the Print dialog by the check button "Show Frame" to control the visibility of the page frame.
- Extend GUI API command Gui:AppendPopup by the optional argument "menuname" to specify

the name of the cascade menu.

- Select multiple Pins/Ports using the NW mouse stroke now also includes portBus and pinBus objects.
- Add new option "-sdbl" to the rtl2zdb executable to restore a Verific binary database (sdbl).
- New executable ndl2zdb that translates NDL netlists into a binary zdb database.
- The "Beautify->Select Symbol" dialog can now handle black box instantiations with unknown port directions.
- Change behavior of "Open Editor" in the Source window: if there is either an object marked by the Goto function or a line marked by "Goto Line" then this information is used as the start line passed to the editor.

RTLvision PRO 4.4.2

This is a maintenance release, the following features were fixed and/or added:

- Fix potential crash in liberty2zdb.

RTLvision PRO 4.4.1

This is a maintenance release, the following features were fixed and/or added:

- Add new graphical marks to the Meta Attributes documentation.
- Extend the primetime.tcl and pathmill.tcl Userware example to display multiple values in the Schem and Cone window at a time.
- New GUI API function Gui:DataBaseModified to update the GUI after modifying the currently loaded database.

RTLvision PRO 4.4.0

This is a major release, the following features were fixed and/or added:

- Fix flat foreach signal (in spice examples with empty sub-circuits, nets may have been skipped).
- Fix sym2zdb: if no binlib is specified then create cells with symbol shapes for all symbols in the given symlib files else only add symbol shapes to existing cells in the binlib.
- Add the options -ignorePragmas and -ignoreTranslate to RTLvision PRO and the executables rtl2zdb and vhdl2vdb to either ignore all pragmas or to ignore only synthesis pragmas.
- Matching symlib file entries with symbol-ports that don't match by name now issue a warning.
- Symlib file may define a mapping to built-in symbol shapes. This mapping may additionally define certain ports as hidden, e.g. "symbol NDX3 * NAND port vdd input.hidden" will display cell NDX3 with a NAND shape and will hide the cell port "vdd" (internally, the port's hide flag is set).

RTLvision PRO 4.3.2

This is a maintenance release, the following features were fixed and/or added:

- Fix Tcl error while updating the Schem window after the database was modified.
- Extend the database API command $db search by the option *-icase* to search for objects in a case insensitive manner.
- New Userware example removeBuf.tcl to remove all buffer elements and merge the nets connected to the buffer.
- Add the possibility to display a dotted grid in the Schem and Cone window.
- Add the option to turn off the visibility of the hierarchy boxes in the displayed result of the Clock Domain Analyzer.
- Now an operator (module with a known bus-level function) is treated as a primitive (this behavior can be changed in the Preferences dialog).

RTLvision PRO 4.3.1

This is a maintenance release, the following features were fixed and/or added:

- Changed VHDL -procHier handling to use process label names instead of "PROCESS_<d>".
- Added source info to hierarchical instances created by -procHier option.

RTLvision PRO 4.3.0

This is a major release, the following features were fixed and/or added:

- New API command $db setPrimitive to specify hierarchical cells as primitives.
- Extend the Preferences dialog to specify hierarchical cells as primitives.
- Dropping objects to the Schem and Cone window will set the input focus to the window under the mouse.
- New API command $db flatattr propagateValue to propagate net values to the connect pins.
- New API command $db isOneToOneConnection to check if all subnets of a netBus are connected to a pinBus or portBus.
- New API command $db oid print to print the string representation of an OID.
- Attributes named "value" are automatically displayed at pin and port objects.
- New workspace file that defines a Virtuoso like color scheme for the Schematic.
- Fix performance issue of Source window for very long lines containing a lot of OIDs.
- Fix performance issue of Source window if the displayed part of the file contains a lot of OIDs.
- Extend spos API command foreachrange by the option *-uniq* to suppresses oids which have the same type and lie on the same begin and end position as previous OIDs.

RTLvision PRO 4.2.2

This is a maintenance release, the following features were fixed and/or added:

- Ignore case of attribute names for attribute values displayed in the Schem window.
- Avoid error in the set "Current Module" function in the context menu of the Tree window.
- New API command $db flatattr deleteAll removes all flat occurrences of a given attribute.
- Unhighlight a specific color removed all highlight infos from the Clock Domain Analyzer window.
- Custom power/ground symbol shapes with wrong graphical pin direction now create a warning instead of an error.
- Bugfix EDIF export: built-in as well as user-defined port symbols are exported as GRAPHIC cells (the EDIF standard requires this in cases where the ports are issued as portImplementation constructs).

RTLvision PRO 4.2.1

This is a maintenance release, the following features were fixed and/or added:

- Add new option to the Source window to search for a text pattern in the currently displayed file.
- Fix a performance issue in the Source window if the displayed part of a file contains heavy connected nets.

RTLvision PRO 4.2.0

This is a major release, the following features were fixed and/or added:

- Remove the predefined color schemes light, dark and bisque. Now there is only one "default" color scheme where the color values can still be changed using the Colors tab of the Preferences dialog.
- The default color scheme now defines 19 different highlight colors.
- The workspace file demo/api/printableColors.ws defines a printer friendly color set.
- The Verilog Netlist Parser (verilog2zdb) now reports an error when it detects an recursive instantiation.
- Bugfix M280: the -define (and \+define) options for the Verilog Netlist Parser (verilog2zdb) now has priority over the `define in the Verilog files.
- Upgrade FlexNet license mechanism to FlexNet-11.6 for the following platforms: linux23, linux23.x86_64, sun10.x86_64 and win32. All other platforms still use FlexNet-10.8.
- New "Optimize Wiring" option in the context menu of the Schem and Cone window to re-compute the net routing.
- Bugfix M055: The API function rmhier (if called without -hiersep) now automatically tries up to 9 different prefixes to avoid name clashes before it gives up.

- Add the option to turn off the visibility of hierarchy boxes in the Cone window.
- Simplify the Search window.
- Fix Control-Double-Click in the Cone window. Now the Bus Dialog is always displayed.
- Autohide unconnected pins in the Cone window will not hide pins at instances with less or equal 8 pins. Toggle autohide can still hide the unconnected pins.

RTLvision PRO 4.1.0

This is a major release, the following features were fixed and/or added:

- Add new Clock Domain Analyzer Tutorial.
- Filter database cells in the Clocked Cells dialog: show only primitives and empty modules.
- The liberty2zdb executable automatically recognize clocked cells.
- Net name labels use transparent background to avoid visual outages at wires.
- New API command $db spos addfile to create a new file entry in the spos database.
- Add a "Dialog" command to the "Beautify->Select Symbol" menu. The Symlib dialog assists the user to select symbols for unknown cells and specify port assignments.
- Add selection of builtin symbol shapes to the "Beautify->Select Symbol" menu for UNKNOWN primitives.
- New feature Nethide to hide all nets in the Schem window.
- New Userware example annotation.tcl to add comments to objects and show/print a report.
- New zdb API example createZdb.c demonstrates how to create and fill a binary database using the C-level API.

RTLvision PRO 4.0.6

This is a maintenance release, the following features were fixed and/or added:

- Add file extension ".edn" to the open EDIF file dialog.
- Add support for built-in symbol shapes in symlib2zdb and zsymlib command.
- Avoid highlight error if a hierPin contained highlight information.

RTLvision PRO 4.0.5

This is a maintenance release, the following features were fixed and/or added:

- Fix GUI API function Gui:HighlightChanged: Source window was not updated after the highlight has been deleted.

RTLvision PRO 4.0.4

This is a maintenance release, the following features were fixed and/or added:

- Extend GUI API function Gui:Goto to select a target Window.
- Fixed win32 filename problem in the VHDL Library Compiler.

RTLvision PRO 4.0.3

This is a maintenance release, the following features were fixed and/or added:

RTLvision PRO 4.0.2

This is a maintenance release, the following features were fixed and/or added:

- Fix zdb2sym: also write symbols from cell attribute @symbol.
- New GUI API function Gui:WriteDspf to Write the displayed RC network as a Spice netlist.

RTLvision PRO 4.0.1

This is a maintenance release, the following features were fixed and/or added:

RTLvision PRO 4.0.0

This is a major release, the following features were fixed and/or added:

- New Minimap to provide an overview of the displayed schematic.
- New database command $db tools createHier to recreate hierarchy from flat instance names.
- Speedup (approx. 2X) for the -gunzip option at the Verilog Netlist Parser (verilog2zdb). The progress bar calculation with -gunzip fixed.
- Changed scaling in cadence2symlib.il and skill export, to get better on grid placement.
- For convenience there are new scripts in the package root which determine the platform dependent application path automatically.

RTLvision PRO 3.4.3

This is a maintenance release, the following features were fixed and/or added:

- New GUI API function Gui:HighlightSourceLine to highlight an arbitrary line in the Source window.

RTLvision PRO 3.4.2

This is a maintenance release, the following features were fixed and/or added:

- Fix: crash in nested filesets.
- Fix error in Print dialog for print views "Full page" and "All Pages".
- While selecting multiple objects with a rectangle the direction controls the selected object type.

RTLvision PRO 3.4.1

This is a maintenance release, the following features were fixed and/or added:

- Fix Infobox's Bus tab: enable Drag & Drop and tooltip for net objects which are part of a netBus and for all bundle objects.
- New GUI API functions Gui:InsertCustomWidget and Gui:RemoveCustomWidget to insert and remove a custom widget (similar to Gui:NewVisualizer).

RTLvision PRO 3.4.0

This is a major release, the following features were fixed and/or added:

- New database command $db oid resetAllOIDs to reset the string representation of all Tcl variables referring to OIDs.
- The Gui:Cd command is gone. The Tcl builtin "cd" should be used now.
- API extended by the search command.
- Search window limits the number of selected objects to 10,000.
- Search window's "Stop" button (interrupt) now accepts a button press at all times.
- Drag & Drop now waits 300 milliseconds before it switches the Tabs also on Windows (if the cursor stays on a Tab field).
- Reduce flicker in tooltips.
- Move exportsymlib.il to package Symutils as cadence2symlib.il.
- Fix broken tooltips.
- The files in a Verilog fileset are relative to the current working directory.
- The Read Verilog dialog displays the fileset as one list similar to the Read RTL dialog.
- Enable SDF support for RTLvision PRO.
- New appearance of the Clock Domain Analyzer window for better usability.
- Binfile is incompatible to previous releases.
- New Rotate submenu in the Beautify menu of the Schem and Cone window to rotate and mirror instances.
- Enable dragging of instances in the Schem and Cone window.
- Introduce Signal Mode in the Schem and Cone window for fast access to signal OIDs.
- New option Optimize in the Schem and Cone window.

RTLvision PRO 3.3.3

This is a maintenance release, the following features were fixed and/or added:

- Invalid objects can be dropped to the Memory window.

- Display bus width for pinBus, portBus and netBus objects in tooltips, Memory window and Last Selection label.

RTLvision PRO 3.3.2

This is a maintenance release, the following features were fixed and/or added:

- Fix broken table layout in built in html browser.
- Fix broken tooltips on the Windows Platform.

RTLvision PRO 3.3.1

This is a maintenance release, the following features were fixed and/or added:

- Verilog gate-level parser supports multiple instances of an UNDEFINED module (black box) with different named-port references (different interface).

RTLvision PRO 3.3.0

This is a major release, the following features were fixed and/or added:

- Simplify the Read Edif dialog.
- If a gzipped file is loaded to the "Read Edif" or "Read Verilog" dialog then the option "Read gzipped Edif" respectively "Read gzipped Verilog" is turned on automatically.
- Bugfix: dropping objects from Source window to Cone window always failed.
- The number of available encodings no longer depends on a local Tcl installation. Now all encodings from a standard Tcl installation are packed into the tool.

RTLvision PRO 3.2.3

This is a maintenance release, the following features were fixed and/or added:

- New binding to the mouse wheel with the Control button pressed enables zoom in and zoom out in the Schem and Cone window.

RTLvision PRO 3.2.2

This is a maintenance release, the following features were fixed and/or added:

- Tooltips now disappear after max. 7 seconds.
- Verilog Netlist Parser (verilog2zdb) now silently ignores the "generate" and "genvar" keywords (Verilog 2001).
- EDIF: written section don’t require a timeStamp.
- Bugfix: Remove newline in error message. Change warning for redefined modules into two messages to display both file locations.

- The Cone history function is turned on by default.

RTLvision PRO 3.2.1

This is a maintenance release, the following features were fixed and/or added:

- Bugfix: Verilog Netlist Parser (verilog2zdb) now handles comments, quotes, etc in escaped identifiers correctly (escaped identifiers start with a backslash). Also the error messages are improved to locate the start of an unterminated comment. Unterminated quoted strings are reported at end-of-line.

RTLvision PRO 3.2.0

This is a major release, the following features were fixed and/or added:

- Add support to read gzipped Verilog files.
- Verilog parser reports meaningful error message for unterminated strings and comments.
- The contents of the Schem and Cone window can be saved as gif image.
- New userware example deleteBuses.tcl to delete all netBuses from a database.
- New userware example dumpIO.tcl to dump all I/O pins of all modules to a text file.
- New userware example readBookmark.tcl to restore a bookmark file specified on the command line.

RTLvision PRO 3.1.2

This is a maintenance release, the following features were fixed and/or added:

- Fix Application Error while selecting multiple objects in the Schem and Cone window with the selection rectangle.

RTLvision PRO 3.1.1

This is a maintenance release, the following features were fixed and/or added:

- The Verilog parser produces more debug output if "-info debug" is specified.

RTLvision PRO 3.1.0

This is a major release, the following features were fixed and/or added:

- New userware example floatingGate.tcl to check if there are floating gate pins at transistors.
- New database command $db oper changecellref to change the referenced cell of an instance.
- New database command $db oper rename to change object names.
- The list of SDF files moved from the Verilog dialog to its own dialog accessible from the toolbar

and main menu.

- sdf: support hierarchical paths in INTERCONNECT.
- The Cone tab of the Cone Extraction dialog accepts instances as start point.
- Add an example in (demo/api/zdb) how to use the C-Level API provided as a separate package zdb_api-<version>.tgz.
- Add a highlight color menu to the toolbar for fast access to all possible highlight colors.
- New database command $db identicalInterface to compare if the interface of two given Cells is identical.
- The Clear command of the Search window also clear the search string.
- Add support to drag objects, tooltips and a context menu and to the Clock Domain Analyzer window.
- Add support to save the Clock Domain Analyzer results as a text file, respectively postscript.
- In the rare case that all clock pins are undriven, the Clock Domain Analyzer hung in an endless loop.
- symlibexport.il: better handling for special Tcl characters like newline and curly braces.

RTLvision PRO 3.0.6

This is a maintenance release, the following features were fixed and/or added:

- Verilog preprocessor no longer checkout an extra gv-gate2zdb license on Windows.
- SDF reader: fixed portdirupdate for INTERCONNECT ports.
- New database command $db write to save the contents of a database as Tcl "$db load" commands.
- Remove API example demo/api/oem/savedb.tcl

RTLvision PRO 3.0.5

This is a maintenance release, the following features were fixed and/or added:

- SDF reader: several changes to improve stability and fix crashes.

RTLvision PRO 3.0.4

This is a maintenance release, the following features were fixed and/or added:

RTLvision PRO 3.0.3

This is a maintenance release, the following features were fixed and/or added:

- Error messages in the Console window are mouse sensitive. A click on a message jumps to the corresponding file and line.

- New GUI API function Gui:RemoveRegisteredDataBaseChanged.
- Replace the "Clear Schematic Cache" button in the Preferences dialog by a "Regenerate" button with the same effect.
- API extended: (1) the flag clear command without argument clears all flags; (2) new report netCount command; (3) new widthOf command; (4) new foreach oPort and foreach oPin to loop over ports and portBuses in Verilog declaration order; (5) new libcell flag; (6) new foreach cell to loop over modules and primitives; (7) shorter command name primFuncOf.

RTLvision PRO 3.0.2

This is a maintenance release, the following features were fixed and/or added:

- Speedup and less consumption of temp memory on Cone Extraction and Clock Domain Analyzer.
- Add progress bar and Interrupt button to Cone Extraction and Clock Domain Analyzer dialog windows.
- New command line option -info to specify the info level of the Rtl, Verilog and Edif parser.
- New command line option -sdf to specify a SDF file for a Verilog design.

RTLvision PRO 3.0.1

This is a maintenance release, the following features were fixed and/or added:

- Command line options simplified: RTLvision PRO option -verilog2001 is default (no need to specify it); GateVision option -verilog is default (no need to specify it).
- Verilog Netlist Parser (verilog2zdb) now supports some 2001 features, including (a) module header with inline port and parameter declaration, (b) the `elsif directive, (c) extra net-type information ("wire", "reg", etc.) in the input/output port declaration (d) extended grammar to gracefully ignore unsupported RTL-level elements.
- Speedup Clock Domain Analyzer.
- Clock Domain Analyzer displayed wrong highlight colors.
- Improve Progress Bar: in most cases, don’t jump back to 0 if multiple actions are processed in series.
- Improve printing: In the GUI dialog you can select Landscape or Portrait as orientation. Add two new print views: Fullfit and Visible.

RTLvision PRO 3.0.0

This is a major release, the following features were fixed and/or added:

- First Final Release of RTLvision PRO.