CVC

Quick Start Guide

Revision: 8 June 2014
www.tachyon-da.com

Contents

1 Getting Started

1.1 Introduction

OSS CVC is a high performance Verilog HDL compiled simulator released under a modifed version of the Perl Artistic License 2.0 with dual track single copyright holder licensing. CVC supports the full IEEE 1364 standard including accurate delay gate and SDF annotation. CVC compiles directory to a cvcsim file that is then executed under X86 64 or 32 bit Linux.

CVC contains some features from SystemVerilog that makes CVC easier to use with modern design methods such as System C or for simulating the output of ESL tools or synthesizers. CVC supports the DPI PLI interface which allows direct calling of C or C++ functions from Verilog models or direct calling of Verilog functions or tasks from C/C++ code. The new SystemVerilog 2-state types corresponding to C/C++ types have been added to make the DPI interface efficient and seamless. Enough SystemVerilog features have been added to run all popular PFGA libraries.

However, OSS CVC will remain a static C or Pascal type language without dynamic classes.

1.2 Computer System Requirements

OSS CVC currently only runs only on X86 Linux It runs on both 32 bit and 64 bit platforms. The GNU Compiler Collection (gcc) and the GNU assembler (gas) must also be installed. These are almost always included in the default install of Linux distributions. Each system must also contain the math lib (-lm) and zip lib (-lz) for each intended architecture (32/64).

OSS CVC is shipped with full source. CVC binaries can be made using the included make file or you can purchase Enterprise OSS CVC to receive support and be supplied with carefully tested CVC binaries. See the OSS CVC Perl artistic style open source license in your release directory or at the top of every source file.

1.3 Installation

Installation of CVC is easy because all you need to do is copy the one self contained CVC binary to a directory (usually /usr/local/bin or /usr/bin/) that is on your PATH environment variable. Another option is to change your PATH environment variable to include the directory with the 'cvc' binary in it.

Currently Enterprise support binaries are made on one of the RHEL (Centos/SL) linux systems (EL4x, EL5x or EL6x).

We recommend you run cvc64 because it is slightly faster on modern CPUs. Although CVC64 uses more memory. For large designs and FST value change dump files larger than 2GB, you must run cvc64.

Prior to running CVC, make sure your system configuration is correct by running the 'checkcvc64' and checkcvc in the installation bin directory (you can also make the check binaries that are in the checkcvc.src directory). The following message should be printed:

"System configuration is fine for CVC compilation."

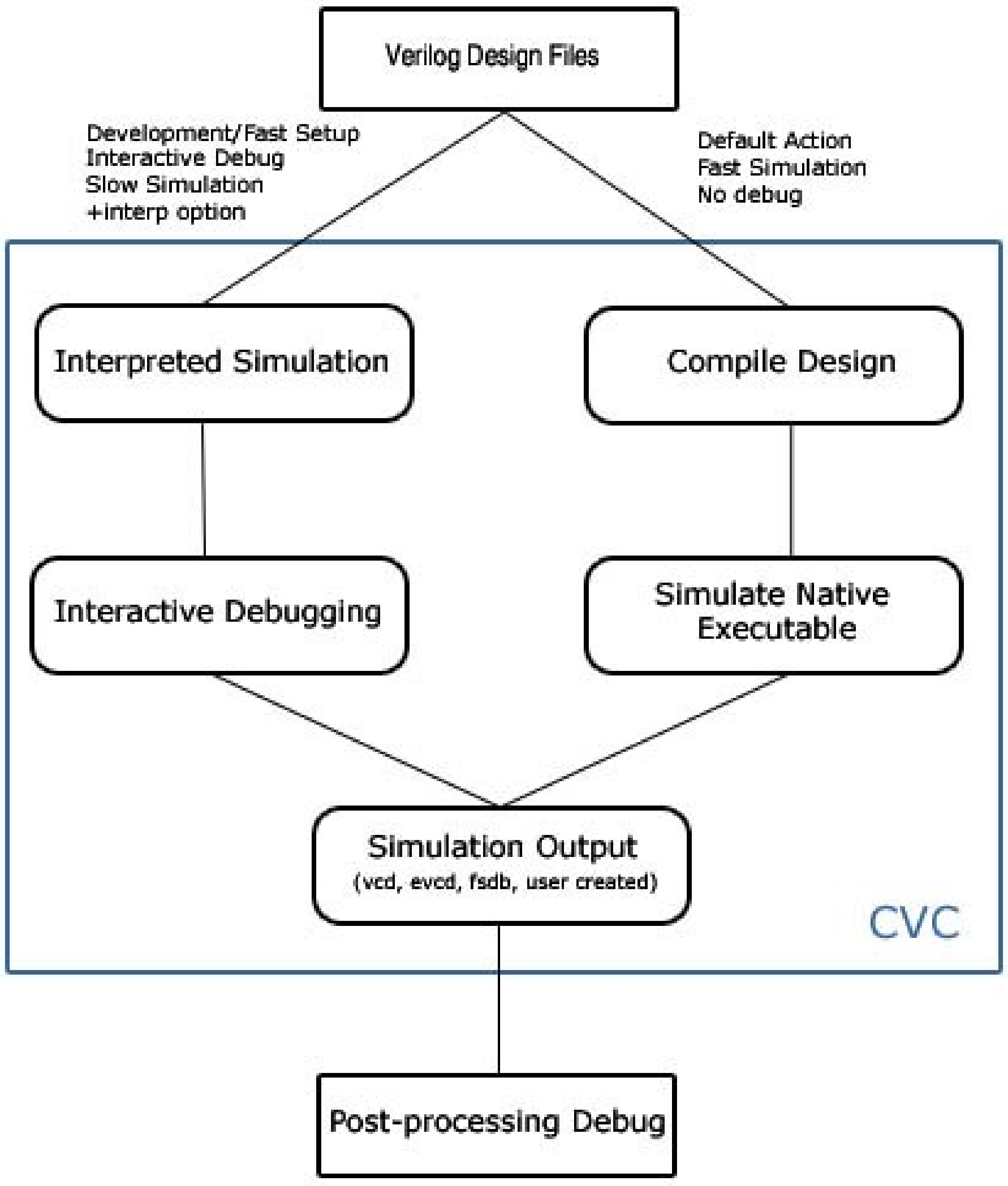
If anything other than this message is output and you have purchased OSS CVC Enterprise support, contact us..

We recommend that you next run the various installation tests in the "tests_and_examples" directory. You should always run the basic functionality test in the install.test directory by executing the "inst_test.sh" script. There should be no diff output messages printed. See the README file in the install.test directory for more information. The install scripts assume a cvc64/cvc binaries is either already installed on the PATH environment variable or if you want to test before installing in the OSS CVC release bin directory or in /tmp so you can be sure CVC installation is good before installing it. Use the cvc64 flag as an argument to any install test script to test 64 bit 'cvc64'.

The PLI tests need only be run if you are going to use CVC with the PLI. See the README files in each of the "tests_and_examples" sub-directories for detailed instructions on running the tests.

2 CVC interpreted/compiled design process

2.1 CVC dual verification flow



2.2 Ability to run in interpreted/compiled modes

OSS CVC' runs in either compiled or interpreted mode sets it apart. When doing initial design, CVC offers the fastest design elaboration of any Verilog simulator allowing for elaboration of even the most complex designs in only a few seconds. When doing exhaustive design testing, CVC's fast simulation speed allows more comprehensive testing. When run in interpreted mode, CVC allows debugging designs using a gdb style programmable integrated debugger.

2.3 Comparison of interpreted/compiled mode advantages

Features	Interpreted Mode	Compiled Mode
Elaboration Time	X	
Simulation Speed		X
PLI	X	X
Interactive Debugger	X	

3 Compiled Mode Simulation

3.1 Compiling a design

CVC compiles a design by default. To run the interpreter CVC must be run with the '+interp' option (see section 4 on running on interpreted mode). For basic compilation invoke CVC on the command line by typing:

```
cvc64 mytest.v
```

This creates a native executable named 'cvcsim'.4 Now all that is needed is execute the simulation by running the created executable:

```
cvcsim
```

To compile and immediately execute you can run with the '+exe' option. This option compiles into the default 'cvcsim' and creates another process to run it:

```
cvc +exe mytest.v
```

To save a compiled executable to a name other than 'cvcsim' a user can use the '-o [name of executable]' option on the command line.   For example to create an executable named 'mytest' run the following:

```
cvc -o mytest mytest.v
```

OSS CVC has now created a 'mytest' which will execute the simulation.

3.2 Optimizing a design with -O option

Optimizing compilation can be done with passing '-O' option on the command line. To run a long regression it may be wise to use the -O optoin to increase simulation speed.

```
cvc64 -O -f myslowsim.vc
```

For many designs, use of -O will increase simulation speed by up to 30% without much cost in compilation time. There are some designs for which -O leads to unacceptably long compilation time or object output file size. Users must decide whether this option is best on a per design basis. It is normally fine to use this option for most RTL designs.

3.3 Optimizing a gate-level design with -Ogate option

The '-Ogate' option can be used to optimize simulation speed for gate level designs. It invokes the gate simulation subset of the -O optimizations. For large gate level designs, compilation time using the -O option may be very long. For large gate level designs, most simulation speed improvements will still be achieved with much shorter compile time.

3.4 Optimizing a design with the +2state option

Using the **+2state** option may increase compiled simulation performance by simulating in 2-state mode. All variables are initialized to 0 and simulated using only 0/1 values, x/z values will be converted to 0. Simulating in 2-state mode may increase performance up to 50% without having to change the design itself.

Alternatively, the **+2state_no_wires** option may be used. It is the same as **+2state** except wires remain initialized and simulated using 4-state values. This allows existing designs that depend on 4-state wire values to simulate as expected and initialize from the x/z state.

CVC now supports SystemVerilog 2-state variable types (bit/int/byte/shotint/longint) which can also be used.   These types can be used in place of regular 4-state Verilog types, when a variable is known to only contain 2-state values (1/0).

3.5 Optimizing a design with the +nbaopt option

Some Verilog coding styles always code inter-assignment delays (usually 1) for non blocking assigns (NBAs) in RTL. Such delays slow down simulation and are not needed by modern synthesis algorithms, but can make debugging easier. Once your design is debugged and you are sure there are no clock cycle problems, use this option to significantly speed up simulation.

3.6 Compiling a design for use with PLI

Compiling a design that uses the PLI may need to use the +acc+[level number] option. This informs the compiler of the specific access level the PLI needs to perform correctly. When passing a PLI dynamic library on the command line CVC assumes minimal PLI access. Running your PLI without the use of +acc+ may result in incorrect results. The levels must be in the range of 1 to 4:

+acc+1 or +acc

Enable all PLI capabilities except value change callbacks and delay annotation.

+acc+2

All features above plus value change callbacks.

+acc+3

All features above plus module path delay.

+acc+4

All features above plus gate delay annotation.

The +acc+ option decreases performance as the level number increases. Due to the performance degradation of using +acc+ options for an entire design a better method is to use the -optconfigfile option for setting specific per design element PLI access levels. Using +acc+2 will set callbacks for an entire design which will slow simulation so it is desirable to only set callbacks for the modules/nets which need to check for PLI callbacks. For example, if you know your design only needs to have callbacks for a specific module ('testbench') this can be done with the following line using an option configuration file:

file: myoptions.opt

```
module {testbench} {acc_cbk};
```

See CVC log file generated by running 'cvc -h' for more information (file is also available in the release doc directory as cvc_help.txt). Also see the toggle_coverage.txt file in the doc directory for more information on using optconfigfiles since net toggle recording and reporting options are also selected using optconfigfiles (or for an entire design with options).

3.7 Constraints on created executables

Compiled executables still need to perform some initial setup using the original Verilog source files which were originally used to compile the design. Due to this fact some restrictions are necessary to allow previously compiled executables to run and be reusuable.

If any of these have changed, the design must be recompiled:

- Any Verilog source files have been modified
- Any files have been moved to new directories or replaced with links
- You want to simulate with different Verilog command line options
- You want to run the CVC executable (cvcsim) from a different directory

3.8 Using +large for large designs using 64-bit CVC

Compile with the +large option may be necessary for lvery arge designs compiled with cvc64. This is mostly caused by a design using large memories or large number of gates. Use the large model if you get the following error during compilation:

'relocation truncated to fit: R_X86_64_32S'

This will tell CVC to compile in 64-bit large model. It is best to avoid using the +large option if possible since it may slow down simulation a small amount.

4 Interpreted Mode Simulation

4.1 Simulation in interpreted mode

When doing initial design work with frequent changes, your may find it beneficial to run the interpreter. This saves time consumed by compilation when simulation speed is not the primary concern. Another good reason to run the interpreter is to use the interactive debugger which is not available in compiled mode. To run in interpreted mode simple run with '+interp' option. A simple interpreter run would resemble the following:

```
cvc +interp mytest.v
```

4.2 Using the XL with GDB style command based interactive debugger

When running in interpreted mode (+interp option), a command based interactive debugger is available. The debugger follows the old Verilog XL interactive debugger as documented in 1995 IEEE 1364 LRM. The command based interactive debugger can be entered using either the $stop system task executed from your Verilog source or by pressing the shell interrupt key (usually ctrl+c) during simulation. Following Verilog XL, the '-s' command line option can be used to immediately enter the interactive debugger before simulation starts. The '-i [file name]' option can be used to execute interactive debugger commands from [file name] once interactive mode is entered. The $input system task can be used to execute Verilog statements from inside the interactive debugger.

OSS CVC also supports more modern debugger commands that are similar to GDB style debugging commands. The legacy Verilog XL debugger is entered and expects either Verilog source to be entered or the few special commands such as <number> to re-execute command <number>, . (dot) to continue simulation, and -<number> to disable an executing Verilog statement with delay controls. The new debugger commands start with the : (colon) character. For example to set an instance specific break point type ":ibreak file1:44". All command based debugger commands are ended when CR (carriage return) is pressed.   For multi-line commands the CR must be escaped.

If you prefer to debug using a wave form debugger. Both CVC compiled and interpreted modes support writing of VCD/EVCD/FST value change dump files. We recommend using FST format with the “+fst_parallel2=on” option and debugging wave forms using the open source gtkwave program because this combination has the lowest value change writing overhead and produces smallest and fastest loading dumpvars files.

4.3 Getting command debugger help

The interactive debugger contains a help system that works like the gdb help system. Type “:help [topic]” to cause a brief help message to be printed.   To get help for the :ibreak command type: “:help :ibreak”.   For extended debugger command help the : (colon) is required, but for help topics it must not be typed.   See the file 'dbg_help.txt' that contains every help topic collected together in the doc/ directory.

Here is the output from typing “:help” after entering the interactive debugger.

debugging: entering and supplying input for +interp interactive debugging.

compiling: No interactive debugger when compiling – alternatives.

tracing: Separate batch tracing system for debugging.

differences: Interactive debugger differences from XL behavior.

statements: Using traditional interactive Verilog statement debugger.

data: Examining variables and expressions.

source: Listing source and navigating between files.

scope: Setting and navigating between instance and task scopes.

break points: Behavioral statement breakpoints and single stepping.

changes breaking: Breaking on net changes and edges.

history: Command history and history number command enable/disable.

info: Determining and setting debugger status and settings.

tasks: System task and functions useful for interactive debugging.

commands: A list of :[command] debugger commands.

5 Dumping Simulation State

5.1 Supported dumping formats

CVC has built in support for VCD and FST dumping formats. Value change dump (VCD) is the standard uncompressed format defined by the Verilog LRM.   FST is the format used with the GTKWave program. FST produces the smallest compressed dumpvars output files and requires the least writing overhead during simulation of any available wave form formats on any Verilog simulator.

CVC supports parallel FST dumping which can greatly speed up simulation times when dumping FST format, see the **+fst+parallel2=on** option. We strongly recommend using FST parallel mode.   FST also has, **+fst+parallel=on.** This mode may be required on olderX86 CPU types without modern multi-core support. +fst+parallel2=on mode will use up to 3 simultaneous cores so should only be run on systems that have unused additional cores. Alternatively parallel FST duimping can be turned on by setting the CVC_FST_PARALLEL2 environment variable.

5.2 System calls for dumping state

The following system calls and their FST equivalent are built in:

LRM Defined	**FST Version**
$dumpvars	$fstDumpvars
$dumpall	$fstDumpall
$dumpfile	$fstDumpfile
$dumpflush	$fstDumpflush
$dumplimit	$fstDumplimit
$dumpoff	$fstDumpoff
$dumpon	$fstDumpon

FST format contains an extra system call, $fstDumprepack. This will turn on FST repacking on simulation finish, resulting in a smaller file but will take a bit longer at the end of simulation to do the repacking. FST parallel mode can be turned on and off using the $fstParallelon/$fstParalleloff system calls.

5.3 Dumping array (memory) values

Array values are not dumped by default since it is non-standard. However, this may be accomplished using **+dump_arrays** or **+dump_array_cells** on the command line.

+dump_arrays - Any array form allowed in a $dumpvars statement: entire arrays or selects of one array cell. The various instance and all forms for $dumpvars arguments will dump all arrays in the instances (or subtree). This may result in large dump files for large arrays which will slow down simulation.

+dump_array_cells - Only allow dumping of specific array elements in a $dumpvars statement. Each element must be passed as an argument to $dumpvars. This allows dumping of just the desired array elements - not the entire array.

5.4 Options for controlling dumping formats

The following commands can tell CVC to handle regular VCD system calls in Verilog as FST format:

+dump2fst - Convert all $dump* [VCD] system calls to $fst* [FST] format.

The following command line options may be used on the command line to turn on dumping for an entire design without requiring calls to turn on dumping in the Verilog source files:

+dumpvars - VCD dumpvars for all variables in a design.

+fstvars - FST (GTKWave format) value dumping for all variables in a design.

+fstvars_repack - Perform FST value dumping for all variables in a design with repacking on simulation finish. Results in smaller file but takes a bit longer to repack FST file.

+fst+parallel2=on - Turn on parallel FST dumping - mode 2. Same as **+fst+parallel** plus uses an additional thread (core) and memory to further increase FST dumping performance. This mode will use up to 3 simultaneous cores so should only be ran on systems with at least 4 cores (quad-core). Alternatively can be turned on by setting the CVC_FST_PARALLEL2 environment variable.

+fst+parallel=on - Turn on parallel FST dumping. This can result in 50% less simulation time when used on multi-core systems. We strongly recommend using parallel mode then dumping FST. Alternatively, parallel dumping can be turned on by setting the *CVC_FST_PARALLEL* environment variable. If *CVC_FST_PARALLEL* is set it will always use parallel for every FST dumping simulation. If the cores are available on modern CPUs we recommend using +fst+parallel2=on.

5.5 Dumping port values (EVCD)

CVC supports extended value change dumping (EVCD) as defined by the Verilog LRM. The following system calls are supported:

$dumpports, $dumpportsall, $dumpportsflush, $dumpportslimit, $dumpportsoff, $dumpportson

6 Support

Further CVC commands and help can be run with '-h' to see a list of CVC options. Full support of CVC is provided if you purchase Enterprise OSS CVC support. Contact us directly more information.

7 SystemVerilog (SV) features in OSS CVC

Currently CVC supports the SystemVerilog Direct Programming Interface (DPI), and 2-state variable types. The DPI allows a simple interface between Verilog and C/C++/SystemC. See the examples.dpi directory for some examples on how to build and run DPI code with CVC. SystemVerilog variable types (bit/logic/int/byte/shortint/longint/chandle) are now supported. To parse SV types run with -sv or -sverilog. Use of the -sv_lib option for dpi_ PLI automatically turns on the -sv option.

OSS CVC is an IEEE 2005 standard Verilog simulator based on the C/Pascal compiler organization. It does not have classes (either static or dynamic).

SystemVerilog features to support simulation of common vendor libraries have been and will be added.

8 Improving Simulation Performance

8.1 General methods that effect performance.

First, 64-bit cvc64 and cvc32 versions may provide different performance results. Starting with EL 6.x libraries 32-bit and 64-bit simulation speeds are about the same with 64 bit even sometimes running faster on modern X86 64 processors. If your design is too large for your system RAM, 32-bit cvc is better because it uses up to 1/2 less RAM. One needs to experiment with 32/64 OSS CVC on a per design basis to determine which provides the best performance trade offs. 'cvc64' can be faster than 32-bit cvc for designs with complex expressions because the old X86 architecture does not have enough registers e

Second, compile with optimization turned on using the -O option. This option will do extra optimizations at compile time which increases compile time but results in a faster executable. Rarely -O will add an unacceptable amount of compile time (mostly for large gate-level designs), therefore using it may not be desirable.

Dumping an entire design can decrease performance since the simulator must record all changes and the overhead of the disk IO while writing the dump file. Therefore only turn on dumpvars for the variables whose state you need to monitor. Using +dump_arrays can also slow down simulation when dumping each element of a large array. Always use FST format and GTKwave for smallest dumpvars files and fastest debugging performance.

Use SystemVerilog 2-state types in your code. 2-state types (int/bit/byte/shortint/longint) use less storage and need less computation at simulation run time. For example declaring int i;, instead of integer i;, in your testbench code for loop counters can increase simulation speed.

Using the PLI also contains some overhead. The higher value you pass to the +acc+[number] option the more the PLI will slowdown simulation. If you are using the PLI to simply put/get values, or interface C/C++/SystemC, the DPI (direct programming interface) is better since it has less overhead, but does not allow interaction with timing based call backs.

8.2 Optimization options by design type.

The following options should be used with caution since not all errors and/or timing problems may be caught.

For procedural **RTL** simulations, these options an be selected:

-O

Optimize. For many designs use of -O will increase simulation speed by up to 30% without much cost in compilation time. There are still some designs for which -O leads to unacceptably long compilation time or object output file size. Users must decide whether this option is best on a per design basis.

+2state/+2state_no_wires

Use the +2state option to simulate without x and z values if your design can be simulated with

2-state (1/0) values. This option can increase simulation performance up to 2X. There is also the +2state_no_wires option which only simulates reg but not wires in two state. This option is sometimes needed if x/z values are needed for a design to initialize property.

+nbaopt

This causes non blocking assignments (NBAs) to always simulate without delays. Some design styles always put a delay on NBAs, but the delay is really not needed. If there are RTL timing problems, this can cause incorrect simulation results with no errors or warnings.

+unroll_loops

Unroll loops whose number of iterations can be determined at compile time or upon entry to the loop. This option makes the compiled executable larger, and may or may not make simulation faster. Generally, +unroll_loops will not speed up simulation.

For accurate **gate** and **path** level simulation, these options can be selected:

-Ogate

Optimize gate-level. Subset of -O optimizations for gate level. Use this option instead of -O for large gate-level designs when the -O option results in very long compilation time. This option increases simulation speed of large gate-level designs.

+notimingchecks

Turn off specify timing checks. Setup and hold type timing violations will not be caught. Use of this option can increase simulation speed by up to 25% at the cost of less checking of design timing problems.

+mipdopt

This option simplifies the complex Verilog algorithm for annotated module input port (MIPD) delays. As long as not gate delay path through a component is longer than the sum of annotated MIPD and path delays, this option will work. Use of this option can increase simulation speed by up to 33% at the cost of less checking of design timing problems.

9 Frequently Asked Questions

9.1 What platforms does OSS CVC run on?

OSS CVC runs on X86 architecture Linux. CVC can run on 32 or 64 bit processors. To run CVC in 64 bit run cvc64. OSS CVC only runs on X86/64 architectures because it generates machine instructions.

9.2 Can I run my PLI with OSS CVC?

Yes. See section 3.6.

9.3 Does OSS CVC support SystemVerilog Direct Programming Interface (DPI)?

Yes. You can compile in your dynamic library of C/C++ routines with the -sv_lib [lib.so] option. See the examples.dpi directory in the release directory for examples of how to build and run DPI with OSS CVC. If you do not need full vpi_ event communication dpi_ is much faster because there is no overhead. It just uses the Linux ABI calling conventions.

9.4 How do I learn how to use the VPI and DPI interfaces?

See the examples in the test_and_examples directory. For getting started with the vpi_ interface, study the examples in the examples.vpi directory. It contains examples of common usage of the VPI and is a good place to look prior to building your VPI application. Likewise for the DPI, look at the examples.dpi directory for common usage of how to build and link DPI applications.

9.5 Does CVC have toggle coverage?

Yes. CVC has built in net toggle coverage. Read the CVC coverage document ('how-to-use-new-toggle-coverage-feature.README' in the doc directory) for detailed instructions on how to use CVC

CVC concise wire toggle recording and reporting. Wire toggle coverage has been completely re-written to allow per instance recording and reporting and exclusion of wires that are tied to fixed values. Best way to learn to use CVC new concise toggle overage feature is to run the various toggle coverage .vc files in the tests_and_examples/some-benchmarks.dir/verilog_da_bnchmrks/das_cpu directory. See the README file for more instructions. das_cpu is a older cpu design that was fabricated so it has a realistic cpu model and good toggle coverage. New OSS CVC toggle coverage reporting using simple format ascii files for reporting that can easily be analyzed with your custom scripts or programs.

9.6 Can I run in interpreted mode?

Yes. CVC runs in compiled mode by default and in interpreted mode if the '+interp' option is given.

9.7 Why would I want to run in interpreted mode?

Some of the reasons to use the interpreted mode as opposed to the default compiled:

- Compile time vs. runtime
- The need for the interactive debugger ($stop).
- Various interpreter run time warning are not emitted by CVC compiled code. For example x/z loop count variables and out of range selects mimic interpreted results but no warning messages are printed.
- The need to use the non standard reason flag vpiAddDriver with the PLI vpi_put_value call.

9.8 Why is compile time slow?

We believe OSS CVC's compile time is good, if you experience a design which is abnormally slow please contact us with a detailed description or Verilog source of the slow compile time. If you are running with optimizing compile option '-O' try running without it. CVC's **-pipe** option can greatly decrease compile time. Use of **-pipe** can decrease overall compile time by up to 2X.

9.9 Can I compile and link separate libraries?

No. You must recompile your designs as a whole without linking precompiled libraries. This is an area we are working on.

9.10 What dumping formats does CVC support?

CVC supports VCD/EVCD/FST, see section 5 on dumping simulation state.

9.11 How do I dump array (memory) values?

With +dump_arrays or +dump_array_cells, see section 5.3.

9.12 Why do I get the following error: 'truncated to fit: R_X86_64_32S'

If you get this error run with the +large option. See section 3.7.

9.13 Do I need to recompile my design every time I need to run it?

No, see section 3.6 on restrictions on rerunning previously compiled executables.

9.14 Why doesn't CVC support $stop in compiled mode?

Currently CVC ignores $stop unless the +interp option is selected. It does this because there is no interactive mode (no ":" debugger), if you need the interactive mode run in interpreted mode (+interp). Otherwise to terminate simulation use $finish. We are not sure how to handle $stop in the compiler. It could be changed so that $stop is treated the same as $finish in CVC. We are interested in user feedback regarding this.

9.15 Why doesn't CVC support $save/$restart in compiled mode?

Operating system level programs and system calls exist in modern operating systems so it is better to simply stop a process and restart it. There is no need for CVC to even know it was stopped and its process image saved to disk.

9.16 Why can't I use the run time debugger in compiled mode?

The CVC interpreter debugger can't be supported by the compiler because the extra checking would slow down simulation too much. You can run the interpreter (+interp) to debug your hardware designs.

9.17 Why doesn't CVC support $save/$restart in compiled mode?

Operating system level programs and system calls exist in modern operating systems so it is better to simply stop a process and restart it. There is no need for CVC to even know it was stopped and its process image saved to disk.

9.18 What is the new +xprop option and why would I want to use it?

There is a problem with IEEE 1364 expression evaluation when values are unknown (X), Namely, normal Verilog semantics is too optimistic. If you select the +xprop option, if and case statements plus event controls will be more pessismistic and evaluate any expressions (say both the if and else) if there is a possibility a selector can be x. This slows down simultaion but can find gate level problems earlier. See the cvc_xpropagation.pdf file in the doc directory for detailed instructions and list of options to use. The problem with the +xprop feature is that it requires a very specific synthesis methodology and is also inaccurate in some ways compared to Spice simulation.

9.19 How do I report bugs?

Before reporting a bug check the Changelog file to see if the problem is already known. Also, make sure you are running the latest version of OSS CVC. Bug reports should include a small failing example if possible. To have Tachyon Design Automation fix bugs and provide support you must purchase Enterprise OSS CVC support.

9.20 Why doesn't CVC mimic XL style port collapsing? i.e. why are some nets multiply driven in XL but not in CVC?

CVC follows the 1364 LRM and treats input and output ports as no delay continuous assignments and inout ports as non strength reducing "virtual" tran gates. This has the advantage that there is no need for changing wire types when wires with different types are collapsed into the same net and allows warnings to be emitted for incorrectly declared ports. A port is incorrectly declared if an input port has a driver on the lowconn side or if an output port has a driver on the highconn side.

Because many older designs depend on the XL port collapsing algorithm which silently changes port type depending on pattern of net drivers, CVC supports the +change_port_type command option that causes CVC to change ports according to driving pattern to mimic the XL port collapsing algorithm. CVC always emits a warning message if there is a possibility that a port may be changed to inout by XL port collapsing algorithm (messages is an inform if +change_port_type option is selected). Use +suppress_warnings+3107+3108+ to suppress the warning if you intend backward direction signal flow to be blocked by a port.

There are many advantages to avoiding the XL algorithm such as: fewer multi driver nets, no need to distinguish simulated nets in PLI and in system task output, etc.