GTKWave 3.0 Wave Analyzer User's Guide

User's Guide

GTKWave

Created April 27, 2006
This manual supports GTKWave 3.0 and higher versions.

Copyright (c)1998-2006 BSI
GTKWave is free software. See http://www.gnu.org for more information on the GNU GPL General Public License. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

The information in this document is subject to change without notice.

Contents

Using This Manual **9**
Printing Conventions 9

Compiling and Installing GTKWave **11**
Unix and Linux Operating Systems 11
Microsoft Windows Operating Systems 12

Introduction **15**
GTKWave Overview 15
Why Use GTKWave? 16
What Is GTKWave? 18

GTKWave User Interface **19**
GTKWave 19
Main Window 19
Signal Subwindow 21
Wave Subwindow 23
Navigation and Status Panel 24
Menu Bar 25
TwinWave 26
RTLBrowse 27
Ergonomic Extras 30
Scroll Wheels 30
The Primary Marker 30
Interactive VCD 30

GTKWave Menu Functions **31**
File 31
Edit 32
Search 36
Time 37
Markers 38

View 39
Help 40

Quick Start 41
Sample Design 41
Launching GTKWave 42
Displaying Waveforms 44
Signal Search 44
Hierarchy Search 45
Tree Search 45
Signal Save Files 46
Pattern Search 46
Alias Files and Attaching External Disassemblers 47
Debugging the Source Code 49

Appendix A: Command Line Options Reference 51
gtkwave 51
twinwave 54
lxt2miner 55
lxt2vcd 56
mvl2lxt 57
mvl2vcd 57
rtlbrowse 58
tex2vcd 59
tla2vcd 59
vcd2lxt 60
vcd2lxt2 61
vcd2vzt 63
vertex 64
vzt2vcd 65
vztminer 66
shmidcat 67

Appendix B: .gtkwaverc Variable Reference 69

Appendix C: VCD Recoding 77
VList Recoding Strategy 77
Time Encoding 78
Single-bit Encoding 78
Multi-bit Encoding 79
Reals and String Encoding 80
Final Notes on VCD Recoding 80

Appendix D: LXT File Format **82**
LXT Framing 82
LXT Section Pointers 83
LXT Section Definitions 85
The lxt_write API 94

Index **97**
Illustration Index 97
Alphabetical Index 97

Using This Manual

Printing Conventions

Text printed in the font `courier` reflects messages that will be seen on screen at a command prompt or as program output.

Text printed in **`courier bold`** is to be entered by the user.

Text printed in `smaller monospace` is help available either as a manual page or as a program help option.

Text printed in *italics* is a pathname in the file system or is the name of an application program.

Compiling and Installing GTKWave

Unix and Linux Operating Systems

Compiling GTKWave on Unix or Linux operating systems should be a relatively straightforward process as GTKWave was developed under both Linux and AIX. External software packages required are any version of GTK (http://www.gtk.org) greater than or equal to 1.2, and *gperf* (for RTLBrowse) which can be downloaded from the GNU website (http://www.gnu.org). The compression libraries libz (zlib) and libbz2 (bzip2) are not required to be installed on a target system as their source code is already included in the GTKWave tarball. Likewise goes for PCCTS-1.33 source code (required for compiling *vertex*) which is included in the *contrib/* directory. Note that PCCTS-2.0 will not work as the grammar file format for v2.0 is incompatible with v1.33.

Compiling and Installing

Un-tar the source code into any temporary directory then change directory into it. After doing this, invoke the configure script. Note that if you wish to change the install point, use the doubledash `–prefix` option to point to the absolute pathname. For example, to install in /usr, type **`./configure –prefix=/usr`**.

```
1 :/tmp/gtkwave-3.0.0> ./configure
Build GTKWave for GTK+-1.x or 2.x? Please enter '1' or '2'
```

Enter '1' or '2' depending on which version of GTKWave you would like to compile against. Typically, you will enter '2' (or just press enter).

```
2 :/tmp/gtkwave-3.0.0> make
```

Wait for the compile to finish. This will take some amount of time. Then log on as the superuser.

```
3 :/tmp/gtkwave-3.0.0> su
```

```
Password:
[root@localhost gtkwave-3.0.0]# make install
```

Wait for the install to finish. It should proceed relatively quickly. When finished, exit as superuser.

```
[root@localhost gtkwave-3.0.0]# exit
exit
```

GTKWave is now installed on your Unix or Linux system. To use it, make sure that the *bin/* directory off the install point is in your path. For example, if the install point is */usr/local*, ensure that */usr/local/bin* is in your path. How to do this will vary from shell to shell.

Microsoft Windows Operating Systems

Cygwin

The best way to run GTKWave under Windows is to compile it as an application to run under Cygwin. This will provide the same functionality as compared to the Unix/Linux version and better graphical performance than the native binary version. Follow the directions for Unix compiles in the preceding section. Note that launching RTLBrowse requires Cygserver to be enabled. Please see the Cygwin documentation for information on how to enable Cygserver for your version of Cygwin.

MinGW versus VC++ for Native Binaries

It is recommended that Windows compiles and installs are done in the MinGW environment in order to mimic the Unix shell environment as well as produce binaries that are natively usable on Windows. Producing native binaries with VisualC++ has not been attempted for some time so it is currently untested and might not be supported. Proceed to use VisualC++ at your own risk.

MinGW with GTK-1.2

If you are missing a working version of *gtk-config*, you will need a fake *gtk-config* file in order to compile under GTK-1.2. It will look like this with the include and linker search directories modified accordingly:

```
#!/bin/sh
```

```
if [ "$1" == "--libs" ]
        then
        echo -L/home/bybell/libs -lgck -lgdk-1.3 -lgimp-1.2 -lgimpi -lgimpui-1.2
-lglib-1.3 -lgmodule-1.3 -lgnu-intl -lgobject-1.3 -lgthr
ead-1.3 -lgtk-1.3 -liconv-1.3 -ljpeg -llibgplugin_a -llibgplugin_b -lpng
-lpthread32 -ltiff-lzw -ltiff-nolzw -ltiff
        fi

if [ "$1" == "--cflags" ]
        then
        echo " -mms-bitfields -I/home/bybell/src/glib -I/home/bybell/src/gtk+/gtk
-I/home/bybell/src/gtk+/gdk -I/home/bybell/src/gtk+ "
        fi
```

Compiling as under Unix/Linux is the same however none of the helper applications will be compiled. The .exe file can be found in the *bin/* directory. Copy it to wherever you store binaries on your system. (n.b., **`make install`** does not work for MinGW.)

MinGW with GTK-2.0

You do not need to do anything special except ensure that *pkg-config* is pointed to by your PATH environment variable. Proceed as with GTK-1.2.

Introduction

GTKWave Overview

GTKWave is an analysis tool used to perform debugging on Verilog or VHDL simulation models. With the exception of interactive VCD viewing, it is not intended to be run interactively with simulation, but instead relies on a post-mortem approach through the use of dumpfiles. Various dumpfile formats are supported:

- VCD: Value Change Dump. This is an industry standard file format generated by most Verilog simulators and is specified in IEEE-1364. This is the slowest of the formats for the viewer to process and requires the most memory, however the format is ubiquitous and almost all tools support it, which is why native support remains. Note that recent versions of the viewer default to dynamic VCD recoding in memory through some interesting tricks with zlib compressed VLists. (See Appendix C: VCD Recoding on page 77.) This greatly reduces the amount of memory required to store a large, full (non-interactive) VCD trace in memory such that in many cases, less memory is required than the actual size of the trace itself. Nevertheless, using one of the database formats will almost always be more efficient for larger traces, especially if they are to be viewed repeatedly. (i.e., the speed hit for converting a trace to a database format is offset by the repeated cost of recoding VCD every time the trace is viewed.) The more physical memory that is available on a machine being used to view VCD, the better.
- LXT: InterLaced eXtensible Trace. This is an optimized format utilizing interleaved back pointers and value changes. Processing LXT files is faster than VCD. It was created specifically for use with GTKWave, however some other simulators (notably, Icarus Verilog) support it natively.
- LXT2: InterLaced eXtensible Trace Version 2. This is a block-based variant of LXT that allows for greater compression and access speeds than can be achieved with LXT. It allows random-access at the block level and also optionally allows partial loading of blocks for even faster operation. Icarus

Verilog also supports LXT2 natively.

- VZT: Verilog Zipped Trace. This is an outgrowth of LXT2 as it is also block based, however it employs a different heuristic for compression that allows for file sizes much smaller than most other dumpfile formats including commercial ones. VZT file write performance is the slowest of all the formats, however reading them can be extremely fast on multiprocessor machines as the file format has been designed such that the reader was able to be parallelized.
- GHW: GHDL Wavefile. This is a nine state (“01XZHUWL-”) file format written by the VHDL simulator GHDL.
- AET2: All Events Trace Version 2. This is format used by various IBM EDA tools. File size is very small and access is extremely fast. Support for it is determined at compile time. If the AET2 reader API libraries are not found, it is disabled. Users of IBM toolsets can modify the *./configure3* script to point to the *libae2rw.a* and/or *libae2rw.so* files in order to enable this feature.

Converter helper applications are packaged with the viewer in order to convert VCD files into LXT, LXT2, or VZT files. Conversion from LXT2 and VZT back into VCD is possible. Wholesale conversion from LXT is not currently possible, however it is possible to save the traces visible in the main GTKWave window as VCD so conversion to LXT is not strictly irreversible.

Why Use GTKWave?

GTKWave has been developed to perform debug tasks on large systems on a chip and has been used in this capacity as an offline replacement for third-party debug tools. It is 64-bit clean and is ready for the largest of designs given that it is run on a workstation with a sufficient amount of physical memory. The file formats LXT2 and VZT have been specifically designed to cope with large, real-world designs and AET2 (available to IBM EDA tool users only) has been designed to handle extremely large designs efficiently.

For Verilog, GTKWave allows users to debug simulation results at both the net level by providing a bird's eye view of multiple signal values over varying periods of time and also at the RTL level through annotation of signal values back into the RTL for a given timestep. The RTL browser frees up users from needing to be concerned with the actual location of where a given module resides in the RTL as the view provided by the RTL browser defaults to the module level. This provides quick access to modules in the RTL as navigation has been reduced simply to moving up and down in a tree structure that represents the actual design.

Source code annotation is currently not available for VHDL, however all of GTKWave's other debug features are readily accessible. VHDL support is planned for a future release.

What Is GTKWave?

GTKWave as a collection of binaries is comprised of two interlocking tools: the *gtkwave* viewer application and *rtlbrowse*. In addition, a collection of helper applications are used to facilitate such tasks as file conversions and simulation data mining. They are intended to function together in a cohesive system although their modular design allows each to function independently of the others if need be.

gtkwave is the waveform analyzer and is the primary tool used for visualization. It provides a method for viewing simulation results for both analog and digital data, allows for various search operations and temporal manipulations, can save partial results (i.e., “signals of interest”) extracted from a full simulation dump, and finally can generate PostScript and FrameMaker output for hard copy.

rtlbrowse is used to view and navigate through RTL source code that has been parsed and processed into a stems file by the helper application *vertex*. It allows for viewing of RTL at both the file and module level and when invoked by *gtkwave*, allows for source code annotation.

The helper applications perform various specialized tasks such as file conversion, RTL parsing, and other data manipulation operations considered outside of the scope of what a visualization tool needs to perform.

GTKWave User Interface

GTKWave

Main Window

The GTKWave visualization tool main window is comprised of a menu bar section, a status window, several groups of buttons, a time status section, and signal and wave value sections. New with GTKWave 3.0 is the inclusion of an embedded Signal Search Tree (SST) expander to the left of the signal section. The viewer typically appears as below when the embedded SST is disabled.

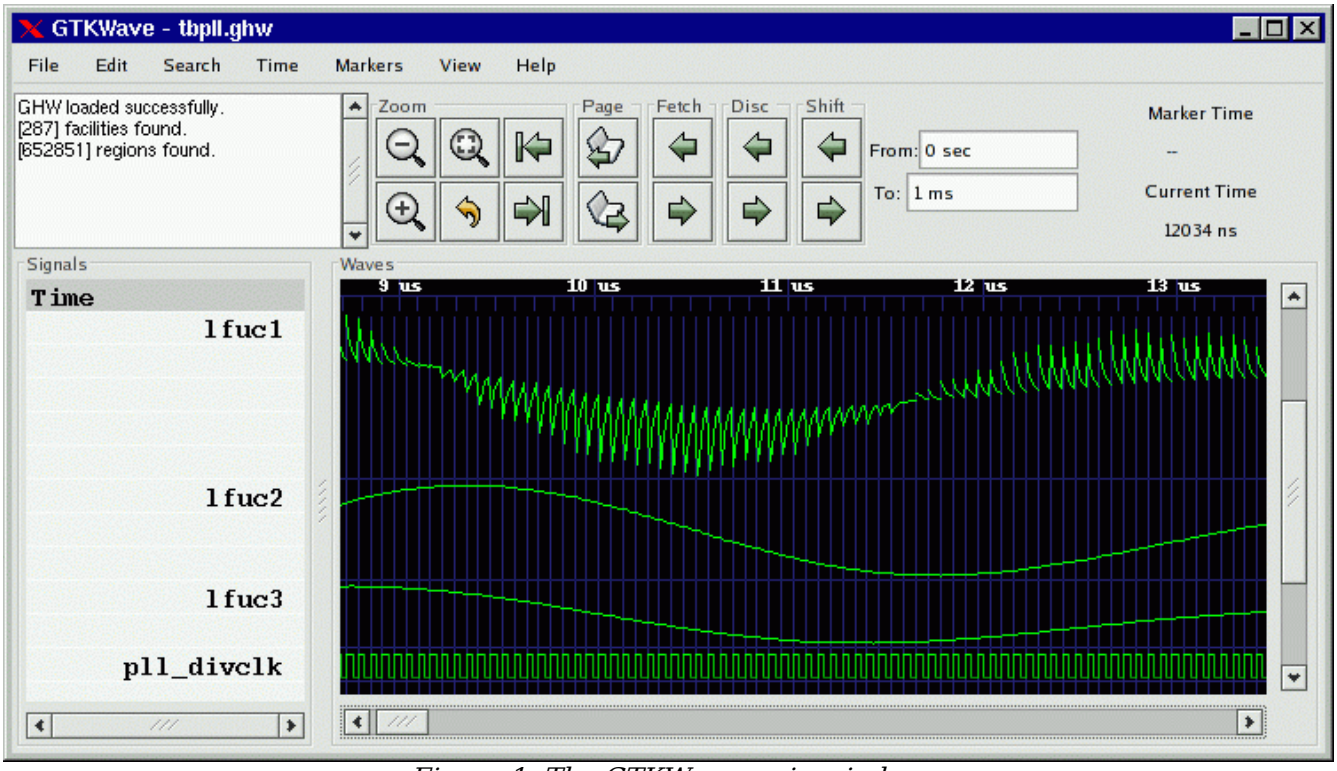

Figure 1: The GTKWave main window

To the extreme left in a frame marked “Signals” is the signal section. Signal names can be left or right aligned (left aligned being useful for detection of hierarchy differences) and the number of levels of hierarchy (as counting from the rightmost side of a signal name) displayed can be set by the user.

To the right of the signal section is the wave section in a frame marked “Waves”. The top line is used as a timescale and all other lines are used to render trace value data with respect the the timescale. The vertical blue lines in the trace value data section are not normally present. In this case they are the result of keying on the rising edge of the digital signal “pll_divclk”. Analog traces of varying heights can be seen as well. Analog traces can dynamically be made as tall or short as desired in order to make the viewing of them easier, however the size is limited to integer multiples of the height of one digital trace.

With GTK versions greater than or equal to 2.4, an embedded SST is available. Drag and Drop of signals from the “Signals” pane inside the SST into the “Signals” pane outside of the SST is a convenient way to import signals into the viewer.

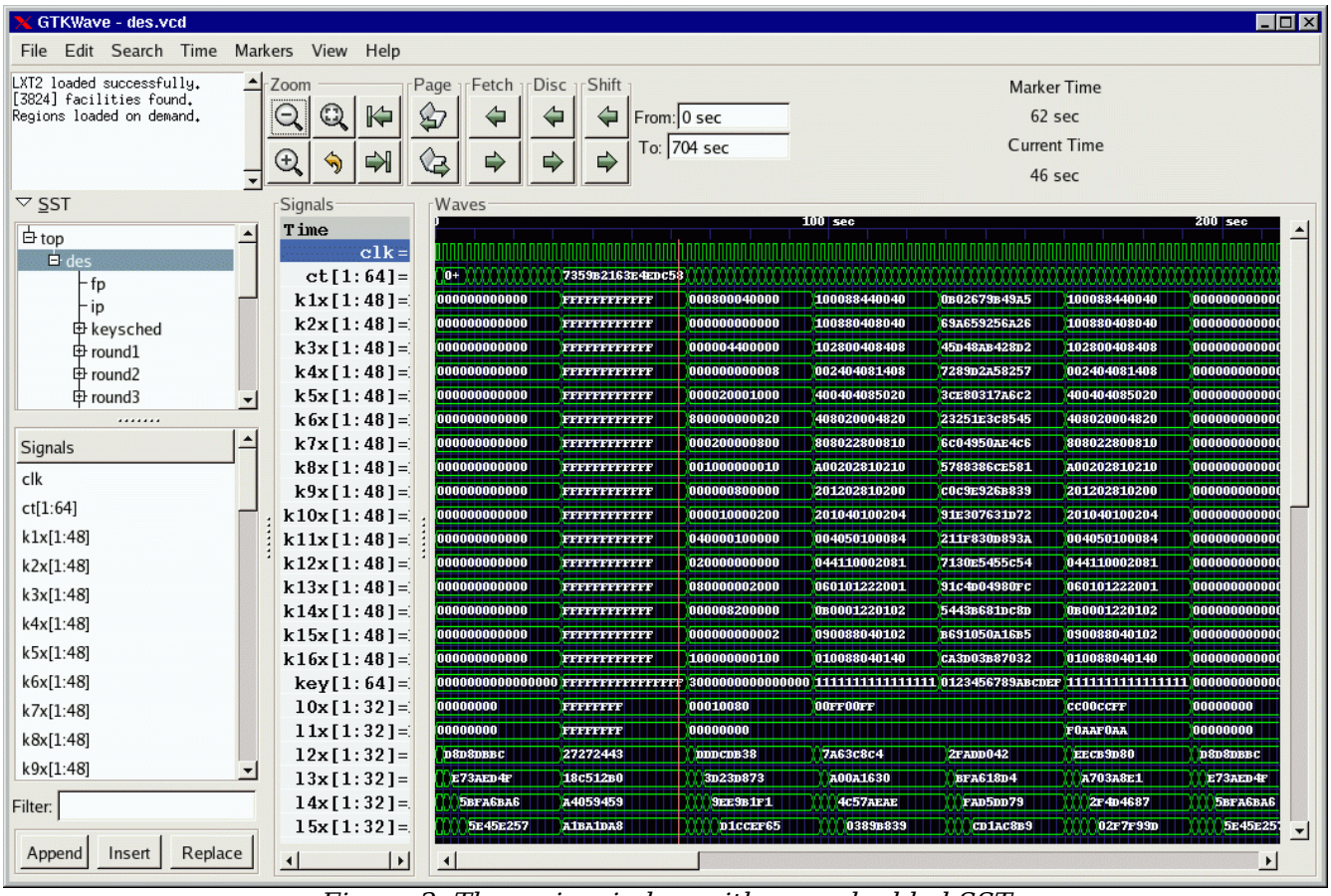

Figure 2: The main window with an embedded SST

The main window size and position can be saved between sessions as well as the current viewer state. (i.e., which signals are visible, any attributes set for those

signals such as alignment and inversion, where the markers are set, and what pattern marking is active.)

Signal Subwindow

The signal subwindow is nothing more than a list of signals, optional comments, and optional blank lines. The following is a sample view of the signal subwindow showing a highlighted trace ("clk") and a comment trace (in blue with asterisks, "*** Non-clock traces ***". In between the two is a blank trace inserted by the user. Note that the highlighting of a trace can be toggled by clicking the left mouse button on an entry in the signal subwindow.

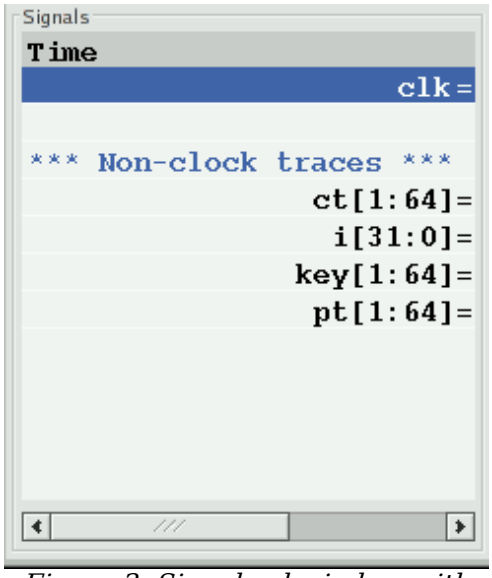

Figure 3: Signal subwindow with scrollbar and an "open" collapsible trace

Groups of traces that start with a comment trace and end with a blank line are considered to be part of a collapsible group. Collapsible groups can be toggled open or closed by holding down the control key and then clicking on the comment trace. When a group is collapsed, the comment trace will be grayed out and the actual traces for that group will be hidden.

You will notice that the scrollbar along the bottom of the subwindow in Figure 3 indicates that there is a hidden section to the right. This hidden area contains the values of the signals shown. The scrollbar can be manually moved to show this area or the pane to the right of the signal subwindow can be enlarged in order to allow full viewing of the subwindow.

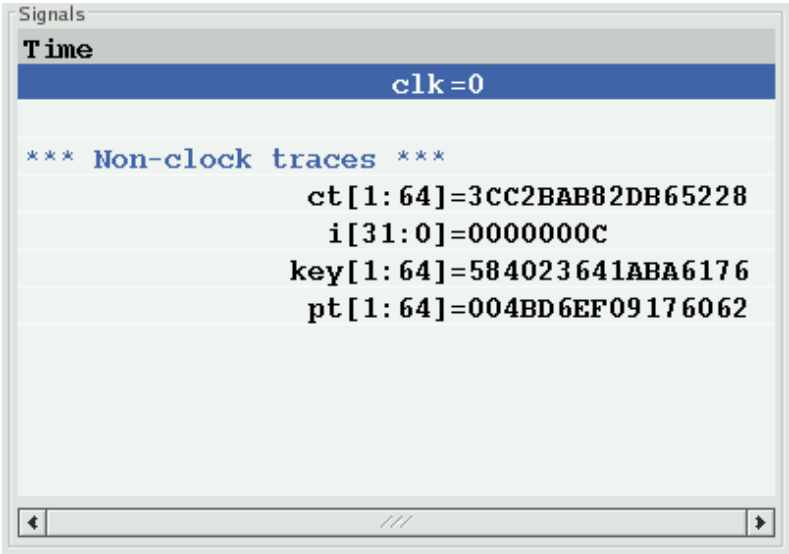

Figure 4: Signal subwindow with no hidden area from left to right

Expanding the size of the subwindow by increasing the width of the pane is illustrated in Figure 4. No area is hidden as reflected by the scrollbar which is completely filled in from left to right along its length. In addition, the signal values which are present can be read. Any time the primary marker is nailed down, there will be an equals ("=") sign

indicating that signal values are present.

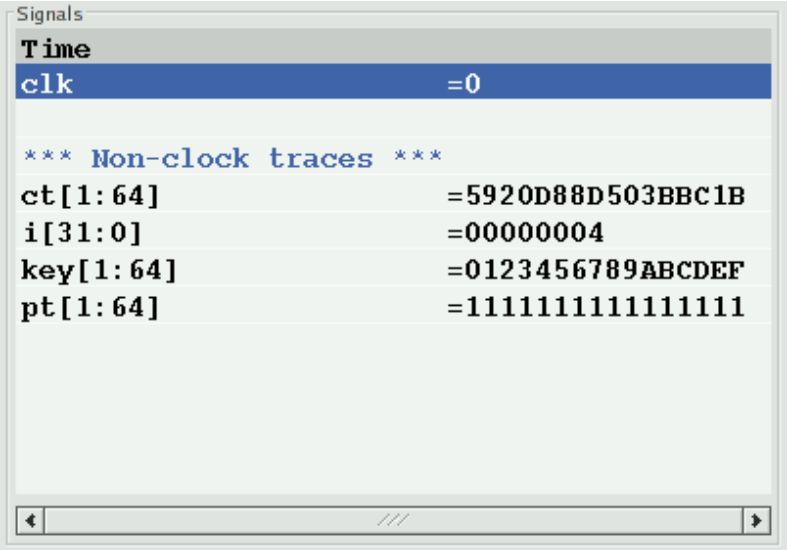

Figure 5: Signal subwindow with left justified signal names

As seen in both Figure 3 and Figure 4, the signal names are right justified and are flush against the equals signs. This is only a matter of personal preference, and if desired, as shown in Figure 5, the signals can be left justified against the left margin of the signal subwindow by pressing the key combination of Shift-Home. This is useful when looking at signals if one is attempting to determine where hierarchies for different net names differ. Press Shift-End to right justify the signal names. (Right justification is the default behavior). Regardless of the state of signal name justification, the signal values are left justified against the equals sign and cannot be moved.

Note that the signal subwindow supports a form of self-contained Drag and Drop such that the right mouse button can be used to harvest all the highlighted traces in the window. By holding the right button and moving the mouse up and down, a destination for the traces can be selected. When the mouse button is released, the traces are dropped at the trace following the one the mouse pointer is pointing to.

Multiple traces can be selected by marking the first trace to highlight, move the cursor to the destination trace, and Shift click with the left mouse button. All the traces between the two will highlight or unhighlight accordingly. To highlight all the traces in the signal subwindow, Alt-H can be pressed. To unhighlight them, also press the Shift key in conjunction with Alt-H. (This can also be achieved by clocking on Highlight All or Unhighlight All in the Edit menu.)

Highlighting or unhighlighting traces by entering regular expressions will be covered in the menu section.

Wave Subwindow

The wave subwindow reformats simulation data into a visual format similar to that seen for digital storage scopes. As seen in Figure 6, the wave subwindow contains two scrollbars and a viewing area.

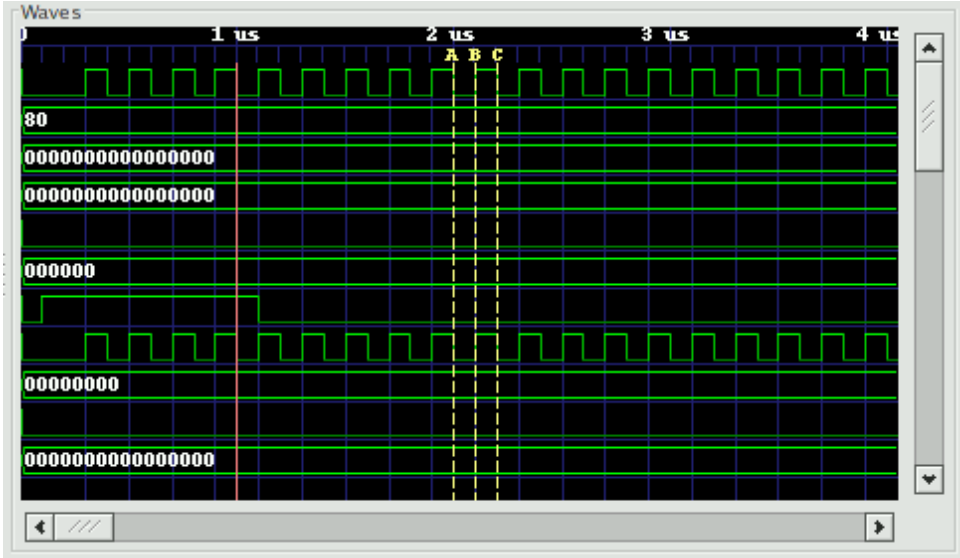

Figure 6: A typical view of the wave subwindow

The scrollbar on the right scrolls not only the wave subwindow, but the signal subwindow in lockstep as well. The scrollbar on the bottom is used to scroll the simulation data with respect to the timescale that is shown on the top line of the wave subwindow.

The simulation data itself is shown as a horizontal series of traces. Values for multi-bit signals can be displayed in varying numeric bases such as binary, octal, hexadecimal, decimal, and ASCII. Values for single-bit traces are shown as “high” for zero and “low” for one, “z” (middle), and “x” (filled-in box). VHDL values are represented in a similar fashion but with different colors. The signal subwindow can always be used to verify the value of a value, so don't be too concerned right now if you are not sure of what the single-bit representation of a signal looks like or are not sure if you can remember.

Two functional markers are available: the primary marker (red, left mouse button) which the signal window uses as its pointer for value data, and the baseline marker (white, middle mouse button) which is used to perform time measurements with respect to the primary marker. Twenty-six lettered markers “A” through “Z” (dropped or collected through menu options) are provided to the user as convenience markers for indexing various points of interest in a simulation.

The primary marker can also be used to navigate with respect to time. It can be dropped with the right mouse button and dragged to “open” up a region for zooming in closer or out farther in time. It can also be used to scroll by holding down the left mouse button and dragging the mouse outside the signal subwindow. The simulation data outside of the window will then scroll into view with the scrolling being in the opposite direction that the primary marker is

“pulling” outside of the subwindow.

Trace data in the signal subwindow can also be timeshifted as shown in Figure 7. In order to timeshift a trace, highlight the trace in the signal window the move over to the wave subwindow and then hold down the left mouse button in order to set the primary marker. Press the Ctrl key then move the primary marker left or right. When the timeshift is as desired, release the mouse button then release Ctrl. If you do not wish to go through with the timeshift, release the Ctrl key before releasing the left mouse button. The trace(s) will then spring back to their original pre-shifted position.

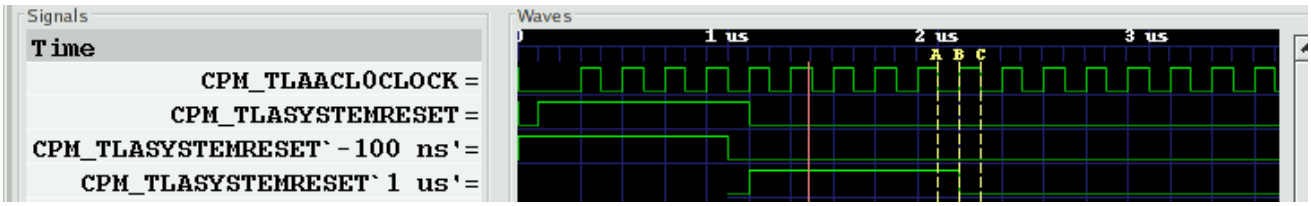

Figure 7: An example of both positively and negatively timeshifted traces

To achieve a finer level of granularity for timeshifting, menu options are available that allow the user to set specific values for a time shift. In this way, the pixel resolution of zoom is not the limiting factor in achieving an “exact” shift that suits a user's needs.

Navigation and Status Panel

The navigation and status panel occupies the top part of the main window just below the menu bar.

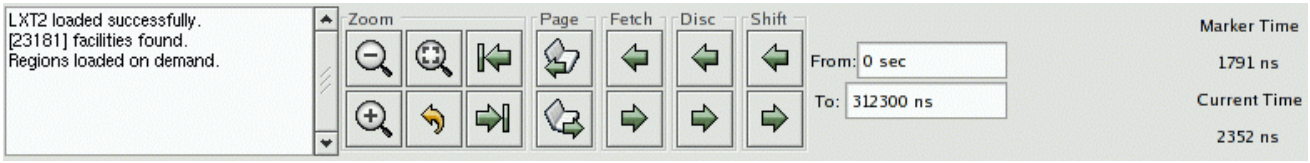

Figure 8: The Navigation and Status Panel

The leftmost part contains a status window used for displaying various relevant messages to the user such as the dumpfile type, the number of facilities (nets) in a dumpfile, and any other information such as an operation that fails or completes successfully.

The Zoom subframe contains six buttons. Three are magnifying glass icons. The one marked with a minus (“-”) zooms out which displays a larger amount of simulation time. The one marked with a plus (“+”) zooms in closer, displaying less simulation time. The one with a square in it is “Zoom Full” which is used either to zoom out to display the full range of simulation time or zooms between the primary and baseline marker when the baseline marker is set. The

remaining non-magnifying glass buttons are a back arrow which is a zoom undo. The left arrow “zooms” to the start time of simulation and the right arrow zooms to the end time. The left and right arrows do not affect the zoom level in or out like the plus and minus buttons do; they simply are a shortcut to keep from having to move the scrollbar at the bottom of the wave subwindow.

The Page subframe contains left and right arrows. It scrolls the wave window left or right the granularity of one page. It is similar to clicking to the left or right of the “visible” gadget in a scrollbar, however, given the limited resolution of the GTK scrollbar (floating point), for simulations that have large time values, it might be necessary to use the page buttons rather than the scrollbar.

The Shift subframe contains similar arrows that scroll the display one pixel or timestep (depending on what the zoom level is).

The “From” and “To” boxes indicate the start and end times for what part of the simulation run shall be visible and can be navigated inside the wave subwindow. Values can directly be entered into these boxes and units (e.g., ns, ps, fs) can also be affixed to values.

The Fetch and Discard subframes modify the “From” and “To” box times. Clicking the left Fetch arrow decreases the “From” value. Clicking the right Fetch arrow increases the “To” value. Clicking the left Discard value increases the “From” value and clicking the right Discard button decreases the “To” value.

The Marker Time label indicates where the primary marker is located. If it is not present, a double-dash (“--”) is displayed. The Current Time label indicates where the mouse is pointing. Its function is to determine the time under the cursor without having to activate or move the primary marker. Note that when the primary marker is being click-dragged, the Marker Time label will indicate the delta time off the initial marker click.

When the baseline marker is set, the Marker Time and Current Time labels change. The Marker time label indicates the delta time between the baseline marker and the primary marker. The Current Time label is replaced with a Base Time label that indicates the value of the baseline marker.

Menu Bar

There are seven submenus in the menu bar: File, Edit, Search, Time, Markers, View, and Help. The functions of the individual items in each of those submenus will be covered in GTKWave Menu Functions on page 31.

TwinWave

TwinWave is a front end to GTKWave that allows two sessions to be open at one time in a single window. The horizontal scrolling, zoom factor, primary marker, and secondary marker are synchronized between the two sessions.

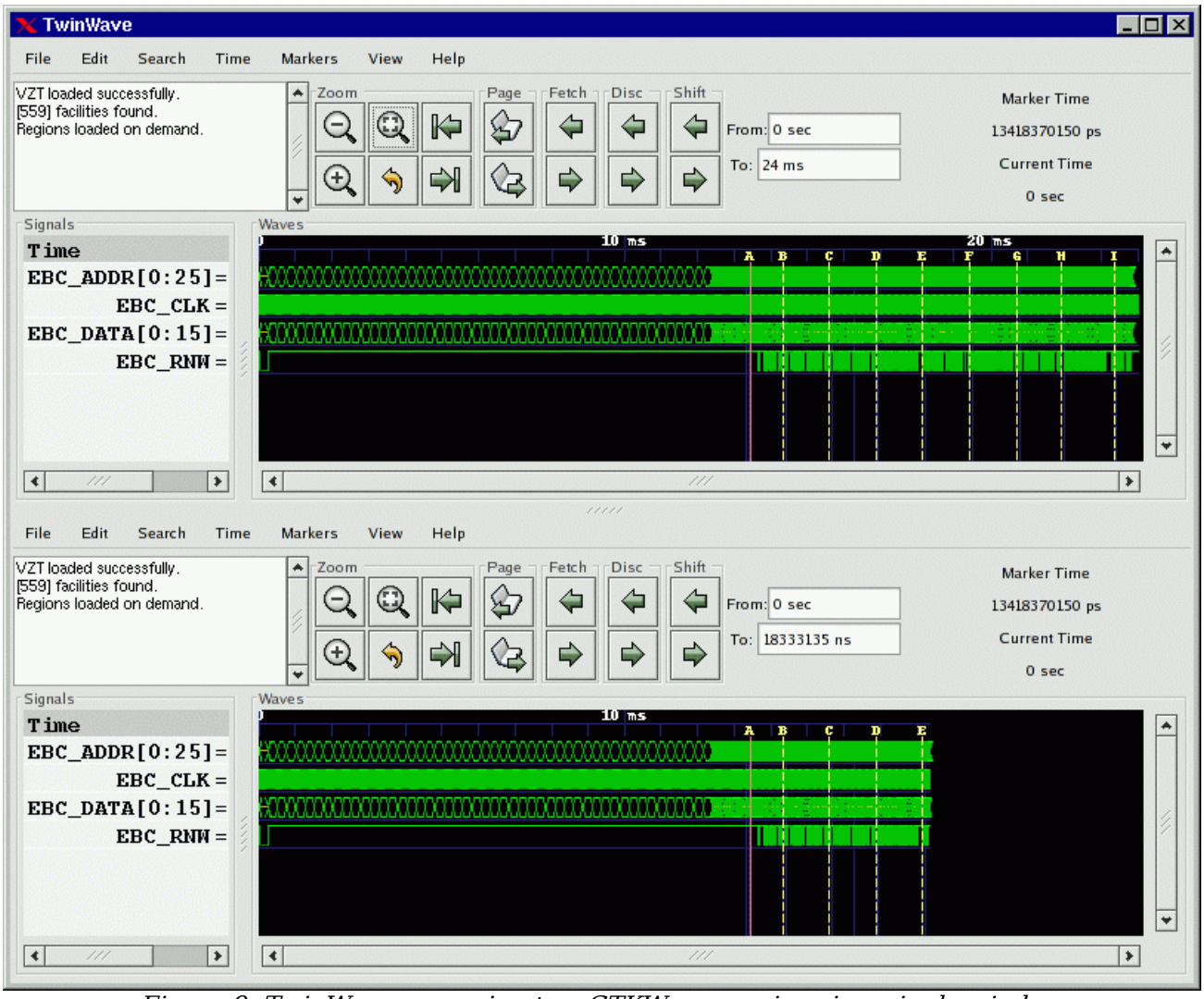

Figure 9: TwinWave managing two GTKWave sessions in a single window

Starting a TwinWave session is easy: simply invoke `twinwave` with the arguments for each gtkwave session listed fully separating them with a plus sign.

```
twinwave a.vcd a.sav + b.vcd b.sav
```

RTLBrowse

rtlbrowse is usually called as a helper application by *gtkwave*. In order to use RTLBrowse, Verilog source code must first be compiled with *vertex* in order to generate a stems file. A stems file contains hierarchy and component instantiation information used to navigate quickly through the source code. If GTKWave is started with the **`--stems`** option, the stems file is parsed and *rtlbrowse* is launched.

The main window for RTLBrowse depicts the design as a tree-like structure. (See Figure 10: The RTLBrowse RTL Design Hierarchy window on page 28.) Nodes in the tree may be clicked open or closed in order to navigate through the design hierarchy. Missing modules (unparsed, but instantiated as components) will be marked as “[MISSING]”.

When an item is selected, another window is opened showing only the source code the selected module. If the primary marker is set, then the source code will be annotated with values as shown in Figure 11: Source code annotated by RTLBrowse on page 29. The values shown are the full, wide value of the signal. RTLBrowse currently does not perform bit extractions on multi-bit vectors. If it is desired to see the full source code file for a module, click on the “View Full File” button at the bottom of the window.

Note that it is possible to descend deeper into the design hierarchy by selecting the component name in the annotated or unannotated source code.

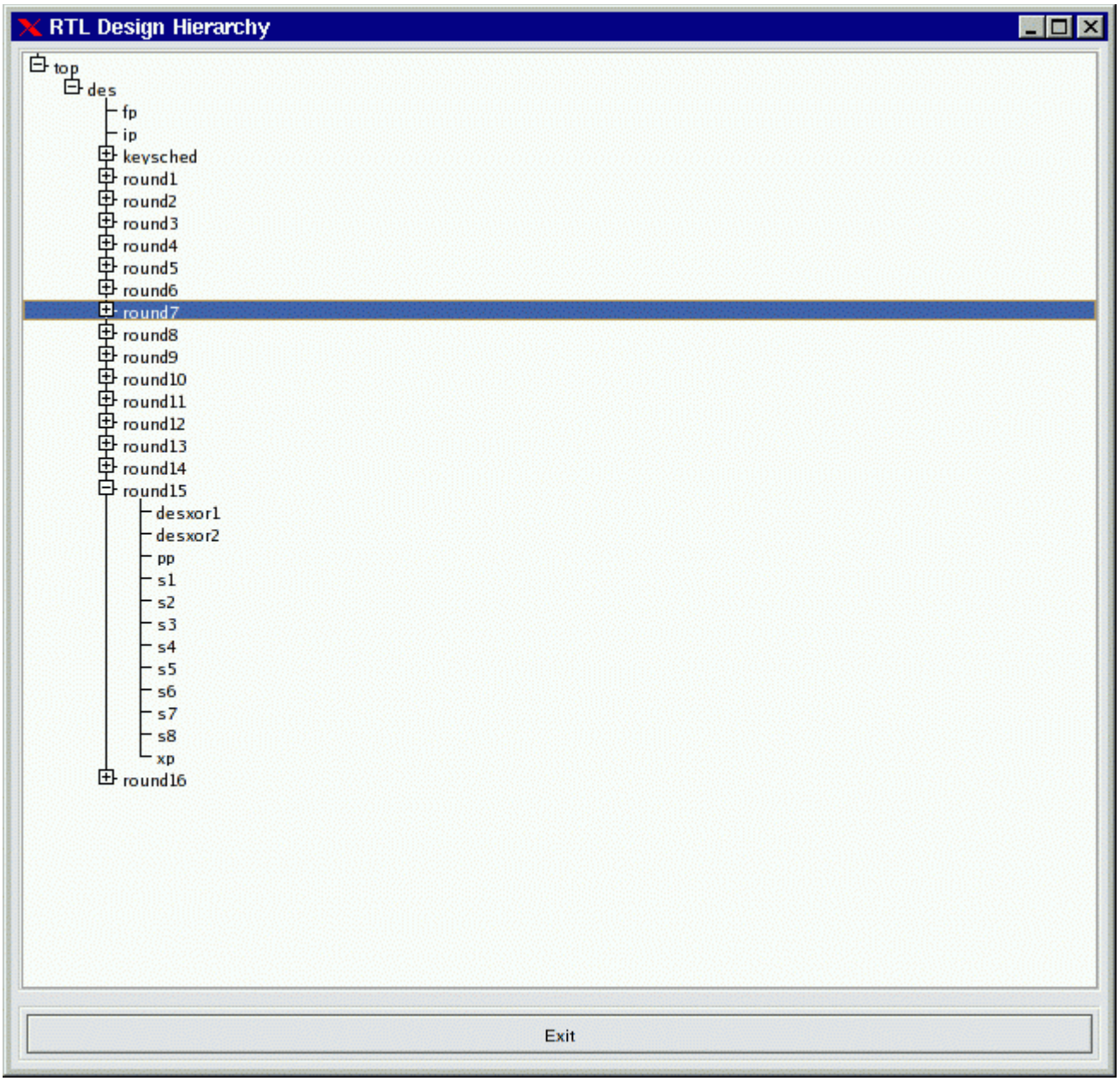

Figure 10: The RTLBrowse RTL Design Hierarchy window

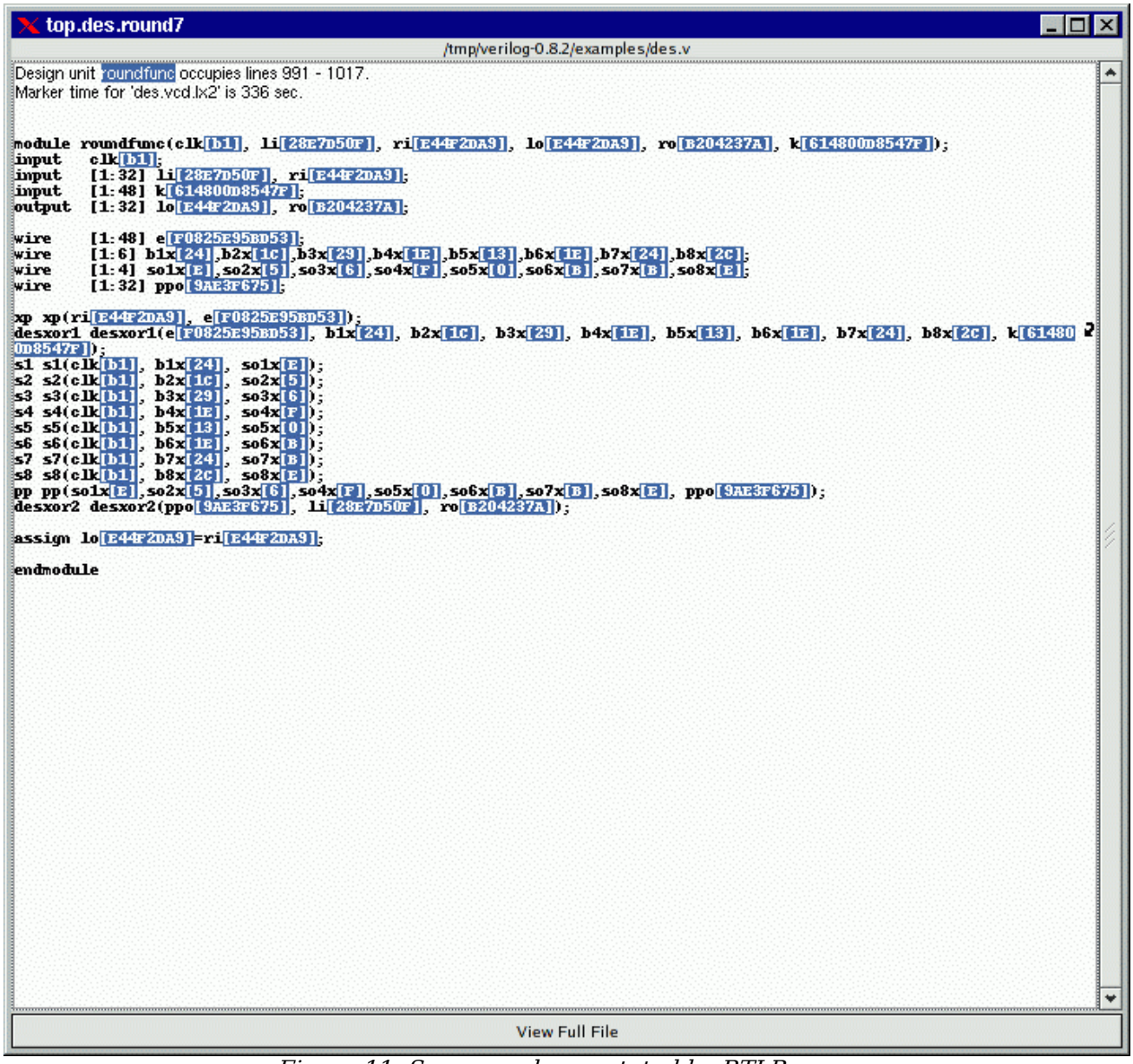

Figure 11: Source code annotated by RTLBrowse

Ergonomic Extras

Scroll Wheels

Users with a scroll wheel mouse have some extra one-handed operations options available which correspond to some functions found in the Navigation and Status Panel description on page 24.

- Shift Right – Ctrl + scroll wheel down
- Shift Left – Ctrl + scroll wheel up
- Page Right – scroll wheel down
- Page Left – scroll wheel up
- Zoom In – Alt + scroll wheel down
- Zoom Out – Alt + scroll wheel up

Turning the scroll wheel “presses” the shift, page, and zoom options repeatedly far faster than is possible with the navigation buttons. Zoom functions are especially smooth this way.

The Primary Marker

The primary marker has also had function overloaded onto it for user convenience. Besides being used as a marker, it can also be used to navigate with respect to time. It can be dropped with the right mouse button and dragged to “open” up a region for zooming in closer or out farther in time. It can also be used to scroll by holding down the left mouse button and dragging the mouse outside the signal subwindow. The simulation data outside of the window will then scroll into view with the scrolling being in the opposite direction that the primary marker is “pulling” outside of the subwindow.

Interactive VCD

VCD files may be viewed as they are generated provided that they are written to a fifo (pipe) and are trampolined through shmidcat first (assume the simulator will normally generate `outfile.vcd`):

```
mkfifo outfile.vcd
cver myverilog.v &
shmidcat outfile.vcd | gtkwave -v -I myverilog.sav
```

You can then navigate the file as simulation is running and watch it update.

GTKWave Menu Functions

File

The File submenu contains various items related to the accessing of files, printing, and application respawning and exiting.

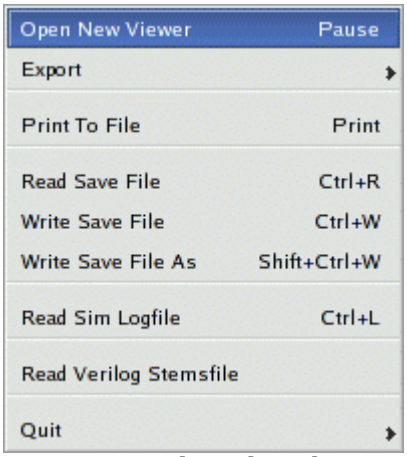

Figure 12: The File submenu

Open New Viewer will open a file requester that will ask for the name of a VCD or AET file to view. This will fork off a new viewer process.

Export-Write VCD File As will open a file requester that will ask for the name of a VCD dumpfile. The contents of the dumpfile generated will be the vcd representation of the traces onscreen that can be seen by manipulating the signal and wavewindow scrollbars. The data saved corresponds to the trace information needed to allow viewing when used in tandem with the corresponding GTKWave save file.

Export-Write LXT File As will open a file requester that will ask for the name of an LXT dumpfile. The contents of the dumpfile generated will be the vcd representation of the traces onscreen that can be seen by manipulating the signal and wavewindow scrollbars. The data saved corresponds to the trace information needed to allow viewing when used in tandem with the corresponding GTKWave save file.

Print To File will open up a requester that will allow you to select print options (PS or MIF; Letter, A4, or Legal; Full or Minimal). After selecting the options you want, a file requester will ask for the name of the output file to generate that reflects the current main window display's contents.

Read Save File will open a file requester that will ask for the name of a GTKWave save file. The contents of the save file will determine which traces and vectors as well as their format (binary, decimal, hex, reverse, etc.) are to be appended to the display. Note that the marker positional data and zoom factor present in the save file will replace any current settings.

Write Save File will invoke Write Save File As if no save file name has been specified previously. Otherwise it will write the save file data without prompting.

Write Save File As will open a file requester that will ask for the name of a GTKWave save file. The contents of the save file generated will be the traces as well as their format (binary, decimal, hex, reverse, etc.) which are currently a part of the display. Marker positional data and the zoom factor are also a part of the save file.

Read Logfile will open a file requester that will ask for the name of a plaintext simulation log. By clicking on the numbers in the logfile, the marker will jump to the appropriate time value in the wave window.

Read Verilog Stemsfile will open a file requester that will ask for the name of a Verilog stemsfile. This will then launch an RTL browser and allow sourcecode annotation based on the primary marker position. Stems files are generated by Vertex. Please see its manpage for syntax and more information on stems file generation.

Quit exits GTKWave after an additional confirmation requester is given the OK to quit.

Edit

The Edit submenu is used to perform sorts on net names, perform various utility functions such as attaching disassemblers and other external programs to GTKWave, and to change the data representation of values in the wave subwindow.

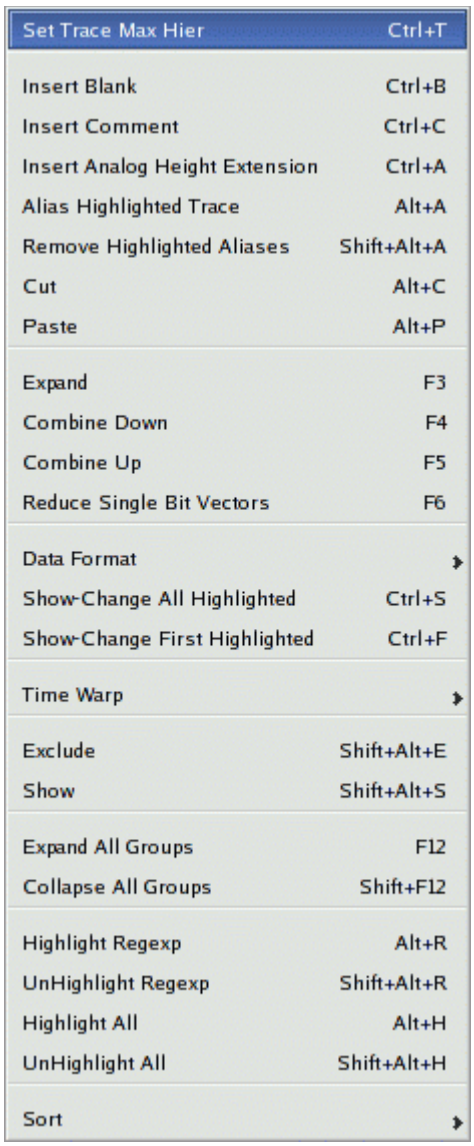

Figure 13: The Edit submenu

The Set Max Hier sets the maximum hierarchy depth (counting from the right with bit numbers or ranges ignored) that is displayable for trace names. Zero indicates that no truncation will be performed (default). Note that any aliased signals (prefix of a "+") will not have truncated names.

Insert Blank inserts a blank trace after the last highlighted trace. If no traces are highlighted, the blank is inserted after the last trace.

Insert Comment inserts a comment trace after the last highlighted trace. If no traces are highlighted, the comment is inserted after the last trace.

Insert Analog Height Extension inserts a blank

analog extension trace after the last highlighted trace. If no traces are highlighted, the blank is inserted after the last trace. This type of trace is used to increase the height of analog traces.

Alias Highlighted Trace only works when at least one trace has been highlighted. With this function, you will be prompted for an alias name for the first highlighted trace. After successfully aliasing a trace, the aliased trace will be unhighlighted. Single bits will be marked with a leading "+" and vectors will have no such designation. The purpose of this is to provide a fast method of determining which trace names are real and which ones are aliases.

Remove Highlighted Aliases only works when at least one trace has been highlighted. Any aliased traces will have their names restored to their original names. As vectors get their names from aliases, vector aliases will not be removed.

Cut removes highlighted signals from the display and places them in an offscreen cut buffer for later Paste operations. Cut implicitly destroys the previous contents of the cut buffer.

Paste pastes signals from an offscreen cut buffer and places them in a group after the last highlighted signal, or at the end of the display if no signal is highlighted. Paste implicitly destroys the previous contents of the cut buffer.

Expand decomposes the highlighted signals into their individual bits. The resulting bits are converted to traces and inserted after the last highlighted trace. The original unexpanded traces will be placed in the cut buffer. It will function seemingly randomly when used upon real valued single-bit traces. When used upon multi-bit vectors that contain real valued traces, those traces will expand to their normal "correct" values, not individual bits.

Combine Down coalesces the highlighted signals into a single vector named "<Vector>" in a top to bottom fashion placed after the last highlighted trace. The original traces will be placed in the cut buffer. It will function seemingly randomly when used upon real valued single-bit traces.

Combine Up coalesces the highlighted signals into a single vector named "<Vector>" in a bottom to top fashion placed after the last highlighted trace. The original traces will be placed in the cut buffer. It will function seemingly randomly when used upon real valued single-bit traces.

Reduce Single Bit Vectors decomposes the highlighted traces into their individual bits only if the highlighted traces are one bit wide vectors. In effect, this function allows single-bit vectors to be viewed as signals. The resulting bits are converted to traces and inserted after the last converted trace with the pre-conversion traces being placed in the cut buffer.

Data Format-Hex will step through all highlighted traces and ensure that vectors with this qualifier will be displayed with hexadecimal values.

Data Format-Decimal will step through all highlighted traces and ensure that vectors with this qualifier will be displayed with decimal values.

Data Format-Signed will step through all highlighted traces and ensure that vectors with this qualifier will be displayed as sign extended decimal values.

Data Format-Binary will step through all highlighted traces and ensure that vectors with this qualifier will be displayed with binary values.

Data Format-Octal will step through all highlighted traces and ensure that vectors with this qualifier will be displayed with octal values.

Data Format-ASCII will step through all highlighted traces and ensure that vectors with this qualifier will be displayed with ASCII values.

Data Format-BitsToReal will step through all highlighted traces and ensure that vectors with this qualifier will be displayed with Real values. Note that this only works for 64-bit values and that ones of other sizes will display as binary.

Data Format-Right Justify-On will step through all highlighted traces and ensure that vectors with this qualifier will be displayed right justified.

Data Format-Right Justify-Off will step through all highlighted traces and ensure that vectors with this qualifier will not be displayed right justified.

Data Format-Invert-On will step through all highlighted traces and ensure that bits and vectors with this qualifier will be displayed with 1's and 0's inverted.

Data Format-Invert-Off will step through all highlighted traces and ensure that bits and vectors with this qualifier will not be displayed with 1's and 0's inverted.

Data Format-Reverse Bits-On will step through all highlighted traces and ensure that vectors with this qualifier will be displayed in reversed bit order.

Data Format-Reverse Bits-Off will step through all highlighted traces and ensure that vectors with this qualifier will not be displayed in reversed bit order.

Translate Filter File Disable will remove translation filtering used to reconstruct enums for marked traces.

Translate Filter File will enable translation on marked traces using a filter file. A requester will appear to get the filter filename.

Translate Filter Process Disable will remove translation filtering used to reconstruct enums for marked traces.

Translate Filter Process will enable translation on marked traces using a filter process. A requester will appear to get the filter filename.

Analog Off causes the waveform data for all currently highlighted traces to be displayed as normal.

Analog Step causes the waveform data for all currently highlighted traces to be displayed as stepwise analog waveform.

Analog Interpolate causes the waveform data for all currently highlighted traces to be displayed as interpolated analog waveform.

Show-Change All Highlighted provides an easy means of changing trace attributes en

masse. Various functions are provided in a Show-Change requester.

Show-Change First Highlighted provides a means of changing trace attributes for the first highlighted trace. Various functions are provided in a Show-Change requester. When a function is applied, the trace will be unhighlighted.

Warp Marked offsets all highlighted traces by the amount of time entered in the requester. (Positive values will shift traces to the right.) Attempting to shift greater than the absolute value of total simulation time will cap the shift magnitude at the length of simulation. Note that you can also warp traces dynamically by holding down CTRL and dragging a group of highlighted traces to the left or right with the left mouse button pressed. When you release the mouse button, if CTRL is pressed, the drag warp commits, else it reverts to its pre-drag condition.

Unwarp Marked removes all offsets on all highlighted traces.

Unwarp All unconditionally removes all offsets on all traces.

Exclude causes the waveform data for all currently highlighted traces to be blanked out.

Show causes the waveform data for all currently highlighted traces to be displayed as normal if the exclude attribute is currently set on the highlighted traces.

Expand All Groups causes all groups defined by comment traces to expand. Groups are manually toggled collapsed and uncollapsed individually by holding down CTRL and pressing the left mouse button on a comment trace.

Collapse All Groups causes all groups defined by comment traces to collapse. Groups are manually toggled collapsed and uncollapsed individually by holding down CTRL and pressing the left mouse button on a comment trace.

Highlight Regexp brings up a text requester that will ask for a regular expression that may contain text with POSIX regular expressions. All traces meeting this criteria will be highlighted.

UnHighlight Regexp brings up a text requester that will ask for a regular expression that may contain text with POSIX regular expressions. All traces meeting this criteria will be unhighlighted if they are currently highlighted.

Highlight All simply highlights all displayed traces.

UnHighlight All simply unhighlights all displayed traces.

Alphabetize All alphabetizes all displayed traces. Blank traces are sorted to the bottom.

Alphabetize All (CaseIns) alphabetizes all displayed traces without regard to case. Blank traces are sorted to the bottom.

Sigsort All sorts all displayed traces with the numeric parts being taken into account. Blank traces are sorted to the bottom.

Reverse All reverses all displayed traces unconditionally.

Search

The Search submenu is used to perform searches on net names and values.

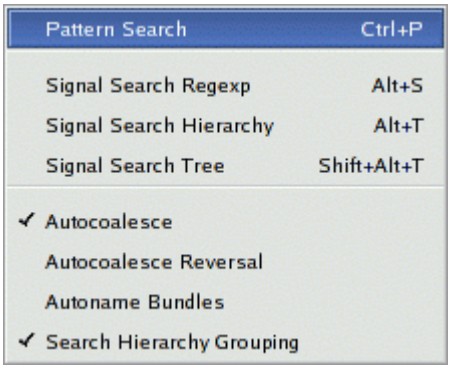

Figure 14: The Search submenu

Pattern Search only works when at least one trace is highlighted. A requester will appear that lists all the selected traces (maximum of 500) and allows various criteria to be specified for each trace. Searches can go forward or backward from the primary (unnamed) marker. If the primary marker has not been set, the search starts at the beginning of the displayed data ("From") for a forwards search and starts at the end of the displayed data ("To") for a backwards search. "Mark" and "Clear" are used to modify the normal time vertical markings such that they can be used to indicate all the times that a specific pattern search condition is true (e.g., every upclock of a specific signal). The "Mark Count" field indicates how many times the specific pattern search condition was encountered. The "Marking Begins at" and "Marking Stops at" fields are used to limit the time over which marking is applied (but they have no effect on searching).

Signal Search Regexp provides an easy means of adding traces to the display. Various functions are provided in the Signal Search requester which allow searching using POSIX regular expressions and bundling (coalescing individual bits into a single vector).

Signal Search Hierarchy provides an easy means of adding traces to the display in a text based treelike fashion.

Signal Search Tree provides an easy means of adding traces to the display. Various functions are provided in the Signal Search Tree requester which allow searching a treelike hierarchy and bundling (coalescing individual bits into a single vector).

Autocoalesce when enabled allows the wave viewer to reconstruct split vectors. Split vectors will be indicated by a "[]" prefix in the search requesters.

Autocoalesce Reversal causes split vectors to be reconstructed in reverse order (only if autocoalesce is also active). This is necessary with some simulators. Split vectors will be indicated by a "[]" prefix in the search requesters.

Autoname Bundles when enabled modifies the bundle up/down operations in the hierarchy and tree searches such that a NULL bundle name is implicitly created which informs GTKWave to create bundle and signal names based on the position in the hierarchy. When disabled, it modifies the bundle up/down operations in the hierarchy and tree searches such that a NULL bundle name is not implicitly created. This informs GTKWave to create bundle and signal names based on the position in the hierarchy only if the user enters a zero-length bundle name. This behavior is the default.

Search Hierarchy Grouping when enabled ensures that new members added to the ``Tree Search'' and ``Hierarchy Search'' widgets are added alphanumerically: first hierarchy names as a group followed by signal names as a group. This is the

default and is recommended. When disabled, hierarchy names and signal names are interleaved together in strict alphanumerical ordering. Note that due to the caching mechanism in ``Tree Search'', dynamically changing this flag when the widget is active may not produce immediately obvious results. Closing the widget then opening it up again will ensure that it follows the behavior of this flag.

Time

The Time submenu contains a superset of the functions performed by the Navigation and Status Panel button groups (see page 24).

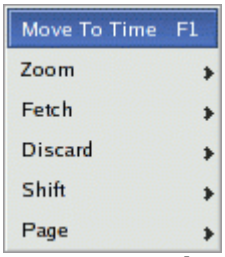

Figure 15: The Time submenu

Move To Time scrolls the waveform display such that the left border is the time entered in the requester.

Zoom Amount allows entry of zero or a negative value for the display zoom. Zero is no magnification.

Zoom Base allows entry of a zoom base for the zoom (magnification per integer step) Allowable values are 1.5 to 10.0. Default is 2.0.

Zoom In is used to increase the zoom factor. Alt + Scrollwheel Down also performs this function.

Zoom Out is used to decrease the zoom factor. Alt + Scrollwheel Up also performs this function.

Zoom Full attempts a "best fit" to get the whole trace onscreen. Note that the trace may be more or less than a whole screen since this isn't a "perfect fit."

Zoom Best Fit attempts a "best fit" to get the whole trace onscreen. Note that the trace may be more or less than a whole screen since this isn't a "perfect fit." Also, if the middle button baseline marker is nailed down, the zoom instead of getting the whole trace onscreen will use the part of the trace between the primary marker and the baseline marker.

Zoom To Start is used to jump scroll to the trace's beginning.

Zoom To End is used to jump scroll to the trace's end.

Zoom Undo is used to revert to the previous zoom value used. Undo only works one level deep.

Fetch Size brings up a requester which allows input of the number of ticks used for fetch/discard operations. Default is 100.

Fetch Right increases the "To" time, which allows more of the trace to be displayed if the "From" and "To" times do not match the actual bounds of the trace.

Fetch Left decreases the "From" time, which allows more of the trace to be displayed if the "From" and "To" times do not match the actual bounds of the trace.

Discard Right decreases the "To" time, which allows less of the trace to be displayed.

Discard Left increases the "From" time, which allows less of the trace to be displayed.

Shift Right scrolls the display window right one tick worth of data. The net action is that the data scrolls left a tick. Ctrl + Scrollwheel Down also performs this function.

Shift Left scrolls the display window left one tick worth of data. The net action is that the data scrolls right a tick. Ctrl + Scrollwheel Up also performs this function.

Page Right scrolls the display window right one page worth of data. The net action is that the data scrolls left a page. Scrollwheel Down also performs this function.

Page Left scrolls the display window left one page worth of data. The net action is that the data scrolls right a page. Scrollwheel Up also performs this function.

Markers

The Markers submenu is used to perform various manipulations on the markers as well as control scrolling offscreen.

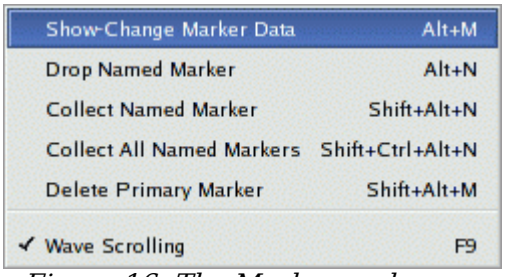

Figure 16: The Markers submenu

Show-Change Marker Data displays and allows the modification of the times for all 26 named markers. The time for each marker must be unique.

Drop Named Marker drops a named marker where the current primary (unnamed) marker is placed. A maximum of 26 named markers are allowed and the times for all must be different.

Collect Named Marker collects a named marker where the current primary (unnamed) marker is placed if there is a named marker at its position.

Collect All Named Markers simply collects any and all named markers which have been dropped.

Delete Primary Marker removes the primary marker from the display if present.

Wave Scrolling allows movement of the primary marker beyond screen boundaries which causes the wave window to scroll when enabled. When disabled, it disallows movement of the primary marker beyond screen boundaries.

View

The View submenu is used to control various attributes dealing with the graphical rendering of status items as well as values in the signal subwindow.

Figure 17: The View submenu

Show Grid toggles the drawing of gridlines in the waveform display.

Show Mouseover toggles the dynamic tooltip for signal names and values which follow the marker on mouse button presses in the waveform display. This is useful for examining the values of closely packed value changes without having to zoom outward and without having to refer to the signal name pane to the left.

Show Base Symbols enables the display of leading base symbols ('$' for hex, '%' for binary, '#' for octal if they are turned off and disables the drawing of leading base symbols if they are turned on. Base symbols are displayed by default.

Dynamic Resize allows GTKWave to dynamically resize the signal window for you when toggled active. This can be helpful during numerous signal additions and/or deletions. This is the default behavior.

Center Zooms when enabled configures zoom in/out operations such that all zooms use the center of the display as the fixed zoom origin if the primary (unnamed) marker is not present, otherwise, the primary marker is used as the center origin. When disabled, it configures zoom in/out operations such that all zooms use the left margin of the display as the fixed zoom origin.

Toggle Max-Marker allows you to switch between the maximum time and marker time for display in the upper right corner of the main window. Default behavior is that the maximum time is displayed.

Constant Marker Update when enabled, allows GTKWave to dynamically show the changing values of the traces under the primary marker while it is being dragged across the screen. This works best with dynamic resizing disabled. When disabled, it restricts GTKWave to only update the trace values when the left mouse button is initially pressed then again when it is released. This is the default behavior.

Draw Roundcapped Vectors draws vector transitions that have sloping edges when enabled. Draws vector transitions that have sharp edges when disabled; this is the default.

Left Justify Signals draws signal names flushed to the left border of the signal window.

Right Justify Signals draws signal names flushed to the right ("equals") side of

the signal window.

Zoom Pow10 Snap snaps time values to a power of ten boundary when active. Fractional zooms are internally stored, but what is actually displayed will be rounded up/down to the nearest power of 10. This only works when the ticks per frame is greater than 100 units.

Full Precision does not round time values when the number of ticks per pixel onscreen is greater than 10 when active. The default is that this feature is disabled.

Remove Pattern Marks removes any vertical traces on the display caused by the Mark feature in pattern search and reverts to the normal format.

LXT Clock Compress to Z reduces memory usage when active as clocks compressed in LXT format are kept at Z in order to save memory. Traces imported with this are permanently kept at Z.

Help

The Help submenu contains options for enabling on-line help as well as displaying program version information.

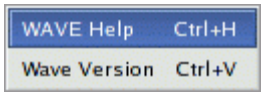

Figure 18: The Help submenu

Wave Help brings up a help window that will show the function of any menu option when that option is selected. Closing the help window will turn off help and return back to normal menu function.

Wave Version merely brings up a requester which indicates the current version of this program.

Quick Start

Sample Design

In the *examples/* directory of the source code distribition a sample Verilog design and testbench for a DES encryptor can be found as *des.v*.

```
10 :/home/bybell/gtkwave-3.0.0pre21/examples> ls -al
total 132
drwxrwxr-x    2 bybell   bybell       4096 Apr 30 14:12 .
drwxr-xr-x    8 bybell   bybell       4096 Apr 29 22:05 ..
-rw-rw-r--    1 bybell   bybell        187 Apr 29 22:09 des.sav
-rw-r--r--    1 bybell   bybell      47995 Apr 29 22:05 des.v
-rw-rw-r--    1 bybell   bybell      68801 Apr 29 22:06 des.vzt
```

If you have a Verilog simulator handy, you can simulate the design to create a VCD file. To try the example in Icarus Verilog (http://www.icarus.com), type the following:

```
/tmp/gtkwave-3.0.0/examples> iverilog des.v && a.out
VCD info: dumpfile des.vcd opened for output.
/tmp/gtkwave-3.0.0/examples> ls -la des.vcd
-rw-rw-r--    1 bybell   bybell    3465481 Apr 30 13:39 des.vcd
```

If you do not have a simulator readily available, you can expand the *des.vzt* file into *des.vcd* by typing the following:

```
/tmp/gtkwave-3.0.0/examples> vzt2vcd des.vzt >des.vcd
VZTLOAD | 1432 facilities
VZTLOAD | Total value bits: 22921
VZTLOAD | Read 1 block header OK
VZTLOAD | [0] start time
VZTLOAD | [704] end time
VZTLOAD |
```

```
VZTLOAD | block [0] processing 0 / 704
/tmp/gtkwave-3.0.0/examples> ls -la des.vcd
-rw-rw-r--    1 bybell   bybell    3456247 Apr 30 13:42 des.vcd
```

You will notice that the generated VCD file is about fifty times larger than the VZT file. This illustrates the compressibility of VCD files and the space saving advantages of using the database formats that GTKWave supports. Normally we would not want to work with VCD as GTKWave is forced to process the whole file rather than access only the data needed, but in the next section we will show how to invoke GTKWave such that VCD files are automatically converted into LXT2 ones.

Next, let's create a stems file that allows us to bring up RTLBrowse.

```
/tmp/gtkwave-3.0.0/examples> vertex des.v -emitstems >des.stems
Vertex: Verilog Parser v0.1.0 (w)1999-2006 BSI

Processing file 'des.v' ...
/tmp/gtkwave-3.0.0/examples> ls -la des.stems
-rw-rw-r--    1 bybell   bybell       4662 Apr 30 13:50 des.stems
```

Stems files only need to be generated when the source code undergoes file layout and/or hierarchy changes.

Now that we have a VCD file and a stems file, we can bring up the viewer.

Launching GTKWave

We already have a VZT file available, but to illustrate the automatic conversion of VCD files, let's use the `-o` option. The `-t` option is used to specify the stems file. The .sav file is a “save file” that contains GTKWave scope state.

```
/tmp/gtkwave-3.0.0/examples> gtkwave -o -t des.stems des.vcd des.sav

GTKWave Analyzer v3.0.0 (w)1999-2006 BSI


Converting VCD File 'des.vcd' to LXT2 file 'des.vcd.lx2'...

1432 symbols span ID range of 1395, using indexing...
[0] start time.
[704] end time.
```

```
LXTLOAD | 1432 facilities
LXTLOAD | Read 2 block headers OK
LXTLOAD | [0] start time
LXTLOAD | [704] end time
LXTLOAD |
LXTLOAD | Finished building 1432 facs.
LXTLOAD | Merging in 145 aliases.
LXTLOAD | Building facility hierarchy tree.
LXTLOAD | Sorting facility hierarchy tree.
LXTLOAD | 1432 facilities
LXTLOAD | Read 2 block headers OK
LXTLOAD | [0] start time
LXTLOAD | [704] end time
LXTLOAD |
```

You will see two sets of LXTLOAD messages. This is normal as RTLBrowse is launched as an external process in order to keep its operations from bogging down the viewer. After these messages scroll by, the GTKWave main window and an RTLBrowse hierarchy window will appear. We are now ready to start experimenting with various features of the wave viewer and RTLBrowse.

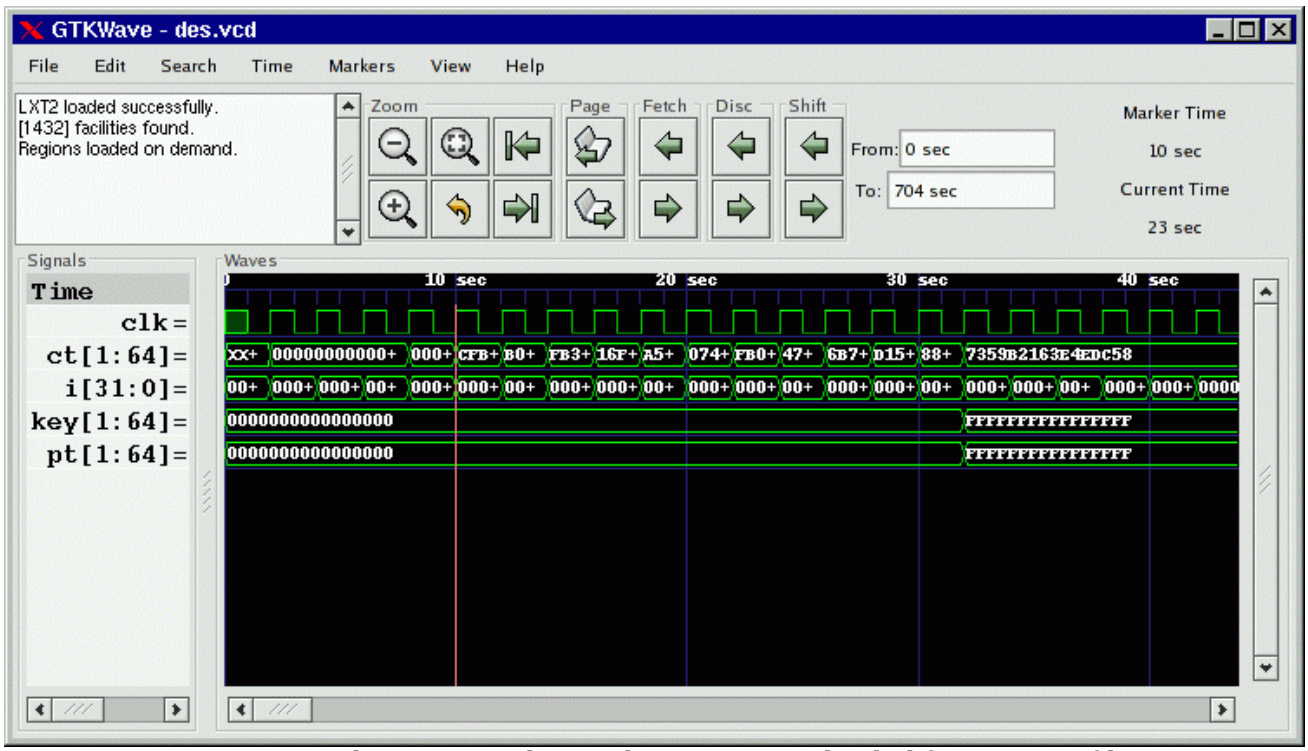

Figure 19: The main window with viewer state loaded from a save file

The RTLBrowse window will come up as seen in Figure 10: The RTLBrowse RTL Design Hierarchy window on page 28, however none of the tree nodes will be opened yet.

Displaying Waveforms

In the preceding section, the viewer was brought up with a save file so when the viewer did appear, the main window already had signals present as seen in Figure 19 on page 43. All the signals in a model do not appear on their own as this would be unwieldy for large models. Instead, it is up to the user to import signals manually. An exception to this exists for VCD files, see the definition of the `enable_vcd_autosave` *`.gtkwaverc`* variable on page . That said, several requesters exist for importing signals into the main window.

Signal Search

The signal search requester accepts a search string as a POSIX regular expression. Any signals found in the dumpfile that match that regular expression are listed in the Matches box and may be individually or multiply selected and imported into the viewer window. The regular expression can be modified in one of four ways: WRange, WStrand, Range, and Strand. No modification is possible with None. This optionally matches the string you enter in the search string above with a Verilog format range (signal[7:0]), a strand (signal.1, signal.0), or with no suffix. The “W” modifier for “Range” and “Strand” explicitly matches on word boundaries. (addr matches unit.freezeaddr[63:0] for “Range” but only unit.addr[63:0] for “WRange” since addr has to be on a word boundary.) Note that when “None” is selected, the search string may be located anywhere in the signal name.

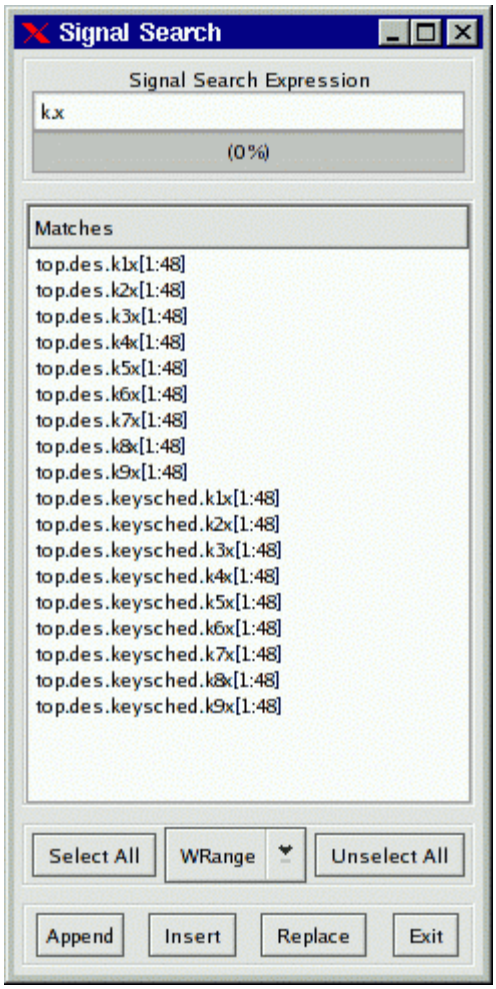

Figure 20: The Signal Search (regular expression search) Requester

Append will add the selected signals to end of the display on the main window.

Insert will add selected signals after last highlighted signal on the main window.

Replace will replace highlighted signals on the main window with signals selected.

Hierarchy Search

Figure 21: The Hierarchy Search Requester

The hierarchy search requester provides a view of the hierarchy in a format similar to the current working directory of a file in a filesystem on a computer. The Signal Hierarchy box contains the current hierarchy and the Children box contain all of the signals in that immediate level of hierarchy and all of the component instantiation names for that level of hierarchy (denoted by a "(+)" prefix). To navigate down a level of hierarchy, click on an item with a "(+)" prefix. To move up a level of hierarchy, click on the ".." line.

Selecting individual items allow you to import traces singly when the Append, Insert, or Replace buttons are clicked. Not selecting anything will do a "deep import" such that all the child signals are imported. Use of that feature is not recommended for very large designs.

Note that is is possible to modify the display order such that components and signals are intermixed in this gadget rather than being separated such that all the components for a given level of hierarchy are listed alphabetically at the top and all signals are listed alphabetically at the bottom. In order to do this, toggle the Search submenu item Search Hierarchy Grouping as described on page 36.

Tree Search

The Tree Search Requester is the requester that most users will feel comfortable using and is also the requester that can optionally be embedded in the main window on versions of GTK greater than or equal to 2.4. See Figure 2: The main window with an embedded SST on page 20 for an example of this.

The Tree Search Requester is composed of a top tree selection box, a signals box, and a POSIX regular expression filter. The tree selection box is used to navigate at the hierarchy level. Click on an item in order to show the signals at that level of hierarchy. In the figure on page 46, the "top" level of hierarchy is selected and the signals box shows what signals are available at that level of hierarchy. Signals may be individually or multiply selected and can be dragged and dropped into the signal window. In addition, a POSIX filter can be specified that allows the selective filtering of signal names at a level of hierarchy which is

handy for finding a specific signal at a level of hierarchy that is very large (e.g., in a synthesized netlist).

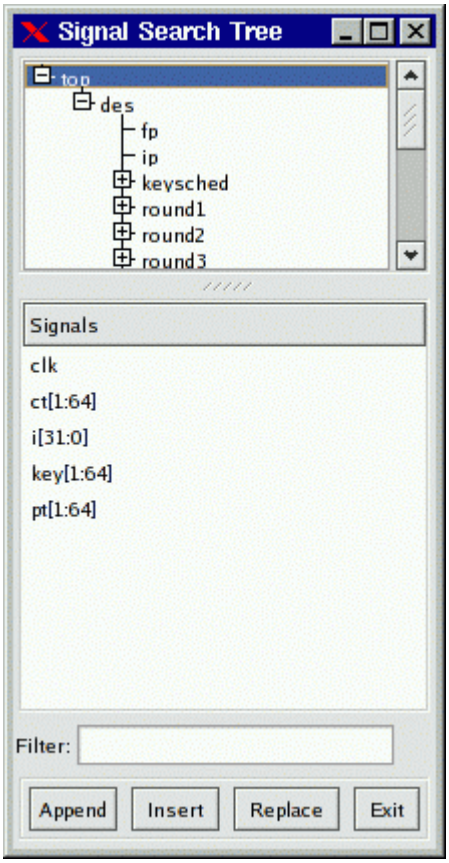

Figure 22: The Signal Search Tree Requester

Signal Save Files

The signals show in the main window can be saved to a file so they can automatically be imported without reselection the next time the viewer is started. In order to save signals to a save file, select the File submenu option Write Save File (As). Save files can also be loaded at any time by selecting the Read Save File option.

Pattern Search

Values, not only nets may be searched on and marked in the wave window. In order to do this, select one or more nets in the signal window and then click on the Search submenu option Pattern Search. A Pattern Search Requester will then appear that will allow various types of search operations for the signals that have been selected.

The following is an example of a Pattern Search Requester being used to mark the rising edges for the clock signal in a simulation model.

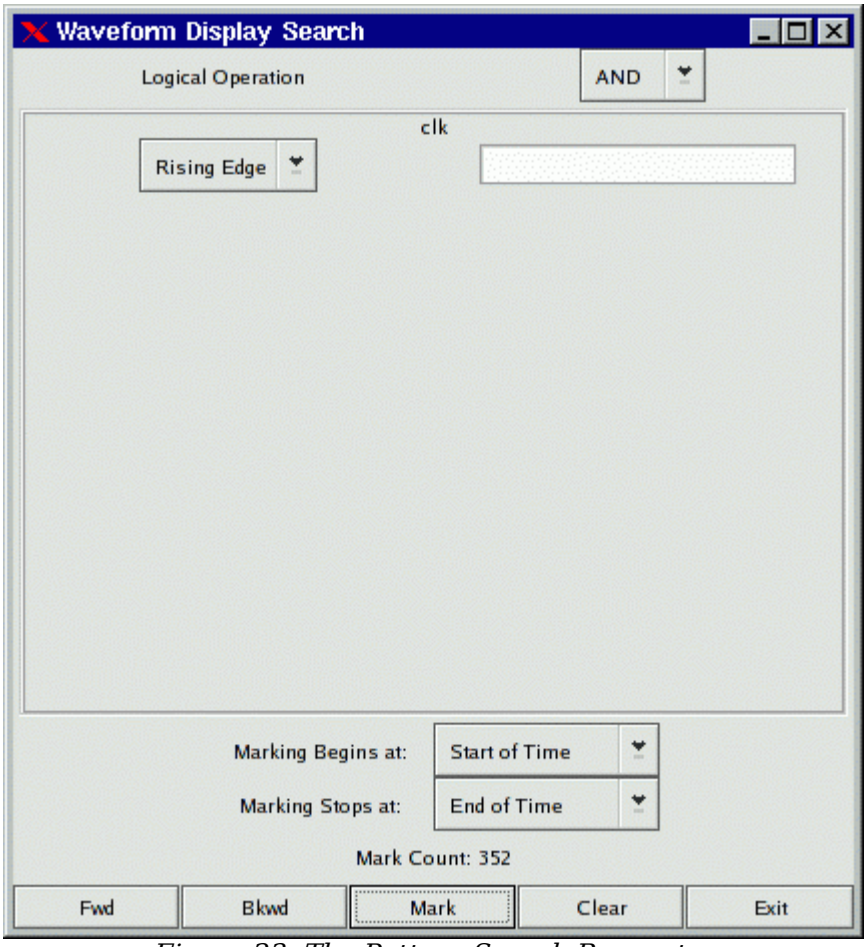

Figure 23: The Pattern Search Requester

The edges as they are marked by the configuration of the Requester in Figure 23 can be seen in Figure 2: The main window with an embedded SST on page 20.

To remove pattern marks, either select another pattern or select the View submenu option Remove Pattern Marks. Note that pattern marks save to the save file and that the actual pattern search criteria is saved, not the absolute times of the individual marks themselves.

Search criteria for individual nets can be edge or value based. For “String” searches (the entry box to the right of the search type box which in the case above is marked “Rising Edge”), note that you must press Enter for the string in order to commit the value to the search.

Alias Files and Attaching External Disassemblers

The viewer supports signal aliasing through both plaintext filters and through external program filters. Note that signal aliasing is a strict one-to-one correspondence so the value represented in the viewer must exactly represent what format your filter expects. (e.g., binary, hexadecimal, with leading base markers, etc.) For your convenience, the comparisons are case insensitive.

For text filters, the viewer looks at an ASCII text file of the following format:

```
#
# this is a comment
#
00 Idle
01 Advance
10 Stop
11 Reset
```

The first non-whitespace item is treated as a literal value that would normally be printed by the viewer and the remaining items on the line are substitution text. Any time this text is encountered if the filter is active, it will replace the left-hand side text with the right-hand side. Leading and trailing whitespaces are removed from the right-hand side item.

To turn on the filter:

1) Highlight the signals you want filtered
2) Edit->Data Format->Translate Filter File->Enable and Select
3) Add Filter to List
4) Click on filter filename
5) Select filter filename from list
6) OK

To turn off the filter:

1) Highlight the signals you want unfiltered.
2) Edit->Data Format->Translate Filter File->Disable

NOTE: Filter configurations load and save properly to and from save files.

An external process that accepts one line in from stdin and returns with data on stdout can be used as a process filter. An example of this are disassemblers. The following sample code would show how to interface with a disassembler function in C:

```
int main(int argc, char **argv)
{
while(1)
        {
        char buf[1025], buf2[1025];

        buf[0] = 0;
        fscanf(stdin, "%s", buf);
        if(buf[0])
                {
                int hx;
                sscanf(buf, "%x", &hx);
```

```
                    ppc_dasm_one(buf2, 0, hx);
                    printf("%s\n", buf2);
                    fflush(stdout);
                    }
            }

    return(0);
    }
```

Note that the fflush(stdout) is necessary, otherwise *gtkwave* will hang. Also note that every line of input needs to generate a line of output or the viewer will hang too.

To turn on the filter:

1) Highlight the signals you want filtered
2) Edit->Data Format->Translate Filter Process->Enable and Select
3) Add Proc Filter to List
4) Click on filter filename
5) Select filter filename from list
6) OK

To turn off the filter:

1) Highlight the signals you want unfiltered.
2) Edit->Data Format->Translate Filter Process->Disable

Debugging the Source Code

See the description for RTLBrowse on page 27. More features are planned to be added in future releases.

Appendix A: Command Line Options Reference

---

gtkwave

```
GTKWAVE(1)                  Simulation Wave Viewer                  GTKWAVE(1)

NAME
       gtkwave - Visualization tool for VCD, LXT, and VZT files

SYNTAX
       gtkwave [option]... [DUMPFILE] [SAVEFILE] [RCFILE]

DESCRIPTION
       Visualization  tool for VCD, LXT, LXT2, VZT, and GHW.  VCD is an indus-
       try standard simulation dump format.  LXT,  LXT2,  and  VZT  have  been
       designed  specifically  for  use  with gtkwave.  GHW is the native VHDL
       format generated by GHDL.  Native dumpers exist in Icarus  Verilog  for
       the  LXT  formats  so  conversion with vcd2lxt(1) or vcd2lxt2(1) is not
       necessary to take direct advantage of LXT with  that  simulator.   AET2
       files  can  also  be  processed provided that libae2rw is available but
       this is only of interest to people who use IBM EDA toolsets.

OPTIONS
       -n,--nocli <directory name>
              Use file requester for dumpfile name.

       -f,--dump <filename>
              Specify dumpfile name.

       -o,--optimize
              optimize VCD to LXT2.  This will automatically call  vcd2lxt2(1)
              to  perform  the  file conversion.  This option is highly recom-
              mended with large VCD files in order to cut down on  the  memory
              usage  required  for  file  viewing.  Can be used in conjunction
              with -v,--vcd.

       -r,--rcfile <filename>
              Specify override .gtkwaverc filename.
```

-i,--indirect <filename>
Specify indirect facs file name. The file contains a series of regular expressions used to limit what signals can be browsed. Signal names which match any of the regular expressions will be viewable. Typically, indirect files are used to reduce memory requirements for extremely large models containing millions of facilities or to strip out top-level hierarchy clutter from BugSpray models. This feature is only available with the AET2 loader.

-l,--logfile <filename>
Specify simulation logfile name. Multiple logfiles may be specified by preceeding each with the command flag. By selecting the numbers in the text widget, the marker will immediately zoom to the specific time value.

-d,--defaultskip
If there is not a .gtkwaverc file in the home directory or current directory and it is not explicitly specified on the command line, when this option is enabled, do not use an implicit configuration file and instead default to the old "whitescreen" behavior.

-D,--dualid <which>
Specify multisession identifier information. The format of "which" is m+nnnnnnnn where m is the session number 0 or 1 and nnnnnnnn is a hexadecimal value indicating the shared memory ID of an array of two gtkwave_dual_ipc_t data structures. The intended use of this flag is for front ends such as twinwave(1).

-s,--start <time>
Specify start time for LXT2/VZT block skip.

-e,--end <time>
Specify end time for LXT2/VZT block skip.

-t,--stems <filename>
Specify stems file for source code annotation. This will automatically launch the rtlbrowse(1) helper process. See vertex(1) for information on stems file generation.

-c,--cpu <numcpus>
Specify number of CPUs available for parallelizable ops (e.g., block prefetching on VZT reads).

-N,--nowm
Disable window manager for most windows. The intended use of this is to be used in conjunction with the --script option, however this also can be used to reparent into an alternate window manager.

-M,--nomenus
Do not render menubar. This is mainly used for making a restricted applet that cannot initiate file I/O on its own, how-

ever it also can be used as a workaround in earlier versions of GTK+ that do not handle GTKSocket/GTKPlug focus interactions properly.

-S,--script <filename>
Specify GUI command script file for execution. Comments are indicated by the '#' character. Lines in the script are the full menu pathname including the first forward slash. Arguments for an option (such as a filename) will follow with one argument each on its own separate line. Menu options that bring up complex requesters (e.g., search) will hang in the GUI waiting for user intervention. The intended use of this option is to allow automated subset extraction to VCD as well as automated Postscript file generation.

-X,--xid <XID>
Specify XID of window for a GtkPlug to connect to. GTKWave does not directly render to a window but instead renders into a GtkPlug expecting a GtkSocket at the other end. Note that there are issues with accelerators working properly so menus are disabled in the componentized version of GTKWave when it functions as a plug-in.

-I,--interactive
Specifies that "interactive" VCD mode is to be used which allows a viewer to navigate a VCD trace while GTKWave is processing the VCD file. When this option is used, the filename is overloaded such that it is the hexadecimal value for the shared memory ID of a writer. Note that the shared memory ID can be passed straight from stdin by using the --vcd option; see the manpage for shmidcat(1) for more details.

-L,--legacy
Specifies that the viewer should use legacy VCD mode rather than the VCD recoder. Note that using legacy mode will require considerably more memory than the recoder and its use is discouraged for very large traces.

-v,--vcd
Use stdin as a VCD dumpfile.

-V,--version
Display version banner then exit.

-h,--help
Display help then exit.

-x,--exit
Exit after loading trace (for loader benchmarking).

FILES

~/.gtkwaverc

EXAMPLES

To run this program the standard way type:

```
               gtkwave dumpfile.vcd

       Alternatively you can run it with a save file as:
               gtkwave dumpfile.vcd dumpfile.sav

       To run interactively using shared memory handle 0x00050003:
               gtkwave -I 00050003 dumpfile.sav

       Command  line  options are not necessary for representing the dumpfile,
       savefile, and rcfile names.  They are merely provided to allow specify-
       ing  them  out of order.

BUGS
       AIX requires  -bmaxdata:0x80000000 to be added to your list of compiler
       flags for xlc if you want GTKWave to be able to access more than  256MB
       of  virtual  memory.  The  value shown allows the VMM to use up to 2GB.
       This may be necessary for very large traces.

       Shift  and Page operations using the wave window hscrollbar may be non-
       functional as you move away from the dump start for very large  traces.
       A  trace  that  goes  out to 45 billion ticks has been known to exhibit
       this problem. This stems from using the gfloat element of the  horizon-
       tal  slider to encode the time value for the left margin. The result is
       a loss of precision for very large values. Use the hotkeys  or  buttons
       at the top of the screen if this is a problem.

       When running under Cygwin, it is required to enable Cygserver if shared
       memory IPC is being used.  Specifically, this occurs when  rtlbrowse(1)
       is launched as a helper process.  See the Cygwin documentation for more
       information on how to enable Cygserver.

AUTHORS
       Anthony Bybell <bybell@nc.rr.com>

SEE ALSO
       gtkwaverc(5)  lxt2vcd(1)  vcd2lxt(1)  vcd2lxt2(1) vzt2vcd(1) vcd2vzt(1)
       vertex(1) rtlbrowse(1) twinwave(1) shmidcat(1)

Anthony Bybell                      3.0.15                          GTKWAVE(1)
```

---

twinwave

```
TWINWAVE(1)           Simulation Wave Viewer Multiplexer           TWINWAVE(1)

NAME
       twinwave - Wraps multiple GTKWave sessions in one window

SYNTAX
       twinwave <arglist1> + <arglist2>
```

```
DESCRIPTION
       Wraps  multiple  GTKWave sessions in one window with synchronized mark-
       ers, horizontal scrolling, and zooming.

EXAMPLES
       To run this program the standard way type:

       twinwave filename1.vcd filename1.sav + filename2.vcd filename2.sav
              Two viewers are then opened in one window.

LIMITATIONS
       twinwave uses the GtkSocket/GtkPlug mechanism to embed  two  gtkwave(1)
       sessions  into one window.  The amount of coupling is currently limited
       to communication of temporal information.  Other  than  that,  the  two
       gtkwave  processes  are isolated from each other as if the viewers were
       spawned separately.  Keep in mind that using the  same  save  file  for
       each session may cause unintended behavior problems if the save file is
       written back to disk: only the session  written  last  will  be  saved.
       (i.e., the save file isn't cloned and made unique to each session.)

AUTHORS
       Anthony Bybell <bybell@nc.rr.com>

SEE ALSO
       gtkwave(1)

Anthony Bybell                       3.0.7                        TWINWAVE(1)
```

---

lxt2miner

```
LXT2MINER(1)                 Dumpfile Data Mining                 LXT2MINER(1)

NAME
       lxt2miner - Data mining of LXT2 files

SYNTAX
       lxt2miner [option]... [LXT2FILE]

DESCRIPTION
       Mines  LXT2  files  for specific data values and generates gtkwave save
       files to stdout for future reload.

OPTIONS
       -d,--dumpfile <filename>
              Specify LXT2 input dumpfile.

       -m,--match <value>
              Specifies "bitwise" match data (binary, real, string)

       -x,--hex <value>
```

```
              Specifies hexadecimal match data that will automatically be con-
              verted to binary for searches

       -n,--namesonly
              Indicates  that  only  facnames  should  be printed in a gtkwave
              savefile compatible format.  By doing this, the file can be used
              to specify which traces are to be imported into gtkwave.

       -h,--help
              Show help screen.

EXAMPLES
       lxt2miner dumpfile.lxt2 --hex 20470000 -n

       This attempts to match the hex value 20470000 across all facilities and
       when the value is encountered, the facname only is printed to stdout in
       order to generate a gtkwave compatible save file.

LIMITATIONS
       lxt2miner  only prints the first time a value is encountered for a spe-
       cific net.  This is done in order to cut down on  the  size  of  output
       files  and to aid in following data such as addresses through a simula-
       tion model.

AUTHORS
       Anthony Bybell <bybell@nc.rr.com>

SEE ALSO
       vztminer(1) vzt2vcd(1) lxt2vcd(1) vcd2lxt2(1) gtkwave(1)

Anthony Bybell                       1.3.64                        LXT2MINER(1)
```

---

lxt2vcd

```
LXT2VCD(1)                    Filetype Conversion                   LXT2VCD(1)

NAME
       lxt2vcd - Coverts LXT2 files to VCD

SYNTAX
       lxt2vcd <filename>

DESCRIPTION
       Converts  LXT2  files to VCD files on stdout.  Note that "regular" LXT2
       files will convert to VCD files with monotonically increasing time val-
       ues.   LXT2  files which are dumped with the "partial" option (to speed
       up access in wave viewers) will dump with monotonically increasing time
       values  per  2k  block of nets.  This may be fixed in later versions of
       lxt2vcd.
```

```
EXAMPLES
       To run this program the standard way type:

       lxt2vcd filename.lxt
              The VCD conversion is emitted to stdout.

LIMITATIONS
       lxt2vcd does not re-create glitches as  these  are  coalesced  together
       into one value change during the writing of the LXT2 file.

AUTHORS
       Anthony Bybell <bybell@nc.rr.com>

SEE ALSO
       vcd2lxt2(1) vcd2lxt(1) gtkwave(1)

Anthony Bybell                      1.3.34                          LXT2VCD(1)
```

---

mvl2lxt

```
MVL2LXT(1)                    Filetype Conversion                   MVL2LXT(1)

NAME
       mvl2lxt - Coverts MVLSIM AET files to LXT

SYNTAX
       mvl2lxt <filename.aet> <filename.lxt>

DESCRIPTION
       Converts  AET  files  to LXT.  This is experimental as it is not a com-
       plete implementation and is not intended for general use.

AUTHORS
       Anthony Bybell <bybell@nc.rr.com>

SEE ALSO
       tex2vcd(1) mvl2vcd(1) lxt2vcd(1) vcd2lxt2(1) vcd2lxt(1) gtkwave(1)

Anthony Bybell                      1.3.34                          MVL2LXT(1)
```

---

mvl2vcd

```
MVL2VCD(1)                    Filetype Conversion                   MVL2VCD(1)

NAME
```

```
       mvl2vcd - Coverts MVLSIM AET files to VCD

SYNTAX
       mvl2vcd <filename.aet>

DESCRIPTION
       Converts AET files to VCD on stdout.  This is experimental as it is not
       a complete implementation and is not intended for general use.

AUTHORS
       Anthony Bybell <bybell@nc.rr.com>

SEE ALSO
       tex2vcd(1) mvl2lxt(1) lxt2vcd(1) vcd2lxt2(1) vcd2lxt(1) gtkwave(1)

Anthony Bybell                      1.3.34                          MVL2VCD(1)
```

---

rtlbrowse

```
RTLBROWSE(1)                      File Viewing                      RTLBROWSE(1)

NAME
       rtlbrowse  - Allows hierarchical browsing of Verilog HDL sourcecode and
       library design files.

SYNTAX
       rtlbrowse <stemsfilename>

DESCRIPTION
       Allows hierarchical browsing of  Verilog  HDL  sourcecode  and  library
       design files.  Navigation through the hierarchy may be done by clicking
       open areas of the tree widget and clicking on the individual levels  of
       hierarchy.   Inside  the sourcecode, selecting the module instantiation
       name by double clicking or selecting part of  the  name  through  drag-
       clicking will descend deeper into the RTL hierarchy.  Note that it per-
       forms optional source code annotation when called as a helper  applica-
       tion  by   gtkwave(1)  and when the primary marker is set.  Source code
       annotation is not available for all supported dumpfile  types.   It  is
       directly  available  for  LXT2,  VZT,  and  AET2.   For  VCD,  use  the
       -o,--optimize option of gtkwave(1) in order to optimize  the  VCD  file
       into LXT2.  All other dumpfile types (LXT, GHW) are unsupported at this
       time.

EXAMPLES
       To run this program the standard way type:

       rtlbrowse stemsfile
              The RTL is then brought up in a GTK  tree  viewer.   Stems  must
              have  been  previously generated with vertex(1).  Note that gtk-
              wave(1) will bring up this program as a client  application  for
```

```
              sourcecode  annotation.   It does that by bringing up the viewer
              with the shared memory ID of a segment of memory in  the  viewer
              rather than using a stems filename.

AUTHORS
       Anthony Bybell <bybell@nc.rr.com>

SEE ALSO
       vertex(1) gtkwave(1)

Anthony Bybell                        0.1.0                        RTLBROWSE(1)
```

---

tex2vcd

```
TEX2VCD(1)                     Filetype Conversion                   TEX2VCD(1)

NAME
       tex2vcd - Coverts TEXSIM AET files to VCD

SYNTAX
       tex2vcd <filename.aet>

DESCRIPTION
       Converts AET files to VCD on stdout.  This is experimental as it is not
       a complete implementation and is not intended for general use.

AUTHORS
       Anthony Bybell <bybell@nc.rr.com>

SEE ALSO
       mvl2vcd(1) mvl2lxt(1) lxt2vcd(1) vcd2lxt2(1) vcd2lxt(1) gtkwave(1)

Anthony Bybell                       1.3.34                          TEX2VCD(1)
```

---

tla2vcd

```
TLA2VCD(1)             Tektronix File Format Conversion             TLA2VCD(1)

NAME
       tla2vcd - Converts TLA to VCD or LST files

SYNTAX
```

```
       tla2vcd [option]... TLAFILE [[TLAFILE]...]

DESCRIPTION
       Converts  TLA files from Tektronix logic analyzers to VCD or LST files.
       Conversion to VCD allows viewing in tools such as  gtkwave(1).

OPTIONS
       -t     Generate text LST file instead of VCD file

       -l     List the available channels/groups and exit

       -s  <signal_list>
              Select signals from regular expression list

       -a     All channels

       -m     Append MagniVu info (only in listing mode)

       -z     Compress output

       -v     Verbose

AUTHORS
       Emil

SEE ALSO
       lxt2vcd(1) vzt2vcd(1) gtkwave(1)

Emil                                   1.0                            TLA2VCD(1)
```

---

vcd2lxt

```
VCD2LXT(1)                     Filetype Conversion                   VCD2LXT(1)

NAME
       vcd2lxt - Converts VCD files to interlaced or linear LXT files

SYNTAX
       vcd2lxt [VCDFILE] [LXTFILE] [option]...

DESCRIPTION
       Converts  VCD  files  to interlaced or linear LXT files.  Noncompressed
       interlaced files will provide the fastest  access,  linear  files  will
       provide the slowest yet have the greatest compression ratios.

OPTIONS
       -stats Prints  out  statistics  on  all nets in VCD file in addition to
              performing the conversion.

       -clockpack
```

```
              Apply two-way subtraction algorithm in order  to  identify  nets
              whose   value   changes   by  a  constant  XOR  or  whose  value
              increases/decreases by a constant amount per  constant  unit  of
              time.   This  option  can  reduce  dumpfile size dramatically as
              value changes can be represented  by  an  equation  rather  than
              explicitly as a triple of time, net, and value.

       -chgpack
              Emit data to file after being filtered through zlib (gzip).

       -linear
              Write  out  LXT  in "linear" format with no backpointers.  These
              are re-generated during initialization  in  gtkwave.   Addition-
              ally, use libbz2 (bzip2) as the compression filter.

       -dictpack <size>
              Store  value  changes  greater  than or equal to size bits as an
              index into a dictionary.  Experimentation shows that a value  of
              18 is optimal for most cases.

EXAMPLES
       Note  that  you  should  specify  dumpfile.vcd  directly or use "-" for
       stdin.

       vcd2lxt dumpfile.vcd dumpfile.lxt -clockpack -chgpack -dictpack 18
              This turns on clock packing, zlib compression, and  enables  the
              dictionary  encoding.   Note  that using no options writes out a
              normal LXT file.

       vcd2lxt dumpfile.vcd dumpfile.lxt -clockpack -linear -dictpack 18
              Uses linear mode for even smaller files.

AUTHORS
       Anthony Bybell <bybell@nc.rr.com>
```

---

vcd2lxt2

```
VCD2LXT2(1)                    Filetype Conversion                  VCD2LXT2(1)

NAME
       vcd2lxt2 - Converts VCD files to LXT2 files

SYNTAX
       vcd2lxt2 [option]... [VCDFILE] [LXTFILE]

DESCRIPTION
       Converts VCD files to LXT2 files.

OPTIONS
       -v,--vcdname <filename>
```

```
               Specify VCD input filename.

       -l,--lxtname <filename>
               Specify LXT2 output filename.

       -d,--depth <value>
               Specify 0..9 gzip compression depth, default is 4.

       -m,--maxgranule <value>
               Specify number of granules per section, default is 8.  One gran-
               ule is equal to 32 timsteps.

       -b,--break <value>
               Specify break size (default = 0 = off).  When the break size  is
               exceeded, the LXT2 dumper will dump all state information at the
               next convenient granule plus dictionary boundary.

       -p,--partialmode <mode>
               Specify partial zip mode 0 = monolithic, 1 = separation.   Using
               a  value of 1 expands LXT2 filesize but provides fast access for
               very large traces.  Note that the default mode is neither  mono-
               lithic nor separation: partial zip is disabled.

       -c,--checkpoint <mode>
               Specify  checkpoint  mode.   0  is on which is default, and 1 is
               off.  This is disabled when the break size is active.

       -h,--help
               Show help screen.

       EXAMPLES

       Note that you should specify  dumpfile.vcd  directly  or  use  "-"  for
       stdin.

       vcd2lxt dumpfile.vcd dumpfile.lxt --depth 9 --break 1073741824
               This  sets the compression level to 9 and sets the break size to
               1GB.

       vcd2lxt dumpfile.vcd dumpfile.lxt --depth 9 --maxgranule 256
               Allows more granules per section which allows for  greater  com-
               pression.

LIMITATIONS
       vcd2lxt2  does  not store glitches as these are coalesced together into
       one value change during the writing of the LXT2 file.

AUTHORS
       Anthony Bybell <bybell@nc.rr.com>

SEE ALSO
       lxt2vcd(1) vcd2lxt2(1) gtkwave(1)

Anthony Bybell                      1.3.42                          VCD2LXT2(1)
```

vcd2vzt

```
VCD2VZT(1)                    Filetype Conversion                    VCD2VZT(1)

NAME
       vcd2vzt - Converts VCD files to VZT files

SYNTAX
       vcd2vzt [option]... [VCDFILE] [VZTFILE]

DESCRIPTION
       Converts VCD files to VZT files.

OPTIONS
       -v,--vcdname <filename>
              Specify VCD input filename.

       -l,--vztname <filename>
              Specify VZT output filename.

       -d,--depth <value>
              Specify 0..9 gzip compression depth, default is 4.

       -m,--maxgranule <value>
              Specify number of granules per section, default is 8.  One gran-
              ule is equal to 32 timesteps.

       -b,--break <value>
              Specify break size (default = 0 = off).  When the break size  is
              exceeded,  the VZT dumper will dump all state information at the
              next convenient granule plus dictionary boundary.

       -z,--ziptype <value>
              Specify zip type (default = 0 gzip, 1 = bzip2).  This allows you
              to  override  the  default  compression  algorithm to use a more
              effective one at the expense  of  greater  runtime.   Note  that
              bzip2  does not decompress as fast as gzip so the viewer will be
              about two times slower when decompressing blocks.

       -t,--twostate
              Forces MVL2 twostate mode (default is MVL4).  When enabled,  the
              trace will only store 0/1 values for binary facilities.  This is
              useful for functional simulation and will speed  up  dumping  as
              well as make traces somewhat smaller.

       -r, --rle
              Uses  an bitwise RLE compression on the value table.  Default is
              off.  When enabled, this causes  the  trace  data  table  to  be
              stored  using an alternate representation which can improve com-
```

```
              pression in many cases.

       -h,--help
              Show help screen.

       EXAMPLES

       Note that you should specify  dumpfile.vcd  directly  or  use  "-"  for
       stdin.

       vcd2vzt dumpfile.vcd dumpfile.lxt --depth 9 --break 1073741824
              This  sets the compression level to 9 and sets the break size to
              1GB.

       vcd2vzt dumpfile.vcd dumpfile.lxt --depth 9 --maxgranule 512
              Allows more granules per section which allows for  greater  com-
              pression at the expense of memory usage.

LIMITATIONS
       vcd2vzt  does  not  store glitches as these are coalesced together into
       one value change during the writing of the VZT file.

AUTHORS
       Anthony Bybell <bybell@nc.rr.com>

SEE ALSO
       vzt2vcd(1) lxt2vcd(1) vcd2lxt2(1) gtkwave(1)

Anthony Bybell                      1.3.48                          VCD2VZT(1)
```

---

vertex

```
VERTEX(1)                     Verilog Compilation                    VERTEX(1)

NAME
       vertex - Parses and processes Verilog HDL files

SYNTAX
       vertex [VERILOGFILE]... [option]...

DESCRIPTION
       Parses  Verilog  HDL files for use by other tools.  The Verilog grammar
       used is 1364-1995.

OPTIONS
       -h[elp]
              Prints out help screen.

       -emitmono fname
              Emit monolithic (parser view of) file to fname.
```

```
       -emitstems
              Emit source code stems to stdout.

       -emitvars
              Emit source code variables to stdout.

       -Dx=y  Equivalent to `define X Y in source.

       +define+x=y
              Equivalent to `define X Y in source.

       +incdir+dirname
              Add dirname to include search path.

       +libext+ext
              Add ext to filename when searching for files.

       -pragma name
              Add name (synopsys, verilint, vertex) to accepted pragmas.

       -y dirname
              Add directory dirname to source input path.

       -yi dirname
              Add directory dirname to source  input  path  (case  insensitive
              search).

       -f filename
              Insert args from filename.  Does not work recursively.

EXAMPLES
       The  following  indicates that the library extension is .v and that the
       include directory is the current working directory and that the library
       directory  is  also the current working directory.  Stems generation is
       enabled to generate a stems file for use  with  other  tools.   Various
       compile-time defines are also defined on the command line.

       vertex  XYZ450AC6V1.v  -emitstems -y . +incdir+. +libext+.v -DUTLB_Out-
       putData=0 -D__PORTALS_VERILOG__

AUTHORS
       Anthony Bybell <bybell@nc.rr.com>

SEE ALSO
       rtlbrowse(1) gtkwave(1)

Anthony Bybell                        0.1.1                            VERTEX(1)
```

---

vzt2vcd

```
VZT2VCD(1)                    Filetype Conversion                  VZT2VCD(1)

NAME
       vzt2vcd - Coverts VZT files to VCD

SYNTAX
       vzt2vcd <filename>

DESCRIPTION
       Converts VZT files to VCD files on stdout.

EXAMPLES
       To run this program the standard way type:

       vzt2vcd filename.vzt
              The VCD conversion is emitted to stdout.

LIMITATIONS
       vzt2vcd  does  not  re-create  glitches as these are coalesced together
       into one value change during the writing of the VZT file.

AUTHORS
       Anthony Bybell <bybell@nc.rr.com>

SEE ALSO
       vcd2lxt2(1) vcd2lxt(1) lxt2vcd(1) gtkwave(1)

Anthony Bybell                       1.3.44                          VZT2VCD(1)
```

---

vztminer

```
VZTMINER(1)                  Dumpfile Data Mining                  VZTMINER(1)

NAME
       vztminer - Data mining of VZT files

SYNTAX
       vztminer [option]... [VZTFILE]

DESCRIPTION
       Mines  VZT  files  for  specific data values and generates gtkwave save
       files to stdout for future reload.

OPTIONS
       -d,--dumpfile <filename>
              Specify VZT input dumpfile.

       -m,--match <value>
              Specifies "bitwise" match data (binary, real, string)
```

```
       -x,--hex <value>
              Specifies hexadecimal match data that will automatically be con-
              verted to binary for searches

       -n,--namesonly
              Indicates  that  only  facnames  should  be printed in a gtkwave
              savefile compatible format.  By doing this, the file can be used
              to specify which traces are to be imported into gtkwave.

       -h,--help
              Show help screen.

EXAMPLES
       vztminer dumpfile.vzt --hex 20470000 -n

       This attempts to match the hex value 20470000 across all facilities and
       when the value is encountered, the facname only is printed to stdout in
       order to generate a gtkwave compatible save file.

LIMITATIONS
       vztminer  only  prints the first time a value is encountered for a spe-
       cific net.  This is done in order to cut down on  the  size  of  output
       files  and to aid in following data such as addresses through a simula-
       tion model.

AUTHORS
       Anthony Bybell <bybell@nc.rr.com>

SEE ALSO
       lxt2miner(1) vzt2vcd(1) lxt2vcd(1) vcd2lxt2(1) gtkwave(1)

Anthony Bybell                      1.3.64                        VZTMINER(1)
```

---

shmidcat

```
SHMIDCAT(1)                Shared Memory Trampoline                SHMIDCAT(1)

NAME
       shmidcat - Copies stdin/file to a shared memory block for gtkwave(1)

SYNTAX
       shmidcat [VCDFILE]

DESCRIPTION
       Copies  either  the  file specified at the command line or stdin (if no
       file specified) line by line to a shared  memory  block.   stdout  will
       contain a shared memory ID which should be passed on to gtkwave(1).

EXAMPLES
```

```
       To run this program the standard way type:

       cat whatever.vcd | shmidcat
              The shared memory ID is emitted to stdout.

       shmidcat whatever.vcd | gtkwave -v -I whatever.sav
              GTKWave directly grabs the ID from stdin.

LIMITATIONS
       This program is mainly for illustrative and testing purposes only.  Its
       primary use is for people who wish to write interactive VCD dumpers for
       gtkwave(1)  as  its  source  code  may  be  examined,  particularly the
       emit_string() function.  It can also be used to test if an existing VCD
       file  will load properly in interactive mode.  Note that it can also be
       used to redirected VCD files which are written into a pipe to  gtkwave(1)
       in a non-blocking fashion.

AUTHORS
       Anthony Bybell <bybell@nc.rr.com>

SEE ALSO
       gtkwave(1)

Anthony Bybell                       3.0.8                        SHMIDCAT(1)
```

Appendix B: .gtkwaverc Variable Reference

A difference in Windows to be aware of is that the default (if unspecified) *.gtkwaverc* file is known as *gtkwave.ini* and resides in the current working directory.

```
GTKWAVERC(5)              GTKWave Configuration File              GTKWAVERC(5)

NAME
       gtkwaverc - GTKWave Configuration File

SYNTAX
       option <value>

       The  configuration file is a series of option and value pairs.  Comment
       lines marked with an initial  '#'  character  are  permissible.   Blank
       lines are ignored.

DESCRIPTION
       Configuration  file for gtkwave(1).  The search path for the configura-
       tion file (if unspecified) is the current working directory followed by
       the user's home directory.

OPTIONS

       accel <"pathvalue" accelerator>
              This allows replacement of menu accelerator keys.  See the .gtk-
              waverc file in the source distribution for examples on pathvalue
              and accelerator syntax.  The special accelerator value of (null)
              means that no accelerator is bound to the menu item.

       alt_hier_delimeter <value>
              This  allows another character in addition to the hier_delimeter
              to be used to delimit levels in the hierarchy for VCD. Only  the
              first  character  in the value is significant. Note that this is
              normally off. The intended use is to resolve the hierarchies  of
              netlist based models that often contain slashes to delimit hier-
              archy inside of $var statements.
```

```
append_vcd_hier <value>
       Allows the specification of a prefix hierarchy  for  VCD  files.
       This can be done in "pieces," so that multiple layers of hierar-
       chy are prepended to symbol  names  with  the  most  significant
       addition  occurring  first  (see .gtkwaverc in the  examples/vcd
       directory). The intended use of this is to have the  ability  to
       add  "project"  prefixes  which allow easier selection of every-
       thing from the tree hierarchy.

atomic_vectors <value>
       Speeds up vcd loading and takes up less memory. This  option  is
       deprecated; it is currently the default.

autocoalesce <value>
       A  nonzero  value  enables  autocoalescing  of  VCD vectors when
       applicable. This may be toggled dynamically during  wave  viewer
       usage.

autocoalesce_reversal <value>
       causes  split vectors to be reconstructed in reverse order (only
       if autocoalesce is also active).

autoname_bundles <value>
       A nonzero value indicates that GTKWave will create its own  bun-
       dle names rather than prompting the user for them.

color_0 <value>
       trace color when 0.

color_1 <value>
       trace color when 1.

color_back <value>
       background color.

color_baseline <value>
       middle mouse button marker color.

color_black <value>
       color value for "black" in signal window.

color_dash <value>
       trace color when don't care ("-").

color_dashfill <value>
       trace color (inside of box) when don't care ("-").

color_dkblue <value>
       color value for "dark blue" in signal window.

color_dkgray <value>
       color value for "dark gray" in signal window.

color_grid <value>
```

```
       grid color (use Alt-G/Shift-Alt-G to show/hide grid).

color_high <value>
       trace color when high ("H").

color_low <value>
       trace color when low ("L").

color_ltgray <value>
       color value for "light gray" in signal window.

color_mark <value>
       color of the named markers.

color_mdgray <value>
       color value for "medium gray" in signal window.

color_mid <value>
       trace color when floating ("Z").

color_normal <value>
       color value for "normal" GTK state in signal window.

color_time <value>
       text color for timebar.

color_timeb <value>
       text color for timebar's background.

color_trans <value>
       trace color when transitioning.

color_u <value>
       trace color when undefined ("U").

color_ufill <value>
       trace color (inside of box) when undefined ("U").

color_umark <value>
       color of the unnamed (primary) marker.

color_value <value>
       text color for vector values.

color_vbox <value>
       vector color (horizontal).

color_vtrans <value>
       vector color (verticals/transitions).

color_w <value>
       trace color when weak ("W").

color_wfill <value>
       trace color (inside of box) when weak ("W").
```

```
color_white <value>
       color value for "white" in signal window.

color_x <value>
       trace color when undefined ("X") (collision for VHDL).

color_xfill <value>
       trace  color (inside of box) when undefined ("X") (collision for
       VHDL).

constant_marker_update <value>
       A nonzero value indicates that the values for traces  listed  in
       the  signal  window  are  to be updated constantly when the left
       mouse button is being held down rather  than  only  when  it  is
       first pressed then when released (which is the default).

convert_to_reals <value>
       Converts all integer and parameter VCD declarations to real-val-
       ued ones when set to a nonzero/yes value. The positive aspect of
       this  is that integers and parameters will take up less space in
       memory and will automatically display  in  decimal  format.  The
       negative  aspect  of  this  is that integers and parameters will
       only be displayable as  decimals  and  can't  be  bit  reversed,
       inverted, etc.

cursor_snap <value>
       A nonzero value indicates the number of pixels the marker should
       snap to for the nearest signal transition.

disable_mouseover <value>
       A nonzero value indicates that signal/value tooltip pop up  bub-
       bles  on  mouse  button  presses should be disabled in the value
       window. A zero value indicates that  value  tooltips  should  be
       active. (default is disabled).

disable_tooltips <value>
       A nonzero value indicates that tooltip pop up bubbles should  be
       disabled.  A zero value indicates that tooltips should be active
       (default).

do_initial_zoom_fit <value>
       A nonzero value indicates that the  trace  should  initially  be
       crunched to fit the screen. A zero value indicates that the ini-
       tial zoom should be zero (default).

dynamic_resizing <value>
       A nonzero value indicates that dynamic resizing should  be  ini-
       tially  enabled  (default).  A zero value indicates that dynamic
       resizing should be initially disabled.

enable_fast_exit <value>
       Allows exit without bringing up a confirmation requester.

enable_ghost_marker <value>
```

lets the user turn on/off the ghost marker during primary marker dragging. Default is enabled.

enable_horiz_grid <value>
A nonzero value indicates that when grid drawing is enabled, horizontal lines are to be drawn. This is the default.

enable_vcd_autosave <value>
causes the vcd loader to automatically generate a .sav file (vcd_autosave.sav) in the cwd if a save file is not specified on the command line. Note that this mirrors the VCD $var defs and no attempt is made to coalesce split bitvectors back together.

enable_vert_grid <value>
A nonzero value indicates that when grid drawing is enabled, vertical lines are to be drawn. This is the default. Note that all possible combinations of enable_horiz_grid and enable_vert_grid values are acceptable.

fontname_logfile <value>
When followed by an argument, this indicates the name of the X11 font that you wish to use for the logfile browser. You may generate appropriate fontnames using the xfontsel program.

fontname_signals <value>
When followed by an argument, this indicates the name of the X11 font that you wish to use for signals. You may generate appropriate fontnames using the xfontsel program.

fontname_waves <value>
When followed by an argument, this indicates the name of the X11 font that you wish to use for waves. You may generate appropriate fontnames using the xfontsel program. Note that the signal font must be taller than the wave font or the viewer will complain then terminate.

force_toolbars <value>
When enabled, this forces everything above the signal and wave windows to be rendered as toolbars. This allows for them to be detached which allows for more usable wave viewer space. By default this is off.

hide_sst <value>
Hides the Signal Search Tree widget for GTK2.4 and greater such that it is not embedded into the main viewer window. It is still reachable as an external widget through the menus.

hier_delimeter <value>
This allows characters other than '/' to be used to delimit levels in the hierarchy. Only the first character in the value is significant.

hier_grouping <value>
For the tree widgets, this allows the hierarchies to be grouped

in a single place rather than spread among the netnames.

hier_max_level <value>
Sets the maximum hierarchy depth (from the right side) to display for trace names. Note that a value of zero displays the full hierarchy name.

hpane_pack <value>
A nonzero value indicates that the horizontal pane should be constructed using the gtk_paned_pack functions (default and recommended). A zero value indicates that gtk_paned_add will be used instead.

initial_window_x <value>
Sets the size of the initial width of the wave viewer window. Values less than or equal to zero will set the initial width equal to -1 which will let GTK determine the minimum size.

initial_window_xpos <value>
Sets the size of the initial x coordinate of the wave viewer window. -1 which will let the window manager determine the position.

initial_window_y <value>
Sets the size of the initial height of the wave viewer window. Values less than or equal to zero will set the initial width equal to -1 which will let GTK determine the minimum size.

initial_window_ypos <value>
Sets the size of the initial y coordinate of the wave viewer window. -1 which will let the window manager determine the position.

left_justify_sigs <value>
When nonzero, indicates that the signal window signal name justification should default to left, else the justification is to the right (default).

lxt_clock_compress_to_z <value>
For LXT (not LXT2) allows clocks to compress to a 'z' value so that regular/periodic value changes may be noted.

page_divisor <value>
Sets the scroll amount for page left and right operations. (The buttons, not the hscrollbar.) Values over 1.0 are taken as 1/x and values equal to and less than 1.0 are taken literally. (i.e., 2 gives a half-page scroll and .67 gives 2/3). The default is 1.0.

ps_maxveclen <value>
sets the maximum number of characters that can be printed for a value in the signal window portion of a postscript file (not including the net name itself). Legal values are 4 through 66 (default).

show_base_symbols <value>
A nonzero value (default) indicates that the numeric base symbols for hexadecimal ('$'), binary ('%'), and octal ('#') should be rendered. Otherwise they will be omitted.

show_grid <value>
A nonzero value (default) indicates that a grid should be drawn behind the traces. A zero indicates that no grid should be drawn.

splash_disable <value>
Turning this off enables the splash screen with the GTKWave mascot when loading a trace. Default is on.

sst_expanded <value>
When true allows the SST dialog (when not hidden) to come up already expanded.

use_big_fonts <value>
A nonzero value indicates that any text rendered into the wave window will use fonts that are four points larger in size than normal. This can enhance readability. A zero value indicates that normal font sizes should be used.

use_full_precision <value>
does not round time values when the number of ticks per pixel onscreen is greater than 10 when active. The default is that this feature is disabled.

use_maxtime_display <value>
A nonzero value indicates that the maximum time will be displayed in the upper right corner of the screen. Otherwise, the current primary (unnamed) marker time will be displayed. This can be toggled at any time with the Toggle Max-Marker menu option.

use_nonprop_fonts <value>
Allows accelerated redraws of the signalwindow that can be done because the font width is constant. Default is off.

use_roundcaps <value>
A nonzero value indicates that vector traces should be drawn with rounded caps rather than perpendicular ones. The default for this is zero.

use_scrollbar_only <value>
A nonzero value indicates that the page, shift, fetch, and discard buttons should not be drawn (i.e., time manipulations should be through the scrollbar only rather than front panel buttons). The default for this is zero.

vcd_explicit_zero_subscripts <value>
indicates that signal names should be stored internally as name.bitnumber when enabled. When disabled, a more "normal" ordering of name[bitnumber] is used. Note that when disabled,

the Bundle Up and Bundle Down options are disabled in the Signal Search Regexp, Signal Search Hierarchy, and Signal Search Tree options. This is necessary as the internal data structures for signals are represented with one "less" level of hierarchy than when enabled and those functions would not work properly. This should not be an issue if atomic_vectors are enabled. Default for vcd_explicit_zero_subscripts is disabled.

vcd_preserve_glitches <value>
indicates that any repeat equal values for a net spanning different time values in the vcd file are not to be compressed into a single value change but should remain in order to allow glitches to be present for this case. Default for vcd_preserve_glitches is disabled.

vcd_warning_filesize <value>
produces a warning message if the VCD filesize is greater than the argument's size in MB. Set to zero to disable this.

vector_padding <value>
indicates the number of pixels of extra whitespace that should be added to any strings for the purpose of calculating text in vectors. Permissible values are 0 to 16 with the default being 4.

vlist_compression <value>
indicates the value to pass to zlib during vlist processing (which is used in the VCD recoder). -1 disables compression, 0-9 correspond to the value zlib expects. 4 is default.

wave_scrolling <value>
a nonzero value enables scrolling by dragging the marker off the left or right sides of the wave window. A zero value disables it.

zoom_base <value>
allows setting of the zoom base with a value between 1.5 and 10.0. Default is 2.0.

zoom_center <value>
a nonzero value enables center zooming, a zero value disables it.

zoom_pow10_snap <value>
corresponds to the Zoom Pow10 Snap menu option. Default for this is disabled (zero).

AUTHORS
Anthony Bybell <bybell@nc.rr.com>

SEE ALSO
gtkwave(1)

Anthony Bybell 3.0.0 GTKWAVERC(5)

Appendix C: VCD Recoding

VList Recoding Stategy

VCD files can now be recoded in gtkwave on a per-signal basis using a modified form of VList. The VList structure used by gtkwave is as follows:

```
struct vlist_t
{
struct vlist_t *next;
unsigned int siz;
int offs;
unsigned int elem_siz;
};
```

The `elem_siz` is always equal to 1 byte. For the first structure, the `siz` field is 1. For the next one, it will be 2, then 4, and so forth. Given this doubling property, this structure allows a dynamically growing *indexable* array. The `offs` field is a pointer to the next element to be written to the array. It starts at zero. When the `offs` value is equal to `siz`, another VList is prepended in front of the existing one. Note that the `siz` number of elements are allocated directly after the `vlist_t` structure, so the first element can be found by skipping `sizeof(vlist_t)` bytes from the start of the `vlist_t` structure.

When a new `vlist_t` is prepended in front of an old one, a compaction of the data elements following the old `vlist_t` is attempted with zlib when the number of bytes to compact is 64 or greater. If the compaction results in a savings of space, the uncompressed `vlist_t` is discarded and the compressed one is kept. To signify that a particular `vlist_t` is compressed, the offs field is negated. (Thus, when accessing the list later, a negative offset indicates that the `vlist_t` structure in question is compressed.) Note that for a given VList, it is possible that there will be both compressed and uncompressed `vlist_t` structures, and

this will have to be taken into account when they are accessed later.

When a `vlist_t` is finalized (i.e., when no more elements are to be added to it), a compression is attempted. If that fails (i.e., not appreciable size savings), a naive truncation of the unused bytes `(siz-offs)*elem_siz` is done. Given the nature of the data in this list, compression usually succeeds.

These VList structures remain dormant in memory in their (possibly) compressed form until they are needed to be accessed. At that time they are decompressed (if required), traversed, and destroyed as they will no longer be needed. The actual data contained in the memory area following the `vlist_t` structures to represent VHDL/Verilog value changes will now be described.

Time Encoding

Along with the value changes, an uncompressed VList of 64-bit integers is also generated as time values are parsed from the VCD file. (i.e., lines of the form "#1000") As the time values are added to that VList, a numerical index (zeroth, first, second) is maintained separately that indicates what the current time index is.

The reason for maintaining a list of indices is so that value changes can be encoded as relative distances in this list rather than actual 64-bit integers.

Single-bit Encoding

Single bit value changes are encoded as a variable length integer of the format (delta<<2)|(zero_one_bit<<1) when the value is zero or one, or (delta<<4)| rcv_bit_value for all other bit values.

The "delta" value represents how many timesteps in the VCD file have taken place since the previous value change for a signal. Look at the following for an example:

```
#1000 clk = 0    index = n
#1001                    n + 1
#1010                    n + 2
#1013                    n + 3
```

```
#1100 clk = 1            n + 4
```

...so for the value change on `clk` at time `#1100`, the delta is 4.

The rcv_bit_value when the bit value is not zero or one is encoded as the position numbered 0-7 in the string "XZHUWL-?" multiplied by 2 with one added to the result. (i.e., 1, 3, 5, 7, 9, 11, 13, 15)

The variable length integer is generated with the following algorithm. It shifts the value out seven bits at a time and sets the high bit on the last byte of the variable length integer:

```
unsigned int v;   // value
char *pnt;        // destination pointer

while((nxt = v>>7))
        {
        *(pnt++) = (v&0x7f);
        v = nxt;
        }

*pnt = (v&0x7f) | 0x80;
```

Using this scheme, most value changes can be encoded in one or two (uncompressed) bytes. For the example above, the tuple (4, '1') encodes into an integer as:

(4<<2) | (1<<1) = 0x12

As a variable length integer, it is stored as a single (uncompressed) byte 0x92.

Multi-bit Encoding

Multiple bits are encoded as a variable length integer representing the time delta (without any left shifting), and immediately after that a reformatted string is encoded as packed nybbles against the MVL9 string "0XZ1HUWL-". As multi-bit strings can be of any length, the value of 15 is used to signify the end of string marker. An example:

```
#1000 val = 1010 index = n
#1001                    n + 1
#1010                    n + 2
#1013                    n + 3
#1100 val = 1zz0         n + 4
```

Thus, the tuple (4, "1ZZ0") at time #1100 is encoded bytewise as:

```
0x84 [variable length integer for delta of 4]
0x32 ["1Z": "0XZ1HUWL-"[3], "0XZ1HUWL-"[2]]
0x20 ["Z0": "0XZ1HUWL-"[2], "0XZ1HUWL-"[0]]
0xf0 [end of string marker + nybble pad to byte boundary]
```

For longer strings, this provides a 2:1 space compaction prior to calling zlib.

Reals and String Encoding

They are stored simply as (delta, null terminated string) without any re-encoding of the real or string from its ASCII representation. So for the value (4, "3.14159"), it is encoded bytewise as

```
0x84 [variable length integer for delta of 4]
0x33 0x2e 0x31 0x34 0x31 0x35 0x39 0x00 ["3.14159" with null termination]
```

Final Notes on VCD Recoding

Even with zlib compression disabled (which gtkwave allows for performance), the memory usage savings are substantial. There are several reasons for this:

- Storing VCD identifiers is completely unnecessary as the value change data is routed to its appropriate VList. Hence, the identifier implicitly is represented by the VList itself.
- Single-bit changes can be represented in only one or two bytes in most cases.
- Multi-bit changes can be represented with slightly less than half the amount of memory required normally (as the VCD identifier is no longer required).
- The amount of "next" pointers required per-VList is lg(n bytes). This allows a low overhead even when having a large number of active growable VList “streams" in memory at once.
- VList truncation when the lists are finalized at the end of the VCD file read ensure that unused VList space is returned to the operating system.

Appendix D: LXT File Format

LXT Framing

The three most important values in an LXT(interLaced eXtensible Trace) file are the following:

```
#define LT_HDRID (0x0138)
#define LT_VERSION (0x0001)
#define LT_TRLID (0xB4)
```

An LXT file starts with a two byte LT_HDRID with the two byte version number LT_VERSION concatenated onto it. The last byte in the file is the LT_TRLID. These five bytes are the only "absolutes" in an LXT file.

```
01 38 00 01 ...file body... B4
```

As one may guess from the example above, *all* integer values represented in LXT files are stored in big endian order.

Note that all constant definitions found in this appendix may be found in the header file src/helpers/lxt_write.h. Note that LXT2 files use a completely different file format as well as different constant values.

LXT Section Pointers

Preceeding the trailing ID byte `B4` is a series of tag bytes which themselves are preceeded by 32-bit offsets into the file which indicate where the sections pointed to by the tags are located. The exception is tag `00` (LT_SECTION_END) which indicates that no more tags/sections are specified:

```
00 ... offset_for_tag_2, tag_2, offset_for_tag_1, tag_1, B4
```

Currently defined tags are:

```
#define LT_SECTION_END                  (0)
#define LT_SECTION_CHG                  (1)
#define LT_SECTION_SYNC_TABLE           (2)
#define LT_SECTION_FACNAME              (3)
#define LT_SECTION_FACNAME_GEOMETRY     (4)
#define LT_SECTION_TIMESCALE            (5)
#define LT_SECTION_TIME_TABLE           (6)
#define LT_SECTION_INITIAL_VALUE        (7)
#define LT_SECTION_DOUBLE_TEST          (8)
#define LT_SECTION_TIME_TABLE64         (9)
```

Let's put this all together with an example:

The first tag encountered is `08` (`LT_SECTION_DOUBLE_TEST`) at 339. Its offset value indicates the position of the double sized floating point comparison testword. Thus, the section location for the testword is at 0309 from the beginning of the file.

```
00000300: XX XX XX XX XX XX XX XX XX 6e 86 1b f0 f9 21 09  .........n....!.
00000310: 40 00 00 00 00 04 01 00 00 02 4b 02 00 00 00 be  @.........K.....
00000320: 03 00 00 01 4b 04 00 00 03 08 05 00 00 02 8b 06  ....K...........
00000330: 00 00 03 07 07 00 00 03 09 08 b4 -- -- -- -- --  ...........
```

The next tag encountered is `07` (`LT_SECTION_INITIAL_VALUE`) at 334. Its offset value indicates the position of the simulation initial value. Even though this value is a single byte, its own section is defined. The reasoning behind this is that older versions of LXT readers would be able to skip unknown sections without needing to know the size of the section, how it functions, etc.

```
00000300: XX XX XX XX XX XX XX XX XX 6e 86 1b f0 f9 21 09  .........n....!.
00000310: 40 00 00 00 00 04 01 00 00 02 4b 02 00 00 00 be  @.........K.....
00000320: 03 00 00 01 4b 04 00 00 03 08 05 00 00 02 8b 06  ....K...........
00000330: 00 00 03 07 07 00 00 03 09 08 b4 -- -- -- -- --  ...........
```

The next tag encountered is 06 (LT_SECTION_TIME_TABLE) at 32F. Its offset value (the underlined four byte number) indicates the position of the time table which stores the time value vs positional offset for the value change data.

```
00000300: XX XX XX XX XX XX XX XX XX 6e 86 1b f0 f9 21 09  .........n....!.
00000310: 40 00 00 00 00 04 01 00 00 02 4b 02 00 00 00 be  @.........K.....
00000320: 03 00 00 01 4b 04 00 00 03 08 05 00 00 02 8b 06  ....K...........
00000330: 00 00 03 07 07 00 00 03 09 08 b4 -- -- -- -- --  ...........
```

The next tag encountered is 05 (LT_SECTION_TIMESCALE) at 32A. Its offset value indicates the position of the timescale byte.

```
00000300: XX XX XX XX XX XX XX XX XX 6e 86 1b f0 f9 21 09  .........n....!.
00000310: 40 00 00 00 00 04 01 00 00 02 4b 02 00 00 00 be  @.........K.....
00000320: 03 00 00 01 4b 04 00 00 03 08 05 00 00 02 8b 06  ....K...........
00000330: 00 00 03 07 07 00 00 03 09 08 b4 -- -- -- -- --  ...........
```

The next tag encountered is 04 (LT_SECTION_FACNAME_GEOMETRY) at 325. Its offset value indicates the geometry (array/msb/lsb/type/etc) of the dumped facilities (signals) in the file.

```
00000300: XX XX XX XX XX XX XX XX XX 6e 86 1b f0 f9 21 09  .........n....!.
00000310: 40 00 00 00 00 04 01 00 00 02 4b 02 00 00 00 be  @.........K.....
00000320: 03 00 00 01 4b 04 00 00 03 08 05 00 00 02 8b 06  ....K...........
00000330: 00 00 03 07 07 00 00 03 09 08 b4 -- -- -- -- --  ...........
```

The next tag encountered is 03 (LT_SECTION_FACNAME) at 320. Its offset value indicates where the compressed facility names are stored.

```
00000300: XX XX XX XX XX XX XX XX XX 6e 86 1b f0 f9 21 09  .........n....!.
00000310: 40 00 00 00 00 04 01 00 00 02 4b 02 00 00 00 be   @.........K.....
00000320: 03 00 00 01 4b 04 00 00 03 08 05 00 00 02 8b 06  ....K...........
00000330: 00 00 03 07 07 00 00 03 09 08 b4 -- -- -- -- --  ...........
```

The next tag encountered is 02 (LT_SECTION_SYNC_TABLE) at 31B. Its offset value points to a table where the final value changes for each facility may be found.

```
00000300: XX XX XX XX XX XX XX XX XX 6e 86 1b f0 f9 21 09  .........n....!.
00000310: 40 00 00 00 00 04 01 00 00 02 4b 02 00 00 00 be  @.........K.....
00000320: 03 00 00 01 4b 04 00 00 03 08 05 00 00 02 8b 06  ....K...........
00000330: 00 00 03 07 07 00 00 03 09 08 b4 -- -- -- -- --  ...........
```

The next tag encountered is 01 (LT_SECTION_CHG) at 316. Its offset value points to the actual value changes in the file.

```
00000300: XX XX XX XX XX XX XX XX XX 6e 86 1b f0 f9 21 09  .........n....!.
00000310: 40 00 00 00 00 04 01 00 00 02 4b 02 00 00 00 be  @.........K.....
00000320: 03 00 00 01 4b 04 00 00 03 08 05 00 00 02 8b 06  ....K...........
00000330: 00 00 03 07 07 00 00 03 09 08 b4 -- -- -- -- --  ...........
```

The final tag encountered is `00` at 311. It signifies that there are no more tags.

```
00000300: XX XX XX XX XX XX XX XX XX 6e 86 1b f0 f9 21 09  .........n....!.
00000310: 40 00 00 00 00 04 01 00 00 02 4b 02 00 00 00 be  @.........K.....
00000320: 03 00 00 01 4b 04 00 00 03 08 05 00 00 02 8b 06  ....K...........
00000330: 00 00 03 07 07 00 00 03 09 08 b4 -- -- -- -- --  ..........
```

Note that with the exception of the termination tag `00`, tags may be encountered in any order. The fact that they are encountered in monotonically decreasing order in the example above is an implementation detail of the lxt_write dumper. Code which processes LXT files should be able to handle tags which appear in any order. For tags which are defined multiple times, it is to be assumed that the tag instance closest to the termination tag is the one to be used unless each unique instantiation possesses a special meaning. Currently, repeated tags have no special semantics.

LXT Section Definitions

The body of each section (as currently defined) will now be explained in detail.

08: LT_SECTION_DOUBLE_TEST

This section is only present if double precision floating point data is to be found in the file. In order to resolve byte ordering issues across various platforms, a rounded version of *pi* (3.14159) is stored in eight consecutive bytes. This value was picked because each of its eight bytes are unique. It is the responsibility of an LXT reader to compare the byte ordering found in the `LT_SECTION_DOUBLE_TEST` section to that of the same rounded version of *pi* as represented by reader's processor. By comparing the position on the host and in the file, it may be determined how the values stored in the LXT file need to be rearranged. The following bit of code shows one possible implementation for this:

```
static char double_mask[8]={0,0,0,0,0,0,0,0};
static char double_is_native=0;

static void create_double_endian_mask(double *pnt)
{
static double p = 3.14159;
```

```
double d;
int i, j;

d= *pnt;
if(p==d)
        {
        double_is_native=1;
        }
        else
        {
        char *remote, *here;

        remote = (char *)&d;
        here = (char *)&p;
        for(i=0;i<8;i++)
                {
                for(j=0;j<8;j++)
                        {
                        if(here[i]==remote[j])
                                {
                                double_mask[i]=j;
                                break;
                                }
                        }
                }
        }
}
```

If `double_is_native` is zero, the following function will then be needed to be called to rearrange the file byte ordering to match the host every time a double is encountered in the value change data:

```
static char *swab_double_via_mask(double *d)
{
char swapbuf[8];
char *pnt = malloc(8*sizeof(char));
int i;

memcpy(swapbuf, (char *)d, 8);
for(i=0;i<8;i++)
        {
        pnt[i]=swapbuf[double_mask[i]];
        }

return(pnt);
}
```

07: LT_SECTION_INITIAL_VALUE

This section is used as a "shortcut" representation to flash all facilities in a dumpfile to a specific value at the initial time. Permissible values are '0', '1', 'Z', 'X', 'H', 'U', 'W', 'L', and '-' stored as the byte values 00 through 08 in the LXT file.

06: LT_SECTION_TIME_TABLE / 08: LT_SECTION_TIME_TABLE64

This section marks the time vs positional data for the LXT file. It is represented in the following format:

4 bytes: *number of entries (n)*
4 bytes: *min time in dump* (8 bytes for `LT_SECTION_TIME_TABLE64`)
4 bytes: *max time in dump* (8 bytes for `LT_SECTION_TIME_TABLE64`)

n 4-byte positional delta entries follow
n 4-byte time delta entries follow (8 byte entries for `LT_SECTION_TIME_TABLE64`)

It is assumed that the delta values are represented as *current_value - previous_value*, which means that deltas should always be positive. In addition, the *previous_value* for delta number zero for both position and time is zero. This will allow for sanity checking between the time table and the min/max time fields if it is so desired or if the min/max fields are needed before the delta entries are traversed.

Example:

```
00000005    5 entries are in the table
00000000    Min time of simulation is 0
00000004    Max time of simulation is 4.

00000000    time[0]=0
00000001    time[1]=1
00000001    time[2]=2
00000001    time[3]=3
00000001    time[4]=4

00000004    pos[0]=0x4
00000010    pos[1]=0x14
00000020    pos[2]=0x34
00000002    pos[3]=0x36
00000300    pos[4]=0x336
```

05: LT_SECTION_TIMESCALE

This section consists of a single signed byte. Its value (*x*) is the exponent of a base-10 timescale. Thus, each increment of '1' in the time value table represented in the previous section represents 10^{x} seconds. Use -9 for nanoseconds, -12 for picoseconds, etc. Any eight-bit signed value (-128 to +127) is permissible, but in actual practice only a handful are useful.

03: LT_SECTION_FACNAME

No, section `04: LT_SECTION_FACNAME_GEOMETRY` hasn't been forgotten. It's more logical to cover the facilities themselves before their geometries.

4 bytes: *number of facilities (n)*
4 bytes: *amount of memory required in bytes for the decompressed facilities*

n compressed facilities follow, where a compressed facility consists of two values:

2 bytes: *number of prefix bytes* (min=0, max=65535)
zero terminated string: *suffix bytes*

An example should clarify things (prefix lengths are in bold):

```
00000020: 00 00 00 04 00 00 00 1d 00 00 61 6c 70 68 61 00  ..........alpha.
00000030: 00 01 70 70 6c 65 00 00 04 69 63 61 74 69 6f 6e  ..pple...ication
00000040: 00 00 00 7a 65 72 6f 00 00 00 00 01 00 00 00 00  ...zero.........
```

Four facilities (underlined) are defined and they occupy 0x0000001d bytes (second underlined value).

This first prefix length is 0000 (offset 28).
The first suffix is "alpha", therefore the first facility is "alpha". This requires six bytes.

The second prefix length is 0001 (offset 30).
The second suffix is "pple", therefore the second facility is "apple". This requires six bytes.

The third prefix length is 0004 (offset 37).
The third suffix is "ication", therefore the third facility is "application". This requires twelve bytes.

The fourth prefix length is 0000 (offset 41).
The fourth suffix is "zero", therefore the fourth facility is "zero". This requires five bytes.

6 + 6 + 12 + 5 = 29 which indeed is 0x1d.

It is suggested that the facilities are dumped out in alphabetically sorted order in order to increase the compression ratio of this section.

04: LT_SECTION_FACNAME_GEOMETRY

This section consists of a repeating series of sixteen byte entries. Each entry corresponds in order with a facility as defined in `03: LT_SECTION_FACNAME`. As such there is a 1:1 in-order correspondence between the two sections.

4 bytes: *rows* (typically zero, only used when defining arrays: this indicates the max row value+1)
4 bytes: *msb*
4 bytes: *lsb*
4 bytes: *flags*

flags are defined as follows:

```
#define LT_SYM_F_BITS         (0)
#define LT_SYM_F_INTEGER      (1<<0)
#define LT_SYM_F_DOUBLE       (1<<1)
#define LT_SYM_F_STRING       (1<<2)
#define LT_SYM_F_ALIAS        (1<<3)
```

When an `LT_SYM_F_ALIAS` is encountered, it indicates that the rows field instead means "alias this to facility number *rows*", there the facility number corresponds to the definition order in `03: LT_SECTION_FACNAME` and starts from zero.

02: LT_SECTION_SYNC_TABLE

This section indicates where the final value change (as a four-byte offset from the beginning of the file) for every facility is to be found. Facilities which do not change value contain a value of zero for their final value change. This section is necessary as value changes are stored as a linked list of backward directed pointers. There is a 1:1 in-order correspondence between this section and the definitions found in `LT_SECTION_FACNAME`.

4 bytes: *final offset for facility* (repeated for each facility in order as they were defined)

01: LT_SECTION_CHG

This section is usually the largest section in a file as is composed of value changes, however its structure is set up such that individual facilities can be quickly accessed without the necessity of reading in the entire file. In spite of this format, this does not prevent one from stepping through the entire section *backwards* in order to process it in one pass. The method to achieve this will be described later.

The final offset for a value change for a specific facility is found in `02: LT_SECTION_SYNC_TABLE`. Since value changes for a facility are linked together,

one may follow pointers backward from the sync value offset for a facility in order to read in an entire trace. This is used to accelerate sparse reads of an LXT file's change data such as those that a visualization tool such as a wave viewer would perform.

The format for a value change is as follows:

command_byte, delta_offset_of_previous_change[, [row_changed,] change_data]

Command Bytes

The *command_byte* is broken into two (as currently defined) bitfields: bits [5:4] contain a byte count (minus one) required for the *delta_offset_of_previous_change* (thus a value from one to four bytes), and bits [3:0] contain the command used to determine the format of the change data (if any change data is necessary as dictated by the command byte).

Bits [3:0] of the *command_byte* are defined as follows. Note that this portion of the *command_byte* is ignored for strings and doubles and typically x0 is placed in the dumpfile in those cases:

`0`	MVL_2 [or default (for datatype) value change follows]
`1`	MVL_4
`2`	MVL_9
`3`	flash whole value to 0s
`4`	flash whole value to 1s
`5`	flash whole value to Zs
`6`	flash whole value to Xs
`7`	flash whole value to Hs
`8`	flash whole value to Us
`9`	flash whole value to Ws
`A`	flash whole value to Ls
`B`	flash whole value to -s
`C`	clock compress use 1 byte repeat count
`D`	clock compress use 2 byte repeat count
`E`	clock compress use 3 byte repeat count
`F`	clock compress use 4 byte repeat count

Commands x3-xB only make sense for MVL_2/4/9 (and integer in the case for x3 and x4 when an integer is 0 or ~0) facilities. They are provided as a space saving device which obviates the need for dumping value change data when all the bits in a facility are set to the same exact value. For single bit facilities, these commands suffice in all cases.

Command x0 is used when *change_data* can be stored as MVL_2. Bits defined in

MVL_2 are '0' and '1' and as such, one bit of storage in an LXT file corresponds to one bit in the facility value.

Command x1 is used when *change_data* can't be stored as MVL_2 but can stored as MVL_4. Bits defined in MVL_4 are '0', '1', 'Z', and 'X' are stored in an LXT file as the two-bit values 00, 01, 10, and 11.

Command x2 is used when change_data can't be stored as either MVL_2 or MVL_4 but can be stored as MVL_9. Bits defined in MVL_9 are '0', '1', 'Z', 'X', 'H', 'U', 'W', 'L', and '-' corresponding to the four-bit values 0000 through 1000.

Commands xC-xF are used to repeat a clock. It is assumed that at least two clock phase changes are present before the current command. Their time values are subtracted in order to obtain a delta. The delta is used as the change time between future phase changes with respect to the time value of the previous command which is used as a "base time" and "base value" for repeat_count+1 phase changes.

Note that these repeat command nybbles *also* are applicable to multi-bit facilities which are 32-bits or less and MVL_2 in nature. In this case, the preceeding two deltas are subtracted such that a recurrence equation can reconstruct any specific item of the compressed data:

```
unsigned int j   = item_in_series_to_create + 1;
unsigned int res = base + (j/2)*rle_delta[1] + ((j/2)+(j&1))*rle_delta[0];
```

For a sequence of: `7B 7C 7D 7E 7F 80 81 82` ...

```
base = 82
rle_delta[1] = 82 - 81 == 01
rle_delta[0] = 81 - 80 == 01
```

Two deltas are used in order to handle the case where a vector which changes value by a constant XOR. In that case, the `rle_delta` values will be different. In this way, one command encoding can handle both XOR and incrementer/decrementer type compression ops.

Delta Offsets

Delta offsets indicate where the preceeding change may be found with respect to the beginning of the LXT file. In order to calculate where the preceeding change for a facility is, take the offset of the *command_byte*, subtract the *delta_offset_of_previous_change* from it, then subtract 2 bytes more. As an example:

`00001000: 13 02 10` ...

The command byte is 13. Since bits [5:4] are "01", this means that the *delta_offset_of_previous_change* is two bytes since 1 + 1 = 2.

The next two bytes are 0210, so 1000 - 0210 - 2 = 0DEE. Hence, the preceeding value change can be found at 0DEE. This process is to be continued until a value change offset of 0 is obtained. This is impossible because of the existance of the LXT header bytes.

Row Changed

This field is *only* present in value changes for arrays. The value is 2, 3, or 4 bytes depending on the magnitude of the array size: greater than 16777215 rows requires 4 bytes, greater than 65535 requires 3 bytes, and so on down to one byte. Note that any value type can be part of an array.

Change Data

This is only present for *command_byte*s x0-x2 for MVL_2, MVL_4, and MVL_9 data, and any *command_byte* for strings and doubles. Strings are stored in zero terminated format and doubles are stored as eight bytes in machine-native format with `08: LT_SECTION_DOUBLE_TEST` being used to resolve possible differences in endianness on the machine reading the LXT file.

Values are stored left justified in big endian order and unused bits are zeroed out. Examples with "_" used to represent the boundary between consecutive bytes:

MVL_2: "0101010110101010" (16 bits) is stored as 01010101_10101010
MVL_2: "011" (3 bits) is stored as 01100000
MVL_2: "11111110011" (11 bits) is stored as 11111110_01100000

MVL_4: "01ZX01ZX" (8 bits) is stored as 00011011_00011011
MVL_4: "ZX1" (3 bits) is stored as 10110100
MVL_4: "XXXXZ" (5 bits) is stored as 11111111_10000000

MVL_9: "01XZHUWL-" (9 bits) is stored as
00000001_00100011_01000101_01100111_10000000

Correlating Time Values to Offsets

This is what the purpose of `06: LT_SECTION_TIME_TABLE` is. Given the offset of a *command_byte*, `bsearch(3)` an array of ascending position values (not deltas)

and pick out the maximum position value which is less than or equal to the offset of the *command_byte*. The following code sequence illustrates this given two arrays `positional_information[]` and `time_information[]`. Note that `total_cycles` corresponds to *number_of_entries* as defined in `06: LT_SECTION_TIME_TABLE`.

```
static int max_compare_time_tc, max_compare_pos_tc;
static int compar_mvl_timechain(const void *s1, const void *s2)
{
int key, obj, delta;
int rv;

key=*((int *)s1);
obj=*((int *)s2);

if((obj<=key)&&(obj>max_compare_time_tc))
        {
        max_compare_time_tc=obj;
        max_compare_pos_tc=(int *)s2 - positional_information;
        }

delta=key-obj;
if(delta<0) rv=-1;
else if(delta>0) rv=1;
else rv=0;

return(rv);
}

static int bsearch_position_versus_time(int key)
{
max_compare_time_tc=-1; max_compare_pos_tc=-1;

bsearch((void *)&key, (void *)positional_information, total_cycles, sizeof(int),
compar_mvl_timechain);
if((max_compare_pos_tc<=0)||(max_compare_time_tc<0))
        {
        max_compare_pos_tc=0; /* aix bsearch fix */
        }

return(time_information[max_compare_pos_tc]);
}
```

Reading All Value Changes in One Pass

This requires a little bit more work but it can be done. Basically what you have to do is the following:

1. Read in all the sync offsets from `02: LT_SECTION_SYNC_TABLE` and put each in a structure which contains the sync offset and the facility index. All of these structures will compose an array that is as large as the number of facilities which exist.

2. Heapify the array such that the topmost element of the heap has the largest positional offset.
3. Change the topmost element's offset to its preceeding offset (as determined by examining the *command_byte*, bits [5:4] and calculating the preceeding offset by subtracting the *delta_offset_of_previous_change* then subtracting 2 bytes.
4. Continue with step 2 until the topmost element's offset is zero after performing a heapify().

00: LT_SECTION_END

As a section pointer doesn't exist for this, there's no section body either.

The lxt_write API

In order to facilitate the writing of LXT files, an API has been provided which does most of the hard work.

```
struct lt_trace *lt_init(const char *name)
```

This opens an LXT file. The pointer returned by this function is NULL if unsuccessful. If successful, the pointer is to be used as a "context" for all the remaining functions in the API. In this way, multiple LXT files may be generated at once.

```
void lt_close(struct lt_trace *lt)
```

This fixates and closes an LXT file. This is extremely important because if the file is not fixated, it will be impossible to use the value change data in it! For this reason, it is recommended that the function be placed in an `atexit(3)` handler in environments where trace generation can be interrupted through program crashes or external signals such as control-C.

```
struct lt_symbol *lt_symbol_add(struct lt_trace *lt, const char *name, unsigned
int rows, int msb, int lsb, int flags)
```

This creates a facility. Since the facility and related tables are written out during fixation, one may arbitrarily add facilities up until the very moment that lt_close() is called. For facilities which are not arrays, a value of 0 or 1 for rows. As such, only values 2 and greater are used to signify arrays. Flags are defined above as in 04: LT_SECTION_FACNAME_GEOMETRY.

```
struct lt_symbol *lt_symbol_find(struct lt_trace *lt, const char *name)
```

This finds if a symbol has been previously defined. If returns non-NULL on success. It actually returns a symbol pointer, but you shouldn't be deferencing the fields inside it unless you know what you are doing.

```
struct lt_symbol *lt_symbol_alias(struct lt_trace *lt, const char *existing_name,
const char *alias, int msb, int lsb)
```

This assigns an alias to an existing facility. This is to create signals which traverse multiple levels of hierarchy, but are the same net, possibly with different MSB and LSB values (though the distance between them will be the same).

```
void lt_symbol_bracket_stripping(struct lt_trace *lt, int doit)
```

This is to be used when facilities are defined in Verilog format such that exploded bitvectors are dumped as x[0], x[1], x[2], etc. If doit is set to a nonzero value, the bracketed values will be stripped off. In order to keep visualization and other tools from becoming confused, the MSB/LSB values must be unique for every bit. The tool `vcd2lxt` shows how this works and should be used. If vectors are dumped atomically, this function need not be called.

```
void lt_set_timescale(struct lt_trace *lt, int timescale)
```

This sets the simulation timescale to $10^{timescale}$ seconds where timescale is an 8-bit signed value. As such, negative values are the only useful ones.

```
void lt_set_initial_value(struct lt_trace *lt, char value)
```

This sets the initial value of every MVL (bitwise) facility to whatever the value is. Permissible values are '0', '1', 'Z', 'X', 'H', 'U', 'W', 'L', and '-'.

```
int lt_set_time(struct lt_trace *lt, unsigned int timeval)
int lt_inc_time_by_delta(struct lt_trace *lt, unsigned int timeval)
```

```
int lt_set_time64(struct lt_trace *lt, lxttime_t timeval)
int lt_inc_time_by_delta64(struct lt_trace *lt, lxttime_t timeval)
```

This is how time is dynamically updated in the LXT file. Note that for the non-delta functions, timeval changes are expected to be monotonically increasing. In addition, time values dumped to the LXT file are coalesced if there are no value changes for a specific time value. (Note: lxttime_t is defined as an unsigned long long.)

```
void lt_set_clock_compress(struct lt_trace *lt)
```

Enables clock compression heuristics for the current trace. This cannot be turned off once it is on.

```
int lt_emit_value_int(struct lt_trace *lt, struct lt_symbol *s, unsigned int row,
int value)
```

This dumps an MVL_2 value for a specific facility which is 32-bits or less. Note that this does not work for strings or doubles.

```
int lt_emit_value_double(struct lt_trace *lt, struct lt_symbol *s, unsigned int
row, double value)
```

This dumps a double value for a specific facility. Note that this only works for doubles.

```
int lt_emit_value_string(struct lt_trace *lt, struct lt_symbol *s, unsigned int
row, char *value)
```

This dumps a string value for a specific facility. Note that this only works for strings.

```
int lt_emit_value_bit_string(struct lt_trace *lt, struct lt_symbol *s, unsigned
int row, char *value)
```

This dumps an MVL_2, MVL_4, or MVL_9 value out to the LXT file for a specific facility. Note that the value is parsed in order to determine how to optimally represent it in the file. In addition, note that if the value's string length is shorter than the facility length, it will be left justified with the rightmost character will be propagated to the right in order to pad the value string out to the correct length. Therefore, "10x" for 8-bits becomes "10xxxxxx" and "z" for 8-bits becomes "zzzzzzzz".

Index

Illustration Index

Figure 1: The GTKWave main window.....19
Figure 2: The main window with an embedded SST.....20
Figure 3: Signal subwindow with scrollbar and an “open” collapsible trace.....21
Figure 4: Signal subwindow with no hidden area from left to right.....21
Figure 5: Signal subwindow with left justified signal names.....22
Figure 6: A typical view of the wave subwindow.....23
Figure 7: An example of both positively and negatively timeshifted traces.....24
Figure 8: The Navigation and Status Panel.....24
Figure 9: TwinWave managing two GTKWave sessions in a single window.....26
Figure 10: The RTLBrowse RTL Design Hierarchy window.....28
Figure 11: Source code annotated by RTLBrowse.....29
Figure 12: The File submenu.....31
Figure 13: The Edit submenu.....32
Figure 14: The Search submenu.....36
Figure 15: The Time submenu.....37
Figure 16: The Markers submenu.....38
Figure 17: The View submenu.....39
Figure 18: The Help submenu.....40
Figure 19: The main window with viewer state loaded from a save file.....43
Figure 20: The Signal Search (regular expression search) Requester.....44
Figure 21: The Hierarchy Search Requester.....45
Figure 22: The Signal Search Tree Requester.....46
Figure 23: The Pattern Search Requester.....47

Alphabetical Index

accel.....**69**
accessing of files.....**31**

AET2....**16**
AET2 reader API....**16**
Alias Files....**47**
Alias Highlighted Trace....**33**
All Events Trace....**16**
Alphabetize....**35**
alt_hier_delimeter....69
Analog....**34**
Analog traces....**20**
Append....**44**
append_vcd_hier....70
application respawning....**31**
ASCII....**34**
atomic_vectors....70
attaching disassemblers....**32**
autocoalesce....**36,** 70
autocoalesce_reversal....70
automatic conversion of VCD files....**42**
Autoname Bundles....**36**
autoname_bundles....70
Base Time label....**25**
baseline marker....**23pp., 37**
Binary....**34**
BitsToReal....**34**
bundling....**36**
bzip2....**11**
Center Zooms....**39**
Children box....**45**
click-dragged....**25**
Collapse All Groups....**35**
Collapsible groups....**21**
Collect All Named Markers....**38**
Collect Named Marker....**38**
Combine Down....**33**
Combine Up....**33**
comment trace....**21**
Compiling....**11**
compressibility of VCD files....**42**
Constant Marker Update....**39**
constant_marker_update....72
convert_to_reals....72
Current Time label....**25**
current version....**40**
cursor_snap....72
Cut....**33**
Cygserver....**12**

Cygwin....**12**
data representation of values....**32**
Decimal....**33**
deep import....**45**
Delete Primary Marker....**38**
delta time....**25**
disable_mouseover....**72**
disable_tooltips....72
Discard....**38**
do_initial_zoom_fit....72
Drag and Drop....**20, 22**
drag warp....**35**
Draw Roundcapped Vectors....**39**
Drop Named Marker....**38**
dumpfiles....**15**
Dynamic Resize....**39**
dynamic tooltip....**39**
dynamic_resizing....72
enable_fast_exit....72
enable_ghost_marker....72
enable_horiz_grid....73
enable_vcd_autosave....73
enable_vert_grid....73
enums....**34**
Exclude....**35**
Expand....**33**
Expand All Groups....**35**
facilities....**24**
Fetch....**37**
file conversion....**18, 51**
Filetype Conversion....**56p., 59pp., 63, 66**
fontname_logfile....73
fontname_signals....73
fontname_waves....73
force_toolbars....73
FrameMaker....**18**
Full Precision....**40**
GHDL....**16**
GHDL Wavefile....**16**
GHW....**16**
glitch....**57, 62, 64, 66, 76**
GNU GPL General Public License....**4**
gperf....**11**
GTK....**11**
GtkPlug....**53**
GtkSocket....**53**

GTKWave scope state....**42**
gtkwave.ini **69**
helper applications....**16**
Hex....**33**
hide_sst....73
hier_delimeter....73
hier_grouping....73
hier_max_level....74
Hierarchy Search....**45**
Highlight All....**35**
Highlight Regexp....**35**
highlighted trace....**21**
hpane_pack....74
Icarus Verilog....**15, 41**
importing signals....**44**
indirect facs file....**52**
initial_window_x....74
initial_window_xpos....74
initial_window_y....74
initial_window_ypos....74
Insert....**44**
Insert Analog Height Extension....**32**
Insert Blank....**32**
Insert Comment....**32**
Installing....**11**
interactive....**53**
interactive VCD....30, **68**
InterLaced eXtensible Trace....**15**
Introduction....**15**
Invert....**34**
Left Justify Signals....**39**
left mouse button....**21pp., 30, 35, 39**
left_justify_sigs....74
legacy VCD mode....**53**
lettered markers....**23**
LXT....**15**
LXT Clock Compress to Z....**40**
LXT File Format....**82**
LXT Framing....**82**
LXT Section Definitions....**85**
LXT Section Pointers....**83**
lxt_clock_compress_to_z....74
LXT2....**15**
LXT2/VZT block skip....**52**
lxt2miner....**55**
lxt2vcd....**56**

magnifying glass icons....**24**
MagniVu....**60**
Main Window....**19**
Mark Count....**36**
Marker time label....**25**
Matches box....**44**
maximum hierarchy depth....**32**
menu accelerator keys, replacement of....**69**
Microsoft Windows Operating Systems....**12**
middle mouse button....**23, 70**
MinGW environment....**12**
Missing modules....**27**
Move To Time....**37**
multiprocessor machines....**16**
mvl2lxt....**57**
mvl2vcd....**57**
MVL9....79
Navigation and Status Panel....**24**
nomenus....52
nowm....52
Octal....**34**
Open New Viewer....**31**
override .gtkwaverc filename....**51**
Overview....**15**
Page....**38**
page_divisor....74
partial zip mode....**62**
Paste....**33**
pattern marks....**47**
Pattern Search....**36, 46**
PCCTS....**11**
pipe....30
plug-in....**53**
POSIX filter....**45**
POSIX regular expression....**35p., 44p.**
PostScript....**18**
primary marker....**21, 23pp., 27, 30, 32, 36pp., 58, 73**
Print To File....**31**
printing....**31**
ps_maxveclen....74
Quit....**32**
Range....**44**
Read Logfile....**32**
Read Save File....**31**
Read Verilog Stemsfile....**32**
redirected VCD....**68**

Reduce Single Bit Vectors....**33**
Remove Highlighted Aliases....**33**
Remove Pattern Marks....**40**
Replace....**44**
Reverse....**35**
Reverse Bits....**34**
Right Justify....**34**
Right Justify Signals....**39**
right mouse button....**22p., 30**
RTLBrowse....**27**
sample Verilog design....**41**
script file....**53**
Scroll Wheels....**30**
Search Hierarchy Grouping....**36**
Set Max Hier....**32**
shared memory ID....**59**
sharp edges....**39**
Shift....**38**
Shift-End....**22**
Shift-Home....**22**
shmidcat....30, 53, **67**
Show....**35**
Show Base Symbols....**39**
Show Grid....**39**
Show Mouseover....**39**
show_base_symbols....75
show_grid....75
Show-Change All Highlighted....**34**
Show-Change First Highlighted....**35**
Show-Change Marker Data....**38**
Signal Hierarchy box....**45**
Signal Save Files....**46**
Signal Search....**44**
Signal Search Hierarchy....**36**
Signal Search Regexp....**36**
Signal Search Tree....**36**
Signed....**33**
Sigsort....**35**
sloping edges....**39**
splash_disable....75
spring back....**24**
sst_expanded....75
status window....**19, 24**
stems file....**18, 27, 32, 42, 52, 59, 65**
Strand....**44**
superuser....**11**

Tektronix logic analyzers....**60**
tex2vcd....**59**
The lxt_write API....**94**
time measurements....**23**
timeshift....**24**
tla2vcd....**59**
Toggle Max-Marker....**39**
top-level hierarchy clutter....**52**
trace color....70
Translate Filter File....**34**
Translate Filter Process....**34**
Tree Search....**45**
TWINWAVE....**26, 52, 54p.**
UnHighlight All....**35**
UnHighlight Regexp....**35**
Unix and Linux Operating Systems....**11**
Unwarp....**35**
use_big_fonts....75
use_full_precision....75
use_maxtime_display....75
use_nonprop_fonts....75
use_roundcaps....75
use_scrollbar_only....75
utility functions....**32**
Value Change Dump....**15**
value tooltip....**72**
variable length integer....**79**
VCD....**15**
VCD recoder....**53**
VCD Recoding....**77**
vcd_explicit_zero_subscripts....75
vcd_preserve_glitches....76
vcd_warning_filesize....76
vcd2lxt....**60**
vcd2lxt2....**61**
vcd2vzt....**63**
vector_padding....76
Verilog Zipped Trace....**16**
vertex....**64**
vertical blue lines....**20**
VisualC++....**12**
VList....76, **77**
vlist_compression....76
VZT....**16**
vzt2vcd....**65**
vztminer....**66**

Warp....**35**
Wave Help....**40**
Wave Scrolling....**38**
Wave Subwindow....**23**
Wave Version....**40**
wave_scrolling....76
window manager....**74**
WRange....**44**
Write LXT File As....**31**
Write Save File....**32**
Write Save File As....**32**
Write VCD File As....**31**
WStrand....**44**
XID....**53**
zlib....**11,** 77
Zoom....**37**
Zoom Amount....**37**
Zoom Base....**37**
Zoom Pow10 Snap....**40**
zoom_base....76
zoom_center....76
zoom_pow10_snap....76
.gtkwaverc....**69**
'#'....**39**
'%'....**39**
'$'....**39**
"[]"....**36**
“--”....**25**
“..”....**45**
“(+)” prefix....**45**
“[MISSING]”....**27**
“String” searches....**47**