GTKWave 3.3 Wave Analyzer User's Guide

User's Guide
GTKWave

Updated May 31, 2011.
This manual supports GTKWave 3.3.22 and higher versions.

Contents

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, however the system ones will be used if located. The PCCTS-1.33 source code (required for compiling *vermin*) 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-3.1.3> ./configure
```

Use the `--help` flag to see which options are available. Typically, outside of `--prefix`, no flags are needed.

```
2 :/tmp/gtkwave-3.1.3> make
```

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

```
3 :/tmp/gtkwave-3.1.3> su
Password:
[root@localhost gtkwave-3.1.3]# make install
```

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

```
[root@localhost gtkwave-3.1.3]# 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. (http://www.cygwin.com/cygwin-ug-net/using-cygserver.html)

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.

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. Pre-made
binaries can be found at the http://www.dspia.com/gtkwave.html website.

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 89.) 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 Wave file. This is a nine state ("01XZHUWL-") file format written by the VHDL simulator GHDL.
- AET2: All Events Trace Version 2. This is a 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 tool sets can modify the *configure* script to point to the *libae2rw.a* and/or *libae2rw.so* files in order to enable this feature.
- IDX: VCD Recoder Index File. This format is written by GTKWave when instructed to generate fastload files.
- FST: Fast Signal Trace. This format is a block-based variant of IDX which is designed for very fast sequential and random access.

Converter helper applications are packaged with the viewer in order to convert VCD files into LXT, LXT2, VZT, or FST files. Conversion from LXT2, VZT, and FST 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) and FST have 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 *vermin*. 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.
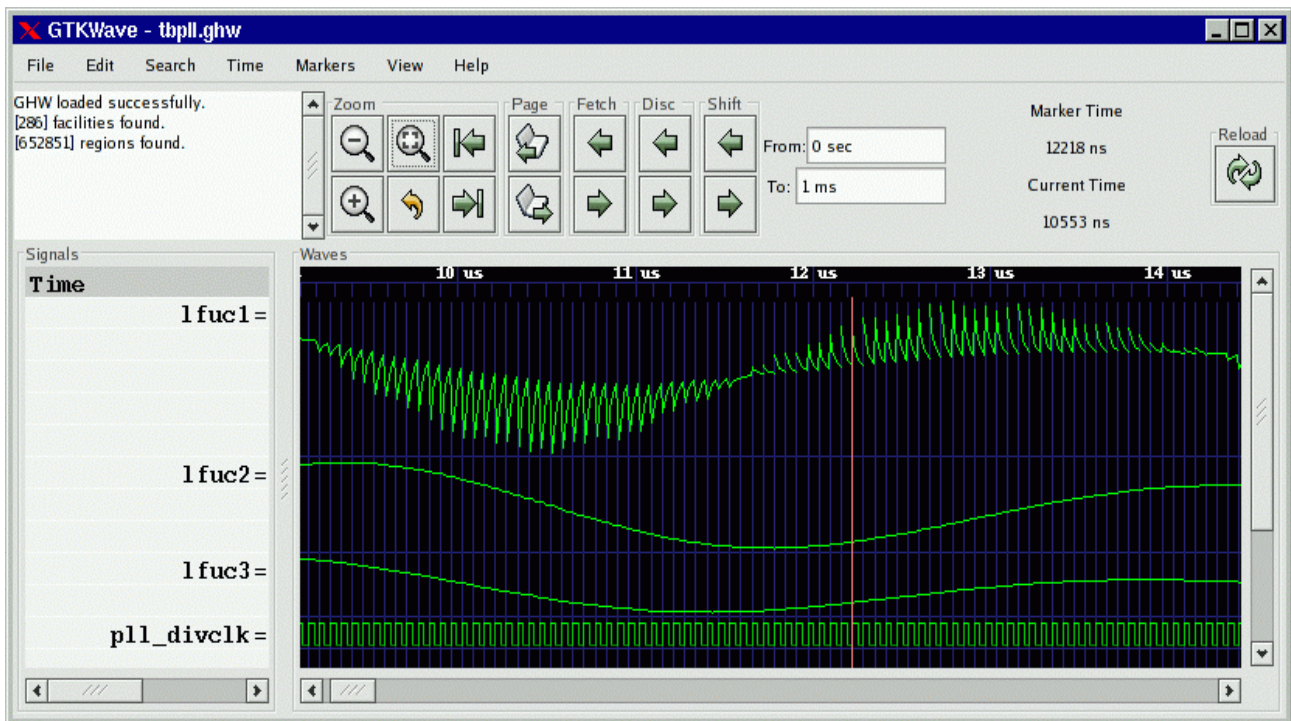
Figure 1: The GTKWave main window

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.
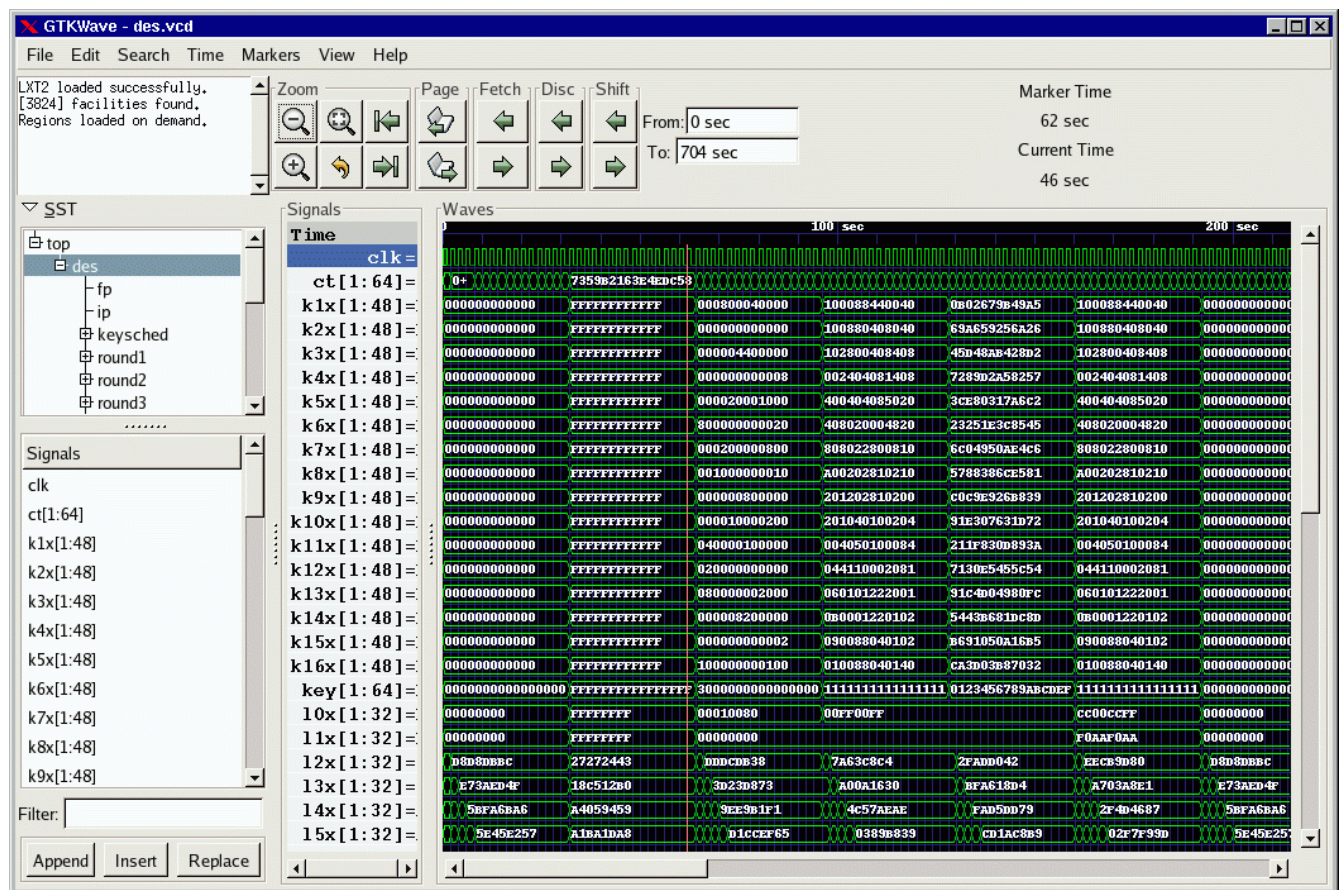
Figure 2: The main window with an embedded SST

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.)

Toolbutton Interface

The use_toolbutton_interface rc variable controls how the user interface appears. Recent versions of the viewer have this variable set to "on" which modifies the viewer to use GTK themes and a more compact button layout as shown below.
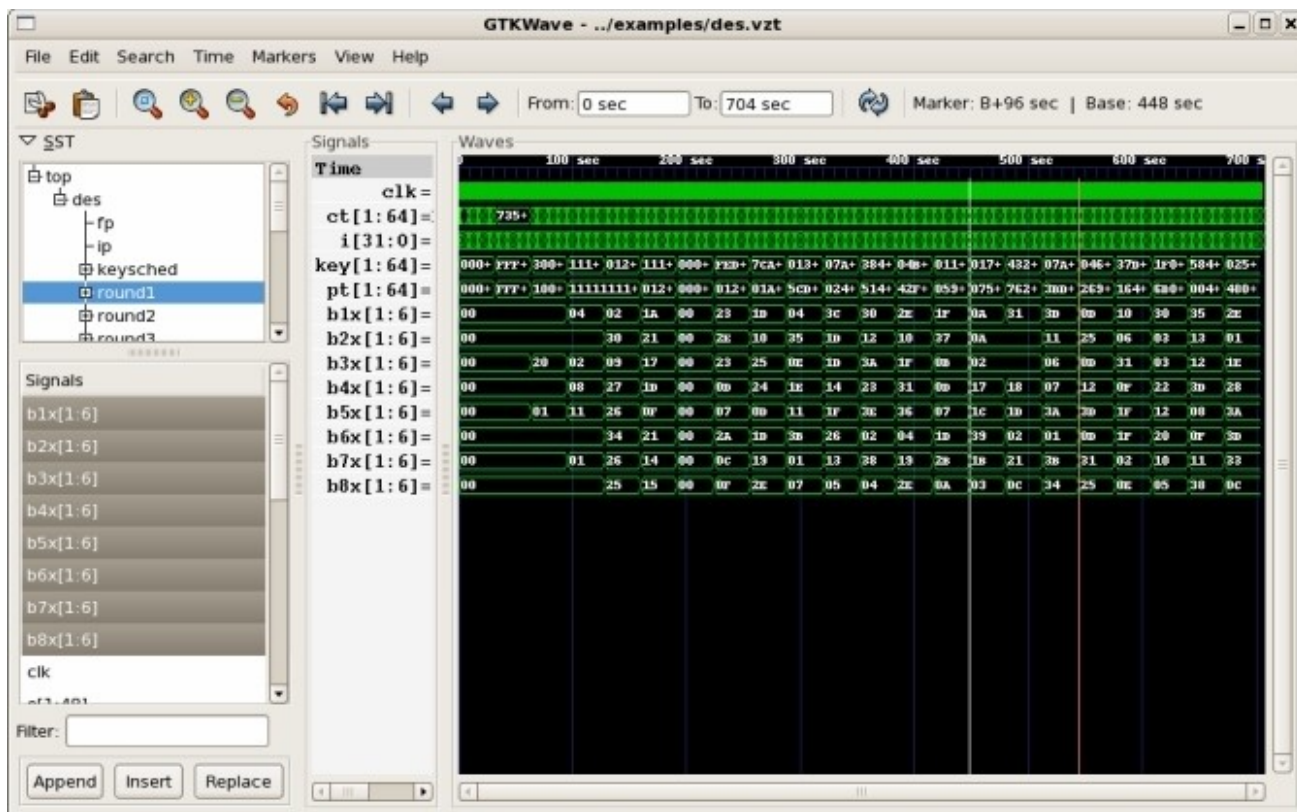
Figure 3: The main window using the toolbutton interface

For those who wish to use the old interface, the rc variable must be set to "off." In future versions of the viewer, it will be possible for the layout of the Toolbutton bar to be specified by a user's configuration.

Signal Subwindow
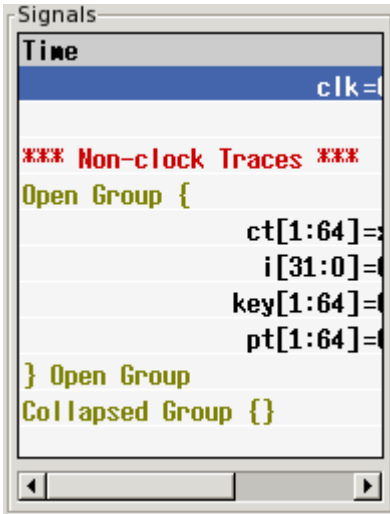
Figure 4: Signal subwindow with scrollbar and an "open" collapsible trace

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 , "Non-clock Traces ***". In between the two is a blank trace inserted by the user. Note that the highlighting of a trace can be accomplished by clicking the left mouse button on an entry in the signal subwindow. (Use ctrl-click to deselect.)

You will notice that the scrollbar along the bottom of the subwindow in Figure 4 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.

Expanding the size of the subwindow by increasing the width of the pane is illustrated in Figure 5. 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.
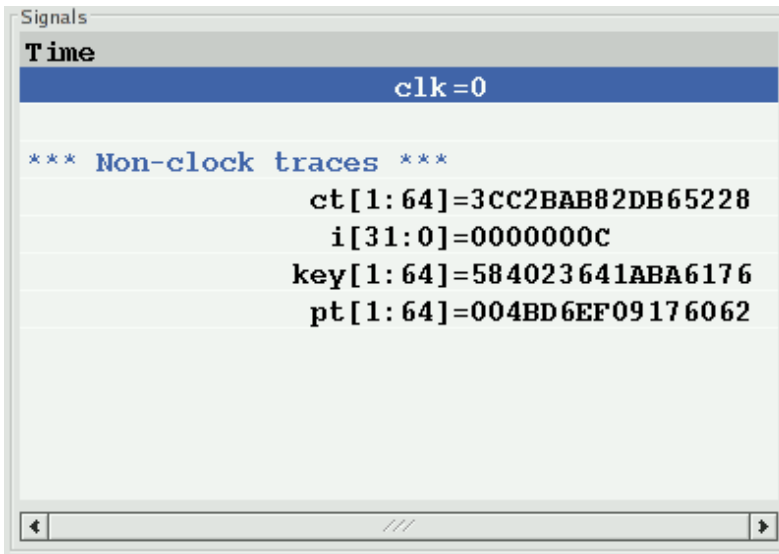
Figure 5: Signal subwindow with no hidden area from left to right

As seen in both Figure 4 and Figure 5, 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 6, 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

Figure 6: Signal subwindow with left justified signal names

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.

Note: the rc variable use_standard_clicking no longer has any effect. Regular GTK semantics for this subwindow are always enabled: shift and control function as most users expect. In addition, the scroll wheel will scroll the traces up and down provided the signal subwindow has input focus.

Wave Subwindow

The wave subwindow reformats simulation data into a visual format similar to that seen for digital storage scopes. As seen in Figure 7, the wave subwindow contains two scrollbars and a viewing area.
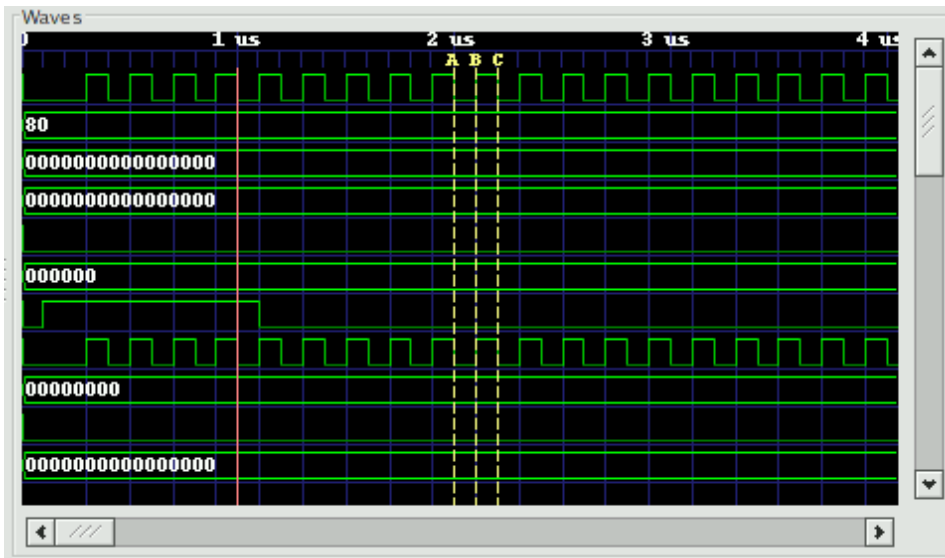
Figure 7: A typical view of the wave subwindow

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 8. 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.
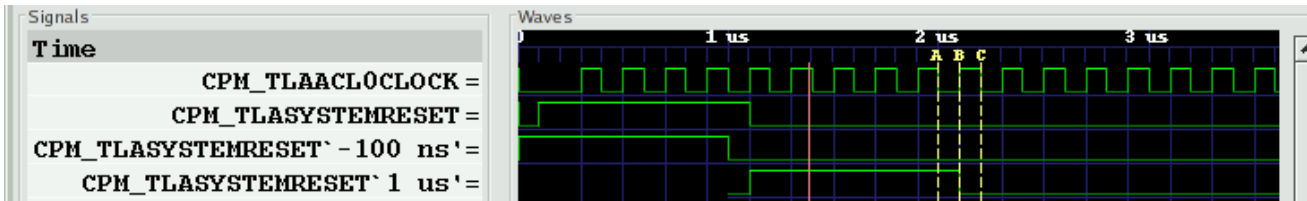
Figure 8: An example of both positively and negatively timeshifted traces

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 9: 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.

With some dumpfile types, a reload button can be found at the extreme right side of the Navigation and Status Panel. It may be seen in Figure 1: The GTKWave main window on page 19.

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 33.

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.
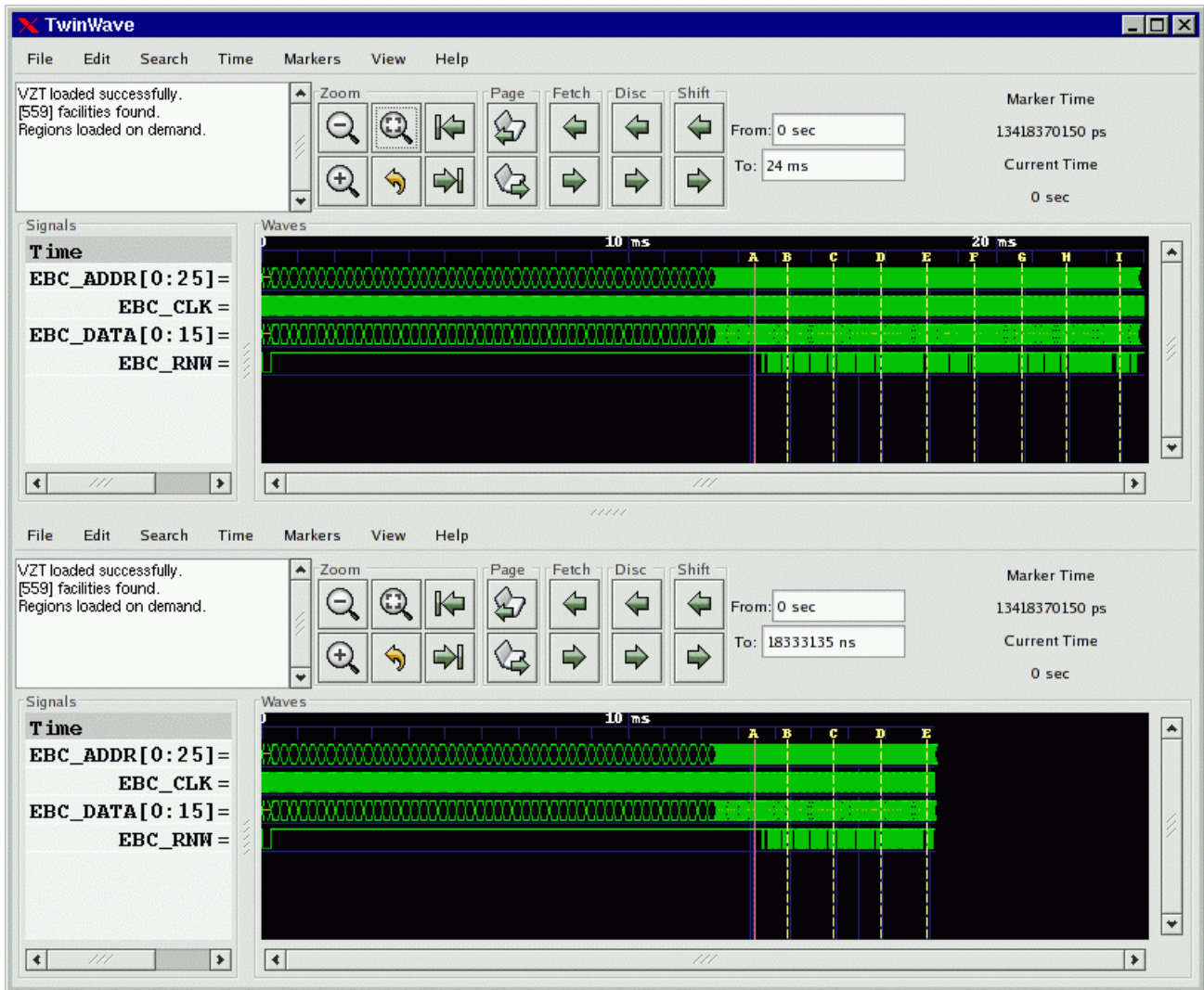
Figure 10: TwinWave managing two GTKWave sessions in a single window

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 *vermin* 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 11: The RTLBrowse RTL Design Hierarchy window on page 29.) 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 12: Source code annotated by RTLBrowse on page 30. If the primary marker moves or is deleted, then the values annotated into the source code will be updated dynamically. 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.
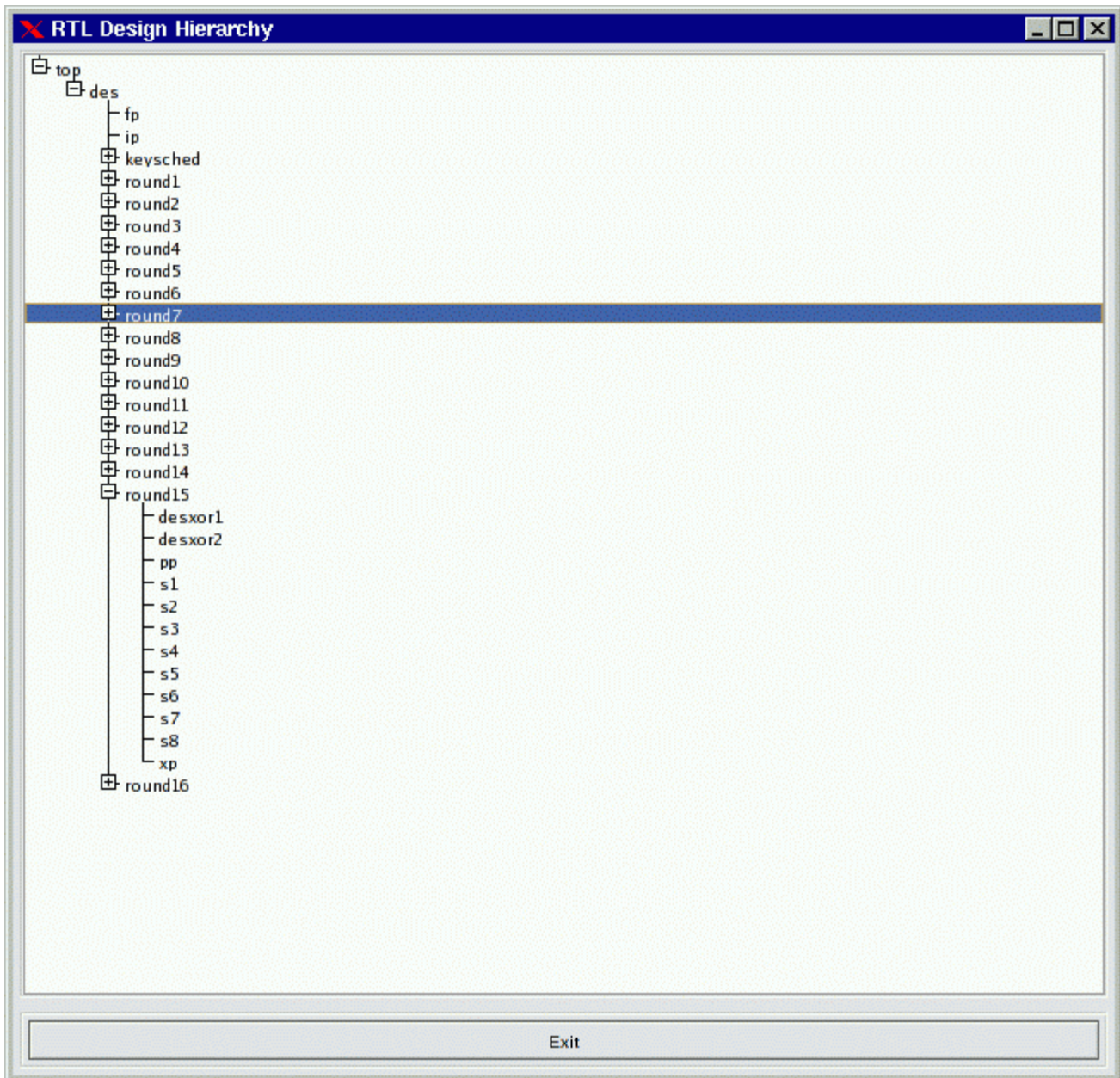
Figure 11: The RTLBrowse RTL Design Hierarchy window

top.des.round7

/tmp/verilog-0.8.2/examples/des.v

Design unit roundfunc occupies lines 991 - 1017.
Marker time for 'des.vcd.lx2' is 336 sec.

```
module  roundfunc(clk[b1],  li[28E7D50F],  ri[E44F2DA9],  lo[E44F2DA9],  ro[B204237A],  k[614800D8547F]);
input    clk[b1];
input    [1:32] li[28E7D50F],  ri[E44F2DA9];
input    [1:48] k[614800D8547F];
output   [1:32] lo[E44F2DA9],  ro[B204237A];

wire     [1:48] e[F0825E95BD53];
wire     [1:6] b1x[24],b2x[1C],b3x[29],b4x[1E],b5x[13],b6x[1E],b7x[24],b8x[2C];
wire     [1:4] so1x[E],so2x[5],so3x[6],so4x[F],so5x[0],so6x[B],so7x[B],so8x[E];
wire     [1:32] ppo[9AE3F675];

xp xp(ri[E44F2DA9], e[F0825E95BD53]);
desxor1 desxor1(e[F0825E95BD53], b1x[24], b2x[1C], b3x[29], b4x[1E], b5x[13], b6x[1E], b7x[24], b8x[2C], k[614800D8547F]);
s1 s1(clk[b1], b1x[24], so1x[E]);
s2 s2(clk[b1], b2x[1C], so2x[5]);
s3 s3(clk[b1], b3x[29], so3x[6]);
s4 s4(clk[b1], b4x[1E], so4x[F]);
s5 s5(clk[b1], b5x[13], so5x[0]);
s6 s6(clk[b1], b6x[1E], so6x[B]);
s7 s7(clk[b1], b7x[24], so7x[B]);
s8 s8(clk[b1], b8x[2C], so8x[E]);
pp pp(so1x[E],so2x[5],so3x[6],so4x[F],so5x[0],so6x[B],so7x[B],so8x[E], ppo[9AE3F675]);
desxor2 desxor2(ppo[9AE3F675], li[28E7D50F], ro[B204237A]);

assign lo[E44F2DA9]=ri[E44F2DA9];

endmodule
```

View Full File

Figure 12: 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 25.

- 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.

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.

Open New Tab will open a file requester that will ask for the name of a VCD or AET file to view. This will create a tabbed page.

Reload Current Waveform will reload the currently displayed waveform from a potentially updated file. Note that this menu option will only be displayed if the current waveform type supports reloading. (i.e., it is not sourced from standard input or from shared memory)

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.

Export-Write TIM File As will open a file requester that will ask for the name of a TimingAnalyzer .tim file. The contents of the file generated will be the representation of the traces onscreen. If the baseline and primary marker are

set, the time range written to the file will be between the two markers, otherwise it will be the entire time range.

Close immediately closes the current tab if multiple tabs exist or exits GTKWave after an additional confirmation requester is given the OK to quit.

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 Vermin. Please see its manpage for syntax and more information on stems file generation.

Read Script File will open a file requester that will ask for the name of a Tcl script to run. This menu option itself is not callable by Tcl scripts.

Quit exits GTKWave.

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.

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.

Toggle Trace Hier toggles the maximum hierarchy depth from zero to whatever was previously set.

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.

Copy copies highlighted signals from the display and places them in an offscreen cut/copy buffer for later Paste operations. Copy implicitly destroys the previous contents of the cut/copy 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.

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.

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-RealToBits-On will step through all highlighted traces and ensure that Real vectors with this qualifier will be displayed as Hex values. Note that this only works for Real quantities and other ones will remain to be displayed as binary. This is a pre-filter so it is possible to invert, reverse, apply Decimal, etc. It will not be possible however to expand those values into their constituent bits.

Data Format-RealToBits-Off will step through all highlighted traces and ensure that the RealToBits qualifier is removed from those traces.

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.

Transaction Filter Process will enable transaction filtering on marked traces using a filter process. A requester will appear to get the filter filename.

Transaction Filter Process Disable will remove transaction filtering.

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.

Analog Resizing-Screen Data causes the waveform data for all currently

highlighted traces to be displayed such that the y-value scaling maximizes the on-screen trace data so if fills the whole trace width at all times.

Analog Resizing-All Data causes the waveform data for all currently highlighted traces to be displayed such that the y-value scaling maximizes the on-screen trace data so if fills the whole trace width only when fully zoomed out. (i.e., the scale used goes across all trace data)

Data Format-Range Fill With 0s will step through all highlighted traces and ensure that vectors with this qualifier will be displayed as if the bitrange of the MSB or LSB as appropriate goes to zero. Zero bits will be filled in for the missing bits.

Data Format-Range Fill With 1s will step through all highlighted traces and ensure that vectors with this qualifier will be displayed as if the bitrange of the MSB or LSB as appropriate goes to zero. One bits will be filled in for the missing bits; this is mostly intended to be used when viewing values which are inverted in the logic and need to be inverted in the viewer.

Data Format-Zero Range Fill Off will step through all highlighted traces and ensure that normal bitrange displays are used.

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.

Toggle Group toggles a group opened or closed. Double-clicking does the same action as selecting this menu option.

Create Group creates a group of traces which may be opened or closed. It is permitted for groups to be nested.

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.

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.

Set Pattern Search Repeat Count sets the number of times that both edge and pattern searches iterate forward or backward when marker forward/backward is selected. Default value is one. This can be used, for example, to skip forward 10 clock edges at a time rather than a single edge.

Time

The Time submenu contains a superset of the functions performed by the Navigation and Status Panel button groups (see page 25).

Move To Time scrolls the waveform display such that the left border is the time entered in the requester. Use one of the letters A-Z to move to a named marker.

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.

Show-Change Marker Data displays and allows the modification of the times for all 26 named markers by filling in the leftmost entry boxes. In addition, optional marker text rather than a generic single letter name may be specified by filling in the rightmost entry boxes. Note that 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.

Lock to Lesser Named Marker locks the primary marker to a named marker. If no named marker is currently selected, the last defined one is used, otherwise the marker selected will be one lower in the alphabet, scrolling through to the end of the alphabet on wrap. If no named marker exists, one is dropped down for 'A' and the primary marker is locked to it.

Lock to Greater Named Marker locks the primary marker to a named marker. If no named marker is currently selected, the first defined one is used, otherwise the marker selected will be one higher in the alphabet, scrolling through to the beginning of the alphabet on wrap. If no named marker exists, one is dropped down for 'A' and the primary marker is locked to it.

Unlock from Named Marker unlocks the primary marker from the currently selected named marker.

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.

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. Note that an encoded string will be displayed next to the signal name that indicates what data format flags are currently active for that signal. Flags are as follows:

 + = Signed Decimal
X = Hexadecimal
A = ASCII
D = Decimal
B = Binary
O = Octal
J = Right Justify
 ~ = Invert
V = Reverse

```
* = Analog Step+Interpolated
S = Analog Step
I = Analog Interpolated
R = Real
r = Real To Bits
0 = Range Fill with 0s
1 = Range Fill with 1s
G = Binary to Gray
g = Gray to Binary
```

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 Delta-Frequency allows you to switch between the delta time and frequency display in the upper right corner of the main window when measuring distances between markers. Default behavior is that the delta time is displayed.

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.

<u>Right Justify Signals</u> draws signal names flushed to the right ("equals") side of the signal window.

<u>Zoom Pow10 Snap</u> 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.

<u>Partial VCD Dynamic Zoom Full</u> causes the screen to be in full zoom mode while a VCD file is loading incrementally.

<u>Partial VCD Dynamic Zoom To End</u> causes the screen to zoom to the end while a VCD file is loading incrementally.

<u>Full Precision</u> 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.

<u>Define Time Ruler Marks</u> changes the ruler markings such that the Baseline marker defines the origin and the Primary marker distance from the Baseline marker defines the period. If either the Baseline marker or Primary marker are not present, the default ruler markers are used. If the Baseline marker and Primary marker have the same value, the default ruler markers are used.

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

<u>Use Color</u> draws signal names and trace data in color. This is normal operation.

<u>Use Black and White</u> draws signal names and trace data in black and white. This is intended for use in black and white screen dumps.

<u>Scale To Time Dimension: None</u> turns off time dimension conversion.

<u>Scale To Time Dimension: sec</u> changes the time dimension conversion value to seconds.

<u>Scale To Time Dimension: ms</u> changes the time dimension conversion value to milliseconds.

<u>Scale To Time Dimension: us</u> changes the time dimension conversion value to microseconds.

<u>Scale To Time Dimension: ns</u> changes the time dimension conversion value to nanoseconds.

Scale To Time Dimension: ps changes the time dimension conversion value to picoseconds.

Scale To Time Dimension: fs changes the time dimension conversion value to femtoseconds.

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.

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 distribution 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> vermin des.v -emitstems >des.stems
Vermin: 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 v3.3.18 (w)1999-2010 BSI

FSTLOAD | Processing 1432 facs.
FSTLOAD | Built 1287 signals and 145 aliases.
FSTLOAD | Building facility hierarchy tree.
FSTLOAD | Sorting facility hierarchy tree.
```

In some cases, for example if the dumpfile format is LXT2, you will see two sets of loader 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.
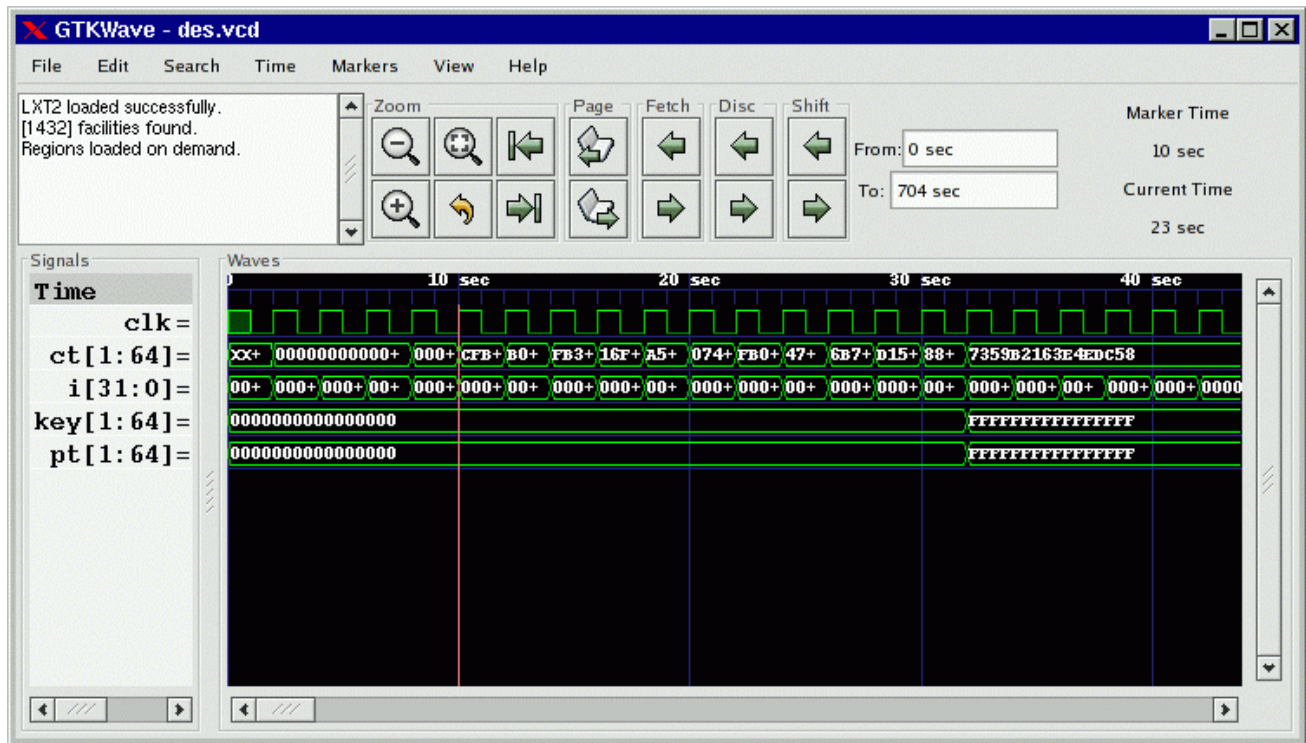
Figure 13: The main window with viewer state loaded from a save file

The RTLBrowse window will come up as seen in Figure 11: The RTLBrowse RTL
Design Hierarchy window on page 29, 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 13 on page 49. 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 Error: Reference source not found. 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.

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.
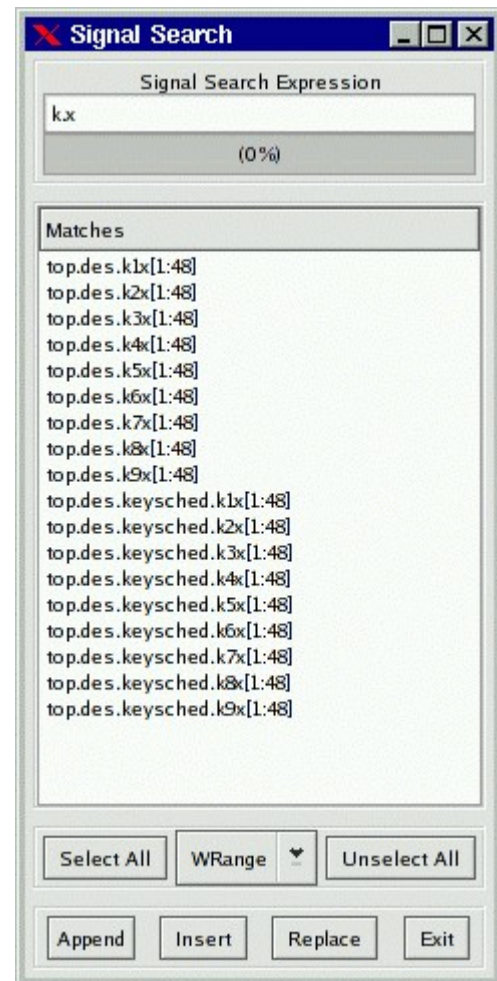
Replace will replace highlighted signals on

Figure 14: The Signal Search (regular expression search) Requester

the main window with signals selected.

Hierarchy Search

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 40.

*Figure 15: The Hierarchy
Search Requester*

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 52, 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).
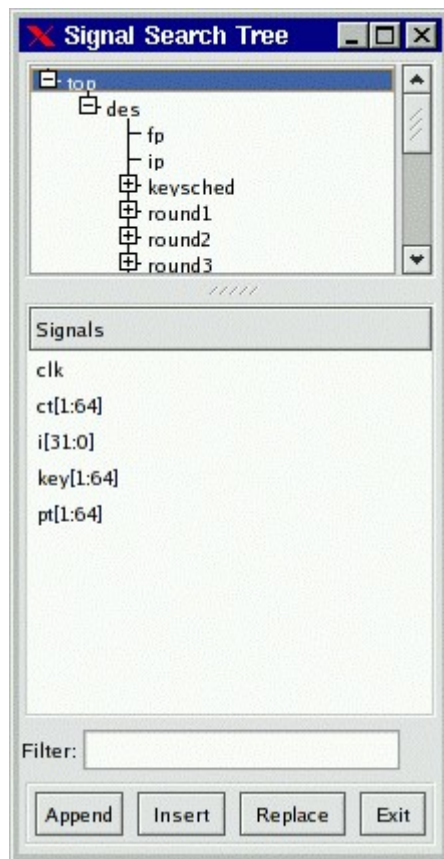
Figure 16: The Signal Search Tree Requester

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.
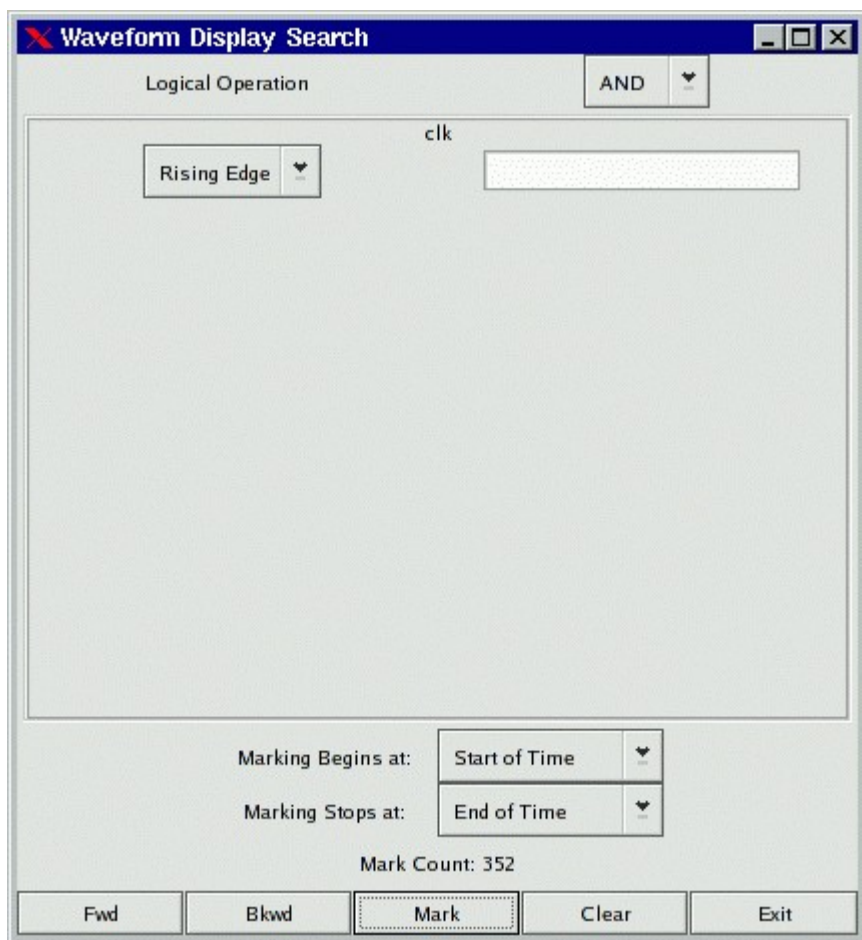
Figure 17: The Pattern Search Requester

The edges as they are marked by the configuration of the Requester in Figure 17 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 is is no longer required 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:

```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

Note: In order to use the filter to modify the background color of a trace, you can prefix the return string to stdout with the X11 color name surrounded by '?' characters as follows:

```
?CadetBlue?isync
?red?xor r0,r0,r0
?lavender?lwz r2,0(r7)
```

Legal color names may be found in the rgb.c file in the sourcecode distribution.

Transaction Filters

Either single traces or grouped vector data (created by Combine Down (F4)on some signals) can be used to signify a transaction that can be parsed by an external process.

An external process that can accept a simplified VCD file from stdin and return with trace data on stdout can be used as a transaction filter. An example of the VCD file received from stdin is the following:

```
$comment data_start 0x124c0798 $end
$comment val[7:0] $end
$timescale 1ms $end
$comment min_time 0 $end
$comment max_time 348927 $end
$comment max_seqn 1 $end
$scope module top $end
$comment seqn 1 top.val[7:0] $end
$var wire 8 1 val[7:0] $end
$upscope $end
$enddefinitions $end
$dumpvars
#0
b10000000 1
#1
b10000101 1
#2
b10001010 1

…

#348927
b110010 1
$comment data_end 0x124c0798 $end
```

To aid in processing and parsing, some extra comments are added to the VCD file:
data_start, a value to match against data end to know that all trace data has been received
min_time, the start time of the wave data
max_time, the ending time of the wave data
max_seqn, indicates the relative ordering of the trace data being presented. This can be used to provide "anonymous" signal name matching
seqn, gives the "flat earth" signal name

Note that the VCD identifies are numbers starting from 1. These are to be correlated with the max_seqn count.

An example of data generated on stdout after all data has been received is as follows:

```
$name Decoded Data
#0
#186608 ?darkblue?sync
MA196608 Sync Mark
#196860
```

```
MB196864 Num Blocks
#196864 ?gray24?04
#197116
MC197120 Hdr 0
#197120 ?purple3?04
#197372
$next
$name Another Trace
#0
#10000 This is a test!
#200000
$finish
```

Time values with no data after them are rendered as a horizontal "z" bar.
Lines that start with M are used to place the markers A-Z.
$name indicates the name to give to the trace.
$next indicated that more trace data follows for a new trace.
$finish is used to signal to gtkwave that there is no more trace data.

The data received by gtkwave will be used to generate transaction traces in the viewer. In order to make traces created by $next visible, insert blank lines under the trace that the transaction filter has been added.

To turn on the filter:

1) Highlight the signals you want filtered
2) Edit->Data Format->Transaction Filter Process->Enable and Select
3) Add Transaction 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->Transaction Filter Process->Disable

Note: In order to use the filter to modify the background color of a trace, you can prefix the return string to stdout with the X11 color name surrounded by '?' characters as follows:

```
?CadetBlue?isync
?red?xor r0,r0,r0
?lavender?lwz r2,0(r7)
```

Legal color names may be found in the rgb.c file in the sourcecode distribution.

Debugging the Source Code

See the description for RTLBrowse on page 28. 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.

       -F,--fastload
              generate/use VCD recoder fastload files.  This is similar to the
              -g,--giga option,  however  the  spill  file  generated  is  not
              deleted.   Reloading  the  VCD file another time (either through
              pressing the reload button or by re-invoking gtkwave at a  later
              time)  will  use  this generated spill file rather than read the
              value change section of  the  VCD  file.   This  will  speed  up
              reloads  on large files greatly as only the variable declaration
              section needs to be parsed.  Note that the spill  file  contains
              the  file size and modification date of the VCD file in order to
```

detect if it is stale and needs to be regenerated.

    -o,--optimize
          optimize VCD to FST. This will automatically call vcd2fst(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.

    -a, --save=FILE
          Specify savefile name.

    -A, --autosavename
           Assume savefile is suffix modified dumpfile name.

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

    -l,--logfile <filename>
          Specify simulation logfile name. Multiple logfiles may be spec-
          ified 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 cur-
          rent directory and it is not explicitly specified on the command
          line, when this option is enabled, do not use an implicit con-
          figuration 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 auto-
          matically launch the rtlbrowse(1) helper process. See vermin(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, how-
ever 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 Tcl script for execution.

-R,--repscript <filename>
        Specifies Tcl command script file for periodic execution.

-P,--repperiod <filename>
        Specifies delay in milliseconds between successive executions of
        the repscript. Default is 500.

-T,--tcl_script <filename>
        Specifies Tcl command script file to be loaded on startup.   Implies
        --wish command flag.

-W,--wish
        Enables Tcl command line on stdio. All script commands can be
        typed in on stdin.

-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 Gtk-
        Plug expecting a GtkSocket at the other end.   Note that there
        are issues with accelerators working properly so menus are dis-
        abled 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.

-g, --giga
        Specifies to use gigabyte mempacking when recoding (slower).

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

```
-C,--comphier
       Specifies  that  the  viewer  should use compressed hierarchy names
       when loading the dumpfile (available for VCD recoder, LXT, LXT2, and
       VZT).  This will use less memory at the expense of
       compression/decompression delay.

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

-O,--output <filename>
       Specify filename for stdout/stderr redirect.  To disable messages to
       the console, use /dev/null as the filename.

-z,--slider-zoom
       Enable  slider  stretch  zoom  for  the  horizontal time slider.
       Clicking then dragging the very left or right edge of the slider
       can be used to provide fine-grained real-time zooming.

-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

Note that to pass non-flag items which start with a dash,  that  it  is
required  to specify -- in order to turn off flag parsing.  A second --
will disable parsing of any following arguments such that they  can  be
passed on to Tcl scripts and retrieved via gtkwave::getArgv.

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)
       vermin(1) rtlbrowse(1) twinwave(1) shmidcat(1)

Anthony Bybell   3.3.20   GTKWAVE(1)

fst2vcd

FST2VCD(1)   Filetype Conversion   FST2VCD(1)

NAME
       fst2vcd - Coverts FST files to VCD

SYNTAX
       fst2vcd [option]... [FSTFILE]

DESCRIPTION
       Converts FST files to VCD files on stdout.

OPTIONS
       -f,--fstname <filename>
             Specify FST input filename.

       -h,--help
             Display help then exit.

EXAMPLES
       To run this program the standard way type:

       fst2vcd filename.fst
             The VCD conversion is emitted to stdout.

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

SEE ALSO
       vcd2fst(1) gtkwave(1)

vcd2fst

VCD2FST(1)   Filetype Conversion   VCD2FST(1)

NAME
       vcd2fst - Converts VCD files to FST files

SYNTAX
       vcd2fst [option]... [VCDFILE] [FSTFILE]

DESCRIPTION
       Converts VCD files to FST files.

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

       -f,--fstname <filename>
            Specify FST output filename.

       -F,--fastpack
            Indicates that fastlz should be used instead of gzip for block
            data.

       -c,--compress
            Indicates that the entire file should be run through gzip on
            close.   This results in much smaller files at the expense of a
            one-time decompression penalty on file open during reads.

       -h,--help
            Show help screen.

       EXAMPLES

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

       vcd2fst dumpfile.vcd dumpfile.fst --compress
            This indicates that the FST file should be post-compressed on
            close.

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

SEE ALSO
       fst2vcd(1) vcd2lxt(1) vcd2lxt2(1) lxt2vcd(1) vcd2vzt(1) vzt2vcd(1) gtk-
       wave(1)

Anthony Bybell   3.2.2   VCD2FST(1)

evcd2vcd

```
EVCD2VCD(1)                    Filetype Conversion                    EVCD2VCD(1)


NAME
       evcd2vcd - Converts EVCD files to VCD files

SYNTAX
       evcd2vcd [option]... [EVCDFILE]

DESCRIPTION
       Converts  EVCD  files  with bidirectional port definitions to VCD files
       with separate in and out ports.

OPTIONS
       -f,--filename <filename>
             Specify EVCD input filename.

       -h,--help
             Show help screen.

       EXAMPLES

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

       evcd2vcd dumpfile.evcd
             VCD is emitted to stdout.

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

SEE ALSO
       vcd2fst(1)  fst2vcd(1)  vcd2lxt(1)  vcd2lxt2(1)  lxt2vcd(1)  vcd2vzt(1)
       vzt2vcd(1) gtkwave(1)

Anthony Bybell                       3.2.2                            EVCD2VCD(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.

     -c,--comprehensive
         Indicates that results are not to stop after the first match.
         This can be used to extract all the matching values in the
         trace.

     -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   3.2.1   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   1.3.64   LXT2VCD(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,  FST, and AET2.  For VCD, use the
      -o,--optimize option of gtkwave(1) in order to optimize  the  VCD  file
      into  FST.  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 vermin(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
       vermin(1) gtkwave(1)

vcd2lxt

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

GTKWave 3.3 Wave Analyzer User's Guide   69

```
        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, 2 = lzma).   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   3.1.21   VCD2VZT(1)

vermin

vermin(1)   Verilog Compilation   vermin(1)

NAME
      vermin - Parses and processes Verilog HDL files

SYNTAX
      vermin [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, vermin) 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.

          vermin 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   vermin(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.

       -c,--comprehensive
             Indicates  that  results  are not to stop after the first match.
             This can be used to extract  all  the  matching  values  in  the
             trace.

       -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   3.2.1   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)

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.
```

```
analog_redraw_skip_count <value>
        Specifies  how many overlapping analog segments can be drawn for
        a given X position onscreen.  (Default: 20)  If there  are  gaps
        in analog traces, this value is too low.
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_brkred <value>
        brick red color for comments.

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

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_gmstrd <value>
                color value for trace groupings.

color_grid <value>
                grid color (use Alt-G/Shift-Alt-G to show/hide grid).

color_grid2 <value>
                grid color for secondary pattern search.

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

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

color_ltblue <value>
                color value for shadowed traces.

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).

context_tabposition <value>
        Use zero for tabbed viewing with named tabs at the top. Nonzero
        places numerically indexed tabs at the left.

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_ae2_alias <value>
        A nonzero value indicates that the AE2 loader is to ignore the
        aliasdb keyword and is not to construct facility aliases.

disable_empty_gui <value>
        A nonzero value indicates that if gtkwave is invoked without a

dumpfile name, then an empty gtkwave session is to be sup-
pressed. Default is a zero value: to bring up an empty session
which needs a file loaded or dragged into it.

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 gen-
       erate 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 appro-
priate 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 appropri-
ate fontnames using the xfontsel program. Note that the signal
font must be taller than the wave font or the viewer will com-
plain 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 lev-
els 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 dis-
play 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 rec-
ommended). A zero value indicates that gtk_paned_add will be
used instead.

ignore_savefile_pos <value>
If nonzero, specifies that the window position attribute is to
be ignored during savefile loading and is to be skipped during
saving. Default is that the position attribute is used.

ignore_savefile_size <value>
If nonzero, specifies that the window size attribute is to be
ignored during savefile loading and is to be skipped during sav-
ing. Default is that the size attribute is used.

initial_signal_window_width <value>
Sets the creation width for the signal pane on GUI initializa-
tion. Also sets another potential minimum value for dynamic

resizing.

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 posi-
    tion.

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 posi-
    tion.

left_justify_sigs <value>
    When nonzero, indicates that the signal window signal name jus-
    tification 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.

ruler_origin <value>
    sets the zero origin for alternate time tick marks.

ruler_step <value>
    sets the left/right step value for the alternate time tick marks
    from the origin. When this value is zero, alternate time tick
    marks are disabled.

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).

scale_to_time_dimension <value>
    The value can be any of the characters m, u, n, f, p, or s,
    which indicates which time dimension to convert the time values

to. The default for this is * which means that time dimension conversion is disabled.

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_dynamic_filter <value>
   When true (default) allows the SST dialog signal filter to filter signals while keys are being pressed, otherwise enter must be pressed to cause the filter to go active.

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

strace_repeat_count <value>
   Determines how many times that edge search and pattern search will iterate on a search. This allows, for example, skipping ahead 10 clock edges instead of 1.

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_frequency_delta <value>
   allows you to switch between the delta time and frequency display in the upper right corner of the main window when measuring distances between markers. Default behavior is that the delta time is displayed (off).

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_pango_fonts <value>
        Uses anti-aliased pango fonts (GTK2) rather than bitmapped X11
        ones. Default is on.

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 dis-
        card 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.

use_scrollwheel_as_y <value>
        A nonzero value indicates that the scroll wheel on the mouse
        should be used to scroll the signals up and down rather than
        scrolling the time value from left to right.

use_standard_clicking <value>
        This option no longer has any effect in gtkwave: normal GTK
        click semantics are used in the signalwindow.

use_toolbutton_interface <value>
        A nonzero value indicates that a toolbar with buttons should be
        at the top of the screen instead of the traditional style gtk-
        wave button groups. Default is on.

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 dif-
        ferent 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_pre-
        serve_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.

vlist_prepack <value>
indicates that the VCD recoder should pre-compress data going
into the value change vlists in order to reduce memory usage.
This is done before potential zlib packing. Default is off.

vlist_spill <value>
indicates that the VCD recoder should spill all generated vlists
to a tempfile on disk in order to reduce memory usage. Default
is off.

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_dynamic <value>
a nonzero value enables dynamic full zooming when using the par-
tial VCD (incremental) loader, a zero value disables it.

zoom_dynamic_end <value>
a nonzero value enables dynamic zoom to the end when using the
partial VCD (incremental) loader, 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   3.2.0   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".

Appendix E: Tcl Command Syntax

Tcl Command Syntax

Besides being able to access the menu options (e.g., gtkwave::/File/Quit), within Tcl scripts there are more commands available for manipulating the viewer.

addSignalsFromList: adds signals to the viewer

Syntax: `set num_found [ gtkwave::addSignalsFromList list ]`

Example:
```
set clk48 [list]
lappend clk48 "$facname1"
lappend clk48 "$facname2"
...
set num_added [ gtkwave::addSignalsFromList $clk48 ]
```

deleteSignalsFromList: deletes signals from the viewer. This deletes only the first instance found unless the signal is specified multiple times in the list.

Syntax: `set num_deleted [ gtkwave::deleteSignalsFromList list ]`

Example:
```
set clk48 [list]
lappend clk48 "$facname1"
lappend clk48 "$facname2"
...
set num_deleted [ gtkwave::deleteSignalsFromList $clk48 ]
```

deleteSignalsFromListIncludingDuplicates: deletes signals from the viewer. This deletes all the instances found so there is no need to specify the same signal multiple times in the list.

Syntax: `set num_deleted`
`[ gtkwave::deleteSignalsFromListIncludingDuplicates list ]`

Example:
```
set clk48 [list]
lappend clk48 "$facname1"
lappend clk48 "$facname2"
...

set num_deleted [ gtkwave::deleteSignalsFromListIncludingDuplicates
$clk48 ]
```

findNextEdge: advances the marker to the next edge for highlighted signals

Syntax: `set marker_time [ gtkwave::findNextEdge ]`

Example:
```
gtkwave::highlightSignalsFromList "top.clk"
set time_value [ gtkwave::findNextEdge ]
puts "time_value: $time_value"
```

findPrevEdge: moves the marker to the previous edge for highlighted signals

Syntax: `set marker_time [ gtkwave::findPrevEdge ]`

Example:
```
gtkwave::highlightSignalsFromList "top.clk"
set time_value [ gtkwave::findPrevEdge ]
puts "time_value: $time_value"
```

forceOpenTreeNode: forces open one tree node in the Signal Search Tree and closes the rest. If upper levels are not open, the tree will remain closed however once the upper levels are opened, the hierarchy specified will become open. If path is missing or is an empty string, the function returns the current hierarchy path selected by the SST or -1 in case of error.

Syntax: `gtkwave::forceOpenTreeNode hierarchy_path`

Returned value:
    0 - success
    1 - path not found in the tree
    -1 - SST tree does not exist

Example:
```
  set path tb.HDT.cpu
  switch -- [gtkwavetcl::forceOpenTreeNode $path] {
    -1 {puts "Error: SST is not supported here"}
    1 {puts "Error: '$path' was not recorder to dump file"}
    0 {}
  }
```

getArgv: returns a list of arguments which were used to start gtkwave from the command line

Syntax: set argvlist [gtkwave::getArgv]

Example:
set argvs [gtkwave::getArgv]
puts "$argvs"

getBaselineMarker: returns the numeric value of the baseline marker time

Syntax: set baseline_time [gtkwave::getBaselineMarker]

Example:
set baseline [gtkwave::getBaselineMarker]
puts "$baseline"

getDisplayedSignals: returns a list of all signals currently on display

Syntax: set display_list [gtkwave::getDisplayedSignals]

Example:
set display_list [gtkwave::getDisplayedSignals]
puts "$display_list"

getDumpFileName: returns the filename for the loaded dumpfile

Syntax: set loaded_file_name [gtkwave::getDumpFileName]

Example: set nfacs [gtkwave::getNumFacs]
set dumpname [gtkwave::getDumpFileName]
set dmt [gtkwave::getDumpType]
puts "number of signals in dumpfile '$dumpname' of type $dmt: $nfacs"

getDumpType: returns the dump type as a string (VCD, PVCD, LXT, LXT2, GHW, VZT)

Syntax: set dump_type [gtkwave::getDumpType]

Example:
set nfacs [gtkwave::getNumFacs]
set dumpname [gtkwave::getDumpFileName]
set dmt [gtkwave::getDumpType]
puts "number of signals in dumpfile '$dumpname' of type $dmt: $nfacs"

getFacName: returns a string for the facility name which corresponds to a given facility number

Syntax: set fac_name [gtkwave::getFacName fac_number]

Example:
set nfacs [gtkwave::getNumFacs]
for {set i 0} {$i < $nfacs } {incr i} {
    set facname [gtkwave::getFacName $i]
    puts "$i: $facname"

}

getFontHeight: returns the font height for signal names

Syntax: set font_height [gtkwave::getFontHeight]

Example:
set font_height [gtkwave::getFontHeight]
puts "$font_height"

getFromEntry: returns the time value string in the "From:" box.

Syntax: set from_entry [gtkwave::getFromEntry]

Example:
set from_entry [gtkwave::getFromEntry]
puts "$from_entry"

getHierMaxLevel: returns the max hier value which is set in the viewer

Syntax: set hier_max_level [gtkwave::getHierMaxLevel]

Example:
set max_level [gtkwave::getHierMaxLevel]
puts "$max_level"

getLeftJustifySigs: returns 1 if signals are left justified, else 0

Syntax: set left_justify [gtkwave::getLeftJustifySigs]

Example:
set justify [gtkwave::getLeftJustifySigs]
puts "$justify"

getLongestName: returns number of characters of the longest name in the dumpfile

Syntax: set longestname_len [gtkwave::getLongestName]

Example:
set longest [gtkwave::getLongestName]
puts "$longest"

getMarker: returns the numeric value of the primary marker position

Syntax: set marker_time [gtkwave::getMarker]

Example:
set marker_time [gtkwave::getMarker]
puts "$marker_time"

getMaxTime: returns the numeric value of the last time value in the dumpfile

Syntax: set max_time [gtkwave::getMaxTime]

Example:
```
set max_time [ gtkwave::getMaxTime ]
puts "$max_time"
```

getMinTime: returns the numeric value of the first time value in the dumpfile

Syntax: set min_time [gtkwave::getMinTime]

Example:
```
set min_time [ gtkwave::getMinTime ]
puts "$min_time"
```

getNamedMarker: returns the numeric value of the named marker position

Syntax: set time_value [gtkwave::getNamedMarker which]
such that which = A-Z or a-z

Example:
```
set marker_time [ gtkwave::getNamedMarker A ]
puts "$marker_time"
```

getNumFacs: returns the number of facilities encountered in the dumpfile

Syntax: set numfacs [gtkwave::getNumFacs]

Example:
```
set nfacs [ gtkwave::getNumFacs ]
set dumpname [ gtkwave::getDumpFileName ]
set dmt [ gtkwave::getDumpType ]
puts "number of signals in dumpfile '$dumpname' of type $dmt: $nfacs"
```

getNumTabs: returns the number of tabs shown on the viewer

Syntax: set numtabs [gtkwave::getNumTabs]

Example:
```
set ntabs [ gtkwave::getNumTabs ]
puts "number of tabs: $ntabs"
```

getPixelsUnitTime: returns the number of pixels per unit time

Syntax: set pxut [gtkwave::getPixelsUnitTime]

Example:
```
set pxut [ gtkwave::getPixelsUnitTime ]
puts "$pxut"
```

getSaveFileName: returns the save filename

Syntax: set save_file_name [gtkwave::getSaveFileName]

Example:
```
set savename [ gtkwave::getSaveFileName ]
```

```
puts "$savename"
```
getStemsFileName: returns the stems filename

Syntax: `set stems_file_name [ gtkwave::getStemsFileName ]`

Example:
```
set stemsname [ gtkwave::getStemsFileName ]
puts "$stemsname"
```
getTimeDimension: returns the first character of the time units that the trace was saved in (e.g., "u" for us, "n" for "ns", "s" for sec, etc.)

Syntax: `set dimension_first_char [ gtkwave::getTimeDimension ]`

Example:
```
set dimch [ gtkwave::getTimeDimension ]
puts "$dimch"
```
getToEntry: returns the time value string in the "To:" box.

Syntax: `set to_entry [ gtkwave::getToEntry ]`

Example:
```
set to_entry [ gtkwave::getFromEntry ]
puts "$to_entry"
```
getTotalNumTraces: returns the total number of traces that are being displayed currently

Syntax: `set total_traces [ gtkwave::getTotalNumTraces ]`

Example:
```
set totnum [ gtkwave::getTotalNumTraces ]
puts "$totnum"
```
getTraceFlagsFromIndex: returns the decimal value of the sum of all flags for a given trace

Syntax: `set flags [ gtkwave::getTraceFlagsFromIndex trace_number ]`

Example:
```
set tflags [ gtkwave::getTraceFlagsFromIndex 0 ]
puts "$tflags"
```
getTraceFlagsFromName: returns the decimal value of the sum of all flags for a given trace

Syntax: `set flags [ gtkwave::getTraceFlagsFromName trace_name ]`

Example:
```
set tflags [ gtkwave::getTraceFlagsFromName {top.des.k1x[1:48]} ]
puts "$tflags"
```

getTraceNameFromIndex: returns the name of a trace when given the index value

Syntax: set trace_name [gtkwave::getTraceNameFromIndex trace_number]

Example:
set tname [gtkwave::getTraceNameFromIndex 1]
puts "$tname"

getTraceScrollbarRowValue: returns the scrollbar value (which corresponds to the trace index for the topmost trace on screen)

Syntax: set scroller_value [gtkwave::getTraceScrollbarRowValue]

Example:
set scroller [gtkwave::getTraceScrollbarRowValue]
puts "$scroller"

getTraceValueAtMarkerFromIndex: returns the value under the marker for the trace numbered trace index

Syntax: set ascii_value [gtkwave::getTraceValueAtMarkerFromIndex trace_number]

Example:
set tvi [gtkwave::getTraceValueAtMarkerFromIndex 2]
puts "$tvi"

getTraceValueAtMarkerFromName: returns the value under the primary marker for the given trace name

Syntax: set ascii_value [gtkwave::getTraceValueAtMarkerFromName fac_name]

Example:
set tvn [gtkwave::getTraceValueAtMarkerFromName
{top.des.k2x[1:48]}]
puts "$tvn"

getTraceValueAtNamedMarkerFromName: returns the value under the named marker for the given trace name

Syntax: set ascii_value
[gtkwave::getTraceValueAtNamedMarkerFromName which fac_name]
such that which = A-Z or a-z

Example:
set tvn [gtkwave::getTraceValueAtNamedMarkerFromName A
{top.des.k2x[1:48]}]
puts "$tvn"

getUnitTimePixels: returns the number of time units per pixel

Syntax: set utpx [gtkwave::getUnitTimePixels]

Example:
set utpx [gtkwave::getUnitTimePixels]
puts "$utpx"

getVisibleNumTraces: returns number of non-collapsed traces

Syntax: set num_visible_traces [gtkwave::getVisibleNumTraces]

Example:
set nvt [gtkwave::getVisibleNumTraces]
puts "$nvt"

getWaveHeight: returns the height of the wave window in pixels

Syntax: set wave_height [gtkwave::getWaveHeight]

Example:
set wht [gtkwave::getWaveHeight]
puts "$wht"

getWaveWidth: returns the width of the wave window in pixels

Syntax: set wave_width [gtkwave::getWaveWidth]

Example:
set wwt [gtkwave::getWaveWidth]
puts "$wwt"

getWindowEndTime: returns the end time of the wave window

Syntax: set end_time_value [gtkwave::getWindowEndTime]

Example:
set wet [gtkwave::getWindowEndTime]
puts "$wet"

getWindowStartTime: returns the start time of the wave window

Syntax: set start_time_value [gtkwave::getWindowStartTime]

Example:
set wst [gtkwave::getWindowStartTime]
puts "$wst"

getZoomFactor: returns the zoom factor of the wave window

Syntax: set zoom_value [gtkwave::getZoomFactor]

Example:
set zf [gtkwave::getZoomFactor]

```
puts "$zf"
```

highlightSignalsFromList: highlights the facilities contained in the list argument

Syntax: `set num_highlighted [ gtkwave::highlightSignalsFromList list ]`

Example:
```
set clk48 [list]
lappend clk48 "$facname1"
lappend clk48 "$facname2"
...
set num_highlighted [ gtkwave::highlightSignalsFromList $clk48 ]
```

installFileFilter: installs file filter number which across all highlighted traces. Using zero for which removes the filter.

Syntax: `set num_updated [ gtkwave::installFileFilter which ]`

Example:
```
set num_updated [ gtkwave::installFileFilter 0 ]
puts "$num_updated"
```

installProcFilter: installs process filter number which across all highlighted traces. Using zero for which removes the filter.

Syntax: `set num_updated [ gtkwave::installProcFilter which ]`

Example:
```
set num_updated [ gtkwave::installProcFilter 0 ]
puts "$num_updated"
```

installTransFilter: installs transaction process filter number which across all highlighted traces. Using zero for which removes the filter.

Syntax: `set num_updated [ gtkwave::installTransFilter which ]`

Example:
```
set num_updated [ gtkwave::installTransFilter 0 ]
puts "$num_updated"
```

loadFile: loads a new file

Syntax: `gtkwave::loadFile filename`

Example:
```
gtkwave::loadFile "$filename"
```

nop: calls the GTK main loop in order to update the gtkwave GUI

Syntax: `gtkwave::nop`

Example:

```
gtkwave::nop
```

presentWindow: raises the main window in the stacking order or deiconifies it

Syntax: `gtkwave::presentWindow`

Example:
```
gtkwave::presentWindow
```

reLoadFile: reloads the current active file

Syntax: `gtkwave::reLoadFile`

Example:
```
gtkwave::reLoadFile
```

setBaselineMarker: sets the time for the baseline marker (-1 removes it)

Syntax: `gtkwave::setBaselineMarker time_value`

Example:
```
gtkwave::setBaselineMarker 128
```

setCurrentTranslateEnums: sets the enum list to function as the current translate file and returns the corresponding which value to be used with gtkwave::installFileFilter. As a real file is not used, the results of this are not recreated when a save file is loaded or if the waveform is reloaded.

Syntax: `set which_f [ gtkwave::setCurrentTranslateEnums elist ]`

Example:
```
Set enums [list]
lappend enums 0000000000000000        IDLE
lappend enums FFFFFFFFFFFFFFFF        BUSY
lappend enums 3000000000000000        OTHER
lappend enums 0123456789ABCDEF        HEXSTATE
lappend enums 1111111111111111       "All 1s"
set which_f [ gtkwave::setCurrentTranslateFile $enums ]
puts "$which_f"
```

setCurrentTranslateFile: sets the filename to the current translate file and returns the corresponding which value to be used with gtkwave::installFileFilter.

Syntax: `set which_f [ gtkwave::setCurrentTranslateFile filename ]`

Example:
```
set which_f [ gtkwave::setCurrentTranslateFile ./zzz.txt ]
puts "$which_f"
```

setCurrentTranslateProc: sets the filename to the current translate process (executable) and returns the corresponding which value to be used with gtkwave::installProcFilter.

Syntax: set which_f [gtkwave::setCurrentTranslateProc filename]

Example:
set which_f [gtkwave::setCurrentTranslateProc ./zzz.exe]
puts "$which_f"

setCurrentTranslateTransProc: sets the filename to the current transaction translate process (executable) and returns the corresponding which value to be used with gtkwave::installTransFilter.

Syntax: set which_f [gtkwave::setCurrentTranslateTransProc filename]

Example:
set which_f [gtkwave::setCurrentTranslateTransProc ./zzz.exe]
puts "$which_f"

setFromEntry: sets the time in the "From:" box.

Syntax: gtkwave::setFromEntry time_value

Example:
gtkwave::setFromEntry 100

setLeftJustifySigs: turns left justification for signal names on or off

Syntax: gtkwave::setLeftJustifySigs on_off_value

Example:
gtkwave::setLeftJustifySigs on
gtkwave::setLeftJustifySigs off

setMarker: sets the time for the primary marker (-1 removes it)

Syntax: gtkwave::setMarker time_value

Example:
gtkwave::setMarker 128

setNamedMarker: sets named marker A-Z (a-z) to a given time value and optionally renames the marker text to a string (-1 removes the marker)

Syntax: gtkwave::setNamedMarker which time_value [string]

Example:
gtkwave::setNamedMarker A 400 "Example Named Marker"
gtkwave::setNamedMarker A 400

setTabActive: sets the active tab in the viewer (0..getNumTabs-1)

Syntax: gtkwave::setTabActive which

Example:

```
gtkwave::setTabActive 0
```

setToEntry: sets the time in the "To:" box.

Syntax: `gtkwave::setToEntry time_value`

Example:
```
gtkwave::setToEntry 600
```

setTraceHighlightFromIndex: highlights or unhighlights the specified trace

Syntax: `gtkwave::setTraceHighlightFromIndex trace_index on_off`

Example:
```
gtkwave::setTraceHighlightFromIndex 2 on
gtkwave::setTraceHighlightFromIndex 2 off
```

setTraceHighlightFromNameMatch: highlights or unhighlights the specified trace

Syntax: `gtkwave::setTraceHighlightFromNameMatch fac_name on_off`

Example:
```
gtkwave::setTraceHighlightFromNameMatch top.des.clk on
gtkwave::setTraceHighlightFromNameMatch top.clk off
```

setTraceScrollbarRowValue: sets the scrollbar for traces a number of traces down from the very top

Syntax: `gtkwave::setTraceScrollbarRowValue scroller_value`

Example:
```
gtkwave::setTraceScrollbarRowValue 10
```

setWindowStartTime: scrolls the traces such that the start time is at the left margin (as long as the zoom level permits this)

Syntax: `gtkwave::setWindowStartTime start_time`

Example:
```
gtkwave::setWindowStartTime 100
```

setZoomFactor: sets the zoom factor for the trace data (i.e., how compressed it is with respect to time)

Syntax: `gtkwave::setZoomFactor zoom_value`

Example:
```
gtkwave::setZoomFactor -3
```

setZoomRangeTimes: sets the visible time range for the trace data

Syntax: `gtkwave::setZoomRangeTimes time1 time2`

Example:
```
gtkwave::setZoomRangeTimes 100 217
```

showSignal: sets the scrollbar for traces a number of traces down from the very top (0), center (1), or bottom (2)

Syntax: `gtkwave::setTraceScrollbarRowValue scroller_value position`
Example:
```
gtkwave::setTraceScrollbarRowValue 10 0
```

signalChangeList: returns time and value changes for the signals indicated by the argument names

Syntax: `gtkwave::signalChangeList signal_name options`

Where `options` is are one or more of the following:

`-start_time` start-time (default 0)

`-end_time` end-time (default last sample in dump file)

`-max` maximum-number-of-samples (default 0x7fffffff)

`-dir` forward|backward (default forward)

The function returns a Tcl list of value changes for the `signal-name` starting at `start-time` and ending at `end-time` or an empty list in any other case.

Even members of the list hold the time of change and the odd members hold the value that is associated with the time the precedes it. Values are given as strings in the base of the signal.

If `signal-name` is not present then the first highlighted signal is taken.

Length of the list is defined by both `end-time` and `max`, whichever comes first.

To specify backward search, `end-time` should be smaller than `start-time` or `dir` should have the value of `backward` and `end-time` is not defined.

A conflict between timing (start/end-time) and direction (forward/backward) returns an empty list.

Examples:

1. prints the first 100 changes of the signal tb.HDT.cpu.CS starting at time 10000

```
set signal tb.HDT.cpu.CS
set start_time 10000
foreach {time value} [gtkwave::signalChangeList $signal -start_time
$start_time -max 100] {
    puts "Time: $time value: $value"
}
```

2. retrieve the value of tb.HDT.cpu.CS at time 123456

```
    lassign [gtkwave::signalChangeList tb.HDT.cpu.CS -start_time
123456
-max 1] dont_care value
```

unhighlightSignalsFromList: unhighlights the facilities contained in the list argument

Syntax: set num_unhighlighted [gtkwave::unhighlightSignalsFromList list]

Example:
```
set clk48 [list]
lappend clk48 "$facname1"
lappend clk48 "$facname2"
...
set num_highlighted [ gtkwave::unhighlightSignalsFromList $clk48 ]
```

Tcl Callbacks

When gtkwave performs various functions, global callback variables prepended with gtkwave:: are modified within the Tcl interpreter. By using the trace write feature in Tcl, scripts can achieve a very tight integration with gtkwave. Global variables which may be used to register callback procedures are as follows:

gtkwave::cbCloseTabNumber contains the value returned is the number of the tab which is going to be closed, starting from zero. As this is set before the tab actually closes, scripts can interrogate for further information.

gtkwave::cbCloseTraceGroup contains the name of the expanded trace or trace group being closed.

gtkwave::cbCurrentActiveTab contains the number of the tab currently selected. Note that when new tabs are being created, this callback sometimes will oscillate between the old and new tab number, finally settling on the new tab being created.

gtkwave::cbError contains an error string such as "reload failed", "gtkwave::loadFile prohibited in callback", "gtkwave::reLoadFile prohibited in callback", or "gtkwave::setTabActive prohibited in callback".

gtkwave::cbFromEntryUpdated contains the value stored in the "From:" widget when it is updated.

gtkwave::cbOpenTraceGroup contains the name of a trace being expanded or trace group being opened.

gtkwave::cbQuitProgram contains the tab number which initiated a Quit operation. Tabs are numbered starting from zero.

gtkwave::cbReloadBegin contains the name of a trace being reloaded. This is called at the start of a reload sequence.

gtkwave::cbReloadEnd contains the name of a trace being reloaded. This is called at the end of a reload sequence.

gtkwave::cbStatusText contains the status text which goes to stderr.

gtkwave::cbTimerPeriod contains the timer period in milliseconds (default is 250), and this callback is invoked every timer period expiration. If Tcl code modifies this value, the timer period can be changed dynamically.

gtkwave::cbToEntryUpdated contains the value stored in the "To:" widget when it is updated.

gtkwave::cbTracesUpdated contains the total number of traces. This is called when traces are added, deleted, etc. from the viewer.

gtkwave::cbTreeCollapse contains the flattened hierarchical name of the SST tree node being collapsed.

gtkwave::cbTreeExpand contains the flattened hierarchical name of the SST tree node being expanded.

gtkwave::cbTreeSelect contains the flattened hierarchical name of the SST tree node being selected.

gtkwave::cbTreeSigDoubleClick contains the name of the signal being double-clicked in the signals section of the SST.

gtkwave::cbTreeSigSelect contains the name of the signal being selected in the signals section of the SST.

gtkwave::cbTreeSigUnselect contains the name of the signal being unselected in the signals section of the SST.

gtkwave::cbTreeUnselect contains the flattened hierarchical name of the SST tree node being unselected.

An example Tcl script follows to illustrate usage.

```
proc tracer {varname args} {
    upvar #0 $varname var
    puts "$varname was updated to be \"$var\""
}

proc tracer_error {varname args} {
    upvar #0 $varname var
    puts "*** ERROR: $varname was updated to be \"$var\""
}

set ie [ info exists tracer_defined ]
if { $ie == 0 } {
        set tracer_defined 1

        trace add variable gtkwave::cbTreeExpand write "tracer
gtkwave::cbTreeExpand"
        trace add variable gtkwave::cbTreeCollapse write "tracer
gtkwave::cbTreeCollapse"

        trace add variable gtkwave::cbTreeSelect write "tracer
gtkwave::cbTreeSelect"
        trace add variable gtkwave::cbTreeUnselect write "tracer
gtkwave::cbTreeUnselect"

        trace add variable gtkwave::cbTreeSigSelect write "tracer
gtkwave::cbTreeSigSelect"
        trace add variable gtkwave::cbTreeSigUnselect write "tracer
gtkwave::cbTreeSigUnselect"

        trace add variable gtkwave::cbTreeSigDoubleClick write
"tracer gtkwave::cbTreeSigDoubleClick"
        }

puts "Exiting script!"
```

Index

Illustration Index

Alphabetical Index

Software Failure.   Press left mouse button to continue.

Guru Meditation #0100000C.000FE800