VERILATOR

Verilator

Release 4.215

Wilson Snyder

2021-11-27

GETTING STARTED

CHAPTER

ONE

OVERVIEW

Welcome to Verilator!

The Verilator package converts Verilog[1] and SystemVerilog[2] hardware description language (HDL) designs into a C++ or SystemC model that after compiling can be executed. Verilator is not a traditional simulator, but a compiler.

Verilator is typically used as follows:

1. The **`verilator`** executable is invoked with parameters similar to GCC, or other simulators such as Cadence Verilog-XL/NC-Verilog, or Synopsys VCS. Verilator reads the specified SystemVerilog code, lints it, optionally adds coverage and waveform tracing support, and compiles the design into a source level multithreaded C++ or SystemC "model". The resulting model's C++ or SystemC code is output as .cpp and .h files. This is referred to as "Verilating" and the process is "to Verilate"; the output is a "Verilated" model.

2. For simulation, a small user written C++ wrapper file is required, the "wrapper". This wrapper defines the C++ standard function "main()" which instantiates the Verilated model as a C++/SystemC object.

3. The user C++ wrapper, the files created by Verilator, a "runtime library" provided by Verilator, and if applicable SystemC libraries are then compiled using a C++ compiler to create a simulation executable.

4. The resulting executable will perform the actual simulation, during "simulation runtime".

5. If appropriately enabled, the executable may also generate waveform traces of the design that may be viewed. It may also create coverage analysis data for post-analysis.

The best place to get started is to try the *Examples*.

[1] Verilog is defined by the *Institute of Electrical and Electronics Engineers (IEEE) Standard for Verilog Hardware Description Language*, Std. 1364, released in 1995, 2001, and 2005. The Verilator documentation uses the shorthand e.g. "IEEE 1394-2005" to refer to the e.g. 2005 version of this standard.

[2] SystemVerilog is defined by the *Institute of Electrical and Electronics Engineers (IEEE) Standard for SystemVerilog - Unified Hardware Design, Specification, and Verification Language*, Standard 1800, released in 2005, 2009, 2012, and 2017. The Verilator documentation uses the shorthand e.g. "IEEE 1800-2017" to refer to the e.g. 2017 version of this standard.

CHAPTER

TWO

EXAMPLES

This section covers the following examples:

- *Example C++ Execution*
- *Example SystemC Execution*
- *Examples in the Distribution*

2.1 Example C++ Execution

We'll compile this example into C++. For an extended and commented version of what this C++ code is doing, see `examples/make_tracing_c/sim_main.cpp` in the distribution.

First you need Verilator installed, see *Installation*. In brief, if you installed Verilator using the package manager of your operating system, or did a **`make install`** to place Verilator into your default path, you do not need anything special in your environment, and should not have VERILATOR_ROOT set. However, if you installed Verilator from sources and want to run Verilator out of where you compiled Verilator, you need to point to the kit:

```
# See above; don't do this if using an OS-distributed Verilator
export VERILATOR_ROOT=/path/to/where/verilator/was/installed
export PATH=$VERILATOR_ROOT/bin:$PATH
```

Now, let's create an example Verilog, and C++ wrapper file:

```
mkdir test_our
cd test_our

cat >our.v <<'EOF'
  module our;
     initial begin $display("Hello World"); $finish; end
  endmodule
EOF

cat >sim_main.cpp <<'EOF'
  #include "Vour.h"
  #include "verilated.h"
  int main(int argc, char** argv, char** env) {
      VerilatedContext* contextp = new VerilatedContext;
      contextp->commandArgs(argc, argv);
      Vour* top = new Vour{contextp};
      while (!contextp->gotFinish()) { top->eval(); }
      delete top;
      delete contextp;
```

(continues on next page)

(continued from previous page)

```
      return 0;
  }
EOF
```

Now we run Verilator on our little example.

```
verilator -Wall --cc --exe --build sim_main.cpp our.v
```

Breaking this command down:

1. *-Wall* so Verilator has stronger lint warnings enabled.
2. *--cc* to get C++ output (versus e.g. SystemC or only linting).
3. *--exe*, along with our **sim_main.cpp** wrapper file, so the build will create an executable instead of only a library.
4. *--build* so Verilator will call make itself. This is we don't need to manually call make as a separate step. You can also write your own compile rules, and run make yourself as we show in *Example SystemC Execution*.)
5. An finally, **our.v** which is our SystemVerilog design file.

Once Verilator completes we can see the generated C++ code under the `obj_dir` directory.

```
ls -l obj_dir
```

(See *Files Read/Written* for descriptions of some of the files that were created.)

And now we run it:

```
obj_dir/Vour
```

And we get as output:

```
Hello World
- our.v:2: Verilog $finish
```

Really, you're better off using a Makefile to run the steps for you so when your source changes it will automatically run all of the appropriate steps. To aid this Verilator can create a makefile dependency file. For examples that do this see the `examples` directory in the distribution.

2.2 Example SystemC Execution

This is an example similar to the *Example C++ Execution*, but using SystemC. We'll also explicitly run make.

First you need Verilator installed, see *Installation*. In brief, if you installed Verilator using the package manager of your operating system, or did a **make install** to place Verilator into your default path, you do not need anything special in your environment, and should not have VERILATOR_ROOT set. However, if you installed Verilator from sources and want to run Verilator out of where you compiled Verilator, you need to point to the kit:

```
# See above; don't do this if using an OS-distributed Verilator
export VERILATOR_ROOT=/path/to/where/verilator/was/installed
export PATH=$VERILATOR_ROOT/bin:$PATH
```

Now, let's create an example Verilog, and SystemC wrapper file:

```
mkdir test_our_sc
cd test_our_sc

cat >our.v <<'EOF'
  module our (clk);
     input clk;  // Clock is required to get initial activation
     always @(posedge clk)
        begin $display("Hello World"); $finish; end
  endmodule
EOF

cat >sc_main.cpp <<'EOF'
  #include "Vour.h"
  int sc_main(int argc, char** argv) {
      Verilated::commandArgs(argc, argv);
      sc_clock clk{"clk", 10, SC_NS, 0.5, 3, SC_NS, true};
      Vour* top = new Vour{"top"};
      top->clk(clk);
      while (!Verilated::gotFinish()) { sc_start(1, SC_NS); }
      delete top;
      return 0;
  }
EOF
```

Now we run Verilator on our little example:

```
verilator -Wall --sc --exe sc_main.cpp our.v
```

This example does not use –build, therefore we need to explicitly compile it:

```
make -j -C obj_dir -f Vour.mk Vour
```

And now we run it:

```
obj_dir/Vour
```

And we get, after the SystemC banner, the same output as the C++ example:

```
SystemC 2.3.3-Accellera

Hello World
- our.v:4: Verilog $finish
```

Really, you're better off using a Makefile to run the steps for you so when your source changes it will automatically run all of the appropriate steps. For examples that do this see the `examples` directory in the distribution.

2.3 Examples in the Distribution

See the `examples/` directory that is part of the distribution, and is installed (in a OS-specific place, often in e.g. `/usr/local/share/verilator/examples`). These examples include:

examples/make_hello_c Example GNU-make simple Verilog->C++ conversion

examples/make_hello_sc Example GNU-make simple Verilog->SystemC conversion

examples/make_tracing_c Example GNU-make Verilog->C++ with tracing

examples/make_tracing_sc Example GNU-make Verilog->SystemC with tracing

examples/make_protect_lib Example using –protect-lib

examples/cmake_hello_c Example building make_hello_c with CMake

examples/cmake_hello_sc Example building make_hello_sc with CMake

examples/cmake_tracing_c Example building make_tracing_c with CMake

examples/cmake_tracing_sc Example building make_tracing_sc with CMake

examples/cmake_protect_lib Example building make_protect_lib with CMake

To run an example copy the example to a new directory and run it.

```
cp -rp {path_to}/examples/make_hello_c make_hello_c
cd make_hello_c
make
```

CHAPTER

THREE

INSTALLATION

This section discusses how to install Verilator.

3.1 Package Manager Quick Install

Using a distribution's package manager is the easiest way to get started. (Note packages are unlikely to have the most recent version, so *Git Quick Install*, might be a better alternative.) To install as a package:

```
apt-get install verilator   # On Ubuntu
```

For other distributions refer to Repology Verilator Distro Packages.

3.2 Git Quick Install

Installing Verilator from Git provides the most flexibility. For additional options and details see *Detailed Build Instructions* below.

In brief, to install from git:

```
# Prerequisites:
#sudo apt-get install git perl python3 make autoconf g++ flex bison ccache
#sudo apt-get install libgoogle-perftools-dev numactl perl-doc
#sudo apt-get install libfl2  # Ubuntu only (ignore if gives error)
#sudo apt-get install libfl-dev  # Ubuntu only (ignore if gives error)
#sudo apt-get install zlibc zlib1g zlib1g-dev  # Ubuntu only (ignore if gives error)

git clone https://github.com/verilator/verilator   # Only first time

# Every time you need to build:
unsetenv VERILATOR_ROOT  # For csh; ignore error if on bash
unset VERILATOR_ROOT  # For bash
cd verilator
git pull         # Make sure git repository is up-to-date
git tag          # See what versions exist
#git checkout master      # Use development branch (e.g. recent bug fixes)
#git checkout stable      # Use most recent stable release
#git checkout v{version}  # Switch to specified release version

autoconf         # Create ./configure script
./configure      # Configure and create Makefile
```

(continues on next page)

(continued from previous page)

```
make -j `nproc`  # Build Verilator itself (if error, try just 'make')
sudo make install
```

3.3 Detailed Build Instructions

This section describes details of the build process, and assumes you are building from Git. For using a pre-built binary for your Linux distribution, see instead *Package Manager Quick Install*.

3.3.1 OS Requirements

Verilator is developed and has primary testing on Ubuntu, with additional testing on FreeBSD and Apple OS-X. Versions have also built on Redhat Linux, and other flavors of GNU/Linux-ish platforms. Verilator also works on Windows Subsystem for Linux (WSL2), Windows under Cygwin, and Windows under MinGW (gcc -mno-cygwin). Verilated output (not Verilator itself) compiles under all the options above, plus using MSVC++.

3.3.2 Install Prerequisites

To build or run Verilator you need these standard packages:

```
sudo apt-get install git perl python3 make
sudo apt-get install g++  # Alternatively, clang
sudo apt-get install libgz  # Non-Ubuntu (ignore if gives error)
sudo apt-get install libfl2  # Ubuntu only (ignore if gives error)
sudo apt-get install libfl-dev  # Ubuntu only (ignore if gives error)
sudo apt-get install zlibc zlib1g zlib1g-dev  # Ubuntu only (ignore if gives error)
```

To build or run the following are optional but should be installed for good performance:

```
sudo apt-get install ccache  # If present at build, needed for run
sudo apt-get install libgoogle-perftools-dev numactl perl-doc
```

To build Verilator you will need to install these packages; these do not need to be present to run Verilator:

```
sudo apt-get install git autoconf flex bison
```

Those developing Verilator itself may also want these (see internals.rst):

```
sudo apt-get install gdb graphviz cmake clang clang-format-11 gprof lcov
sudo pip3 install sphinx sphinx_rtd_theme breathe
cpan install Pod::Perldoc
cpan install Parallel::Forker
```

Install SystemC

If you will be using SystemC (vs straight C++ output), download SystemC. Follow their installation instructions. You will need to set the *SYSTEMC_INCLUDE* environment variable to point to the include directory with `systemc.h` in it, and set the *SYSTEMC_LIBDIR* environment variable to point to the directory with `libsystemc.a` in it.

Install GTKWave

To make use of Verilator FST tracing you will want GTKwave installed, however this is not required at Verilator build time.

3.3.3 Obtain Sources

Get the sources from the git repository: (You need do this only once, ever.)

```
git clone https://github.com/verilator/verilator   # Only first time
## Note the URL above is not a page you can see with a browser, it's for git only
```

Enter the checkout and determine what version/branch to use:

```
cd verilator
git pull        # Make sure we're up-to-date
git tag         # See what versions exist
#git checkout master      # Use development branch (e.g. recent bug fix)
#git checkout stable      # Use most recent release
#git checkout v{version}  # Switch to specified release version
```

3.3.4 Auto Configure

Create the configuration script:

```
autoconf        # Create ./configure script
```

3.3.5 Eventual Installation Options

Before configuring the build, you have to decide how you're going to eventually install Verilator onto your system. Verilator will be compiling the current value of the environment variables *VERILATOR_ROOT*, *SYSTEMC_INCLUDE*, and *SYSTEMC_LIBDIR* as defaults into the executable, so they must be correct before configuring.

These are the installation options:

1. Run-in-Place from VERILATOR_ROOT

Our personal favorite is to always run Verilator in-place from its Git directory. This allows the easiest experimentation and upgrading, and allows many versions of Verilator to co-exist on a system.

```
export VERILATOR_ROOT=`pwd`   # if your shell is bash
setenv VERILATOR_ROOT `pwd`   # if your shell is csh
./configure
# Running will use files from $VERILATOR_ROOT, so no install needed
```

Note after installing (below steps), a calling program or shell must set the environment variable *VERILATOR_ROOT* to point to this Git directory, then execute `$VERILATOR_ROOT/bin/verilator`, which will find the path to all needed files.

2. Install into a specific location

You may eventually be installing onto a project/company-wide "CAD" tools disk that may support multiple versions of every tool. Tell configure the eventual destination directory name. We recommend the destination location include the Verilator version name:

```
unset VERILATOR_ROOT      # if your shell is bash
unsetenv VERILATOR_ROOT   # if your shell is csh
# For the tarball, use the version number instead of git describe
./configure --prefix /CAD_DISK/verilator/`git describe | sed "s/verilator_//"`
```

Note after installing (below steps), if you use modulecmd, you'll want a module file like the following:

```
set install_root /CAD_DISK/verilator/{version-number-used-above}
unsetenv VERILATOR_ROOT
prepend-path PATH $install_root/bin
prepend-path MANPATH $install_root/man
prepend-path PKG_CONFIG_PATH $install_root/share/pkgconfig
```

3. Install into a Specific Path

You may eventually install Verilator into a specific installation prefix, as most GNU tools support:

```
unset VERILATOR_ROOT      # if your shell is bash
unsetenv VERILATOR_ROOT   # if your shell is csh
./configure --prefix /opt/verilator-VERSION
```

Then after installing (below steps) you will need to add `/opt/verilator-VERSION/bin` to your `$PATH` environment variable.

4. Install System Globally

The final option is to eventually install Verilator globally, using configure's default system paths:

```
unset VERILATOR_ROOT      # if your shell is bash
unsetenv VERILATOR_ROOT   # if your shell is csh
./configure
```

Then after installing (below) the binaries should be in a location that is already in your `$PATH` environment variable.

3.3.6 Configure

The command to configure the package was described in the previous step. Developers should configure to have more complete developer tests. Additional packages may be required for these tests.

```
export VERILATOR_AUTHOR_SITE=1     # Put in your .bashrc
./configure --enable-longtests  ...above options...
```

3.3.7 Compile

Compile Verilator:

```
make -j `nproc`  # Or if error on `nproc`, the number of CPUs in system
```

3.3.8 Test

Check the compilation by running self-tests:

```
make test
```

3.3.9 Install

If you used any install option other than the *1. Run-in-Place from VERILATOR_ROOT* scheme, install the files:

```
make install
```

3.4 Verilator Build Docker Container

This Verilator Build Docker Container is set up to compile and test a Verilator build. It uses the following parameters:

- Source repository (default: https://github.com/verilator/verilator)
- Source revision (default: master)
- Compiler (GCC 9.3.0, clang 10.0.0, default: 9.3.0)

The container is published as `verilator/verilator-buildenv` on docker hub.

To run the basic build using the current Verilator master:

```
docker run -ti verilator/verilator-buildenv
```

To also run tests:

```
docker run -ti verilator/verilator-buildenv test
```

To change the compiler:

```
docker run -ti -e CC=clang-10 -e CXX=clang++-10 verilator/verilator-buildenv test
```

The tests that involve gdb are not working due to security restrictions. To run those too:

```
docker run -ti -e CC=clang-10 -e CXX=clang++-10 --cap-add=SYS_PTRACE --security-opt␣
↪seccomp=unconfined verilator/verilator-buildenv test
```

Rather then building using a remote git repository you may prefer to use a working copy on the local filesystem. Mount the local working copy path as a volume and use that in place of git. When doing this be careful to have all changes committed to the local git area. To build the current HEAD from top of a repository:

```
docker run -ti -v ${PWD}:/tmp/repo -e REPO=/tmp/repo -e REV=`git rev-parse --short␣
↪HEAD` --cap-add=SYS_PTRACE --security-opt seccomp=unconfined verilator/verilator-
↪buildenv test
```

3.4.1 Rebuilding

To rebuild the Verilator-buildenv docker image, run:

```
docker build .
```

This will also build SystemC under all supported compiler variants to reduce the SystemC testing time.

3.5 Verilator Executable Docker Container

The Verilator Executable Docker Container allows you to run Verilator easily as a docker image, e.g.:

```
docker run -ti verilator/verilator:latest --version
```

This will install the container, run the latest Verilator and print Verilator's version.

Containers are automatically built for all released versions, so you may easily compare results across versions, e.g.:

```
docker run -ti verilator/verilator:4.030 --version
```

Verilator needs to read and write files on the local system. To simplify this process, use the `verilator-docker` convenience script. This script takes the version number, and all remaining arguments are passed through to Verilator. e.g.:

```
./verilator-docker 4.030 --version
```

or

```
./verilator-docker 4.030 --cc test.v
```

If you prefer not to use `verilator-docker` you must give the container access to your files as a volume with appropriate user rights. For example to Verilate test.v:

```
docker run -ti -v ${PWD}:/work --user $(id -u):$(id -g) verilator/verilator:latest --
↪cc test.v
```

This method can only access files below the current directory. An alternative is setup the volume `-workdir`.

You can also work in the container by setting the entrypoint (don't forget to mount a volume if you want your work persistent):

```
docker run -ti --entrypoint /bin/bash verilator/verilator:latest
```

You can also use the container to build Verilator at a specific commit:

```
docker build --build-arg SOURCE_COMMIT=<commit> .
```

3.5.1 Internals

The Dockerfile builds Verilator and removes the tree when completed to reduce the image size. The entrypoint is set as a wrapper script (`verilator-wrap.sh`). That script 1. calls Verilator, and 2. copies the Verilated runtime files to the `obj_dir` or the `-Mdir` respectively. This allows the user to have the files to they may later build the C++ output with the matching runtime files. The wrapper also patches the Verilated Makefile accordingly.

There is also a hook defined that is run by docker hub via automated builds.

CHAPTER

FOUR

VERILATING

Verilator may be used in five major ways:

- With the `--cc` or `--sc` options, Verilator will translate the design into C++ or SystemC code respectively. See *C++ and SystemC Generation*.
- With the `--lint-only` option, Verilator will lint the design to check for warnings but will not typically create any output files.
- With the `--xml-only` option, Verilator will create XML output that may be used to feed into other user-designed tools. See `docs/xml.rst` in the distribution.
- With the `-E` option, Verilator will preprocess the code according to IEEE preprocessing rules, and write the output to standard out. This is useful to feed other tools, and to debug how "`define" statements are expanded.

4.1 C++ and SystemC Generation

Verilator will translate a SystemVerilog design into C++ with the `--cc` option, or into SystemC with the `--sc` option.

When using these options:

1. Verilator reads the input Verilog code, determines all "top modules" that is modules or programs that are not used as instances under other cells. If `--top-module` is used, then that determines the top module and all other top modules are removed, otherwise a `MULTITOP` warning is given.
2. Verilator writes the C++/SystemC code to output files into the `--Mdir` option-specified directory, or defaults to "obj_dir". The prefix is set with `--prefix`, or defaults to the name of the top module.
3. If `--exe` is used, Verilator creates makefiles to generate a simulation executable, otherwise it creates makefiles to generate an archive (.a) containing the objects.
4. If `--build` option was used, it calls *GNU Make* or *CMake* to build the model.

Once a model is built it is then typically run, see *Simulating (Verilated-Model Runtime)*.

4.2 Hierarchical Verilation

Large designs may take long (e.g. 10+ minutes) and huge memory (e.g. 100+ GB) to Verilate. In hierarchical mode, the user manually selects some large lower-level hierarchy blocks to separate from the larger design. For example a core may be the hierarchy block, and separated out of a multi-core SoC.

Verilator is run in hierarchical mode on the whole SoC. Verilator will make two models, one for the CPU hierarchy block, and one for the SoC. The Verialted code for the SoC will automatically call the CPU Verilated model.

The current hierarchical Verilation is based on `--lib-create`. Each hierarchy block is Verilated into a library. User modules of the hierarchy blocks will see a tiny wrapper generated by `--lib-create`.

4.2.1 Usage

Users need to mark one or more moderate size module as hierarchy block(s). There are two ways to mark a module:

- Write `/*verilator&32;hier_block*/` metacomment in HDL code.
- Add a `hier_block` line in a the *Configuration Files*.

Then pass the `--hierarchical` option to Verilator

Compilation is the same as when not using hierarchical mode.

```
make -C obj_dir -f Vtop_module_name.mk
```

4.2.2 Limitations

Hierarchy blocks have some limitations including:

- The hierarchy block cannot be accessed using dot (.) from upper module(s) or other hierarchy blocks.
- Signals in the block cannot be traced.
- Modport cannot be used at the hierarchical block boundary.
- The simulation speed is likely to not be as fast as flat Verilation, in which all modules are globally scheduled.
- Generated clocks may not work correctly if they are generated in the hierarchical model and pass up into another hierarchical model or the top module.

But, the following usage is supported:

- Nested hierarchy blocks. A hierarchy block may instantiate other hierarchy blocks.
- Parameterized hierarchy block. Parameters of a hierarchy block can be overridden using `#(.param_name(value))` construct.

4.2.3 Overlapping Verilation and Compilation

Verilator needs to run 2 + *N* times in hierarchical Verilation, where *N* is the number of hierarchy blocks. One of the two is for the top module which refers wrappers of all other hierarchy blocks. The second one of the two is the initial run that searches modules marked with */*verilator&32;hier_block*/* metacomment and creates a plan and write in *prefix*_hier.mk. This initial run internally invokes other *N* + 1 runs, so you don't have to care about these *N* + 1 times of run. The additional *N* is the Verilator run for each hierarchical block.

If :*-j {jobs}* option is specified, Verilation for hierarchy blocks runs in parallel.

If *--build* option is specified, C++ compilation also runs as soon as a hierarchy block is Verilated. C++ compilation and Verilation for other hierarchy blocks run simultaneously.

4.3 Cross Compilation

Verilator supports cross-compiling Verilated code. This is generally used to run Verilator on a Linux system and produce C++ code that is then compiled on Windows.

Cross compilation involves up to three different OSes. The build system is where you configured and compiled Verilator, the host system where you run Verilator, and the target system where you compile the Verilated code and run the simulation.

Currently, Verilator requires the build and host system type to be the same, though the target system type may be different. To support this, **./configure** and make Verilator on the build system. Then, run Verilator on the host system. Finally, the output of Verilator may be compiled on the different target system.

To support this, none of the files that Verilator produces will reference any configure generated build-system specific files, such as config.h (which is renamed in Verilator to config_build.h to reduce confusion.) The disadvantage of this approach is that include/verilatedos.h must self-detect the requirements of the target system, rather than using configure.

The target system may also require edits to the Makefiles, the simple Makefiles produced by Verilator presume the target system is the same type as the build system.

4.4 Multithreading

Verilator supports multithreaded simulation models.

With *--no-threads*, the default, the model is not thread safe, and any use of more than one thread calling into one or even different Verilated models may result in unpredictable behavior. This gives the highest single thread performance.

With *--threads 1*, the generated model is single threaded, however the support libraries are multithread safe. This allows different instantiations of model(s) to potentially each be run under a different thread. All threading is the responsibility of the user's C++ testbench.

With *--threads {N}*, where N is at least 2, the generated model will be designed to run in parallel on N threads. The thread calling eval() provides one of those threads, and the generated model will create and manage the other N-1 threads. It's the client's responsibility not to oversubscribe the available CPU cores. Under CPU oversubscription, the Verilated model should not livelock nor deadlock, however, you can expect performance to be far worse than it would be with proper ratio of threads and CPU cores.

The remainder of this section describe behavior with *--threads 1* or *--threads {N}* (not *--no-threads*).

VL_THREADED is defined in the C++ code when compiling a threaded Verilated module, causing the Verilated support classes become threadsafe.

The thread used for constructing a model must be the same thread that calls `eval()` into the model, this is called the "eval thread". The thread used to perform certain global operations such as saving and tracing must be done by a "main thread". In most cases the eval thread and main thread are the same thread (i.e. the user's top C++ testbench runs on a single thread), but this is not required.

When making frequent use of DPI imported functions in a multi-threaded model, it may be beneficial to performance to adjust the *--instr-count-dpi* option based on some experimentation. This influences the partitioning of the model by adjusting the assumed execution time of DPI imports.

The *--trace-threads* options can be used to produce trace dumps using multiple threads. If *--trace-threads* is set without *--threads*, then *--trace-threads* will imply *--threads 1*, i.e.: the support libraries will be thread safe.

With *--trace-threads 0*, trace dumps are produced on the main thread. This again gives the highest single thread performance.

With *--trace-threads {N}*, where N is at least 1, N additional threads will be created and managed by the trace files (e.g.: VerilatedVcdC or VerilatedFstC), to generate the trace dump. The main thread will be released to proceed with execution as soon as possible, though some blocking of the main thread is still necessary while capturing the trace. Different trace formats can utilize a various number of threads. See the *--trace-threads* option.

When running a multithreaded model, the default Linux task scheduler often works against the model, by assuming threads are short lived, and thus often schedules threads using multiple hyperthreads within the same physical core. For best performance use the **`numactl`** program to (when the threading count fits) select unique physical cores on the same socket. The same applies for *--trace-threads* as well.

As an example, if a model was Verilated with *--threads 4*, we consult:

```
egrep 'processor|physical id|core id' /proc/cpuinfo
```

To select cores 0, 1, 2, and 3 that are all located on the same socket (0) but different physical cores. (Also useful is **`numactl --hardware`**, or **`lscpu`** but those doesn't show Hyperthreading cores.) Then we execute:

```
numactl -m 0 -C 0,1,2,3 -- verilated_executable_name
```

This will limit memory to socket 0, and threads to cores 0, 1, 2, 3, (presumably on socket 0) optimizing performance. Of course this must be adjusted if you want another simulator using e.g. socket 1, or if you Verilated with a different number of threads. To see what CPUs are actually used, use *--prof-threads*.

4.4.1 Multithreaded Verilog and Library Support

$display/$stop/$finish are delayed until the end of an eval() call in order to maintain ordering between threads. This may result in additional tasks completing after the $stop or $finish.

If using *--coverage*, the coverage routines are fully thread safe.

If using the DPI, Verilator assumes pure DPI imports are thread safe, balancing performance versus safety. See *--threads-dpi*.

If using *--savable*, the save/restore classes are not multithreaded and must be called only by the eval thread.

If using *--sc*, the SystemC kernel is not thread safe, therefore the eval thread and main thread must be the same.

If using *--trace*, the tracing classes must be constructed and called from the main thread.

If using *--vpi*, since SystemVerilog VPI was not architected by IEEE to be multithreaded, Verilator requires all VPI calls are only made from the main thread.

4.5 GNU Make

Verilator defaults to creating GNU Make makefiles for the model. Verilator will call make automatically when the :vlopt:'–build' option is used.

If calling Verilator from a makefile, the :vlopt:'-MMD' option will create a dependency file which will allow Make to only run Verilator if input Verilog files change.

4.6 CMake

Verilator can be run using CMake, which takes care of both running Verilator and compiling the output. There is a CMake example in the examples/ directory. The following is a minimal CMakeLists.txt that would build the code listed in *Example C++ Execution*

```
project(cmake_example)
find_package(verilator HINTS $ENV{VERILATOR_ROOT})
add_executable(Vour sim_main.cpp)
verilate(Vour SOURCES our.v)
```

`find_package` will automatically find an installed copy of Verilator, or use a local build if VERILATOR_ROOT is set.

It is recommended to use CMake >= 3.12 and the Ninja generator, though other combinations should work. To build with CMake, change to the folder containing CMakeLists.txt and run:

```
mkdir build
cd build
cmake -GNinja ..
ninja
```

Or to build with your system default generator:

```
mkdir build
cd build
cmake ..
cmake --build .
```

If you're building the example you should have an executable to run:

```
./Vour
```

The package sets the CMake variables verilator_FOUND, VERILATOR_ROOT and VERILATOR_BIN to the appropriate values, and also creates a verilate() function. verilate() will automatically create custom commands to run Verilator and add the generated C++ sources to the target specified.

4.6.1 Verilate in CMake

```
verilate(target SOURCES source ... [TOP_MODULE top] [PREFIX name]
         [TRACE] [TRACE_FST] [SYSTEMC] [COVERAGE]
         [INCLUDE_DIRS dir ...] [OPT_SLOW ...] [OPT_FAST ...]
         [OPT_GLOBAL ..] [DIRECTORY dir] [THREADS num]
         [TRACE_THREADS num] [VERILATOR_ARGS ...])
```

Lowercase and ... should be replaced with arguments, the uppercase parts delimit the arguments and can be passed in any order, or left out entirely if optional.

verilate(target ...) can be called multiple times to add other Verilog modules to an executable or library target.

When generating Verilated SystemC sources, you should also include the SystemC include directories and link to the SystemC libraries.

target
: Name of a target created by add_executable or add_library.

COVERAGE
: Optional. Enables coverage if present, equivalent to "VERILATOR_ARGS –coverage"

DIRECTORY
: Optional. Set the verilator output directory. It is preferable to use the default, which will avoid collisions with other files.

INCLUDE_DIRS
: Optional. Sets directories that Verilator searches (same as -y).

OPT_SLOW
: Optional. Set compiler options for the slow path. You may want to reduce the optimization level to improve compile times with large designs.

OPT_FAST
: Optional. Set compiler options for the fast path.

OPT_GLOBAL
: Optional. Set compiler options for the common runtime library used by Verilated models.

PREFIX
: Optional. Sets the Verilator output prefix. Defaults to the name of the first source file with a "V" prepended. Must be unique in each call to verilate(), so this is necessary if you build a module multiple times with different parameters. Must be a valid C++ identifier, i.e. contains no white space and only characters A-Z, a-z, 0-9 or _.

SOURCES
: List of Verilog files to Verilate. Must have at least one file.

SYSTEMC
: Optional. Enables SystemC mode, defaults to C++ if not specified.

  When using Accellera's SystemC with CMake support, a CMake target is available that simplifies the SystemC steps. This will only work if the SystemC installation can be found by CMake. This can be configured by setting the CMAKE_PREFIX_PATH variable during CMake configuration.

  Don't forget to set the same C++ standard for the Verilated sources as the SystemC library. This can be specified using the SYSTEMC_CXX_FLAGS environment variable.

THREADS
: Optional. Generated a multi-threaded model, same as "–threads".

TRACE_THREADS
: Optional. Generated multi-threaded trace dumping, same as "–trace-threads".

TOP_MODULE
Optional. Sets the name of the top module. Defaults to the name of the first file in the SOURCES array.

TRACE
Optional. Enables VCD tracing if present, equivalent to "VERILATOR_ARGS –trace".

TRACE_FST
Optional. Enables FST tracing if present, equivalent to "VERILATOR_ARGS –trace-fst".

VERILATOR_ARGS
Optional. Extra arguments to Verilator. Do not specify *--Mdir* or *--prefix* here, use DIRECTORY or PREFIX.

4.6.2 SystemC Link in CMake

Verilator's CMake support provides a convenience function to automatically find and link to the SystemC library. It can be used as:

```
verilator_link_systemc(target)
```

where target is the name of your target.

The search paths can be configured by setting some variables:

SYSTEMC_INCLUDE
Sets the direct path to the SystemC includes.

SYSTEMC_LIBDIR
Sets the direct path to the SystemC libraries.

SYSTEMC_ROOT
Sets the installation prefix of an installed SystemC library.

SYSTEMC
Sets the installation prefix of an installed SystemC library. (Same as SYSTEMC_ROOT).

CHAPTER

FIVE

CONNECTING TO VERILATED MODELS

5.1 Structure of the Verilated Model

Verilator outputs a *prefix*.h header file which defines a class named {prefix} which represents the generated model the user is supposed to instantiate. This model class defines the interface of the Verilated model.

Verilator will additionally create a *prefix*.cpp file, together with additional .h and .cpp files for internals. See the examples directory in the kit for examples. See *Files Read/Written* for information on all the files Verilator might output.

The output of Verilator will contain a *prefix*.mk file that may be used with Make to build a *prefix*__ALL.a library with all required objects in it.

The generated model class file manages all internal state required by the model, and exposes the following interface that allows interaction with the model:

- Top level IO ports are exposed as references to the appropriate internal equivalents.
- Public top level module instances are exposed as pointers to allow access to /* verilator public */ items.
- The root of the design hierarchy (as in SystemVerilog $root) is exposed via the rootp member pointer to allow access to model internals, including /* verilator public_flat */ items.

5.1.1 Model interface changes in version 4.210

Starting from version 4.210, the model class is an interface object.

Up until Verilator version 4.204 inclusive, the generated model class was also the instance of the top level instance in the design hierarchy (what you would refer to with $root in SystemVerilog). This meant that all internal variables that were implemented by Verilator in the root scope were accessible as members of the model class itself. Note there were often many such variable due to module inlining, including /* verilator public_flat */ items.

This means that user code that accesses internal signals in the model (likely including /* verilator public_flat */ signals, as they are often inlined into the root scope) will need to be updated as follows:

- No change required for accessing top level IO signals. These are directly accessible in the model class via references.
- No change required for accessing /* verilator public */ items. These are directly accessible via sub-module pointers in the model class.
- Accessing any other internal members, including /* verilator public_flat */ items requires the following changes:

- Additionally include *prefix*___024root.h. This header defines type of the rootp pointer within the model class. Note the __024 substring is the Verilator escape sequence for the $ character, i.e.: rootp points to the Verilated SystemVerilog $root scope.
- Replace modelp->internal->member->lookup references with modelp->rootp->internal->member->lookup references, which contain one additional indirection via the rootp pointer.

5.2 Connecting to C++

In C++ output mode (*--cc*), the Verilator generated model class is a simple C++ class. The user must write a C++ wrapper and main loop for the simulation, which instantiates the model class, and link with the Verilated model. Here is a simple example:

```
#include <verilated.h>          // Defines common routines
#include <iostream>             // Need std::cout
#include "Vtop.h"               // From Verilating "top.v"

Vtop *top;                      // Instantiation of model

vluint64_t main_time = 0;       // Current simulation time
// This is a 64-bit integer to reduce wrap over issues and
// allow modulus.  This is in units of the timeprecision
// used in Verilog (or from --timescale-override)

double sc_time_stamp() {        // Called by $time in Verilog
    return main_time;           // converts to double, to match
                                // what SystemC does
}

int main(int argc, char** argv) {
    Verilated::commandArgs(argc, argv);   // Remember args

    top = new Vtop;             // Create model

    top->reset_l = 0;           // Set some inputs

    while (!Verilated::gotFinish()) {
        if (main_time > 10) {
            top->reset_l = 1;   // Deassert reset
        }
        if ((main_time % 10) == 1) {
            top->clk = 1;       // Toggle clock
        }
        if ((main_time % 10) == 6) {
            top->clk = 0;
        }
        top->eval();            // Evaluate model
        cout << top->out << endl;       // Read a output
        main_time++;            // Time passes...
    }

    top->final();               // Done simulating
    //    // (Though this example doesn't get here)
    delete top;
}
```

Note top level IO signals are read and written as members of the model. You call the `eval()` method to evaluate the model. When the simulation is complete call the `final()` method to execute any SystemVerilog final blocks, and complete any assertions. See *Wrappers and Model Evaluation Loop*.

5.3 Connecting to SystemC

In SystemC output mode (*--sc*), the Verilator generated model class is a SystemC SC_MODULE. This module will attach directly into a SystemC netlist as an instantiation.

The SC_MODULE gets the same pinout as the Verilog module, with the following type conversions: Pins of a single bit become bool. Pins 2-32 bits wide become uint32_t's. Pins 33-64 bits wide become sc_bv's or vluint64_t's depending on the *--no-pins64* option. Wider pins become sc_bv's. (Uints simulate the fastest so are used where possible.)

Model internals, including lower level sub-modules are not pure SystemC code. This is a feature, as using the SystemC pin interconnect scheme everywhere would reduce performance by an order of magnitude.

5.4 Direct Programming Interface (DPI)

Verilator supports SystemVerilog Direct Programming Interface import and export statements. Only the SystemVerilog form ("DPI-C") is supported, not the original Synopsys-only DPI.

5.4.1 DPI Example

In the SYSTEMC example above, if you wanted to import C++ functions into Verilog, put in our.v:

```
import "DPI-C" function int add (input int a, input int b);

initial begin
   $display("%x + %x = %x", 1, 2, add(1,2));
endtask
```

Then after Verilating, Verilator will create a file Vour__Dpi.h with the prototype to call this function:

```
extern int add(int a, int b);
```

From the sc_main.cpp file (or another .cpp file passed to the Verilator command line, or the link), you'd then:

```
#include "svdpi.h"
#include "Vour__Dpi.h"
int add(int a, int b) { return a+b; }
```

5.4.2 DPI System Task/Functions

Verilator extends the DPI format to allow using the same scheme to efficiently add system functions. Simply use a dollar-sign prefixed system function name for the import, but note it must be escaped.

```
export "DPI-C" function integer \$myRand;

initial $display("myRand=%d", $myRand());
```

Going the other direction, you can export Verilog tasks so they can be called from C++:

```
export "DPI-C" task publicSetBool;

task publicSetBool;
   input bit in_bool;
   var_bool = in_bool;
endtask
```

Then after Verilating, Verilator will create a file Vour__Dpi.h with the prototype to call this function:

```
extern void publicSetBool(svBit in_bool);
```

From the sc_main.cpp file, you'd then:

```
#include "Vour__Dpi.h"
publicSetBool(value);
```

Or, alternatively, call the function under the design class. This isn't DPI compatible but is easier to read and better supports multiple designs.

```
#include "Vour__Dpi.h"
Vour::publicSetBool(value);
// or top->publicSetBool(value);
```

Note that if the DPI task or function accesses any register or net within the RTL, it will require a scope to be set. This can be done using the standard functions within svdpi.h, after the module is instantiated, but before the task(s) and/or function(s) are called.

For example, if the top level module is instantiated with the name "dut" and the name references within tasks are all hierarchical (dotted) names with respect to that top level module, then the scope could be set with

```
#include "svdpi.h"
...
svSetScope(svGetScopeFromName("TOP.dut"));
```

(Remember that Verilator adds a "TOP" to the top of the module hierarchy.)

Scope can also be set from within a DPI imported C function that has been called from Verilog by querying the scope of that function. See the sections on DPI Context Functions and DPI Header Isolation below and the comments within the svdpi.h header for more information.

5.4.3 DPI Imports that access signals

If a DPI import accesses a signal through the VPI Verilator will not be able to know what variables are accessed and may schedule the code inappropriately. Ideally pass the values as inputs/outputs so the VPI is not required. Alternatively a workaround is to use a non-inlined task as a wrapper:

```
logic din;

// This DPI function will read "din"
import "DPI-C" context function void dpi_that_accesses_din();

always @(...)
   dpi_din_args(din);

task dpi_din_args(input din);
   /* verilator no_inline_task */
   dpi_that_accesses_din();
endtask
```

5.4.4 DPI Display Functions

Verilator allows writing $display like functions using this syntax:

```
import "DPI-C" function void
      \$my_display(input string formatted /*verilator sformat*/ );
```

The */*verilator&32;sformat*/* metacomment indicates that this function accepts a $display like format specifier followed by any number of arguments to satisfy the format.

5.4.5 DPI Context Functions

Verilator supports IEEE DPI Context Functions. Context imports pass the simulator context, including calling scope name, and filename and line number to the C code. For example, in Verilog:

```
import "DPI-C" context function int dpic_line();
initial $display("This is line %d, again, line %d\n", `line, dpic_line());
```

This will call C++ code which may then use the svGet* functions to read information, in this case the line number of the Verilog statement that invoked the dpic_line function:

```
int dpic_line() {
    // Get a scope:  svScope scope = svGetScope();

    const char* scopenamep = svGetNameFromScope(scope);
    assert(scopenamep);

    const char* filenamep = "";
    int lineno = 0;
    if (svGetCallerInfo(&filenamep, &lineno)) {
        printf("dpic_line called from scope %s on line %d\n",
           scopenamep, lineno);
        return lineno;
    } else {
        return 0;
```

(continues on next page)

(continued from previous page)

```
    }
}
```

See the IEEE Standard for more information.

5.4.6 DPI Header Isolation

Verilator places the IEEE standard header files such as svdpi.h into a separate include directory, vltstd (VeriLaTor STandarD). When compiling most applications $VERILATOR_ROOT/include/vltstd would be in the include path along with the normal $VERILATOR_ROOT/include. However, when compiling Verilated models into other simulators which have their own svdpi.h and similar standard files with different contents, the vltstd directory should not be included to prevent picking up incompatible definitions.

5.4.7 Public Functions

Instead of DPI exporting, there's also Verilator public functions, which are slightly faster, but less compatible.

5.5 Verification Procedural Interface (VPI)

Verilator supports a limited subset of the VPI. This subset allows inspection, examination, value change callbacks, and depositing of values to public signals only.

VPI is enabled with the Verilator *--vpi* option.

To access signals via the VPI, Verilator must be told exactly which signals are to be accessed. This is done using the Verilator public pragmas documented below.

Verilator has an important difference from an event based simulator; signal values that are changed by the VPI will not immediately propagate their values, instead the top level header file's `eval()` method must be called. Normally this would be part of the normal evaluation (i.e. the next clock edge), not as part of the value change. This makes the performance of VPI routines extremely fast compared to event based simulators, but can confuse some test-benches that expect immediate propagation.

Note the VPI by its specified implementation will always be much slower than accessing the Verilator values by direct reference (structure->module->signame), as the VPI accessors perform lookup in functions at simulation runtime requiring at best hundreds of instructions, while the direct references are evaluated by the compiler and result in only a couple of instructions.

For signal callbacks to work the main loop of the program must call `VerilatedVpi::callValueCbs()`.

5.5.1 VPI Example

In the below example, we have readme marked read-only, and writeme which if written from outside the model will have the same semantics as if it changed on the specified clock edge.

```
cat >our.v <<'EOF'
  module our (input clk);
     reg readme   /*verilator public_flat_rd*/;
     reg writeme  /*verilator public_flat_rw @(posedge clk) */;
     initial $finish;
  endmodule
EOF
```

There are many online tutorials and books on the VPI, but an example that accesses the above signal "readme" would be:

```
cat >sim_main.cpp <<'<<EOF'
  #include "Vour.h"
  #include "verilated.h"
  #include "verilated_vpi.h"  // Required to get definitions

  vluint64_t main_time = 0;   // See comments in first example
  double sc_time_stamp() { return main_time; }

  void read_and_check() {
      vpiHandle vh1 = vpi_handle_by_name((PLI_BYTE8*)"TOP.our.readme", NULL);
      if (!vh1) vl_fatal(__FILE__, __LINE__, "sim_main", "No handle found");
      const char* name = vpi_get_str(vpiName, vh1);
      printf("Module name: %s\n", name);  // Prints "readme"

      s_vpi_value v;
      v.format = vpiIntVal;
      vpi_get_value(vh1, &v);
      printf("Value of v: %d\n", v.value.integer);  // Prints "readme"
  }

  int main(int argc, char** argv, char** env) {
      Verilated::commandArgs(argc, argv);
      Vour* top = new Vour;
      Verilated::internalsDump();  // See scopes to help debug
      while (!Verilated::gotFinish()) {
          top->eval();
          VerilatedVpi::callValueCbs();  // For signal callbacks
          read_and_check();
      }
      delete top;
      return 0;
  }
EOF
```

5.6 Wrappers and Model Evaluation Loop

When using SystemC, evaluation of the Verilated model is managed by the SystemC kernel, and for the most part can be ignored. When using C++, the user must call `eval()`, or `eval_step()` and `eval_end_step()`.

1. When there is a single design instantiated at the C++ level that needs to evaluate within a given context, call `designp->eval()`.

2. When there are multiple designs instantiated at the C++ level that need to evaluate within a context, call `first_designp->eval_step()` then `->eval_step()` on all other designs. Then call `->eval_end_step()` on the first design then all other designs. If there is only a single design, you would call `eval_step()` then `eval_end_step()`; in fact `eval()` described above is just a wrapper which calls these two functions.

When `eval()` (or `eval_step()`) is called Verilator looks for changes in clock signals and evaluates related sequential always blocks, such as computing always_ff @ (posedge...) outputs. Then Verilator evaluates combinatorial logic.

Note combinatorial logic is not computed before sequential always blocks are computed (for speed reasons). Therefore it is best to set any non-clock inputs up with a separate `eval()` call before changing clocks.

Alternatively, if all always_ff statements use only the posedge of clocks, or all inputs go directly to always_ff statements, as is typical, then you can change non-clock inputs on the negative edge of the input clock, which will be faster as there will be fewer `eval()` calls.

For more information on evaluation, see `docs/internals.rst` in the distribution.

5.7 Verilated and VerilatedContext

Multiple Verilated models may be part of the same simulation context, that is share a VPI interface, sense of time, and common settings. This common simulation context information is stored in a `VerilatedContext` structure. If a `VerilatedContext` is not created prior to creating a model, a default global one is created automatically.

The `Verilated::` methods, including the `Verilated::commandArgs` call shown above, simply call VerilatedContext methods using the default global VerilatedContext. (Technically they operate on the last one used by a given thread.) If you are using multiple simulation contexts you should not use the Verilated:: methods, and instead always use VerilatedContext methods called on the appropriate VerilatedContext object.

For methods available under Verilated and VerilatedContext see `include/verilated.h` in the distribution.

CHAPTER

SIX

SIMULATING (VERILATED-MODEL RUNTIME)

This section describes items related to simulating, that using a Verilated model's executable. For the runtime arguments to a simulated model, see *Simulation Runtime Arguments*.

6.1 Benchmarking & Optimization

For best performance, run Verilator with the *-O3 --x-assign fast --x-initial fast --noassert* options. The *-O3* option will require longer time to run Verilator, and *--x-assign fast --x-initial fast* may increase the risk of reset bugs in trade for performance; see the above documentation for these options.

If using Verilated multithreaded, use `numactl` to ensure you are using non-conflicting hardware resources. See *Multithreading*. Also consider using profile-guided optimization, see *Thread Profile-Guided Optimization*.

Minor Verilog code changes can also give big wins. You should not have any UNOPTFLAT warnings from Verilator. Fixing these warnings can result in huge improvements; one user fixed their one UNOPTFLAT warning by making a simple change to a clock latch used to gate clocks and gained a 60% performance improvement.

Beyond that, the performance of a Verilated model depends mostly on your C++ compiler and size of your CPU's caches. Experience shows that large models are often limited by the size of the instruction cache, and as such reducing code size if possible can be beneficial.

The supplied $VERILATOR_ROOT/include/verilated.mk file uses the OPT, OPT_FAST, OPT_SLOW and OPT_GLOBAL variables to control optimization. You can set these when compiling the output of Verilator with Make, for example:

```
make OPT_FAST="-Os -march=native" -f Vour.mk Vour__ALL.a
```

OPT_FAST specifies optimization options for those parts of the model that are on the fast path. This is mostly code that is executed every cycle. OPT_SLOW applies to slow-path code, which executes rarely, often only once at the beginning or end of simulation. Note that OPT_SLOW is ignored if VM_PARALLEL_BUILDS is not 1, in which case all generated code will be compiled in a single compilation unit using OPT_FAST. See also the Verilator *--output-split* option. The OPT_GLOBAL variable applies to common code in the runtime library used by Verilated models (shipped in $VERILATOR_ROOT/include). Additional C++ files passed on the verilator command line use OPT_FAST. The OPT variable applies to all compilation units in addition to the specific "OPT" variables described above.

You can also use the *-CFLAGS* and/or *-LDFLAGS* options on the verilator command line to pass arguments directly to the compiler or linker.

The default values of the "OPT" variables are chosen to yield good simulation speed with reasonable C++ compilation times. To this end, OPT_FAST is set to "-Os" by default. Higher optimization such as "-O2" or "-O3" may help (though often they provide only a very small performance benefit), but compile times may be excessively large even with medium sized designs. Compilation times can be improved at the expense of simulation speed by reducing optimization, for example with OPT_FAST="-O0". Often good simulation speed can be achieved with OPT_FAST="-O1

-fstrict-aliasing" but with improved compilation times. Files controlled by OPT_SLOW have little effect on performance and therefore OPT_SLOW is empty by default (equivalent to "-O0") for improved compilation speed. In common use-cases there should be little benefit in changing OPT_SLOW. OPT_GLOBAL is set to "-Os" by default and there should rarely be a need to change it. As the runtime library is small in comparison to a lot of Verilated models, disabling optimization on the runtime library should not have a serious effect on overall compilation time, but may have detrimental effect on simulation speed, especially with tracing. In addition to the above, for best results use OPT="-march=native", the latest Clang compiler (about 10% faster than GCC), and link statically.

Generally the answer to which optimization level gives the best user experience depends on the use case and some experimentation can pay dividends. For a speedy debug cycle during development, especially on large designs where C++ compilation speed can dominate, consider using lower optimization to get to an executable faster. For throughput oriented use cases, for example regressions, it is usually worth spending extra compilation time to reduce total CPU time.

If you will be running many simulations on a single model, you can investigate profile guided optimization. See *Compiler Profile-Guided Optimization*.

Modern compilers also support link-time optimization (LTO), which can help especially if you link in DPI code. To enable LTO on GCC, pass "-flto" in both compilation and link. Note LTO may cause excessive compile times on large designs.

Unfortunately, using the optimizer with SystemC files can result in compilation taking several minutes. (The SystemC libraries have many little inlined functions that drive the compiler nuts.)

If you are using your own makefiles, you may want to compile the Verilated code with `--MAKEFLAGS -DVL_INLINE_OPT=inline`. This will inline functions, however this requires that all cpp files be compiled in a single compiler run.

You may uncover further tuning possibilities by profiling the Verilog code. See *Code Profiling*.

When done optimizing, please let the author know the results. We like to keep tabs on how Verilator compares, and may be able to suggest additional improvements.

6.2 Coverage Analysis

Verilator supports adding code to the Verilated model to support SystemVerilog code coverage. With `--coverage`, Verilator enables all forms of coverage:

- *Functional Coverage*
- *Line Coverage*
- *Toggle Coverage*

When a model with coverage is executed, it will create a coverage file for collection and later analysis, see *Coverage Collection*.

6.2.1 Functional Coverage

With `--coverage` or `--coverage-user`, Verilator will translate functional coverage points which the user has inserted manually into the SystemVerilog design, into the Verilated model.

Currently, all functional coverage points are specified using SystemVerilog assertion syntax which must be separately enabled with `--assert`.

For example, the following SystemVerilog statement will add a coverage point, under the coverage name "DefaultClock":

```
DefaultClock: cover property (@(posedge clk) cyc==3);
```

6.2.2 Line Coverage

With *--coverage* or *--coverage-line*, Verilator will automatically add coverage analysis at each code flow change point (e.g. at branches). At each such branch a unique counter is incremented. At the end of a test, the counters along with the filename and line number corresponding to each counter are written into the coverage file.

Verilator automatically disables coverage of branches that have a $stop in them, as it is assumed $stop branches contain an error check that should not occur. A */*verilator&32;coverage_block_off*/* metacomment will perform a similar function on any code in that block or below, or */*verilator&32;coverage_off*/* and */*verilator&32;coverage_on*/* will disable and enable coverage respectively around a block of code.

Verilator may over-count combinatorial (non-clocked) blocks when those blocks receive signals which have had the UNOPTFLAT warning disabled; for most accurate results do not disable this warning when using coverage.

6.2.3 Toggle Coverage

With *--coverage* or *--coverage-toggle*, Verilator will automatically add toggle coverage analysis into the Verilated model.

Every bit of every signal in a module has a counter inserted. The counter will increment on every edge change of the corresponding bit.

Signals that are part of tasks or begin/end blocks are considered local variables and are not covered. Signals that begin with underscores (see *--coverage-underscore*), are integers, or are very wide (>256 bits total storage across all dimensions, see *--coverage-max-width*) are also not covered.

Hierarchy is compressed, such that if a module is instantiated multiple times, coverage will be summed for that bit across **all** instantiations of that module with the same parameter set. A module instantiated with different parameter values is considered a different module, and will get counted separately.

Verilator makes a minimally-intelligent decision about what clock domain the signal goes to, and only looks for edges in that clock domain. This means that edges may be ignored if it is known that the edge could never be seen by the receiving logic. This algorithm may improve in the future. The net result is coverage may be lower than what would be seen by looking at traces, but the coverage is a more accurate representation of the quality of stimulus into the design.

There may be edges counted near time zero while the model stabilizes. It's a good practice to zero all coverage just before releasing reset to prevent counting such behavior.

A */*verilator&32;coverage_off*/ /*verilator&32;coverage_on*/* metacomment pair can be used around signals that do not need toggle analysis, such as RAMs and register files.

6.2.4 Coverage Collection

When any coverage flag was used to Verilate, Verilator will add appropriate coverage point insertions into the model and collect the coverage data.

To get the coverage data from the model, in the user wrapper code, typically at the end once a test passes, call `Verilated::threadContextp()->coveragep()->write` with an argument of the filename for the coverage data file to write coverage data to (typically "logs/coverage.dat").

Run each of your tests in different directories, potentially in parallel. Each test will create a `logs/coverage.dat` file.

After running all of the tests, execute the **verilator_coverage** command, passing arguments pointing to the filenames of all of the individual coverage files. **verilator_coverage** will reads the `logs/coverage.dat` file(s), and create an annotated source code listing showing code coverage details.

verilator_coverage may also be used for test grading, that is computing which tests are important to fully cover the design.

For an example, see the `examples/make_tracing_c/logs` directory. Grep for lines starting with '%' to see what lines Verilator believes need more coverage.

Additional options of **verilator_coverage** allow for merging of coverage data files or other transformations.

Info files can be written by verilator_coverage for import to **lcov**. This enables use of **genhtml** for HTML reports and importing reports to sites such as https://codecov.io.

6.3 Code Profiling

The Verilated model may be code-profiled using GCC or Clang's C++ profiling mechanism. Verilator provides additional flags to help map the resulting C++ profiling results back to the original Verilog code responsible for the profiled C++ code functions.

To use profiling:

1. Use Verilator's *--prof-cfuncs*.
2. Build and run the simulation model.
3. The model will create gmon.out.
4. Run **gprof** to see where in the C++ code the time is spent.
5. Run the gprof output through the **verilator_profcfunc** program and it will tell you what Verilog line numbers on which most of the time is being spent.

6.4 Thread Profiling

When using multithreaded mode (*--threads*), it is useful to see statistics and visualize how well the multiple CPUs are being utilized.

With the *--prof-threads* option, Verilator will:

- Add code to the Verilated model to record the start and end time of each macro-task across a number of calls to eval. (What is a macro-task? See the Verilator internals document (`docs/internals.rst` in the distribution.)
- Add code to save profiling data in non-human-friendly form to the file specified with *+verilator+prof+threads+file+<filename>*.
- Add code to save profiling data for thread profile-guided optimization. See *Thread Profile-Guided Optimization*.

The **verilator_gantt** program may then be run to transform the saved profiling file into a nicer visual format and produce some related statistics.

For more information see **verilator_gantt**.

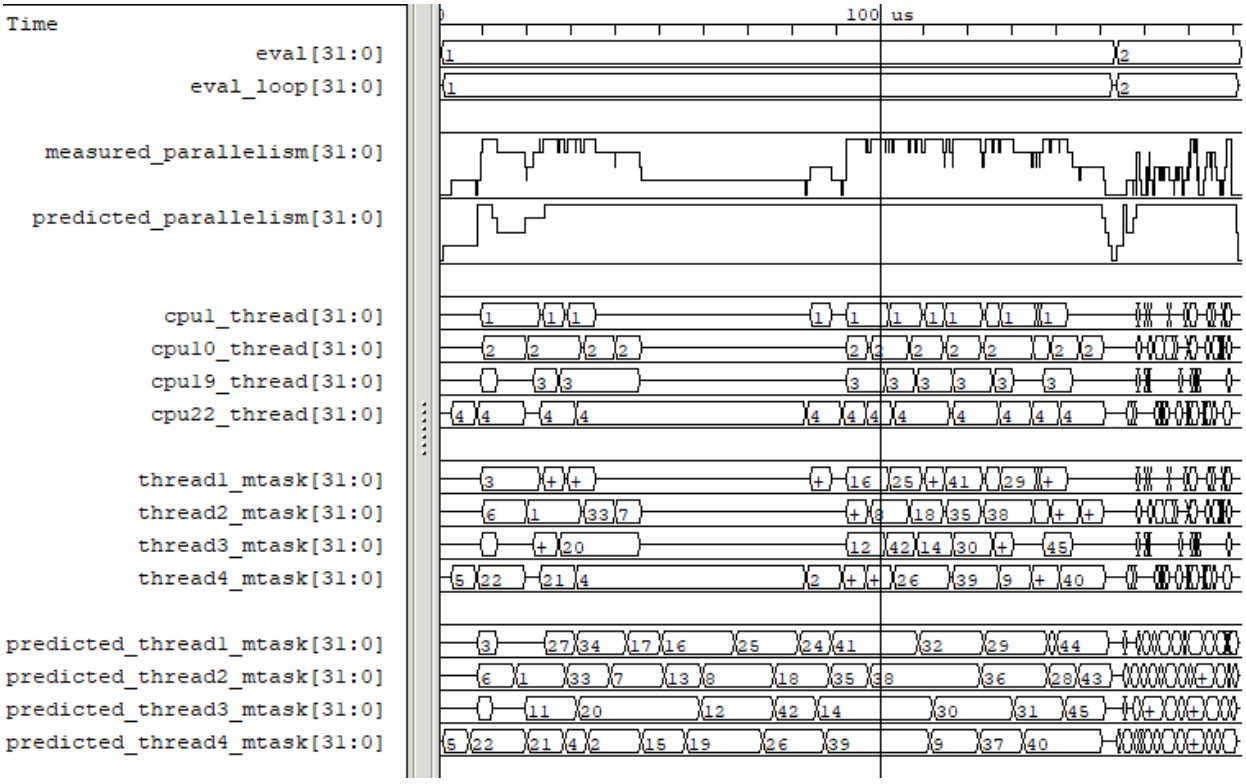

Fig. 6.1: Example verilator_gantt output, as viewed with GTKWave.
The measured_parallelism shows the number of CPUs being used at a given moment.
The cpu_thread section shows which thread is executing on each of the physical CPUs.
The thread_mtask section shows which macro-task is running on a given thread.

6.5 Profiling ccache efficiency

The Verilator generated Makefile provides support for basic profiling of ccache behavior during the build. This can be used to track down files that might be unnecessarily rebuilt, though as of today even small code changes will usually require rebuilding a large number of files. Improving ccache efficiency during the edit/compile/test loop is an active area of development.

To get a basic report of how well ccache is doing, add the *ccache-report* target when invoking the generated Makefile:

```
make -C obj_dir -f Vout.mk Vout ccache-report
```

This will print a report based on all executions of ccache during this invocation of Make. The report is also written to a file, in this example *obj_dir/Vout__cache_report.txt*.

To use the *ccache-report* target, at least one other explicit build target must be specified, and OBJCACHE must be set to 'ccache'.

This feature is currently experimental and might change in subsequent releases.

6.6 Save/Restore

The intermediate state of a Verilated model may be saved, so that it may later be restored.

To enable this feature, use *--savable*. There are limitations in what language features are supported along with *--savable*; if you attempt to use an unsupported feature Verilator will throw an error.

To use save/restore, the user wrapper code must create a VerilatedSerialize or VerilatedDeserialze object then calling the << or >> operators on the generated model and any other data the process needs saved/restored. These functions are not thread safe, and are typically called only by a main thread.

For example:

```
void save_model(const char* filenamep) {
    VerilatedSave os;
    os.open(filenamep);
    os << main_time;  // user code must save the timestamp, etc
    os << *topp;
}
void restore_model(const char* filenamep) {
    VerilatedRestore os;
    os.open(filenamep);
    os >> main_time;
    os >> *topp;
}
```

6.7 Profile-Guided Optimization

Profile-guided optimization is the technique where profiling data is collected by running your simulation executable, then this information is used to guide the next Verilation or compilation.

There are two forms of profile-guided optimizations. Unfortunately for best results they must each be performed from the highest level code to the lowest, which means performing them separately and in this order:

- *Thread Profile-Guided Optimization*

- *Compiler Profile-Guided Optimization*

Other forms of PGO may be supported in the future, such as clock and reset toggle rate PGO, branch prediction PGO, statement execution time PGO, or others as they prove beneficial.

6.7.1 Thread Profile-Guided Optimization

Verilator supports thread profile-guided optimization (Thread PGO) to improve multithreaded performance.

When using multithreading, Verilator computes how long macro tasks take and tries to balance those across threads. (What is a macro-task? See the Verilator internals document (`docs/internals.rst` in the distribution.) If the estimations are incorrect, the threads will not be balanced, leading to decreased performance. Thread PGO allows collecting profiling data to replace the estimates and better optimize these decisions.

To use Thread PGO, Verilate the model with the *`--prof-threads`* option.

Run the model executable. When the executable exits, it will create a profile.vlt file.

Rerun Verilator, optionally omitting the *`--prof-threads`* option, and adding the profile.vlt generated earlier to the command line.

Note there is no Verilator equivalent to GCC's –fprofile-use. Verilator's profile data file (profile.vlt) can be placed on the verilator command line directly without any prefix.

If results from multiple simulations are to be used in generating the optimization, multiple simulation's profile.vlt may be concatenated externally, or each of the files may be fed as separate command line options into Verilator. Verilator will simply sum the profile results, so a longer running test will have proportionally more weight for optimization than a shorter running test.

If you provide any profile feedback data to Verilator, and it cannot use it, it will issue the *`PROFOUTOFDATE`* warning that threads were scheduled using estimated costs. This usually indicates that the profile data was generated from different Verilog source code than Verilator is currently running against. Therefore, repeat the data collection phase to create new profiling data, then rerun Verilator with the same input source files and that new profiling data.

6.7.2 Compiler Profile-Guided Optimization

GCC and Clang support compiler profile-guided optimization (PGO). This optimizes any C/C++ program including Verilated code. Using compiler PGO typically yields improvements of 5-15% on both single-threaded and multi-threaded models.

To use compiler PGO with GCC or Clang, please see the appropriate compiler documentation. The process in GCC 10 was as follows:

1. Compile the Verilated model with the compiler's "-fprofile-generate" flag:

   ```
   verilator [whatever_flags] --make \
       -CFLAGS -fprofile-generate -LDFLAGS -fprofile-generate
   ```

   or, if calling make yourself, add -fprofile-generate appropriately to your Makefile.

2. Run your simulation. This will create *.gcda file(s) in the same directory as the source files.

3. Recompile the model with -fprofile-use. The compiler will read the *.gcda file(s).

   For GCC:

   ```
   verilator [whatever_flags] --build \
       -CFLAGS "-fprofile-use -fprofile-correction"
   ```

For Clang:

```
llvm-profdata merge -output default.profdata *.profraw
verilator [whatever_flags] --build \
    -CFLAGS "-fprofile-use -fprofile-correction"
```

or, if calling make yourself, add these CFLAGS switches appropriately to your Makefile.

Clang and GCC also support -fauto-profile which uses sample-based feedback-directed optimization. See the appropriate compiler documentation.

CHAPTER

SEVEN

CONTRIBUTING AND REPORTING BUGS

7.1 Announcements

To get notified of new releases and other important announcements, go to Verilator announcement repository and follow the instructions there.

7.2 Reporting Bugs

First, check the *Language Limitations* section.

Next, try the *--debug* option. This will enable additional internal assertions, and may help identify the problem.

Finally, reduce your code to the smallest possible routine that exhibits the bug. Even better, create a test in the `test_regress/t` directory, as follows:

```
cd test_regress
cp -p t/t_EXAMPLE.pl t/t_BUG.pl
cp -p t/t_EXAMPLE.v t/t_BUG.v
```

There are many hints on how to write a good test in the `test_regress/driver.pl` documentation which can be seen by running:

```
cd $VERILATOR_ROOT  # Need the original distribution kit
test_regress/driver.pl --help
```

Edit `t/t_BUG.pl` to suit your example; you can do anything you want in the Verilog code there; just make sure it retains the single clk input and no outputs. Now, the following should fail:

```
cd $VERILATOR_ROOT  # Need the original distribution kit
cd test_regress
t/t_BUG.pl  # Run on Verilator
t/t_BUG.pl --debug # Run on Verilator, passing --debug to Verilator
t/t_BUG.pl --vcs  # Run on VCS simulator
t/t_BUG.pl --nc|--iv|--ghdl  # Likewise on other simulators
```

The test driver accepts a number of options, many of which mirror the main Verilator options. For example the previous test could have been run with debugging enabled. The full set of test options can be seen by running **`driver.pl --help`** as shown above.

Finally, report the bug at Verilator Issues. The bug will become publicly visible; if this is unacceptable, mail the bug report to `wsnyder@wsnyder.org`.

7.3 Contributing to Verilator

Thanks for using Verilator! We welcome your contributions in whatever form.

This contributing document contains some suggestions that may make contributions flow more efficiently.

7.3.1 Did you find a bug?

- Please **Ensure the bug was not already reported** by searching Verilator Issues.
- If you're unable to find an open issue addressing the problem, open a new Verilator issue.
  - Be sure to include a **code sample** or an **executable test case** demonstrating the bug and expected behavior that is not occurring.
  - The ideal example works against other simulators, and is in the test_regress/t test format, as described in docs/internals.

7.3.2 Did you write a patch that fixes a bug?

- Please Open a new issue.
- You may attach a patch to the issue, or (preferred) may request a GitHub pull request.
  - Verilator uses GitHub Actions to provide continuous integration. You may want to enable Actions on your GitHub branch to ensure your changes keep the tests passing. See docs/internals.
- Your source-code contributions must be certified as open source, under the Developer Certificate of Origin. On your first contribution, you must either:
  - Have your patch include the addition of your name to docs/CONTRIBUTORS (preferred).
  - Use "git -s" as part of your commit. This adds a "signed-of-by" attribute which will certify your contribution as described in the Signed-of-By convention.
  - Email, or post in an issue a statement that you certify your contributions.
  - In any of these cases your name will be added to docs/CONTRIBUTORS and you are agreeing all future contributions are also certified.
  - We occasionally accept contributions where people do not want their name published. Please email us; you must still privately certify your contribution.
- Your test contributions are generally considered released into the Creative Commons Public Domain (CC0), unless you request otherwise or put a GNU/Artistic license on your file.
- Most important is we get your patch. If you'd like to clean up indentation and related issues ahead of our feedback, that is appreciated; please see the coding conventions in docs/internals.

7.3.3 Do you have questions?

- Please see FAQ section and rest of the Verilator manual, or Verilator manual (PDF).
- Ask any question in the Verilator forum.

7.3.4 Code of Conduct

- Our contributors and participants pledge to make participation in our project and our community a positive experience for everyone. We follow the Contributor Covenant version 1.4.

Thanks!

CHAPTER

EIGHT

FAQ/FREQUENTLY ASKED QUESTIONS

8.1 Questions

8.1.1 Can I contribute?

Please contribute! Just submit a pull request, or raise an issue to discuss if looking for something to help on. For more information see our contributor agreement.

8.1.2 How widely is Verilator used?

Verilator is used by many of the largest silicon design companies, large organizations such as CERN, and even by college student projects.

Verilator is one of the "big 4" simulators, meaning one of the 4 main SystemVerilog simulators available, namely the closed-source products Synopsys VCS (tm), Mentor Questa/ModelSim (tm), Cadence Xcelium/Incisive/NC-Verilog/NC-Sim (tm), and the open-source Verilator. The three closed-source offerings are often collectively called the "big 3" simulators.

8.1.3 Does Verilator run under Windows?

Yes, ideally run Ubuntu under Windows Subsystem for Linux (WSL2). Alternatively use Cygwin, though this tends to be slower and is not regurally tested. Verilated output also compiles under Microsoft Visual C++, but this is also not regurally tested.

8.1.4 Can you provide binaries?

You can install Verilator via the system package manager (apt, yum, etc.) on many Linux distributions, including Debian, Ubuntu, SuSE, RedHat, and others. These packages are provided by the Linux distributions and generally will lag the version of the mainline Verilator repository. If no binary package is available for your distribution, how about you set one up?

8.1.5 How can it be faster than (name-a-big-3-closed-source-simulator)?

Generally, the implied part of the question is "... with all of the manpower they can put into developing it."

Most simulators have to be compliant with the complete IEEE 1364 (Verilog) and IEEE 1800 (SystemVerilog) standards, meaning they have to be event driven. This prevents them from being able to reorder blocks and make netlist-style optimizations, which are where most of the gains come from.

You should not be scared by non-compliance. Your synthesis tool isn't compliant with the whole standard to start with, so your simulator need not be either. Verilator is closer to the synthesis interpretation, so this is a good thing for getting working silicon.

8.1.6 Will Verilator output remain under my own license/copyright?

Yes, it's just like using GCC on your programs; this is why Verilator uses the "GNU **Lesser** Public License Version 3" instead of the more typical "GNU Public License". See the licenses for details, but in brief, if you change Verilator itself or the header files Verilator includes, you must make the source code available under the GNU Lesser Public License. However, Verilator output (the Verilated code) only "include"s the licensed files, and so you are **not** required to open-source release any output from Verilator.

You also have the option of using the Perl Artistic License, which again does not require you to release your Verilog or generated code, and also allows you to modify Verilator for internal use without distributing the modified version. But please contribute back to the community!

One limit is that you cannot under either license release a closed-source Verilog simulation product incorporating Verilator. That is you can have a commercial product, but must make the source code available.

As is standard with Open Source, contributions back to Verilator will be placed under the Verilator copyright and LGPL/Artistic license. Small test cases will be released into the public domain so they can be used anywhere, and large tests under the LGPL/Artistic, unless requested otherwise.

8.1.7 Why is running Verilator (to create a model) so slow?

Verilator may require more memory than the resulting simulator will require, as Verilator internally creates all of the state of the resulting generated simulator in order to optimize it. If it takes more than a few minutes or so (and you're not using `--debug` since debug mode is disk bound), see if your machine is paging; most likely you need to run it on a machine with more memory. Very large designs are known to have topped 64 GB resident set size. Alternatively, see *Hierarchical Verilation*.

8.1.8 How do I generate waveforms (traces) in C++?

See also the next question for tracing in SystemC mode.

A. Pass the `--trace` option to Verilator, and in your top level C code, call `Verilated::traceEverOn(true)`. Then you may use `$dumpfile` and `$dumpvars` to enable traces, same as with any Verilog simulator. See `examples/make_tracing_c` in the distribution.

B. Or, for finer-grained control, or C++ files with multiple Verilated modules you may also create the trace purely from C++. Create a VerilatedVcdC object, and in your main loop right after `eval()` call `trace_object->dump(contextp->time())` every time step, and finally call `trace_object->close()`.

```
#include "verilated_vcd_c.h"
...
int main(int argc, char** argv, char** env) {
    const std::unique_ptr<VerilatedContext> contextp{new VerilatedContext};
    ...
    Verilated::traceEverOn(true);
    VerilatedVcdC* tfp = new VerilatedVcdC;
    topp->trace(tfp, 99);  // Trace 99 levels of hierarchy
    tfp->open("obj_dir/t_trace_ena_cc/simx.vcd");
    ...
    while (contextp->time() < sim_time && !contextp->gotFinish()) {
        contextp->timeInc(1);
        topp->eval();
        tfp->dump(contextp->time());
    }
    tfp->close();
}
```

You also need to compile `verilated_vcd_c.cpp` and add it to your link, preferably by adding the dependencies in your Makefile's `$(VK_GLOBAL_OBJS)` link rule. This is done for you if using the Verilator *--exe* option.

you can call `trace_object->trace()` on multiple Verilated objects with the same trace file if you want all data to land in the same output file.

8.1.9 How do I generate waveforms (traces) in SystemC?

A. Pass the *--trace* option to Verilator, and in your top level `sc_main()`, call `Verilated::traceEverOn(true)`. Then you may use `$dumpfile` and code:*$dumpvars* to enable traces, same as with any Verilog simulator, see the non-SystemC example in `examples/make_tracing_c`. This will trace only the module containing the `$dumpvar`.

B. Or, you may create a trace purely from SystemC, which may trace all Verilated designs in the SystemC model. Create a VerilatedVcdSc object as you would create a normal SystemC trace file. For an example, see the call to `VerilatedVcdSc` in the `examples/make_tracing_sc/sc_main.cpp` file of the distribution, and below.

C. Alternatively you may use the C++ trace mechanism described in the previous question, note the timescale and timeprecision will be inherited from your SystemC settings.

```
#include "verilated_vcd_sc.h"
...
int main(int argc, char** argv, char** env) {
    ...
    Verilated::traceEverOn(true);
    VerilatedVcdSc* tfp = new VerilatedVcdSc;
    topp->trace(tfp, 99);  // Trace 99 levels of hierarchy
    tfp->open("obj_dir/t_trace_ena_cc/simx.vcd");
    ...
    sc_start(1);
    ...
    tfp->close();
}
```

You also need to compile `verilated_vcd_sc.cpp` and `verilated_vcd_c.cpp` and add them to your link, preferably by adding the dependencies in your Makefile's `$(VK_GLOBAL_OBJS)` link rule. This is done for you if using the Verilator *--exe* option.

You can call `->trace()` on multiple Verilated objects with the same trace file if you want all data to land in the same output file.

When using SystemC 2.3, the SystemC library must have been built with the experimental simulation phase callback based tracing disabled. This is disabled by default when building SystemC with its configure based build system, but when building SystemC with CMake, you must pass `-DENABLE_PHASE_CALLBACKS_TRACING=OFF` to disable this feature.

8.1.10 How do I generate FST waveforms (traces) in C++ or SystemC?

FST is a trace file format developed by GTKWave. Verilator provides basic FST support. To dump traces in FST format, add the *--trace-fst* option to Verilator and either A. use `$dumpfile & $dumpvars` in Verilog as described in the VCD example above,

Or, in C++ change the include described in the VCD example above:

```
#include "verilated_fst_c.h"
VerilatedFstC* tfp = new VerilatedFstC;
```

Or, in SystemC change the include described in the VCD example above:

```
#include "verilated_fst_sc.h"
VerilatedFstC* tfp = new VerilatedFstSc;
```

Note that currently supporting both FST and VCD in a single simulation is impossible, but such requirement should be rare. You can however ifdef around the trace format in your C++ main loop, and select VCD or FST at build time, should you require.

8.1.11 How do I view waveforms (aka dumps or traces)?

Verilator creates standard VCD (Value Change Dump) and FST files. VCD files are viewable with the open source GTKWave (recommended) or Dinotrace (legacy) programs, or any of the many closed-source offerings; FST is supported only by GTKWave.

8.1.12 How do I speed up writing large waveform (trace) files?

A. Instead of calling `VerilatedVcdC->open` or `$dumpvars` at the beginning of time, delay calling it until the time stamp where you want tracing to begin.

B. Add the */*verilator&32;tracing_off*/* metacomment to any very low level modules you never want to trace (such as perhaps library cells).

C. Use the *--trace-depth* option to limit the depth of tracing, for example *--trace-depth 1* to see only the top level signals.

D. You can also consider using FST tracing instead of VCD. FST dumps are a fraction of the size of the equivalent VCD. FST tracing can be slower than VCD tracing, but it might be the only option if the VCD file size is prohibitively large.

E. Be sure you write your trace files to a local solid-state drive, instead of to a network drive. Network drives are generally far slower.

8.1.13 Where is the translate_off command? (How do I ignore a construct?)

Translate on/off pragmas are generally a bad idea, as it's easy to have mismatched pairs, and you can't see what another tool sees by just preprocessing the code. Instead, use the preprocessor; Verilator defines the `\`VERILATOR` define for you, so just wrap the code in an ifndef region:

```
`ifndef VERILATOR
   Something_Verilator_Dislikes;
`endif
```

Most synthesis tools similarly define SYNTHESIS for you.

8.1.14 Why do I get "unexpected 'do'" or "unexpected 'bit'" errors?

The words `do`, `bit`, `ref`, `return`, and others are reserved keywords in SystemVerilog. Older Verilog code might use these as identifiers. You should change your code to not use them to ensure it works with newer tools. Alternatively, surround them by the Verilog 2005/SystemVerilog begin_keywords pragma to indicate Verilog 2001 code.

```
`begin_keywords "1364-2001"
   integer bit; initial bit = 1;
`end_keywords
```

If you want the whole design to be parsed as Verilog 2001, see the *--default-language* option.

8.1.15 How do I prevent my assertions from firing during reset?

Call `Verilated::assertOn(false)` before you first call the model, then turn it back on after reset. It defaults to true. When false, all assertions controlled by *--assert* are disabled.

8.1.16 Why do I get "undefined reference to sc_time_stamp()?

In Verilator 4.200 and later, using the timeInc function is recommended instead. See the *Connecting to C++* examples. Some linkers (MSVC++) still require `sc_time_stamp()` to be defined, either define this with `double sc_time_stamp() { return 0; }` or compile the Verilated code with `-CFLAGS -DVL_TIME_CONTEXT`.

Prior to Verilator 4.200, the `sc_time_stamp()` function needs to be defined in C++ (non SystemC) to return the current simulation time.

8.1.17 Why do I get "undefined reference to `VL_RAND_RESET_I' or `Verilated::...'"?

You need to link your compiled Verilated code against the `verilated.cpp` file found in the include directory of the Verilator kit. This is one target in the `$(VK_GLOBAL_OBJS)` make variable, which should be part of your Makefile's link rule. If you use *--exe*, this is done for you.

8.1.18 Is the PLI supported?

Only somewhat. More specifically, the common PLI-ish calls $display, $finish, $stop, $time, $write are converted to C++ equivalents. You can also use the "import DPI" SystemVerilog feature to call C code (see the chapter above). There is also limited VPI access to public signals.

If you want something more complex, since Verilator emits standard C++ code, you can simply write your own C++ routines that can access and modify signal values without needing any PLI interface code, and call it with $c("{any_c++_statement}").

See the *Connecting to Verilated Models* section.

8.1.19 How do I make a Verilog module that contain a C++ object?

You need to add the object to the structure that Verilator creates, then use $c to call a method inside your object. The `test_regress/t/t_extend_class` files in the distribution show an example of how to do this.

8.1.20 How do I get faster build times?

- When running make, pass the make variable VM_PARALLEL_BUILDS=1 so that builds occur in parallel. Note this is now set by default if an output file was large enough to be split due to the `--output-split` option.
- Verilator emits any infrequently executed "cold" routines into separate __Slow.cpp files. This can accelerate compilation as optimization can be disabled on these routines. See the OPT_FAST and OPT_SLOW make variables and *Benchmarking & Optimization*.
- Use a recent compiler. Newer compilers tend to be faster.
- Compile in parallel on many machines and use caching; see the web for the ccache, distcc and icecream packages. ccache will skip GCC runs between identical source builds, even across different users. If ccache was installed when Verilator was built it is used, or see OBJCACHE environment variable to override this. Also see the `--output-split` option and :ref: *Profiling ccache efficiency*
- To reduce the compile time of classes that use a Verilated module (e.g. a top CPP file) you may wish to add a `/*verilator&32;no_inline_module*/` metacomment to your top level module. This will decrease the amount of code in the model's Verilated class, improving compile times of any instantiating top level C++ code, at a relatively small cost of execution performance.
- Use *Hierarchical Verilation*.

8.1.21 Why do so many files need to recompile when I add a signal?

Adding a new signal requires the symbol table to be recompiled. Verilator uses one large symbol table, as that results in 2-3 less assembly instructions for each signal access. This makes the execution time 10-15% faster, but can result in more compilations when something changes.

8.1.22 How do I access Verilog functions/tasks in C?

Use the SystemVerilog Direct Programming Interface. You write a Verilog function or task with input/outputs that match what you want to call in with C. Then mark that function as a DPI export function. See the DPI chapter in the IEEE Standard.

8.1.23 How do I access C++ functions/tasks in Verilog?

Use the SystemVerilog Direct Programming Interface. You write a Verilog function or task with input/outputs that match what you want to call in with C. Then mark that function as a DPI import function. See the DPI chapter in the IEEE Standard.

8.1.24 How do I access signals in C?

The best thing to do is to make a SystemVerilog "export DPI" task or function that accesses that signal, as described in the DPI chapter in the manual and DPI tutorials on the web. This will allow Verilator to better optimize the model and should be portable across simulators.

If you really want raw access to the signals, declare the signals you will be accessing with a */*verilator&32; public*/* metacomment before the closing semicolon. Then scope into the C++ class to read the value of the signal, as you would any other member variable.

Signals are the smallest of 8-bit unsigned chars (equivalent to uint8_t), 16-bit unsigned shorts (uint16_t), 32-bit unsigned longs (uint32_t), or 64-bit unsigned long longs (uint64_t) that fits the width of the signal. Generally, you can use just uint32_t's for 1 to 32 bits, or vluint64_t for 1 to 64 bits, and the compiler will properly up-convert smaller entities. Note even signed ports are declared as unsigned; you must sign extend yourself to the appropriate signal width.

Signals wider than 64 bits are stored as an array of 32-bit uint32_t's. Thus to read bits 31:0, access signal[0], and for bits 63:32, access signal[1]. Unused bits (for example bit numbers 65-96 of a 65-bit vector) will always be zero. If you change the value you must make sure to pack zeros in the unused bits or core-dumps may result, because Verilator strips array bound checks where it believes them to be unnecessary to improve performance.

In the SYSTEMC example above, if you had in our.v:

```
input clk /*verilator public*/;
// Note the placement of the semicolon above
```

From the sc_main.cpp file, you'd then:

```
#include "Vour.h"
#include "Vour_our.h"
cout << "clock is " << top->our->clk << endl;
```

In this example, clk is a bool you can read or set as any other variable. The value of normal signals may be set, though clocks shouldn't be changed by your code or you'll get strange results.

8.1.25 Should a module be in Verilog or SystemC?

Sometimes there is a block that just interconnects instances, and have a choice as to if you write it in Verilog or SystemC. Everything else being equal, best performance is when Verilator sees all of the design. So, look at the hierarchy of your design, labeling instances as to if they are SystemC or Verilog. Then:

- A module with only SystemC instances below must be SystemC.
- A module with a mix of Verilog and SystemC instances below must be SystemC. (As Verilator cannot connect to lower-level SystemC instances.)
- A module with only Verilog instances below can be either, but for best performance should be Verilog. (The exception is if you have a design that is instantiated many times; in this case Verilating one of the lower modules and instantiating that Verilated instances multiple times into a SystemC module *may* be faster.)

CHAPTER

NINE

INPUT LANGUAGES

This section describes the languages Verilator takes as input. See also *Configuration Files*.

9.1 Language Standard Support

9.1.1 Verilog 2001 (IEEE 1364-2001) Support

Verilator supports most Verilog 2001 language features. This includes signed numbers, "always @*", generate statements, multidimensional arrays, localparam, and C-style declarations inside port lists.

9.1.2 Verilog 2005 (IEEE 1364-2005) Support

Verilator supports most Verilog 2005 language features. This includes the `begin_keywords and `end_keywords compiler directives, $clog2, and the uwire keyword.

9.1.3 SystemVerilog 2005 (IEEE 1800-2005) Support

Verilator supports ==? and !=? operators, ++ and – in some contexts, $bits, $countbits, $countones, $error, $fatal, $info, $isunknown, $onehot, $onehot0, $unit, $warning, always_comb, always_ff, always_latch, bit, byte, chandle, const, do-while, enum, export, final, import, int, interface, logic, longint, modport, package, program, shortint, struct, time, typedef, union, var, void, priority case/if, and unique case/if.

It also supports .name and .* interconnection.

Verilator partially supports concurrent assert and cover statements; see the enclosed coverage tests for the syntax which is allowed.

Verilator has limited support for class and related object-oriented constructs.

9.1.4 SystemVerilog 2012 (IEEE 1800-2012) Support

Verilator implements a full SystemVerilog-compliant preprocessor, including function call-like preprocessor defines, default define arguments, `__FILE__, `__LINE__ and `undefineall.

9.1.5 SystemVerilog 2017 (IEEE 1800-2017) Support

Verilator supports the 2017 "for" loop constructs, and several minor cleanups IEEE made in 1800-2017.

9.1.6 Verilog AMS Support

Verilator implements a very small subset of Verilog AMS (Verilog Analog and Mixed-Signal Extensions) with the subset corresponding to those VMS keywords with near equivalents in the Verilog 2005 or SystemVerilog 2009 languages.

AMS parsing is enabled with *--language VAMS* or *--language 1800+VAMS*.

At present Verilator implements ceil, exp, floor, ln, log, pow, sqrt, string, and wreal.

9.1.7 Synthesis Directive Assertion Support

With the *--assert* option, Verilator reads any `//synopsys full_case` or `//synopsys parallel_case` directives. The same applies to any `//ambit synthesis`, `//cadence` or `//pragma` directives of the same form.

When these synthesis directives are discovered, Verilator will either formally prove the directive to be true, or failing that, will insert the appropriate code to detect failing cases at simulation runtime and print an "Assertion failed" error message.

Verilator likewise also asserts any "unique" or "priority" SystemVerilog keywords on case statement, as well as "unique" on if statements. However, "priority if" is currently simply ignored.

9.2 Language Limitations

This section describes the language limitations of Verilator. Many of these restrictions are by intent.

9.2.1 Synthesis Subset

Verilator supports the Synthesis subset with other verification constructs being added over time. Verilator also simulates events as Synopsys's Design Compiler would; namely given a block of the form:

```
always @(x) y = x & z;
```

This will recompute y when there is even a potential for change in x or a change in z, that is when the flops computing x or z evaluate (which is what Design Compiler will synthesize.) A compliant simulator would only calculate y if x changes. We recommend using always_comb to make the code run the same everywhere. Also avoid putting $displays in combo blocks, as they may print multiple times when not desired, even on compliant simulators as event ordering is not specified.

9.2.2 Signal Naming

To avoid conflicts with C symbol naming, any character in a signal name that is not alphanumeric nor a single underscore will be replaced by __0hh where hh is the hex code of the character. To avoid conflicts with Verilator's internal symbols, any double underscore are replaced with ___05F (5F is the hex code of an underscore.)

9.2.3 Bind

sVerilator only supports bind to a target module name, not to an instance path.

9.2.4 Class

Verilator class support is limited but in active development. Verilator supports members, and methods. Verilator does not support class static members, class extend, or class parameters.

9.2.5 Dotted cross-hierarchy references

Verilator supports dotted references to variables, functions and tasks in different modules. The portion before the dot must have a constant value; for example a[2].b is acceptable, while a[x].b is generally not.

References into generated and arrayed instances use the instance names specified in the Verilog standard; arrayed instances are named `{instanceName}[{instanceNumber}]` in Verilog, which becomes `{instanceName}__BRA__{instanceNumber}__KET__` inside the generated C++ code.

9.2.6 Latches

Verilator is optimized for edge sensitive (flop based) designs. It will attempt to do the correct thing for latches, but most performance optimizations will be disabled around the latch.

9.2.7 Structures and Unions

Presently Verilator only supports packed structs and packed unions. Rand and randc tags on members are simply ignored. All structures and unions are represented as a single vector, which means that generating one member of a structure from blocking, and another from non-blocking assignments is unsupported.

9.2.8 Time

All delays (#) are ignored, as they are in synthesis.

9.2.9 Unknown States

Verilator is mostly a two state simulator, not a four state simulator. However, it has two features which uncover most initialization bugs (including many that a four state simulator will miss.)

Identity comparisons (=== or !==) are converted to standard ==/!= when neither side is a constant. This may make the expression yield a different result compared to a four state simulator. An === comparison to X will always be false, so that Verilog code which checks for uninitialized logic will not fire.

Assigning X to a variable will actually assign a constant value as determined by the *--x-assign* option. This allows runtime randomization, thus if the value is actually used, the random value should cause downstream errors. Integers also get randomized, even though the Verilog 2001 specification says they initialize to zero. Note however that randomization happens at initialization time and hence during a single simulation run, the same constant (but random) value will be used every time the assignment is executed.

All variables, depending on *--x-initial* setting, are typically randomly initialized using a function. By running several random simulation runs you can determine that reset is working correctly. On the first run, have the function initialize variables to zero. On the second, have it initialize variables to one. On the third and following runs have it initialize them randomly. If the results match, reset works. (Note this is what the hardware will really do.) In practice, just setting all variables to one at startup finds most problems (since typically control signals are active-high).

--x-assign applies to variables explicitly initialized or assigned an X. Uninitialized clocks are initialized to zero, while all other state holding variables are initialized to a random value. Event driven simulators will generally trigger an edge on a transition from X to 1 (posedge) or X to 0 (negedge). However, by default, since clocks are initialized to zero, Verilator will not trigger an initial negedge. Some code (particularly for reset) may rely on X->0 triggering an edge. The *--x-initial-edge* option enables this behavior. Comparing runs with and without this option will find such problems.

9.2.10 Tri/Inout

Verilator converts some simple tristate structures into two state. Pullup, pulldown, bufif0, bufif1, notif0, notif1, pmos, nmos, tri0 and tri1 are also supported. Simple comparisons with `=== 1'bz` are also supported.

An assignment of the form:

```
inout driver;
wire driver = (enable) ? output_value : 1'bz;
```

Will be converted to:

```
input driver;       // Value being driven in from "external" drivers
output driver__en;  // True if driven from this module
output driver__out; // Value being driven from this module
```

External logic will be needed to combine these signals with any external drivers.

Tristate drivers are not supported inside functions and tasks; an inout there will be considered a two state variable that is read and written instead of a four state variable.

9.2.11 Functions & Tasks

All functions and tasks will be inlined (will not become functions in C.) The only support provided is for simple statements in tasks (which may affect global variables).

Recursive functions and tasks are not supported. All inputs and outputs are automatic, as if they had the Verilog 2001 "automatic" keyword prepended. (If you don't know what this means, Verilator will do what you probably expect, what C does. The default behavior of Verilog is different.)

9.2.12 Generated Clocks

Verilator attempts to deal with generated and gated clocks correctly, however some cases cause problems in the scheduling algorithm which is optimized for performance. The safest option is to have all clocks as primary inputs to the model, or wires directly attached to primary inputs. For proper behavior clock enables may also need the `/*verilator&32;clock_enable*/` metacomment.

9.2.13 Gate Primitives

The 2-state gate primitives (and, buf, nand, nor, not, or, xnor, xor) are directly converted to behavioral equivalents. The 3-state and MOS gate primitives are not supported. Tables are not supported.

9.2.14 Specify blocks

All specify blocks and timing checks are ignored. All min:typ:max delays use the typical value.

9.2.15 Array Initialization

When initializing a large array, you need to use non-delayed assignments. Verilator will tell you when this needs to be fixed; see the BLKLOOPINIT error for more information.

9.2.16 Array Out of Bounds

Writing a memory element that is outside the bounds specified for the array may cause a different memory element inside the array to be written instead. For power-of-2 sized arrays, Verilator will give a width warning and the address. For non-power-of-2-sizes arrays, index 0 will be written.

Reading a memory element that is outside the bounds specified for the array will give a width warning and wrap around the power-of-2 size. For non-power-of-2 sizes, it will return a unspecified constant of the appropriate width.

9.2.17 Assertions

Verilator is beginning to add support for assertions. Verilator currently only converts assertions to simple `if (...) error` statements, and coverage statements to increment the line counters described in the coverage section.

Verilator does not support SEREs yet. All assertion and coverage statements must be simple expressions that complete in one cycle.

9.2.18 Encrypted Verilog

Open source simulators like Verilator are unable to use encrypted RTL (i.e. IEEE P1735). Talk to your IP vendor about delivering IP blocks via Verilator's `--protect-lib` feature.

9.3 Language Keyword Limitations

This section describes specific limitations for each language keyword.

`__FILE__, `__LINE__, `begin_keywords, `begin_keywords, `begin_keywords, `begin_keywords, `begin_keywords, `define, `
Fully supported.

always, always_comb, always_ff, always_latch, and, assign, begin, buf, byte, case, casex, casez, default, defparam, do-while, else,
Generally supported.

++, – operators Increment/decrement can only be used as standalone statements or in certain limited cases.

'{} operator Assignment patterns with order based, default, constant integer (array) or member identifier (struct/union) keys are supported. Data type keys and keys which are computed from a constant expression are not supported.

`uselib Uselib, a vendor specific library specification method, is ignored along with anything following it until the end of that line.

cast operator Casting is supported only between simple scalar types, signed and unsigned, not arrays nor structs.

chandle Treated as a "longint"; does not yet warn about operations that are specified as illegal on chandles.

disable Disable statements may be used only if the block being disabled is a block the disable statement itself is inside. This was commonly used to provide loop break and continue functionality before SystemVerilog added the break and continue keywords.

inside Inside expressions may not include unpacked array traversal or $ as an upper bound. Case inside and case matches are also unsupported.

interface Interfaces and modports, including with generated data types are supported. Generate blocks around modports are not supported, nor are virtual interfaces nor unnamed interfaces.

shortreal Short floating point (shortreal) numbers are converted to real. Most other simulators either do not support float, or convert likewise.

specify specparam All specify blocks and timing checks are ignored.

uwire Verilator does not perform warning checking on uwires, it treats the uwire keyword as if it were the normal wire keyword.

$bits, $countbits, $countones, $finish, $isunknown, $onehot, $onehot0, $signed, $stime, $stop, $time, $unsigned,
Generally supported.

$dump/$dumpports and related $dumpfile or $dumpports will create a VCD or FST file (which is based on the `--trace` option given when the model was Verilated). This will take effect starting at the next eval() call. If you have multiple Verilated designs under the same C model, then this will dump signals only from the design containing the $dumpvars.

$dumpvars and $dumpports module identifier is ignored; the traced instances will always start at the top of the design. The levels argument is also ignored, use tracing_on/tracing_off pragmas instead.

$dumpportson/$dumpportsoff/$dumpportsall/$dumpportslimit filename argument is ignored, only a single trace file may be active at once.

$dumpall/$dumpportsall, $dumpon/$dumpportson, $dumpoff/$dumpportsoff, and $dumplimit/$dumpportlimit are currently ignored.

$error, $fatal, $info, $warning. Generally supported.

$exit, $finish, $stop The rarely used optional parameter to $finish and $stop is ignored. $exit is aliased to $finish.

$fopen, $fclose, $fdisplay, $ferror, $feof, $fflush, $fgetc, $fgets, $fscanf, $fwrite, $fscanf, $sscanf Generally supported.

$fullskew, $hold, $nochange, $period, $recovery, $recrem, $removal, $setup, $setuphold, $skew, $timeskew, $width All specify blocks and timing checks are ignored.

$random, $urandom, $urandom_range Use *+verilator+seed+<value>* runtime option to set the seed if there is no $random nor $urandom optional argument to set the seed. There is one random seed per C thread, not per module for $random, nor per object for random stability of $urandom/$urandom_range.

$readmemb, $readmemh Read memory commands are supported. Note Verilator and the Verilog specification does not include support for readmem to multi-dimensional arrays.

$test$plusargs, $value$plusargs Supported, but the instantiating C++/SystemC wrapper must call

```
Verilated::commandArgs(argc, argv);
```

to register the command line before calling $test$plusargs or $value$plusargs.

CHAPTER

TEN

LANGUAGE EXTENSIONS

The following additional constructs are the extensions Verilator supports on top of standard Verilog code. Using these features outside of comments or "*ifdef*"'s may break other tools.

`__FILE__
: The *`__FILE__* define expands to the current filename as a string, like C++'s __FILE__. This Verilator feature added in 2006 was incorporated into the IEEE 1800-2009 standard.

`__LINE__
: The *`__LINE__* define expands to the current filename as a string, like C++'s __LINE__. This Verilator feature added in 2006 was incorporated into the IEEE 1800-2009 standard.

**`error** `[string]`
: This will report an error when encountered, like C++'s #error.

$c([string], ...);
: The string will be embedded directly in the output C++ code at the point where the surrounding Verilog code is compiled. It may either be a standalone statement (with a trailing ; in the string), or a function that returns up to a 32-bit number (without a trailing ;). This can be used to call C++ functions from your Verilog code.

  String arguments will be put directly into the output C++ code, except the word 'this' (i.e.: the object pointer) might be replaced with a different pointer as Verilator might implement logic with non-member functions. For this reason, any references to class members must be made via an explicit 'this->' pointer dereference.

  Expression arguments will have the code to evaluate the expression inserted. Thus to call a C++ function, `$c("func(",a,")")` will result in `func(a)` in the output C++ code. For input arguments, rather than hard-coding variable names in the string `$c("func(a)")`, instead pass the variable as an expression :`$c("func(",a,")")`. This will allow the call to work inside Verilog functions where the variable is flattened out, and also enable other optimizations.

  Verilator does not use any text inside the quotes for ordering/scheduling. If you need the $c to be called at a specific time, e.g. when a variable changes, then the $c must be under an appropriate sensitivity statement, e.g. `always @(posedge clk) $c("func()")` to call it on every edge, or e.g. `always @* c("func(",a,")")` to call it when `a` changes (the latter working because `a` is outside the quotes).

  If you will be reading or writing any Verilog variables inside the C++ functions, the Verilog signals must be declared with */*verilator&32;public*/* metacomments.

  You may also append an arbitrary number to $c, generally the width of the output; `signal_32_bits = $c32("...");`. This allows for compatibility with other simulators which require a differently named PLI function name for each different output width.

$display, $write, $fdisplay, $fwrite, $sformat, $swrite
: Format arguments may use C fprintf sizes after the % escape. Per the Verilog standard, %x prints a number with the natural width, and %0x prints a number with minimum width. Verilator extends this so %5x prints 5 digits per the C standard (this is unspecified in Verilog, but was incorporated into the 1800-2009).

`coverage_block_off
Specifies the entire begin/end block should be ignored for coverage analysis. Must be inside a code block, e.g. within a begin/end pair. Same as *coverage_block_off* in *Configuration Files*.

`systemc_header
Take remaining text up to the next *`verilog* or *`systemc_...* mode switch and place it verbatim into the output .h file's header. Must be placed as a module item, e.g. directly inside a module/endmodule pair. Despite the name of this macro, this also works in pure C++ code.

`systemc_ctor
Take remaining text up to the next *`verilog* or *`systemc_...* mode switch and place it verbatim into the C++ class constructor. Must be placed as a module item, e.g. directly inside a module/endmodule pair. Despite the name of this macro, this also works in pure C++ code.

`systemc_dtor
Take remaining text up to the next *`verilog* or *`systemc_...* mode switch and place it verbatim into the C++ class destructor. Must be placed as a module item, e.g. directly inside a module/endmodule pair. Despite the name of this macro, this also works in pure C++ code.

`systemc_interface
Take remaining text up to the next *`verilog* or *`systemc_...* mode switch and place it verbatim into the C++ class interface. Must be placed as a module item, e.g. directly inside a module/endmodule pair. Despite the name of this macro, this also works in pure C++ code.

`systemc_imp_header
Take remaining text up to the next *`verilog* or *`systemc_...* mode switch and place it verbatim into the header of all files for this C++ class implementation. Must be placed as a module item, e.g. directly inside a module/endmodule pair. Despite the name of this macro, this also works in pure C++ code.

`systemc_implementation
Take remaining text up to the next *`verilog* or *`systemc_...* mode switch and place it verbatim into a single file of the C++ class implementation. Must be placed as a module item, e.g. directly inside a module/endmodule pair. Despite the name of this macro, this also works in pure C++ code.

If you will be reading or writing any Verilog variables in the C++ functions, the Verilog signals must be declared with a */*verilator&32;public*/* metacomment. See also the public task feature; writing an accessor may result in cleaner code.

`SYSTEMVERILOG
The SYSTEMVERILOG, SV_COV_START and related standard defines are set by default when *--language* is "1800-*".

`VERILATOR

`verilator

`verilator3
The VERILATOR, verilator and verilator3 defines are set by default so you may "`ifdef" around tool specific constructs.

`verilator_config
Take remaining text up to the next *`verilog* mode switch and treat it as Verilator configuration commands. See *Configuration Files*.

`verilog
Switch back to processing Verilog code after a *`systemc_...* mode switch. The Verilog code returns to the last language mode specified with "`begin_keywords", or SystemVerilog if none was specified.

/*verilator&32;clock_enable*/
Used after a signal declaration to indicate the signal is used to gate a clock, and the user takes responsibility for

insuring there are no races related to it. (Typically by adding a latch, and running static timing analysis.) For example:

```
reg enable_r /*verilator clock_enable*/;
wire gated_clk = clk & enable_r;
always_ff @(posedge clk)
   enable_r <= enable_early;
```

The clock_enable attribute will cause the clock gate to be ignored in the scheduling algorithm, sometimes required for correct clock behavior, and always improving performance. It's also a good idea to enable the *IMPERFECTSCH* warning, to ensure all clock enables are properly recognized.

Same as *clock_enable* configuration file option.

/*verilator&32;clocker*/

/*verilator&32;no_clocker*/

Specifies that the signal is used as clock or not. This information is used by Verilator to mark the signal and any derrived signals as clocker. See *--clk*.

Same as *clocker* and *no_clocker* in configuration files.

/*verilator&32;coverage_block_off*/

Specifies the entire begin/end block should be ignored for coverage analysis purposes.

Same as *coverage_block_off* configuration file option.

/*verilator&32;coverage_off*/

Specifies that following lines of code should have coverage disabled. Often used to ignore an entire module for coverage analysis purposes.

/*verilator&32;coverage_on*/

Specifies that following lines of code should have coverage re-enabled (if appropriate *--coverage* flags are passed) after being disabled earlier with */*verilator&32;coverage_off*/*.

/*verilator&32;hier_block*/

Specifies that the module is a unit of hierarchical Verilation. This metacomment must be between `module module_name(...);` and `endmodule`. The module will not be inlined nor uniquified for each instance in hierarchical Verilation. Note that the metacomment is ignored unless the *--hierarchical* option is specified.

See *Hierarchical Verilation*.

/*verilator&32;inline_module*/

Specifies the module the comment appears in may be inlined into any modules that use this module. This is useful to speed up simulation runtime. Note if using *--public* that signals under inlined submodules will be named `{submodule}__DOT__{subsignal}` as C++ does not allow "." in signal names.

Same as *inline* configuration file option.

/*verilator&32;isolate_assignments*/

Used after a signal declaration to indicate the assignments to this signal in any blocks should be isolated into new blocks. When there is a large combinatorial block that is resulting in an `UNOPTFLAT` warning, attaching this to the signal causing a false loop may clear up the problem.

IE, with the following:

```
reg splitme /* verilator isolate_assignments*/;
// Note the placement of the semicolon above
always_comb begin
  if (....) begin
```

(continues on next page)

(continued from previous page)

```
    splitme = ....;
    other assignments
  end
end
```

Verilator will internally split the block that assigns to "splitme" into two blocks:

It would then internally break it into (sort of):

```
// All assignments excluding those to splitme
always_comb begin
  if (....) begin
    other assignments
  end
end
// All assignments to splitme
always_comb begin
  if (....) begin
    splitme = ....;
  end
end
```

Same as *isolate_assignments* configuration file option.

/*verilator&32;lint_off <msg>*/
Disable the specified warning message for any warnings following the comment.

/*verilator&32;lint_on <msg>*/
Re-enable the specified warning message for any warnings following the comment.

/*verilator&32;lint_restore*/
After a /*verilator&32;lint_save*/, pop the stack containing lint message state. Often this is useful at the bottom of include files.

/*verilator&32;lint_save*/
Push the current state of what lint messages are turned on or turned off to a stack. Later meta-comments may then lint_on or lint_off specific messages, then return to the earlier message state by using /*verilator&32;lint_restore*/. For example:

```
// verilator lint_save
// verilator lint_off WIDTH
...  // code needing WIDTH turned off
// verilator lint_restore
```

If WIDTH was on before the lint_off, it will now be restored to on, and if it was off before the lint_off it will remain off.

/*verilator&32;no_inline_module*/
Specifies the module the comment appears in should not be inlined into any modules that use this module.

Same as *no_inline* configuration file option.

/*verilator&32;no_inline_task*/
Used in a function or task variable definition section to specify the function or task should not be inlined into where it is used. This may reduce the size of the final executable when a task is used a very large number of times. For this flag to work, the task and tasks below it must be pure; they cannot reference any variables outside the task itself.

Same as *no_inline* configuration file option.

/*verilator&32;public*/ (on parameter)
: Used after a parameter declaration to indicate the emitted C code should have the parameter values visible. Due to C++ language restrictions, this may only be used on 64-bit or narrower integral enumerations.

```
parameter [2:0] PARAM /*verilator public*/ = 2'b0;
```

/*verilator&32;public*/ (on typedef enum)
: Used after an enum typedef declaration to indicate the emitted C code should have the enum values visible. Due to C++ language restrictions, this may only be used on 64-bit or narrower integral enumerations.

```
typedef enum logic [2:0] { ZERO = 3'b0 } pub_t /*verilator public*/;
```

/*verilator&32;public*/ (on variable)
: Used after an input, output, register, or wire declaration to indicate the signal should be declared so that C code may read or write the value of the signal. This will also declare this module public, otherwise use /*verilator&32;public_flat*/.

  Instead of using public variables, consider instead making a DPI or public function that accesses the variable. This is nicer as it provides an obvious entry point that is also compatible across simulators.

  Same as *public* configuration file option.

/*verilator&32;public*/ (on task/function)
: Used inside the declaration section of a function or task declaration to indicate the function or task should be made into a C++ function, public to outside callers. Public tasks will be declared as a void C++ function, public functions will get the appropriate non-void (bool, uint32_t, etc) return type. Any input arguments will become C++ arguments to the function. Any output arguments will become C++ reference arguments. Any local registers/integers will become function automatic variables on the stack.

  Wide variables over 64 bits cannot be function returns, to avoid exposing complexities. However, wide variables can be input/outputs; they will be passed as references to an array of 32-bit numbers.

  Generally, only the values of stored state (flops) should be written, as the model will NOT notice changes made to variables in these functions. (Same as when a signal is declared public.)

  You may want to use DPI exports instead, as it's compatible with other simulators.

  Same as *public* configuration file option.

/*verilator&32;public_flat*/ (on variable)
: Used after an input, output, register, or wire declaration to indicate the signal should be declared so that C code may read or write the value of the signal. This will not declare this module public, which means the name of the signal or path to it may change based upon the module inlining which takes place.

  Same as *public_flat* configuration file option.

/*verilator&32;public_flat_rd*/ (on variable)
: Used after an input, output, register, or wire declaration to indicate the signal should be declared public_flat (see above), but read-only.

  Same as *public_flat_rd* configuration file option.

/*verilator&32;public_flat_rw @(<edge_list>)*/ (on variable)
: Used after an input, output, register, or wire declaration to indicate the signal should be declared public_flat_rd (see above), and also writable, where writes should be considered to have the timing specified by the given sensitivity edge list. Set for all variables, ports and wires using the *--public-flat-rw* option.

  Same as *public_flat_rw* configuration file option.

/*verilator&32;public_module*/
: Used after a module statement to indicate the module should not be inlined (unless specifically requested) so

that C code may access the module. Verilator automatically sets this attribute when the module contains any public signals or `systemc_ directives. Also set for all modules when using the *--public* option.

Same as *public* configuration file option.

/*verilator&32;sc_clock*/
Deprecated and ignored. Previously used after an input declaration to indicate the signal should be declared in SystemC as a sc_clock instead of a bool. This was needed in SystemC 1.1 and 1.2 only; versions 2.0 and later do not require clock pins to be sc_clocks and this is no longer needed and is ignored.

/*verilator&32;sc_bv*/
Used after a port declaration. It sets the port to be of `sc_bv<{width}>` type, instead of bool, vluint32_t or vluint64_t. This may be useful if the port width is parameterized and the instantiating C++ code wants to always have a sc_bv so it can accept any width. In general you should avoid using this attribute when not necessary as with increasing usage of sc_bv the performance decreases significantly.

Same as *sc_bv* configuration file option.

/*verilator&32;sformat*/
Attached to the final argument of type "input string" of a function or task to indicate the function or task should pass all remaining arguments through $sformatf. This allows creation of DPI functions with $display like behavior. See the `test_regress/t/t_dpi_display.v` file for an example.

Same as *sformat* configuration file option.

/*verilator&32;split_var*/
Attached to a variable or a net declaration to break the variable into multiple pieces typically to resolve `UNOPTFLAT` performance issues. Typically the variables to attach this to are recommended by Verilator itself, see *UNOPTFLAT*.

For example, Verilator will internally convert a variable with the metacomment such as:

```
logic [7:0] x [0:1]  /*verilator split_var*/;
```

To:

```
logic [7:0] x__BRA__0__KET__ /*verilator split_var*/;
logic [7:0] x__BRA__1__KET__ /*verilator split_var*/;
```

Note that the generated packed variables retain the split_var metacomment because they may be split into further smaller pieces according to the access patterns.

This only supports unpacked arrays, packed arrays, and packed structs of integer types (reg, logic, bit, byte, int...); otherwise if a split was requested but cannot occur a SPLITVAR warning is issued. Splitting large arrays may slow down the Verilation speed, so use this only on variables that require it.

Same as *split_var* configuration file option.

/*verilator&32;tag `<text...>*/`
Attached after a variable or structure member to indicate opaque (to Verilator) text that should be passed through to the XML output as a tag, for use by downstream applications.

/*verilator&32;tracing_off*/
Disable waveform tracing for all future signals that are declared in this module, or instances below this module. Often this is placed just after a primitive's module statement, so that the entire module and instances below it are not traced.

/*verilator&32;tracing_on*/
Re-enable waveform tracing for all future signals or instances that are declared.

CHAPTER

ELEVEN

EXECUTABLE AND ARGUMENT REFERENCE

This section describes the executables that are part of Verilator, and the options to each executable.

11.1 verilator Arguments

The following are the arguments that may be passed to the "verilator" executable.

Summary:

```
<file.v>                     Verilog package, module and top module filenames
<file.c/cc/cpp>              Optional C++ files to compile in
<file.a/o/so>                Optional C++ files to link in

 +1364-1995ext+<ext>         Use Verilog 1995 with file extension <ext>
 +1364-2001ext+<ext>         Use Verilog 2001 with file extension <ext>
 +1364-2005ext+<ext>         Use Verilog 2005 with file extension <ext>
 +1800-2005ext+<ext>         Use SystemVerilog 2005 with file extension <ext>
 +1800-2009ext+<ext>         Use SystemVerilog 2009 with file extension <ext>
 +1800-2012ext+<ext>         Use SystemVerilog 2012 with file extension <ext>
 +1800-2017ext+<ext>         Use SystemVerilog 2017 with file extension <ext>
--assert                     Enable all assertions
--autoflush                  Flush streams after all $displays
--bbox-sys                   Blackbox unknown $system calls
--bbox-unsup                 Blackbox unsupported language features
--bin <filename>             Override Verilator binary
--build                      Build model executable/library after Verilation
 -CFLAGS <flags>             C++ compiler arguments for makefile
--cc                         Create C++ output
--cdc                        Clock domain crossing analysis
--clk <signal-name>          Mark specified signal as clock
--make <build-tool>          Generate scripts for specified build tool
--compiler <compiler-name>   Tune for specified C++ compiler
--converge-limit <loops>     Tune convergence settle time
--coverage                   Enable all coverage
--coverage-line              Enable line coverage
--coverage-max-width <width>   Maximum array depth for coverage
--coverage-toggle            Enable toggle coverage
--coverage-user              Enable SVL user coverage
--coverage-underscore        Enable coverage of _signals
 -D<var>[=<value>]           Set preprocessor define
--debug                      Enable debugging
--debug-check                Enable debugging assertions
--no-debug-leak              Disable leaking memory in --debug mode
```

(continues on next page)

(continued from previous page)

```
--debugi <level>            Enable debugging at a specified level
--debugi-<srcfile> <level>  Enable debugging a source file at a level
--default-language <lang>   Default language to parse
 +define+<var>=<value>      Set preprocessor define
--dpi-hdr-only              Only produce the DPI header file
--dump-defines              Show preprocessor defines with -E
--dump-tree                 Enable dumping .tree files
--dump-treei <level>        Enable dumping .tree files at a level
--dump-treei-<srcfile> <level>  Enable dumping .tree file at a source file␣
↪at a level
--dump-tree-addrids         Use short identifiers instead of addresses
 -E                         Preprocess, but do not compile
--error-limit <value>       Abort after this number of errors
--exe                       Link to create executable
--expand-limit <value>      Set expand optimization limit
 -F <file>                  Parse arguments from a file, relatively
 -f <file>                  Parse arguments from a file
 -FI <file>                 Force include of a file
--flatten                   Force inlining of all modules, tasks and␣
↪functions
 -G<name>=<value>           Overwrite top-level parameter
--gdb                       Run Verilator under GDB interactively
--gdbbt                     Run Verilator under GDB for backtrace
--generate-key              Create random key for --protect-key
--getenv <var>              Get environment variable with defaults
--help                      Display this help
--hierarchical              Enable hierarchical Verilation
 -I<dir>                    Directory to search for includes
 -j <jobs>                  Parallelism for --build
--gate-stmts <value>        Tune gate optimizer depth
--if-depth <value>          Tune IFDEPTH warning
 +incdir+<dir>              Directory to search for includes
--inline-mult <value>       Tune module inlining
--instr-count-dpi <value>   Assumed dynamic instruction count of DPI imports
 -LDFLAGS <flags>           Linker pre-object arguments for makefile
--l2-name <value>           Verilog scope name of the top module
--language <lang>           Default language standard to parse
--lib-create <name>         Create a DPI library
 +libext+<ext>+[ext]...     Extensions for finding modules
--lint-only                 Lint, but do not make output
 -MAKEFLAGS <flags>         Arguments to pass to make during --build
--max-num-width <value>     Maximum number width (default: 64K)
--MMD                       Create .d dependency files
--MP                        Create phony dependency targets
--Mdir <directory>          Name of output object directory
--no-merge-const-pool       Disable merging of different types in const pool
--mod-prefix <topname>      Name to prepend to lower classes
--no-clk <signal-name>      Prevent marking specified signal as clock
--no-decoration             Disable comments and symbol decorations
--no-pins64                 Don't use vluint64_t's for 33-64 bit sigs
--no-skip-identical         Disable skipping identical output
 +notimingchecks            Ignored
 -O0                        Disable optimizations
 -O3                        High performance optimizations
 -O<optimization-letter>    Selectable optimizations
 -o <executable>            Name of final executable
--no-order-clock-delay      Disable ordering clock enable assignments
```

(continues on next page)

(continued from previous page)

```
--no-verilate               Skip verilation and just compile previously␣
↪Verilated code.
--output-split <statements>          Split .cpp files into pieces
--output-split-cfuncs <statements>   Split model functions
--output-split-ctrace <statements>   Split tracing functions
 -P                         Disable line numbers and blanks with -E
--pins-bv <bits>            Specify types for top level ports
--pins-sc-uint              Specify types for top level ports
--pins-sc-biguint           Specify types for top level ports
--pins-uint8                Specify types for top level ports
--pipe-filter <command>     Filter all input through a script
--pp-comments               Show preprocessor comments with -E
--prefix <topname>          Name of top level class
--prof-c                    Compile C++ code with profiling
--prof-cfuncs               Name functions for profiling
--prof-threads              Enable generating gantt chart data for threads
--protect-key <key>         Key for symbol protection
--protect-ids               Hash identifier names for obscurity
--protect-lib <name>        Create a DPI protected library
--private                   Debugging; see docs
--public                    Debugging; see docs
--public-flat-rw            Mark all variables, etc as public_flat_rw
 -pvalue+<name>=<value>     Overwrite toplevel parameter
--quiet-exit                Don't print the command on failure
--relative-includes         Resolve includes relative to current file
--reloop-limit              Minimum iterations for forming loops
--report-unoptflat          Extra diagnostics for UNOPTFLAT
--rr                        Run Verilator and record with rr
--savable                   Enable model save-restore
--sc                        Create SystemC output
--stats                     Create statistics file
--stats-vars                Provide statistics on variables
 -sv                        Enable SystemVerilog parsing
 +systemverilogext+<ext>    Synonym for +1800-2017ext+<ext>
--threads <threads>         Enable multithreading
--threads-dpi <mode>        Enable multithreaded DPI
--threads-max-mtasks <mtasks>  Tune maximum mtask partitioning
--timescale <timescale>     Sets default timescale
--timescale-override <timescale>  Overrides all timescales
--top <topname>             Alias of --top-module
--top-module <topname>      Name of top level input module
--trace                     Enable waveform creation
--trace-coverage            Enable tracing of coverage
--trace-depth <levels>      Depth of tracing
--trace-fst                 Enable FST waveform creation
--trace-max-array <depth>   Maximum bit width for tracing
--trace-max-width <width>   Maximum array depth for tracing
--trace-params              Enable tracing of parameters
--trace-structs             Enable tracing structure names
--trace-threads <threads>   Enable waveform creation on separate threads
--trace-underscore          Enable tracing of _signals
 -U<var>                    Undefine preprocessor define
--unroll-count <loops>      Tune maximum loop iterations
--unroll-stmts <stmts>      Tune maximum loop body size
--unused-regexp <regexp>    Tune UNUSED lint signals
 -V                         Verbose version and config
 -v <filename>              Verilog library
```

(continues on next page)

(continued from previous page)

```
 +verilog1995ext+<ext>      Synonym for +1364-1995ext+<ext>
 +verilog2001ext+<ext>      Synonym for +1364-2001ext+<ext>
--version                   Displays program version and exits
--vpi                       Enable VPI compiles
--waiver-output <filename>  Create a waiver file based on the linter warnings
 -Wall                      Enable all style warnings
 -Werror-<message>          Convert warnings to errors
 -Wfuture-<message>         Disable unknown message warnings
 -Wno-<message>             Disable warning
 -Wno-context               Disable source context on warnings
 -Wno-fatal                 Disable fatal exit on warnings
 -Wno-lint                  Disable all lint warnings
 -Wno-style                 Disable all style warnings
 -Wpedantic                 Warn on compliance-test issues
--x-assign <mode>           Assign non-initial Xs to this value
--x-initial <mode>          Assign initial Xs to this value
--x-initial-edge            Enable initial X->0 and X->1 edge triggers
--xml-only                  Create XML parser output
--xml-output                XML output filename
 -y <dir>                   Directory to search for modules
```

<file.v>
: Specifies the Verilog file containing the top module to be Verilated.

<file.c/.cc/.cpp/.cxx>
: Used with *--exe* to specify optional C++ files to be linked in with the Verilog code. The file path should either be absolute, or relative to where the make will be executed from, or add to your makefile's VPATH the appropriate directory to find the file.

  See also *-CFLAGS* and *-LDFLAGS* options, which are useful when the C++ files need special compiler flags.

<file.a/.o/.so>
: Specifies optional object or library files to be linked in with the Verilog code, as a shorthand for *-LDFLAGS <file>*. The file path should either be absolute, or relative to where the make will be executed from, or add to your makefile's VPATH the appropriate directory to find the file.

  If any files are specified in this way, Verilator will include a make rule that uses these files when linking the module's executable. This generally is only useful when used with the *--exe* option.

+1364-1995ext+<ext>

+1364-2001ext+<ext>

+1364-2005ext+<ext>

+1800-2005ext+<ext>

+1800-2009ext+<ext>

+1800-2012ext+<ext>

+1800-2017ext+<ext>
: Specifies the language standard to be used with a specific filename extension, <ext>.

  For compatibility with other simulators, see also the synonyms *+verilog1995ext+<ext>*, *+verilog2001ext+<ext>*, and *+systemverilogext+<ext>*.

  For any source file, the language specified by these options takes precedence over any language specified by the *--default-language* or *--language* options.

These options take effect in the order they are encountered. Thus the following would use Verilog 1995 for `a.v` and Verilog 2001 for `b.v`:

```
verilator ... +1364-1995ext+v a.v +1364-2001ext+v b.v
```

These options are only recommended for legacy mixed language designs, as the preferable option is to edit the code to repair new keywords, or add appropriate `` `begin_keywords ``.

Note: `` `begin_keywords `` is a SystemVerilog construct, which specifies *only* the set of keywords to be recognized. This also controls some error messages that vary between language standards. Note at present Verilator tends to be overly permissive, e.g. it will accept many grammar and other semantic extensions which might not be legal when set to an older standard.

--assert
Enable all assertions.

--autoflush
After every $display or $fdisplay, flush the output stream. This ensures that messages will appear immediately but may reduce performance. For best performance call `fflush(stdout)` occasionally in the C++ main loop. Defaults to off, which will buffer output as provided by the normal C/C++ standard library IO.

--bbox-sys
Black box any unknown $system task or function calls. System tasks will simply become no-operations, and system functions will be replaced with unsized zero. Arguments to such functions will be parsed, but not otherwise checked. This prevents errors when linting in the presence of company specific PLI calls.

Using this argument will likely cause incorrect simulation.

--bbox-unsup
Black box some unsupported language features, currently UDP tables, the cmos and tran gate primitives, deassign statements, and mixed edge errors. This may enable linting the rest of the design even when unsupported constructs are present.

Using this argument will likely cause incorrect simulation.

--bin <filename>
Rarely needed. Override the default filename for Verilator itself. When a dependency (.d) file is created, this filename will become a source dependency, such that a change in this binary will have make rebuild the output files.

--build
After generating the SystemC/C++ code, Verilator will invoke the toolchain to build the model library (and executable when *--exe* is also used). Verilator manages the build itself, and for this –build requires GNU Make to be available on the platform.

-CFLAGS <flags>
Add specified C compiler argument to the generated makefiles. For multiple flags either pass them as a single argument with space separators quoted in the shell (**-CFLAGS "-a -b"**), or use multiple -CFLAGS options (**-CFLAGS -a -CFLAGS -b**).

When make is run on the generated makefile these will be passed to the C++ compiler (g++/clang++/msvc++).

--cc
Specifies C++ without SystemC output mode; see also *--sc* option.

--cdc
Permanently experimental. Perform some clock domain crossing checks and issue related warnings (CDCRSTLOGIC) and then exit; if warnings other than CDC warnings are needed make a second run with *--lint-only*. Additional warning information is also written to the file <prefix>__cdc.txt.

Currently only checks some items that other CDC tools missed; if you have interest in adding more traditional CDC checks, please contact the authors.

--clk <signal-name>
With *--clk*, the specified signal-name is taken as a root clock into the model; Verilator will mark the signal as clocker and propagate the clocker attribute automatically to other signals downstream in that clock tree.

The provided signal-name is specified using a RTL hierarchy path. For example, v.foo.bar. If the signal is the input to top-module, then directly provide the signal name. Alternatively, use a */*verilator&32; clocker*/* metacomment in RTL file to mark the signal directly.

If clock signals are assigned to vectors and then later used as individual bits, Verilator will attempt to decompose the vector and connect the single-bit clock signals.

The clocker attribute is useful in cases where Verilator does not properly distinguish clock signals from other data signals. Using clocker will cause the signal indicated to be considered a clock, and remove it from the combinatorial logic reevaluation checking code. This may greatly improve performance.

--compiler <compiler-name>
Enables workarounds for the specified C++ compiler (list below). Currently this does not change any performance tuning options, but it may in the future.

clang Tune for clang. This may reduce execution speed as it enables several workarounds to avoid silly hard-coded limits in clang. This includes breaking deep structures as for msvc as described below.

gcc Tune for GNU C++, although generated code should work on almost any compliant C++ compiler. Currently the default.

msvc Tune for Microsoft Visual C++. This may reduce execution speed as it enables several workarounds to avoid silly hard-coded limits in MSVC++. This includes breaking deeply nested parenthesized expressions into sub-expressions to avoid error C1009, and breaking deep blocks into functions to avoid error C1061.

--converge-limit <loops>
Rarely needed. Specifies the maximum number of runtime iterations before creating a model failed to converge error. Defaults to 100.

--coverage
Enables all forms of coverage, alias for *--coverage-line --coverage-toggle --coverage-user*.

--coverage-line
Enables basic block line coverage analysis. See *Line Coverage*.

--coverage-max-width <width>
Rarely needed. Specify the maximum bit width of a signal that is subject to toggle coverage. Defaults to 256, as covering large vectors may greatly slow coverage simulations.

--coverage-toggle
Enables adding signal toggle coverage. See *Toggle Coverage*.

--coverage-underscore
Enable coverage of signals that start with an underscore. Normally, these signals are not covered. See also *--trace-underscore* option.

--coverage-user
Enables adding user inserted functional coverage. See *Functional Coverage*.

-D<var>=<value>
Defines the given preprocessor symbol. Similar to *+define*, but does not allow multiple definitions with a single option using plus signs. "+define" is fairly standard across Verilog tools while "-D" is similar to **gcc -D**.

--debug
Run under debug.

- Select the debug executable of Verilator (if available), this generally is a less-optimized binary with symbols present (so GDB can be used on it).
- Enable debugging messages (equivalent to *--debugi 3*).
- Enable internal assertions (equivalent to *--debug-check*).
- Enable intermediate form dump files (equivalent to *--dump-treei 3*).
- Leak to make node numbers unique (equivalent to *--debug-leak*.
- Call abort() instead of exit() if there are any errors (so GDB can see the program state).

--debug-check
Rarely needed. Enable internal debugging assertion checks, without changing debug verbosity. Enabled automatically with *--debug* option.

--no-debug-leak
In *--debug* mode, by default Verilator intentionally leaks AstNode instances instead of freeing them, so that each node pointer is unique in the resulting tree files and dot files.

This option disables the leak. This may avoid out-of-memory errors when Verilating large models in *--debug* mode.

Outside of *--debug* mode, AstNode instances should never be leaked and this option has no effect.

--debugi <level>
Rarely needed - for developer use. Set internal debugging level globally to the specified debug level (1-10). Higher levels produce more detailed messages.

--debugi-<srcfile> <level>
Rarely needed - for developer use. Set the specified Verilator source file to the specified level (e.g. *--debugi-V3Width 9*). Higher levels produce more detailed messages. See *--debug* for other implications of enabling debug.

--default-language <value>
Select the language to be used by default when first processing each Verilog file. The language value must be "VAMS", "1364-1995", "1364-2001", "1364-2001-noconfig", "1364-2005", "1800-2005", "1800-2009", "1800-2012", "1800-2017", or "1800+VAMS".

Any language associated with a particular file extension (see the various +<lang>*ext+ options) will be used in preference to the language specified by *--default-language*.

The *--default-language* is only recommended for legacy code using the same language in all source files, as the preferable option is to edit the code to repair new keywords, or add appropriate \`begin_keywords. For legacy mixed language designs, the various +<lang>ext+ options should be used.

If no language is specified, either by this option or +<lang>ext+ options, then the latest SystemVerilog language (IEEE 1800-2017) is used.

+define+<var>=<value>

+define+<var>=<value>[+<var2>=<value2>][...]
Defines the given preprocessor symbol, or multiple symbols if separated by plus signs. Similar to *-D*; +define is fairly standard across Verilog tools while *-D* is similar to **gcc -D**.

--dpi-hdr-only
Only generate the DPI header file. This option has no effect on the name or location of the emitted DPI header file, it is output in *--Mdir* as it would be without this option.

--dump-defines
With *-E*, suppress normal output, and instead print a list of all defines existing at the end of pre-processing the input files. Similar to GCC "-dM" option. This also gives you a way of finding out what is predefined in Verilator using the command:

```
touch foo.v ; verilator -E --dump-defines foo.v
```

--dump-tree
Rarely needed. Enable writing .tree debug files with dumping level 3, which dumps the standard critical stages. For details on the format see the Verilator Internals manual. *--dump-tree* is enabled automatically with *--debug*, so *--debug --no-dump-tree* may be useful if the dump files are large and not desired.

--dump-treei <level>

--dump-treei-<srcfile> <level>
Rarely needed - for developer use. Set internal tree dumping level globally to a specific dumping level or set the specified Verilator source file to the specified tree dumping level (e.g. *--dump-treei-V3Order 9*). Level 0 disables dumps and is equivalent to *--no-dump-tree*. Level 9 enables dumping of every stage.

--dump-tree-addrids
Rarely needed - for developer use. Replace AST node addresses with short identifiers in tree dumps to enhance readability. Each unique pointer value is mapped to a unique identifier, but note that this is not necessarily unique per node instance as an address might get reused by a newly allocated node after a node with the same address has been dumped then freed.

-E
Preprocess the source code, but do not compile, similar to C++ preprocessing using **gcc -E**. Output is written to standard out. Beware of enabling debugging messages, as they will also go to standard out.

See also *--dump-defines*, *-P*, and *--pp-comments* options.

--error-limit <value>
After this number of errors are encountered during Verilator run, exit. Warnings are not counted in this limit. Defaults to 50.

Does not affect simulation runtime errors, for those see *+verilator+error+limit+<value>*.

--exe
Generate an executable. You will also need to pass additional .cpp files on the command line that implement the main loop for your simulation.

--expand-limit <value>
Rarely needed. Fine-tune optimizations to set the maximum size of an expression in 32-bit words to expand into separate word-based statements.

-F <file>
Read the specified file, and act as if all text inside it was specified as command line arguments. Any relative paths are relative to the directory containing the specified file. See also *-f* option. Note *-F* is fairly standard across Verilog tools.

-f <file>
Read the specified file, and act as if all text inside it was specified as command line arguments. Any relative paths are relative to the current directory. See also *-F* option. Note *-f* is fairly standard across Verilog tools.

The file may contain `//` comments which are ignored to the end of the line. It may also contain `/* .. */` comments which are ignored, be cautious that wildcards are not handled in -f files, and that `directory/*` is the beginning of a comment, not a wildcard. Any `$VAR`, `$(VAR)`, or `${VAR}` will be replaced with the specified environment variable.

-FI <file>
Force include of the specified C++ header file. All generated C++ files will insert a #include of the specified file

before any other includes. The specified file might be used to contain define prototypes of custom VL_VPRINTF functions, and may need to include verilatedos.h as this file is included before any other standard includes.

--flatten
Force flattening of the design's hierarchy, with all modules, tasks and functions inlined. Typically used with *--xml-only*. Note flattening large designs may require significant CPU time, memory and storage.

-G<name>=<value>
Overwrites the given parameter of the toplevel module. The value is limited to basic data literals:

Verilog integer literals The standard Verilog integer literals are supported, so values like 32'h8, 2'b00, 4 etc. are allowed. Care must be taken that the single quote (I') is properly escaped in an interactive shell, e.g., as -GWIDTH=8'hx.

C integer literals It is also possible to use C integer notation, including hexadecimal (0x..), octal (0..) or binary (0b..) notation.

Double literals

Double literals must be one of the following styles:

- contains a dot (.) (e.g. 1.23)
- contains an exponent (e/E) (e.g. 12e3)
- contains p/P for hexadecimal floating point in C99 (e.g. 0x123.ABCp1)

Strings Strings must be in double quotes (""). They must be escaped properly on the command line, e.g. as -GSTR="\"My String\"" or -GSTR='"My String"'.

--gate-stmts <value>
Rarely needed. Set the maximum number of statements that may be present in an equation for the gate substitution optimization to inline that equation.

--gdb
Run Verilator underneath an interactive GDB (or VERILATOR_GDB environment variable value) session. See also *--gdbbt* option.

--gdbbt
If *--debug* is specified, run Verilator underneath a GDB process and print a backtrace on exit, then exit GDB immediately. Without *--debug* or if GDB doesn't seem to work, this flag is ignored. Intended for easy creation of backtraces by users; otherwise see the *--gdb* option.

--generate-key
Generate a true-random key suitable for use with *--protect-key*, print it, and exit immediately.

--getenv <variable>
If the variable is declared in the environment, print it and exit immediately. Otherwise, if it's built into Verilator (e.g. VERILATOR_ROOT), print that and exit immediately. Otherwise, print a newline and exit immediately. This can be useful in makefiles. See also *-V*, and the various *.mk files.

--help
Displays this message and program version and exits.

--hierarchical
Enable hierarchical Verilation otherwise */*verilator&32;hier_block*/* metacomment is ignored. See *Hierarchical Verilation*.

-I<dir>
See *-y*.

--if-depth <value>
Rarely needed. Set the depth at which the IFDEPTH warning will fire, defaults to 0 which disables this warning.

+incdir+<dir>
See *-y*.

--inline-mult <value>
Tune the inlining of modules. The default value of 2000 specifies that up to 2000 new operations may be added to the model by inlining, if more than this number of operations would result, the module is not inlined. Larger values, or a value < 1 will inline everything, will lead to longer compile times, but potentially faster simulation speed. This setting is ignored for very small modules; they will always be inlined, if allowed.

--instr-count-dpi <value>
Assumed dynamic instruction count of the average DPI import. This is used by the partitioning algorithm when creating a multithread model. The default value is 200. Adjusting this to an appropriate value can yield performance improvements in multithreaded models. Ignored when creating a single threaded model.

-j [<value>]
Specify the level of parallelism for *--build*. The <value> must be a positive integer specifying the maximum number of parallel build jobs, or can be omitted. When <value> is omitted, the build will not try to limit the number of parallel build jobs but attempt to execute all independent build steps in parallel.

-LDFLAGS <flags>
Add specified C linker arguments to the generated makefiles. For multiple flags either pass them as a single argument with space separators quoted in the shell (`-LDFLAGS "-a -b"`), or use multiple -LDFLAGS arguments (`-LDFLAGS -a -LDFLAGS -b`).

When make is run on the generated makefile these will be passed to the C++ linker (ld) **after** the primary file being linked. This flag is called *-LDFLAGS* as that's the traditional name in simulators; it's would have been better called LDLIBS as that's the Makefile variable it controls. (In Make, LDFLAGS is before the first object, LDLIBS after. -L libraries need to be in the Make variable LDLIBS, not LDFLAGS.)

--l2-name <value>
Instead of using the module name when showing Verilog scope, use the name provided. This allows simplifying some Verilator-embedded modeling methodologies. Default is an l2-name matching the top module. The default before Verilator 3.884 was `--l2-name v`.

For example, the program `module t; initial $display("%m"); endmodule` will show by default "t". With `--l2-name v` it will print "v".

--language <value>
A synonym for *--default-language*, for compatibility with other tools and earlier versions of Verilator.

+libext+<ext>[+<ext>][...]
Specify the extensions that should be used for finding modules. If for example module "my" is referenced, look in `my.<ext>`. Note "+libext+" is fairly standard across Verilog tools. Defaults to ".v+.sv".

--lib-create <name>
Produces C++, Verilog wrappers and a Makefile which can in turn produce a DPI library which can be used by Verilator or other simulators along with the corresponding Verilog wrapper. The Makefile will build both a static and dynamic version of the library named `lib<name>.a` and `lib<name>.so` respectively. This is done because some simulators require a dynamic library, but the static library is arguably easier to use if possible. *--protect-lib* implies *--protect-ids*.

When using *--lib-create* it is advised to also use *--timescale-override /1fs* to ensure the model has a time resolution that is always compatible with the time precision of the upper instantiating module.

See also *--protect-lib*.

--lint-only
Check the files for lint violations only, do not create any other output.

You may also want the *-Wall* option to enable messages that are considered stylistic and not enabled by default.

If the design is not to be completely Verilated see also the *--bbox-sys* and *--bbox-unsup* options.

--make <build-tool>
Generates a script for the specified build tool.

Supported values are `gmake` for GNU Make and `cmake` for CMake. Both can be specified together. If no build tool is specified, gmake is assumed. The executable of gmake can be configured via environment variable "MAKE".

When using *--build* Verilator takes over the responsibility of building the model library/executable. For this reason *--make* cannot be specified when using *--build*.

-MAKEFLAGS <string>
When using *--build*, add the specified argument to the invoked make command line. For multiple flags either pass them as a single argument with space separators quoted in the shell (e.g. `-MAKEFLAGS "-a -b"`), or use multiple -MAKEFLAGS arguments (e.g. `-MAKEFLAGS -l -MAKEFLAGS -k`). Use of this option should not be required for simple builds using the host toolchain.

--max-num-width <value>
Set the maximum number literal width (e.g. in 1024'd22 this it the 1024). Defaults to 64K.

--MMD `=item --no-MMD`
Enable/disable creation of .d dependency files, used for make dependency detection, similar to gcc -MMD option. By default this option is enabled for *--cc* or *--sc* modes.

--MP
When creating .d dependency files with *--MMD* option, make phony targets. Similar to **gcc -MP** option.

--Mdir <directory>
Specifies the name of the Make object directory. All generated files will be placed in this directory. If not specified, "obj_dir" is used. The directory is created if it does not exist and the parent directories exist; otherwise manually create the Mdir before calling Verilator.

--no-merge-const-pool
Rarely needed. In order to minimize cache footprint, values of different data type, that are yet emitted identically in C++ are merged in the constant pool. This option disables this and causes every constant pool entry with a distinct data type to be emitted separately.

--mod-prefix <topname>
Specifies the name to prepend to all lower level classes. Defaults to the same as *--prefix*.

--no-clk <signal-name>
Prevent the specified signal from being marked as clock. See *--clk*.

--no-decoration
When creating output Verilated code, minimize comments, white space, symbol names and other decorative items, at the cost of greatly reduced readability. This may assist C++ compile times. This will not typically change the ultimate model's performance, but may in some cases.

--no-pins64
Backward compatible alias for *--pins-bv 33*.

--no-skip-identical `=item --skip-identical`
Rarely needed. Disables or enables skipping execution of Verilator if all source files are identical, and all output files exist with newer dates. By default this option is enabled for *--cc* or *--sc* modes only.

+notimingchecks
Ignored for compatibility with other simulators.

-O0
Disables optimization of the model.

-O3
Enables slow optimizations for the code Verilator itself generates (as opposed to *-CFLAGS -O3* which effects the C compiler's optimization. *-O3* may improve simulation performance at the cost of compile time. This currently sets *--inline-mult -1*.

-O<optimization-letter>
Rarely needed. Enables or disables a specific optimizations, with the optimization selected based on the letter passed. A lowercase letter disables an optimization, an upper case letter enables it. This is intended for debugging use only; see the source code for version-dependent mappings of optimizations to -O letters.

-o <executable>
Specify the name for the final executable built if using *--exe*. Defaults to the *--prefix* if not specified.

--no-order-clock-delay
Rarely needed. Disables a bug fix for ordering of clock enables with delayed assignments. This option should only be used when suggested by the developers.

--output-split <statements>
Enables splitting the output .cpp files into multiple outputs. When a C++ file exceeds the specified number of operations, a new file will be created at the next function boundary. In addition, if the total output code size exceeds the specified value, VM_PARALLEL_BUILDS will be set to 1 by default in the generated make files, making parallel compilation possible. Using *--output-split* should have only a trivial impact on model performance. But can greatly improve C++ compilation speed. The use of "ccache" (set for you if present at configure time) is also more effective with this option.

This option is on by default with a value of 20000. To disable, pass with a value of 0.

--output-split-cfuncs <statements>
Enables splitting functions in the output .cpp files into multiple functions. When a generated function exceeds the specified number of operations, a new function will be created. With *--output-split*, this will enable the C++ compiler to compile faster, at a small loss in performance that gets worse with decreasing split values. Note that this option is stronger than *--output-split* in the sense that *--output-split* will not split inside a function.

Defaults to the value of *--output-split*, unless explicitly specified.

--output-split-ctrace <statements>
Similar to *--output-split-cfuncs*, enables splitting trace functions in the output .cpp files into multiple functions.

Defaults to the value of *--output-split*, unless explicitly specified.

-P
With *-E*, disable generation of `&96;line` markers and blank lines, similar to **gcc -P**.

--pins64
Backward compatible alias for *--pins-bv 65*. Note that's a 65, not a 64.

--pins-bv <width>
Specifies SystemC inputs/outputs of greater than or equal to <width> bits wide should use sc_bv's instead of uint32/vluint64_t's. The default is "–pins-bv 65", and the value must be less than or equal to 65. Versions before Verilator 3.671 defaulted to "–pins-bv 33". The more sc_bv is used, the worse for performance. Use the */*verilator&32;sc_bv*/* metacomment to select specific ports to be sc_bv.

--pins-sc-uint
Specifies SystemC inputs/outputs of greater than 2 bits wide should use sc_uint between 2 and 64. When combined with the *--pins-sc-biguint* combination, it results in sc_uint being used between 2 and 64 and sc_biguint being used between 65 and 512.

--pins-sc-biguint
Specifies SystemC inputs/outputs of greater than 65 bits wide should use sc_biguint between 65 and 512, and

sc_bv from 513 upwards. When combined with the *--pins-sc-uint* combination, it results in sc_uint being used between 2 and 64 and sc_biguint being used between 65 and 512.

--pins-uint8
Specifies SystemC inputs/outputs that are smaller than the *--pins-bv* setting and 8 bits or less should use uint8_t instead of uint32_t. Likewise pins of width 9-16 will use uint16_t instead of uint32_t.

--pipe-filter <command>
Rarely needed. Verilator will spawn the specified command as a subprocess pipe, to allow the command to perform custom edits on the Verilog code before it reaches Verilator.

Before reading each Verilog file, Verilator will pass the file name to the subprocess' stdin with `read "<filename>"`. The filter may then read the file and perform any filtering it desires, and feeds the new file contents back to Verilator on stdout by first emitting a line defining the length in bytes of the filtered output `Content-Length: <bytes>`, followed by the new filtered contents. Output to stderr from the filter feeds through to Verilator's stdout and if the filter exits with non-zero status Verilator terminates. See the file:*t/t_pipe_filter* test for an example.

To debug the output of the filter, try using the *-E* option to see preprocessed output.

--pp-comments
With *-E*, show comments in preprocessor output.

--prefix <topname>
Specifies the name of the top level class and makefile. Defaults to V prepended to the name of the *--top* option, or V prepended to the first Verilog filename passed on the command line.

--prof-c
When compiling the C++ code, enable the compiler's profiling flag (e.g. `g++ -pg`). See *Code Profiling*.

Using *--prof-cfuncs* also enables *--prof-c*.

--prof-cfuncs
Modify the created C++ functions to support profiling. The functions will be minimized to contain one "basic" statement, generally a single always block or wire statement. (Note this will slow down the executable by ~5%.) Furthermore, the function name will be suffixed with the basename of the Verilog module and line number the statement came from. This allows gprof or oprofile reports to be correlated with the original Verilog source statements. See *Code Profiling*.

Using *--prof-cfuncs* also enables *--prof-c*.

--prof-threads
Enable gantt chart data collection for threaded builds. See *Thread Profiling* and *Thread Profile-Guided Optimization*.

--protect-key <key>
Specifies the private key for *--protect-ids*. For best security this key should be 16 or more random bytes, a reasonable secure choice is the output of **verilator --generate-key** . Typically, a key would be created by the user once for a given protected design library, then every Verilator run for subsequent versions of that library would be passed the same *--protect-key*. Thus, if the input Verilog is similar between library versions (Verilator runs), the Verilated code will likewise be mostly similar.

If *--protect-key* is not specified and a key is needed, Verilator will generate a new key for every Verilator run. As the key is not saved, this is best for security, but means every Verilator run will give vastly different output even for identical input, perhaps harming compile times (and certainly thrashing any "ccache").

--protect-ids
Hash any private identifiers (variable, module, and assertion block names that are not on the top level) into hashed random-looking identifiers, resulting after compilation in protected library binaries that expose less design information. This hashing uses the provided or default *--protect-key*, see important details there.

Verilator will also create a <prefix>__idmap.xml file which contains the mapping from the hashed identifiers back to the original identifiers. This idmap file is to be kept private, and is to assist mapping any simulation runtime design assertions, coverage, or trace information, which will report the hashed identifiers, back to the original design's identifier names.

Using DPI imports/exports is allowed and generally relatively safe in terms of information disclosed, which is limited to the DPI function prototyptes. Use of the VPI is not recommended as many design details may be exposed, and an INSECURE warning will be issued.

--protect-lib <name>
Produces a DPI library similar to *--lib-create*, but hides internal design details. *--protect-lib* implies *--protect-ids*, and *--lib-create*.

This allows for the secure delivery of sensitive IP without the need for encrypted RTL (i.e. IEEE P1735). See examples/make_protect_lib in the distribution for a demonstration of how to build and use the DPI library.

--private
Opposite of *--public*. Is the default; this option exists for backwards compatibility.

--public
This is only for historical debug use. Using it may result in mis-simulation of generated clocks.

Declares all signals and modules public. This will turn off signal optimizations as if all signals had a */*verilator&32;public*/* metacomments and inlining. This will also turn off inlining as if all modules had a */*verilator&32;public_module*/*, unless the module specifically enabled it with */*verilator&32;inline_module*/*.

--public-flat-rw
Declares all variables, ports and wires public as if they had /*verilator public_flat_rw @(<variable's_source_process_edge>)*/ metacomments. This will make them VPI accessible by their flat name, but not turn off module inlining. This is particularly useful in combination with *--vpi*. This may also in some rare cases result in mis-simulation of generated clocks. Instead of this global option, marking only those signals that need public_flat_rw is typically significantly better performing.

-pvalue+<name>=<value>
Overwrites the given parameter(s) of the toplevel module. See *-G* for a detailed description.

--quiet-exit
When exiting due to an error, do not display the "Exiting due to Errors" nor "Command Failed" messages.

--relative-includes
When a file references an include file, resolve the filename relative to the path of the referencing file, instead of relative to the current directory.

--reloop-limit
Rarely needed. Verilator attempts to turn some common sequences of statements into loops in the output. This argument specifies the minimum number of iterations the resulting loop needs to have in order to perform this transformation. Default limit is 40. A smaller number may slightly improve C++ compilation time on designs where these sequences are common, however effect on model performance requires benchmarking.

--report-unoptflat
Extra diagnostics for UNOPTFLAT warnings. This includes for each loop, the 10 widest variables in the loop, and the 10 most fanned out variables in the loop. These are candidates for splitting into multiple variables to break the loop.

In addition produces a GraphViz DOT file of the entire strongly connected components within the source associated with each loop. This is produced irrespective of whether *--dump-tree* is set. Such graphs may help in analyzing the problem, but can be very large indeed.

Various commands exist for viewing and manipulating DOT files. For example the "dot" command can be used to convert a DOT file to a PDF for printing. For example:

```
dot -Tpdf -O Vt_unoptflat_simple_2_35_unoptflat.dot
```

will generate a PDF `Vt_unoptflat_simple_2_35_unoptflat.dot.pdf` from the DOT file.

As an alternative, the **xdot** command can be used to view DOT files interactively:

```
xdot Vt_unoptflat_simple_2_35_unoptflat.dot
```

--rr
Run Verilator and record with the **rr** command. See: rr-project.org.

--savable
Enable including save and restore functions in the generated model. See *Save/Restore*.

--sc
Specifies SystemC output mode; see also *--cc* option.

--stats
Creates a dump file with statistics on the design in `<prefix>__stats.txt`.

--stats-vars
Creates more detailed statistics, including a list of all the variables by size (plain *--stats* just gives a count). See *--stats*, which is implied by this.

--structs-packed
Converts all unpacked structures to packed structures and issues a UNPACKED warning. Currently this is the default and *--no-structs-packed* will not work. Specifying this option allows for forward compatibility when a future version of Verilator no longer always packs unpacked structures.

-sv
Specifies SystemVerilog language features should be enabled; equivalent to *--language 1800-2017*. This option is selected by default, it exists for compatibility with other simulators.

+systemverilogext+<ext>
A synonym for *+1800-2017ext+<ext>*.

--threads `<threads>`

--no-threads
With "–threads 0" or "–no-threads", the default, the generated model is not thread safe. With "–threads 1", the generated model is single threaded but may run in a multithreaded environment. With "–threads N", where N >= 2, the model is generated to run multithreaded on up to N threads. See *Multithreading*.

--threads-dpi `all`

--threads-dpi `none`

--threads-dpi `pure`
When using *--threads*, controls which DPI imported tasks and functions are considered thread safe.

With "–threads-dpi all", Enable Verilator to assume all DPI imports are threadsafe, and to use thread-local storage for communication with DPI, potentially improving performance. Any DPI libraries need appropriate mutexes to avoid undefined behavior.

With "–threads-dpi none", Verilator assume DPI imports are not thread safe, and Verilator will serialize calls to DPI imports by default, potentially harming performance.

With "–threads-dpi pure", the default, Verilator assumes DPI pure imports are threadsafe, but non-pure DPI imports are not.

See also *--instr-count-dpi* option.

--threads-max-mtasks <value>
Rarely needed. When using *--threads*, specify the number of mtasks the model is to be partitioned into. If unspecified, Verilator approximates a good value.

--timescale <timeunit>/<timeprecision>
Sets default timescale, timeunit and timeprecision for when "*timescale" does not occur before a given module. Default is "1ps/1ps" (to match SystemC). This is overridden by :vlopt:*–timescale-override`.

--timescale-override <timeunit>/<timeprecision>

--timescale-override /<timeprecision>
Overrides all "`timescale"s in sources. The timeunit may be left empty to specify only to override the timeprecision, e.g. "/1fs".

The time precision must be consistent with SystemC's "sc_set_time_resolution()", or the C++ code instantiating the Verilated module. As "1fs" is the finest time precision it may be desirable to always use a precision of "1fs".

--top <topname>

--top-module <topname>
When the input Verilog contains more than one top level module, specifies the name of the Verilog module to become the top level module, and sets the default for *--prefix* if not explicitly specified. This is not needed with standard designs with only one top. See also *MULTITOP* warning.

--trace
Adds waveform tracing code to the model using VCD format. This overrides *--trace-fst*.

Verilator will generate additional <prefix>__Trace*.cpp files that will need to be compiled. In addition verilated_vcd_sc.cpp (for SystemC traces) or verilated_vcd_c.cpp (for both) must be compiled and linked in. If using the Verilator generated Makefiles, these files will be added to the source file lists for you. If you are not using the Verilator Makefiles, you will need to add these to your Makefile manually.

Having tracing compiled in may result in some small performance losses, even when tracing is not turned on during model execution.

See also *--trace-threads* option.

--trace-coverage
With *--trace* and --coverage-*, enable tracing to include a traced signal for every *--coverage-line* or *--coverage-user*-inserted coverage point, to assist in debugging coverage items. Note *--coverage-toggle* does not get additional signals added, as the original signals being toggle-analyzed are already visible.

The added signal will be a 32-bit value which will increment on each coverage occurrence. Due to this, this option may greatly increase trace file sizes and reduce simulation speed.

--trace-depth <levels>
Specify the number of levels deep to enable tracing, for example *--trace-depth 1* to only see the top level's signals. Defaults to the entire model. Using a small number will decrease visibility, but greatly improve simulation performance and trace file size.

--trace-fst
Enable FST waveform tracing in the model. This overrides *--trace*. See also *--trace-threads* option.

--trace-max-array *depth*
Rarely needed. Specify the maximum array depth of a signal that may be traced. Defaults to 32, as tracing large arrays may greatly slow traced simulations.

--trace-max-width *width*
Rarely needed. Specify the maximum bit width of a signal that may be traced. Defaults to 256, as tracing large vectors may greatly slow traced simulations.

--no-trace-params
Disable tracing of parameters.

--trace-structs
Enable tracing to show the name of packed structure, union, and packed array fields, rather than a single combined packed bus. Due to VCD file format constraints this may result in significantly slower trace times and larger trace files.

--trace-threads *threads*
Enable waveform tracing using separate threads. This is typically faster in simulation runtime but uses more total compute. This option is independent of, and works with, both *--trace* and *--trace-fst*. Different trace formats can take advantage of more trace threads to varying degrees. Currently VCD tracing can utilize at most "–trace-threads 1", and FST tracing can utilize at most "–trace-threads 2". This overrides *--no-threads* .

--trace-underscore
Enable tracing of signals or modules that start with an underscore. Normally, these signals are not output during tracing. See also *--coverage-underscore* option.

-U<var>
Undefines the given preprocessor symbol.

--unroll-count <loops>
Rarely needed. Specifies the maximum number of loop iterations that may be unrolled. See also *BLKLOOPINIT* warning.

--unroll-stmts *statements*
Rarely needed. Specifies the maximum number of statements in a loop for that loop to be unrolled. See also *BLKLOOPINIT* warning.

--unused-regexp *regexp*
Rarely needed. Specifies a simple regexp with * and ? that if a signal name matches will suppress the UNUSED warning. Defaults to "*unused*". Setting it to "" disables matching.

-V
Shows the verbose version, including configuration information compiled into Verilator. (Similar to **perl -V**.) See also *--getenv* option.

-v *filename*
Read the filename as a Verilog library. Any modules in the file may be used to resolve instances in the top level module, else ignored. Note "-v" is fairly standard across Verilog tools.

--no-verilate
When using *--build*, disable generation of C++/SystemC code, and execute only the build. This can be useful for rebuilding Verilated code produced by a previous invocation of Verilator.

+verilog1995ext+<ext>
Synonym for *+1364-1995ext+<ext>*.

+verilog2001ext+<ext>
Synonym for *+1364-2001ext+<ext>*.

--version
Displays program version and exits.

--vpi
Enable use of VPI and linking against the `verilated_vpi.cpp` files.

--waiver-output *filename*
: Generate a waiver file which contains all waiver statements to suppress the warnings emitted during this Verilator run. This in particular is useful as a starting point for solving linter warnings or suppressing them systematically.

  The generated file is in the Verilator Configuration format, see *Configuration Files*, and can directly be consumed by Verilator. The standard file extension is ".vlt".

-Wall
: Enable all code style warnings, including code style warnings that are normally disabled by default. Equivalent to *-Wwarn-lint -Wwarn-style*. Excludes some specialty warnings, i.e. IMPERFECTSCH.

-Werror-<message>
: Promote the specified warning message into an error message. This is generally to discourage users from violating important site-wide rules, for example "-Werror-NOUNOPTFLAT".

-Wfuture-<message>
: Rarely needed. Suppress unknown Verilator comments or warning messages with the given message code. This is used to allow code written with pragmas for a later version of Verilator to run under a older version; add "-Wfuture-" arguments for each message code or comment that the new version supports which the older version does not support.

-Wno-<message>
: Disable the specified warning/error message. This will override any lint_on directives in the source, i.e. the warning will still not be printed.

-Wno-context
: Disable showing the suspected context of the warning message by quoting the source text at the suspected location. This can be used to appease tools which process the warning messages but may get confused by lines from the original source.

-Wno-fatal
: When warnings are detected, print them, but do not terminate Verilator.

  Having warning messages in builds can be sloppy. It is recommended you cleanup your code, use inline lint_off, or use `-Wno-...` options rather than using this option.

-Wno-lint
: Disable all lint related warning messages, and all style warnings. This is equivalent to `-Wno-ALWCOMBORDER -Wno-BSSPACE -Wno-CASEINCOMPLETE -Wno-CASEOVERLAP -Wno-CASEX -Wno-CASTCONST -Wno-CASEWITHX -Wno-CMPCONST -Wno-COLONPLUS -Wno-ENDLABEL -Wno-IMPLICIT -Wno-LITENDIAN -Wno-PINCONNECTEMPTY -Wno-PINMISSING -Wno-SYNCASYNCNET -Wno-UNDRIVEN -Wno-UNSIGNED -Wno-UNUSED -Wno-WIDTH` plus the list shown for Wno-style.

  It is strongly recommended you cleanup your code rather than using this option, it is only intended to be use when running test-cases of code received from third parties.

-Wno-style
: Disable all code style related warning messages (note by default they are already disabled). This is equivalent to `-Wno-DECLFILENAME -Wno-DEFPARAM -Wno-EOFNEWLINE -Wno-IMPORTSTAR -Wno-INCABSPATH -Wno-PINCONNECTEMPTY -Wno-PINNOCONNECT -Wno-SYNCASYNCNET -Wno-UNDRIVEN -Wno-UNUSED -Wno-VARHIDDEN`.

-Wpedantic
: Warn on any construct demanded by IEEE, and disable all Verilator extensions that may interfere with IEEE compliance to the standard defined with *--default-language* (etc). Similar to **gcc -Wpedantic**. Rarely used, and intended only for strict compliance tests.

-Wwarn-<message>
: Enables the specified warning message.

-Wwarn-lint
Enable all lint related warning messages (note by default they are already enabled), but do not affect style messages. This is equivalent to `-Wwarn-ALWCOMBORDER -Wwarn-BSSPACE -Wwarn-CASEINCOMPLETE -Wwarn-CASEOVERLAP -Wwarn-CASEX -Wwarn-CASTCONST -Wwarn-CASEWITHX -Wwarn-CMPCONST -Wwarn-COLONPLUS -Wwarn-ENDLABEL -Wwarn-IMPLICIT -Wwarn-LITENDIAN -Wwarn-PINMISSING -Wwarn-REALCVT -Wwarn-UNSIGNED -Wwarn-WIDTH`.

-Wwarn-style
Enable all code style related warning messages. This is equivalent to `-Wwarn ASSIGNDLY -Wwarn-DECLFILENAME -Wwarn-DEFPARAM -Wwarn-EOFNEWLINE -Wwarn-INCABSPATH -Wwarn-PINNOCONNECT -Wwarn-SYNCASYNCNET -Wwarn-UNDRIVEN -Wwarn-UNUSED -Wwarn-VARHIDDEN`.

--x-assign `0`

--x-assign `1`

--x-assign `fast (default)`

--x-assign `unique`
Controls the two-state value that is substituted when an explicit X value is encountered in the source. "–x-assign fast", the default, converts all Xs to whatever is best for performance. "–x-assign 0" converts all Xs to 0s, and is also fast. "–x-assign 1" converts all Xs to 1s, this is nearly as fast as 0, but more likely to find reset bugs as active high logic will fire. Using "–x-assign unique" will result in all explicit Xs being replaced by a constant value determined at runtime. The value is determined by calling a function at initialization time. This enables randomization of Xs with different seeds on different executions. This method is the slowest, but safest for finding reset bugs.

If using "–x-assign unique", you may want to seed your random number generator such that each regression run gets a different randomization sequence. The simplest is to use the *+verilator+seed+<value>* runtime option. Alternatively use the system's `srand48()` or for Windows `srand()` function to do this. You'll probably also want to print any seeds selected, and code to enable rerunning with that same seed so you can reproduce bugs.

Note: This option applies only to values which are explicitly written as X in the Verilog source code. Initial values of clocks are set to 0 unless *–x-initial-edge* is specified. Initial values of all other state holding variables are controlled with *–x-initial*.

--x-initial `0`

--x-initial `fast`

--x-initial `unique (default)`
Controls the two-state value that is used to initialize variables that are not otherwise initialized.

"–x-initial 0", initializes all otherwise uninitialized variables to zero.

"–x-initial unique", the default, initializes variables using a function, which determines the value to use each initialization. This gives greatest flexibility and allows finding reset bugs. See *Unknown States*.

"–x-initial fast", is best for performance, and initializes all variables to a state Verilator determines is optimal. This may allow further code optimizations, but will likely hide any code bugs relating to missing resets.

Note: This option applies only to initial values of variables. Initial values of clocks are set to 0 unless *--x-initial-edge* is specified.

--x-initial-edge
Enables emulation of event driven simulators which generally trigger an edge on a transition from X to 1 (posedge) or X to 0 (negedge). Thus the following code, where `rst_n` is uninitialized would set `res_n` to `1'b1` when `rst_n` is first set to zero:

```
reg  res_n = 1'b0;

always @(negedge rst_n) begin
   if (rst_n == 1'b0) begin
      res_n <= 1'b1;
   end
end
```

In Verilator, by default, uninitialized clocks are given a value of zero, so the above `always` block would not trigger.

While it is not good practice, there are some designs that rely on X rarr 0 triggering a negedge, particularly in reset sequences. Using *--x-initial-edge* with Verilator will replicate this behavior. It will also ensure that X rarr 1 triggers a posedge.

Note: Using this option can affect convergence, andit may be necessary to use *--converge-limit* to increase the number of convergence iterations. This may be another indication of problems with the modeled design that should be addressed.

--xml-only
Create XML output only, do not create any other output.

The XML format is intended to be used to leverage Verilator's parser and elaboration to feed to other downstream tools. Be aware that the XML format is still evolving; there will be some changes in future versions.

--xml-output <filename>
Filename for XML output file. Using this option automatically sets *--xml-only*.

-y <dir>
Add the directory to the list of directories that should be searched for include files or libraries. The three flags *-y*, *+incdir+<dir>* and *-I<dir>* have similar effect; *+incdir+<dir>* and *-y* are fairly standard across Verilog tools while *-I<dir>* is used by many C++ compilers.

Verilator defaults to the current directory "-y ." and any specified *--Mdir*, though these default paths are used after any user specified directories. This allows '-y "$(pwd)"' to be used if absolute filenames are desired for error messages instead of relative filenames.

11.2 Configuration Files

In addition to the command line, warnings and other features for the **verilator** command may be controlled with configuration files, typically named with the .vlt extension (what makes it a configuration file is the *`verilator_config* directive). An example:

```
`verilator_config
lint_off -rule WIDTH
lint_off -rule CASEX  -file "silly_vendor_code.v"
```

This disables WIDTH warnings globally, and CASEX for a specific file.

Configuration files are fed through the normal Verilog preprocessor prior to parsing, so "`ifdef", "`define", and comments may be used as if the configuration file was normal Verilog code.

Note that file or line-specific configuration only applies to files read after the configuration file. It is therefore recommended to pass the configuration file to Verilator as the first file.

The grammar of configuration commands is as follows:

`verilator_config
Take remaining text and treat it as Verilator configuration commands.

coverage_on [-file "<filename>" [-lines <line> [- <line>]]]

coverage_off [-file "<filename>" [-lines <line> [- <line>]]]
Enable/disable coverage for the specified filename (or wildcard with '*' or '?', or all files if omitted) and range of line numbers (or all lines if omitted). Often used to ignore an entire module for coverage analysis purposes.

clock_enable -module "<modulename>" -var "<signame>"
Indicate the signal is used to gate a clock, and the user takes responsibility for insuring there are no races related to it.

Same as */*verilator&32;clock_enable*/* metacomment.

clocker -module "<modulename>" [-task "<taskname>"] -var "<signame>"

clocker -module "<modulename>" [-function "<funcname>"] -var "<signame>"

no_clocker -module "<modulename>" [-task "<taskname>"] -var "<signame>"

no_clocker -module "<modulename>" [-function "<funcname>"] -var "<signame>"
Indicates that the signal is used as clock or not. This information is used by Verilator to mark the signal and any derrived signals as clocker. See *--clk*.

Same as */*verilator&32;clocker*/* metacomment.

coverage_block_off -module "<modulename>" -block "<blockname>"

coverage_block_off -file "<filename>" -line <lineno>
Specifies the entire begin/end block should be ignored for coverage analysis purposes. Can either be specified as a named block or as a filename and line number.

Same as */*verilator&32;coverage_block_off*/* metacomment.

full_case -file "<filename>" -lines <lineno>

parallel_case -file "<filename>" -lines <lineno>
Same as `//synopsys full_case` and `//synopsys parallel_case`. When these synthesis directives are discovered, Verilator will either formally prove the directive to be true, or failing that, will insert the appropriate code to detect failing cases at simulation runtime and print an "Assertion failed" error message.

hier_block -module "<modulename>"
Specifies that the module is a unit of hierarchical Verilation. Note that the setting is ignored unless the *--hierarchical* option is specified. See *Hierarchical Verilation*.

inline -module "<modulename>"
Specifies the module may be inlined into any modules that use this module. Same as */*verilator&32;inline_module*/* metacomment.

isolate_assignments -module "<modulename>" [-task "<taskname>"] -var "<signame>"

isolate_assignments -module "<modulename>" [-function "<funcname>"] -var "<signame>"

isolate_assignments -module "<modulename>" -function "<fname>"
Used to indicate the assignments to this signal in any blocks should be isolated into new blocks. When there is a large combinatorial block that is resulting in an UNOPTFLAT warning, attaching this to the signal causing a false loop may clear up the problem.

Same as */*verilator&32;isolate_assignments*/* metacomment.

no_inline -module "<modulename>"
: Specifies the module should not be inlined into any modules that use this module. Same as */*verilator&32;no_inline_module*/* metacomment.

no_inline [-module "<modulename>"] -task "<taskname>"

no_inline [-module "<modulename>"] -function "<funcname>"
: Specify the function or task should not be inlined into where it is used. This may reduce the size of the final executable when a task is used a very large number of times. For this flag to work, the task and tasks below it must be pure; they cannot reference any variables outside the task itself.

  Same as */*verilator&32;no_inline_task*/* metacomment.

lint_on [-rule <message>] [-file "<filename>" [-lines <line> [- <line>]]]

lint_off [-rule <message>] [-file "<filename>" [-lines <line> [- <line>]]]

lint_off [-rule <message>] [-file "<filename>"] [-match "<string>"]
: Enable/disables the specified lint warning, in the specified filename (or wildcard with '*' or '?', or all files if omitted) and range of line numbers (or all lines if omitted).

  With lint_off using "*" will override any lint_on directives in the source, i.e. the warning will still not be printed.

  If the -rule is omitted, all lint warnings (see list in *-Wno-lint*) are enabled/disabled. This will override all later lint warning enables for the specified region.

  If -match is set the linter warnings are matched against this (wildcard) string and are waived in case they match and iff rule and file (with wildcard) also match.

  In previous versions -rule was named -msg. The latter is deprecated, but still works with a deprecation info, it may be removed in future versions.

public [-module "<modulename>"] [-task/-function "<taskname>"] -var "<signame>"

public_flat [-module "<modulename>"] [-task/-function "<taskname>"] -var "<signame>"

public_flat_rd [-module "<modulename>"] [-task/-function "<taskname>"] -var "<signame>"

public_flat_rw [-module "<modulename>"] [-task/-function "<taskname>"] -var "<signame>" "(
: Sets the variable to be public. Same as */*verilator&32;public*/* or */*verilator&32;public_flat*/*, etc, metacomments. See e.g. *VPI Example*.

profile_data -mtask "<mtask_hash>" -cost <cost_value>
: Feeds profile-guided optimization data into the Verilator algorithms in order to improve model runtime performance. This option is not expected to be used by users directly. See *Thread Profile-Guided Optimization*.

sc_bv -module "<modulename>" [-task "<taskname>"] -var "<signame>"

sc_bv -module "<modulename>" [-function "<funcname>"] -var "<signame>"
: Sets the port to be of sc_bv<{width}> type, instead of bool, vluint32_t or vluint64_t. Same as */*verilator&32;sc_bv*/* metacomment.

sformat [-module "<modulename>"] [-task "<taskname>"] -var "<signame>"

sformat [-module "<modulename>"] [-function "<funcname>"] -var "<signame>"
: Must be applied to the final argument of type `input string` of a function or task to indicate the function or task should pass all remaining arguments through $sformatf. This allows creation of DPI functions with $display like behavior. See the `test_regress/t/t_dpi_display.v` file for an example.

  Same as */*verilator&32;sformat*/* metacomment.

split_var [-module "<modulename>"] [-task "<taskname>"] -var "<varname>"

split_var [-module "<modulename>"] [-function "<funcname>"] -var "<varname>"
Break the variable into multiple pieces typically to resolve UNOPTFLAT performance issues. Typically the variables to attach this to are recommended by Verilator itself, see *UNOPTFLAT*.

Same as */*verilator&32;split_var*/* metacomment.

tracing_on [-file "<filename>" [-lines <line> [- <line>]]]

tracing_off [-file "<filename>" [-lines <line> [- <line>]]]
Enable/disable waveform tracing for all future signals declared in the specified filename (or wildcard with '*' or '?', or all files if omitted) and range of line numbers (or all lines if omitted).

For tracing_off, instances below any module in the files/ranges specified will also not be traced.

11.3 verilator_coverage

Verilator_coverage processes Verilated model-generated coverage reports.

With –annotate, it reads the specified coverage data file and generates annotated source code with coverage metrics annotated. If multiple coverage points exist on the same source code line, additional lines will be inserted to report the additional points.

Additional Verilog-XL-style standard arguments specify the search paths necessary to find the source code that the coverage analysis was performed on.

To filter those items to be included in coverage, you may read logs/coverage.dat into an editor and do a M-x keep-lines to include only those statistics of interest and save to a new .dat file.

For Verilog conditions that should never occur, either add a $stop statement to the appropriate statement block, or see */*verilator&32;coverage_off*/*. This will remove the coverage points after the model is re-Verilated.

For an overview of use of verilator_coverage, see *Coverage Analysis*.

11.3.1 verilator_coverage Example Usage

verilator_coverage –help verilator_coverage –version

verilator_coverage –annotate <obj>

verilator_coverage -write merged.dat -read <datafiles>...

verilator_coverage -write-info merged.info -read <datafiles>...

11.3.2 verilator_coverage Arguments

<filename>

Specifies the input coverage data file. Multiple filenames may be provided to read multiple inputs. If no data file is specified, by default "coverage.dat" will be read.

--annotate <output_directory>

Specifies the directory name that source files with annotated coverage data should be written to.

--annotate-all

Specifies all files should be shown. By default, only those source files which have low coverage are written to the output directory.

--annotate-min <count>

Specifies if the coverage point does not include the count number of coverage hits, then the coverage point will be considered above the threashold, and the coverage report will put a "%" to indicate the coverage is not sufficient. Defaults to 10.

--help

Displays a help summary, the program version, and exits.

--rank

Prints an experimental report listing the relative importance of each test in covering all of the coverage points. The report shows "Covered" which indicates the number of points that test covers; a test is considered to cover a point if it has a bucket count of at least 1. The "rank" column has a higher number t indicate the test is more important, and rank 0 means the test does not need to be run to cover the points. "RankPts" indicates the number of coverage points this test will contribute to overall coverage if all tests are run in the order of highest to lowest rank.

--unlink

With *--write*, unlink all input files after the output has been successfully created.

--version

Displays program version and exits.

--write <filename>

Specifies the aggregate coverage results, summed across all the files, should be written to the given filename in verilator_coverage data format. This is useful for use in scripts to combine many coverage data files (likely generated from random test runs) into one master coverage file.

--write-info <filename.info>

Specifies the aggregate coverage results, summed across all the files, should be written to the given filename in **lcov** .info format. This may be used to feed into **lcov** to aggregate or generate reports.

The info format loses data compared to the Verilator coverage data format; the info will all forms of coverage converted to line style coverage, and if there are multiple coverage points on a single line, the minimum coverage across those points will be used to report coverage of the line.

11.4 verilator_gantt

Verilator_gantt creates a visual representation to help analyze Verilator multithreaded simulation performance, by showing when each macro-task starts and ends, and showing when each thread is busy or idle.

For an overview of use of verilator_gantt, see *Code Profiling*.

11.4.1 Gantt Chart VCD

Verilated_gnatt creates a value change dump (VCD) format dump file which may be viewed in a waveform viewer (e.g. C<GTKWave>):

The viewed waveform chart has time on the X-axis, with one unit for each time tick of the system's high-performance counter.

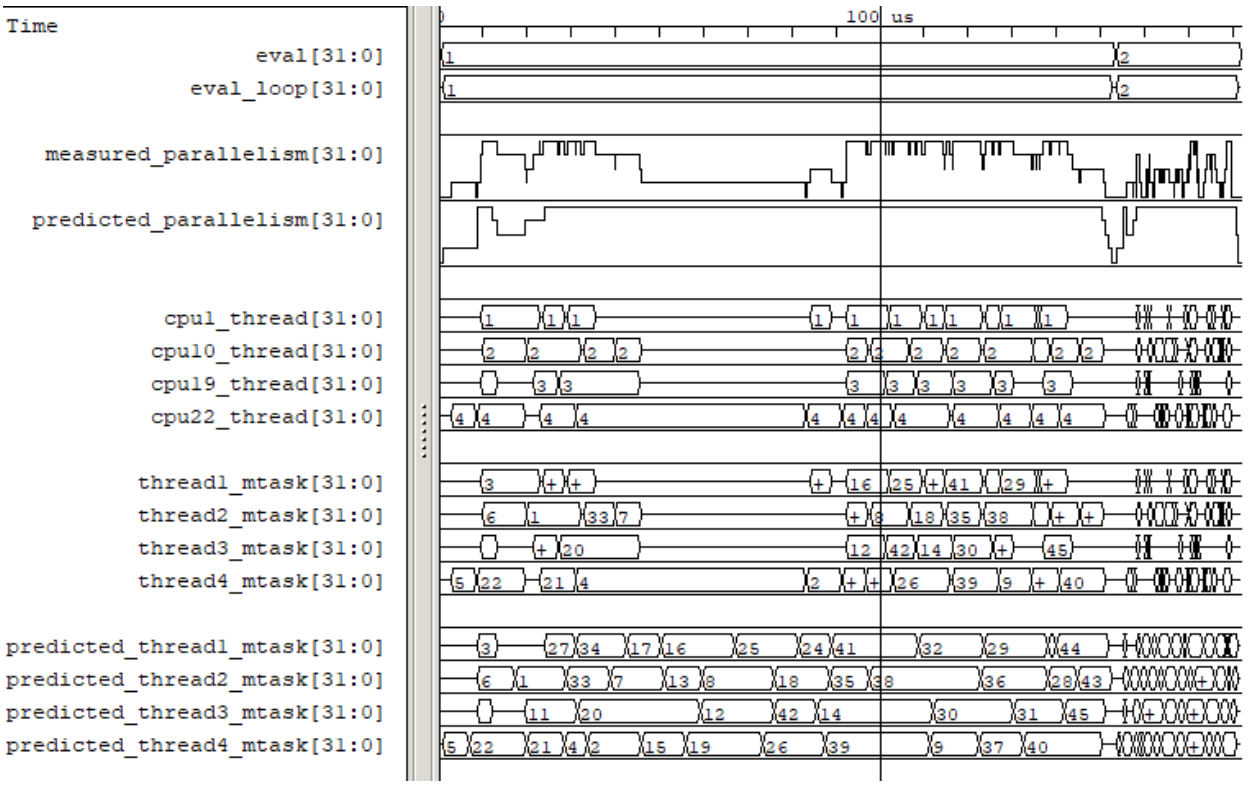

Fig. 11.1: Example verilator_gantt output, as viewed with GTKWave.

11.4.2 Gantt Chart VCD Signals

In waveforms there are the following signals. In GTKWave, using a data format of "decimal" will remove the leading zeros and make the traces easier to read.

evals Increments each time when eval_step was measured to be active. This allow visualization of how much time eval_step was active.

eval_loop Increments each time when the evaluation loop within eval_step was measured to be active. For best performance there is only a single evaluation loop within each eval_step call, that is the eval_loop waveform looks identical to the evals waveform.

measured_parallelism The number of mtasks active at this time, for best performance this will match the thread count. In GTKWave, use a data format of "analog step" to view this signal.

predicted_parallelism The number of mtasks Verilator predicted would be active at this time, for best performance this will match the thread count. In GTKWave, use a data format of "analog step" to view this signal.

cpu#_thread For the given CPU number, the thread number measured to be executing.

mtask#_cpu For the given mtask id, the CPU it was measured to execute on.

thread#_mtask For the given thread number, the mtask id it was executing.

predicted_thread#_mtask For the given thread number, the mtask id Verilator predicted would be executing.

11.4.3 verilator_gantt Arguments

<filename>

The filename to read data from, defaults to "profile_threads.dat".

--help

Displays a help summary, the program version, and exits.

--no-vcd

Disables creating a .vcd file.

--vcd <filename>

Sets the output filename for vcd dump. Default is "verilator_gantt.vcd".

11.5 verilator_profcfunc

Verilator_profcfunc reads a profile report created by gprof. The names of the functions are then transformed, assuming the user used Verilator's –prof-cfuncs, and a report printed showing the percentage of time, etc, in each Verilog block.

Due to rounding errors in gprof reports, the input report's percentages may not total to 100%. In the verilator_profcfunc report this will get reported as a rounding error.

For an overview of use of verilator_profcfunc, see *Code Profiling*.

11.5.1 verilator_profcfunc Arguments

<filename>

The **gprof**-generated filename to read data from. Typically "gprof.out".

--help

Displays a help summary, the program version, and exits.

11.6 Simulation Runtime Arguments

The following are the arguments that may be passed to a Verilated executable, provided that executable calls `Verilated::commandArgs()`.

All simulation runtime arguments begin with "+verilator", so that the user's executable may skip over all "+verilator" arguments when parsing its command line.

Summary:

```
+verilator+debug                    Enable debugging
+verilator+debugi+<value>           Enable debugging at a level
+verilator+error+limit+<value>      Set error limit
+verilator+help                     Display help
+verilator+noassert                 Disable assert checking
+verilator+prof+threads+file+<filename>  Set profile filename
+verilator+prof+threads+start+<value>    Set profile starting point
+verilator+prof+threads+window+<value>   Set profile duration
+verilator+prof+vlt+file+<filename>      Set profile guided filename
```

(continues on next page)

(continued from previous page)

```
+verilator+rand+reset+<value>     Set random reset technique
+verilator+seed+<value>           Set random seed
+verilator+V                      Verbose version and config
+verilator+version                Show version and exit
```

+verilator+debug
Enable simulation runtime debugging. Equivalent to *+verilator+debugi+4*.

+verilator+debugi+<value>
Enable simulation runtime debugging at the provided level.

+verilator+error+limit+<value>
Set number of non-fatal errors (e.g. assertion failures) before exiting simulation runtime. Also affects number of $stop calls needed before exit. Defaults to 1.

+verilator+help
Display help and exit.

+verilator+prof+threads+file+<filename>
When a model was Verilated using *--prof-threads*, sets the simulation runtime filename to dump to. Defaults to `profile_threads.dat`.

+verilator+prof+threads+start+<value>
When a model was Verilated using *--prof-threads*, the simulation runtime will wait until $time is at this value (expressed in units of the time precision), then start the profiling warmup, then capturing. Generally this should be set to some time that is well within the normal operation of the simulation, i.e. outside of reset. If 0, the dump is disabled. Defaults to 1.

+verilator+prof+threads+window+<value>
When a model was Verilated using *--prof-threads*, after $time reaches *+verilator+prof+threads+start+<value>*, Verilator will warm up the profiling for this number of eval() calls, then will capture the profiling of this number of eval() calls. Defaults to 2, which makes sense for a single-clock-domain module where it's typical to want to capture one posedge eval() and one negedge eval().

+verilator+prof+vlt+file+<filename>
When a model was Verilated using *--prof-threads*, sets the profile-guided optimization data runtime filename to dump to. Defaults to `profile.vlt`.

+verilator+rand+reset+<value>
When a model was Verilated using *--x-initial unique*, sets the simulation runtime initialization technique. 0 = Reset to zeros. 1 = Reset to all-ones. 2 = Randomize. See *Unknown States*.

+verilator+seed+<value>
For $random and *--x-initial unique*, set the simulation runtime random seed value. If zero or not specified picks a value from the system random number generator.

+verilator+noassert
Disable assert checking per runtime argument. This is the same as calling `Verilated::assertOn(false)` in the model.

+verilator+V
Shows the verbose version, including configuration information.

+verilator+version
Displays program version and exits.

CHAPTER

TWELVE

ERRORS AND WARNINGS

12.1 Disabling Warnings

Warnings may be disabled in multiple ways:

1. Disable the warning in the source code. When the warning is printed it will include a warning code. Simply surround the offending line with a `/*verilator&32;lint_off*/` and `/*verilator&32;lint_on*/` metacomment pair:

```
// verilator lint_off UNSIGNED
if (`DEF_THAT_IS_EQ_ZERO <= 3) $stop;
// verilator lint_on UNSIGNED
```

2. Disable the warning using *Configuration Files* with a `lint_off` command. This is useful when a script is suppressing warnings and the Verilog source should not be changed. This method also allows matching on the warning text.

```
lint_off -rule UNSIGNED -file "*/example.v" -line 1
```

3. Disable the warning globally invoking Verilator with the `-Wno-{warning-code}` option. This should be avoided, as it removes all checking across the designs, and prevents other users from compiling your code without knowing the magic set of disables needed to successfully compile your design.

12.2 Error And Warning Format

Warnings and errors printed by Verilator always match this regular expression:

```
%(Error|Warning)(-[A-Z0-9_]+)?: ((\S+):(\d+):((\d+):)? )?.*
```

Errors and warning start with a percent sign (historical heritage from Digital Equipment Corporation). Some errors or warning have a code attached, with meanings described below. Some errors also have a filename, line number and optional column number (starting at column 1 to match GCC).

Following the error message, Verilator will typically show the user's source code corresponding to the error, prefixed by the line number and a "|". Following this is typically an arrow and ~ pointing at the error on the source line directly above.

12.3 List Of Warnings

Internal Error
: This error should never occur first, though may occur if earlier warnings or error messages have corrupted the program. If there are no other warnings or errors, submit a bug report.

Unsupported: ....
: This error indicates that the code is using a Verilog language construct that is not yet supported in Verilator. See the Limitations chapter.

ALWCOMBORDER
: Warns that an `always_comb` block has a variable which is set after it is used. This may cause simulation-synthesis mismatches, as not all simulators allow this ordering.

```
always_comb begin
   a = b;
   b = 1;
end
```

Ignoring this warning will only suppress the lint check, it will simulate correctly.

ASSIGNDLY
: Warns that the code has an assignment statement with a delayed time in front of it, for example:

```
a <= #100 b;
assign #100 a = b;
```

Ignoring this warning may make Verilator simulations differ from other simulators, however at one point this was a common style so disabled by default as a code style warning.

ASSIGNIN
: Error that an assignment is being made to an input signal. This is almost certainly a mistake, though technically legal.

```
input a;
assign a = 1'b1;
```

Ignoring this warning will only suppress the lint check, it will simulate correctly.

BADSTDPRAGMA
: Error that a pragma is badly formed, when that pragma is defined by IEEE 1800-2017. For example, an empty pragma line, or an incorrect specified 'pragma protect'. Note that 3rd party pragmas not defined by IEEE 1800-2017 are ignored.

BLKANDNBLK
: BLKANDNBLK is an error that a variable comes from a mix of blocking and non-blocking assignments.

This is not illegal in SystemVerilog, but a violation of good coding practice. Verilator reports this as an error, because ignoring this warning may make Verilator simulations differ from other simulators.

It is generally safe to disable this error (with a `// verilator lint_off BLKANDNBLK` metacomment or the `-Wno-BLKANDNBLK` option) when one of the assignments is inside a public task, or when the blocking and non-blocking assignments have non-overlapping bits and structure members.

Generally, this is caused by a register driven by both combo logic and a flop:

```
logic [1:0] foo;
always @(posedge clk)  foo[0] <= ...
always_comb foo[1] = ...
```

Simply use a different register for the flop:

```
logic [1:0] foo;
always @(posedge clk)  foo_flopped[0] <= ...
always_comb foo[0] = foo_flopped[0];
always_comb foo[1] = ...
```

Or, this may also avoid the error:

```
logic [1:0] foo /*verilator split_var*/;
```

BLKLOOPINIT
This indicates that the initialization of an array needs to use non-delayed assignments. This is done in the interest of speed; if delayed assignments were used, the simulator would have to copy large arrays every cycle. (In smaller loops, loop unrolling allows the delayed assignment to work, though it's a bit slower than a non-delayed assignment.) Here's an example

```
always @(posedge clk)
   if (~reset_l)
       for (i=0; i<`ARRAY_SIZE; i++)
           array[i] = 0;  // Non-delayed for verilator
```

This message is only seen on large or complicated loops because Verilator generally unrolls small loops. You may want to try increasing *--unroll-count* (and occasionally *--unroll-stmts*) which will raise the small loop bar to avoid this error.

BLKSEQ
This indicates that a blocking assignment (=) is used in a sequential block. Generally non-blocking/delayed assignments (<=) are used in sequential blocks, to avoid the possibility of simulator races. It can be reasonable to do this if the generated signal is used ONLY later in the same block, however this style is generally discouraged as it is error prone.

```
always @(posedge clk)  foo = ...;  //<--- Warning
```

Disabled by default as this is a code style warning; it will simulate correctly.

Other tools with similar warnings: Verible's always-ff-non-blocking, "Use only non-blocking assignments inside 'always_ff' sequential blocks."

BSSPACE
Warns that a backslash is followed by a space then a newline. Likely the intent was to have a backslash directly followed by a newline (e.g. when making a "`define") and there's accidentally white space at the end of the line. If the space is not accidental, suggest removing the backslash in the code as it serves no function.

Ignoring this warning will only suppress the lint check, it will simulate correctly.

CASEINCOMPLETE
Warns that inside a case statement there is a stimulus pattern for which there is no case item specified. This is bad style, if a case is impossible, it's better to have a `default: $stop;` or just `default: ;` so that any design assumption violations will be discovered in simulation.

Ignoring this warning will only suppress the lint check, it will simulate correctly.

CASEOVERLAP
Warns that inside a case statement has case values which are detected to be overlapping. This is bad style, as moving the order of case values will cause different behavior. Generally the values can be respecified to not overlap.

Ignoring this warning will only suppress the lint check, it will simulate correctly.

CASEWITHX
Warns that a case statement contains a constant with a `x`. Verilator is two-state so interpret such items as always false. Note a common error is to use a `X` in a case or casez statement item; often what the user instead intended is to use a casez with `?`.

Ignoring this warning will only suppress the lint check, it will simulate correctly.

CASEX
Warns that it is simply better style to use casez, and "?" in place of "x"'s. See http://www.sunburst-design.com/papers/CummingsSNUG1999Boston_FullParallelCase_rev1_1.pdf

Ignoring this warning will only suppress the lint check, it will simulate correctly.

CASTCONST
Warns that a dynamic cast ($cast) is unnecessary as the $cast will always succeed or fail. If it will always fail, the $cast is useless. If it will always succeed a static cast may be preferred.

Ignoring this warning will only suppress the lint check, it will simulate correctly. On other simulators, not fixing CASTCONST may result in decreased performance.

CDCRSTLOGIC
With *--cdc* only, warns that asynchronous flop reset terms come from other than primary inputs or flopped outputs, creating the potential for reset glitches.

CLKDATA
Warns that clock signal is mixed used with/as data signal. The checking for this warning is enabled only if user has explicitly marked some signal as clocker using command line option or in-source meta comment (see *--clk*).

The warning can be disabled without affecting the simulation result. But it is recommended to check the warning as this may degrade the performance of the Verilated model.

CMPCONST
Warns that the code is comparing a value in a way that will always be constant. For example `X > 1` will always be true when X is a single bit wide.

Ignoring this warning will only suppress the lint check, it will simulate correctly.

COLONPLUS
Warns that a `:+` is seen. Likely the intent was to use `+:` to select a range of bits. If the intent was a range that is explicitly positive, suggest adding a space, e.g. use `:  +`.

Ignoring this warning will only suppress the lint check, it will simulate correctly.

COMBDLY
Warns that there is a delayed assignment inside of a combinatorial block. Using delayed assignments in this way is considered bad form, and may lead to the simulator not matching synthesis. If this message is suppressed, Verilator, like synthesis, will convert this to a non-delayed assignment, which may result in logic races or other nasties. See http://www.sunburst-design.com/papers/CummingsSNUG2000SJ_NBA_rev1_2.pdf

Ignoring this warning may make Verilator simulations differ from other simulators.

CONTASSREG
Error that a continuous assignment is setting a reg. According to IEEE Verilog, but not SystemVerilog, a wire must be used as the target of continuous assignments.

This error is only reported when *--language 1364-1995*, *--language 1364-2001*, or *--language 1364-2005* is used.

Ignoring this error will only suppress the lint check, it will simulate correctly.

DECLFILENAME
Warns that a module or other declaration's name doesn't match the filename with path and extension stripped

that it is declared in. The filename a modules/interfaces/programs is declared in should match the name of the module etc. so that *-y* option directory searching will work. This warning is printed for only the first mismatching module in any given file, and *-v* library files are ignored.

Disabled by default as this is a code style warning; it will simulate correctly.

DEFPARAM

Warns that the `defparam` statement was deprecated in Verilog 2001 and all designs should now be using the `#(...)` format to specify parameters.

Defparams may be defined far from the instantiation that is affected by the defparam, affecting readability. Defparams have been formally deprecated since IEEE 1800-2005 25.2 and may not work in future language versions.

Disabled by default as this is a code style warning; it will simulate correctly.

Faulty example:

```
   module parameterized
      #(parameter int MY_PARAM = 0);
   endmodule
   module upper;
     defparam p0.MY_PARAM = 1;  //<--- Warning
     parameterized p0();
   endmodule
```

Results in:

```
%Warning-DEFPARAM: example.v:5:15: defparam is deprecated (IEEE 1800-2017 C.4.1)
                                 : ... Suggest use instantiation with #(.MY_
↪PARAM(...etc...))
```

To repair use `#(.PARAMETER(...))` syntax. Repaired Example:

```
   module parameterized
      #(parameter int MY_PARAM = 0);
   endmodule
   module upper
     parameterized
        #(.MY_PARAM(1))  //<--- Repaired
        p0();
   endmodule
```

Other tools with similar warnings: Veriable's forbid_defparam_rule.

DEPRECATED

Warning that a Verilator metacomment, or configuration file command uses syntax that has been deprecated. Upgrade the code to the replacement that should be suggested by the warning message.

Ignoring this warning will only suppress the lint check, it will simulate correctly.

DETECTARRAY

Error when Verilator tries to deal with a combinatorial loop that could not be flattened, and which involves a datatype which Verilator cannot handle, such as an unpacked struct or a large unpacked array. This typically occurs when *-Wno-UNOPTFLAT* has been used to override an UNOPTFLAT warning (see below).

The solution is to break the loop, as described for UNOPTFLAT.

DIDNOTCONVERGE

Error at simulation runtime when model did not properly settle.

Verilator sometimes has to evaluate combinatorial logic multiple times, usually around code where a UNOPTFLAT warning was issued, but disabled.

Faulty example:

```
1  always_comb b = ~a;
2  always_comb a = b;
```

Results in at runtime (not when Verilated):

```
%Error: t/t_lint_didnotconverge_bad.v:7: Verilated model didn't converge
```

This is because the signals keep toggling even with out time passing. Thus to prevent an infinite loop, the Verilated executable gives the DIDNOTCONVERGE error.

To debug this, first review any UNOPT or UNOPTFLAT warnings that were ignored. Though typically it is safe to ignore UNOPTFLAT (at a performance cost), at the time of issuing a UNOPTFLAT Verilator did not know if the logic would eventually converge and assumed it would.

Next, run Verilator with *--prof-cfuncs -CFLAGS -DVL_DEBUG*. Rerun the test. Now just before the convergence error you should see additional output similar to this:

```
-V{t#,#}+    Vt_lint_didnotconverge_bad___024root___change_request
-V{t#,#}+    Vt_lint_didnotconverge_bad___024root___change_request_1
-V{t#,#}        CHANGE: t/t_lint_didnotconverge_bad.v:14: a
%Error: t/t_lint_didnotconverge_bad.v:7: Verilated model didn't converge
```

The CHANGE line means that on the given filename and line number that drove a signal, the signal 'a' kept changing. Inspect the code that modifies these signals. Note if many signals are getting printed then most likely all of them are oscillating. It may also be that e.g. "a" may be oscillating, then "a" feeds signal "c" which then is also reported as oscillating.

One way DIDNOTCONVERGE may occur is flops are built out of gate primitives. Verilator does not support building flops or latches out of gate primitives, and any such code must change to use behavioral constructs (e.g. always_ff and always_latch).

Another way DIDNOTCONVERGE may occur is if # delays are used to generate clocks. Verilator ignores the delays and gives an *ASSIGNDLY* or *STMTDLY* warning. If these were suppressed, due to the absense of the delay, the code may now oscillate.

Finally, rare, more difficult cases can be debugged like a C++ program; either enter **gdb** and use its tracing facilities, or edit the generated C++ code to add appropriate prints to see what is going on.

ENDCAPSULATED
: Warns that a class member is declared is local or protected, but is being accessed from outside that class (if local) or a derrived class (if protected).

Ignoring this warning will only suppress the lint check, it will simulate correctly.

ENDLABEL
: Warns that a label attached to a "end"-something statement does not match the label attached to the block start.

Ignoring this warning will only suppress the lint check, it will simulate correctly.

Faulty example:

```
1  module mine;
2  endmodule : not_mine  //<--- Warning
```

Results in:

```
%Warning-ENDLABEL: example.v:2:13: End label 'not_mine' does not match begin␣
↪label 'mine'
```

To repair either fix the end label's name, or remove entirely.

```
1   module mine;
2   endmodule : mine  //<--- Repaired
```

Other tools with similar warnings: Verible's mismatched-labels, "Begin/end block labels must match." or "Matching begin label is missing."

EOFNEWLINE
Warns that a file does not end in a newline. POSIX defines that a line must end in newline, as otherwise for example **cat** with the file as an argument may produce undesirable results.

Repair by appending a newline to the end of the file.

Disabled by default as this is a code style warning; it will simulate correctly.

Other tools with similar warnings: Verible's posix-eof, "File must end with a newline."

GENCLK
Deprecated and no longer used as a warning. Used to indicate that the specified signal was is generated inside the model, and also being used as a clock.

HIERBLOCK
Warns that the top module is marked as a hierarchy block by the */*verilator&32;hier_block*/* meta-comment, which is not legal. This setting on the top module will be ignored.

IFDEPTH
Warns that if/if else statements have exceeded the depth specified with *--if-depth*, as they are likely to result in slow priority encoders. Statements below unique and priority if statements are ignored. Solutions include changing the code to a case statement, or a SystemVerilog `unique if` or `priority if`.

Disabled by default as this is a code style warning; it will simulate correctly.

IGNOREDRETURN
Warns that a non-void function is being called as a task, and hence the return value is being ignored. This warning is required by IEEE.

```
1   function int function_being_called_as_task;
2      return 1;
3   endfunction
4
5   initial function_being_called_as_task();  //<--- Warning
```

Results in:

```
%Warning-IGNOREDRETURN: example.v:5:9: Ignoring return value of non-void function␣
↪(IEEE 1800-2017 13.4.1)
```

The portable way to suppress this warning (in SystemVerilog) is to use a void cast, for example:

```
1   function int function_being_called_as_task;
2      return 1;
3   endfunction
4
5   initial void'(function_being_called_as_task());  //<--- Repaired
```

Ignoring this warning will only suppress the lint check, it will simulate correctly.

IMPERFECTSCH
Warns that the scheduling of the model is not absolutely perfect, and some manual code edits may result in faster performance. This warning defaults to off, is not part of -Wall, and must be turned on explicitly before the top module statement is processed.

IMPLICIT
Warns that a wire is being implicitly declared (it is a single bit wide output from a sub-module.) While legal in Verilog, implicit declarations only work for single bit wide signals (not buses), do not allow using a signal before it is implicitly declared by an instance, and can lead to dangling nets. A better option is the `/*AUTOWIRE*/` feature of Verilog-Mode for Emacs, available from https://www.veripool.org/verilog-mode

Ignoring this warning will only suppress the lint check, it will simulate correctly.

Other tools with similar warnings: Icarus Verilog's implicit, "warning: implicit definition of wire '...'".

IMPORTSTAR
Warns that an `import {package}::*` statement is in $unit scope. This causes the imported symbols to pollute the global namespace, defeating much of the purpose of having a package. Generally `import ::*` should only be used inside a lower scope such as a package or module.

Disabled by default as this is a code style warning; it will simulate correctly.

IMPURE
Warns that a task or function that has been marked with a */*verilator&32;no_inline_task*/* meta-comment, but it references variables that are not local to the task. Verilator cannot schedule these variables correctly.

Ignoring this warning may make Verilator simulations differ from other simulators.

INCABSPATH
Warns that an "`include" filename specifies an absolute path. This means the code will not work on any other system with a different file system layout. Instead of using absolute paths, relative paths (preferably without any directory specified whatsoever) should be used, and +incdir used on the command line to specify the top include source directories.

Disabled by default as this is a code style warning; it will simulate correctly.

INFINITELOOP
Warns that a while or for statement has a condition that is always true. and thus results in an infinite loop if the statement ever executes.

This might be unintended behavior if the loop body contains statements that in other simulators would make time pass, which Verilator is ignoring due to e.g. `STMTDLY` warnings being disabled.

Ignoring this warning will only suppress the lint check, it will simulate correctly (i.e. hang due to the infinite loop).

INITIALDLY
Warns that the code has a delayed assignment inside of an initial or final block. If this message is suppressed, Verilator will convert this to a non-delayed assignment. See also *COMBDLY*.

Ignoring this warning may make Verilator simulations differ from other simulators.

INSECURE
Warns that the combination of options selected may be defeating the attempt to protect/obscure identifiers or hide information in the model. Correct the options provided, or inspect the output code to see if the information exposed is acceptable.

Ignoring this warning will only suppress the lint check, it will simulate correctly.

LATCH
Warns that a signal is not assigned in all control paths of a combinational always block, resulting in the inference

of a latch. For intentional latches, consider using the always_latch (SystemVerilog) keyword instead. The warning may be disabled with a lint_off pragma around the always block.

Ignoring this warning will only suppress the lint check, it will simulate correctly.

LITENDIAN
Warns that a packed vector is declared with little endian bit numbering (i.e. [0:7]). Big endian bit numbering is now the overwhelming standard, and little numbering is now thus often due to simple oversight instead of intent.

Also warns that an instance is declared with little endian range (i.e. [0:7] or [7]) and is connected to a N-wide signal. Based on IEEE the bits will likely be backwards from what people may expect (i.e. instance [0] will connect to signal bit [N-1] not bit [0]).

Ignoring this warning will only suppress the lint check, it will simulate correctly.

MODDUP
Warns that a module has multiple definitions. Generally this indicates a coding error, or a mistake in a library file, and it's good practice to have one module per file (and only put each file once on the command line) to avoid these issues. For some gate level netlists duplicates are sometimes unavoidable, and MODDUP should be disabled.

Ignoring this warning will cause the more recent module definition to be discarded.

MULTIDRIVEN
Warns that the specified signal comes from multiple always blocks each with different clocking. This warning does not look at individual bits (see example below).

This is considered bad style, as the consumer of a given signal may be unaware of the inconsistent clocking, causing clock domain crossing or timing bugs.

Faulty example:

```
1  always @(posedge clk) begin
2      out2[7:0] <= d0;  // <--- Warning
3  end
4  always @(negedge clk) begin
5      out2[15:8] <= d0;  // <--- Warning
6  end
```

Results in:

```
%Warning-MULTIDRIVEN: example.v:1:22 Signal has multiple driving blocks with␣
↪different clocking: 'out2'
                      example.v:1:7 ... Location of first driving block
                      example.v:1:7 ... Location of other driving block
```

Ignoring this warning will only slow simulations, it will simulate correctly. It may however cause longer simulation runtimes due to reduced optimizations.

MULTITOP
Warns that there are multiple top level modules, that is modules not instantiated by any other module, and both modules were put on the command line (not in a library). Three likely cases:

1. A single module is intended to be the top. This warning then occurs because some low level instance is being read in, but is not really needed as part of the design. The best solution for this situation is to ensure that only the top module is put on the command line without any flags, and all remaining library files are read in as libraries with *-v*, or are automatically resolved by having filenames that match the module names.

2. A single module is intended to be the top, the name of it is known, and all other modules should be ignored if not part of the design. The best solution is to use the *--top* option to specify the top module's name. All other

modules that are not part of the design will be for the most part ignored (they must be clean in syntax and their contents will be removed as part of the Verilog module elaboration process.)

3. Multiple modules are intended to be design tops, e.g. when linting a library file. As multiple modules are desired, disable the MULTITOP warning. All input/outputs will go uniquely to each module, with any conflicting and identical signal names being made unique by adding a prefix based on the top module name followed by __02E (a Verilator-encoded ASCII "."). This renaming is done even if the two modules' signals seem identical, e.g. multiple modules with a "clk" input.

NOLATCH
Warns that no latch was detected in an always_latch block. The warning may be disabled with a lint_off pragma around the always block, but recoding using a regular always may be more appropriate.

Ignoring this warning will only suppress the lint check, it will simulate correctly.

NULLPORT
Warns that a null port was detected in the module definition port list. Null ports are empty placeholders, i.e. either one ore more commas at the beginning or the end of a module port list, or two or more consecutive commas in the middle of a module port list. A null port cannot be accessed within the module, but when instantiating the module by port order, it is treated like a regular port and any wire connected to it is left unconnected. For example:

```
1 module a
2     (a_named_port, );  //<--- Warning
```

This is considered a warning because null ports are rarely used, and is mostly the result of a typing error such as a dangling comma at the end of a port list.

Ignoring this warning will only suppress the lint check, it will simulate correctly.

PINCONNECTEMPTY
Warns that an instance has a pin which is connected to `.pin_name()`, e.g. not another signal, but with an explicit mention of the pin. It may be desirable to disable PINCONNECTEMPTY, as this indicates intention to have a no-connect.

Disabled by default as this is a code style warning; it will simulate correctly.

PINMISSING
Warns that a module has a pin which is not mentioned in an instance. If a pin is not missing it should still be specified on the instance declaration with a empty connection, using `(.pin_name()).`

Ignoring this warning will only suppress the lint check, it will simulate correctly.

Other tools with similar warnings: Icarus Verilog's portbind, "warning: Instantiating module ... with dangling impot port (...)". Slang's unconnected-port, "port '...' has no connection".

PINNOCONNECT
Warns that an instance has a pin which is not connected to another signal.

Disabled by default as this is a code style warning; it will simulate correctly.

PINNOTFOUND
Warns that an instance port or Parameter was not found in the module being instantiated. Note that Verilator raises these errors also on instances that should be disabled by generate/if/endgenerate constructs:

```
1 module a;
2   localparam A=1;
3   generate
4     if (A==0) begin
5       b b_inst1 (.x(1'b0));  //<--- error nonexistent port
6       b #(.PX(1'b0)) b_inst2 (); //<--- error nonexistent parameter
```

(continues on next page)

(continued from previous page)

```
      end
    endgenerate
endmodule

module b;
endmodule
```

In the example above, b is instantiated with a port named x, but module b has no such port. In the next line, b is instantiated again with a nonexistent parameter PX. Technically, this code is incorrect because of this, but other tools may ignore it because module b is not instantiated due to the generate/if condition being false.

This error may be disabled with a lint_off PINNOTFOUND metacomment.

PORTSHORT
Warns that an output port is connected to a constant.

```
module a;
  sub sub
     (.out(1'b1));  //<--- error PORTSHORT
endmodule

module sub (output out);
  assign out = '1;
endmodule
```

In the example above, out is an output but is connected to a constant implying it is an input.

This error may be disabled with a lint_off PORTSHORT metacomment.

PKGNODECL
Error that a package/class appears to have been referenced that has not yet been declared. According to IEEE 1800-2017 26.3 all packages must be declared before being used.

PROCASSWIRE
Error that a procedural assignment is setting a wire. According to IEEE, a var/reg must be used as the target of procedural assignments.

PROFOUTOFDATE
Warns that threads were scheduled using estimated costs, despite the fact that data was provided from profile-guided optimization (see *Thread Profile-Guided Optimization*) as fed into Verilator using the `profile_data` configuration file option. This usually indicates that the profile data was generated from different Verilog source code than Verilator is currently running against.

It is recommended to create new profiling data, then rerun Verilator with the same input source files and that new profiling data.

Ignoring this warning may only slow simulations, it will simulate correctly.

PROTECTED
Warning that a 'pragma protected' section was encountered. The code inside the protected region will be partly checked for correctness, but is otherwise ignored.

Suppressing the warning may make Verilator differ from a simulator that accepts the protected code.

RANDC
Warns that the `randc` keyword is currently unsupported, and that it is being converted to `rand`.

REALCVT
Warns that a real number is being implicitly rounded to an integer, with possible loss of precision.

Faulty example:

```
   int i;
   i = 2.3;  //<--- Warning
```

Results in:

```
%Warning-REALCVT: example.v:2:5: Implicit conversion of real to integer
```

If the code is correct, the portable way to suppress the warning is to add a cast. This will express the intent and should avoid future warnings on any linting tool.

```
   int i;
   i = int'(2.3);  //<--- Repaired
```

REDEFMACRO

Warns that the code has redefined the same macro with a different value, for example:

```
   `define DUP def1
   //...
   `define DUP def2  //<--- Warning
```

Results in:

```
%Warning-REDEFMACRO: example.v:3:20: Redefining existing define: 'DUP', with␣
↪different value: 'def1'
                     example.v:1:20: ... Location of previous definition, with␣
↪value: 'def2'
```

The best solution is to use a different name for the second macro. If this is not possible, add a undef to indicate the code is overriding the value. This will express the intent and should avoid future warnings on any linting tool:

```
`define DUP def1
//...
`undef DUP  //<--- Repaired
`define DUP def2
```

Other tools with similar warnings: Icarus Verilog's macro-redefinition, "warning: redefinition of macro ... from value '...' to '...'". Yosys's "Duplicate macro arguments with name".

SELRANGE

Warns that a selection index will go out of bounds.

Faulty example:

```
   wire vec[6:0];
   initial out = vec[7];  //<--- Warning (there is no [7])
```

Verilator will assume zero for this value, instead of X. Note that in some cases this warning may be false, when a condition upstream or downstream of the access means the access out of bounds will never execute or be used.

Repaired example:

```
   wire vec[6:0];
   initial begin
      index = 7;
      ...
      if (index < 7) out = vec[index];  // Never will use vec[7]
```

Other tools with similar warnings: Icarus Verilog's select-range, "warning: ... [...] is selecting before vector" or "is selecting before vector".

SHORTREAL

Warns that Verilator does not support `shortreal` and they will be automatically promoted to `real`.

```
1   shortreal sig;  //<--- Warning
```

The recommendation is to replace any `shortreal` in the code with `real`, as `shortreal` is not widely supported across industry tools.

```
1   real sig;  //<--- Repaired
```

Ignoring this warning may make Verilator simulations differ from other simulators, if the increased precision of real affects your model or DPI calls.

SPLITVAR

Warns that a variable with a */*verilator&32;split_var*/* metacomment was not split. Some possible reasons for this are:

- The datatype of the variable is not supported for splitting. (e.g. is a real).
- The access pattern of the variable can not be determined statically. (e.g. is accessed as a memory).
- The index of the array exceeds the array size.
- The variable is accessed from outside using dotted reference. (e.g. top.instance0.variable0 = 1).
- The variable is not declared in a module, but in a package or an interface.
- The variable is a parameter, localparam, genvar, or queue.
- The variable is tristate or bidirectional. (e.g. inout or ref).

STMTDLY

Warns that the code has a statement with a delayed time in front of it.

Ignoring this warning may make Verilator simulations differ from other simulators.

Faulty example:

```
      #100 $finish;  //<--- Warning
```

Results in:

```
%Warning-STMTDLY: example.v:1:8 Unsupported: Ignoring delay on this delayed␣
↪statement.
```

This is a warning because Verilator does not support delayed statements. It will simply ignore all such delays. In many cases ignoring a delay might be harmless, but if the delayed statement is, as in this example, used to cause some important action at a later time, it might be an important difference.

Some possible work arounds:

- Move the delayed statement into the C++ wrapper file, where the stimulus and clock generation can be done in C++.
- Convert the statement into a FSM, or other statement that tests against $time.

SYMRSVDWORD

Warning that a symbol matches a C++ reserved word and using this as a symbol name would result in odd C++ compiler errors. You may disable this warning, but the symbol will be renamed by Verilator to avoid the conflict.

SYNCASYNCNET

Warns that the specified net is used in at least two different always statements with posedge/negedges (i.e. a flop). One usage has the signal in the sensitivity list and body, probably as an async reset, and the other usage has the signal only in the body, probably as a sync reset. Mixing sync and async resets is usually a mistake. The warning may be disabled with a lint_off pragma around the net, or either flopped block.

Disabled by default as this is a code style warning; it will simulate correctly.

TASKNSVAR

Error when a call to a task or function has an inout from that task tied to a non-simple signal. Instead connect the task output to a temporary signal of the appropriate width, and use that signal to set the appropriate expression as the next statement. For example:

```
   task foo(inout sig); ... endtask
   // ...
   always @* begin
        foo(bus_we_select_from[2]);  // Will get TASKNSVAR error
   end
```

Change this to:

```
task foo(inout sig); ... endtask
// ...
reg foo_temp_out;
always @* begin
   foo(foo_temp_out);
   bus_we_select_from[2] = foo_temp_out;
end
```

Verilator doesn't do this conversion for you, as some more complicated cases would result in simulator mismatches.

TICKCOUNT

Warns that the number of ticks to delay a $past variable is greater than 10. At present Verilator effectively creates a flop for each delayed signals, and as such any large counts may lead to large design size increases.

Ignoring this warning will only slow simulations, it will simulate correctly.

TIMESCALEMOD

Warns that "`timescale" is used in some but not all modules.

This may be disabled similar to other warnings. Ignoring this warning may result in a module having an unexpected timescale.

IEEE recommends this be an error, for that behavior use *-Werror-TIMESCALEMOD*.

Faulty example:

```
   module mod1;
     sub sub();
   endmodule
   `timescale 1ns/1ns
   module sub;  //<--- Warning
   endmodule
```

Results in:

```
%Warning-TIMESCALEMOD: example.v:1:8: Timescale missing on this module as other␣
↪modules have it (IEEE 1800-2017 3.14.2.3)
```

Recommend using *--timescale* argument, or in front of all modules use:

```
`include "timescale.vh"
```

Then in that file set the timescale.

Other tools with similar warnings: Icarus Verilog's timescale, "warning: Some design elements have no explicit time unit and/or time precision. This may cause confusing timing results." Slang's: "[WRN:PA0205] No timescale set for "...."".

UNDRIVEN
Warns that the specified signal has no source. Verilator is fairly liberal in the usage calculations; making a signal public, or setting only a single array element marks the entire signal as driven.

Disabled by default as this is a code style warning; it will simulate correctly.

Other tools with similar warnings: Odin's "[NETLIST] This output is undriven (...) and will be removed".

UNOPT
Warns that due to some construct, optimization of the specified signal or block is disabled. The construct should be cleaned up to improve simulation performance.

A less obvious case of this is when a module instantiates two submodules. Inside submodule A, signal I is input and signal O is output. Likewise in submodule B, signal O is an input and I is an output. A loop exists and a UNOPT warning will result if AI & AO both come from and go to combinatorial blocks in both submodules, even if they are unrelated always blocks. This affects performance because Verilator would have to evaluate each submodule multiple times to stabilize the signals crossing between the modules.

Ignoring this warning will only slow simulations, it will simulate correctly.

UNOPTFLAT
Warns that due to some construct, optimization of the specified signal is disabled. The signal reported includes a complete scope to the signal; it may be only one particular usage of a multiply instantiated block. The construct should be cleaned up to improve simulation performance; two times better performance may be possible by fixing these warnings.

Unlike the `UNOPT` warning, this occurs after flattening the netlist, and indicates a more basic problem, as the less obvious case described under `UNOPT` does not apply.

Often UNOPTFLAT is caused by logic that isn't truly circular as viewed by synthesis which analyzes interconnection per-bit, but is circular to simulation which analyzes per-bus.

Faulty example:

```
wire [2:0] x = {x[1:0], shift_in};
```

This statement needs to be evaluated multiple times, as a change in `shift_in` requires "x" to be computed 3 times before it becomes stable. This is because a change in "x" requires "x" itself to change value, which causes the warning.

For significantly better performance, split this into 2 separate signals:

```
wire [2:0] xout = {x[1:0], shift_in};
```

and change all receiving logic to instead receive "xout". Alternatively, change it to:

```
wire [2:0] x = {xin[1:0], shift_in};
```

and change all driving logic to instead drive "xin".

With this change this assignment needs to be evaluated only once. These sort of changes may also speed up your traditional event driven simulator, as it will result in fewer events per cycle.

The most complicated UNOPTFLAT path we've seen was due to low bits of a bus being generated from an always statement that consumed high bits of the same bus processed by another series of always blocks. The fix is the same; split it into two separate signals generated from each block.

Occasionally UNOPTFLAT may be indicated when there is a true circulation. e.g. if trying to implement a flop or latch using individual gate primitives. If UNOPTFLAT is suppressed the code may get a DIDNOTCONVERGE error. Verilator does not support building flops or latches out of gate primitives, and any such code must change to use behavioral constructs (e.g. always_ff and always_latch).

Another way to resolve this warning is to add a */*verilator&32;split_var*/* metacomment described above. This will cause the variable to be split internally, potentially resolving the conflict. If you run with *–report-unoptflat* Verilator will suggest possible candidates for */*verilator&32;split_var*/*.

The UNOPTFLAT warning may also be due to clock enables, identified from the reported path going through a clock gating instance. To fix these, use the clock_enable meta comment described above.

The UNOPTFLAT warning may also occur where outputs from a block of logic are independent, but occur in the same always block. To fix this, use the */*verilator&32;isolate_assignments*/* metacomment described above.

To assist in resolving UNOPTFLAT, the option *--report-unoptflat* can be used, which will provide suggestions for variables that can be split up, and a graph of all the nodes connected in the loop. See the Arguments section for more details.

Ignoring this warning will only slow simulations, it will simulate correctly.

UNOPTTHREADS

Warns that the thread scheduler was unable to partition the design to fill the requested number of threads.

One workaround is to request fewer threads with *--threads*.

Another possible workaround is to allow more MTasks in the simulation runtime, by increasing the value of *--threads-max-mtasks*. More MTasks will result in more communication and synchronization overhead at simulation runtime; the scheduler attempts to minimize the number of MTasks for this reason.

Ignoring this warning will only slow simulations, it will simulate correctly.

UNPACKED

Warns that unpacked structs and unions are not supported.

Ignoring this warning will make Verilator treat the structure as packed, which may make Verilator simulations differ from other simulators. This downgrading may also result what would normally be a legal unpacked struct/array inside an unpacked struct/array becoming an illegal unpacked struct/array inside a packed struct/array.

UNSIGNED

Warns that the code is comparing a unsigned value in a way that implies it is signed, for example "X < 0" will always be false when X is unsigned.

Ignoring this warning will only suppress the lint check, it will simulate correctly.

UNSUPPORTED

Error that a construct might be legal according to IEEE but is not currently supported by Verilator.

A typical workaround is to recode the construct into a simpler and more common alternative language construct.

Alternatively, check if the construct is supported by other tools, and if so please consider submitting a github pull request against the Verilator sources to implement the missing unsupported feature.

This error may be ignored with *--bbox-unsup*, however this will make the design simulate incorrectly and is only intended for lint usage; see the details under *--bbox-unsup*.

UNUSED

Warns that the specified signal or parameter is never used/consumed. Verilator is fairly liberal in the usage calculations; making a signal public, a signal matching the *--unused-regexp* option (default "*unused*" or accessing only a single array element marks the entire signal as used.

Disabled by default as this is a code style warning; it will simulate correctly.

A recommended style for unused nets is to put at the bottom of a file code similar to the following:

```
wire _unused_ok = &{1'b0,
                    sig_not_used_a,
                    sig_not_used_yet_b,  // To be fixed
                    1'b0};
```

The reduction AND and constant zeros mean the net will always be zero, so won't use simulation runtime. The redundant leading and trailing zeros avoid syntax errors if there are no signals between them. The magic name "unused" (controlled by *--unused-regexp* option) is recognized by Verilator and suppresses warnings; if using other lint tools, either teach it to the tool to ignore signals with "unused" in the name, or put the appropriate lint_off around the wire. Having unused signals in one place makes it easy to find what is unused, and reduces the number of lint_off pragmas, reducing bugs.

USERERROR

A SystemVerilog elaboration-time assertion error was executed. IEEE 1800-2017 20.11 requires this error.

Faulty example:

```
$error("User elaboration-time error");
```

Results in:

```
%Warning-USERERROR: example.v:1:7 User elaboration-time error
```

To resolve, examine the code and rectify the cause of the error.

USERFATAL

A SystemVerilog elaboration-time assertion fatal was executed. IEEE 1800-2017 20.11 requires this error.

Faulty example:

```
$fatal(0, "User elaboration-time fatal");
```

Results in:

```
%Warning-USERFATAL: example.v:1:7 User elaboration-time fatal
```

To resolve, examine the code and rectify the cause of the fatal.

USERINFO

A SystemVerilog elaboration-time assertion print was executed. This is not an error nor warning. IEEE 1800-2017 20.11 requires this behavior.

Example:

```
$info("User elaboration-time info");
```

Results in:

```
-Info: example.v:1:7 User elaboration-time info
```

USERWARN

A SystemVerilog elaboration-time assertion warning was executed. IEEE 1800-2017 20.11 requires this warning.

Faulty example:

```
$warning("User elaboration-time warning");
```

Results in:

```
%Warning-USERWARN: example.v:1:7 User elaboration-time warning
```

To resolve, examine the code and rectify the cause of the error.

VARHIDDEN

Warns that a task, function, or begin/end block is declaring a variable by the same name as a variable in the upper level module or begin/end block (thus hiding the upper variable from being able to be used.) Rename the variable to avoid confusion when reading the code.

Disabled by default as this is a code style warning; it will simulate correctly.

Faulty example:

```
1 module t;
2    integer t;  //<--- Warning ('t' hidden by module 't')
3 endmodule
```

Results in:

```
%Warning-VARHIDDEN: example.v:2:12 Declaration of signal hides declaration in␣
↪upper scope: 't'
                    example.v:1:8 ... Location of original declaration
```

To resolve, rename the variable to a unique name.

WIDTH

Warns that based on width rules of Verilog:

- Two operands have different widths, e.g. adding a 2-bit and 5-bit number.
- A part select has a different size then needed to index into the packed or unpacked array (etc).

Verilator attempts to track the minimum width of unsized constants and will suppress the warning when the minimum width is appropriate to fit the required size.

Ignoring this warning will only suppress the lint check, it will simulate correctly.

The recommendation is to fix these issues by:

- Resizing the variable or constant to match the needed size for the expression. E.g. `2'd2` instead of `3'd2`.
- Using `'0` or `'1` which automatically resize in an expression.
- Using part selects to narrow a variable. E.g. `too_wide[1:0]`.
- Using concatenate to widen a variable. E.g. `{1'b1, too_narrow}`.
- Using cast to resize a variable. E.g. `23'(wrong_sized)`.

For example this is a missized index:

```
1    int array[5];
2    bit [1:0] rd_addr;
3    wire int rd_value = array[rd_addr];  //<--- Warning
```

Results in:

```
%Warning-WIDTH: example.v:3:29 Bit extraction of array[4:0] requires 3 bit index,␣
↪not 2 bits.
```

One possible fix:

```
   wire int rd_value = array[{1'b0, rd_addr}];  //<--- Fixed
```

WIDTHCONCAT

Warns that based on width rules of Verilog, a concatenate or replication has an indeterminate width. In most cases this violates the Verilog rule that widths inside concatenates and replicates must be sized, and should be fixed in the code.

Faulty example:

```
wire [63:0] concat = {1, 2};
```

An example where this is technically legal (though still bad form) is:

```
parameter PAR = 1;
wire [63:0] concat = {PAR, PAR};
```

The correct fix is to either size the 1 (`32'h1`), or add the width to the parameter definition (`parameter [31:0]`), or add the width to the parameter usage (`{PAR[31:0], PAR[31:0]}`).

CHAPTER

THIRTEEN

FILES

13.1 Files in the Git Tree

The following is a summary of the files in the Git Tree (distribution) of Verilator:

```
Changes                     => Version history
README.rst                  => This document
bin/verilator               => Compiler wrapper invoked to Verilate code
docs/                       => Additional documentation
examples/                   => Examples (see manual for descriptions)
include/                    => Files that should be in your -I compiler path
include/verilated*.cpp      => Global routines to link into your simulator
include/verilated*.h        => Global headers
include/verilated.mk        => Common Makefile
src/                        => Translator source code
test_regress                => Internal tests
```

13.2 Files Read/Written

All output files are placed in the output directory specified with the *--Mdir* option, or "obj_dir" if not specified.

Verilator creates the following files in the output directory:

For –cc/–sc, it creates:

{prefix}.cmake	CMake include script for compiling (from –make cmake)
{prefix}.mk	Make include file for compiling (from –make gmake)
*{prefix}*_classes.mk	Make include file with class names (from –make gmake)
*{prefix}*_hier.mk	Make file for hierarchy blocks (from –make gmake)
{prefix|_hierMkArgs.f	Arguments for hierarchical Verilation (from –make gmake)
*{prefix}*_hierCMakeArgs.f	Arguments for hierarchical Verilation (from –make cmake)
{prefix}.h	Model header
{prefix}.cpp	Model C++ file
*{prefix}*___024root.h	Top level (SystemVerilog $root) internal header file
*{prefix}*___024root.cpp	Top level (SystemVerilog $root) internal C++ file
*{prefix}*___024root*{__n}*.cpp	Additional top level internal C++ files (from –output-split)
*{prefix}*___024root__Slow*{__n}*.cpp	Infrequent cold routines
*{prefix}*___024root__Trace{__n}*.cpp	Wave file generation code (from –trace)
*{prefix}*___024root__Trace__Slow{__n}*.cpp	Wave file generation code (from –trace)
*{prefix}*__Dpi.h	DPI import and export declarations (from –dpi)
*{prefix}*__Dpi.cpp	Global DPI export wrappers (from –dpi)
*{prefix}*__Dpi_Export*{__n}*.cpp	DPI export wrappers scoped to this particular model (from –dpi)
*{prefix}*__Inlines.h	Inline support functions
*{prefix}*__Syms.h	Global symbol table header
*{prefix}*__Syms.cpp	Global symbol table C++
{prefix}{each_verilog_module}.h	Lower level internal header files
{prefix}{each_verilog_module}.cpp	Lower level internal C++ files
{prefix}{each_verilog_module}{__n}.cpp	Additional lower C++ files (from –output-split)

For –hierarchy mode, it creates:

V*{hier_block}*/	Directory to Verilate each hierarchy block (from –hierarchy)
*{prefix}*__hierVer.d	Make dependencies of the top module (from –hierarchy)
*{prefix}*__hier.dir	Directory to store .dot, .vpp, .tree of top module (from –hierarchy)

In certain debug and other modes, it also creates:

{prefix}.xml	XML tree information (from –xml)
*{prefix}*__cdc.txt	Clock Domain Crossing checks (from –cdc)
*{prefix}*__stats.txt	Statistics (from –stats)
*{prefix}*__idmap.txt	Symbol demangling (from –protect-ids)
*{prefix}*__ver.d	Make dependencies (from -MMD)
*{prefix}*__verFiles.dat	Timestamps (from –skip-identical)
{prefix}{misc}.dot	Debugging graph files (from –debug)
{prefix}{misc}.tree	Debugging files (from –debug)
{mod_prefix}_{each_verilog_module}*{__n}*.vpp	Pre-processed verilog (from –debug)

After running Make, the C++ compiler may produce the following:

verilated{misc}*.d	Intermediate dependencies
verilated{misc}*.o	Intermediate objects
{mod_prefix}{misc}*.d	Intermediate dependencies
{mod_prefix}{misc}*.o	Intermediate objects
{prefix}	Final executable (from –exe)
*{prefix}*__ALL.a	Library of all Verilated objects
*{prefix}*__ALL.cpp	Include of all code for single compile
{prefix}{misc}.d	Intermediate dependencies
{prefix}{misc}.o	Intermediate objects

The Verilated executable may produce the following:

coverage.dat	Code coverage output, and default input filename for **`verilator_coverage`**
gmon.out	GCC/clang code profiler output, often fed into **`verilator_profcfunc`**
profile.vlt	-profile data file for *Thread Profile-Guided Optimization*
profile_threads.dat	-profile-threads data file for **`verilator_gnatt`**

Verilator_gantt may produce the following:

profile_threads.vcd	Gantt report waveform output

CHAPTER

FOURTEEN

ENVIRONMENT

This section describes the environment variables used by Verilator and associated programs.

LD_LIBRARY_PATH
: A generic Linux/OS variable specifying what directories have shared object (.so) files. This path should include SystemC and any other shared objects needed at simulation runtime.

MAKE
: Names the executable of the make command invoked when using the *--build* option. Some operating systems may require "gmake" to this variable to launch GNU make. If this variable is not specified, "make" is used.

OBJCACHE
: Optionally specifies a caching or distribution program to place in front of all runs of the C++ compiler. For example, "ccache". If using **distcc** or **icecc/icecream**, they would generally be run under **ccache**; see the documentation for those programs. If OBJCACHE is not set, and at configure time ccache was present, ccache will be used as a default.

SYSTEMC
: Deprecated. Used only if *SYSTEMC_INCLUDE* or *SYSTEMC_LIBDIR* is not set. If set, specifies the directory containing the SystemC distribution. If not specified, it will come from a default optionally specified at configure time (before Verilator was compiled).

SYSTEMC_ARCH
: Deprecated. Used only if *SYSTEMC_LIBDIR* is not set. Specifies the architecture name used by the SystemC kit. This is the part after the dash in the "lib-{...}" directory name created by a **make** in the SystemC distribution. If not set, Verilator will try to intuit the proper setting, or use the default optionally specified at configure time (before Verilator was compiled).

SYSTEMC_CXX_FLAGS
: Specifies additional flags that are required to be passed to GCC when building the SystemC model. System 2.3.0 may need this set to "-pthread".

SYSTEMC_INCLUDE
: If set, specifies the directory containing the systemc.h header file. If not specified, it will come from a default optionally specified at configure time (before Verilator was compiled), or computed from SYSTEMC/include.

SYSTEMC_LIBDIR
: If set, specifies the directory containing the libsystemc.a library. If not specified, it will come from a default optionally specified at configure time (before Verilator was compiled), or computed from SYSTEMC/lib-SYSTEMC_ARCH.

VERILATOR_BIN
: If set, specifies an alternative name of the `verilator` binary. May be used for debugging and selecting between multiple operating system builds.

VERILATOR_COVERAGE_BIN
: If set, specifies an alternative name of the `verilator_coverage` binary. May be used for debugging and selecting between multiple operating system builds.

VERILATOR_GDB
: If set, the command to run when using the *--gdb* option, such as "ddd". If not specified, it will use "gdb".

VERILATOR_ROOT
: Specifies the directory containing the distribution kit. This is used to find the executable, Perl library, and include files. If not specified, it will come from a default optionally specified at configure time (before Verilator was compiled). It should not be specified if using a pre-compiled Verilator package as the hard-coded value should be correct. See *Installation*.

CHAPTER

FIFTEEN

DEPRECATIONS

The following deprecated items are scheduled for future removal:

C++11 compiler support Verilator currently requires C++11 or newer compilers. Verilator will require C++14 or newer compilers for both compiling Verilator and compiling Verilated models no sooner than January 2023.

Verilated_heavy.h The legacy "verilated_heavy.h" include was replaced with just including "verilated.h". Verilated_heavy.h is planned for removal no sooner than July 2022.

Configuration File -msg The `lint_off` "-msg" option has been replaced with the "-rule" option. "-msg" is planned for removal no sooner than January 2021.

XML locations The XML "fl" attribute has been replaced with the "loc" attribute. "fl" is planned for removal no sooner than January 2021.

CHAPTER

SIXTEEN

CONTRIBUTORS AND ORIGINS

16.1 Authors

When possible, please instead report bugs at Verilator Issues.

Primary author is Wilson Snyder <wsnyder@wsnyder.org>.

Major concepts by Paul Wasson, Duane Galbi, John Coiner, Geza Lore, Yutetsu Takatsukasa, and Jie Xu.

16.2 Contributors

Many people have provided ideas and other assistance with Verilator.

Verilator is receiving major development support from the CHIPS Alliance.

Previous major corporate sponsors of Verilator, by providing significant contributions of time or funds included include: Atmel Corporation, Cavium Inc., Compaq Corporation, Digital Equipment Corporation, Embecosm Ltd., Hicamp Systems, Intel Corporation, Mindspeed Technologies Inc., MicroTune Inc., picoChip Designs Ltd., Sun Microsystems Inc., Nauticus Networks Inc., and SiCortex Inc.

The people who have contributed major functionality are: Byron Bradley, Jeremy Bennett, Lane Brooks, John Coiner, Duane Galbi, Geza Lore, Todd Strader, Stefan Wallentowitz, Paul Wasson, Jie Xu, and Wilson Snyder. Major testers included Jeff Dutton, Jonathon Donaldson, Ralf Karge, David Hewson, Iztok Jeras, Wim Michiels, Alex Solomatnikov, Sebastien Van Cauwenberghe, Gene Weber, and Clifford Wolf.

Some of the people who have provided ideas, and feedback for Verilator include: David Addison, Nikana Anastasiadis, Vasu Arasanipalai, Jens Arm, Tariq B. Ahmad, Sharad Bagri, Matthew Ballance, Andrew Bardsley, Matthew Barr, Geoff Barrett, Kaleb Barrett, Julius Baxter, Jeremy Bennett, Michael Berman, Jean Berniolles, Victor Besyakov, Moinak Bhattacharyya, Krzysztof Bieganski, David Binderman, Piotr Binkowski, Johan Bjork, David Black, Tymoteusz Blazejczyk, Daniel Bone, Morten Borup Petersen, Gregg Bouchard, Christopher Boumenot, Nick Bowler, Byron Bradley, Bryan Brady, Charlie Brej, J Briquet, Lane Brooks, John Brownlee, Jeff Bush, Lawrence Butcher, Tony Bybell, Iru Cai, Ted Campbell, Chris Candler, Lauren Carlson, Donal Casey, Alex Chadwick, Terry Chen, Yi-Chung Chen, Enzo Chi, Robert A. Clark, Ryan Clarke, Allan Cochrane, John Coiner, Gianfranco Costamagna, Sean Cross, George Cuan, Joe DErrico, Lukasz Dalek, Gunter Dannoritzer, Ashutosh Das, Maarten De Braekeleer, Bernard Deadman, Alberto Del Rio, John Demme, Mike Denio, John Deroo, Philip Derrick, John Dickol, Ruben Diez, Danny Ding, Jacko Dirks, Ivan Djordjevic, Jonathon Donaldson, Sebastian Dressler, Alex Duller, Jeff Dutton, Tomas Dzetkulic, Richard E George, Edgar E. Iglesias, Usuario Eda, Charles Eddleston, Chandan Egbert, Jan Egil Ruud, Joe Eiler, Ahmed El-Mahmoudy, Trevor Elbourne, Mats Engstrom, Charles Eric LaForest, Robert Farrell, Eugen Fekete, Fabrizio Ferrandi, Udi Finkelstein, Brian Flachs, Andrea Foletto, Bob Fredieu, Duane Galbi, Benjamin Gartner, Christian Gelinek, Peter Gerst, Glen Gibb, Michael Gielda, Barbara Gigerl, Shankar Giri, Dan Gisselquist, Petr Gladkikh, Sam Gladstone, Andrew Goessling, Amir Gonnen, Chitlesh Goorah, Tomasz Gorochowik, Kai Gossner, Sergi Granell, Al Grant, Alexander Grobman, Xuan Guo, Driss Hafdi, Neil Hamilton, James Hanlon, Oyvind Harboe, Jannis Harder,

Junji Hashimoto, Thomas Hawkins, Mitch Hayenga, Harald Heckmann, Robert Henry, Stephen Henry, David Hewson, Jamey Hicks, Joel Holdsworth, Andrew Holme, Hiroki Honda, Alex Hornung, Pierre-Henri Horrein, David Horton, Peter Horvath, Jae Hossell, Kuoping Hsu, Alan Hunter, James Hutchinson, Anderson Ignacio da Silva, Jamie Iles, Thomas J Whatson, Ben Jackson, Mark Jackson Pulver, Shareef Jalloq, Marlon James, Krzysztof Jankowski, HyungKi Jeong, Iztok Jeras, James Johnson, Christophe Joly, Franck Jullien, James Jung, Mike Kagen, Arthur Kahlich, Kaalia Kahn, Guy-Armand Kamendje, Vasu Kandadi, Kanad Kanhere, Patricio Kaplan, Pieter Kapsenberg, Rafal Kapuscik, Ralf Karge, Dan Katz, Sol Katzman, Ian Kennedy, Michael Killough, Jonathan Kimmitt, Olof Kindgren, Kevin Kiningham, Dan Kirkham, Sobhan Klnv, Gernot Koch, Jack Koenig, Soon Koh, Nathan Kohagen, Steve Kolecki, Brett Koonce, Will Korteland, Wojciech Koszek, Varun Koyyalagunta, Markus Krause, David Kravitz, Roland Kruse, Andreas Kuster, Sergey Kvachonok, Ed Lander, Steve Lang, Stephane Laurent, Walter Lavino, Christian Leber, Larry Lee, Igor Lesik, John Li, Eivind Liland, Charlie Lind, Andrew Ling, Jiuyang Liu, Paul Liu, Derek Lockhart, Jake Longo, Geza Lore, Arthur Low, Stefan Ludwig, Dan Lussier, Fred Ma, Duraid Madina, Odd Magne Reitan, Affe Mao, Julien Margetts, Mark Marshall, Alfonso Martinez, Unai Martinez-Corral, Yves Mathieu, Patrick Maupin, Conor McCullough, Jason McMullan, Elliot Mednick, Wim Michiels, Miodrag Milanovic, Peter Monsson, Sean Moore, Dennis Muhlestein, John Murphy, Matt Myers, Nathan Myers, Richard Myers, Dimitris Nalbantis, Peter Nelson, Bob Newgard, Paul Nitza, Yossi Nivin, Pete Nixon, Lisa Noack, Mark Nodine, Kuba Ober, Andreas Olofsson, Baltazar Ortiz, Aleksander Osman, Don Owen, James Pallister, Vassilis Papaefstathiou, Brad Parker, Dan Petrisko, Maciej Piechotka, David Pierce, Cody Piersall, Dominic Plunkett, David Poole, Mike Popoloski, Roman Popov, Rich Porter, Niranjan Prabhu, Usha Priyadharshini, Prateek Puri, Marshal Qiao, Nandu Raj, Danilo Ramos, Chris Randall, Anton Rapp, Josh Redford, Frederic Requin, Dustin Richmond, Samuel Riedel, Eric Rippey, Oleg Rodionov, Ludwig Rogiers, Paul Rolfe, Arjen Roodselaar, Tobias Rosenkranz, Huang Rui, Denis Rystsov, John Sanguinetti, Galen Seitz, Joseph Shaker, Salman Sheikh, Yu Sheng Lin, Hao Shi, Mike Shinkarovsky, Rafael Shirakawa, Jeffrey Short, Fan Shupei, Rodney Sinclair, Steven Slatter, Brian Small, Garrett Smith, Tim Snyder, Wilson Snyder, Maciej Sobkowski, Stan Sokorac, Alex Solomatnikov, Wei Song, Art Stamness, David Stanford, John Stevenson, Pete Stevenson, Patrick Stewart, Rob Stoddard, Todd Strader, John Stroebel, Sven Stucki, Howard Su, Emerson Suguimoto, Gene Sullivan, Wai Sum Mong, Qingyao Sun, Renga Sundararajan, Rupert Swarbrick, Yutetsu Takatsukasa, Thierry Tambe, Drew Taussig, Peter Tengstrand, Wesley Terpstra, Rui Terra, Stefan Thiede, Gary Thomas, Ian Thompson, Kevin Thompson, Mike Thyer, Hans Tichelaar, Viktor Tomov, Steve Tong, Alex Torregrosa, Michael Tresidder, David Turner, Neil Turton, Cong Van Nguyen, Hans Van Antwerpen, Jan Van Winkel, Sebastien Van Cauwenberghe, Laurens van Dam, Leendert van Doorn, Srini Vemuri, Yuri Victorovich, Bogdan Vukobratovic, Holger Waechtler, Philipp Wagner, Stefan Wallentowitz, Shawn Wang, Paul Wasson, Greg Waters, Thomas Watts, Eugene Weber, David Welch, Martin Whitaker, Marco Widmer, Leon Wildman, Daniel Wilkerson, Gerald Williams, Trevor Williams, Jeff Winston, Joshua Wise, Clifford Wolf, Tobias Wolfel, Johan Wouters, Paul Wright, Junyi Xi, Ding Xiaoliang, Jie Xu, Mandy Xu, Yinan Xu, Luke Yang, Amir Yazdanbakhsh, and Keyi Zhang.

Thanks to them, and all those we've missed including above, or wished to remain anonymous.

16.3 Historical Origins

Verilator was conceived in 1994 by Paul Wasson at the Core Logic Group at Digital Equipment Corporation. The Verilog code that was converted to C was then merged with a C based CPU model of the Alpha processor and simulated in a C based environment called CCLI.

In 1995 Verilator started being used also for Multimedia and Network Processor development inside Digital. Duane Galbi took over active development of Verilator, and added several performance enhancements. CCLI was still being used as the shell.

In 1998, through the efforts of existing DECies, mainly Duane Galbi, Digital graciously agreed to release the source code. (Subject to the code not being resold, which is compatible with the GNU Public License.)

In 2001, Wilson Snyder took the kit, and added a SystemC mode, and called it Verilator2. This was the first packaged public release.

In 2002, Wilson Snyder created Verilator 3.000 by rewriting Verilator from scratch in C++. This added many optimizations, yielding about a 2-5x performance gain.

In 2009, major SystemVerilog and DPI language support was added.

In 2018, Verilator 4.000 was released with multithreaded support.

In 2019, Verilator joined the CHIPS Alliance.

Currently, various language features and performance enhancements are added as the need arises. Verilator is now about 3x faster than in 2002, and is faster than most (if not every) other simulator.

CHAPTER

SEVENTEEN

REVISION HISTORY

Changes are contained in the `Changes` file of the distribution, and also summarized below. To subscribe to new versions see Verilator Announcements.

17.1 Revision History and Change Log

The changes in each Verilator version are described below. The contributors that suggested a given feature are shown in []. Thanks!

17.1.1 Verilator 4.215 devel

Major:

- Add –lib-create, similar to –protect-lib but without protections.

Minor:

- Internal code cleanups and improvements. [Geza Lore]
- Improve –thread verilation-time performance.
- Support task name in $display %m (#3211). [Julie Schwartz]
- Make 'bit', 'logic' and 'time' types unsigned by default. [Geza Lore]
- Fix array method names with parenthesis (#3181) (#3183). [Teng Huang]
- Fix split_var assign merging (#3177) (#3179). [Yutetsu TAKATSUKASA]
- Fix wrong bit op tree optimization (#3185). [Yutetsu TAKATSUKASA]
- Fix some SliceSels not being constants (#3186) (#3218). [Michaël Lefebvre]
- Fix nested generate if genblk naming (#3189). [yanx21]
- Fix hang on recursive definition error (#3199). [Jonathan Kimmitt]
- Fix display of signed without format (#3204). [Julie Schwartz]
- Fix display of empty string constant (#3207) (#3215). [Julie Schwartz]
- Fix incorrect width after and-or optimization (#3208). [Julie Schwartz]
- Fix $fopen etc on integer arrays (#3214). [adrienlemasle]
- Fix $size on dynamic strings (#3216).
- Fix %0 format on $value$plusargs (#3217).

17.1.2 Verilator 4.214 2021-10-17

Major:

- Add profile-guided optmization of mtasks (#3150).

Minor:

- Verilator_gantt has removed the ASCII graphics, use the VCD output instead.
- Verilator_gantt now shows the predicted mtask times, eval times, and additional statistics.
- Verilator_gantt data files now include processor information, to allow later processing.
- Support displaying x and z in $display task (#3107) (#3109). [Iru Cai]
- Fix verilator_profcfunc profile accounting (#3115).
- Fix display has no time units on class function (#3116). [Damien Pretet]
- Fix removing if statement with side effect in condition (#3131). [Alexander Grobman]
- Fix –waiver-output for multiline warnings (#2429) (#3141). [Keith Colbert]
- Fix internal error on bad widths (#3140) (#3145). [Zhanglei Wang]
- Fix crash on clang 12/13 (#3148). [Kouping Hsu]
- Fix cygwin compile error due to missing -std=gnu++14 (#3149). [Sun Kim]
- Fix $urandom_range when the range is 0 . . . UINT_MAX (#3161). [Iru Cai]
- Fix constructor-parameter argument comma-separation in C++ (#3162). [Matthew Ballance]
- Fix missing install of vl_file_copy/vl_hier_graph (#3165). [Popolon]
- Fix calling new with arguments in same class (#3166). [Matthew Ballance]
- Fix false EOFNEWLINE warning when DOS carriage returns present (#3171).

17.1.3 Verilator 4.212 2021-09-01

Minor:

- Fix re-evaluation of logic dependent on state set in DPI exports (#3091). [Geza Lore]
- Support unpacked array localparams in tasks/functions (#3078). [Geza Lore]
- Support timeunit/timeprecision in $unit.
- Support assignment patterns as children of pins (#3041). [Krzysztof Bieganski]
- Add –instr-count-dpi to tune assumed DPI import cost for multithreaded model scheduling. Default value changed to 200 (#3068). [Yinan Xu]
- Output files are split based on the set of headers required in order to aid incremental compilation via ccache (#3071). [Geza Lore]
- Parameter values are now emitted as ‘static constexpr’ instead of enum. C++ direct references to parameters might require updating (#3077). [Geza Lore]
- Refactored Verilated include files; include verilated.h not verilated_heavy.h.
- Add header guards on Dpi.h generated files (#2979). [Tood Strader]
- Add XML ccall, constpool, initarray, and if/while begins (#3080). [Steven Hugg]

- Add error when constant function under a generate (#3103). [Don Owen]
- Fix -G to treat simple integer literals as signed (#3060). [Anikin1610]
- Fix emitted string array initializers (#2895). [Iztok Jeras]
- Fix bitop tree optimization dropping necessary & operator (#3096). [Flavien Solt]
- Fix internal error on wide -x-initial unique (#3106). [Alexandre Joannou]
- Fix traces to show array instances with brackets (#3092) (#3095). [Pieter Kapsenberg]

17.1.4 Verilator 4.210 2021-07-07

Major:

- Generated code is now emitted as global functions rather than methods. '$c' contents might need to be updated, see the docs (#3006). [Geza Lore]
- The generated model class instantiated by the user is now an interface object and no longer the TOP module instance. User code with direct C++ member access to model internals, including verilator public_flat items will likely need to be updated. See the manual for instructions: https://verilator.org/guide/latest/connecting.html#porting-from-pre-4-210 (#3036). [Geza Lore]

Minor:

- Add –prof-c to pass profiling to compiler (#3059). [Alexander Grobman]
- Optimize a lot more model variables into function locals (#3027). [Geza Lore]
- Support middle-of-design nested topmodules (#3026). [Dan Petrisko]
- Remove deprecated –no-relative-cfuncs option (#3024). [Geza Lore]
- Remove deprecated –inhibit-sim option (#3035). [Geza Lore]
- Merge const static data globally into a new constant pool (#3013). [Geza Lore]
- Allow configure override of AR program (#2999). [ahouska]
- In XML, show pinIndex information (#2877). [errae233]
- Fix error on unsupported recursive functions (#2957). [Trefor Southwell]
- Fix type parameter specialization when struct names are same (#3055). [7FM]
- Improve speed of table optimization (-OA) pass. [Geza Lore]

17.1.5 Verilator 4.204 2021-06-12

Minor:

- Add 'make ccache-report' (#3011). [Geza Lore]
- Add –reloop-limit argument (#2943) (#2960). [Geza Lore]
- Add –expand-limit argument (#3005). [Julien Margetts]
- Add TRACE_THREADS to CMake (#2934). [Jonathan Drolet]
- Optimize large lookup tables to static data (#2925). [Geza Lore]
- Optimize reloop to accept constant index offsets (#2939). [Geza Lore]
- Split always blocks to better respect –output-split-cfuncs. [Geza Lore]

- Support ignoring "&96;pragma protect ..." (#2886). [Udi Finkelstein]
- Support –trace-fst for SystemC with CMake (#2927). [Jonathan Drolet]
- Update cmake latest C++ Standard Compilation flag (#2951). [Ameya Vikram Singh]
- Prep work towards better ccache hashing/performance. [Geza Lore]
- Fix assertion failure in bitOpTree optimization (#2891) (#2899). [Raynard Qiao]
- Fix DPI functions not seen as vpiModule (#2893). [Todd Strader]
- Fix bounds check in VL_SEL_IWII (#2910). [Krzysztof Bieganski]
- Fix slowdown in elaboration (#2911). [Nathan Graybeal]
- Fix initialization of assoc in assoc array (#2914). [myftptoyman]
- Fix make support for gmake 3.x (#2920) (#2921). [Philipp Wagner]
- Fix VPI memory access for packed arrays (#2922). [Todd Strader]
- Fix MCD close also closing stdout (#2931). [Alexander Grobman]
- Fix split procedures to better respect –output-split-cfuncs (#2942). [Geza Lore]
- Fix to emit 'else if' without nesting (#2944). [Geza Lore]
- Fix part select issues in LATCH warning (#2948) (#2938). [Julien Margetts]
- Fix to not emit empty files with low split limits (#2961). [Geza Lore]
- Fix merging of assignments in C++ code (#2970). [Ruper Swarbrick]
- Fix unused variable warnings (#2991). [Pieter Kapsenberg]
- Fix –protect-ids when using SV classes (#2994). [Geza Lore]
- Fix constant function calls with uninit value (#2995). [yanx21]
- Fix Makefiles to support Windows EXEEXT usage (#3008). [Miodrag Milanovic]

17.1.6 Verilator 4.202 2021-04-24

Major:

- Documentation has been rewritten into a book format.
- Verilated signals now use VlWide and VlPacked in place of C arrays.

Minor:

- Add an URL on warnings to point to the manual's description.
- Add EOFNEWLINE warning when missing a newline at EOF.
- Changed TIMESCALEMOD from error into a warning.
- Mark –no-relative-cfuncs as scheduled for deprecation.
- Add –coverage-max-width (#2853). [xuejiazidi]
- Add VerilatedCovContext::forcePerInstance (#2793). [Kevin Laeufer]
- Add FST SystemC tracing (#2806). [Alex Torregrosa]
- Add PINNOTFOUND warning in place of error (#2868). [Udi Finkelstein]
- Support overlaps in priority case statements (#2864). [Rupert Swarbrick]

- Support for null ports (#2875). [Udi Finkelstein]
- Fix class unpacked-array compile error (#2774). [Iru Cai]
- Fix scope types in FST and VCD traces (#2805). [Alex Torregrosa]
- Fix exceeding command-line ar limit (#2834). [Yinan Xu]
- Fix false $dumpfile warning on model save (#2834). [Yinan Xu]
- Fix –timescale-override not suppressing TIMESCALEMOD (#2838). [Kaleb Barrett]
- Fix false TIMESCALEMOD on generate-ignored instances (#2838). [Kaleb Barrett]
- Fix –output-split with class extends (#2839). [Iru Cai]
- Fix false WIDTHCONCAT on casted constant (#2849). [Rupert Swarbrick]
- Fix tracing of long hashed names (#2854). [Graham Rushton]
- Fix –public-flat-rw / DPI issue (#2858). [Todd Strader]
- Fix interface localparam access (#2859). [Todd Strader]
- Fix Cygwin example compile issues (#2856). [Mark Shaw]
- Fix select of with index variable (#2880). [Alexander Grobman]
- Fix cmake version number to be numeric (#2881). [Yuri Victorovich]
- Fix MinGW not supportting 'localtime_r' (#2882). [HyungKi Jeong]
- Fix cast from packed, typedef'ed interface signal (#2884). [Todd Strader]
- Fix VPI package reported as vpiModule (#2885). [Todd Strader]
- Fix dumping waveforms to multiple FST files (#2889). [David Metz]
- Fix assertion failure in bitOpTree (#2892). [Yutetsu TAKATSUKASA]
- Fix V3Premit infinite loop on always read-and-write (#2898). [Raynard Qiao]
- Fix VPI packed vectors (#2900). [Todd Strader]
- Fix VPI public interface parameters (#2901). [Todd Strader]

17.1.7 Verilator 4.200 2021-03-12

Announcement:

- –inhibit-sim is planned for deprecation, file a bug if this is still being used.

Major:

- Add simulation context (VerilatedContext) to allow multiple fully independent models to be in the same process. Please see the updated examples. (#2660)
- Add context->time() and context->timeInc() API calls, to set simulation time. These now are recommended in place of the legacy sc_time_stamp().

Minor:

- Converted Asciidoc documentation into reStructuredText (RST) format.
- Fix range inheritance on port without data type (#2753). [Embedded Go]
- Fix slice-assign overflow (#2803) (#2811). [David Turner]
- Fix interface array connection ordering broken in v4.110 (#2827). [Don Owen]

- Fix or-reduction on different scopes broken in 4.110 (#2828). [Yinan Xu]
- Fix MSVC++ compile error. (#2831) (#2833) [Drew Taussig]

17.1.8 Verilator 4.110 2021-02-25

Major:

- Optimize bit operations and others (#2186) (#2632) (#2633) (#2751) (#2800) [Yutetsu TAKATSUKASA]

Minor:

- Support concat selection (#2721).
- Support struct scopes when dumping structs to VCD (#2776) [Alex Torregrosa]
- Generate SELRANGE for potentially unreachable code (#2625) (#2754) [Pierre-Henri Horrein]
- For –flatten, override inlining of public and no_inline modules (#2761) [James Hanlon]
- Fix little endian interface pin swizzling (#2475). [Don Owen]
- Fix range inheritance on port without data type (#2753). [Embedded Go]
- Fix TIMESCALE warnings on primitives (#2763). [Xuanqi]
- Fix to exclude strings from toggle coverage (#2766) (#2767) [Paul Wright]
- Fix $fread extra semicolon inside statements. [Leendert van Doorn]
- Fix class extends with VM_PARALLEL_BUILDS (#2775). [Iru Cai]
- Fix shifts by > 32 bit values (#2785). [qrq992]
- Fix examples not flushing vcd (#2787). [Richard E George]
- Fix little endian packed array pattern assignment (#2795). [Alex Torregrosa]

17.1.9 Verilator 4.108 2021-01-10

Major:

- Many VPI changes for IEEE compatibility, which may alter behavior from previous releases.
- Support randomize() class method and rand (#2607). [Krzysztof Bieganski]

Minor:

- Support $cast and new CASTCONST warning.
- Add –top option as alias of –top-module.
- Add LATCH and NOLATCH warnings (#1609) (#2740). [Julien Margetts]
- Remove Unix::Processors internal test dependency.
- Report UNUSED on parameters, localparam and genvars (#2627). [Charles Eric LaForest]
- Add error on real to non-real output pins (#2690). [Peter Monsson]
- Support package imports before parameters in interfaces (#2714). [James Hanlon]
- Support –sanitize in internal tests (#2705). [Yutetsu TAKATSUKASA]
- Fix passing parameter type instantiations by position number.
- Fix DPI open array handling issues.

- Fix error when dotted refers to missing module (#2095). [Alexander Grobman]
- Fix little endian packed array counting (#2499). [phantom-killua]
- Fix showing reference locations for BLKANDNBLK (#2170). [Yuri Victorovich]
- Fix genblk naming to match IEEE (#2686). [tinshark]
- Fix VPI memory word indexing (#2695). [Marlon James]
- Fix vpiLeftRange on little-endian memories (#2696). [Marlon James]
- Fix VPI module tree (#2704). [Todd Strader]
- Fix vpi_release_handle to be called implicitly per IEEE (#2706).
- Fix to allow inheriting 'VerilatedVcdFile' class. (#2720) [HyungKi Jeong]
- Fix $urandom_range maximum value (#2723). [Nandu Raj]
- Fix tracing empty sc module (#2729).
- Fix generate for unrolling to be signed (#2730). [yanx21]
- Fix to emit timescale in hierarchical blocks (#2735). [Yutetsu TAKATSUKASA]
- Fix to ignore coverage on real ports (#2741) (#2745). [Paul Wright]

17.1.10 Verilator 4.106 2020-12-02

Major:

- Change -sv option to select 1800-2017 instead of 1800-2005.

Minor:

- Check for proper 'local' and 'protected' (#2228).
- Support $random and $urandom seeds.
- Support $monitor and $strobe.
- Support complex function arguments.
- Support 'super'.
- Support 'with item.index'.
- Fix the default GNU Make executable name on FreeBSD (#2553). [Yuri Victorovich]
- Fix trace signal names getting hashed (#2643). [Barbara Gigerl]
- Fix unpacked array parameters near functions (#2639). [Anderson Ignacio da Silva]
- Fix access to non-overridden base class variable (#2654). [Tobias Rosenkranz]

17.1.11 Verilator 4.104 2020-11-14

Minor:

- Support queue and associative array 'with' statements (#2616).
- Support queue slicing (#2326).
- Support associative array pattern assignments and defaults.
- Support static methods and typedefs in classes (#2615). [Krzysztof Bieganski]
- Add error on typedef referencing self (#2539). [Cody Piersall]
- With –debug, turn off address space layout randomization.
- Fix iteration over mutating list bug in VPI (#2588). [Kaleb Barrett]
- Fix cast width propagation (#2597). [flex-liu]
- Fix return from callValueCbs (#2589) (#2605). [Marlon James]
- Fix WIDTH warnings on comparisons with nullptr (#2602). [Rupert Swarbrick]
- Fix fault when $fgets, $sscanf, etc used with string (#2604). [Yutetsu TAKATSUKASA]
- Fix WIFEXITED missing from MinGW/MSYS2 (#2609). [Jean Berniolles]
- Fix queue poping wrong value when otherwise unused (#2512). [nanduraj1]
- Fix arrays of modport interfaces (#2614). [Thierry Tambe]
- Fix split_var internal error (#2640) (#2641). [Yutetsu TAKATSUKASA]

17.1.12 Verilator 4.102 2020-10-15

Minor:

- Support const object new() assignments.
- Support # as a comment in -f files (#2497). [phantom-killua]
- Support 'this' (#2585). [Rafal Kapuscik]
- Support defines for FST tracing (#2592). [Markus Krause]
- Support non-overapping implication inside properties (#1292). [Peter Monsson]
- Fix timescale with –hierarchical (#2554). [Yutetsu TAKATSUKASA]
- Fix cmake build with –hierarchical (#2560). [Yutetsu TAKATSUKASA]
- Fix -G dropping public indication (#2561). [Andrew Goessling]
- Fix $urandom_range passed variable (#2563). [nanduraj1]
- Fix method calls to package class functions (#2565). [Peter Monsson]
- Fix class wide member display (#2567). [Nandu Raj P]
- Fix hierarchical references inside function (#2267) (#2572). [James Pallister]
- Fix flushCall for backward compatibility (#2580). [chenguokai]
- Fix preprocessor stringify of undefined macro. [Martin Whitaker]

17.1.13 Verilator 4.100 2020-09-07

Major:

- C++11 or newer compilers are now required.
- SystemC 2.3.0 or newer (SYSTEMC_VERSION >= 20111121) is now required.
- Support hierarchical Verilation (#2206). [Yutetsu TAKATSUKASA]

Minor:

- Support (with limitations) class extern, class extends, virtual class.
- Support $urandom, $urandom_range without stability.
- Support assume property. [Peter Monsson]
- Support non-overapping implication inside properties (#1292). [Peter Monsson]
- Fix false DECLFILENAME on black-boxed modules (#2430). [Philipp Wagner]
- Fix naming of "id : begin" blocks.
- Fix class constructor error on assignments to const.
- Fix splitting eval functions with –output-split-cfuncs (#2368). [Geza Lore]
- Fix queues as class members (#2525). [nanduraj1]

17.1.14 Verilator 4.040 2020-08-15

Announcement:

- Version 4.040 is planned to be the final version that will support pre-C++11 compilers. Please move to C++11 or newer compilers.

Minor:

- Fix arrayed interfaces, broke in 4.038 (#2468). [Josh Redford]
- Support $stable, $rose and $fell. (#2148) (#2501) [Peter Monsson]
- Support simple function localparams (#2461). [James Hanlon]
- Miscellaneous parsing error changes towards UVM support.
- Fix arrayed interfaces (#2469). [Josh Redford]
- Fix protect lib VCS warning. (#2479) [Julien Margetts]
- Fix combining different-width parameters (#2484). [abirkmanis]
- Fix protect-lib without sequential logic (#2492). [Yutetsu TAKATSUKASA]
- Fix V3Unknown from running with flat XML output (#2494). [James Hanlon]
- Fix non-32 bit conversion to float (#2495). [dsvf]
- Fix casting non-self-determined subexpressions (#2493). [phantom-killua]
- Fix SystemC net names (#2500). [Edgar E. Iglesias]
- Fix build with Bison 3.7 and newer (#2505). [Rupert Swarbrick]
- Fix slice of unpacked array (#2506) (#2507). [Yutetsu TAKATSUKASA]

17.1.15 Verilator 4.038 2020-07-11

Announcement:

- Versions 4.038 and 4.040 are planned to be the final versions that will support pre-C++11 compilers. Please move to C++11 or newer compilers.

Minor:

- Support VPI access to parameters and localparam. [Ludwig Rogiers]
- Support parsing (not elaboration, yet) of UVM.
- Add new UNSUPPORTED error code to replace most previous Unsupported: messages.
- With –bbox-unsup continue parsing on many (not all) UVM constructs.
- Support for-loop increments with commas.
- Support $swrite with arbitrary arguments.
- Support $writememb (#2450). [Fan Shupei]
- Fix OS X, Free BSD, and -m32 portability issues. [Geza Lore]
- Fix to flush FST trace on termination due to $stop or assertion failure.
- Fix part select error when multipling by power-of-two (#2413). [Conor McCullough]
- Fix division exception (#2460) [Kuoping Hsu]

17.1.16 Verilator 4.036 2020-06-06

Major:

- OPT_FAST is now -Os by default. See the BENCHMARKING & OPTIMIZATION part of the manual if you experience issues with compilation speed.
- –output-split is now on by default. VM_PARALLEL_BUILDS is set by default iff the –output-split caused an actual file split to occur. –output-split-cfuncs and –output-split-ctrace now default to the value of –output-split. These changes should improve build times of medium and large designs with default options. User makefiles may require changes.

Minor:

- Configure now enables SystemC if it is installed as a system headers, e.g. with 'apt-get install systemc-dev'.
- Add –waiver-output flag that writes a verilator config file (.vlt) with waivers to the warnings emitted during a Verilator run.
- Support verilator_coverage –write-info for lcov HTML reports.
- Line Coverage now tracks all statement lines, not just branch lines.
- The run-time library is now compiled with -Os by default. (#2369, #2373)
- Support multi channel descriptor I/O (#2190) [Stephen Henry]
- Support $countbits. (#2287) [Yossi Nivin]
- Support $isunbounded and parameter $. (#2104)
- Support unpacked array .sum and .product.
- Support prefix/postfix increment/decrement. (#2223) [Maciej Sobkowski]

- Fix FST tracing of little bit endian signals. [Geza Lore]
- Fix +: and -: on unpacked arrays. (#2304) [engr248]
- Fix $isunknown with constant Z's.
- Fix queues and dynamic array wide ops. (#2352) [Vassilis Papaefstathiou]

17.1.17 Verilator 4.034 2020-05-03

Major:

- Support simplistic classes with many restrictions, see manual. (#377)
- Support IEEE time units and time precisions. (#234) Includes &96;timescale, $printtimescale, $timeformat. VL_TIME_MULTIPLIER, VL_TIME_PRECISION, VL_TIME_UNIT have been removed and the time precision must now match the SystemC time precision. To get closer behavior to older versions, use e.g. –timescale-override "1ps/1ps".
- Add –build to call make automatically. (#2249) [Yutetsu TAKATSUKASA]
- Configuring with ccache present now defaults to using it; see OBJCACHE.
- Fix DPI import/export to be standard compliant. (#2236) [Geza Lore]
- Add –trace-threads for general multithreaded tracing. (#2269) [Geza Lore]

Minor:

- Add –flatten for use with –xml-only. (#2270) [James Hanlon]
- Greatly improve FST/VCD dump performance (#2244) (#2246) (#2250) (#2257) [Geza Lore]
- Support $ferror, and $fflush without arguments. (#1638)
- Support event data type (with some restrictions).
- Support $root. (#2150) [Keyi Zhang]
- Add error if use SystemC 2.2 and earlier (pre-2011) as is deprecated.
- Add support of –trace-structs for CMake (#2986). [Martin Schmidt]
- Fix arrayed instances connecting to slices. (#2263) [Don/engr248]
- Fix error on unpacked connecting to packed. (#2288) [Joseph Shaker]
- Fix logical not optimization with empty begin. (#2291) [Baltazar Ortiz]
- Fix reduction OR on wide data, broke in v4.026. (#2300) [Jack Koenig]
- Fix clock enables with bit-extends. (#2299) [Marco Widmer]
- Fix MacOs Homebrew by removing default LIBS. (#2298) [Ryan Clarke]

17.1.18 Verilator 4.032 2020-04-04

Minor:

- Add column numbers to errors and warnings.
- Add GCC 9-style line number prefix when showing source text for errors.
- Add setting VM_PARALLEL_BUILDS=1 when using –output-split. (#2185)
- Change –quiet-exit to also suppress 'Exiting due to N errors'.
- Suppress REALCVT for whole real numbers.
- Support split_var in vlt files. (#2219) [Marco Widmer]
- Fix parameter type redeclaring a type. (#2195) [hdzhangdoc]
- Fix VCD open with empty filename. (#2198) [Julius Baxter]
- Fix packages as enum base types. (#2202) [Driss Hafdi]
- Fix duplicate typedefs in generate for. (#2205) [hdzhangdoc]
- Fix MinW portability. (#2114) [Sean Cross]
- Fix assertions with unique case inside. (#2199) [hdzhangdoc]
- Fix implicit conversion of floats to wide integers.

17.1.19 Verilator 4.030 2020-03-08

Major:

- Add split_var metacomment to assist UNOPTFLAT fixes. (#2066) [Yutetsu TAKATSUKASA]
- Support $dumpfile and $dumpvars. (#2126) [Alexander Grobman]
- Support dynamic arrays. (#379)

Minor:

- Add +verilator+noassert flag to disable assertion checking. [Tobias Wölfel]
- Add check for assertOn for asserts. (#2162) [Tobias Wölfel]
- Add –structs-packed for forward compatibility.
- Support $displayb/o/h, $writeb/o/h, etc. (#1637)
- Use gcc -Os in examples instead of -O2 for better average performance.
- Fix genblk naming with directly nested generate blocks. (#2176) [Alexander Grobman]
- Fix undeclared VL_SHIFTR_WWQ. (#2114) [Alex Solomatnikov]

17.1.20 Verilator 4.028 2020-02-08

Major:

- Support attributes (public, isolate_assignments, etc.) in configuration files.
- Add -match to lint_off to waive warnings. [Philipp Wagner]

Minor:

- Link Verilator binary partially statically. (#2146) [Geza Lore]
- Verilation speed improvements (#2133) (#2138) [Geza Lore]
- Support libgoogle-perftools-dev's libtcmalloc if available. (#2137) [Geza Lore]
- Support $readmem/$writemem with assoc arrarys. (#2100) [agrobman]
- Support type(expression) operator and $typename. (#1650)
- Support left justified $display. (#2101) [Pieter Kapsenberg]
- Support string character access via indexing.
- Support enum.next(k) with constant k > 1. (#2125) [Tobias Rosenkranz]
- Support parameter access from arrays of interfaces. (#2155) [Todd Strader]
- Add parameter values in XML. #2110. [Pieter Kapsenberg]
- Add loc column location in XML (replaces fl). (#2122) [Pieter Kapsenberg]
- Add error on misused define. [Topa Tota]
- Add parameter to set maximum signal width. (#2082) [Øyvind Harboe]
- Add warning on genvar in normal for loop. (#2143) [Yuri Victorovich]
- Fix VPI scope naming for public modules. [Nandu Raj]
- Fix FST tracing of enums inside structs. [fsiegle]
- Fix WIDTH warning on </<= of narrower value. (#2141) [agrobman]
- Fix OpenSolaris issues. (#2154) [brancoliticus]
- Fix gated clocks under –protect-lib. (#2169) [Todd Strader]

17.1.21 Verilator 4.026 2020-01-11

Major:

- Docker images are now available for Verilator releases.

Minor:

- Support bounded queues.
- Support non-overlapping implication operator in assertions. (#2069) [Peter Monsson]
- Support string compare, ato*, etc methods. (#1606) [Yutetsu TAKATSUKASA]
- Support immediate cover statements.
- Ignore &96;uselib to end-of-line. (#1634) [Frederic Antonin]
- Update FST trace API for better performance.

- Add vpiTimeUnit and allow to specify time as string. (#1636) [Stefan Wallentowitz]
- Add error when &96;resetall inside module (IEEE 2017-22.3).
- Add cleaner error on version control conflicts in sources.
- Fix little endian cell ranges. (#1631) [Julien Margetts]
- Fix queue issues (#1641) (#1643) [Peter Monsson, Stefan Wallentowitz]
- Fix strcasecmp for windows. (#1651) [Kuba Ober]
- Fix disable iff in assertions. Closes #1404. [Peter Monsson]
- Fix huge case statement performance. Closes #1644. [Julien Margetts]
- Fix tracing -1 index arrays. Closes #2090. [Yutetsu Takatsukasa]
- Fix expand optimization slowing –lint-only. Closes #2091. [Thomas Watts]
- Fix %{number}s with strings. #2093. [agrobman]
- Fix shebang breaking some shells. Closes #2067. [zdave]

17.1.22 Verilator 4.024 2019-12-08

Major:

- Support associative arrays (excluding [*] and pattern assignments). (#544)
- Support queues (excluding {} notation and pattern assignments). (#545)

Minor:

- Add +verilator+error+limit to see more assertion errors. [Peter Monsson]
- Support string.toupper and string.tolower.
- Support $rewind and $ungetc.
- Support shortreal as real, with a SHORTREAL warning.
- Add -Wpedantic and -Wno-context for compliance testing.
- Add error on redefining preprocessor directives. [Piotr Binkowski]
- Support $value$plusargs float and shorts. (#1592) (#1619) [Garrett Smith]
- Fix gate lvalue optimization error. (#831) [Jonathon Donaldson, Driss Hafdi]
- Fix color assertion on empty if. (#1604) [Andrew Holme]
- Fix for loop missing initializer. (#1605) [Andrew Holme]
- Fix hang on concat error. (#1608) [Bogdan Vukobratovic]
- Fix VPI timed callbacks to be one-shot, pull5. [Matthew Ballance]
- Fix // in filenames. (#1610) [Peter Nelson]
- Fix $display("%p") to be closer to IEEE.
- Fix labels on functions with returns. (#1614) [Mitch Hayenga]
- Fix false unused message on __Vemumtab. (#2061) [Tobias Rosenkranz]
- Fix assertion on dotted parameter arrayed function. (#1620) [Rich Porter]
- Fix interface reference tracing. (#1595) [Todd Strader]

- Fix error on unpacked concatenations. (#1627) [Driss Hafdi]

17.1.23 Verilator 4.022 2019-11-10

Major:

- Add –protect-lib. (#1490) [Todd Strader]
- Add cmake support. (#1363) [Patrick Stewart]

Minor:

- Examples have been renamed.
- Add –protect-ids to obscure information in objects. (#1521) [Todd Strader]
- Add –trace-coverage.
- Add –xml-output.
- Support multithreading on Windows. [Patrick Stewart]
- Suppress 'command failed' on normal errors.
- Support some unpacked arrays in parameters. (#1315) [Marshal Qiao]
- Add interface port visibility in traces. (#1594) [Todd Strader]
- Increase case duplicate/incomplete to 16 bit tables. (#1545) [Yossi Nivin]
- Support quoted arguments in -f files. (#1535) [Yves Mathieu]
- Optimize modulus by power-of-two constants, and masked conditionals.
- Fix detecting missing reg types. (#1570) [Jacko Dirks]
- Fix multithreaded yield behavior when no work. [Patrick Stewart]
- Fix bad-syntax crashes. (#1548, #1550-#1553, #1557-#1560, #1563, #1573-#1577, #1579, #1582-#1591) [Eric Rippey]
- Fix false CMPCONST/UNSIGNED warnings on "inside". (#1581) [Mitch Hayenga]

17.1.24 Verilator 4.020 2019-10-06

Minor:

- Add –public-flat-rw. (#1511) [Stefan Wallentowitz]
- Support $fseek, $ftell, $frewind. (#1496) [Howard Su]
- Support vpiModule. (#1469) [Stefan Wallentowitz]
- Make Syms file honor –output-split-cfuncs. (#1499) [Todd Strader]
- Fix make test with no VERILATOR_ROOT. (#1494) [Ahmed El-Mahmoudy]
- Fix error on multidimensional cells. (#1505) [Anderson Ignacio Da Silva]
- Fix config_rev revision detection on old versions.
- Fix false warning on backward indexing. (#1507) [Hao Shi]
- Fix vpiType accessor. (#1509) (#1510) [Stefan Wallentowitz]
- Fix ugly error on interface misuse. (#1525) [Bogdan Vukobratovic]

- Fix misc bad-syntax crashes. (#1529) (#1530) (#1531) (#1532) (#1533) [Eric Rippey]
- Fix case statements with strings. (#1536) [Philipp Wagner]
- Fix some coverage lost when multithreaded. (#2151)

17.1.25 Verilator 4.018 2019-08-29

Major:

- When showing an error, show source code and offer suggestions of replacements.
- When showing an error, show the instance location. (#1305) [Todd Strader]

Minor:

- Add –rr. (#1481) [Todd Strader]
- Change MULTITOP to warning to help linting, see manual.
- Add XSim support to driver.pl. (#1493) [Todd Strader]
- Add –dpi-hdr-only. (#1491) [Todd Strader]
- Show included-from filenames in warnings. (#1439) [Todd Strader]
- Fix elaboration time errors. (#1429) [Udi Finkelstein]
- Fix not reporting some duplicate signals/ports. (#1462) [Peter Gerst]
- Fix not in array context on non-power-of-two slices. (#2027) [Yu Sheng Lin]
- Fix system compile flags injection. [Gianfranco Costamagna]
- Fix enum values not being sized based on parent. (#1442) [Dan Petrisko]
- Fix internal error on gate optimization of assign. (#1475) [Oyvind Harboe]

17.1.26 Verilator 4.016 2019-06-16

Minor:

- Add –quiet-exit. (#1436) [Todd Strader]
- Error continuation lines no longer have %Error prefix.
- Support logical equivalence operator <->.
- Support VerilatedFstC set_time_unit. (#1433) [Pieter Kapsenberg]
- Support deferred assertions. (#1449) [Charles Eddleston]
- Mark infrequently called functions with GCC cold attribute.
- Fix sign-compare warning in verilated.cpp. (#1437) [Sergey Kvachonok]
- Fix fault on $realtime with %t. (#1443) [Julien Margetts]
- Fix $display with string without %s. (#1441) [Denis Rystsov]
- Fix parameter function string returns. (#1441) [Denis Rystsov]
- Fix invalid XML output due to special chars. (#1444) [Kanad Kanhere]
- Fix performance when mulithreaded on 1 CPU. (#1455) [Stefan Wallentowitz]
- Fix type and real parameter issues (#1427) (#1456) (#1458) [Todd Strader]

- Fix build error on MinGW. (#1460) [Richard Myers]
- Fix not reporting some duplicate signals. (#1462) [Peter Gerst]
- Fix –savable invalid C++ on packed arrays. (#1465) [Alex Chadwick]
- Fix constant function return of function var. (#1467) [Roman Popov]

17.1.27 Verilator 4.014 2019-05-08

Minor:

- Add –trace-fst-thread.
- Support '#' comments in $readmem. (#1411) [Frederick Requin]
- Support "'dx" constants. (#1423) [Udi Finkelstein]
- For FST tracing use LZ4 compression. [Tony Bybell]
- Add error when use parameters without value. (#1424) [Peter Gerst]
- Auto-extend and WIDTH warn on unsized X/Zs. (#1423) [Udi Finkelstein]
- Fix missing VL_SHIFTL errors. (#1412) (#1415) [Larry Lee]
- Fix MinGW GCC 6 printf formats. (#1413) [Sergey Kvachonok]
- Fix test problems when missing fst2vcd. (#1417) [Todd Strader]
- Fix GTKWave register warning. (#1421) [Pieter Kapsenberg]
- Fix FST enums not displaying. (#1426) [Danilo Ramos]
- Fix table compile error with multiinterfaces. (#1431) [Bogdan Vukobratovic]

17.1.28 Verilator 4.012 2019-03-23

Minor:

- Add +verilator+seed. (#1396) [Stan Sokorac]
- Support $fread. [Leendert van Doorn]
- Support void' cast on functions called as tasks. (#1383) [Al Grant]
- Add IGNOREDRETURN warning. (#1383)
- Report PORTSHORT errors on concat constants. (#1400) [Will Korteland]
- Fix VERILATOR_GDB being ignored. (#2017) [Yu Sheng Lin]
- Fix $value$plus$args missing verilated_heavy.h. [Yi-Chung Chen]
- Fix MSVC compile error. (#1406) [Benjamin Gartner]
- Fix maintainer test when no Parallel::Forker. (#1977) [Enzo Chi]
- Fix +1364-1995ext flags applying too late. (#1384) [Al Grant]

17.1.29 Verilator 4.010 2019-01-27

Minor:

- Removed –trace-lxt2, use –trace-fst instead.
- For –xml, add additional information. (#1372) [Jonathan Kimmitt]
- Add circular typedef error. (#1388) [Al Grant]
- Add unsupported for loops error. (#1986) [Yu Sheng Lin]
- Fix FST tracing of wide arrays. (#1376) [Aleksander Osman]
- Fix error when pattern assignment has too few elements. (#1378) [Viktor Tomov]
- Fix error when no modules in $unit. (#1381) [Al Grant]
- Fix missing too many digits warning. (#1380) [Jonathan Kimmitt]
- Fix uninitialized data in verFiles and unroller. (#1385) (#1386) [Al Grant]
- Fix internal error on xrefs into unrolled functions. (#1387) [Al Grant]
- Fix DPI export void compiler error. (#1391) [Stan Sokorac]

17.1.30 Verilator 4.008 2018-12-01

Minor:

- Support “ref” and “const ref” pins and functions. (#1360) [Jake Longo]
- In –xml-only show the original unmodified names, and add module_files and cells similar to Verilog-Perl, msg2719. [Kanad Kanhere]
- Add CONTASSREG error on continuous assignments to regs. (#1369) [Peter Gerst]
- Add PROCASSWIRE error on behavioral assignments to wires, msg2737. [Neil Turton]
- Add IMPORTSTAR warning on import::* inside $unit scope.
- Fix –trace-lxt2 compile error on MinGW. (#1990) [HyungKi Jeong]
- Fix hang on bad pattern keys. (#1364) [Matt Myers]
- Fix crash due to cygwin bug in getline. (#1349) [Affe Mao]
- Fix __Slow files getting compiled with OPT_FAST. (#1370) [Thomas Watts]

17.1.31 Verilator 4.006 2018-10-27

Minor:

- Add –pp-comments. (#1988) [Robert Henry]
- Add –dump-defines.
- For –trace-fst, save enum decoding information. (#1358) [Sergi Granell] (To visualize enumeration data you must use GTKwave 3.3.95 or newer.)
- For –trace-fst, combine hier information into FST. [Tony Bybell]
- Fix –trace-lxt2 compile error on MinGW, msg2667. [HyungKi Jeong]
- Fix Windows .exe not found. (#1361) [Patrick Stewart]

17.1.32 Verilator 4.004 2018-10-06

Major:

- Add GTKWave FST native tracing. (#1356) [Sergi Granell] (Verilator developers need to pull the latest vcddiff.)

Minor:

- Support $past. [Dan Gisselquist]
- Support restrict. (#1350) [Clifford Wolf]
- Rename include/lxt2 to include/gtkwave.
- Fix replication of 64-bit signal change detects.
- Fix Mac OSX 10.13.6 / LLVM 9.1 compile issues. (#1348) [Kevin Kiningham]
- Fix MinGW compile issues. (#1979) [HyungKi Jeong]

17.1.33 Verilator 4.002 2018-09-16

Major:

- This is a major release. Any patches may require major rework to apply. [Thanks everyone]
- Add multithreaded model generation.
- Add runtime arguments.
- Add GTKWave LXT2 native tracing. (#1333) [Yu Sheng Lin]
- Note $random has new algorithm; results may vary vs. previous versions.

Minor:

- Better optimize large always block splitting. (#1244) [John Coiner]
- Add new reloop optimization for repetitive assignment compression.
- Support string.atoi and similar methods. (#1289) [Joel Holdsworth]
- Fix internals to be C++ null-pointer-check clean.
- Fix internals to avoid 'using namespace std'.
- Fix Verilation performance issues. (#1316) [John Coiner]
- Fix clocker attributes to not propagate on concats. [John Coiner]
- Fix first clock edge and –x-initial-edge. (#1327) [Rupert Swarbrick]
- Fix compile error on tracing of string arrays. (#1338) [Iztok Jeras]
- Fix number parsing with newline after radix. (#1340) [George Cuan]
- Fix string ?: conditional type resolution. (#1345) [Iztok Jeras]
- Fix duplicate symbol error on generate tri. (#1347) [Tomas Dzetkulic]

17.1.34 Verilator 3.926 2018-08-22

Minor:

- Add OBJCACHE envvar support to examples and generated Makefiles.
- Change MODDUP errors to warnings. (#1969) [Marshal Qiao]
- Fix define argument stringification (&96;"), broke since 3.914. [Joe DErrico]
- Fix to ignore Unicode UTF-8 BOM sequences. (#1967) [HyungKi Jeong]
- Fix std:: build error. (#1322)
- Fix function inlining inside certain while loops. (#1330) [Julien Margetts]

17.1.35 Verilator 3.924 2018-06-12

Minor:

- Renamed –profile-cfuncs to –prof-cfuncs.
- Report interface ports connected to wrong interface. (#1294) [Todd Strader]
- When tracing, use scalars on single bit arrays to appease vcddiff.
- Fix parsing "output signed" in V2K port list, msg2540. [James Jung]
- Fix parsing error on bad missing #. (#1308) [Dan Kirkham]
- Fix $clog2 to be in verilog 2005. (#1319) [James Hutchinson]

17.1.36 Verilator 3.922 2018-03-17

Major:

- Support IEEE 1800-2017 as default language.

Minor:

- Support trig functions ($sin() etc). (#1281) [Patrick Stewart]
- Support calling system functions as tasks. (#1285) [Joel Holdsworth]
- Support assert properties. (#785) (#1290) [John Coiner, et al]
- Support $writememh. [John Coiner]
- Add –no-debug-leak to reduce memory use under debug. [John Coiner]
- Fix severe runtime performance bug in certain foreach loops. [John Coiner]
- On convergence errors, show activity. [John Coiner]
- Fix GCC 8.0 issues. (#1273)
- Fix pullup/pulldowns on bit selects. (#1274) [Rob Stoddard]
- Fix verilator_coverage –annotate-min. (#1284) [Tymoteusz Blazejczyk]
- Fix quoting of quoted arguments. [John Coiner]

17.1.37 Verilator 3.920 2018-02-01

Announcement:

- Moving forward, use the git "stable" branch to track the latest release, and git "v#.###" tags for specific releases.

Minor:

- Support 'assume' similar to 'assert'. (#1269) [Dan Gisselquist]
- Remove c++filt. (#1265) [Stefan Wallentowitz]
- Fix tracing example file output. (#1268) [Enzo Chi]
- Fix gate optimization out of memory, add –gate-stmts. (#1260) [Alex Solomatnikov]
- Fix compile error on public real parameters by suppressing. (#1261) [Alex Solomatnikov]
- Fix input-only tristate comparisons. (#1267) [Alexis G]
- Fix missing edge type in xml output. (#1955) [Alexis G]
- Fix compile error with –public and interface bind. (#1264) [Alexis G]

17.1.38 Verilator 3.918 2018-01-02

Minor:

- Workaround GCC/clang bug with huge compile times. (#1248)
- Support DPI open arrays. (#909) (#1245) [David Pierce, Victor Besyakov]
- Add INFINITELOOP warning. (#1254) [Alex Solomatnikov]
- Support > 64 bit decimal $display.
- Support DPI time and svLogicVal. [Victor Besyakov] Note older version incorrectly assumed svBitVal even for logicals.
- Support string len() method. [Victor Besyakov]
- Add error if always_comb has sensitivity list. [Arjen Roodselaar]
- Fix SystemC 2.3.2 compile error. (#1251) [Tymoteusz Blazejczyk]
- Fix modport outputs being treated as inputs. (#1246) [Jeff Bush]
- Fix false ALWCOMBORDER on interface references. (#1247) [Josh Redford]
- Fix constant propagation across DPI imports of inout strings. [Victor Besyakov]
- Fix resolving inline nested interface names. (#1250) [Arjen Roodselaar]
- Fix GCC false warning on array bounds. (#2386)

17.1.39 Verilator 3.916 2017-11-25

Minor:

- Support self-recursive modules. (#659) [Sean Moore, et al]
- Support $error/$warning in elaboration time blocks.
- Support $size/$bits/etc on type references.
- Add error when driving input-only modport. (#1110) [Trevor Elbourne]
- Add BSSPACE and COLONPLUS lint warnings.
- Detect MSB overflow when under VL_DEBUG. (#1238) [Junyi Xi]
- Add data types to –xml. [Rui Terra]
- Fix partial slicing with pattern assignments. (#991) [Johan Bjork]
- Fix false unused warning on interfaces. (#1241) [Laurens van Dam]
- Fix error on "unique case" with no cases.
- Fix MacOS portability. (#1232) [Jeff Bush]

17.1.40 Verilator 3.914 2017-10-14

Major:

- Add new examples/ directory with appropriate examples. This replaces the old test_c and test_sc directories.

Minor:

- Add –getenv option for simplifying Makefiles.
- Add –x-initial option for specifying initial value assignment behavior.
- Add –no-relative-cfuncs and related default optimization. (#1224) [John Coiner]
- Add /*verilator tag*/ for XML extraction applications. [Chris Randall]
- The internal test_verilated test directory is moved to be part of test_regress.
- The experimental VL_THREADED setting (only, not normal mode) now requires C++11.
- Fix over-aggressive inlining. (#1223) [John Coiner]
- Fix Ubuntu 17.10 issues. (#1223 partial). [John Coiner]
- Fix compiler warning when WIDTH warning ignored on large compare.
- Fix memory leak in VerilatedVcd dumps. (#1222 partial) [Shareef Jalloq]
- Fix unnecessary Vdly variables. (#1224 partial) [John Coiner]
- Fix conditional slices and add related optimizations.
- Fix &96;&96; expansion of &96;defines. (#1225) (#1227) (#1228) [Odd Magne Reitan]
- Fix -E duplicating output. (#1226) [Odd Magne Reitan]
- Fix float-conversion warning. (#1229) [Robert Henry]
- Fix MacOS portability. (#1230) (#1231) [Jeff Bush]

17.1.41 Verilator 3.912 2017-09-23

Major:

- Verilated headers no longer "use namespace std;" User's code without "std::" prefixes may need "use namespace std;" to compile.

Minor:

- Support or/and/xor array intrinsic methods. (#1210) [Mike Popoloski]
- Support package export. (#1217) [Usuario Eda]
- Support module port parameters without defaults. (#1213) [Mike Popoloski]
- Add performance information to –stats file.
- Simplify VL_CONST_W macro generation for faster compiles.
- Optimize improvements for Shift-And, and replication constructs.
- Fix ordering of arrayed cell wide connections. (#1202 partial) [Mike Popoloski]
- Fix LITENDIAN warning on arrayed cells. (#1202) [Mike Popoloski]
- Fix enum ranges without colons. (#1204) [Mike Popoloski]
- Fix GCC noreturn compile error. (#1209) [Mike Popoloski]
- Fix constant function default parameters. (#1211) [Mike Popoloski]
- Fix non-colon array of interface modports. (#1212) [Mike Popoloski]
- Fix .name connections on interfaces. (#1214) [Mike Popoloski]
- Fix wide array indices causing compile error.

17.1.42 Verilator 3.910 2017-09-07

Major:

- SystemPerl mode (-sp-deprecated) has been removed.

Minor:

- Update keyword warnings to include C++11 and others.

17.1.43 Verilator 3.908 2017-08-28

Minor:

- Support x in $readmem. (#1180) [Arthur Kahlich]
- Support packed struct DPI imports. (#1190) [Rob Stoddard]
- Fix GCC 6 warnings.
- Fix compile error on unused VL_VALUEPLUSARGS_IW. (#1181) [Thomas J Whatson]
- Fix undefined VL_POW_WWI. [Clifford Wolf]
- Fix internal error on unconnected inouts. (#1187) [Rob Stoddard]

17.1.44 Verilator 3.906 2017-06-22

Minor:

- Support set_time_unit/set_time_precision in C traces. (#1937)
- Fix extract of packed array with non-zero LSB. (#1172) [James Pallister]
- Fix shifts by more than 32-bit numbers. (#1174) [Clifford Wolf]
- Fix power operator on wide constants. (#761) [Clifford Wolf]
- Fix .* on interface pins. (#1176) [Maciej Piechotka]

17.1.45 Verilator 3.904 2017-05-30

Minor:

- Fix non-cutable ordering loops on clock arrays. (#1009) [Todd Strader]
- Support ports of array of reals. (#1154) [J Briquet]
- Support arrayed parameter overrides. (#1153) [John Stevenson]
- Support $value$plusargs with variables. (#1165) [Wesley Terpstra]
- Support modport access to un-modport objects. (#1161) [Todd Strader]
- Add stack trace when can't optimize function. (#1158) [Todd Strader]
- Add warning on mis-sized literal. (#1156) [Todd Strader]
- Fix interface functions returning wrong parameters. (#996) [Todd Strader]
- Fix non-arrayed cells with interface arrays. (#1153) [John Stevenson]
- Fix –assert with complex case statements. (#1164) [Enzo Chi]

17.1.46 Verilator 3.902 2017-04-02

Major:

- Add -FI option to force includes. (#1916) [Amir Gonnen]
- Add –relative-includes. [Rob Stoddard]

Minor:

- Add error on duplicate pattern assignments. (#1145) [Johan Bjork]
- Fix error on improperly widthed default function. (#984) [Todd Strader]
- Fix 2009 localparam syntax, msg2139. [Galen Seitz]
- Fix ugly interface-to-non-interface errors. (#1112) [Johan Bjork]
- Fix LDFLAGS and CFLAGS not preserving order. (#1130) [Olof Kindgren]
- Fix internal error on initializing parameter array. (#1131) [Jie Xu]
- Fix internal error on interface arrays. (#1135) [John Stevenson]
- Fix calling sformatf to display, and elab $displays. (#1139) [Johan Bjork]
- Fix realpath compile issue on MSVC++. (#1141) [Miodrag Milanovic]

- Fix missing error on interface size mismatch. (#1143) [Johan Bjork]
- Fix error on parameters with dotted references. (#1146) [Johan Bjork]
- Fix wreal not handling continuous assign. (#1150) [J Briquet]
- Fix nested structure parameter selects. (#1150) [J Briquet]

17.1.47 Verilator 3.900 2017-01-15

Major:

- Internal code changes for improved compatibility and performance.

Minor:

- Support old-style $display($time). (#467) [John Demme]
- With –bbox-unsup, suppress desassign and mixed edges. (#1120) [Galen Seitz]
- Fix parsing sensitivity with &&. (#934) [Luke Yang]
- Fix internal error on double-for loop unrolling. (#1044) [Jan Egil Ruud]
- Fix internal error on unique casez with –assert. (#1117) [Enzo Chi]
- Fix bad code when tracing array of structs. (#1122) [Andrew Bardsley]

17.1.48 Verilator 3.890 2016-11-25

Minor:

- Honor –output-split on coverage constructors. (#1098) [Johan Bjork]
- Fix various issues when making outside of the kit.
- Fix flex 2.6.2 bug. (#1103) [Sergey Kvachonok]
- Fix error on bad interface name. (#1097) [Todd Strader]
- Fix error on referencing variable in parent. (#1099) [Ian Thompson]
- Fix type parameters with low optimization. (#1101) [Stefan Wallentowitz]

17.1.49 Verilator 3.888 2016-10-14

Major:

- Support foreach. (#1078) [Xuan Guo]

Minor:

- Add –no-decoration to remove output comments, msg2015. [Frederic Requin]
- If VM_PARALLEL_BUILDS=1, use OPT_FAST and OPT_SLOW. [Frederic Requin] Set VM_DEFAULT_RULES=0 for old behavior.
- Add error on DPI functions > 32 bits. (#1898) [Elliot Mednick]
- Improve Verilation performance on internal strings. (#1896) [Johan Bjork]
- Improve Verilation performance on trace duplicates. (#1090) [Johan Bjork]
- Fix SystemC compiles with VPI. (#1081) [Arthur Kahlich]

- Fix error on wide numbers that represent shifts, msg1991. (#1088) [Mandy Xu]

17.1.50 Verilator 3.886 2016-07-30

Minor:

- Fix enum values of 11-16 bits wide using .next/.prev. (#1062) [Brian Flachs]
- Fix false warnings on non-power-2 enums using .next/.prev.
- Fix comparison of unpacked arrays. (#1071) [Andrew Bardsley]
- Fix compiler warning in GCC 6. [David Horton]

17.1.51 Verilator 3.884 2016-05-18

Major:

- Support parameter type. (#376) [Alan Hunter, et al]
- Support command-line -G/+pvalue param overrides. (#1045) [Stefan Wallentowitz]
- Add –l2-name option for controlling "v" naming.
- The default l2 scope name is now the same as the top-level module. (#1050) Use "–l2-name v" for the historical behavior.

Minor:

- Fix –output-split of constructors. (#1035) [Johan Bjork]
- Fix removal of empty packages, modules and cells. (#1034) [Johan Bjork]
- Fix core dump on Arch Linux/GCC 6.1.1. (#1058) [Jannis Harder]
- Fix $value$plusargs to string. (#1880) [Frederic Requin]

17.1.52 Verilator 3.882 2016-03-01

Minor:

- Internal Verilation-time performance enhancements. (#1021) [Johan Bjork]
- Support inlining interfaces. (#1018) [Johan Bjork]
- Support SV strings to readmemh. (#1040) [Stefan Wallentowitz]
- Fix unrolling complicated for-loop bounds. (#677) [Johan Bjork]
- Fix stats file containing multiple unroll entries. (#1020) [Johan Bjork]
- Fix using short parameter names on negative params. (#1022) [Duraid Madina]
- Fix read-after-free error. (#1031) [Johan Bjork]
- Fix elaboration-time display warnings. (#1032) [Johan Bjork]
- Fix crash on very deep function trees. (#1028) [Jonathan Kimmitt]
- Fix slicing mix of big and little-endian. (#1033) [Geoff Barrett]
- Fix pattern assignment width propagation. (#1037) [Johan Bjork]

17.1.53 Verilator 3.880 2015-12-19

Minor:

- Support display %u, %v, %p, %z. (#989) [Johan Bjork]
- Fix real parameters causing bad module names. (#992) [Johan Bjork]
- Fix size-changing cast on packed struct. (#993) [Johan Bjork]
- Fix function calls on arrayed interface. (#994) [Johan Bjork]
- Fix arrayed interfaces. (#879) (#1001) [Todd Strader]
- Fix constant function assigned to packed structs. (#997) [Johan Bjork]
- Fix interface inside generate. (#998) [Johan Bjork]
- Fix $signed casts under generates. (#999) [Clifford Wolf]
- Fix genvar constant propagation. (#1003) [Johan Bjork]
- Fix parameter constant propagation from package. (#1004) [Johan Bjork]
- Fix array slicing of non-const indexes. (#1006) [Johan Bjork]
- Fix dotted generated array error. (#1005) [Jeff Bush, Johan Bjork]
- Fix error instead of warning on large concat. (#1865) [Paul Rolfe]
- Fix $bitstoreal constant propagation. (#1012) [Jonathan Kimmitt]
- Fix model restore crash. (#1013) [Jason McMullan]
- Fix arrayed instances to unpacked of same size. (#1015) [Varun Koyyalagunta]
- Fix slices of unpacked arrays with non-zero LSBs.
- Fix ternary operation with unpacked array. (#1017) [Varun Koyyalagunta].

17.1.54 Verilator 3.878 2015-11-01

Major:

- Add –vpi flag, and fix VPI linkage. (#969) [Arthur Kahlich]
- Support genvar indexes into arrayed cells. (#517) [Todd Strader]
- Support $sformatf. (#977) [Johan Bjork]
- Support elaboration assertions. (#973) [Johan Bjork]
- Support $display with non-format arguments. (#467) [Jamey Hicks]

Minor:

- Add VerilatedScopeNameMap for introspection. (#966) [Todd Strader]
- Ignore %l in $display. (#983) [Todd Strader]
- Fix very long module names. (#937) [Todd Strader]
- Fix internal error on dotted refs into generates. (#958) [Jie Xu]
- Fix structure parameter constant propagation. (#968) [Todd Strader]
- Fix enum constant propagation. (#970) [Todd Strader]

- Fix mis-optimizing public DPI functions. (#963) [Wei Song]
- Fix package:scope.scope variable references.
- Fix $fwrite to constant stderr/stdout. (#961) [Wei Song]
- Fix struct.enum.name method calls. (#855) [Jonathon Donaldson]
- Fix dot indexing into arrayed inferfaces. (#978) [Johan Bjork]
- Fix crash in commandArgsPlusMatch. (#987) [Jamie Iles]
- Fix error message on missing interface. (#985) [Todd Strader]

17.1.55 Verilator 3.876 2015-08-12

Minor:

- Add tracing_on, etc to vlt files. (#932) [Frederic Requin]
- Support extraction of enum bits. (#951) [Jonathon Donaldson]
- Fix MinGW compiler error. (#927) (#929) [Hans Tichelaar]
- Fix .c files to be treated as .cpp. (#930) [Jonathon Donaldson]
- Fix string-to-int space conversion. (#931) [Fabrizio Ferrandi]
- Fix dpi imports inside generates. [Michael Tresidder]
- Fix rounding in trace $timescale. (#946) [Frederic Requin]
- Fix $fopen with SV string. (#947) [Sven Stucki]
- Fix hashed error with typedef inside block. (#948) [Sven Stucki]
- Fix makefile with –coverage. (#953) [Eivind Liland]
- Fix coverage documentation. (#954) [Thomas J Whatson]
- Fix parameters with function parameter arguments. (#952) [Jie Xu]
- Fix size casts as second argument of cast item. (#950) [Jonathon Donaldson]

17.1.56 Verilator 3.874 2015-06-06

Minor:

- Add pkg-config .pc file. (#919) [Stefan Wallentowitz]
- Fix installing missing manpages. (#908) [Ahmed El-Mahmoudy]
- Fix sign extension in large localparams. (#910) [Mike Thyer]
- Fix core dump in sync-async warnings. (#911) [Sebastian Dressler]
- Fix truncation warning with -pins-bv. (#912) [Alfonso Martinez]
- Fix Cygwin uint32 compile. (#914) [Matthew Barr]
- Fix preprocessing stringified newline escapes. (#915) [Anton Rapp]
- Fix part-select in constant function. (#916) [Andrew Bardsley]
- Fix width extension on mis-width ports. (#918) [Patrick Maupin]
- Fix width propagation on sized casts. (#925) [Jonathon Donaldson]

- Fix MSVC++ compiler error. (#927) [Hans Tichelaar]

17.1.57 Verilator 3.872 2015-04-05

Minor:

- Add VerilatedVcdFile to allow real-time waveforms. (#890) [HyungKi Jeong]
- Add –clk and related optimizations. (#1840) [Jie Xu]
- Fix order of C style arrays. [Duraid Madina]
- Add –dump-treei-<srcfile>. (#894) [Jie Xu]
- Fix comma-instantiations with parameters. (#884) [Franck Jullien]
- Fix SystemC arrayed bit vectors. (#886) [David Poole]
- Fix compile error on MinGW. (#887) [HyungKi Jeong]

17.1.58 Verilator 3.870 2015-02-12

Minor:

- Suppress COMBDLY when inside always_latch. (#864) [Iztok Jeras]
- Support cast operator with expression size. (#865) [Iztok Jeras]
- Add warning on slice selection out of bounds. (#875) [Cong Van Nguyen].
- Fix member select error broke in 3.868. (#867) [Iztok Jeras]
- Fix $sccanf from string. (#866) [David Pierce]
- Fix VM_PARALLEL_BUILDS broke in 3.868. (#870) [Hiroki Honda]
- Fix non-ANSI modport instantiations. (#868) [Kevin Thompson]
- Fix UNOPTFLAT change detect on multidim arrays. (#872) [Andrew Bardsley]
- Fix slice connections of arrays to ports. (#880) [Varun Koyyalagunta]
- Fix mis-optimizing gate assignments in unopt blocks. (#881) [Mike Thyer]
- Fix sign extension of pattern members. (#882) [Iztok Jeras]
- Fix clang compile warnings.

17.1.59 Verilator 3.868 2014-12-20

Major:

- New verilator_coverage program added to replace SystemPerl's vcoverage.
- PSL support was removed, please use System Verilog assertions.
- SystemPerl mode is deprecated and now untested.

Minor:

- Support enum.first/name and similar methods. (#460) (#848)
- Add 'string' printing and comparisons. (#746) (#747) etc.

- Inline C functions that are used only once. (#1838) [Jie Xu]
- Fix tracing SystemC signals with structures. (#858) [Eivind Liland] Note that SystemC traces will no longer show the signals in the wrapper, they can be seen one level further down.
- Add –stats-vars. (#851) [Jeremy Bennett]
- Fix bare generates in interfaces. (#789) [Bob Newgard]
- Fix underscores in real literals. (#863) [Jonathon Donaldson]

17.1.60 Verilator 3.866 2014-11-15

Minor:

- Fix +define+A+B to define A and B to match other simulators. (#847) [Adam Krolnik]
- Add optimization of wires from arrayed cells. (#1831) [Jie Xu]
- Add optimization of operators between concats. (#1831) [Jie Xu]
- Add public enums. (#833) [Jonathon Donaldson]
- Trace_off now operates on cells. (#826) [Lane Brooks]
- Fix public parameters in unused packages. (#804) [Jonathon Donaldson]
- Fix select when partially out-of-bound. (#823) [Clifford Wolf]
- Fix generate unrolling with function call. (#830) [Steven Slatter]
- Fix cast-to-size context-determined sizing. (#828) [Geoff Barrett]
- Fix not tracing modules following primitives. (#837) [Jie Xu]
- Fix trace overflow on huge arrays. (#834) [Geoff Barrett]
- Fix quoted comment slashes in defines. (#845) [Adam Krolnik]

17.1.61 Verilator 3.864 2014-09-21

Minor:

- Support power operator with real. (#809) [Jonathon Donaldson]
- Improve verilator_profcfunc time attributions. [Jonathon Donaldson]
- Fix duplicate anonymous structures in $root. (#788) [Bob Newgard]
- Fix mis-optimization of bit-swap in wide signal. (#800) [Jie Xu]
- Fix error when tracing public parameters. (#722) [Jonathon Donaldson]
- Fix dpiGetContext in dotted scopes. (#740) [Geoff Barrett]
- Fix over-shift structure optimization error. (#803) [Jeff Bush]
- Fix optional parameter keyword in module #(). (#810) [Iztok Jeras]
- Fix $warning/$error multi-argument ordering. (#816) [Jonathon Donaldson]
- Fix clang warnings. (#818) [Iztok Jeras]
- Fix string formats under deep expressions. (#820) [Iztok Jeras]

17.1.62 Verilator 3.862 2014-06-10

Minor:

- Using command line -Wno-{WARNING} now overrides file-local lint_on.
- Add -P to suppress &96;line and blanks with preprocessing. (#781) [Derek Lockhart]
- Support SV 2012 package import before port list.
- Change SYMRSVDWORD to print as warning rather than error.
- Fix seg-fault with variable of parameterized interface. (#692) [Jie Xu]
- Fix false name conflict on cells in generate blocks. (#749) [Igor Lesik]
- Fix pattern assignment to basic types. (#767) [Jie Xu]
- Fix pattern assignment to conditionals. (#769) [Jie Xu]
- Fix shift corner-cases. (#765) (#766) (#768) (#772) (#774) (#776) [Clifford Wolf]
- Fix C compiler interpreting signing. (#773) [Clifford Wolf]
- Fix late constant division by zero giving X error. (#775) [Clifford Wolf]
- Fix gate primitives with arrays and non-arrayed pins.
- Fix DETECTARRAY error on packed arrays. (#770) [Jie Xu]
- Fix ENDLABEL warnings on escaped identifiers.
- Fix string corruption. (#780) [Derek Lockhart]

17.1.63 Verilator 3.860 2014-05-11

Major:

- PSL is no longer supported, please use System Verilog assertions.
- Support '{} assignment pattern on arrays. (#355)
- Support streaming operators. (#649) [Glen Gibb]
- Fix expression problems with -Wno-WIDTH. (#729) (#736) (#737) (#759) Where WIDTH warnings were ignored this might result in different warning messages and results, though it should better match the spec. [Clifford Wolf]

Minor:

- Add –no-trace-params.
- Add assertions on 'unique if'. (#725) [Jeff Bush]
- Add PINCONNECTEMPTY warning. [Holger Waechtler]
- Support parameter arrays. (#683) [Jeremy Bennett]
- Documentation fixes. (#723) [Glen Gibb]
- Support {} in always sensitivity lists. (#745) [Igor Lesik]
- Fix begin_keywords "1800+VAMS". (#1806)
- Fix tracing of package variables and real arrays.
- Fix tracing of packed arrays without –trace-structs. (#742) [Jie Xu]

- Fix missing coverage line on else-if. (#727) [Sharad Bagri]
- Fix modport function import not-found error.
- Fix power operator calculation. (#730) (#735) [Clifford Wolf]
- Fix reporting struct members as reserved words. (#741) [Chris Randall]
- Fix change detection error on unions. (#758) [Jie Xu]
- Fix -Wno-UNOPTFLAT change detection with 64-bits. (#762) [Clifford Wolf]
- Fix shift-right optimization. (#763) [Clifford Wolf]
- Fix Mac OS-X test issues. [Holger Waechtler]
- Fix C++-2011 warnings.

17.1.64 Verilator 3.856 2014-03-11

Minor:

- Support case inside. (#708) [Jan Egil Ruud]
- Add parameters into trace files. (#706) [Alex Solomatnikov]
- Fix parsing "#0 'b0". (#256)
- Fix array bound checks on real variables.
- Fix –skip-identical mis-detecting on OS-X. (#707)
- Fix missing VL_SHIFTRS_IQI with WIDTH warning. (#714) [Fabrizio Ferrandi]
- Fix signed shift right optimization. (#715) [Fabrizio Ferrandi]
- Fix internal error on "input x =" syntax error. (#716) [Lane Brooks]
- Fix slice extraction from packed array. (#717) [Jan Egil Ruud]
- Fix inside statement EQWILD error. (#718) [Jan Egil Ruud]

17.1.65 Verilator 3.855 2014-01-18

Minor:

- Support modport import. (#696) [Jeremy Bennett]
- Add –trace-structs to show struct names. (#673) [Chris Randall]
- Fix tracing of packed structs. (#705) [Jie Xu]
- Fix –lint-only with MinGW. (#1813) [HyungKi Jeong]
- Fix some delayed assignments of typedefed unpacked arrays.
- Fix wire declarations with size and not range. (#466) [Alex Solomatnikov]
- Fix parameter pin vs. normal pin error. (#704) [Alex Solomatnikov]

17.1.66 Verilator 3.854 2013-11-26

Minor:

- Add UNPACKED warning to convert unpacked structs. [Jeremy Bennett]
- Add –compiler clang to work around compiler bug. (#694) [Stefan Ludwig]
- Support vpi_get of vpiSuppressVal. (#687) [Varun Koyyalagunta]
- Support vpi_get_time. (#688) [Varun Koyyalagunta]
- Fix evaluation of chained parameter functions. (#684) [Ted Campbell]
- Fix enum value extension of '1.
- Fix multiple VPI variable callbacks. (#679) [Rich Porter]
- Fix vpi_get of vpiSize. (#680) [Rich Porter]
- Fix vpi_remove_cb inside callback. (#689) [Varun Koyyalagunta]
- Fix crash with coverage of structures. (#691) [Eivind Liland]
- Fix array assignment from const var. (#693) [Jie Xu]

17.1.67 Verilator 3.853 2013-09-30

Minor:

- Add –no-order-clock-delay to work around #613. [Charlie Brej]

17.1.68 Verilator 3.852 2013-09-29

Minor:

- Support named function and task arguments. [Chris Randall]
- Report SELRANGE warning for non-generate if. (#675) [Roland Kruse]
- Fix ordering of $fgetc. (#1808) [Frederic Requin]
- Fix –output-split-cfunc to count internal functions. [Chris Randall]
- Fix crash on 32-bit Ubuntu. (#670) [Mark Jackson Pulver]

17.1.69 Verilator 3.851 2013-08-15

Minor:

- Fix ordering of clock enables with delayed assigns. (#613) [Jeremy Bennett]
- Fix vpi_iterate on memory words. (#655) [Rich Porter]
- Fix final duplicate declarations when non-inlined. (#661) [Charlie Brej]
- Fix interface ports with comma lists. (#1779) [Ed Lander]
- Fix parameter real conversion from integer.
- Fix clang warnings. (#668) [Yutetsu Takatsukasa]

17.1.70 Verilator 3.850 2013-06-02

Major:

- Support interfaces and modports. (#102) [Byron Bradley, Jeremy Bennett]

Minor:

- Duplicate clock gate optimization on by default. (#621)
- Fix arrayed input compile error. (#645) [Krzysztof Jankowski]
- Fix GCC version runtime changes. (#651) [Jeremy Bennett]
- Fix packed array select internal error. (#652) [Krzysztof Jankowski]

17.1.71 Verilator 3.847 2013-05-11

Minor:

- Add ALWCOMBORDER warning. [KC Buckenmaier]
- Add –pins-sc-uint and –pins-sc-biguint. (#638) [Alex Hornung]
- Support "signal[vec]++".
- Fix simulation error when inputs and MULTIDRIVEN. (#634) [Ted Campbell]
- Fix module resolution with __. (#631) [Jason McMullan]
- Fix packed array non-zero right index select crash. (#642) [Krzysztof Jankowski]
- Fix nested union crash. (#643) [Krzysztof Jankowski]

17.1.72 Verilator 3.846 2013-03-09

Major:

- IEEE 1800-2012 is now the default language. This adds 4 new keywords and updates the svdpi.h and vpi_user.h header files.
- Add –report-unoptflat. (#611) [Jeremy Bennett]

Minor:

- Add duplicate clock gate optimization. (#1772) [Varun Koyyalagunta] Disabled unless -OD or -O3 used, please try it as may get some significant speedups.
- Support pattern assignment features. (#616) (#617) (#618) [Ed Lander]
- Support bind in $unit. (#602) [Ed Lander]
- Support <number>'() sized casts. (#628) [Ed Lander]
- Fix wrong dot resolution under inlining. [Art Stamness]
- Fix DETECTARRAY on packed structures. (#610) [Jeremy Bennett]
- Fix LITENDIAN on unpacked structures. (#614) [Wai Sum Mong]
- Fix 32-bit OS VPI scan issue. (#615) [Jeremy Bennett, Rich Porter]
- Fix opening a VerilatedVcdC file multiple times. (#1774) [Frederic Requin]
- Fix UNOPTFLAT circular array bounds crossing. (#630) [Jie Xu]

17.1.73 Verilator 3.845 2013-02-04

Minor:

- Fix nested packed arrays and struct. (#600) [Jeremy Bennett] Packed arrays are now represented as a single linear vector in Verilated models. This may affect packed arrays that are public or accessed via the VPI.
- Support wires with data types. (#608) [Ed Lander]
- Support bind, to module names only. (#602) [Ed Lander]
- Support VPI product info, warning calls, etc. (#588) [Rick Porter]
- Support $left, $right and related functions. (#448) [Iztok Jeras]
- Support inside expressions.
- Define SYSTEMVERILOG, SV_COV_START and other IEEE mandated predefines.
- Fix pin width mismatch error. (#595) [Alex Solomatnikov]
- Fix implicit one bit parameter selection. (#603) [Jeremy Bennett]
- Fix signed/unsigned parameter misconversion. (#606) [Jeremy Bennett]
- Fix segfault on multidimensional dotted arrays. (#607) [Jie Xu]
- Fix per-bit array output connection error. (#414) [Jan Egil Ruud]
- Fix package logic var compile error.
- Fix enums with X values.

17.1.74 Verilator 3.844 2013-01-09

Minor:

- Support “unsigned int” DPI import functions. (#1770) [Alex Lee]
- Fix package resolution of parameters. (#586) [Jeremy Bennett]
- Fix non-integer vpi_get_value. (#587) [Rich Porter]
- Fix task inlining under $display and case. (#589) (#598) [Holger Waechtler]
- Fix package import of non-localparam parameter. (#474) (#591) [Jeremy Bennett]
- Fix package import of package imports, partial #592. [Jeremy Bennett]
- Fix package import preventing local var. (#599) [Jeremy Bennett]
- Fix array extraction of implicit vars. (#601) [Joe Eiler]

17.1.75 Verilator 3.843 2012-12-01

Minor:

- Add +1364-1995ext and similar language options. (#532) [Jeremy Bennett]
- Fix mis-optimized identical submodule subtract. (#581) [Charlie Brej]
- Fix crash on dotted references into dead modules. (#583) [Jeremy Bennett]
- Fix compile issues on MSVCC. (#571) (#577) [Amir Gonnen]
- Fix –debug overriding preceding –dump-treei. (#580) [Jeremy Bennett]

17.1.76 Verilator 3.842 2012-11-03

Minor:

- Add -x-initial-edge. (#570) [Jeremy Bennett]
- Fix parameter pins interspersed with cells broke in 3.840. [Bernard Deadman]
- Fix large shift error on large shift constants. [David Welch]
- Fix $display mangling on GCC 4.7 and speed up. (#1765) (#373) (#574) [R Diez]
- Fix array of struct references giving false error. (#566) [Julius Baxter]
- Fix missing var access functions when no DPI. (#572) [Amir Gonnen]
- Fix name collision on unnamed blocks. (#567) [Chandan Egbert]
- Fix name collision on task inputs. (#569) [Chandan Egbert]

17.1.77 Verilator 3.841 2012-09-03

Major:

- Add –savable to support model save/restore. [Jeremy Bennett]

Minor:

- Support '{} assignment pattern on structures, part of #355.
- Fix double-deep parameter cell WIDTHs. (#541) [Hiroki Honda]
- Fix imports under multiple instantiated cells. (#542) [Alex Solomatnikov]
- Fix defparam in generate broke in 3.840. (#543) [Alex Solomatnikov]
- Fix duplicate begin error broke in 3.840. (#548) [Alex Solomatnikov]
- Fix triangle symbol resolution error broke in 3.840. (#550) [Ted Campbell]

17.1.78 Verilator 3.840 2012-07-31 Beta

Major:

- Rewrote tristate handling; supports tri0, tri1, tristate bit selects, concatenates and pullup/pulldowns. (#395) (#56) (#54) (#51) [Alex Solomatnikov, Lane Brooks, et al]
- Support packed structures and unions. (#181) Note this was a major internal change that may lead to some instability.

Minor:

- Support tri0 and tri1. (#462) [Alex Solomatnikov]
- Support nmos and pmos. (#488) [Alex Solomatnikov]
- Add INITIALDLY warning on initial assignments. (#478) [Alex Solomatnikov]
- Add PINMISSING and PINNOCONNECT lint checks.
- Add –converge-limit option.
- Fix generate operators not short circuiting. (#413) [by Jeremy Bennett]
- Fix parameters not supported in constant functions. (#474) [Alex Solomatnikov]

- Fix duplicate warnings/errors. (#516) [Alex Solomatnikov]
- Fix signed extending biops with WIDTH warning off. (#511) [Junji Hashimoto]
- Fix ITOD internal error on real conversions. (#491) [Alex Solomatnikov]
- Fix input and real loosing real data type. (#501) [Alex Solomatnikov]
- Fix imports causing symbol table error. (#490) [Alex Solomatnikov]
- Fix newlines in radix values. (#507) [Walter Lavino]
- Fix loop error message to report line. (#513) [Jeremy Bennett]
- Fix false UNUSED warning on file system calls.
- Fix GCC 4.7.0 compile warnings. (#530) [Jeremy Bennett]
- Fix svdpi.h compile error on Apple OS.
- Fix compile error under git submodules. (#534) [Aurelien Francillon]

17.1.79 Verilator 3.833 2012-04-15

Minor:

- Support += and -= in standard for loops. (#463) [Alex Solomatnikov]
- Fix processing unused parameterized modules. (#469) (#470) [Alex Solomatnikov]
- Add SELRANGE as warning instead of error. (#477) [Alex Solomatnikov]
- Add readme.pdf and internal.pdf and doxygen. (#483) [by Jeremy Bennett]
- Fix change detections on arrays. (#364) [John Stevenson, Alex Solomatnikov]
- Fix signed array warning. (#456) [Alex Solomatnikov]
- Fix genvar and begin under generate. (#461) [Alex Solomatnikov]
- Fix real constant parameter functions. (#475) [Alex Solomatnikov]
- Fix and document –gdb option. (#454) [Jeremy Bennett]
- Fix OpenSolaris compile error. [Sanjay Singh]

17.1.80 Verilator 3.832 2012-03-07

Minor:

- Fix memory delayed assignments from multiple clock domains. [Andrew Ling]
- Support arrayed SystemC I/O pins. [Christophe Joly]
- Report MULTIDRIVEN on memories set in multiple clock domains.
- Report ENDLABEL on mismatching end labels. (#450) [Iztok Jeras]
- Fix expansion of back-slashed escaped macros. (#441) [Alberto Del Rio]
- Fix inheriting real and signed type across untyped parameters.
- Fix core dump with over 100 deep UNOPTFLAT. (#432) [Joe Eiler]
- Fix false command not found warning in makefiles. [Ruben Diez]
- Fix hang when functions inside begin block. [David Welch]

- Fix hang on recursive substitution &96;defines. (#443) [Alex Solomatnikov]

17.1.81 Verilator 3.831 2012-01-20

Major:

- Support SystemC 2.3.0 prerelease. This requires setting the new SYSTEMC_INCLUDE and SYSTEMC_LIBDIR variables in place of now deprecated SYSTEMC and SYSTEMC_ARCH. [Iztok Jeras]

Minor:

- Suppress VARHIDDEN on dpi import arguments. [Ruben Diez]
- Support "generate for (genvar i=0; ...". [David Kravitz]
- Fix dpi exports with > 32 bit but < 64 bit args. (#423) [Chandan Egbert]
- Fix array of instantiations with sub-range output. (#414) [Jeremy Bennett]
- Fix BLKSEQ warnings on variables declared inside always. [Ruben Diez]

17.1.82 Verilator 3.830 2011-11-27

Major:

- With "–language VAMS" support a touch of Verilog AMS. [Holger Waechtler]

Minor:

- Add sc_bv attribute to force bit vectors. (#402) [by Stefan Wallentowitz]
- Search for user -y paths before default current directory. [Ruben Diez]
- Support constants in sensitivity lists. (#412) [Jeremy Bennett]
- Support $system. [Ruben Diez]
- Support $sscanf with %g. [Holger Waechtler]
- Indicate 'exiting due to errors' if errors, not warnings. [Ruben Diez]
- Fix bad result with if-else-return optimization. (#420) [Alex Solomatnikov]
- Fix reporting not found modules if generate-off. (#403) [Jeremy Bennett]
- Fix $display with %d following %g. [Holger Waechtler]

17.1.83 Verilator 3.824 2011-10-25

Minor:

- Fix "always @ (*)". (#403) (#404) [Walter Lavino]
- Add ASSIGNIN as suppressible error. [Jeremy Bennett]
- Fix 3.823 constructor core dump on Debian. (#401) [Ahmed El-Mahmoudy]

17.1.84 Verilator 3.823 2011-10-20

Minor:

- Support $ceil, $floor, etc. [Alex Solomatnikov]
- Add configure options for cc warnings and extended tests. [Ruben Diez]
- Add -Wall reporting ASSIGNDLY on assignment delays. [Ruben Diez]
- Fix UNDRIVEN warnings inside DPI import functions. [Ruben Diez]
- Fix –help output to go to stderr, not stdout. (#397) [Ruben Diez]
- Fix DPI import output of 64 bits. (#398) [Mike Denio]
- Fix DPI import false BLKSEQ warnings. [Alex Solomatnikov]
- Fix MSVC compile warning with trunc/round. (#394) [Amir Gonnen]
- Fix autoconf and Makefile warnings. (#396) [Ruben Diez]

17.1.85 Verilator 3.821 2011-09-14

Minor:

- Fix PowerPC runtime error. (#288) [Ahmed El-Mahmoudy]
- Fix internal error on integer casts. (#374) [Chandan Egbert]

17.1.86 Verilator 3.820 2011-07-28

Minor:

- Support ‘real’ numbers and related functions.
- Support ‘const’ variables in limited cases; similar to enums. [Alex Solomatnikov]
- Support disable for loop escapes.
- Support $fopen and I/O with integer instead of &96;verilator_file_descriptor.
- Support coverage in -cc and -sc output modes. [John Li] Note this requires SystemPerl 1.338 or newer.
- Use ‘vluint64_t’ for SystemC instead of (same sized) ‘uint64’ for MSVC++.
- Fix vpi_register_cb using bad s_cb_data. (#370) [by Thomas Watts]
- Fix $display missing leading zeros in %0d. (#367) [Alex Solomatnikov]

17.1.87 Verilator 3.813 2011-06-28

Minor:

- Support bit vectors > 64 bits wide in DPI import and exports.
- Fix out of memory on slice syntax error. (#354) [Alex Solomatnikov]
- Fix error on enum references to other packages. (#339) [Alex Solomatnikov]
- Fix DPI undeclared svBitVecVal compile error. (#346) [Chandan Egbert]
- Fix DPI bit vector compile errors. (#347) (#359) [Chandan Egbert]

- Fix CDCRSTLOGIC report showing endpoint flops without resets.
- Fix compiler warnings on SPARC. (#288) [Ahmed El-Mahmoudy]

17.1.88 Verilator 3.812 2011-04-06

Minor:

- Add –trace-max-width and –trace-max-array. (#319) [Alex Solomatnikov]
- Add –Wno-fatal to turn off abort on warnings. [by Stefan Wallentowitz]
- Support ${...} and $(...) env vars in .vc files. [by Stefan Wallentowitz]
- Support $bits(data_type). (#327) [Alex Solomatnikov]
- Support loop unrolling on width mismatches. (#333) [Joe Eiler]
- Support simple cast operators. (#335) [Alex Solomatnikov]
- Accelerate bit-selected inversions.
- Add error on circular parameter definitions. (#329) [Alex Solomatnikov]
- Fix concatenates and vectored bufif1. (#326) [Iztok Jeras]

17.1.89 Verilator 3.811 2011-02-14

Minor:

- Report error on duplicated or empty pins. (#321) [Christian Leber]
- Report error on function call output tied to constant. [Bernard Deadman]
- Throw UNUSED/UNDRIVEN only once per net in a parameterized module.
- Fix internal error on functions called as SV tasks. [Bernard Deadman]
- Fix internal error on non-inlined inout pins. [Jeff Winston]
- Fix false BLKSEQ on non-unrolled for loop indexes. [Jeff Winston]
- Fix block comment not separating identifiers. (#311) [Gene Sullivan]
- Fix warnings to point to lowest net usage, not upper level ports.
- Fix error on constants connected to outputs. (#323) [Christian Leber]

17.1.90 Verilator 3.810 2011-01-03

Major:

- Add limited support for VPI access to public signals, see docs.
- Add -F option to read relative option files. (#297) [Neil Hamilton]
- Support ++,–,+= etc as standalone statements. [Alex Solomatnikov]
- Add -Wall, -Wwarn-style, -Wno-style to enable code style warnings that have been added to this release, and disabled by default:
- With –Wall, add BLKSEQ warning on blocking assignments in seq blocks.
- With –Wall, add DECLFILENAME warning on modules not matching filename.

- With –Wall, add DEFPARAM warning on deprecated defparam statements.
- With –Wall, add IFDEPTH warning on deep if statements.
- With –Wall, add INCABSPATH warning on &96;include with absolute paths.
- With –Wall, add SYNCASYNCNET warning on mixed sync/async reset nets.
- With –Wall, add UNDRIVEN warning on undriven nets.
- With –Wall, add UNUSED warning on unused nets.

Minor:

- When running with VERILATOR_ROOT, optionally find binaries under bin.
- Suppress WIDTH warnings when adding/subtracting 1'b1.
- The VARHIDDEN warning is now disabled by default, use -Wall to enable.

17.1.91 Verilator 3.805 2010-11-02

Minor:

- Add warning when directory contains spaces. (#1705) [Salman Sheikh]
- Fix wrong filename on include file errors. (#289) [Brad Parker]
- Fix segfault on SystemVerilog "output wire foo=0". (#291) [Joshua Wise]
- Fix DPI export name not found. (#1703) [Terry Chen]

17.1.92 Verilator 3.804 2010-09-20

Minor:

- Support tracing/coverage of underscore signals. (#280) [by Jason McMullan]
- Increase define recursions before error. [Paul Liu]
- On core dump, print debug suggestions.
- Fix preprocessor &96;&96; of existing base define. (#283) [Usha Priyadharshini]

17.1.93 Verilator 3.803 2010-07-10

Minor:

- Fix preprocessor preservation of newlines across macro substitutions.
- Fix preprocessor stringification of nested macros.
- Fix some constant parameter functions causing crash. (#253) [Nick Bowler]
- Fix do {. . . } while() not requiring final semicolon.

17.1.94 Verilator 3.802 2010-05-01

Minor:

- Support runtime access to public signal names.
- Add */verilator public_flat_rw/* for timing-specific public access.
- Fix word size to match uint64_t on -m64 systems. (#238) [Joe Eiler]
- Improve error handling on slices of arrays. (#226) [by Byron Bradley]
- Report errors when extra underscores used in meta-comments.
- Fix bit reductions on multi-packed dimensions. (#227) [by Byron Bradley]
- Fix removing $fscanf if assigned to unused var. (#248) [Ashutosh Das]
- Fix "make install" with configure outside srcdir. [Stefan Wallentowitz]
- Fix loop unroller out of memory; change –unroll-stmts. [Ashutosh Das]
- Fix trace files with empty modules crashing some viewers.
- Fix parsing single files > 2GB. [Jeffrey Short]
- Fix installing data files as non-executable. (#168) [by Ahmed El-Mahmoudy]

17.1.95 Verilator 3.801 2010-03-17

Minor:

- Support "break", "continue", "return".
- Support "&96;default_nettype none|wire". [Dominic Plunkett]
- Skip SystemC tests if not installed. [Iztok Jeras]
- Fix clock-gates with non-AND complex logic. (#220) [Ashutosh Das]
- Fix flushing VCD buffers on $stop. [Ashutosh Das]
- Fix Mac OS-X compile issues. (#217) [Joshua Wise, Trevor Williams]
- Fix make uninstall. (#216) [Iztok Jeras]
- Fix parameterized defines with empty arguments.

17.1.96 Verilator 3.800 2010-02-07

Major application visible changes:

- SystemPerl is no longer required for tracing. Applications must use VerilatedVcdC class in place of SpTraceVcdC.
- SystemVerilog 1800-2009 is now the default language. Thus "global" etc are now keywords. See the –language option.

Major new features:

- Support SystemVerilog types "byte", "chandle", "int", "longint", "shortint", "time", "var" and "void" in variables and functions.
- Support "program", "package", "import" and $unit.

- Support typedef and enum. [by Donal Casey]
- Support direct programming interface (DPI) "import" and "export". Includes an extension to map user $system PLI calls to the DPI.
- Support assignments of multidimensional slices. (#170) [by Byron Bradley]
- Support multidimensional inputs/outputs. (#171) [by Byron Bradley]
- Support "reg [1:0][1:0][1:0]" and "reg x [3][2]". (#176) [Byron Bradley]
- Support declarations in loop initializers. (#172) [by Byron Bradley]
- Support $test$plusargs and $value$plusargs, but see the docs!
- Support $sformat and $swrite.
- Support 1800-2009 define defaults and &96;undefineall.
- Add -CFLAGS, -LDFLAGS, <file>.a, <file>.o, and <file>.so options.
- Speed compiles by avoiding including the STL iostream header. Application programs may need to include it themselves to avoid errors.
- Add experimental clock domain crossing checks.
- Add experimental –pipe-filter to filter all Verilog input.
- Add experimental config files to filter warnings outside of the source.
- Add VARHIDDEN warning when signal name hides module name.
- Support optional cell parenthesis. (#179) [by Byron Bradley]
- Support for-loop i++, ++i, i–, –i. (#175) [by Byron Bradley]
- Support 1800-2009 /*comments*/ in define values.
- Add Makefile VM_GLOBAL_FAST, listing objects needed to link executables.
- Add –bbox-unsup option to black-box unsupported UDP tables.
- Add -Wno-MODDUP option to allow duplicate modules.

Bug fixes:

- Fix implicit variable issues. (#196) (#201) [Byron Bradley]
- Fix 'for' variable typing. (#205) [by Byron Bradley]
- Fix tracing with –pins-bv 1. (#195) [Michael S]
- Fix MSVC++ 2008 compile issues. (#209) [Amir Gonnen]
- Fix MinGW compilation. (#184) (#214) [by Shankar Giri, Amir Gonnen]
- Fix Cygwin 1.7.x compiler error with uint32_t. (#204) [Ivan Djordjevic]
- Fix &96;define argument mis-replacing system task of same name. (#191)
- Fix Verilator core dump on wide integer divides. (#178) [Byron Bradley]
- Fix lint_off/lint_on meta comments on same line as warning.

17.1.97 Verilator 3.720 2009-10-26

Major:

- Support little endian bit vectors ("reg [0:2] x;").
- Support division and modulus of > 64 bit vectors. [Gary Thomas]

Minor:

- Fix writing to out-of-bounds arrays writing element 0.
- Fix core dump with SystemVerilog var declarations under unnamed begins.
- Fix VCD files showing internal flattened hierarchy, broke in 3.714.
- Fix cell port connection to unsized integer causing false width warning.
- Fix erroring on strings with backslashed newlines. (#168) [Pete Nixon]

17.1.98 Verilator 3.714 2009-09-18

Major:

- Add –bbox-sys option to blackbox $system calls.

Minor:

- Support generate for var++, var–, ++var, –var.
- Improved warning when "do" used as identifier.
- Don't require SYSTEMPERL_INCLUDE if SYSTEMPERL/src exists. [Gary Thomas]
- Fix deep defines causing flex scanner overflows. [Brad Dobbie]
- Fix preprocessing commas in deep parameterized macros. [Brad Dobbie]
- Fix tracing escaped dotted identifiers. (#107)
- Fix $display with uppercase %M.
- Fix –error-limit option being ignored.

17.1.99 Verilator 3.713 2009-08-04

Minor:

- Support constant function calls for parameters. [many!]
- Support SystemVerilog "logic". (#101) [by Alex Duller]
- Name SYMRSVDWORD error, and allow disabling it. (#103) [Gary Thomas]
- Fix escaped preprocessor identifiers. (#106) [Nimrod Gileadi]

17.1.100 Verilator 3.712 2009-07-14

Major:

- Patching SystemC is no longer required to trace sc_bvs.

Minor:

- Add verilator –pins-uint8 option to use sc_in<uint8_t/uint16_t>.
- Add verilator -V option, to show verbose version.
- Add BLKLOOPINIT error code, and describe –unroll-count. [Jeff Winston]
- Support zero-width constants in concatenations. [Jeff Winston]
- On WIDTH warnings, show variable name causing error. [Jeff Winston]

17.1.101 Verilator 3.711 2009-06-23

Minor:

- Support decimal constants of arbitrary widths. [Mark Marshall]
- Fix error on case statement with all duplicate items. (#99) [Gary Thomas]
- Fix segfault on unrolling for's with bad inits. (#90) [Andreas Olofsson]
- Fix tristates causing "Assigned pin is neither...". [by Lane Brooks]
- Fix compiler errors under Fedora release candidate 11. [Chitlesh Goorah]

17.1.102 Verilator 3.710 2009-05-19

Major:

- Verilator is now licensed under LGPL v3 and/or Artistic v2.0.

Minor:

- &96;__FILE__ now expands to a string, per draft SystemVerilog 2010(ish).
- The front end parser has been re-factored to enable more SV parsing. Code should parse the same, but minor parsing bugs may pop up.
- Verilator_includer is no longer installed twice. (#48) [Lane Brooks]
- Fix escaped identifiers with '.' causing conflicts. (#83) [J Baxter]
- Fix define formal arguments that contain newlines. (#84) [David A]

17.1.103 Verilator 3.703 2009-05-02

Minor:

- Fix $clog2 calculation error with powers-of-2. (#81) [Patricio Kaplan]
- Fix error with tasks that have output first. (#78) [Andrea Foletto]
- Fix "cloning" error with -y/–top-module. (#76) [Dimitris Nalbantis]
- Fix segfault with error on bad –top-module. (#79) [Dimitris Nalbantis]

- Fix “redefining I” error with complex includes. [Duraid Madina]
- Fix GCC 4.3.2 compile warnings.

17.1.104 Verilator 3.702 2009-03-28

Minor:

- Add –pins-bv option to use sc_bv for all ports. [Brian Small]
- Add SYSTEMPERL_INCLUDE envvar to assist RPM builds. [Chitlesh Goorah]
- Report errors when duplicate labels are used. (#72) [Vasu Kandadi]
- Fix the SC_MODULE name() to not include __PVT__. [Bob Fredieu]

17.1.105 Verilator 3.701 2009-02-26

Minor:

- Support repeat and forever statements. [Jeremy Bennett]
- Add –debugi-<srcfile> option, for internal debugging. [Dennis Muhlestein]
- Fix compile issues with GCC 4.3. (#47) [Lane Brooks]
- Fix VL_RANDom to better randomize bits. [Art Stamness]
- Fix error messages to consistently go to stderr. [Jeremy Bennett]
- Fix left associativity for ?: operators.

17.1.106 Verilator 3.700 2009-01-08

Major:

- Support limited tristate inouts. Written by Lane Brooks, under support by Ubixum Inc. This allows common pad ring and tristate-mux structures to be Verilated. See the documentation for more information on supported constructs.
- Add –coverage_toggle for toggle coverage analysis. Running coverage now requires SystemPerl 1.301 or newer.
- Add coverage_on/_off metacomments to bracket coverage regions.

Minor:

- Support posedge of bit-selected signals. (#45) [Rodney Sinclair]
- Optimize two-level shift and and/or trees, +23% on one test.
- Line coverage now aggregates by hierarchy automatically. Previously this would be done inside SystemPerl, which was slower.
- Minor performance improvements of Verilator compiler runtime.
- Coverage of each parameterized module is counted separately. [Bob Fredieu]
- Fix creating parameterized modules when no parameter values are changed.
- Fix certain generate-if cells causing “clone” error. [Stephane Laurent]
- Fix line coverage of public functions. [Soon Koh]

- Fix SystemC 2.2 deprecated warnings about sensitive() and sc_start().
- Fix arrayed variables under function not compiling. (#44) [Ralf Karge]
- Fix –output-split-cfuncs to also split trace code. [Niranjan Prabhu]
- Fix 'bad select range' warning missing some cases. (#43) [Lane Brooks]
- Fix internal signal names containing control characters (broke in 3.680).
- Fix compile error on Ubuntu 8.10. [Christopher Boumenot]
- Fix internal error on "output x; reg x = y;".
- Fix wrong result for read of delayed FSM signal. (#46) [Rodney Sinclair]

17.1.107 Verilator 3.681 2008-11-12

Minor:

- Support SystemVerilog unique and priority case.
- Include Verilog file's directory name in coverage reports.
- Fix 'for' under 'generate-for' causing error. (#38) [Rafael Shirakawa]
- Fix coverage hierarchy being backwards with inlining. [Vasu Arasanipalai]
- Fix GCC 4.3 compile error. (#35) [Lane Brooks]
- Fix MSVC compile error. (#42) [John Stroebel]

17.1.108 Verilator 3.680 2008-10-08

Major:

- Support negative bit indexes. [Stephane Laurent] Tracing negative indexes requires latest Verilog-Perl and SystemPerl.

Minor:

- Suppress width warnings between constant strings and wider vectors. [Rodney Sinclair]
- Ignore SystemVerilog timeunit and timeprecision.
- Expand environment variables in -f input files. [Lawrence Butcher]
- Report error if port declaration is missing. (#32) [Guy-Armand Kamendje]
- Fix genvars causing link error when using –public. [Chris Candler]

17.1.109 Verilator 3.671 2008-09-19

Major:

- SystemC uint64_t pins are now the default instead of sc_bv<64>. Use –no-pins64 for backward compatibility.
- Support SystemVerilog "cover property" statements.

Minor:

- When warnings are disabled on signals that are flattened out, disable the warnings on the signal(s) that replace it.

- Add by-design and by-module subtotals to verilator_profcfunc.
- Add IMPERFECTSCH warning, disabled by default.
- Support coverage under SystemPerl 1.285 and newer.
- Support arbitrary characters in identifiers. [Stephane Laurent]
- Fix extra evaluation of pure combo blocks in SystemC output.
- Fix stack overflow on large ? : trees. [John Sanguinetti]

17.1.110 Verilator 3.670 2008-07-23

Major:

- Add –x-assign=fast option, and make it the default. This chooses performance over reset debugging. See the manual.
- Add –autoflush, for flushing streams after $display. [Steve Tong]
- Add CASEWITHX lint warning and if disabled fix handling of casez with Xs.

Minor:

- Add $feof, $fgetc, $fgets, $fflush, $fscanf, $sscanf. [Holger Waechtler]
- Add $stime. [Holger Waechtler]
- Add $random.
- Add –Wfuture-, for improving forward compatibility.
- Add WIDTH warning to $fopen etc file descriptors.
- Fix verilator_includer not being installed properly. [Holger Waechtler]
- Fix IMPURE errors due to X-assignment temporary variables. [Steve Tong]
- Fix “lvalue” errors with public functions. (#25) [CY Wang]

17.1.111 Verilator 3.665 2008-06-25

Minor:

- Ignore “// verilator” comments alone on endif lines. [Rodney Sinclair]
- “Make install” now installs verilator_includer and verilator_profcfunc.
- Fix tracing missing changes on undriven public wires. [Rodney Sinclair]
- Fix syntax error when “&96;include &96;defname” is ifdefed. [John Dickol]
- Fix error when macro call has commas in concatenate. [John Dickol]
- Fix compile errors under Fedora 9, GCC 4.3.0. [by Jeremy Bennett]
- Fix Makefile to find headers/libraries under prefix. [by Holger Waechtler]

17.1.112 Verilator 3.664 2008-05-08

Minor:

- Fix missing file in kit.

17.1.113 Verilator 3.663 2008-05-07

Minor:

- Add DESTDIR to Makefiles to assist RPM construction. [Gunter Dannoritzer]
- Fix compiler warnings under GCC 4.2.1.
- Fix preprocessor &96;else after series of &96;elsif. [Mark Nodine]
- Fix parameterized defines calling define with comma. [Joshua Wise]
- Fix comma separated list of primitives. [by Bryan Brady]

17.1.114 Verilator 3.662 2008-04-25

Minor:

- Add Verilog 2005 $clog2() function. This is useful in calculating bus-widths from parameters.
- Support C-style comments in -f option files. [Stefan Thiede]
- Add error message when modules have duplicate names. [Stefan Thiede]
- Support defines terminated in EOF, though against spec. [Stefan Thiede]
- Support optional argument to $finish and $stop. [by Stefan Thiede]
- Support ranges on gate primitive instantiations. [Stefan Thiede]
- Ignore old standard(ish) Verilog-XL defines. [by Stefan Thiede]
- Fix "always @ ((a) or (b))" syntax error. [by Niranjan Prabhu]
- Fix "output reg name=expr;" syntax error. [Martin Scharrer]
- Fix multiple .v files being read in random order. [Stefan Thiede]
- Fix internal error when params get non-constants. [Johan Wouters]
- Fix bug introduced in 3.661 with parameterized defines.

17.1.115 Verilator 3.661 2008-04-04

Major:

- The –enable-defenv configure option added in 3.660 is now the default. This hard-codes a default for VERILATOR_ROOT etc in the executables.
- Add –language option for supporting older code. [Stefan Thiede]
- Add –top-module option to select between multiple tops. [Stefan Thiede]

Minor:

- Unsized concatenates now give WIDTHCONCAT warnings. [Jonathan Kimmitt] Previously they threw fatal errors, which in most cases is correct according to spec, but can be incorrect in presence of parameter values.

- Support functions with "input integer". [Johan Wouters]
- Ignore delays attached to gate UDPs. [Stefan Thiede]
- Fix SystemVerilog parameterized defines with &96;&96; expansion, and fix extra whitespace inserted on substitution. [Vladimir Matveyenko]
- Fix no-module include files on command line. [Stefan Thiede]
- Fix dropping of backslash quoted-quote at end of $display.
- Fix task output pin connected to non-variables. [Jonathan Kimmitt]
- Fix missing test_v in install datadir. [Holger Waechtler]
- Fix internal error after MSB < LSB error reported to user. [Stefan Thiede]

17.1.116 Verilator 3.660 2008-03-23

Minor:

- Support hard-coding VERILATOR_ROOT etc in the executables, to enable easier use of Verilator RPMs. [Gunter Dannoritzer]
- Allow multiple .v files on command line. [Stefan Thiede]
- Convert re-defining macro error to warning. [Stefan Thiede]
- Add –error-limit option. [Stefan Thiede]
- Allow __ in cell names by quoting them in C. [Stefan Thiede]
- Fix genvar to be signed, so "< 0" works properly. [Niranjan Prabhu]
- Fix assignments to inputs inside functions/tasks. [Patricio Kaplan]
- Fix definitions in main file.v, referenced in library. [Stefan Thiede]
- Fix undefined assigns to be implicit warnings. [Stefan Thiede]

17.1.117 Verilator 3.658 2008-02-25

Minor:

- Fix unistd compile error in 3.657. [Patricio Kaplan, Jonathan Kimmitt]

17.1.118 Verilator 3.657 2008-02-20

Minor:

- Fix assignments of {a,b,c} = {c,b,a}. [Jonathan Kimmitt]
- Fix Perl warning with –lint-only. [by Ding Xiaoliang]
- Fix to avoid creating obj_dir with –lint-only. [Ding Xiaoliang]
- Fix parsing of always @(*). [Patricio Kaplan]

17.1.119 Verilator 3.656 2008-01-18

Minor:

- Wide VL_CONST_W_#X functions are now made automatically. [Bernard Deadman] In such cases, a new {prefix}__Inlines.h file will be built and included.
- Fix sign error when extracting from signed memory. [Peter Debacker]
- Fix tracing of SystemC w/o SystemPerl. [Bernard Deadman, Johan Wouters]

17.1.120 Verilator 3.655 2007-11-27

Minor:

- Support "#delay <statement>;" with associated STMTDLY warning.
- Fix generate for loops with constant zero conditions. [Rodney Sinclair]
- Fix divide-by-zero errors in constant propagator. [Rodney Sinclair]
- Fix wrong result with obscure signed-shift underneath a "? :".
- Fix many internal memory leaks, and added leak detector.

17.1.121 Verilator 3.654 2007-10-18

Minor:

- Don't exit early if many warnings but no errors are found. [Stan Mayer]
- Fix parsing module #(parameter x,y) declarations. [Oleg Rodionov]
- Fix parsing system functions with empty parens. [Oleg Rodionov]

17.1.122 Verilator 3.653 2007-08-01

Minor:

- Support SystemVerilog ==? and !=? operators.
- Fix SC_LIBS missing from generated makefiles. [Ding Xiaoliang]

17.1.123 Verilator 3.652 2007-06-21

Minor:

- Report as many warning types as possible before exiting.
- Support V2K portlists with "input a,b,...". [Mark Nodine]
- Support V2K function/task argument lists.
- Optimize constant $display arguments.
- Fix preprocessor dropping some &96;line directives. [Mark Nodine]

17.1.124 Verilator 3.651 2007-05-22

Major:

- Add verilator_profcfunc utility. [Gene Weber]

Minor:

- Treat modules within &96;celldefine and &96;endcelldefine as if in library.
- Support functions which return integers. [Mark Nodine]
- Warn if flex is not installed. [Ralf Karge]
- Ignore &96;protect and &96;endprotect.
- Fix empty case/endcase blocks.

17.1.125 Verilator 3.650 2007-04-20

Major:

- Add –compiler msvc option. This is now required when Verilated code is to be run through MSVC++. This also enables fixing MSVC++ error C1061, blocks nested too deeply. [Ralf Karge]
- Add –lint-only option, to lint without creating other output.

Minor:

- Add */verilator lint_save/* and */verilator lint_restore/* to allow friendly control over re-enabling lint messages. [Gerald Williams]
- Support SystemVerilog .name and .* interconnect.
- Support while and do-while loops.
- Use $(LINK) instead of $(CXX) for Makefile link rules. [Gerald Williams]
- Add USER_CPPFLAGS and USER_LDFLAGS to Makefiles. [Gerald Williams]
- Fix compile errors under Windows MINGW compiler. [Gerald Williams]
- Fix dotted bit reference to local memory. [Eugene Weber]
- Fix 3.640 &96;verilog forcing IEEE 1364-1995 only. [David Hewson]

17.1.126 Verilator 3.640 2007-03-12

Minor:

- Support Verilog 2005 &96;begin_keywords and &96;end_keywords.
- Updated list of SystemVerilog keywords to correspond to IEEE 1800-2005.
- Add */verilator public_flat/*. [Eugene Weber]
- Try all +libext's in the exact order given. [Michael Shinkarovsky]
- Fix elimination of public signals assigned to constants. [Eugene Weber]
- Fix internal error when public for loop has empty body. [David Addison]
- Fix "Loops detected" assertion when model exceeds 4GB. [David Hewson]
- Fix display %m names inside named blocks.

17.1.127 Verilator 3.633 2007-02-07

Minor:

- Add –trace-depth option for minimizing VCD file size. [Emerson Suguimoto]
- With VL_DEBUG, show wires causing convergence errors. [Mike Shinkarovsky]
- Fix isolate_assignments when many signals per always. [Mike Shinkarovsky]
- Fix isolate_assignments across task/func temporaries. [Mike Shinkarovsky]
- Fix $display's with array select followed by wide AND. [David Hewson]

17.1.128 Verilator 3.632 2007-01-17

Minor:

- Add */verilator isolate_assignments/* attribute. [Mike Shinkarovsky]

17.1.129 Verilator 3.631 2007-01-02

Major:

- Support standard NAME[#] for cells created by arraying or generate for. This replaces the non-standard name__# syntax used in earlier versions.

Minor:

- Fix again dotted references into generate cells. [David Hewson] Verilator no longer accepts duplicated variables inside unique generate blocks as this is illegal according to the specification.
- Fix $readmem* with filenames < 8 characters. [Emerson Suguimoto]

17.1.130 Verilator 3.630 2006-12-19

Major:

- Support $readmemb and $readmemh. [Eugene Weber, Arthur Kahlich]

Minor:

- When dotted signal lookup fails, help the user by showing known scopes.
- Fix to reduce depth of priority encoded case statements. [Eugene Weber]
- Fix configure and compiling under Solaris. [Bob Farrell]
- Fix dotted references inside generated cells. [David Hewson]
- Fix missed split optimization points underneath other re-split blocks.

17.1.131 Verilator 3.623 2006-12-05

Major:

- Add –output-split-cfuncs for accelerating GCC compile. [Eugene Weber]

Minor:

- Add M32 make variable to support -m32 compiles. [Eugene Weber]
- Fix $signed mis-extending when input has a WIDTH violation. [Eugene Weber]

17.1.132 Verilator 3.622 2006-10-17 Stable

Minor:

- Fix –skip-identical without –debug, broken in 3.621. [Andy Meier]

17.1.133 Verilator 3.621 2006-10-11 Beta

Major:

- Add */verilator no_inline_task/* to prevent over-expansion. [Eugene Weber]

Minor:

- Public functions now allow > 64 bit arguments.
- Remove .vpp intermediate files when not under –debug.
- Fix link error when using –exe with –trace. [Eugene Weber]
- Fix mis-optimization of wide concats with constants.
- Fix core dump on printing error when not under –debug. [Allan Cochrane]

17.1.134 Verilator 3.620 2006-10-04 Stable

Minor:

- Support simple inout task ports. [Eugene Weber]
- Allow overriding Perl, Flex and Bison versions. [by Robert Farrell]
- Optimize variables set to constants within basic blocks for ~3%.
- Default make no longer makes the docs; if you edit the documentation. sources, run “make info” to get them.
- Optimize additional Boolean identities (ala = a, etc.)
- Fix coredump when dotted cross-ref inside task call. [Eugene Weber]
- Fix dotted variables in always sensitivity lists. [Allan Cochrane]

17.1.135 Verilator 3.610 2006-09-20 Stable

Minor:

- Verilator now works under DJGPP (Pentium GCC). [John Stroebel]
- Add default define for VL_PRINTF. [John Stroebel]
- Removed coverage request variable; see Coverage limitations in docs.
- Fix DOS carriage returns in multiline defines. [Ralf Karge]
- Fix printf format warnings on 64-bit linux.

17.1.136 Verilator 3.602 2006-09-11 Stable

Minor:

- Fix function references under top inlined module. [David Hewson]

17.1.137 Verilator 3.601 2006-09-06 Beta

Major:

- Add –inhibit-sim flag for environments using old __Vm_inhibitSim.
- Add &96;systemc_dtor for destructor extensions. [Allan Cochrane]
- Add -MP to make phony dependencies, ala GCC's.

Minor:

- Changed how internal functions are invoked to reduce aliasing. Useful when using GCC's -O2 or -fstrict-aliasing, to gain another ~4%.
- Declare optimized lookup tables as 'static', to reduce D-Cache miss rate.
- Fix memory leak when destroying modules. [John Stroebel]
- Fix coredump when unused modules have unused cells. [David Hewson]
- Fix 3.600 internal error with arrayed instances. [David Hewson]
- Fix 3.600 internal error with non-unrolled function loops. [David Hewson]
- Fix $display %m name not matching Verilog name inside SystemC modules.

17.1.138 Verilator 3.600 2006-08-28 Beta

Major:

- Support dotted cross-hierarchy variable and task references.

Minor:

- Lint for x's in generate case statements.
- Fix line numbers being off by one when first file starts with newline.
- Fix naming of generate for blocks to prevent non-inline name conflict.
- Fix redundant statements remaining after table optimization.

17.1.139 Verilator 3.542 2006-08-11 Stable

Minor:

- vl_finish and vl_fatal now print via VL_PRINTF rather then cerr/cout.
- Fix extraneous UNSIGNED warning when comparing genvars. [David Hewson]
- Fix extra white space in $display %c. [by David Addison]
- Fix missing VL_CONST_W_24X macro. [Bernard Deadman]

17.1.140 Verilator 3.541 2006-07-05 Beta

Minor:

- Add warning on changeDetect to arrayed structures. [David Hewson]
- Fix "// verilator lint_on" not re-enabling warnings. [David Hewson]
- Fix 3.540's multiple memory assignments to same block. [David Hewson]
- Fix non-zero start number for arrayed instantiations. [Jae Hossell]
- Fix GCC 4.0 header file warnings.

17.1.141 Verilator 3.540 2006-06-27 Beta

Minor:

- Optimize combo assignments that are used only once, ~5-25% faster.
- Optimize delayed assignments to memories inside loops, ~0-5% faster.
- Fix mis-width warning on bit selects of memories. [David Hewson]
- Fix mis-width warning on dead generate-if branches. [Jae Hossell]

17.1.142 Verilator 3.533 2006-06-05 Stable

Minor:

- Add PDF user manual, verilator.pdf.
- Fix delayed bit-selected arrayed assignments. [David Hewson]
- Fix execution path to Perl. [Shanshan Xu]
- Fix Bison compile errors in verilog.y. [by Ben Jackson]

17.1.143 Verilator 3.531 2006-05-10 Stable

Minor:

- Support $c routines which return 64 bit values.
- Fix &96;include &96;DEFINE.
- Fix Verilator core dump when have empty public function. [David.Hewson]

17.1.144 Verilator 3.530 2006-04-24 Stable

Major:

- $time is now 64 bits. The macro VL_TIME_I is now VL_TIME_Q, but calls the same sc_time_stamp() function to get the current time.

17.1.145 Verilator 3.523 2006-03-06 Stable

Minor:

- Fix error line numbers being off due to multi-line defines. [Mat Zeno]
- Fix GCC sign extending (uint64_t)(a<b). [David Hewson]
- Fix &96;systemc_imp_header "undefined macro" error.

17.1.146 Verilator 3.522 2006-02-23 Beta

Minor:

- Add UNUSED error message, for forward compatibility.

17.1.147 Verilator 3.521 2006-02-14 Beta

Major:

- Create new –coverage-line and –coverage-user options. [Peter Holmes]

Minor:

- Add SystemVerilog 'x,'z,'0,'1, and new string literals.
- Fix public module's parent still getting inlined.

17.1.148 Verilator 3.520 2006-01-14 Stable

Major:

- Support $fopen, $fclose, $fwrite, $fdisplay. See documentation, as the file descriptors differ from the standard.

17.1.149 Verilator 3.510 2005-12-17 Stable

Major:

- Improve trace-on performance on large multi-clock designs by 2x or more. This adds a small ~2% performance penalty if traces are compiled in, but not turned on. For best non-tracing performance, do not use –trace.

Minor:

- Fix $'s in specify delays causing bad PLI errors. [Mat Zeno]
- Fix public functions not setting up proper symbol table. [Mat Zeno]
- Fix genvars generating trace compile errors. [Mat Zeno]
- Fix VL_MULS_WWW compile error with MSVC++. [Wim Michiels]

17.1.150 Verilator 3.502 2005-11-30 Stable

Minor:

- Fix local non-IO variables in public functions and tasks.
- Fix bad lifetime optimization when same signal is assigned multiple times in both branch of a if. [Danny Ding]

17.1.151 Verilator 3.501 2005-11-16 Stable

Major:

- Add –prof-cfuncs for correlating profiles back to Verilog.

Minor:

- Fix functions where regs are declared before inputs. [Danny Ding]
- Fix bad deep expressions with bit-selects and rotate. [Prabhat Gupta]

17.1.152 Verilator 3.500 2005-10-30 Stable

Major:

- Support signed numbers, >>>, $signed, $unsigned. [MANY!]
- Support multi-dimensional arrays. [Eugen Fekete]
- Support very limited Property Specification Language (aka PSL or Sugar). The format and keywords are now very limited, but will grow with future releases. The –assert switch enables this feature.
- With –assert, generate assertions for synthesis parallel_case and full_case.

Minor:

- Fix generate if's with empty if/else blocks. [Mat Zeno]
- Fix generate for cell instantiations with same name. [Mat Zeno]

17.1.153 Verilator 3.481 2005-10-12 Stable

Minor:

- Add /*verilator tracing_on/off*/ for waveform control.
- Fix split optimization reordering $display statements.

17.1.154 Verilator 3.480 2005-09-27 Beta

Major:

- Allow coverage of flattened modules, and multiple points per line. Coverage analysis requires SystemPerl 1.230 or newer.

Minor:

- Add preprocessor changes to support meta-comments.
- Optimize sequential assignments of different bits of same bus; ~5% faster.
- Optimize away duplicate lookup tables.
- Optimize wide concatenates into individual words. [Ralf Karge]
- Optimize local variables from delayed array assignments.

17.1.155 Verilator 3.470 2005-09-06 Stable

Minor:

- Optimize staging flops under reset blocks.
- Add '-Werror-...' to upgrade specific warnings to errors.
- Add GCC branch prediction hints on generated if statements.
- Fix bad simulation when same function called twice in same expression.
- Fix preprocessor substitution of quoted parameterized defines.

17.1.156 Verilator 3.464 2005-08-24 Stable

Major:

- Add &96;systemc_imp_header, for use when using –output-split.
- Add –stats option to dump design statistics.

Minor:

- Fix core dump with clock inversion optimizations.

17.1.157 Verilator 3.463 2005-08-05 Stable

Minor:

- Fix case defaults when not last statement in case list. [Wim Michiels]

17.1.158 Verilator 3.462 2005-08-03 Stable

Minor:

- Fix reordering of delayed assignments to same memory index. [Wim Michiels]
- Fix compile error with Flex 2.5.1. [Jens Arm]
- Fix multiply-instantiated public tasks generating non-compilable code.

17.1.159 Verilator 3.461 2005-07-28 Beta

Minor:

- Fix compile error with older versions of bison. [Jeff Dutton]

17.1.160 Verilator 3.460 2005-07-27 Beta

Major:

- Add -output-split option to enable faster parallel GCC compiles. To support –output-split, the makefiles now split VM_CLASSES into VM_CLASSES_FAST and VM_CLASSES_SLOW. This may require a change to local makefiles.
- Support -v argument to read library files.

Minor:

- When issuing unoptimizable warning, show an example path.
- Internal tree dumps now indicate edit number that changed the node.
- Fix false warning when a clock is constant.
- Fix X/Z in decimal numbers. [Wim Michiels]
- Fix genvar statements in non-named generate blocks.
- Fix core dump when missing newline in &96;define. [David van der bokke]

17.1.161 Verilator 3.450 2005-07-12

Major:

- $finish will no longer exit, but set Verilated::gotFinish(). This enables support for final statements, and for other cleanup code. If this is undesired, redefine the vl_user_finish routine. Top level loops should use Verilated::gotFinish() as a exit condition for their loop, and then call top->final(). To prevent a infinite loop, a double $finish will still exit; this may be removed in future releases.
- Support SystemVerilog keywords $bits, $countones, $isunknown, $onehot, $onehot0, always_comb, always_ff, always_latch, finish.

Minor:

- Fix "=== 1'bx" to always be false, instead of random.

17.1.162 Verilator 3.440 2005-06-28 Stable

Major:

- Add Verilog 2001 generate for/if/case statements.

17.1.163 Verilator 3.431 2005-06-24 Stable

Minor:

- Fix selection bugs introduced in 3.430 beta.

17.1.164 Verilator 3.430 2005-06-22 Beta

Minor:

- Add Verilog 2001 variable part selects [n+:m] and [n-:m]. [Wim Michiels]

17.1.165 Verilator 3.422 2005-06-10 Stable

Minor:

- Add Verilog 2001 power (**) operator. [Danny Ding]
- Fix crash and added error message when assigning to inputs. [Ralf Karge]
- Fix tracing of modules with public functions.

17.1.166 Verilator 3.421 2005-06-02 Beta

Minor:

- Fix error about reserved word on non-public signals.
- Fix missing initialization compile errors in 3.420 beta. [Ralf Karge]

17.1.167 Verilator 3.420 2005-06-02 Beta

Minor:

- Performance improvements worth ~20%
- Add -x-assign options; ~5% faster if use -x-assign=0.
- Add error message when multiple defaults in case statement.
- Optimize shifts out of conditionals and if statements.
- Optimize local 'short' wires.
- Fix case defaults when not last statement in case list. [Ralf Karge]
- Fix crash when wire self-assigns x=x.
- Fix gate optimization with top-flattened modules. [Mahesh Kumashikar]

17.1.168 Verilator 3.411 2005-05-30 Stable

Minor:

- Fix compile error in GCC 2.96. [Jeff Dutton]

17.1.169 Verilator 3.410 2005-05-25 Beta

Major:

- Allow functions and tasks to be declared public. They will become public C++ functions, with appropriate C++ types. This allows users to make public accessor functions/tasks, instead of having to use public variables and &96;systemc_header hacks.

Minor:

- Skip producing output files if all inputs are identical This uses timestamps, similar to make. Disable with –no-skip-identical.
- Improved compile performance with large case statements.
- Fix internal error in V3Table. [Jeff Dutton]
- Fix compile error in GCC 2.96, and with SystemC 1.2. [Jeff Dutton]

17.1.170 Verilator 3.400 2005-04-29 Beta

Major:

- Internal changes to support future clocking features.
- Verilog-Perl and SystemPerl are no longer required for C++ or SystemC output. If you want tracing or coverage analysis, they are still needed.
- Add –sc to create pure SystemC output not requiring SystemPerl.
- Add –pins64 to create 64 bit SystemC outputs instead of sc_bv<64>.
- The –exe flag is now required to produce executables inside the makefile. This was previously the case any time .cpp files were passed on the command line.
- Add -O3 and –inline-mult for performance tuning. [Ralf Karge] One experiment regained 5% performance, at a cost of 300% in compile time.

Minor:

- Improved performance of large case/always statements with low fanin by converting to internal lookup tables (ROMs).
- Initialize SystemC port names. [S Shuba]
- Add Doxygen comments to Verilated includes.
- Fix -cc pins 8 bits wide and less to be uint8_t instead of uint16_t.
- Fix crash when Mdir has same name as .v file. [Gernot Koch]
- Fix crash with size mismatches on case items. [Gernot Koch]

17.1.171 Verilator 3.340 2005-02-18 Stable

Minor:

- Report misconnected pins across all modules, instead of just first error.
- Improved large netlist compile times.
- Fix over-active inlining, resulting in compile slowness.

17.1.172 Verilator 3.332 2005-01-27

Major:

- Add -E preprocess only flag, similar to GCC.
- Add CMPCONSTLR when comparison is constant due to > or < with all ones.

Minor:

- Fix loss of first -f file argument, introduced in 3.331.

17.1.173 Verilator 3.331 2005-01-18

Major:

- The Verilog::Perl preprocessor is now C++ code inside of Verilator. This improves performance, makes compilation easier, and enables some future features.

Minor:

- Support arrays of instantiations (non-primitives only). [Wim Michiels]
- Fix unlinked error with defparam. [Shawn Wang]

17.1.174 Verilator 3.320 2004-12-10

Major:

- NEWS is now renamed Changes, to support CPAN indexing.
- If Verilator is passed a C file, create a makefile link rule. This saves several user steps when compiling small projects.

Minor:

- Add new COMBDLY warning in place of fatal error. [Shawn Wang]
- Fix mis-simulation with wide-arrays under bit selects. [Ralf Karge]
- Add NC Verilog as alternative to VCS for reference tests.
- Support implicit wire declarations on input-only signals. (Dangerous, as leads to wires without drivers, but allowed by spec.)
- Fix compile warnings on Suse 9.1

17.1.175 Verilator 3.311 2004-11-29

Major:

- Support implicit wire declarations (as a warning). [Shawn Wang]

Minor:

- Fix over-shift difference in Verilog vs C++. [Ralf Karge]

17.1.176 Verilator 3.310 2004-11-15

Major:

- Support defparam.
- Support gate primitives: buf, not, and, nand, or, nor, xor, xnor.

Minor:

- Ignore all specify blocks.

17.1.177 Verilator 3.302 2004-11-12

Minor:

- Support NAND and NOR operators.
- Better warnings when port widths don't match.
- Fix internal error due to some port width mismatches. [Ralf Karge]
- Fix WIDTH warnings on modules that are only used parameterized, not in 'default' state.
- Fix selection of SystemC library on cygwin systems. [Shawn Wang]
- Fix runtime bit-selection of parameter constants.

17.1.178 Verilator 3.301 2004-11-04

Minor:

- Fix 64 bit [31:0] = {#{}} mis-simulation. [Ralf Karge]
- Fix shifts greater then word width mis-simulation. [Ralf Karge]
- Fix to work around GCC 2.96 negation bug.

17.1.179 Verilator 3.300 2004-10-21

Major:

- New backend that eliminates most VL macros. Improves performance 20%-50%, depending on frequency of use of signals over 64 bits. GCC compile times with -O2 shrink by a factor of 10.

Minor:

- Fix "setting unsigned int from signed value" warning.

17.1.180 Verilator 3.271 2004-10-21

Minor:

- Fix "loops detected" error with some negedge clocks.
- Fix some output code spacing issues.

17.1.181 Verilator 3.270 2004-10-15

Minor:

- Support Verilog 2001 parameters in module headers. [Ralf Karge]
- Faster code to support compilers not inlining all Verilated functions.
- Fix numeric fault when dividing by zero.

17.1.182 Verilator 3.260 2004-10-07

Major:

- Support Verilog 2001 named parameter instantiation. [Ralf Karge]

Minor:

- Return 1's when one bit wide extract indexes outside array bounds.
- Fix compile warnings on 64-bit operating systems.
- Fix incorrect dependency in .d file when setting VERILATOR_BIN.

17.1.183 Verilator 3.251 2004-09-09

Minor:

- Fix parenthesis overflow in Microsoft Visual C++ [Renga Sundararajan]

17.1.184 Verilator 3.250 2004-08-30

Major:

- Support Microsoft Visual C++ [Renga Sundararajan]

Minor:

- SystemPerl 1.161+ is required.

17.1.185 Verilator 3.241 2004-08-17

Minor:

- Support ,'s to separate multiple assignments. [Paul Nitza]
- Fix shift sign extension problem using non-GCC compilers.

17.1.186 Verilator 3.240 2004-08-13

Major:

- Verilator now uses 64 bit math where appropriate. Inputs and outputs of 33-64 bits wide to the C++ Verilated model must now be uint64_t's; SystemC has not changed, they will remain sc_bv's. This increases performance by ~ 9% on x86 machines, varying with how frequently 33-64 bit signals occur. Signals 9-16 bits wide are now stored as 16 bit shorts instead of longs, this aids cache packing.

Minor:

- Fix SystemC compile error with feedthrus. [Paul Nitza]
- Fix concat value error introduced in 3.230.

17.1.187 Verilator 3.230 2004-08-10

Minor:

- Add coverage output to test_sp example, SystemPerl 1.160+ is required.
- Fix time 0 value of signals. [Hans Van Antwerpen] Earlier versions would not evaluate some combinatorial signals until posedge/negedge blocks had been activated.
- Fix wide constant inputs to public submodules [Hans Van Antwerpen]
- Fix wide signal width extension bug. Only applies when width mismatch warnings were overridden.

17.1.188 Verilator 3.220 2004-06-22

Major:

- Many waveform tracing changes:
- Tracing is now supported on C++ standalone simulations. [John Brownlee]

Minor:

- When tracing, SystemPerl 1.150 or newer is required.
- When tracing, Verilator must be called with the –trace switch.
- Add SystemPerl example to documentation. [John Brownlee]
- Various Cygwin compilation fixes. [John Brownlee]

17.1.189 Verilator 3.210 2004-04-01

Major:

- Compiler optimization switches have changed See the BENCHMARKING section of the documentation.
- With Verilog-Perl 2.3 or newer, Verilator supports SystemVerilog preprocessor extensions.

Minor:

- Add localparam. [Thomas Hawkins]
- Add warnings for SystemVerilog reserved words.

17.1.190 Verilator 3.203 2004-03-10

Minor:

- Notes and repairs for Solaris. [Fred Ma]

17.1.191 Verilator 3.202 2004-01-27

Major:

- The beta version is now the primary release. See below for many changes. If you have many problems, you may wish to try release 3.125.
- Verilated::traceEverOn(true) must be called at time 0 if you will ever turn on tracing (waveform dumping) of signals. Future versions will need this switch to disable trace incompatible optimizations.

Minor:

- Optimize common replication operations.
- Fix several tracing bugs

17.1.192 Verilator 3.201-beta 2003-12-10

Major:

- BETA VERSION, USE 3.124 for stable release!
- Version 3.2XX includes a all new back-end. This includes automatic inlining, flattening of signals between hierarchy, and complete ordering of statements. This results in 60-300% execution speedups, though less pretty C++ output. Even better results are possible using GCC 3.2.2 (part of Redhat 9.1), as GCC has fixed some optimization problems which Verilator exposes.

  If you are using &96;systemc_ctor, beware pointers to submodules are now initialized after the constructor is called for a module, to avoid segfaults, move statements that reference subcells into initial statements.
- C++ Constructor that creates a verilog module may take a char* name. This name will be used to prefix any $display %m arguments, so users may distinguish between multiple Verilated modules in a single executable.

17.1.193 Verilator 3.125 2004-01-27

Minor:

- Optimize bit replications

17.1.194 Verilator 3.124 2003-12-05

Major:

- A optimized executable will be made by default, in addition to a debug executable. Invoking Verilator with –debug will pick the debug version.

Minor:

- Many minor invisible changes to support the next version.

17.1.195 Verilator 3.123 2003-11-10

Minor:

- Wide bus performance enhancements.
- Fix function call bug when width warning suppressed. [Leon Wildman]
- Fix __DOT__ compile problem with funcs in last revision. [Leon Wildman]

17.1.196 Verilator 3.122 2003-10-29

Major:

- Modules which are accessed from external code now must be marked with */verilator public_module/* unless they already contain public signals. To enforce this, private cell names now have a string prepended.

Minor:

- Fix replicated function calls in one statement. [Robert A. Clark]
- Fix function call bug when width warning suppressed. [Leon Wildman]

17.1.197 Verilator 3.121 2003-09-29

Minor:

- Support multiplication over 32 bits. [Chris Boumenot] Also improved speed of addition and subtraction over 32 bits.
- Detect bit selection out of range errors.
- Detect integer width errors.
- Fix width problems on function arguments. [Robert A. Clark]

17.1.198 Verilator 3.120 2003-09-24

Minor:

- $finish now exits the model (via vl_finish function).
- Support inputs/outputs in tasks.
- Support V2K "integer int = {INITIAL_VALUE};"
- Ignore floating point delay values. [Robert A. Clark]
- Ignore &96;celldefine, &96;endcelldefine, etc. [Robert A. Clark]
- Optimize reduction operators.
- Fix converting "ooo" into octal values.
- Fix $display("%x");

17.1.199 Verilator 3.112 2003-09-16

Minor:

- Fix functions in continuous assignments. [Robert A. Clark]
- Fix inlining of modules with 2-level deep outputs.

17.1.200 Verilator 3.111 2003-09-15

Minor:

- Fix declaration of functions before using that module. [Robert A. Clark]
- Fix module inlining bug with outputs.

17.1.201 Verilator 3.110 2003-09-12

Major:

- Support Verilog 2001 style input/output declarations. [Robert A. Clark]
- Support local vars in headers of function/tasks. [Leon Wildman]

17.1.202 Verilator 3.109 2003-08-28

Major:

- Support local variables in named begin blocks. [Leon Wildman]

17.1.203 Verilator 3.108 2003-08-11

Major:

- Support functions.

Minor:

- Signals 8 bits and shorter are now stored as chars instead of uint32_t's. This improves Dcache packing and improves performance by ~7%.
- $display now usually results in a single VL_PRINT rather then many.
- Optimize conditionals (?:)

17.1.204 Verilator 3.107 2003-07-15

Major:

- –private and –l2name are now the default, as this enables additional optimizations. Use –noprivate or –nol2name to get the older behavior.

Minor:

- Now support $display of binary and wide format data.
- Add detection of incomplete case statements, and added related optimizations worth ~4%.
- Work around flex bug in Redhat 8.0. [Eugene Weber]
- Add some additional C++ reserved words.
- Additional constant optimizations, ~5% speed improvement.

17.1.205 Verilator 3.106 2003-06-17

Major:

- $c can now take multiple expressions as arguments. For example $c("foo","bar(",32+1,");") will insert "foobar(33);" This makes it easier to pass the values of signals.
- Several changes to support future versions that may have signal-eliminating optimizations. Users should try to use these switch on designs, they will become the default in later versions.
- Add –private switch and */verilator public/* metacomment. This renames all signals so that compile errors will result if any signals referenced by C++ code are missing a */verilator public/* metacomment.
- With –l2name, the second level cell C++ cell is now named "v". Previously it was named based on the name of the verilog code. This means to get to signals, scope to "{topcell} ->v ->{mysignal}" instead of "{topcell} ->{verilogmod}. {mysignal}". This allows different modules to be substituted for the cell without requiring source changes.

Minor:

- Several cleanups for Redhat 8.0.

17.1.206 Verilator 3.105 2003-05-08

Minor:

- Fix more GCC 3.2 errors. [David Black]

17.1.207 Verilator 3.104 2003-04-30

Major:

- Indicate direction of ports with VL_IN and VL_OUT.
- Allow $c32, etc, to specify width of the $c statement for VCS.
- Numerous performance improvements, worth about 25%

Minor:

- Fix false "indent underflow" error inside &96;systemc_ctor sections.
- Fix missing ordering optimizations when outputs also used internally.
- Assign constant cell pins in initial blocks rather then every cycle.
- Promote subcell's combo logic to sequential evaluation when possible.
- Fix GCC 3.2 compile errors. [Narayan Bhagavatula]

17.1.208 Verilator 3.103 2003-01-28

Minor:

- Fix missing model evaluation when clock generated several levels of hierarchy across from where it is used as a clock. [Richard Myers]
- Fix sign-extension bug introduced in 3.102.

17.1.209 Verilator 3.102 2003-01-24

Minor:

- Fix sign-extension of X/Z's ("32'hx")

17.1.210 Verilator 3.101 2003-01-13

Minor:

- Fix 'parameter FOO=#'bXXXX' [Richard Myers]
- Allow spaces inside numbers ("32'h 1234") [Sam Gladstone]

17.1.211 Verilator 3.100 2002-12-23

Major:

- Support for simple tasks w/o vars or I/O. [Richard Myers]

Minor:

- Ignore DOS carriage returns in Linux files. [Richard Myers]

17.1.212 Verilator 3.012 2002-12-18

Minor:

- Fix parsing bug with casex statements containing case items with bit extracts of parameters. [Richard Myers]
- Fix bug which could cause writes of non-power-of-2 sized arrays to corrupt memory beyond the size of the array. [Dan Lussier]
- Fix bug which did not detect UNOPT problems caused by submodules. See the description in the verilator man page. [John Deroo]
- Fix compile with threaded Perl. [Ami Keren]

17.1.213 Verilator 3.010 2002-11-03

Major:

- Support SystemC 2.0.1. SystemPerl version 1.130 or newer is required.

Minor:

- Fix bug with inlined modules under other inlined modules. [Scott Bleiweiss]

17.1.214 Verilator 3.005 2002-10-21

Minor:

- Fix X's in case (not casex/z) to constant propagate correctly.
- Fix missing include. [Kurachi]

17.1.215 Verilator 3.004 2002-10-10

Minor:

- Add module_inline metacomment and associated optimizations.
- Allow coverage_block_off metacomment in place of &96;coverage_block_off. This prevents problems with Emacs AUTORESET. [Ray Strouble]
- Fix &96;coverage_block_off also disabling subsequent blocks.
- Fix unrolling of loops with multiple simple statements.
- Fix compile warnings on newer GCC. [Kurachi]
- Additional concatenation optimizations.

17.1.216 Verilator 3.003 2002-09-13

Minor:

- Now compiles on Windows 2000 with Cygwin.
- Fix bug with pin assignments to wide memories.
- Optimize wire assignments to constants.

17.1.217 Verilator 3.002 2002-08-19

Major:

- First public release of version 3.

17.1.218 Verilator 3.000 2002-08-03

Major:

- All new code base. Many changes too numerous to mention.

Minor:

- Approximately 4 times faster then Verilator 2.
- Support initial statements
- Support correct blocking/nonblocking assignments
- Support &96;defines across multiple modules
- Optimize call ordering, constant propagation, and dead code elimination.

17.1.219 Verilator 2.1.8 2002-04-03

Major:

- All applications must now link against include/verilated.cpp

Minor:

- Paths specified to verilator_make should be absolute, or be formed to allow for execution in the object directory (prepend ../ to each path.) This allows relative filenames for makes which hash and cache dependencies.
- Add warning when parameter constants are too large. [John Deroo]
- Add warning when x/?'s used in non-casez statements.
- Add warning when blocking assignments used in posedge blocks. [Dan Lussier]
- Split evaluation function into clocked and non-clocked, 20% perf gain.

17.1.220 Verilator 2.1.5 2001-12-01

Major:

- Add coverage analysis. In conjunction with SystemC provide line coverage reports, without SystemC, provide a hook to user written accumulation function. See –coverage option of verilator_make.

Minor:

- Relaxed multiply range checking
- Support for constants up to 128 bits
- Randomize values used when assigning to X's.
- Add -guard option of internal testing.
- Changed indentation in emitted code to be automatically generated.
- Fix corruption of assignments of signal over 32 bits with non-0 lsb.

17.1.221 Verilator 2.1.4 2001-11-16

Major:

- Add $c("c_commands();"); for embedding arbitrary C code in Verilog.

17.1.222 Verilator 2.1.3 2001-11-03

Major:

- Support for parameters.

17.1.223 Verilator 2.1.2 2001-10-25

Major:

- Verilog Errors now reference the .v file rather then the .vpp file.

Minor:

- Support strings in assignments: reg [31:0] foo = "STRG";
- Support %m in format strings. Ripped out old $info support, use Verilog-Perl's vpm program instead.
- Convert $stop to call of v_stop() which user can define.
- Fix bug where a==b==c would have wrong precedence rule.
- Fix bug where XNOR on odd-bit-widths (~^ or ^~) had bad value.

17.1.224 Verilator 2.1.1 2001-05-17

Major:

- New test_sp directory for System-Perl (SystemC) top level instantiation of the Verilated code, lower modules are still C++ code. (Experimental).
- New test_spp directory for Pure System-Perl (SystemC) where every module is true SystemC code. (Experimental)

Minor:

- Input ports are now loaded by pointer reference into the sub-cell. This is faster on I-386 machines, as the stack must be used when there are a large number of parameters. Also, this simplifies debugging as the value of input ports exists for tracing.
- Many code cleanups towards standard C++ style conventions.

17.1.225 Verilator 2.1.0 2001-05-08

Minor:

- Many code cleanups towards standard C++ style conventions.

17.1.226 Version history lost

17.1.227 Verilator 1.8 1996-07-08

[Versions 0 to 1.8 were by Paul Wasson] * Fix single bit in concat from instance output incorrect offset bug.

17.1.228 Verilator 1.7 1996-05-20

- Mask unused bits of DONTCAREs.

17.1.229 Verilator 1.6 1996-05-13

- Add fasttrace script

17.1.230 Verilator 1.5 1996-01-09

- Pass structure pointer into translated code, so multiple instances can use same functions.
- Fix static value concat on casex items.

17.1.231 Verilator 1.1 1995-03-30

- Bug fixes, added verimake_partial script, performance improvements.

17.1.232 Verilator 1.0c 1994-09-30

- Initial release of Verilator

17.1.233 Verilator 0.0 1994-07-08

- First code written.

17.1.234 Copyright

CHAPTER

EIGHTEEN

COPYRIGHT

The latest version of Verilator is available from https://verilator.org.