

VERILATOR

Verilator

Release 5.040

Wilson Snyder

2025-08-30

GETTING STARTED

1 Overview **1**

2 Examples **2**
- 2.1 Example Create-Binary Execution . . . 2
- 2.2 Example C++ Execution . . . 3
- 2.3 Example SystemC Execution . . . 4
- 2.4 Examples in the Distribution . . . 5

3 Installation **7**
- 3.1 Package Manager Quick Install . . . 7
- 3.2 pre-commit Quick Install . . . 7
- 3.3 Git Quick Install . . . 7
- 3.4 Detailed Build Instructions . . . 8
- 3.5 Verilator Build Docker Container . . . 11
- 3.6 Verilator Executable Docker Container . . . 12

4 CMake Installation **14**
- 4.1 Quick Install . . . 14
- 4.2 Usage . . . 15
- 4.3 Example . . . 15

5 Verilating **16**
- 5.1 Binary, C++ and SystemC Generation . . . 16
- 5.2 Finding and Binding Modules . . . 17
- 5.3 Hierarchical Verilation . . . 17
- 5.4 Cross Compilation . . . 18
- 5.5 Multithreading . . . 18
- 5.6 GNU Make . . . 20
- 5.7 CMake . . . 20
- 5.8 Verilation Summary Report . . . 22

6 Connecting to Verilated Models **24**
- 6.1 Structure of the Verilated Model . . . 24
- 6.2 Connecting to C++ . . . 25
- 6.3 Connecting to SystemC . . . 25
- 6.4 Verilated API . . . 25
- 6.5 Direct Programming Interface (DPI) . . . 26
- 6.6 Verification Procedural Interface (VPI) . . . 28
- 6.7 Wrappers and Model Evaluation Loop . . . 30
- 6.8 Verilated and VerilatedContext . . . 31

7 Simulating (Verilated-Model Runtime) 32
7.1 Simulation Summary Report . . . 32
7.2 Benchmarking & Optimization . . . 33
7.3 Coverage Analysis . . . 34
7.4 Code Profiling . . . 36
7.5 Execution Profiling . . . 36
7.6 Profiling ccache efficiency . . . 38
7.7 Save/Restore . . . 38
7.8 Profile-Guided Optimization . . . 38
7.9 Runtime Debugging . . . 40

8 Contributing and Reporting Bugs 41
8.1 Announcements . . . 41
8.2 Reporting Bugs . . . 41
8.3 Minimizing bug-inducing code . . . 42
8.4 Contributing to Verilator . . . 42

9 FAQ/Frequently Asked Questions 44
9.1 Questions . . . 44

10 Input Languages 51
10.1 Language Standard Support . . . 51
10.2 Time . . . 52
10.3 Language Limitations . . . 53
10.4 Language Keyword Limitations . . . 55

11 Language Extensions 58

12 Executable and Argument Reference 67
12.1 verilator Arguments . . . 67
12.2 Verilator Control Files . . . 97
12.3 verilator_coverage . . . 100
12.4 verilator_gantt . . . 103
12.5 verilator_profcfunc . . . 104
12.6 Simulation Runtime Arguments . . . 105

13 Errors and Warnings 107
13.1 Disabling Warnings . . . 107
13.2 Error And Warning Format . . . 107
13.3 List Of Warnings . . . 108

14 Files 136
14.1 Files in the Git Tree . . . 136
14.2 Files Read/Written . . . 136

15 Environment 139

16 Make Variables 141

17 Deprecations 142

18 Contributors and Origins 143
18.1 Authors . . . 143
18.2 Contributors . . . 143
18.3 Historical Origins . . . 145

19 Revision History **146**
19.1 Revision History and Change Log 146

20 Copyright **253**

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 1364-2005", 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, 2017, and 2023. The Verilator documentation uses the shorthand e.g., "IEEE 1800-2023", to refer to the, e.g., 2023 version of this standard.

CHAPTER

TWO

EXAMPLES

This section covers the following examples:

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

2.1 Example Create-Binary Execution

We'll compile this SystemVerilog example into a Verilated simulation binary. For an example that discusses the next level of detail see *Example C++ Execution*.

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 file:

```
mkdir test_our
cd test_our

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

Now we run Verilator on our little example.

```
verilator --binary -j 0 -Wall our.v
```

Breaking this command down:

1. `--binary` telling Verilator to do everything needed to create a simulation executable.
2. `-j` *0* to Verilate using use as many CPU threads as the machine has.

3. -Wall so Verilator has stronger lint warnings enabled.
4. An finally, our.v, which is our SystemVerilog design file.

And now we run it:

```
obj_dir/Vour
```

And we get as output:

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

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 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) {
      VerilatedContext* contextp = new VerilatedContext;
      contextp->commandArgs(argc, argv);
      Vour* top = new Vour{contextp};
      while (!contextp->gotFinish()) { top->eval(); }
      delete top;
      delete contextp;
      return 0;
  }
EOF
```

Now we run Verilator on our little example;

```
verilator --cc --exe --build -j 0 -Wall sim_main.cpp our.v
```

Breaking this command down:

1. --cc to get C++ output (versus e.g., SystemC, or only linting).
2. --exe, along with our sim_main.cpp wrapper file, so the build will create an executable instead of only a library.
3. --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*.)
4. -j 0 to Verilate using use as many CPU threads as the machine has.
5. -Wall so Verilator has stronger lint warnings enabled.
6. And 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
```

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.

2.3 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);
```

(continues on next page)

(continued from previous page)

```
    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"
  using namespace sc_core;
  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 --sc --exe -Wall 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.

2.4 Examples in the Distribution

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

examples/make_hello_binary
: Example GNU-make simple Verilog->binary conversion

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 distribution packages almost never have the most recent Verilator version, so we recommend following *Git Quick Install* below, instead.) To install as a package:

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

For other distributions, refer to Repology Verilator Distro Packages.

3.2 pre-commit Quick Install

You can use Verilator's pre-commit hook to lint your code before committing it. It encapsulates the *Verilator Build Docker Container*, so you need docker on your system to use it. The verilator image will be downloaded automatically.

To use the hook, add the following entry to your .pre-commit-config.yaml:

```
repos:
  - repo: https://github.com/verilator/verilator
    rev: v5.026  # or later
    hooks:
      - id: verilator
```

3.3 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 help2man 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
```

(continues on next page)

(continued from previous page)

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

3.4 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*.

3.4.1 OS Requirements

Verilator is developed and has primary testing on Ubuntu, with additional testing on FreeBSD and Apple OS-X. Versions have also been built on Red Hat Linux, other flavors of GNU/Linux-ish platforms, 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++.

3.4.2 Install Prerequisites

To build or run Verilator, you need these standard packages:

```
sudo apt-get install git help2man 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)
```

For SystemC:

```
sudo apt-get install libsystemc libsystemc-dev
```

For constraints:

```
sudo apt-get install z3  # Optional solver
```

The following is optional but is recommended for nicely rendered command line help when running Verilator:

```
sudo apt-get install perl-doc
```

To build or run Verilator, 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 mold  # If present at build, needed for run
sudo apt-get install libgoogle-perftools-dev numactl
```

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 also need these (see internals.rst):

```
sudo apt-get install clang clang-format-18 cmake gdb gprof graphviz lcov
sudo apt-get install python3-clang python3-distro yapf3 bear jq
sudo pip3 install sphinx sphinx_rtd_theme sphinxcontrib-spelling breathe gersemi mbake ruff sarif-tools
sudo pip3 install git+https://github.com/antmicro/astsee.git
cpan install Pod::Perldoc
```

Install SystemC

SystemC code can be generated from Verilator (with --sc) if it is installed as a package (see above).

Alternatively, from their sources, 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.

```
sudo apt-get install gtkwave  # Optional Waveform viewer
```

Install Z3

In order to use constrained randomization the Z3 Theorem Prover must be installed, however this is not required at Verilator build time. There are other compatible SMT solvers, like CVC5/CVC4, but they are not guaranteed to work. Since different solvers are faster for different scenarios, the solver to use at run-time can be specified by the environment variable VERILATOR_SOLVER.

3.4.3 Obtain Sources

Get the sources from the git repository: (You need to 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
```

3.4.4 Auto Configure

Create the configuration script:

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

3.4.5 Eventual Installation Options

Before configuring the build, you must 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, VERILATOR_SOLVER, 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 (don't run make install). 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 (see *Installation*), 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 Prefix

You may be an OS package maintainer building a Verilator package, or 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 that 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 (see *Installation*), you need to add the path to the bin directory to your PATH. Or, 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 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 (see *Installation*), the binaries should be in a location already in your $PATH environment variable.

3.4.6 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...
```

3.4.7 Compile

Compile Verilator:

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

3.4.8 Test

Check the compilation by running self-tests:

```
make test
```

3.4.9 Install

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

```
make install
```

3.5 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 13.3.0, clang 18.1.3, default: 13.3.0)

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 use the *-e* switch to pass environment variables:

```
docker run -ti -e CC=clang-18 -e CXX=clang++-18 verilator/verilator-buildenv test
```

The tests, that involve numactl are not working due to security restrictions. To run those too, add the CAP_SYS_NICE capability during the start of the container:

```
docker run -ti --cap-add=CAP_SYS_NICE 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`␣
↪verilator/verilator-buildenv test
```

3.5.1 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.

3.6 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 pull the container from docker hub, run the latest Verilator and print Verilator's version.

Containers are automatically built and pushed to docker hub 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> .
```

3.6.1 Internals

The Dockerfile builds Verilator and removes the tree when completed to reduce the image size. The entrypoint is 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.

A hook is also defined and run by Docker Hub via automated builds.

CHAPTER

FOUR

CMAKE INSTALLATION

This section discusses how to build and install Verilator using cmake. Currently cmake is only officially supported for Windows builds (not Linux).

4.1 Quick Install

1. Install Python for your platform from https://www.python.org/downloads/.
2. Install CMake for your platform from https://cmake.org/download/ or build it from source.
3. If the compiler of your choice is MSVC, then install https://visualstudio.microsoft.com/downloads/. If the compiler of your choice is Clang, then install https://releases.llvm.org/download.html or build it from source.
4. For flex and bison use https://github.com/lexxmark/winflexbison to build and install.
5. For build on Windows using MSVC set environment variable WIN_FLEX_BISON to install directory. For build on Windows/Linux/OS-X using ninja set the environment variable FLEX_INCLUDE to the directory containing FlexLexer.h and ensure that flex/bison is available within the PATH.

To obtain verilator sources download https://github.com/verilator/verilator/archive/refs/heads/master.zip or clone https://github.com/verilator/verilator using git *Obtain Sources*.

To build using MSVC:

```
cd verilator  # directory containing source files of verilator
mkdir build
cmake .. -DCMAKE_BUILD_TYPE=Release --install-prefix $PWD/../install
cmake --build . --config Release
cmake --install . --prefix $PWD/../install
```

To build using ninja:

```
cd verilator
mkdir build
cmake -G Ninja .. -DCMAKE_BUILD_TYPE=Release --install-prefix $PWD/../install -DCMAKE_
↪MAKE_PROGRAM=<path to ninja binary> -DBISON_EXECUTABLE=<path to bison> -DFLEX_
↪EXECUTABLE=<path to flex>
<path to ninja binary> #execute ninja
cmake --install . --prefix $PWD/../install
```

4.2 Usage

To use Verilator set the environment variable VERILATOR_ROOT to the install directory specified in the above build.

4.3 Example

```
cd verilator/examples
cd cmake_hello_c
mkdir build
cd build
cmake ..  # cmake -G Ninja ..
cmake --build . --config Release # ninja
# execute the generated binary
```

CHAPTER

FIVE

VERILATING

Verilator may be used in five major ways:

- With the --binary option, Verilator will translate the design into an executable, via generating C++ and compiling it. See *Binary, C++ and SystemC Generation*.
- With the --cc or --sc options, Verilator will translate the design into C++ or SystemC code, respectively. See *Binary, 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.

5.1 Binary, C++ and SystemC Generation

Verilator will translate a SystemVerilog design into C++ with the --cc option, or into SystemC with the --sc option. It will translate into C++ and compile it into an executable binary with the --binary option.

When using these options:

1. Verilator reads the input Verilog code and 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 --binary or --main is used, Verilator creates a C++ top wrapper to read command line arguments, create the model, and execute the model.
4. If --binary or --exe is used, Verilator creates makefiles to generate a simulation executable, otherwise, it creates makefiles to generate an archive (.a) containing the objects.
5. If --binary or --build is used, it calls *GNU Make* or *CMake* to build the model.

Once a model is built, the next step is typically for the user to run it, see *Simulating (Verilated-Model Runtime)*.

5.2 Finding and Binding Modules

Verilator provides several mechanisms to find the source code containing a module, primitive, interface, or program ("module" in this section) and bind them to an instantiation. These capabilities are similar to the "Precompiling in a single-pass" use model described in IEEE 1800-2023 33.5.1, although *config* is not yet supported.

Verilator first reads all files provided on the command line and -f files, and parses all modules within. Each module is assigned to the most recent library specified with -work, thus *-work liba a.v -work libb b.v* will assign modules in *a.v* to *liba* and modules in *b.v* to *libb*.

If a module is not defined from a file on the command-line, Verilator attempts to find a filename constructed from the module name using -y and +*libext*.

Binding begins with the --top module, if provided. If not provided Verilator attempts to figure out the top module itself, and if multiple tops result a MULTITOP warning is issued which may be suppressed (see details in MULTITOP).

Verilator will attempt to bind lower unresolved instances first in the same library name as the parent's instantiation library, and if not found search globally across all libraries in the order modules were declared. This allows otherwise conflicting duplicate module names between libraries to coexist uniquely within each library name. When IEEE *config use* is supported, more complicated selections will be able to be specified.

5.3 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 separated out of a multi-core SoC design.

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 Verilated 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.

5.3.1 Usage

Users need to mark one or more moderate-size modules 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 the *Verilator Control Files*.

Then pass the --hierarchical option to Verilator.

The compilation is the same as when not using hierarchical mode.

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

5.3.2 Limitations

Hierarchy blocks have some limitations, including:

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

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.

5.3.3 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 to the wrappers of all other hierarchy blocks. The second 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.

5.4 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 configure and compile Verilator, the host system is where you run Verilator, and the target system is where you compile the Verilated code and run the simulation.

Verilator requires the build and host system types 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_package.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.

5.5 Multithreading

Verilator supports multithreaded simulation models.

With --threads 1, the generated model is single-threaded; however, the support libraries are multithread safe. This allows different instantiations of the 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 the proper ratio of threads and CPU cores.

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 multithreaded 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.

When using --trace-vcd to perform VCD tracing, the VCD trace construction is parallelized using the same number of threads as specified with --threads, and is executed on the same thread pool as the model.

The --trace-threads options can be used with --trace-fst to offload FST tracing using multiple threads. If --trace-threads is given 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, up to N additional threads will be created and managed by the trace files (e.g., VerilatedFstC), to offload construction of the trace dump. The main thread will be released to proceed with execution as soon as possible, though some main thread blocking is still necessary while capturing the trace. FST tracing can utilize up to 2 offload threads, so there is no use of setting --trace-threads higher than 2 at the moment.

When running a multithreaded model, the default Linux task scheduler often works against the model by assuming short-lived threads and thus it often schedules threads using multiple hyperthreads within the same physical core. If there is no affinity already set, on Linux only, Verilator attempts to set thread-to-processor affinity in a reasonable way.

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 have different physical cores. (Also useful is numactl --hardware, or lscpu, but those don'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 to use, e.g., socket 1, or if you Verilated with a different number of threads. To see what CPUs are actually used, use --prof-exec.

5.5.1 Multithreaded Verilog and Library Support

$display/$stop/$finish are delayed until the end of an eval() call 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-vcd or other trace options, 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.

5.6 GNU Make

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

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

5.7 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.

Using CMake >= 3.12 and the Ninja generator is recommended, 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 creates a verilate() function. verilate() will automatically create custom commands to run Verilator and add the generated C++ sources to the target specified.

5.7.1 Verilate in CMake

```
verilate(target SOURCES source ... [TOP_MODULE top] [PREFIX name]
        [COVERAGE] [SYSTEMC]
        [TRACE_FST] [TRACE_SAIF] [TRACE_VCD] [TRACE_THREADS num]
        [INCLUDE_DIRS dir ...] [OPT_SLOW ...] [OPT_FAST ...]
        [OPT_GLOBAL ..] [DIRECTORY dir] [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 list 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. It must be unique in each call to verilate(), so this is necessary if you build a module multiple times with different parameters. It must be a valid C++ identifier, i.e., it contains no white space and only characters A-Z, a-z, 0-9 or _.

SOURCES

List of Verilog files to Verilate. You must provide 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 CMake can find the SystemC installation, and 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. Enable a multithreaded model; see --threads.

TOP_MODULE

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

TRACE

Deprecated. Same as TRACE_VCD, which should be used instead.

TRACE_FST

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

TRACE_SAIF
: Optional. Enables SAIF tracing if present, equivalent to "VERILATOR_ARGS –trace-saif".

TRACE_THREADS
: Optional. Enable multithreaded FST trace; see --trace-threads.

TRACE_VCD
: Optional. Enables VCD tracing if present, equivalent to "VERILATOR_ARGS –trace-vcd".

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

5.7.2 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).

5.8 Verilation Summary Report

When Verilator generates code, unless --quiet-stats is used, it will print a report to stdout summarizing the build. For example:

```
- Verilation Report: Verilator ....
- Verilator: Built from 354 MB sources in 247 modules,
    into 74 MB in 89 C++ files needing 0.192 MB
- Verilator: Walltime 26.580 s (elab=2.096, cvt=18.268,
    bld=2.100); cpu 26.548 s on 1 threads; alloced 2894.672 MB
```

The information in this report is:

"Verilator ..."
: Program version.

"234 MB sources"
: Characters of post-preprocessed text in all input Verilog and Verilator Control files in megabytes.

"247 modules"
: Number of interfaces/modules/classes/packages in design before elaboration.

"into 74 MB"

Characters of output C++ code, including comments in megabytes.

"89 C++ files"

Number of .cpp files created.

"needing 192MB"

Verilation-time minimum-bound estimate of memory needed to run model in megabytes. (Expect to need significantly more.)

"Walltime 26.580 s"

Real elapsed wall time for Verilation and build.

"elab=2.096"

Wall time to read in files and complete elaboration.

"cvt=18.268"

Wall time for Verilator to process and write output.

"bld=2.1"

Wall time to compile gcc/clang (if using --build).

"cpu 22.548 s"

CPU time used, total across all CPU threads.

"4 threads"

Number of simultaneous threads used.

"alloced 123 MB"

Total memory used during build by Verilator executable (excludes --build compiler's usage) in megabytes.

CHAPTER

SIX

CONNECTING TO VERILATED MODELS

6.1 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.

6.1.1 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 references with modelp->rootp->internal->member references, which contain one additional indirection via the rootp pointer.

6.2 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.

Refer to examples/make_tracing_c in the distribution for a detailed commented example.

Top level IO signals are read and written as members of the model. You call the model's eval() method to evaluate the model. When the simulation is complete call the model's final() method to execute any SystemVerilog final blocks, and complete any assertions. If using --timing, there are two additional functions for checking if there are any events pending in the simulation due to delays, and for retrieving the simulation time of the next delayed event. See *Wrappers and Model Evaluation Loop*.

6.3 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 uint64_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.

6.4 Verilated API

The API to a Verilated model is the C++ headers in the include/ directory in the distribution. These headers use Doxygen comments, /// and //<, to indicate and document those functions that are part of the Verilated public API.

6.4.1 Process-Level Clone APIs

Modern operating systems support process-level clone (a.k.a copying, forking) with system call interfaces in C/C++, e.g., fork() in Linux.

However, after cloning a parent process, some resources cannot be inherited in the child process. For example, in POSIX systems, when you fork a process, the child process inherits all the memory of the parent process. However, only the thread that called fork is replicated in the child process. Other threads are not.

Therefore, to support the process-level clone mechanisms, Verilator supports prepareClone() and atClone() APIs to allow the user to manually re-construct the model in the child process. The two APIs handle all necessary resources required for releasing and re-initializing before and after cloning.

The two APIs are supported in the verilated models. Here is an example of usage with Linux fork() and pthread_atfork APIs:

```
// static function pointers to fit pthread_atfork
static auto prepareClone = [](){ topp->prepareClone(); };
static auto atClone = [](){ topp->atClone(); };

// in main function, register the handlers:
pthread_atfork(prepareClone, atClone, atClone);
```

For better flexibility, you can also manually call the handlers before and after fork().

With the process-level clone APIs, users can create process-level snapshots for the verilated models. While the Verilator save/restore option provides persistent and circuit-dependent snapshots, the process-level clone APIs enable in-memory, circuit-transparent, and highly efficient snapshots.

6.5 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.

6.5.1 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; }
```

6.5.2 DPI System Task/Functions

Verilator extends the DPI format to allow using the same scheme to efficiently add system functions. 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"
...
const svScope scope = svGetScopeFromName("TOP.dut");
assert(scope);  // Check for nullptr if scope not found
svSetScope(scope);
```

(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.

6.5.3 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
```

6.5.4 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.

6.5.5 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;
    }
}
```

See the IEEE Standard for more information.

6.5.6 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.

6.5.7 Public Functions

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

6.6 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().

Verilator also tracks when the model state has been modified via the VPI with an evalNeeded flag. This flag can be checked with VerilatedVpi::evalNeeded() and it can be cleared with VerilatedVpi::clearEvalNeeded(). Used together it is possible to skip eval() calls if no model state has been changed since the last eval().

Any data written via vpi_put_value with vpiInertialDelay will be deferred for later. These delayed values can be flushed to the model with VerilatedVpi::doInertialPuts().

6.6.1 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 #(
    parameter WIDTH /*verilator public_flat_rd*/ = 32
  ) (input clk);
    reg [WIDTH-1:0] readme   /*verilator public_flat_rd*/;
    reg [WIDTH-1:0] 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

  uint64_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);
    const char* type = vpi_get_str(vpiType, vh1);
    const int size = vpi_get(vpiSize, vh1);
    printf("register name: %s, type: %s, size: %d\n", name, type, size);  // Prints "register name: readme,
↪ type: vpiReg, size: 32"

    s_vpi_value v;
```

(continues on next page)

(continued from previous page)

```
    v.format = vpiIntVal;
    vpi_get_value(vh1, &v);
    printf("Value of %s: %d\n", name, v.value.integer);  // Prints "Value of readme: 0"
}

int main(int argc, char** argv) {
    Verilated::commandArgs(argc, argv);
    const std::unique_ptr<VerilatedContext> contextp{new VerilatedContext};
    const std::unique_ptr<Vour> top{new Vour{contextp.get()}};

    contextp->internalsDump();  // See scopes to help debug
    while (!contextp->gotFinish()) {
        top->eval();
        VerilatedVpi::callValueCbs();  // For signal callbacks
        read_and_check();
    }
    return 0;
}
EOF
```

6.7 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.

3. If using delays and --timing, there are two additional methods the user should call:

- designp->eventsPending(), which returns true if there are any delayed events pending,
- designp->nextTimeSlot(), which returns the simulation time of the next delayed event. This method can only be called if designp->eventsPending() returned true.

Call eventsPending() to check if you should continue with the simulation, and then nextTimeSlot() to move simulation time forward. --main can be used with --timing to generate a basic example of a timing-enabled eval loop.

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. With --timing, it resumes any delayed processes awaiting the current simulation time. Then Verilator evaluates combinational 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.

6.8 Verilated and VerilatedContext

Multiple C++ 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. SystemC requires using only the single, default VerilatedContext.

The Verilated:: methods, including the Verilated::commandArgs call shown above, 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

SEVEN

SIMULATING (VERILATED-MODEL RUNTIME)

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

7.1 Simulation Summary Report

When simulation finishes, it will print a report to stdout summarizing the simulation. This requires the model being Verilated with --main, or the user's *main()* calling *VerilatedContext->statsPrintSummary()*.

The report may be disabled with +verilator+quiet.

For example:

```
- S i m u l a t i o n   R e p o r t: Verilator ...
- Verilator: End at simtime 123 ns; walltime 1234.001 s; speed 123 ns/s
- Verilator: cpu 22.001 s on 4 threads; alloced 123 MB
```

The information in this report is:

"Verilator ..."

Program version.

"End at simtime 123 ns"

Verilog $time at which the model finished or stopped.

"walltime 1234.001 s"

Real elapsed wall time in seconds.

"speed 123.1 ns/s"

Simulated time (if non-zero) divided by wall time. e.g. *123 ns/s* means 123 simulated nanoseconds took 1 second of wall time; for a model with only a 1 GHz clock that would be equivalent to 123.1 cycles per second. The units are automatically selected to give a number between 1 and 1000. The wall time includes initialization, initial and final process blocks, so indicates a slower speed than if the model had a longer runtime.

"cpu 22 s"

CPU time used total across all CPU threads in seconds.

"4 threads"

Number of simultaneous threads used.

"alloced 123 MB"

Total memory used during simulation in megabytes.

7.2 Benchmarking & Optimization

For best performance, run Verilator with the -O3 --x-assign fast --x-initial fast --no-assert options. The -O3 option will require a longer time to run Verilator, and --x-assign fast --x-initial fast may increase the risk of reset bugs in trade for performance. The --no-assert will suppress checking assertions, which is faster and appropriate for known-good models running software, but may hide design errors. See the above documentation for these options.

If using Verilated multithreaded, consider overriding Verilator's default thread-to-processor assignment by using numactl; 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 clocked latch used to gate clocks and gained a 60% performance improvement.

Beyond that, the performance of a Verilated model depends primarily on your C++ compiler and the size of your CPU's caches. Experience shows that the instruction cache size often limits large models, and 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 on the fast path. This is mostly code that is executed every cycle. OPT_SLOW applies to slow-path code, which rarely executes, often only once at the beginning or end of the simulation. 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 and 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 minimal 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 compared to many Verilated models, disabling optimization on the runtime library should not seriously affect overall compilation time but may have a 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 that 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 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.

7.3 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*.

7.3.1 Functional Coverage

With --coverage or --coverage-user, Verilator will translate functional coverage points the user has inserted manually in SystemVerilog code through into the Verilated model.

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

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

7.3.2 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 counter is incremented. At the end of a test, the counters, filename, and line number corresponding to each counter are written into the coverage file.

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

7.3.3 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, and 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, so 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 receiving logic could never see the edge. This algorithm may improve in the future. The net result is that 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.

7.3.4 Expression Coverage

With --coverage or --coverage-expr, Verilator will automatically add coverage analysis at each expression, indicating with a truth table how every Boolean truth-table possibility in the expression occurred.

Multi-bit expressions are ignored, but sub-expressions with are entirely Boolean are analyzed. Expression coverage does not fully explore the truth table of an expression, rather is looks at each term's contribution. E.g. an AND operation will check coverage for TT, XF and FX.

Some expressions may produce too many cover points. Verilator limits the maximum number of cover points per expression to 32, but this may be controlled with --coverage-expr-max.

Below is an example showing expression coverage produced from *verilator_coverage* as applied to the condition expression inside an if statement. Each line shows the minimum number of terms and their values (e.g. *(t1==0 && t2==1)*) needed to reach a result for the overall expression (e.g. *=> 1*). Each line also shows the number of times this combination was hit. Note that individual lines are not mutually exclusive.

```
%000004        if ((~t1 && t2) || (~t3 && t4)) $write("");
-000002  point: comment=(t1==0 && t2==1) => 1 hier=top.t
-000002  point: comment=(t1==1 && t3==1) => 0 hier=top.t
-000004  point: comment=(t1==1 && t4==0) => 0 hier=top.t
-000002  point: comment=(t2==0 && t3==1) => 0 hier=top.t
-000003  point: comment=(t2==0 && t4==0) => 0 hier=top.t
-000002  point: comment=(t3==0 && t4==1) => 1 hier=top.t
```

7.3.5 Suppressing Coverage

Using /*verilator&32;coverage_off*/ and /*verilator&32;coverage_on*/ around a block of code will disable and enable coverage respectively around that block. Or, use the coverage_block_off control file option.

Verilator automatically disables coverage of lines and branches with a $stop in them, as it is assumed that $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.

7.3.6 Coverage Collection

When any coverage flag is 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, write the coverage with either:

1. Using --binary or --main, and Verilator will dump coverage when the test completes to the filename specified with +verilator+coverage+file+<filename>.

2. 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 the file specified above, e.g. logs/coverage.dat.

After running all of the tests, execute the verilator_coverage command, passing arguments pointing to the filenames of all the individual coverage files. verilator_coverage will read 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, computing which tests are important to give full verification coverage on 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 the merging of coverage data files or other transformations.

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

7.4 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. Make sure the Verilog code will call *$finish* at the end of simulation (otherwise the C library may not correctly create the *gmon.out* file in the later steps below).
2. Run Verilator, adding the --prof-cfuncs option.
3. Build and run the simulation model.
4. The model will create *gmon.out*.
5. Run gprof gmon.out > gprof.log to see where in the C++ code the time is spent.
6. Run verilator_profcfunc gprof.log > profcfunc.log to take the gprof output and translate into output showing the Verilog line numbers on which most of the time is being spent.

7.5 Execution Profiling

For performance optimization, it is helpful to see statistics and visualize how execution time is distributed in a verilated model.

With the --prof-exec option, Verilator will:

- Add code to the Verilated model to record execution flow.
- Add code to save profiling data in non-human-friendly form to the file specified with +verilator+prof+exec+file+<filename>.
- In multithreaded models, add code to record each macro-task's start and end time across several calls to eval. (What is a macro-task? See the Verilator internals document (docs/internals.rst in the distribution.)

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

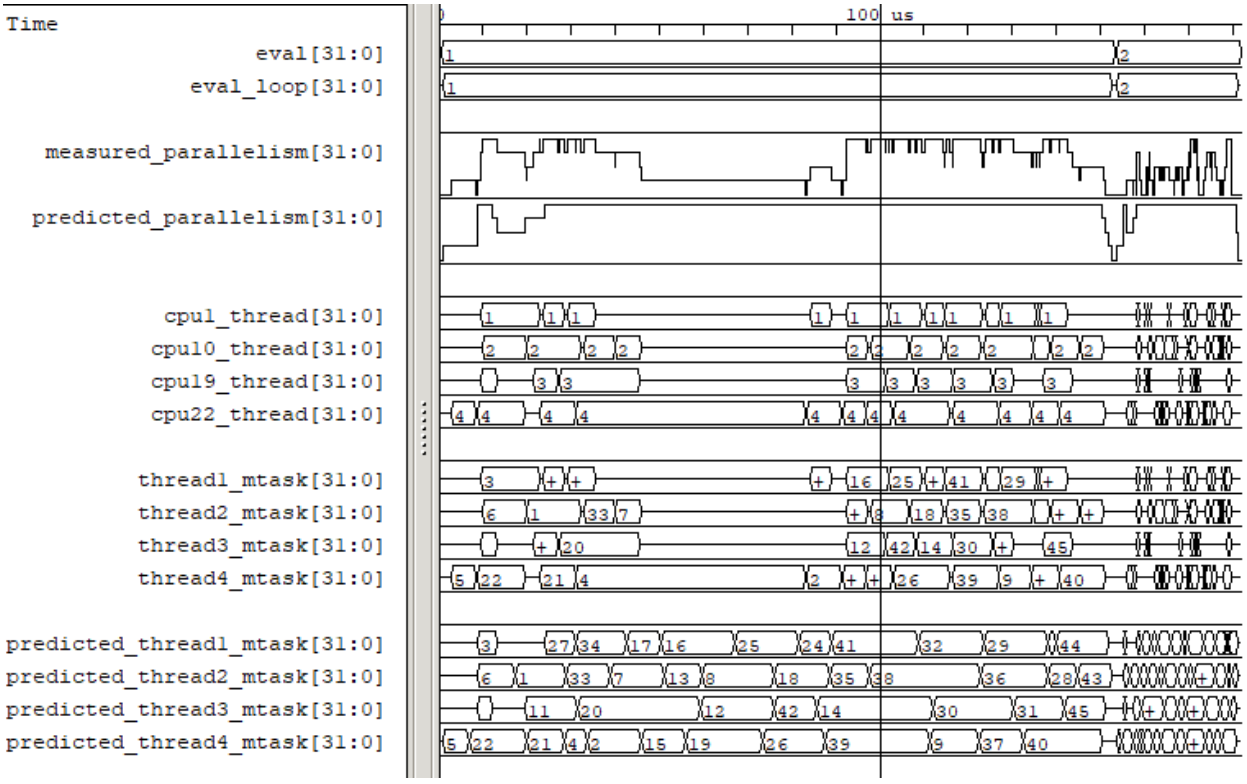


Fig. 7.1: Example verilator_gantt output, as viewed with GTKWave.
The measured_parallelism shows the number of CPUs being used at a given moment.
The cpu_thread section shows which thread is executing on each physical CPU.
The thread_mtask section shows which macro-task is running on a given thread.

For more information, see verilator_gantt.

7.6 Profiling ccache efficiency

The Verilator-generated Makefile supports 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 minor code changes will usually require rebuilding a large number of files. Improving ccache efficiency during the edit/compile/test loop is an active development area.

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.

7.7 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 and then call the << or >> operators on the generated model and any other data the process needs to be 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
    os << *topp;
}
void restore_model(const char* filenamep) {
    VerilatedRestore os;
    os.open(filenamep);
    os >> main_time;
    os >> *topp;
}
```

7.8 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.

7.8.1 Thread Profile-Guided Optimization

Verilator supports profile-guided optimization (Verilation) of multithreaded models (Thread PGO) to improve 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-pgo option. This will code to the verilated model to save profiling data for profile-guided optimization.

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

Rerun Verilator, optionally omitting the --prof-pgo 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 directly on the verilator command line without any option 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 file may be fed as separate command line options into Verilator. Verilator will sum the profile results, so a long-running test will have more weight for optimization proportionally 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 a 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.

7.8.2 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.

Please see the appropriate compiler documentation to use PGO with GCC or Clang. 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.

7.9 Runtime Debugging

To debug a Verilated executable, Verilate with --runtime-debug. This will instruct the compiler to insert debugger, and enable various library assertions. These options slow down the executable, so do this only when debugging.

If you are using your own Makefiles, adapt appropriately to pass the options documented under --runtime-debug to the compiler and linker.

Once you have a debugging-enabled executable, run it using the the standard GNU debugger gdb or a similar tool, and create a backtrace; e.g.:

```
gdb obj_dir/Vtop
 run {Vtop_command_arguments}
 {Vtop prints output, perhaps a segmentation faults}
 bt
```

Rarely the bug may disappear with --runtime-debug; if so, try instead using the sub-options that --runtime-debug documents, to find the maximum subset that still shows the issue. E.g. it is likely that using *-CFLAGS -D_GLIBCXX_DEBUG* will not hide any bug, so may be used.

Using --runtime-debug or *-CFLAGS -DVL_DEBUG=1* will only print a message if something goes wrong. To enable debug print messages at runtime, additionally use the +verilator+debug runtime option.

CHAPTER

EIGHT

CONTRIBUTING AND REPORTING BUGS

8.1 Announcements

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

8.2 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 (see: *Minimizing bug-inducing code*). Even better, create a test in the test_regress/t directory, as follows:

```
cd test_regress
cp -p t/t_EXAMPLE.py t/t_BUG.py
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.py documentation which can be seen by running:

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

Edit t/t_BUG.py 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.py  # Run on Verilator
t/t_BUG.py --debug # Run on Verilator, passing --debug to Verilator
t/t_BUG.py --vcs  # Run on VCS simulator
t/t_BUG.py --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.py --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.

8.3 Minimizing bug-inducing code

In some cases, the part of the code that causes the bug is clearly visible and the design can be easily manually reduced. In other cases, the bug is caused by a complex interaction of many parts of the design, and it is not clear which parts are necessary to reproduce the bug. In these cases, an Open Source tool called sv-bugpoint can be used to automatically reduce a SystemVerilog design to the smallest possible reproducer. It can be used to automatically reduce a design with hundreds of thousands of lines to a minimal test case while preserving the bug-inducing behavior.

With --debug, Verilator will write a *{prefix}*__inputs.vpp file which has all of the individual input files combined and pre-processed, this is often useful as the input design into *sv-bugpoint*.

Please refer to sv-bugpoint README for more information on how to use *sv-bugpoint*.

8.4 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.

8.4.1 Did you find a Verilator bug?

- Please **ensure the bug was not already reported** by searching Verilator Issues.
- Please **download the latest development GitHub version**, build, and see if the issue has been fixed.
- 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 Verilator Internals Documentation.

8.4.2 Did you write a patch that fixes a Verilator bug?

- Please Open a new Verilator issue if there is not one already describing the bug.
- Please Open a Verilator pull request.
- See the coding conventions, and other developer information in docs/internals.rst in the distribution, or as rendered at Verilator Internals Documentation.
- 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.
- 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).
  - 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.

8.4.3 Do you have questions on Verilator?

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

8.4.4 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

NINE

FAQ/FREQUENTLY ASKED QUESTIONS

9.1 Questions

9.1.1 Can I contribute?

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

9.1.2 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 four leading 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.

9.1.3 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 regularly tested. Verilated output also compiles under Microsoft Visual C++, but this is also not regularly tested.

9.1.4 Can you provide binaries?

You can install Verilator via the system package manager (apt, yum, etc.) on many Linux distributions, including Debian, Ubuntu, SuSE, Red Hat, 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?

9.1.5 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 must comply 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, which is a good thing for getting working silicon.

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

Your SystemVerilog, VPI/DPI, or main() C++ code remains under your own license.

It's just like how using GCC on your programs does not change the copyright of your program; 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.

Some examples:

- Any SystemVerilog or other input fed into Verilator remains your own.
- Any of your VPI/DPI C++ routines that Verilator calls remain your own.
- Any of your main() C++ code that calls into Verilator remains your own.
- If you change Verilator itself, for example, changing or adding a file under the src/ directory in the repository, you must make the source code available under the GNU Lesser Public License.
- If you change a header Verilator provides, for example, under include/ in the repository, you must make the source code available under the GNU Lesser Public License.

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

Under both licenses, you can offer a commercial product based on Verilator directly or embedded within. However, under both licenses, any changes you make to Verilator for such a product must be open-sourced.

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.

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

Verilator may require more memory than the resulting simulation, as Verilator internally creates all of the state of the resulting generated simulator 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*.

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

See also the next question for tracing in SystemC mode.

A. Pass the --trace-vcd option to Verilator. Then you may use $dumpfile and $dumpvars to enable traces, the same as with any Verilog simulator, although Verilator ignores the arguments to $dumpvars. See examples/make_tracing_c in the distribution.

   If writing the top-level C code, call Verilated::traceEverOn(true); this is done for you if using --binary.

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) {
    const std::unique_ptr<VerilatedContext> contextp{new VerilatedContext};
    ...
    Verilated::traceEverOn(true);
```

(continues on next page)

(continued from previous page)

```
    VerilatedVcdC* tfp = new VerilatedVcdC;
    topp->trace(tfp, 99);  // Trace 99 levels of hierarchy (or see below)
    // tfp->dumpvars(1, "t");  // trace 1 level under "t"
    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 you are using the Verilator --binary or --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.

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

A. Pass the --trace-vcd 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, 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 standard 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 that the timescale and timeprecision will be inherited from your SystemC settings.

```
#include "verilated_vcd_sc.h"
...
int main(int argc, char** argv) {
    ...
    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 you are using the Verilator --binary or --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.

9.1.10 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:

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;
```

Currently, supporting FST and VCD in a single simulation is not supported, but such usage should be unlikely. You can however ifdef around the trace format in your C++ main loop, and select VCD or FST at compile time.

9.1.11 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, Surfer, Dinotrace (legacy), or any of the many closed-source viewer offerings; FST is supported only by GTKWave and Surfer.

9.1.12 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 tracing depth, 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. Write your trace files to a machine-local solid-state drive instead of a network drive. Network drives are generally far slower.

9.1.13 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.

9.1.14 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, and 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 parsed as Verilog 2001, see the --default-language option.

9.1.15 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 are disabled.

9.1.16 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.

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

9.1.17 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 or --binary, this is done for you.

9.1.18 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 write 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.

9.1.19 How do I make a Verilog module that contains a C++ object?

You need to add the object to the structure 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.

9.1.20 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 is 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, sccache, distcc, or 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*.

9.1.21 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, resulting in 2-3 fewer assembly instructions for each signal access. This makes the execution time 10-15% faster, but can result in more compilations when something changes.

9.1.22 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.

9.1.23 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.

9.1.24 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 optimize the model better 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 fit the width of the signal. Generally, you can use just uint32_t's for 1 to 32 bits, or uint64_t for 1 to 64 bits, and the compiler will properly up-convert smaller entities. Note that 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 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"
std::cout << "clock is " << top->our->clk << std::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 your code shouldn't change clocks, or you'll get strange results.

9.1.25 Should a module be in Verilog or SystemC?

Sometimes there is a block that only interconnects instances, and you have a choice if you write it in Verilog or SystemC. Everything else being equal, the 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

TEN

INPUT LANGUAGES

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

10.1 Language Standard Support

10.1.1 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.

10.1.2 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.

10.1.3 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 allowed syntax.

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

10.1.4 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.

10.1.5 SystemVerilog 2017 (IEEE 1800-2017) Support

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

10.1.6 SystemVerilog 2023 (IEEE 1800-2023) Support

Verilator supports some of the 2023 improvements, including triple-quoted string blocks that may include newlines and single quotes.

Verilator implements a full IEEE 1800-2023 compliant preprocessor, including triple-quoted strings, and `ifdef expressions.

10.1.7 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 Verilog IEEE 1364 or SystemVerilog IEEE 1800.

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

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

10.1.8 Synthesis Directive Assertion Support

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, unless --no-assert-case option is used, 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 statements, as well as "unique" on if statements. However, "priority if" is currently ignored.

10.2 Time

With --timing, all timing controls are supported:

- delay statements,
- event control statements not only at the top of a process,
- intra-assignment timing controls,
- net delays,
- wait statements,

as well as all flavors of fork.

Compiling a Verilated design that uses these features requires a compiler with C++20 coroutine support, e.g. Clang 5, GCC 10, or newer.

#0 delays cause Verilator to issue the ZERODLY warning, as they work differently than described in the LRM. They do not schedule process resumption in the Inactive region, though the process will get resumed in the same time slot.

Rising/falling/turn-off delays are currently unsupported and cause the RISEFALLDLY warning.

Minimum/typical/maximum delays are currently unsupported. The typical delay is always the one chosen. Such expressions cause the MINTYPMAX warning.

Another consequence of using --timing is that the --main option generates a main file with a proper timing eval loop, eliminating the need for writing any driving C++ code. You can simply compile the simulation (perhaps using --build) and run it.

With --no-timing, all timing controls cause the NOTIMING error, except:

- delay statements - they are ignored (as they are in synthesis), though they do issue a STMTDLY warning,
- intra-assignment timing controls - they are ignored, though they do issue an ASSIGNDLY warning,
- net delays - they are ignored,
- event controls at the top of the procedure,

Forks cause this error as well, except:

- forks with no statements,
- fork..join or fork..join_any with one statement,
- forks with --bbox-unsup.

If neither --timing nor --no-timing is specified, all timing controls cause the NEEDTIMINGOPT error, except event controls at the top of the process. Forks cause this error as well, except:

- forks with no statements,
- fork..join or fork..join_any with one statement,
- forks with --bbox-unsup.

Timing controls and forks can also be ignored in specific files or parts of files. The /*verilator&32;timing_off*/ and /*verilator&32;timing_off*/ metacomments will make Verilator ignore the encompassed timing controls and forks, regardless of the chosen --timing or --no-timing option. This can also be achieved using the timing_off and timing_off options in Verilator Control Files.

10.3 Language Limitations

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

10.3.1 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 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 will 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.

10.3.2 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 is replaced with ___05F (5F is the hex code of an underscore.)

10.3.3 Bind

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

10.3.4 Class

Verilator class support is limited but in active development. Verilator supports members, methods, class extend, and class parameters.

10.3.5 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.

10.3.6 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.

10.3.7 Structures and Unions

All structures and unions are scheduled together, which means that generating one member of a structure from blocking, and another from non-blocking assignments is unsupported.

10.3.8 Unknown States

Verilator is mostly a two-state simulator, not a four-state simulator. However, it has two features that 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 than 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 assign a constant value as determined by the --x-assign option. This allows runtime randomization; thus, if the value is used, the random value should cause downstream errors. Integers also get randomized, even though the Verilog 2001 specification says they initialize to zero. However, randomization happens at initialization time; 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. You can determine that reset is working correctly by running several random simulation runs. 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 that this is what the hardware will do.) In practice, setting all variables to one at startup finds the most problems (since control signals are typically 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.

10.3.9 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.

10.3.10 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. User-defined primitive (UDP) tables are supported.

10.3.11 Specify blocks

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

10.3.12 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.

10.3.13 Array Out of Bounds

Writing a memory element 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 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 an unspecified constant of the appropriate width.

10.3.14 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.

10.3.15 Encrypted Verilog

Open-source simulators like Verilator cannot use encrypted RTL (i.e. IEEE P1735). Talk to your IP vendor about delivering IP blocks via Verilator's `--protect-lib` feature.

10.4 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, `else, `elsif, `end_keywords, `endif, `error, `ifdef, `ifndef, `include, `line, `systemc_ctor, `systemc_dtor, `systemc_header, `systemc_header_post, `systemc_imp_header, `systemc_implementation, `systemc_interface, `undef, `verilog
: Fully supported.

always, always_comb, always_ff, always_latch, and, assign, begin, buf, byte, case, casex, casez, default, defparam, do-while, else, end, endcase, endfunction, endgenerate, endmodule, endspecify, endtask, final, for, function, generate, genvar, if, initial, inout, input, int, integer, localparam, logic, longint, macromodule, module, nand, negedge, nor, not, or, output, parameter, posedge, reg, scalared, shortint, signed, supply0, supply1, task, time, tri, typedef, var, vectored, while, wire, xnor, xor
: Generally supported.

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

'{} operator
Assignment patterns with an order based, default, constant integer (array) or member identifier (struct/union) keys are supported. Data type keys and keys 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 specified as illegal on chandles.

checker
Treated as a "module"; does not yet warn about many constructs illegal inside a checker.

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.

force, release
Verilator supports the procedural *force* (and corresponding *release*) statement. However, the behavior of the *force* statement does not entirely comply with IEEE 1800-2023:

1. Using forced variable as a value to another force statement is currently not supported. The dependent force statement is forced by an initial constant value.
2. Force/release with procedural continuous assignment is not supported. The assignment is treated as procedural.
3. Expressions using multiple variable references or function calls on forced right-hand side are not sensitive to dependency changes.

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 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 timing checks and specify blocks (except specparam, which is supported) 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 (based on the --trace-vcd 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, 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. Verilator and the Verilog specification do not include support for readmem to multi-dimensional arrays.

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

```
{VerilatedContext*} ->commandArgs(argc, argv);
```

to register the command line before calling $test$plusargs or $value$plusargs. Or use --binary or --main.

CHAPTER

ELEVEN

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.

""" [string] """

A triple-quoted block specifies a string that may include newlines and single quotes. This extension was standardized in IEEE 1800-2023.

`__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 IEEE 1800-2009.

`__LINE__

The `__LINE__ define expands to the current line number like C++'s __LINE__. This Verilator feature added in 2006 was incorporated into IEEE 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 *Verilator Control Files*.

`error [string]

This will report an error when the preprocessor emits it, similar to C++'s #error.

`line

As a special case *`line `__LINE__ "filename"* allows setting the filename, without changing the line number. This is used for some internal tests, so that debugging can leave the line numbers correctly referring to the test file's line numbers.

`systemc_class_name

Inside one of the `systemc_... text blocks, replaced with the C++ class name generated for the given containing SystemVerilog class or module. Currently this is replaced blindly, ignoring quoting or other escapes; this behavior may change in the future. This attribute is indented only to be used internally in *verilated_std.sv*.

`systemc_ctor

Take the 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 or class item, e.g., directly inside a module/endmodule or class/endclass pair. Despite the name of this macro, this also works in pure C++ code.

`systemc_dtor

Take the 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 or class item, e.g., directly inside a module/endmodule or class/endclass pair. Despite the name of this macro, this also works in pure C++ code.

`systemc_header

Take the 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 or class item, e.g., directly inside a module/endmodule or class/endclass pair. Despite the name of this macro, this also works in pure C++ code.

`systemc_header_post

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

`systemc_imp_header

Take the 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 or class item, e.g., directly inside a module/endmodule or class/endclass pair. Despite the name of this macro, this also works in pure C++ code.

`systemc_implementation

Take the 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 or class item, e.g., directly inside a module/endmodule or class/endclass 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.

`systemc_interface

Take the 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 or class item, e.g., directly inside a module/endmodule or class/endclass pair. Despite the name of this macro, this also works in pure C++ code.

`SYSTEMVERILOG

The SYSTEMVERILOG, SV_COV_START, and related standard predefined macros are defined by default when --language is "1800-*".

`VERILATOR

`verilator

`verilator3

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

`verilator_config

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

`VERILATOR_TIMING

The VERILATOR_TIMING define is set when --timing is used to allow an "`ifdef" of code dependent on this feature. Note that this define is not affected by the timing_off control file option nor timing metacomments.

`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.

$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 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 a number to $c, which specifies the bit width of the output, e.g., 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 extension was standardized into 1800-2009.

$stacktrace

Called as a task, print a stack trace. Called as a function, return a string with a stack trace. This relies on the C++ system trace, which may give less meaningful results if the model is not compiled with debug symbols. Also, the data represents the C++ stack; the SystemVerilog functions/tasks involved may be renamed and/or inlined before becoming the C++ functions that may be visible in the stack trace. This extension was standardized in IEEE 1800-2023.

$timeprecision

Returns the timeprecision of the model as an integer. This extension is experimental and may be removed without deprecation.

$timeunit

Returns the timeunit of the current module as an integer. This extension is experimental and may be removed without deprecation.

/*verilator&32;clock_enable*/

Deprecated and has no effect (ignored).

In versions before 5.000:

Used after a signal declaration to indicate the signal is used to gate a clock, and the user is responsible for ensuring 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.

Same as clock_enable control file option.

/*verilator&32;clocker*/

/*verilator&32;no_clocker*/

Specifies whether the signal is used as clock or not. 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 control file option.

/*verilator&32;coverage_off*/

Specifies that 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 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;forceable*/

Specifies that the signal (net or variable) should be made forceable from C++ code by generating public *<signame>__VforceEn* and *<signame>__VforceVal* signals. The force control signals are created as public_flat signals.

To force a marked signal from C++, set the corresponding *__VforceVal* variable to the desired value, and the *__VforceEn* signal to the bit-mask indicating which bits of the signal to force. To force all bits of the target signal, set *__VforceEn* to all ones. To release the signal (or part thereof), set appropriate bits of the *__VforceEn* signal to zero.

Same as forceable in configuration files.

/*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 control 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 large combinatorial block results in a UNOPTFLAT warning, attaching this to the signal that was causing a false loop may work around the warning.

IE, with the following:

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

(continues on next page)

(continued from previous page)

```
  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 control 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 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 would now be restored to on, and if it was off before the lint_off it would 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 control 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 control 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 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 control file option.

/*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 making a DPI or public function that accesses the variable. This is nicer as it provides an obvious entry point compatible across simulators.

Same as public control file option.

/*verilator&32;public_[|flat|flat_rd|flat_rw]_on [@(<edge_list>)]*/ (as scope)

Used to wrap multiple signals and parameters with the respective public attribute. See attribute above for their respective behavior. Cannot be nested. e.g:

```
/*verilator public_flat_rw_on*/
logic clk;
logic rst;
parameter width = 8;
/* verilator public_off*/
logic data;
```

Is equivalent to:

```
logic clk /*verilator public_flat_rw*/;
logic rst /*verilator public_flat_rw*/;
parameter width /*verilator public_flat_rw*/ = 8;
logic data;
```

/*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 control 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 control 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 writable, where writes should be considered to have the timing specified by the given sensitivity edge list. Use of this is implied when using the --public-flat-rw option.

Same as public_flat_rw control 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 public signals or `systemc_ directives. Use of this is implied when using the --public option.

Same as public control file option.

/*verilator&32;public_off*/

Terminates the previous */*verilator public*_on*/* directive; see above.

/*verilator&32;sc_bv*/

Used after a port declaration. It sets the port to be of sc_bv<{width}> type, instead of bool, uint32_t, or uint64_t. This may be useful if the port width is parameterized and the instantiating C++ code always wants to have a sc_bv accept any width. In general, you should avoid using this attribute when unnecessary, as the performance decreases significantly with increasing usage of sc_bv.

Same as sc_bv control 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;sformat*/

Attached to the final argument of type "input string" of a function or task to indicate that 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 control 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 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.

Packed variables that are only referenced locally (without hierarchical references) via non-overlapping, constant-indexed bit or part select expressions are split automatically. This covers the somewhat common usage pattern:

```
logic [1:0][31:0] tmp;

assign tmp[0] = foo    + a;
assign tmp[1] = tmp[1] + b;
assign bar    = tmp[1] + c;
```

Same as split_var control 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;timing_off*/

Ignore all timing constructs after this metacomment. All timing controls behave as if they were not there (the same way as with --no-timing), and fork/join* blocks are converted into begin/end blocks.

Same as timing_off control file option.

/*verilator&32;timing_on*/

Re-enable all timing constructs after this metacomment (only applicable after timing_off).

Same as timing_on control file option.

/*verilator&32;trace_init_task*/

Removed.

In versions before 5.024:

Attached to a DPI import to indicate that function should be called when initializing tracing. This attribute is indented only to be used internally in code that Verilator generates when --lib-create or --hierarchical is used along with --trace-vcd.

/*verilator&32;tracing_off*/

Disable waveform tracing for all future signals 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.

/*verilator&32;unroll_disable*/

Used in a statement position to indicate the immediately following loop at the same statement level should not be unrolled by Verilator, ignoring --unroll-count. This is similar to clang's #pragma clang loop unroll(disable).

This option does not currently disable the C++ compiler's unrolling (or not) of any loops that make it through to the Verilated C++ code.

/*verilator&32;unroll_full*/

Rarely needed. Used in a statement position to indicate the immediately following loop at the same statement level should always be fully unrolled by Verilator, ignoring --unroll-count. This is similar to clang's #pragma clang loop unroll(full).

CHAPTER

TWELVE

EXECUTABLE AND ARGUMENT REFERENCE

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

12.1 verilator Arguments

The following arguments 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>
 +1800-2023ext+<ext>        Use SystemVerilog 2023 with file extension <ext>
--no-assert                 Disable all assertions
--no-assert-case            Disable unique/unique0/priority-case assertions
--autoflush                 Flush streams after all $displays
--bbox-sys                  Blackbox unknown $system calls
--bbox-unsup                Blackbox unsupported language features
--binary                    Build model binary
--build                     Build model executable/library after Verilation
--build-dep-bin <filename>  Override build dependency Verilator binary
--build-jobs <jobs>         Parallelism for --build
--cc                        Create C++ output
 -CFLAGS <flags>            C++ compiler arguments for makefile
--clk <signal-name>         Mark specified signal as clock
--no-clk <signal-name>      Prevent marking specified signal as clock
--compiler <compiler-name>  Tune for specified C++ compiler
--compiler-include          Include additional header in the precompiled one
--converge-limit <loops>    Tune convergence settle time
--coverage                  Enable all coverage
--coverage-expr             Enable expression coverage
--coverage-expr-max <value>     Maximum permutations allowed for an expression
--coverage-line             Enable line coverage
```

(continues on next page)

(continued from previous page)

```
--coverage-max-width <width>   Maximum array depth for coverage
--coverage-toggle              Enable toggle coverage
--coverage-underscore          Enable coverage of _ signals
--coverage-user                Enable SVL user coverage
 -D<var>[=<value>]             Set preprocessor define
--debug                        Enable debugging
--debug-check                  Enable debugging assertions
--no-debug-leak                Disable leaking memory in --debug mode
--debugi <level>               Enable debugging at a specified level
--debugi-<srcfile> <level>     Enable debugging a source file at a level
--no-decoration                Disable comments and lower spacing level
--decorations <level>          Set output comment and spacing level
--default-language <lang>      Default language to parse
 +define+<var>=<value>         Set preprocessor define
--diagnostics-sarif            Enable SARIF diagnostics output
--diagnostics-sarif-output <filename>  Set SARIF diagnostics output file
--dpi-hdr-only                 Only produce the DPI header file
--dump-<srcfile>               Enable dumping everything in source file
--dump-defines                 Show preprocessor defines with -E
--dump-dfg                     Enable dumping DfgGraphs to .dot files
--dump-graph                   Enable dumping V3Graphs to .dot files
--dump-tree                    Enable dumping Ast .tree files
--dump-tree-addrids            Use short identifiers instead of addresses
--dump-tree-dot                Enable dumping Ast .tree.dot debug files
--dump-tree-json               Enable dumping Ast .tree.json files and .tree.meta.json file
--dumpi-<srcfile> <level>      Enable dumping everything in source file at level
--dumpi-dfg <level>            Enable dumping DfgGraphs to .dot files at level
--dumpi-graph <level>          Enable dumping V3Graphs to .dot files at level
--dumpi-tree <level>           Enable dumping Ast .tree files at level
--dumpi-tree-json <level>      Enable dumping Ast .tree.json files at level
 -E                            Preprocess, but do not compile
--emit-accessors               Emit getter and setter methods for model top class
--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
--future0 <option>             Ignore an option for compatibility
--future1 <option>             Ignore an option with argument for compatibility
 -fno-<optimization>           Disable internal optimization stage
 -G<name>=<value>              Overwrite top-level parameter
--gate-stmts <value>           Tune gate optimizer depth
--gdb                          Run Verilator under GDB interactively
--gdbbt                        Run Verilator under GDB for backtrace
--generate-key                 Create random key for --protect-key
--get-supported <feature>      Get if feature is supported
--getenv <var>                 Get environment variable with defaults
--help                         Show this help
--hierarchical                 Enable hierarchical Verilation
--hierarchical-params-file <name> Internal option that specifies parameters file for hier blocks
```

(continues on next page)

(continued from previous page)

```
--hierarchical-threads <threads>  Number of threads for hierarchical scheduling
 -I<dir>                          Directory to search for includes
--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
 -j <jobs>                        Parallelism for --build-jobs/--verilate-jobs
--no-json-edit-nums               Don't dump editNum in .tree.json files
--no-json-ids                     Don't use short identifiers instead of addresses/paths in .tree.json
--json-only                       Create JSON parser output (.tree.json and .meta.json)
--json-only-meta-output <filename>  Set .tree.meta.json output filename
--json-only-output <filename>     Set .tree.json output filename
--l2-name <value>                 Verilog scope name of the top module
--language <lang>                 Default language standard to parse
 -LDFLAGS <flags>                 Linker pre-object arguments for makefile
--lib-create <name>               Create a DPI library
 +libext+<ext>+[ext]...           Extensions for finding modules
--lint-only                       Lint, but do not make output
--localize-max-size <value>       Tune localize optimization variable size
--main                            Generate C++ main() file
--main-top-name                   Specify top name passed to Verilated model in generated C++ main
--make <build-tool>               Generate scripts for specified build tool
 -MAKEFLAGS <flags>               Arguments to pass to make during --build
--max-num-width <value>           Maximum number width (default: 64K)
--Mdir <directory>                Name of output object directory
--MMD                             Create .d dependency files
--mod-prefix <topname>            Name to prepend to lower classes
--MP                              Create phony dependency targets
 +notimingchecks                  Ignored
 -o <executable>                  Name of final executable
 -O0                              Disable optimizations
 -O3                              High-performance optimizations
 -O<optimization-letter>          Selectable optimizations
--output-groups <numfiles>        Group .cpp files into larger ones
--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-inout-enables              Specify that __en and __out signals be created for inouts
--pins-sc-biguint                 Specify types for top-level ports
--pins-sc-uint                    Specify types for top-level ports
--pins-sc-uint-bool               Specify types for top-level ports
--pins-uint8                      Specify types for top-level ports
--no-pins64                       Don't use uint64_t's for 33-64 bit sigs
--pipe-filter <command>           Filter all input through a script
--pp-comments                     Show preprocessor comments with -E
--prefix <topname>                Name of top-level class
--preproc-resolve                 Include all found modules in the output with -E
--preproc-token-limit             Maximum tokens on a line allowed by preprocessor
--private                         Debugging; see docs
--prof-c                          Compile C++ code with profiling
```

(continues on next page)

(continued from previous page)

```
--prof-cfuncs               Name functions for profiling
--prof-exec                 Enable generating execution profile for gantt chart
--prof-pgo                  Enable generating profiling data for PGO
--protect-ids               Hash identifier names for obscurity
--protect-key <key>         Key for symbol protection
--protect-lib <name>        Create a DPI protected library
--public                    Mark signals as public; see docs
--public-depth <level>      Mark public to specified module depth
--public-flat-rw            Mark all variables, etc as public_flat_rw
--public-ignore             Ignore all public comment markings
--public-params             Mark all parameters as public_flat
 -pvalue+<name>=<value>     Overwrite toplevel parameter
--quiet                     Minimize additional printing
--quiet-exit                Don't print the command on failure
--quiet-stats               Don't print statistics
--relative-includes         Resolve includes relative to current file
--reloop-limit <value>      Minimum iterations for forming loops
--report-unoptflat          Extra diagnostics for UNOPTFLAT
--rr                        Run Verilator and record with rr
--runtime-debug             Enable model runtime debugging
--savable                   Enable model save-restore
--sc                        Create SystemC output
--no-skip-identical         Disable skipping identical output
--stats                     Create statistics file
--stats-vars                Provide statistics on variables
--no-std                    Prevent loading standard files
--no-std-package            Prevent parsing standard package
--no-std-waiver             Prevent parsing standard lint waivers
--no-stop-fail              Do not call $stop when assertion fails
--structs-packed            Convert all unpacked structures to packed structures
 -sv                        Enable SystemVerilog parsing
 +systemverilogext+<ext>    Synonym for +1800-2023ext+<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
--timing                    Enable timing support
--no-timing                 Disable timing support
--top <topname>             Alias of --top-module
--top-module <topname>      Name of top-level input module
--trace                     Enable VCD 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 array depth for tracing
--trace-max-width <width>   Maximum bit width for tracing
--trace-params              Enable tracing of parameters
--trace-saif                Enable SAIF file creation
--trace-structs             Enable tracing structure names
--trace-threads <threads>   Enable FST waveform creation on separate threads
--no-trace-top              Do not emit traces for signals in the top module generated by verilator
```

(continues on next page)

(continued from previous page)

```
--trace-underscore          Enable tracing of _signals
--trace-vcd                 Enable VCD waveform creation
 -U<var>                    Undefine preprocessor define
--no-unlimited-stack        Don't disable stack size limit
--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
--valgrind                  Run Verilator under valgrind
--no-verilate               Skip Verilation and just compile previously Verilated code
--verilate-jobs             Job threads for Verilation stage
 +verilog1995ext+<ext>      Synonym for +1364-1995ext+<ext>
 +verilog2001ext+<ext>      Synonym for +1364-2001ext+<ext>
--version                   Show program version and exits
--vpi                       Enable VPI compiles
--waiver-multiline          Create multiline --match for waivers
--waiver-output <filename>  Create a waiver file based on 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
 -work <libname>            Set config library for following files
 -Wpedantic                 Warn on compliance-test issues
 -Wwarn-<message>           Enable specified warning message
 -Wwarn-lint                Enable lint warning message
 -Wwarn-style               Enable style warning message
--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.a/.o/.so>

Specifies optional object or library files to be linked 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 the appropriate directory to your makefile's VPATH 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.

<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. The compiler flags add by default *-DVERILATOR=1*, so an *#ifdef VERILATOR* may be used to conditionally preprocess .cpp code for different simulators.

<file.v>

Specifies the Verilog file containing the top module to be Verilated.

+1364-1995ext+<ext>

+1364-2001ext+<ext>

+1364-2005ext+<ext>

+1800-2005ext+<ext>

+1800-2009ext+<ext>

+1800-2012ext+<ext>

+1800-2017ext+<ext>

+1800-2023ext+<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. 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.

--no-assert

Disable all assertions. Implies --no-assert-case.

In versions before 5.038, these were disabled by default, and *–assert* was required to enable assertions.

--no-assert-case

Disable unique/unique0/priority case related checks.

In versions before 5.038, these were disabled by default, and *–assert* or *–assert-case* was required to enable case 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 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 of the rest of the design even when unsupported constructs are present.

Using this argument will likely cause incorrect simulation.

--binary

Create a Verilated simulator binary. Alias for --main --exe --build --timing.

See also -j.

--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.

--build cannot be specified when using -E, --dpi-hdr-only, --lint-only, or --xml-only.

--build-dep-bin <filename>

Rarely needed. 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. Defaults to the full path to the Verilator binary.

This option was named *–bin* before version 4.228.

--build-jobs <value>

Specify the level of parallelism for --build. If zero, uses the number of threads in the current hardware. Otherwise, the <value> must be a positive integer specifying the maximum number of parallel build jobs.

If not provided, and -j is provided, the -j value is used.

This forms the make option -j value, unless the MAKEFLAGS environment variable contains -jobserver-auth, in which case Verilator assumes that make's jobserver is being used.

See also -j.

--cc

Specify C++ without SystemC output mode; see also the --sc option.

-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++).

--clk <signal-name>

With --clk, the specified signal is marked as a clock signal.

The provided signal name is specified using a RTL hierarchy path. For example, v.foo.bar. If the signal is the input to the 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 later used as individual bits, Verilator will attempt to decompose the vector and connect the single-bit clock signals.

In versions before 5.000, 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.

--no-clk <signal-name>

Prevent the specified signal from being marked as a clock. See --clk.

--compiler <compiler-name>

Enables workarounds for the specified C++ compiler (list below). This does not change any performance tuning options, but it may in the future. This also does not change default compiler flags; these are determined when Verilator was configured.

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.

--compiler-include <header-path>

Specifies additional headers to be included in the final PCH header. It is required to add them to this header, due to compilers' limitation that allow only one precompiled header per compilation. Use this instead of :-CFLAGS with *-include <header-path>*.

--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, an alias for --coverage-line --coverage-toggle --coverage-expr --coverage-user.

--coverage-expr

Enables expression coverage analysis. See *Expression Coverage*.

--coverage-expr-max <value>

Rarely needed. Specifies the maximum number of permutations able to be covered for a given expression. Defaults to 32. Increasing may slow coverage simulations and make analyzing the results unwieldy.

--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 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 relatively 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 --dumpi-tree 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 the 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.

--no-decoration

Alias for --decorations none.

--decorations <level>

When creating output Verilated code, set level of comment and whitespace decoration.

With "–decorations none",
Minimize comments, white space, symbol names, and other decorative items, at the cost of reduced readability. This may assist C++ compile times. This will not typically change the ultimate model's performance, but may in some cases. See also --no-decoration option.

With "–decorations medium",
The default, put a small amount of comments and white space, for typical level of readability.

With "–decorations node",
Include comments indicating what caused generation of the following text, including what node pointer

(corresponding to --dump-tree .tree printed data), and the source Verilog filename and line number. If subsequent following statements etc have the same filename/line number these comments are omitted. This enables easy debug when looking at the C++ code to determine what Verilog source may be related. As node pointers are not stable between different Verilator runs, this may harm compile caching and should only be used for debug.

--default-language <value>

Select the language 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", "1800-2023", 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 1800-2023) 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 relatively standard across Verilog tools while -D is similar to gcc -D.

--diagnostics-sarif

Enables diagnostics output into a Static Analysis Results Interchange Format (SARIF) file, a standard, JSON-based format for the output of static analysis tools such as linters. See [SARIF](https://sarifweb.azurewebsites.net/), [sarif-tools](https://github.com/microsoft/sarif-tools), and the [SARIF web-based viewer](https://microsoft.github.io/sarif-web-component/).

--diagnostics-sarif-output <filename>

Specifies the filename for the SARIF output file (*.sarif*) of --diagnostics-sarif. Using this option automatically sets --diagnostics-sarif. If not specified, output defaults to <prefix>.sarif.

--dpi-hdr-only

Only generate the DPI header file. This option does not affect on the name or location of the emitted DPI header file, it is output in --Mdir as it would be without this option.

--dump-<srcfile>

Rarely needed - for developer use. Enable all dumping in the given source file at level 3.

--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-dfg

Rarely needed. Enable dumping DfgGraph .dot debug files with dumping level 3.

--dump-graph

Rarely needed. Enable dumping V3Graph .dot debug files with dumping level 3. Before Verilator 4.228, --dump-tree used to include this option.

--dump-tree

Rarely needed. Enable dumping Ast .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-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 and then freed.

--dump-tree-dot

Rarely needed - for developer use. Enable dumping Ast .tree.dot debug files in Graphviz Dot format. This option implies --dump-tree, unless --dumpi-tree was passed explicitly.

--dump-tree-json

Rarely needed. Enable dumping Ast .json.tree debug files with dumping level 3, which dumps the standard critical stages. For details on the format, see the Verilator Internals manual.

--dumpi-<srcfile> <level>

Rarely needed - for developer use. Set the dumping level in the specified Verilator source file to the specified value (e.g., *–dumpi-V3Order 9*). Level 0 disables dumps and is equivalent to *–no-dump-<srcfile>*. Level 9 enables the dumping of everything.

--dumpi-dfg <level>

Rarely needed - for developer use. Set the internal DfgGraph dumping level globally to the specified value.

--dumpi-graph <level>

Rarely needed - for developer use. Set internal V3Graph dumping level globally to the specified value.

--dumpi-tree <level>

Rarely needed - for developer use. Set internal Ast dumping level globally to the specified value.

--dumpi-tree-json <level>

Rarely needed - for developer use. Set internal Ast JSON dumping level globally to the specified value.

-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 --no-std, which is implied by this.

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

--emit-accessors

Emit getter and setter methods for each top-level signal in the model top class. Signals are still available as public members, but with the *__Vm_sig_* prefix.

--error-limit <value>

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

It 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 relatively 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 relatively standard across Verilog tools.

The file may contain // comments which are ignored until 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.

-fdfg-synthesize-all

Rarely needed. Attempt to synthesize all combinational logic in DFG.

-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. Flattening large designs may require significant CPU time, memory and storage.

-fno-acyc-simp

-fno-assemble

-fno-case

-fno-combine

-fno-const

-fno-const-before-dfg

Rarely needed. Do not apply any global expression folding prior to the DFG pass. This option is solely for the purpose of DFG testing and should not be used otherwise.

-fno-const-bit-op-tree

-fno-const-eager

-fno-dedup

-fno-dfg

Rarely needed. Disable all use of the DFG-based combinational logic optimizer. Alias for -fno-dfg-pre-inline, -fno-dfg-post-inline and -fno-dfg-scoped.

-fno-dfg-break-cycles

Rarely needed. Disable breaking combinational cycles during DFG.

-fno-dfg-peephole

Rarely needed. Disable the DFG peephole optimizer.

-fno-dfg-peephole-<pattern>

Rarely needed. Disable individual DFG peephole optimizer pattern.

-fno-dfg-post-inline

Rarely needed. Do not apply the DFG optimizer after inlining.

-fno-dfg-pre-inline

Rarely needed. Do not apply the DFG optimizer before inlining.

-fno-dfg-scoped

Rarely needed. Do not apply the DFG optimizer across module scopes.

-fno-expand

-fno-func-opt

-fno-func-opt-balance-cat

-fno-func-opt-split-cat

-fno-gate

Rarely needed. Do not apply the gate-level wire optimizations. Using this is not recommended as may cause additional warnings and ordering issues.

-fno-inline

-fno-inline-funcs

-fno-life

-fno-life-post

-fno-localize

-fno-merge-cond

-fno-merge-cond-motion

-fno-merge-const-pool

-fno-reloop

-fno-reorder

-fno-slice

-fno-split

-fno-subst

-fno-subst-const

-fno-table

Rarely needed. Disables one of the internal optimization steps. These are typically used only when recommended by a maintainer to help debug or work around an issue.

-fno-var-split

Rarely needed. Do not attempt to split variables automatically. Variables explicitly annotated with /*verilator&32;split_var*/ are still split.

-future0 <option>

Rarely needed. Suppress an unknown Verilator option for an option that takes no additional arguments. This allows scripts written with pragmas for a later version of Verilator to run under an older version. e.g. -future0 option --option would on older versions that do not understand --option or +option suppress what would otherwise be an invalid option error, and on newer versions that implement --option, -future0 option --option would have the -future0 option ignored and the --option would function appropriately.

-future1 <option>

Rarely needed. Suppress an unknown Verilator option for an option that takes an additional argument. This allows scripts written with pragmas for a later version of Verilator to run under an older version. e.g. -future1 option --option arg would on older versions that do not understand --option arg or +option arg suppress what would otherwise be an invalid option error, and on newer versions that implement --option arg, -future1 option --option arg would have the -future1 option ignored and the --option arg would function appropriately.

-G<name>=<value>

Overwrites the given parameter of the top-level 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 appropriately 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 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, print a backtrace on exit, and 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.

--get-supported <feature>

If the given feature is supported, print "1" and exit immediately; otherwise, print a newline and exit immediately. This can be useful in makefiles. See also -V, and the various *.mk files.

Feature may be one of the following: COROUTINES, SYSTEMC.

--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, the /*verilator&32;hier_block*/ metacomment is ignored. See *Hierarchical Verilation*.

--hierarchical-params-file <filename>

Rarely needed - internal use. Internal flag inserted used during --hierarchical; specifies name of hierarchical parameters file for deparametrized modules with /*verilator&32;hier_block*/ metacomment. See *Hierarchical Verilation*.

--hierarchical-threads <threads>

Specifies the number of threads used for scheduling hierarchical blocks. This allows scheduling multi-thread hierarchical blocks on multiple threads, without increasing the parallelism of the entire design.

Set to --threads by default. For optimal performance should not exceed the CPU thread count.

-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. Unlike with -y, multiple directories may be specified separated with a + symbol; this is for Verilog-XL compatibility and is not recommended usage as this is not supported by some third-party tools.

--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 which will inline everything, leads 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>

Tune the 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 if --build-jobs isn't provided, and the internal compilation steps of Verilator if --verilate-jobs isn't provided. Also sets --output-groups if isn't provided.

If zero, uses the number of threads in the current hardware. Otherwise, must be a positive integer specifying the maximum number of parallel build jobs.

--no-json-edit-nums

Don't dump edit number in .tree.json files. This may make the file more run-to-run stable for easier comparison.

--no-json-ids

Don't use short identifiers instead of addresses/paths in .tree.json.

--json-only

Create JSON output only, do not create any other output.

The JSON format is intended to be used to leverage Verilator's parser and elaboration to feed to other downstream tools. For details on the format, see the Verilator Internals manual. Be aware that the JSON format is still evolving; there will be some changes in future versions.

This option disables some more aggressive transformations and dumps only the final state of the AST. For more granular and unaltered dumps, meant mainly for debugging see --dump-tree-json.

--json-only-meta-output <filename>

Specifies the filename for the metadata output file (*.tree.meta.json*) of --json-only. Using this option automatically sets --json-only.

--json-only-output <filename>

Specifies the filename for the main output file (*.tree.json*) of --json-only. Using this option automatically sets --json-only.

--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. The default is an l2-name matching the top module, and 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.

-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.)

--lib-create <name>

Produces C++, Verilog wrappers, and a Makefile which can produce a DPI library that 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.

Designs compiled using this option cannot use --timing with delays.

See also --protect-lib.

+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 relatively standard across Verilog tools. Defaults to ".v+.sv".

--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 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.

--localize-max-size <value>

Rarely needed. Set the maximum variable size in bytes for it to be subject to localizing-to-stack optimization. Defaults to 1024.

--main

Generates a top-level C++ main() file that supports parsing arguments, but does not drive any inputs. This is sufficient to use for top-level SystemVerilog designs that have no inputs.

This option can also be used once to generate the main .cpp file as a starting point for editing. Copy it outside the obj directory, manually edit, and then pass the filename on later Verilator command line invocations.

Typically used with --timing to support delay-generated clocks, and --build.

Implies --cc if no other output mode was provided.

See also --binary.

--main-top-name <string>

Specify the name passed to the Verilated model being constructed, in the generated C++ main() function.

If the string "-" is used, no top level scope is added.

--make <build-tool>

Generates a script for the specified build tool.

Supported values are gmake for GNU Make, or cmake for CMake, or json to create a JSON file to feed other build tools.

Multiple options can be specified together. If no build tool is specified, gmake is assumed. The executable of gmake can be configured via the 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 the 1024). Defaults to 64K.

--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.

--MMD

--no-MMD

Enable/disable the 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.

--mod-prefix <topname>

Specifies the name to prepend to all lower-level classes. Defaults to the same as --prefix.

--MP

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

+notimingchecks

Ignored for compatibility with other simulators.

-o <executable>

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

-O0

Disables optimization of the model.

-O3

Enables slow optimizations for the code Verilator itself generates (as opposed to -CFLAGS -O3 which affects 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 specific optimizations, with the optimization selected based on the letter passed. A lowercase letter disables an optimization, an uppercase letter enables it. This option is deprecated and the various *-f<optimization>* arguments should be used instead.

--no-order-clock-delay

Deprecated and has no effect (ignored).

In versions before 5.000:

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-groups <numfiles>

Enables concatenating the output .cpp files into the given number of effective output .cpp files. This minimizes the compiler startup overhead from compiling many small files, which can happen in designs making extensive use of SystemVerilog classes, templates or generate blocks.

Using --output-groups can adversely impact caching and stability (as in reproducibility) of compiled code. Compilation of larger .cpp files also has higher memory requirements. Too low values might result in swap thrashing with large designs, high values give no benefits.

Typically setting the number of files to the hardware thread count, corresponding to number of compiler jobs that can run in parallel, will lead to fastest build times. (e.g. for small to medium designs the value should range from 2 to 20.)

Zero disables this feature. Negative one, the default, sets the groups to the value from --build-jobs, or from -j, or zero in that priority.

--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 makefiles,

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, it 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.

--pins-bv <width>

Specifies SystemC inputs/outputs greater than or equal to <width> bits wide should use sc_bv's instead of uint32/uint64_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-inout-enables

Specifies that the __en and __out outputs will always be created for inouts in the top-level module. The __en variable has a one in a bit position to indicate the corresponding bit of the __out variable has a value being driven from within the Verilated model.

--pins-sc-biguint

Specifies SystemC inputs/outputs 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-sc-uint

Specifies SystemC inputs/outputs 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-uint-bool

Specifies SystemC inputs/outputs one bit wide should use sc_uint<1>.

--pins-uint8

Specifies SystemC inputs/outputs 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.

--pins64

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

--no-pins64

Backward compatible alias for --pins-bv 33.

--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 the 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.

--preproc-resolve

With -E, resolve referenced instance modules, to include preprocessed output of submodules. Used to convert a multi-file design into a single output file.

See -E.

--preproc-token-limit <value>

Rarely needed. Configure the limit of the number of tokens Verilator can process on a single line to prevent infinite loops and other hangs. Defaults to 40000 tokens.

--private

Rarely needed. Opposite of --public. This is the default; this option exists for backwards compatibility.

--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. (This may slow down the executable by ~5%.) Furthermore, the function name will be suffixed with the basename of the Verilog module and the 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-exec

Enable collection of execution trace, that can be converted into a gantt chart with verilator_gantt See *Execution Profiling*.

--prof-pgo

Enable collection of profiling data for profile-guided Verilation. Currently, this is only useful with --threads. See *Thread Profile-Guided Optimization*.

--prof-threads

Removed in 5.020. Was an alias for –prof-exec and –prof-pgo together.

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

--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-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.

Designs compiled using this option cannot use --timing with delays.

--public

Rarely needed. This is only for historical debugging use and 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-depth <level>

Enables public as with --public-flat-rw, but only to the specified depth of modules. It operates at the module maximum level, so if a module's cells are A.B.X and A.X, the a –public-depth 3 must be used to make module X public, and both A.B.X and A.X will be public.

--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.

--public-ignore

Ignore all /*verilator public* */ metacomments. This is useful for speed-optimizing VPI builds where VPI is not being used. This only affects metacomments; options such as --public, --public-depth, etc. work normally.

--public-params

Declares all parameters public as if they had /*verilator public_flat_rd*/ metacomments.

-pvalue+<name>=<value>

Overwrites the given parameter(s) of the top-level module. See -G for a detailed description.

--quiet

Alias for --quiet-exit --quiet-stats.

--quiet-exit

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

--quiet-stats

Disable printing the Verilation statistics report, see *Verilation Summary Report*.

--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 <value>

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 to perform this transformation. The default limit is 40. A smaller number may slightly improve C++ compilation time on designs where these sequences are common; however, the effect on model performance requires benchmarking.

--report-unoptflat

Enable extra diagnostics for UNOPTFLAT warnings. This includes, for each loop, the ten widest variables in the loop, and the ten 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 analyze the problem, but can be very large.

Various commands exist for viewing and manipulating DOT files, for example, the "dot" command can 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 https://rr-project.org.

--runtime-debug

Enable including debug assertions in the generated model. This may significantly decrease model performance. This option will only work with gcc/clang.

This option has the same effect as the following flags:

--decorations node

Instructs Verilator to add comments to the Verilated C++ code to assist determining what Verilog code was responsible for each C++ statement.

-CFLAGS -ggdb -LDFLAGS -ggdb
: Instructs the compiler and linker to enable debugger symbols.

-CFLAGS -fsanitize=address,undefined -LDFLAGS -fsanitize=address,undefined
: Instructs the compiler and linker to enable the address sanitizer, and undefined behavior sanitizer.

-CFLAGS -D_GLIBCXX_DEBUG
: Instructs the compiler to enable C++ library (glibc) internal assertions to find library-misuse issues.

-CFLAGS -DVL_DEBUG=1
: Instructs the compiler to enable Verilator's runtime assertions and debug capabilities. To enable debug print messages at runtime, see +verilator+debug.

The -CFLAGS and/or -LDFLAGS options used here pass the following argument into the generated Makefile for use as compiler or linker options respectively. If you are using your own Makefiles, adapt appropriately to pass the suggested flags to the compiler and linker.

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

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

--skip-identical

--no-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.

--stats
: Creates a dump file with statistics on the design in <prefix>__stats.txt. Also dumps DFG patterns to <prefix>__stats_dfg_patterns__*.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.

--no-std
: Prevents parsing standard input files, alias for --no-std-package, --no-std-waiver. This may be extended to prevent reading other standardized files in future versions.

--no-std-package
: Prevents parsing standard *std::* package file.

--no-std-waiver
: Prevents parsing standard lint waivers (*verilated_std_waiver.vlt*).

--no-stop-fail
: Don't call $stop when assertion fails. Simulation will continue.

--structs-packed
: Converts all unpacked structures to packed structures, and issues an UNPACKED warning. Specifying this option allows for backward compatibility with versions before Verilator 5.006, when Verilator would always pack unpacked structures.

-sv
: Specifies SystemVerilog language features should be enabled; equivalent to --language 1800-2023. This option is selected by default; it exists for compatibility with other simulators.

+systemverilogext+<ext>

A synonym for +1800-2023ext+<ext>.

--no-threads

Deprecated and has no effect (ignored).

In versions before 5.004, created a model which was not thread-safe.

--threads <threads>

With "–threads 1", the default, 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*. This option also applies to --trace-vcd (but not --trace-fst).

--threads-dpi <mode>

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 thread-safe, 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 assumes 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 thread-safe, 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 timeunit and timeprecision 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 /<timeprecision>

--timescale-override <timeunit>/<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 always to use a precision of "1fs".

--timing

--no-timing

Enables/disables support for timing constructs such as delays, event controls (unless it's at the top of a process), wait statements, and joins. When disabled, timing control constructs are ignored the same way as in earlier versions of Verilator. Enabling this feature requires a C++ compiler with coroutine support (GCC 10, Clang 5, or newer).

--top <topname>

--top-module <topname>

When the input Verilog contains more than one top-level module, it specifies the name of the 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 *Finding and Binding Modules*.

--trace

Deprecated; use --trace-fst, --trace-saif or --trace-vcd instead.

Using --trace without --trace-fst nor --trace-fst requests VCD traces.

Using --trace --trace-fst requests FST traces.

Using --trace --trace-saif requests SAIF traces.

--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, incrementing on each coverage occurrence. Due to this, this option may significantly 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 signals. Defaults to the entire model. Using a small number will decrease visibility, but significantly 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-saif

Enable SAIF tracing in the model. This overrides --trace. Specification of this format can be found in IEEE 1801-2018 (see Annex I).

--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 only applies to --trace-fst. FST tracing can utilize at most "–trace-threads 2". This overrides --no-threads.

This option is accepted, but has absolutely no effect with --trace, which respects --threads instead.

--no-trace-top

Disables tracing for the input and output signals in the top wrapper which Verilator adds to the design. The signals are still traced in the original verilog top modules.

When combined with --main-top-name set to "-" or when the name of the top module is set to "" in its constructor, the generated trace file will have the verilog top module as its root, rather than another module added by Verilator.

--trace-underscore

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

--trace-vcd

Adds waveform tracing code to the model using VCD format.

Verilator will generate additional <prefix>__Trace*.cpp files must 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 small performance losses, even when tracing is not turned on during model execution.

When using --threads, VCD tracing is parallelized, using the same number of threads as passed to --threads.

-U<var>

Undefines the given preprocessor symbol.

--no-unlimited-stack

Verilator tries to disable stack size limit using ulimit -s unlimited command. This option turns this behavior off.

--unroll-count <loops>

Rarely needed. Specifies the maximum number of loop iterations that may be unrolled. See also BLKLOOPINIT warning, and /*verilator&32;unroll_disable*/ and /*verilator&32;unroll_full*/ metacomments.

--unroll-stmts <statements>

Rarely needed. Specifies the maximum number of statements in a loop for that loop to be unrolled. See also BLKLOOPINIT warning, and /*verilator&32;unroll_disable*/ and /*verilator&32;unroll_full*/ metacomments.

--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, otherwise, they are ignored. Note "-v" is relatively standard across Verilog tools.

--valgrind

Rarely needed. Run Verilator under Valgrind. The command may be changed with VERILATOR_VALGRIND.

--no-verilate

When using --build, disable the generation of C++/SystemC code, and execute only the build. This can be useful for rebuilding the Verilated code produced by a previous invocation of Verilator.

--verilate-jobs <value>

Specify the level of parallelism for the internal compilation steps of Verilator. If zero, uses the number of threads in the current hardware. Otherwise, must be a positive integer specifying the maximum number of parallel build jobs.

If not provided, and -j is provided, the -j value is used.

See also -j.

+verilog1995ext+<ext>

Synonym for +1364-1995ext+<ext>.

+verilog2001ext+<ext>

Synonym for +1364-2001ext+<ext>.

--version

Displays program version and exits.

--vpi

Enable the use of VPI and linking against the verilated_vpi.cpp files.

--waiver-multiline

When using --waiver-output <filename>, include a match expression that includes the entire multiline error message as a match regular expression, as opposed to the default of only matching the first line of the error message. This provides a starting point for creating complex waivers, but such generated waivers will likely require editing for brevity before being reused.

--waiver-output <filename>

Generate a waiver file that 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 *Verilator Control Files*. The standard file extension is ".vlt". These files can directly be consumed by Verilator, typically by placing the filename as part of the Verilator command line options. Waiver files need to be listed on the command line before listing the files they are waiving.

-Wall

Enable all code-style warnings, including style warnings that are typically disabled by default. Equivalent to -Wwarn-lint -Wwarn-style. Excludes some specialty warnings.

-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 an 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 that process the warning messages but may get confused by lines quoted from the source.

-Wno-fatal

When warnings are detected, print them, but do not terminate Verilator.

Having warning messages in builds can be sloppy. You should 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-ASCRANGE -Wno-BSSPACE -Wno-CASEINCOMPLETE -Wno-CASEOVERLAP -Wno-CASEX -Wno-CASTCONST -Wno-CASEWITHX -Wno-CMPCONST -Wno-COLONPLUS -Wno-IMPLICIT -Wno-IMPLICITSTATIC -Wno-PINCONNECTEMPTY -Wno-PINMISSING -Wno-STATICVAR -Wno-SYNCASYNCNET -Wno-UNDRIVEN -Wno-UNSIGNED -Wno-UNUSEDGENVAR -Wno-UNUSEDPARAM -Wno-UNUSEDSIGNAL -Wno-WIDTH, plus the list shown for -Wno-style.

It is strongly recommended that you clean up your code rather than using this option; it is only intended to be used when running test-cases of code received from third parties.

-Wno-style

Disable all code style related warning messages (note that by default, they are already disabled). This is equivalent to -Wno-DECLFILENAME -Wno-DEFPARAM -Wno-EOFNEWLINE -Wno-GENUNNAMED -Wno-IMPORTSTAR -Wno-INCABSPATH -Wno-PINCONNECTEMPTY -Wno-PINNOCONNECT -Wno-SYNCASYNCNET -Wno-UNDRIVEN -Wno-UNUSEDGENVAR -Wno-UNUSEDPARAM -Wno-UNUSEDSIGNAL -Wno-VARHIDDEN.

-work <libname>

Use the specified Verilog config library name for all cells read after this argument. May be specified multiple times, it will apply to cells read between the given arguments. E.g. *-work liba a.v -work libb b.v* will use *liba* for modules inside *a.v* or in cells resolved hierarchically under those modules, and will use *libb* for modules inside *b.v* or hierarchically under.

Defaults to "work" (IEEE 1800-2023 3.3.1).

See *Finding and Binding Modules*.

-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.

This option changes ASSIGNIN from an error to a warning.

-Wwarn-<message>

Enables the specified warning message.

-Wwarn-lint

Enable all lint-related warning messages (note that by default, they are already enabled), but do not affect style messages. This is equivalent to -Wwarn-ALWCOMBORDER -Wwarn-ASCRANGE -Wwarn-BSSPACE -Wwarn-CASEINCOMPLETE -Wwarn-CASEOVERLAP -Wwarn-CASEWITHX -Wwarn-CASEX -Wwarn-CASTCONST -Wwarn-CMPCONST -Wwarn-COLONPLUS -Wwarn-IMPLICIT -Wwarn-IMPLICITSTATIC -Wwarn-LATCH -Wwarn-MISINDENT -Wwarn-NEWERSTD -Wwarn-PREPROCZERO -Wwarn-PINMISSING -Wwarn-REALCVT

-Wwarn-STATICVAR -Wwarn-UNSIGNED -Wwarn-WIDTHTRUNC -Wwarn-WIDTHEXPAND -Wwarn-WIDTHXZEXPAND.

-Wwarn-style

Enable all code style-related warning messages. This is equivalent to -Wwarn-ASSIGNDLY -Wwarn-BLKSEQ -Wwarn-DECLFILENAME -Wwarn-DEFPARAM -Wwarn-EOFNEWLINE -Wwarn-GENUNNAMED -Wwarn-IMPORTSTAR -Wwarn-INCABSPATH -Wwarn-PINCONNECTEMPTY -Wwarn-PINNOCONNECT -Wwarn-PROCASSINIT -Wwarn-SYNCASYNCNET -Wwarn-UNDRIVEN -Wwarn-UNUSEDGENVAR -Wwarn-UNUSEDLOOP -Wwarn-UNUSEDPARAM -Wwarn-UNUSEDSIGNAL -Wwarn-VARHIDDEN.

--x-assign <mode>

Controls the two-state value that is substituted when an explicit X value is encountered in the source.

With "–x-assign 0",
converts all Xs to 0s, and is also fast.

With "–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.

With "–x-assign fast", the default,
converts all Xs to whatever is best for performance.

With "–x-assign unique",
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 the 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*, use the +verilator+rand+reset+2 runtime option, and seed the runtime random number generator such that each regression run gets a different randomization sequence with +verilator+seed+<value>. 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 explicitly written as X in modules (not classes, nor parameters) 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 <mode>

Controls the two-state value used to initialize variables that are not otherwise initialized.

With "–x-initial 0",
initializes all otherwise uninitialized variables to zero.

With "–x-initial unique", the default,
initializes variables using a function, which determines the value to use for each initialization. This gives the greatest flexibility and allows for finding reset bugs. See *Unknown States*.

If using *–x-initial unique*, use the +verilator+rand+reset+2 runtime option, and seed the runtime random number generator such that each regression run gets a different randomization sequence with +verilator+seed+<value>. You'll probably also want to print any seeds selected, and code to enable rerunning with that same seed, so you can reproduce bugs.

With "–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 the 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, some designs rely on X->0 triggering a negedge, particularly in reset sequences. Using --x-initial-edge will replicate this behavior. It will also ensure that X->1 triggers a posedge.

> **Note**
>
> Using this option can affect convergence, and it 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.

> **Note**
>
> This feature is deprecated in favor of --json-only.

--xml-output <filename>

Specifies the filename for the XML output file. Using this option automatically sets --xml-only.

> **Note**
>
> This feature is deprecated in favor of --json-only.

-y <dir>

Add the directory to the list of directories that should be searched to find include files or libraries. The three flags -y, +incdir+<dir> and -I<dir> have a similar effect; +incdir+<dir> and -y are relatively 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.

12.2 Verilator Control Files

In addition to the command line, warnings and other features for the verilator command may be controlled with Verilator Control Files, not to be confused with IEEE Configurations blocks (*config...endconfig*). Typically named with the *.vlt* extension, what makes it a Verilator Control File is the `verilator_config directive. These files, when named *.vlt*, are read before source code files; if this behavior is undesired, name the control file with a *.v* or other suffix.

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.

Verilator control files are fed through the normal Verilog preprocessor prior to parsing, so "`ifdef", "`define", and comments may be used as if the control file was standard Verilog code.

Note that file or line-specific control only applies to files read after the control file. It is therefore recommended to pass the control file to Verilator as the first file.

The grammar of control commands is as follows:

`verilator_config

Take the remaining text and treat it as Verilator Control File commands. See *Verilator Control Files*.

clock_enable -module "<modulename>" -var "<signame>"

Deprecated and has no effect (ignored).

In versions before 5.000:

Indicates that the signal is used to gate a clock, and the user takes responsibility for ensuring there are no races related to it.

Same as /*verilator&32;clock_enable*/ metacomment.

clocker -module "<modulename>" [-function "<funcname>"] -var "<signame>"

clocker -module "<modulename>" [-task "<taskname>"] -var "<signame>"

no_clocker -module "<modulename>" [-function "<funcname>"] -var "<signame>"

no_clocker -module "<modulename>" [-task "<taskname>"] -var "<signame>"

Indicates whether the signal is used as clock or not. Verilator uses this information to mark the signal and any derived signals as clocker. See --clk.

Same as /*verilator&32;clocker*/ metacomment.

coverage_block_off -file "<filename>" -line <lineno>

coverage_block_off -module "<modulename>" -block "<blockname>"

Specifies the entire begin/end block should be ignored for coverage analysis purposes. It can either be specified as a named block or as a filename and line number.

Same as /*verilator&32;coverage_block_off*/ metacomment.

coverage_off [-file "<filename>" [-lines <line> [- <line>]]]

coverage_on [-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.

forceable -module "<modulename>" -var "<signame>"

Generate public *<signame>__VforceEn* and *<signame>__VforceVal* signals that can force/release a signal from C++ code. The force control signals are created as public_flat signals.

Same as /*verilator&32;forceable*/ metacomment.

full_case -file "<filename>" -lines <lineno>

Same as //synthesis full_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 an unit of hierarchical Verilation. Note that the setting is ignored unless the --hierarchical option is specified. See *Hierarchical Verilation*.

hier_params -module "<modulename>"

Specifies that the module contains parameters a --hierarchical block. This option is used internally to specify parameters for deparametrized hier block instances. This option should not be used directly. See *Hierarchical Verilation*.

hier_workers -hier-dpi "<function_name>" -workers <worker_count>

Specifies how many threads need to be used for scheduling hierarchical DPI tasks. This data is inserted internally during --hierarchical, based on value specified in *hier_workers -module*. This option should not be used directly. See *Hierarchical Verilation*.

hier_workers -module "<module_name>" -workers <worker_count>

Specifies how many threads need to be used for scheduling given module with /*verilator&32;hier_block*/ metacomment. This number needs to be smaller than --threads to fit in a thread schedule. 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.

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>"] -function "<funcname>"

no_inline [-module "<modulename>"] -task "<taskname>"

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.

isolate_assignments -module "<modulename>" -function "<fname>"

isolate_assignments -module "<modulename>" [-function "<funcname>"] -var "<signame>"

isolate_assignments -module "<modulename>" [-task "<taskname>"] -var "<signame>"

Used to indicate that the assignments to this signal in any blocks should be isolated into new blocks. Same as /*verilator&32;isolate_assignments*/ metacomment.

lint_off [-rule <message>] [-file "<filename>" [-lines <line> [- <line>]]]

lint_off [-rule <message>] [-file "<filename>"] [-contents "<wildcard>"] [-match "<wildcard>"]

lint_on [-rule <message>] [-file "<filename>" [-lines <line> [- <line>]]]

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 -contents is provided, the input files must contain the given wildcard (with '*' or '?'), and are waived in case they match, provided the -rule, -file, and -contents also match. The wildcard should be designed to match a single line; it is unspecified if the wildcard is allowed to match across multiple lines. The input contents does not include --std standard files, nor control files (with verilator_config). Typical use for this is to match a version number present in the Verilog sources, so that the waiver will only apply to that version of the sources.

If -match is provided, the linter warnings are matched against the given wildcard (with '*' or '?'), and are waived in case they match, provided the -rule, -file, and -contents also match. The wildcard is compared across the entire multi-line message; see --waiver-multiline.

Before version 4.026, -rule was named -msg, and -msg remained a deprecated alias until Version 5.000.

parallel_case -file "<filename>" -lines <lineno>

Same as //synthesis 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.

profile_data -hier-dpi "<function_name>" -cost <cost_value>

Internal profiling data inserted during --hierarchical; specifies execution cost of a hierarchical DPI wrappers for modules with /*verilator&32;hier_block*/ metacomment. See *Hierarchical Verilation*.

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*.

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>" "@(edge)"]

Sets the variable to be public. Same as /*verilator&32;public*/ or /*verilator&32;public_flat*/, etc., metacomments. See also *VPI Example*.

sc_bv -module "<modulename>" [-function "<funcname>"] -var "<signame>"

sc_bv -module "<modulename>" [-task "<taskname>"] -var "<signame>"

Sets the port to be of sc_bv<{width}> type, instead of bool, uint32_t, or uint64_t. Same as /*verilator&32; sc_bv*/ metacomment.

sformat [-module "<modulename>"] [-function "<funcname>"] -var "<signame>"

sformat [-module "<modulename>"] [-task "<taskname>"] -var "<signame>"

Must be applied to the final argument of type input string of a function or task to indicate that the function or task should pass all remaining arguments through $sformatf. This allows the 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>"] [-function "<funcname>"] -var "<varname>"

split_var [-module "<modulename>"] [-task "<taskname>"] -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.

timing_off [-file "<filename>" [-lines <line> [- <line>]]]

timing_on [-file "<filename>" [-lines <line> [- <line>]]]

Enables/disables timing constructs for the specified file and lines. When disabled, all timing control constructs in the specified source code locations are ignored the same way as with the --no-timing, and code:*fork*/join* blocks are converted into begin/end blocks.

Same as /*verilator&32;timing_on*/, /*verilator&32;timing_off*/ metacomments.

tracing_off [-file "<filename>" [-lines <line> [- <line>]]]

tracing_on [-file "<filename>" [-lines <line> [- <line>]]]

tracing_off [-scope "<scopename>" [-levels <levels>]]

tracing_on [-scope "<scopename>" [-levels <levels>]]

Enable/disable waveform tracing for all future signals declared in all files.

With -file, enable/disable waveform tracing in the specified filename (or wildcard with '*' or '?'), and -line range of line numbers (or all lines if omitted).

For tracing_off with -file, instances below any module in the files/ranges specified will also not be traced. To overcome this feature, use tracing_on on the upper module declaration and on any cells, or use the -scope flavor of the command.

With -scope enable/disable waveform tracing for the specified scope (or wildcard with '*' or '?'), and optional –levels number of levels below. These controls only operate after other file/line/module-based controls have indicated the signal should be traced.

With -levels (used with -scope), the number of levels below that scope which the rule is to match, where 0 means all levels below, 1 the exact level as the provided scope, and 2 means an additional level of children below the provided scope, etc.

12.3 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. With *–annotate-points* the coverage points corresponding to each line are also shown.

Additional Verilog-XL-style standard arguments specify the search paths necessary to find the source code on which the coverage analysis was performed.

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 the use of verilator_coverage, see *Coverage Analysis*.

12.3.1 verilator_coverage Example Usage

verilator_coverage –help verilator_coverage –version

verilator_coverage –annotate obj_dir coverage.dat

verilator_coverage –write merged.dat coverage.dat ...

verilator_coverage –write-info merged.info coverage.dat ...

12.3.2 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 to which source files with annotated coverage data should be written.

Points are children of each line coverage- branches, expressions or toggle points. When point counts are aggregated into a line, the minimum and maximum counts are used to determine the status of the line (complete, partial, failing) The count is equal to the maximum of the points.

Coverage data is annotated at the beginning of the line and is formatted as a special character followed by the number of coverage hits. The special characters " ,%,~,+,-" indicate summary of the coverage, and allow use of grep to filter the report.

- " " (whitespace) indicates that all points on the line are above the coverage min.
- "%" indicates that all points on the line are below the coverage min.
- "~" indicates that some points on the line are above the coverage min and some are below.
- "+" coverage point was at or above the min. Only used with --annotate-points.
- "-" coverage point was below the min. Only used with --annotate-points.

```
 100000  input logic a;      // Begins with whitespace, because
                            // number of hits (100000) is above the min.
+100000  point: comment=a    // Begins with +, because
                            // number of hits (100000) is above the min.
%000000  input logic b;      // Begins with %, because
                            // number of hits (0) is below the min.
-000000  point: comment=b    // Begins with -, because
                            // number of hits (0) is below the min.
~000010  if (cyc!=0) begin   // Begins with ~, because
                            // branches are below and above the min.
+000010  point: comment=if   // The if branch is above the min.
-000000  point: comment=else // The else branch is below the min.
```

--annotate-all

Specifies all files should be shown. By default, only those source files with low coverage are written to the output directory.

This option should be used together with --annotate.

--annotate-min <count>

Specifies the threshold (<count>) below which coverage point is considered sufficient. If the threshold is not exceeded, then the annotation will begin with a "%" symbol to indicate the coverage is insufficient.

The <count> threshold defaults to 10.

This option should be used together with --annotate.

--annotate-points

Specifies all coverage points should be shown after each line of text. By default, only source lines are shown.

```
 100000  input logic a, b, c;
+100000 point: comment=a  // These lines are only shown
+200000 point: comment=b  // with option --annotate-points
+300000 point: comment=c  // enabled.
```

This option should be used together with --annotate.

--filter-type <regex>

Skips records of coverage types that matches with <regex> Possible values are *toggle*, *line*, *branch*, *expr*, *user* and a wildcard with * or *?*. The default value is *.

--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 the 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 critical, 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 the 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 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. This format lacks the comments for cover points that the verilator_coverage format has. It can be used with genhtml to generate an HTML report. genhtml --branch-coverage will also display the branch coverage, analogous to --annotate-points.

12.4 verilator_gantt

Verilator_gantt creates a visual representation to help analyze Verilator multithreaded simulation performance by showing when each macro-task starts, ends, and when each thread is busy or idle.

For an overview of the use of verilator_gantt, see *Code Profiling*.

12.4.1 Gantt Chart VCD

Verilated_gantt creates a value change dump (VCD) format dump file which may be viewed in a waveform viewer (e.g., C<GTKWave>):

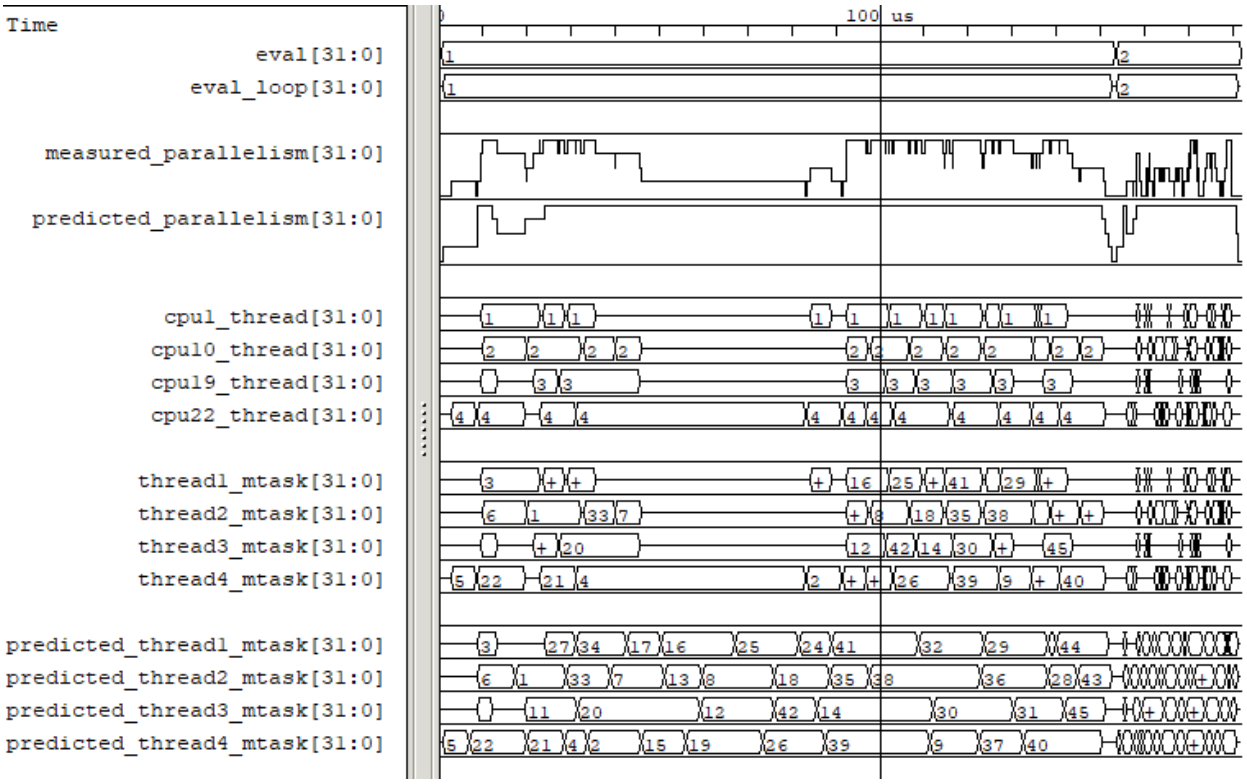


Fig. 12.1: Example verilator_gantt output, as viewed with GTKWave.

The viewed waveform chart has time on the X-axis, with one unit for each time tick of the system's high-performance counter.

12.4.2 Gantt Chart VCD Signals

In waveforms, there are the following signals. In GTKWave, use "decimal" data format to remove the leading zeros and make the traces easier to read.

trace/section
: Shows the name of the current top of the execution section stack. Set GTKWave data format to "ASCII".

trace/depth
: Shows the depth of the execution section stack. Set GTKWave data format to "Analog".

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.

cpu#_waiting
For the given CPU number, aggregated waiting time for mtask dependencies. Visualized as *X* values.

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.

12.4.3 verilator_gantt Example Usage

verilator_gantt –help verilator_gantt –version

verilator_gantt profile_exec.dat

12.4.4 verilator_gantt Arguments

<filename>
The filename to read data from; the default is "profile_exec.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; the default is "verilator_gantt.vcd".

12.5 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 the time, etc., in each Verilog block.

Due to rounding errors in gprof reports, the input report's percentages may not total 100%. In the verilator_profcfunc report this will get reported as a rounding error.

For an overview of the use of verilator_profcfunc, see *Code Profiling*.

12.5.1 verilator_profcfunc Example Usage

verilator_profcfunc –help verilator_profcfunc –version

verilator_profcfunc gprof.out

12.5.2 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.

12.6 Simulation Runtime Arguments

The following are the arguments that may be passed to a Verilated executable, provided that executable calls VerilatedContext*->commandArgs(argc, argv).

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+coverage+file+<filename>   Set coverage output filename
+verilator+debug                      Enable debugging
+verilator+debugi+<value>             Enable debugging at a level
+verilator+error+limit+<value>        Set error limit
+verilator+help                       Show help
+verilator+noassert                   Disable assert checking
+verilator+prof+exec+file+<filename>  Set execution profile filename
+verilator+prof+exec+start+<value>    Set execution profile starting point
+verilator+prof+exec+window+<value>   Set execution profile duration
+verilator+prof+vlt+file+<filename>   Set PGO profile filename
+verilator+quiet                      Minimize additional printing
+verilator+rand+reset+<value>         Set random reset technique
+verilator+seed+<value>               Set random seed
+verilator+V                          Show verbose version and config
+verilator+version                    Show version and exit
```

+verilator+coverage+file+<filename>

When a model was Verilated using --coverage, sets the filename to write coverage data into. Defaults to coverage.dat.

+verilator+debug

Enable simulation runtime debugging. Equivalent to +verilator+debugi+4.

To be useful, the model typically must first be compiled with debug capabilities by Verilating with --runtime-debug or *-CFLAGS -DVL_DEBUG=1*.

+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 *$stop* calls needed before exit. Does not affect *$fatal*. Defaults to 1.

+verilator+help

Display help and exit.

+verilator+noassert

Disable assert checking per runtime argument. This is the same as calling VerilatedContext*->assertOn(false) in the model.

+verilator+prof+exec+file+<filename>

When a model was Verilated using --prof-exec, sets the simulation runtime filename to dump to. Defaults to profile_exec.dat.

+verilator+prof+exec+start+<value>

When a model was Verilated using --prof-exec, 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+exec+window+<value>

When a model was Verilated using --prof-exec, after $time reaches +verilator+prof+exec+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+threads+file+<filename>

Removed in 5.020. Was an alias for +verilator+prof+exec+file+<filename>

+verilator+prof+threads+start+<value>

Removed in 5.020. Was an alias for +verilator+prof+exec+start+<value>

+verilator+prof+threads+window+<value>

Removed in 5.020. Was an alias for +verilator+prof+exec+window+<value>

+verilator+prof+vlt+file+<filename>

When a model was Verilated using --prof-pgo, sets the profile-guided optimization data runtime filename to dump to. Defaults to profile.vlt.

+verilator+quiet

Disable printing the simulation summary report, see *Simulation Summary Report*.

+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+V

Shows the verbose version, including configuration information.

+verilator+version

Displays program version and exits.

CHAPTER

THIRTEEN

ERRORS AND WARNINGS

13.1 Disabling Warnings

Warnings may be disabled in multiple ways:

1. Disable the warning globally by invoking Verilator with the -Wno-{warning-code} option.

   Global disables should be avoided, as they removes all checking across the source files, and prevents other users from compiling the sources without knowing the magic set of disables needed to compile those sources successfully.

2. Disable the warning in the design source code. When the warning is printed, it will include a warning code. 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
   ```

   A lint_off in the design source code will propagate down to any child files (files later included by the file with the lint_off), but will not propagate upwards to any parent file (file that included the file with the lint_off).

3. Disable the warning using *Verilator Control Files* with a lint_off command. This is useful when a script suppresses 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" -lines 1
   ```

13.2 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 warnings start with a percent sign (historical heritage from Digital Equipment Corporation). Some errors or warnings 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.

Instead of parsing this text diagnostic output, tools that need to understand Verilator's warning output should read the SARIF JSON output created with --diagnostics-sarif.

13.3 List Of Warnings

Internal Error

This error should never occur first, though it 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 uses a Verilog language construct that is not yet supported in Verilator. See also *Language Limitations*.

ALWCOMBORDER

Warns that an always_comb block has a variable that 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.

ALWNEVER

Warning that an *always* @* statement has no variables being read, therefore the event list is empty, and as there are no events to wake the process up, the always will never execute.

Faulty example:

```
always @* a = 100;
```

Results in:

```
%Warning-ALWNEVER: example.v:1:3 'always @*' will never execute as expression list is empty
↪(no variables read)
```

To repair, assuming the intent was to execute the statements at e.g. time zero, instead use an *always_comb* statement.

Ignoring this warning will only suppress the lint check; it will simulate correctly.

ASCRANGE

Warns that a packed vector is declared with ascending bit range (i.e. [0:7]). Descending bit range is now the overwhelming standard, and ascending ranges are now thus often due to simple oversight instead of intent (a notable exception is the OpenPOWER code base).

It also warns that an instance is declared with ascending range (i.e. [0:7] or [7]) and is connected to an N-wide signal. The bits will likely be in the reversed order 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.

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, this was a common style at one point, so disabled by default as a code-style warning.

This warning is issued only if Verilator is run with --no-timing.

ASSIGNIN

An 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

An error that a pragma is badly formed, for pragmas defined by IEEE 1800-2023. For example, an empty pragma line, or an incorrectly used 'pragma protect'. Third-party pragmas not defined by IEEE 1800-2023 are ignored.

This error may be disabled with a lint_off BADSTDPRAGMA metacomment.

Ignoring this warning will cause the pragma to be ignored.

BADVLTPRAGMA

An error that a */*verilator ... */* metacomment pragma is badly formed or not understood.

Faulty example:

```
  // verilator lintt_off WIDTH  //<--- Warning (lint_off misspelled)
```

Results in:

```
%Error-BADVLTPRAGMA: example.v:1:4 Unknown verilator comment: '/*verilator lintt_off␣
↪WIDTH <--- Warning (lint_off misspelled)*/'
   7 |    /*verilator lintt_off WIDTH <--- Warning (lint_off misspelled)*/
     |    ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

This error may be disabled with a lint_off BADVLTPRAGMA metacomment.

Ignoring this warning will cause the pragma to be ignored.

BLKANDNBLK

BLKANDNBLK is an error that a variable is driven by 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.

From Verilator 5.038, this warning is only issued when Verilator can't prove that the assignments are to non-overlapping sub-parts, and the blocking assignment is in combinational logic (which is the case where simulation results might differ from other simulators). Review any BLKANDNBLK cases carefully after this version, and sign them off as described above, only if know for sure the updates are not to overlapping parts of the signal.

Generally, this is caused by a register driven by both combo logic and a flop:

```
logic [3:0] foo;
always @(posedge clk) foo[index] <= ...  // With index != 0
always_comb foo[0] = ...
```

Instead, use a different register for the flop:

```
logic [3:0] foo;
logic [3:1] foo_flopped;
always @(posedge clk) foo_flopped[index] <= ... // With index != 0
always_comb foo[0] = ...
always_comb foo[3:1] = foo_flopped;
```

Or, this may also avoid the error:

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

BLKLOOPINIT

Indicates certain constructs where non-blocking assignments to unpacked arrays (memories) are not supported inside loops. These typically appear in initialization/reset code:

```
always @(posedge clk)
  if (~reset_l)
    for (i=0; i<`ARRAY_SIZE; i++)
      array[i] <= 0;  // Non-blocking assignment inside loop
  else
    array[address] <= data;
```

While this is supported in typical synthesizeable code (including the example above), some complicated cases are not supported. Namely:

1. If the above loop is inside a suspendable process or fork statement.

2. If the variable is also the target of a '<=' non-blocking assignment in a suspendable process or fork statement (in addition to a synthesizable loop).

3. If the element type of the array is a compound type.

4. In versions before 5.026, any delayed assignment to an array.

It might slightly improve run-time performance if you change the non-blocking assignment inside the loop into a blocking assignment (that is: use '=' instead of '<='), if possible.

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 no case item is provided. 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 the simulation.

Unique case statements that select on an enumerated variable, where all of the enumerated values are covered by case items, are considered complete even if the case statement does not cover illegal non-enumerated values (IEEE 1800-2023 12.5.3). Verilator checks that illegal values are not hit, unless --no-assert-case was used.

Ignoring this warning will only suppress the lint check; it will simulate correctly.

CASEOVERLAP

Warns that a case statement has case values 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 not to overlap.

Ignoring this warning will only suppress the lint check; it will simulate correctly.

CASEWITHX

Warns that a case statement contains a constant with an x . Verilator is two-state so interpret such items as always false. Note that a frequent 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 better style to use casez, and "?" in place of "x"'s. See http://www.sunburst-design.com/papers/CummingsSNUG1999Boston_FullParallelCase.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, and 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

Historical, never issued since version 5.008.

Warned with a no longer supported clock domain crossing option that asynchronous flop reset terms came from other than primary inputs or flopped outputs, creating the potential for reset glitches.

CLKDATA

Historical, never issued since version 5.000.

Warned that clock signal was mixed used with/as a data signal. The checking for this warning was enabled only if the user has explicitly marked some signal as clocker using the command line option or in-source meta comment (see --clk).

The warning could be disabled without affecting the simulation result. But it was recommended to check the warning as it may have degraded 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 an explicitly positive range, 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.pdf

Ignoring this warning may make Verilator simulations differ from other simulators.

CONSTRAINTIGN

Warns that Verilator does not support certain forms of constraint, constraint_mode, or rand_mode, and the construct was ignored.

Ignoring this warning may make Verilator randomize() simulations differ from other simulators.

CONTASSREG

An 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.

COVERIGN

Warns that Verilator does not support certain forms of covergroup, coverpoint, and coverage options, and the construct was ignored.

Disabling the UNSUPPORTED error also disables this warning.

Ignoring this warning may make Verilator ignore lint checking on the construct, and collect coverage data differently from other simulators.

DECLFILENAME

Warns that a module or other declaration's name doesn't match the filename with the path and extension stripped that it is declared in. The filename a module/interface/program 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.

DEFOVERRIDE

Warns that a macro definition within the code is being overridden by a command line directive:

For example, running Verilator with <+define+<DUP>=<def2>> and

```
1  `define DUP def2 //<--- Warning
```

Results in:

```
%Warning-DEFOVERRIDE: example.v1:20: Overriding define: 'DEF' with value: 'def2' to existing␣
↪command line define value: 'def1'
                ... Location of previous definition, with value: '50'
```

While not explicitly stated in the IEEE 1800-2023 standard, this warning tracks with the other simulators' behavior of overriding macro definitions within code files with the definition passed in through the command line.

DEFPARAM

Warns that the defparam statement was deprecated in IEEE 1364-2001, and all designs should now be using the #(...) format to specify parameters.

Defparams may be defined far from the instantiation 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 1800-2023 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: Verible'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 typically suggested by the warning message.

Ignoring this warning will only suppress the lint check; it will simulate correctly.

DETECTARRAY

Historical, never issued since version 3.862.

Was an error when Verilator tried to deal with a combinatorial loop that could not be flattened, and which involves a datatype that Verilator could not handle, such as an unpacked struct or a large unpacked array.

DIDNOTCONVERGE

Error at simulation runtime when model did not correctly settle.

Verilator sometimes has to evaluate combinatorial logic multiple times, usually around code where an UNOPTFLAT warning was issued but disabled.

Faulty example:

```
always_comb b = ~a;
always_comb a = b;
```

Results in at runtime (not when Verilated):

```
%Error: t/t_lint_didnotconverge_bad.v:7: Settle region did not converge.
```

This is because the signals keep toggling even without time passing. Thus to prevent an infinite loop, the Verilated executable gives the DIDNOTCONVERGE error.

To debug this, first, review any 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#,#}          'stl' region trigger index 1 is active: @([hybrid] b)
%Error: t/t_lint_didnotconverge_bad.v:7: Settle region did not converge.
```

The CHANGE line means that the signal 'a' kept changing on the given filename and line number that drove the signal. Inspect the code that modifies these signals. Note that 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 if Verilator is run with --no-timing. In this mode, Verilator ignores the delays and gives an ASSIGNDLY or STMTDLY warning. If these were suppressed, due to the absence of the delay, the design might 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 local or protected, but is being accessed from outside that class (if local) or a derived class (if protected).

Ignoring this warning will only suppress the lint check; it will simulate correctly.

ENDLABEL

An error that a label attached to a "end"-something statement does not match the label attached to the block start.

IEEE requires this error. Ignoring this warning will only suppress the lint check; it will simulate correctly.

Faulty example:

```
module mine;
endmodule : not_mine  //<--- Warning
```

Results in:

```
%Error-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 it entirely.

```
module mine;
endmodule : mine  //<--- Repaired
```

Other tools with similar warnings: Verible's mismatched-labels, "Begin/end block labels must match." or "Matching begin label is missing."

ENUMITEMWIDTH

An error that an enum item value is being assigned from a value which would be truncated (similar to WIDTHTRUNC), or from a sized literal constant with a different bit width (similar to WIDTHTRUNC or WIDTHEXPAND). IEEE requires this error, but it may be disabled.

Faulty example:

```
typedef enum [3:0] {
   WRONG_WIDTH = 33'h3  //<--- Warning
} enum_t;
```

To repair, correct the size of the item's value directly, or use a cast, so the resulting width matches the enum's width.

ENUMVALUE

An error that an enum data type value is being assigned from another data type that is not implicitly assignment compatible with that enumerated type. IEEE requires this error, but it may be disabled.

Faulty example:

```
typedef enum { ZERO } e_t;
initial e_t en = 0;  //<--- Warning
```

The ideal repair is to use the enumeration value's mnemonic:

```
typedef enum { ZERO } e_t;
initial e_t en = ZERO;  //<--- Repaired
```

Alternatively use a static cast:

```
typedef enum { ZERO } e_t;
initial e_t en = e_t'(0);  //<--- Repaired
```

EOFNEWLINE

Warns that a file does not end in a newline. POSIX defines that a line must end in a 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

Historical, never issued since version 5.000.

Indicated that the specified signal was generated inside the model and used as a clock.

GENUNNAMED

Warns that a generate block was unnamed and "genblk" will be used per IEEE.

The potential issue is that adding additional generate blocks will renumber the assigned names, which may cause eventual problems with synthesis constraints or other tools that depend on hierarchical paths remaining consistent.

Blocks that are empty may not be reported with this warning, as no scopes are created for empty blocks, so there is no harm in having them unnamed.

Disabled by default as this is a code-style warning; it will simulate correctly.

```
generate
  if (PARAM == 1) begin  //<--- Warning
  end
```

Results in:

```
%Warning-GENUNNAMED: example.v:2:9: Unnamed generate block (IEEE 1800-2023 27.6)
```

To fix this assign a label (often with the naming convention prefix of gen_ or g_), for example:

```
generate
  if (PARAM == 1) begin : gen_param_1  //<--- Repaired
  end
```

Other tools with similar warnings: Verible's generate-label, "All generate block statements must have a label."

HIERBLOCK

Warns that the top module is marked as a hierarchy block by the /*verilator&32;hier_block*/ metacomment, 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 using a SystemVerilog unique if or priority if statement.

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. IEEE requires this warning.

```
function int function_being_called_as_task;
  return 1;
endfunction

initial function_being_called_as_task();  //<--- Warning
```

Results in:

```
%Warning-IGNOREDRETURN: example.v:5:9: Ignoring return value of non-void function (IEEE
↪1800-2023 13.4.1)
```

The portable way to suppress this warning (in SystemVerilog) is to use a void cast, for example:

```
function int function_being_called_as_task;
  return 1;
endfunction
```

(continues on next page)

(continued from previous page)

```
4
5   initial void'(function_being_called_as_task());  //<--- Repaired
```

Ignoring this warning will only suppress the lint check; it will simulate correctly.

IMPERFECTSCH

Historical, never issued since version 5.000.

Warned that the scheduling of the model is not perfect, and some manual code edits may result in faster performance. This warning defaulted to off, was not part of -Wall, and had to be turned on explicitly before the top module statement was 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 '...'".

IMPLICITSTATIC

Warns that the lifetime of a task or a function was not provided and so was implicitly set to static. The warning is suppressed when no variables inside the task or a function are assigned to.

This is a warning because the static default differs from C++, differs from class member function/tasks. Static is a more dangerous default then automatic as static prevents the function from being reentrant, which may be a source of bugs, and/or performance issues.

If the function is in a module, and does not require static behavior, change it to "function automatic".

If the function is in a module, and requires static behavior, change it to "function static".

If the function is in a package, it defaults to static, and label the function's variables as static.

Ignoring this warning will only suppress the lint check; it will simulate correctly.

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*/ metacomment, but it references variables that are not local to the task, and 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) 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 Verilator is run with --no-timing and the loop body contains statements that would make time pass otherwise.

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 selected options may defeat 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.

LIFETIME

Error when a variable is referenced in a process that can outlive the process in which it was declared. This can happen when using 'fork..join_none' or 'fork..join_any' blocks, which spawn process that can outlive their parents. This error occurs only when Verilator can't replace the reference with a reference to copy of this variable, local to the forked process. For example:

```
1  task foo(int local_var);
2    fork
3      #10 local_var++;
4      #20 $display("local_var = %d", local_var);
5    join_none
6  endtask
```

In the example above 'local_var' exists only within scope of 'foo', once foo finishes, the stack frame containing 'i' gets removed. However, the process forked from foo continues, as it contains a delay. After 10 units of time pass, this process attempts to modify 'local_var'. However, this variable no longer exits. It can't be made local to the forked process upon spawning, because it's modified and can be referenced somewhere else, for example in the other forked process, that was delayed by 20 units of time in this example. Thus, there's no viable stack allocation for it.

In order to fix it, if the intent is not to share the variable's state outside of the process, then create a local copy of the variable.

For example:

```
1  task foo(int local_var);
2    fork
3      #10 begin
```

(continues on next page)

(continued from previous page)

```
        int forked_var = local_var;
        forked_var++;
      end
      #20 begin
        // Note that we are going to print the original value here,
        // as `forked_var` is a local copy that was initialized while
        // `foo` was still alive.
        int forked_var = local_var;
        $display("forked_var = %d", forked_var)
      end
    join_none
  endtask
```

If you need to share its state, another strategy is to ensure it's allocated statically:

```
int static_var;

task foo();
  fork
    #10 static_var++;
    #20 $display("static_var = %d", static_var);
  join_none
endtask
```

However, if you need to be able to instantiate at runtime, the solution would be to wrap it in an object, since the forked process can hold a reference to that object and ensure that the variable stays alive this way:

```
class Wrapper;
  int m_var;

  // Here we implicitly hold a reference to `this`
  task foo();
    fork
      #10 m_var++;
      #20 $display("this.m_var = %d", m_var);
    join_none
  endtask
endclass

// Here we explicitly hold a handle to an object
task bar(Wrapper wrapper);
  fork
    #10 wrapper.m_var++;
    #20 $display("wrapper.m_var = %d", wrapper.m_var);
  join_none
endtask
```

LITENDIAN

The naming of this warning is in contradiction with the common interpretation of little endian. It was therefore renamed to ASCRANGE. While LITENDIAN remains for backwards compatibility, new projects should use ASCRANGE.

MINTYPMAX

```
#(3:5:8) clk = ~clk;
```

Warns that minimum, typical, and maximum delay expressions are currently unsupported. Verilator uses only the typical delay value.

MISINDENT

Warns that the indentation of a statement is misleading, suggesting the statement is part of a previous if or while block while it is not.

Verilator suppresses this check when there is an inconsistent mix of spaces and tabs, as it cannot ensure the width of tabs. Verilator also ignores blocks with begin/end, as the end visually indicates the earlier statement's end.

Ignoring this warning will only suppress the lint check; it will simulate correctly.

For example

```
1  if (something)
2    statement_in_if;
3    statement_not_in_if;  //<--- Warning
```

Results in:

```
%Warning-MISINDENT: example.v:3:9: Misleading indentation
```

To fix this repair the indentation to match the correct earlier statement, for example:

```
1  if (something)
2    statement_in_if;
3  statement_not_in_if;  //<--- Repaired
```

Other tools with similar warnings: GCC -Wmisleading-indentation, clang-tidy readability-misleading-indentation.

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.

MODMISSING

Error that a module, typically referenced by a cell, was not found. This is typically fatal, but may be suppressed in some linting situations with missing libraries.

Ignoring this error will cause the cell definition to be discarded. Simulation results will likely be wrong, so typically used only with lint-only.

MULTIDRIVEN

Warns that the specified signal comes from multiple always blocks, each with different clocking. This warning does not look at individual bits (see the 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:

```
always @(posedge clk) begin
   out2[7:0] <= d0;  // <--- Warning
end
always @(negedge clk) begin
   out2[15:8] <= d0;  // <--- Warning
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 multiple top-level modules are 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 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.

Ignoring this warning will make multiple tops, as described in (3) above.

NEEDTIMINGOPT

Error when a timing-related construct, such as an event control or delay, has been encountered, without specifying how Verilator should handle it (neither --timing nor --no-timing option was provided).

NEWERSTD

Warns that a feature requires a newer standard of Verilog or SystemVerilog than the one specified by the --language option. For example, unsized unbased literals (*'0*, *'1*, *'z*, *'x*) require IEEE 1800-2005 or later.

To avoid this warning, use a Verilog or SystemVerilog standard that supports the feature. Alternatively, modify your code to use a different syntax that is supported by the Verilog/SystemVerilog standard specified by the --language option.

Ignoring this warning will only suppress the lint check; it will simulate correctly.

NOEFFECT

Warns that the statement will have no effect and is roughly equivalent to not being present. This is only issued when it is "non-obvious", e.g. a `if (0)` will not result in this warning.

Faulty example:

```
foreach (array[]) begin ... end  //<--- Warning
```

For a fix, remove the statement.

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.

NONSTD

Warns when a non-standard language feature is used that has a standard equivalent, which might behave differently in corner cases. For example $psprintf system function is replaced by its standard equivalent $sformatf.

Ignoring this warning will only suppress the lint check; it will simulate correctly.

NOTIMING

Error when a timing-related construct that requires --timing has been encountered. Issued only if Verilator is run with the --no-timing option.

NULLPORT

Warns that a null port was detected in the module definition port list. Null ports are empty placeholders, i.e., either one or 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 commonly 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.

PARAMNODEFAULT

An error that a parameter is being declared that has no default value, and this is being done in a non-ANSI block while this is only legal in ANSI-style *#(...)* declarations. IEEE 1800-2023 6.20.1 requires this error, but some simulators accept this syntax.

Faulty example:

```
1 module sub;
2  parameter NODEF;  //<--- Warning
```

Results in:

```
%Error-PARAMNODEFAULT: example.v:1:13 Parameter without default requires ANSI-style␣
↪parameter list (IEEE 1800-2023 6.20.1): 'NODEF'
   8 |   parameter NODEF;
     |             ^~~~~
```

To fix the issue, move to an ANSI-style declaration.

Suppressing this error will only suppress the IEEE-required check; it will simulate correctly.

PINCONNECTEMPTY

Warns that an instance has a pin that 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 the 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 that is not mentioned in an instance. If a pin is not missing it should still be specified on the instance declaration with an empty connection using (.pin_name()).

Ignoring this warning will only suppress the lint check; it will simulate correctly.

Faulty example:

```
module t;
  sub sub();  // <--- Warning
endmodule
module sub
  (output port);
endmodule
```

Results in:

```
%Warning-PINMISSING: example.v:1:8 Instance has missing pin: 'port'
```

Repaired example:

```
  sub sub (
          .port());
```

Other tools with similar warnings: Icarus Verilog's portbind, "warning: Instantiating module ... with dangling input port (...)". Slang's unconnected-port, "port '...' has no connection".

PINNOCONNECT

Warns that an instance has a pin that 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:

```
module a;
  localparam A=1;
  generate
    if (A==0) begin
      b b_inst1 (.x(1'b0));  //<--- error nonexistent port
      b #(.PX(1'b0)) b_inst2 ();  //<--- error nonexistent parameter
    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 following line, b is instantiated with a nonexistent PX parameter. 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.

PKGNODECL

Never issued since version 5.038. Historically an error that a package/class appears to have been referenced that has not yet been declared. According to IEEE 1800-2023 26.3, all packages must be declared before being used. However, several standard libraries including UVM violate this, and other tools do not warn.

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.

PREPROCZERO

Warns that a preprocessor `ifdef/`ifndef expression (added in IEEE 1800-2023) evaluates a define value which has a value of 0. This will evaluate in the expression as 1 because the define has a definition, unlike in the C preprocessor, which evaluates using the define's value (of 1).

Referring to a define with an empty value does not give this warning, as in C, the preprocessor will give an error on a preprocessor expression of a define that is empty.

```
`define ZERO 0
`ifdef (ZERO || ZERO)  //<--- warning PREPROCZERO
  `error This_will_error_which_might_be_not_the_intent
`endif
```

The portable way to suppress this warning is to use a define value other than zero, when it is to be used in a preprocessor expression.

Ignoring this warning will only suppress the lint check; it will simulate correctly.

PROCASSINIT

Warns that the specified signal is given an initial value where it is declared, and is also driven in an always process. Typically such initial values should instead be set using a reset signal inside the process, to match requirements of ASIC synthesis tools. However, declaration initializers are a valid FPGA design idiom and therefore, FPGA users may want to disable this warning.

Faulty example:

```
  logic flop_out = 1;  // <--- Warning

  always @(posedge clk, negedge reset_l) begin
    if (enable) begin
```

(continues on next page)

(continued from previous page)

```
      flop_out <= ~in;  // <--- Use of initialized
    end
  end
```

Results in:

```
%Warning-PROCASSINIT: example.v:1:21 Procedural assignment to declaration with initial value:
↪'flop_out'
                                                : ... note: In instance 't'
                                                : ... Location of variable initialization
  26 |    logic flop_out = 1;
     |                   ^
                  example.v:1:10 ... Location of variable process write
                                                : ... Perhaps should initialize instead using a reset in this␣
↪process
  30 |       flop_out <= ~in;
     |       ^~~~~~~~
```

One possible fix, adding a reset to the always:

```
  logic flop2_out;

  always @(posedge clk, negedge reset_l) begin
    if (!reset_l) begin
      flop2_out <= '1;  // <--- Added reset init
    end
    else if (enable) begin
      flop2_out <= ~in;
    end
  end
```

Alternatively, use an initial block for the initialization:

```
initial flop_out = 1;  // <--- Fixed
```

Disabled by default as this is a code-style warning; it will simulate correctly.

PROCASSWIRE

An 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, even though 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 a 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.

Ignoring the warning may make Verilator differ from a simulator that accepts the protected code.

PROTOTYPEMIS

Error that a function prototype does not match in some respects the out-of-block declaration of that function. IEEE requires this error.

The typical solution is to fix the prototype to match the declaration exactly, including in number of arguments, name of arguments, argument data types, and return data type (for functions).

Disabling this error will cause Verilator to ignore the prototype and may make the code illegal in other tools.

RANDC

Historical, never issued since version 5.018, when randc became fully supported.

Warned that the randc keyword was unsupported and was converted to rand.

REALCVT

Warns that a real number is being implicitly rounded to an integer, with possible loss of precision.

Ignoring this warning will only suppress the lint check; it will simulate correctly.

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.

Ignoring this warning will only suppress the lint check; it will simulate correctly.

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 infeasible, add an undef to indicate that the code 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".

RISEFALLDLY

```
and #(1,2,3) AND (out, a, b);
```

Warns that rising, falling, and turn-off delays are currently unsupported. The first (rising) delay is used for all cases.

SELRANGE

Warns that a selection index will go out of bounds.

Faulty example:

```
1  wire vec[6:0];
2  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:

```
1  wire vec[6:0];
2  initial begin
3    index = 7;
4    ...
5    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 the modeled values, or DPI calls.

SIDEEFFECT

Warns that an expression has a side effect that might not properly be executed by Verilator.

This often represents a bug in Verilator, as opposed to a bad code construct, however the Verilog code can typically be changed to avoid the warning.

Faulty example:

```
x = y[a++];
```

This example warns because Verilator does not currently handle side effects inside array subscripts; the a++ may be executed multiple times.

Rewrite the code to avoid expression side effects, typically by using a temporary:

```
temp = a++;
x = y[temp];
```

Ignoring this warning may make Verilator simulations differ from other simulators.

SPECIFYIGN

Warns that Verilator does not support certain constructs in specify blocks, nor $sdf_annotate, and the construct was ignored.

Disabling the UNSUPPORTED error also disables this warning.

Ignoring this warning may make Verilator ignore lint checking on the construct, and get different results from other simulators.

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 a 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).

STATICVAR

Warns that a static variable declared in a loop with declaration assignment was converted to automatic. Often such variables were intended to instead be declared "automatic".

Ignoring this warning may make Verilator differ from other simulators, which will treat the variable as static. Verilator may in future versions also treat the variable as static.

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:1:7 Ignoring delay on this statement due to --no-timing
```

This warning is issued only if Verilator is run with --no-timing. All delays on statements are ignored in this mode. 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 later, it might be an important difference.

Some possible workarounds:

- Move the delayed statement into the C++ wrapper file, where the stimulus and clock generation can be done in C++.
- Convert the statement into an FSM, or other statement that tests against $time.
- Run Verilator with --timing.

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 Verilator will rename the symbol to avoid conflict. If you are using *–vpi* and only mark things as public for VPI access (and not C++ access) then it is advisable to disable this warning with -Wno-SYMRSVDWORD.

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 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 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:

```
1  task foo(inout sig); ... endtask
2  // ...
3  always @* begin
4      foo(bus_we_select_from[2]);  // Will get TASKNSVAR error
5  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 signal, 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 1800-2023 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 relatively 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

Historical, never issued since version 5.000.

Warned that due to some construct, optimization of the specified signal or block was disabled.

Ignoring this warning only slowed simulations; it simulated 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.

Often UNOPTFLAT is caused by logic that isn't truly circular as viewed by synthesis, which analyzes interconnection per bit, but is circular to the IEEE event model which analyzes per-signal.

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 three times before it becomes stable. This is because a change in "x" requires "x" itself to change its value, which causes the warning.

For significantly better performance, split this into two 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 drive "xin" instead.

With this change, this assignment needs to be evaluated only once. These sorts 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 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 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.

Before version 5.000, the UNOPTFLAT warning may also have been due to clock enables, identified from the reported path going through a clock gating instance. To fix these, the clock_enable meta comment was used.

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 could not 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 because --structs-packed was used, or by up through version 5.004.

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 in what would typically 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 an 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

An error that a construct might be legal according to IEEE but is not currently supported by Verilator.

A typical workaround is to rewrite the construct into a more common alternative language construct.

Alternatively, check if other tools support the construct, 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

Disabling/enabling UNUSED is equivalent to disabling/enabling the UNUSEDGENVAR, UNUSEDPARAM, and UNUSEDSIGNAL warnings.

Never issued since version 5.000. Historically warned that a variable, parameter, or signal was unused.

UNUSEDGENVAR

Warns that the specified genvar is never used/consumed. See similar UNUSEDSIGNAL.

UNUSEDPARAM

Warns that the specified parameter is never used/consumed. See similar UNUSEDSIGNAL.

UNUSEDSIGNAL

Warns that the specified signal is never used/consumed. Verilator is relatively 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 && &{1'b0,
                  sig_not_used_a,
                  sig_not_used_yet_b,  // To be fixed
                  1'b0};
```

The AND with constant zero 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 the --unused-regexp option) is recognized by Verilator and suppresses warnings; if using other lint tools, either teach 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 1800-2023 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 1800-2023 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 or warning, and IEEE 1800-2023 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 1800-2023 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 this, rename the inner or outer variable to an unique name.

WAITCONST

Warns that a *wait* statement awaits a constant condition, which means it either blocks forever or never blocks.

As a special case *wait(0)* with the literal constant *0* (as opposed to something that elaborates to zero), does not warn, as it is presumed the code is making the intent clear.

Faulty example:

```
wait(1);  // Blocks forever
```

WIDTH

Warns that based on the 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:

- Resize 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  logic [31:0] array[5];
2  bit [1:0] rd_addr;
3  wire [31:0] rd_value = array[rd_addr];  //<--- Warning
```

Results in a WIDTHEXPAND warning:

```
%Warning-WIDTHEXPAND: example.v:3:31 Bit extraction of array[4:0] requires 3 bit index, not 2␣
↪bits.
```

One possible fix:

```
wire [31:0] rd_value = array[{1'b0, rd_addr}];  //<--- Fixed
```

WIDTHCONCAT

Warns that based on the 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), add the width to the parameter definition (parameter [31:0]), or add the width to the parameter usage ({PAR[31:0], PAR[31:0]}).

WIDTHEXPAND

A more granular WIDTH warning, for when a value is zero expanded. See WIDTH.

WIDTHTRUNC

A more granular WIDTH warning, for when a value is truncated. See WIDTH.

WIDTHXZEXPAND

A more granular WIDTH warning, for when a value is X/Z expanded. See WIDTH.

ZERODLY

Warns that *#0* delays do not schedule the process to be resumed in the Inactive region. Such processes do get resumed in the same time slot somewhere in the Active region. Issued only if Verilator is run with the --timing option.

ZEROREPL

Warns that zero is used as the replication value in the replication operator. This is specified as an error by IEEE 1800-2023 11.4.12.1.

Faulty example:

```
1  module dut
2    #(parameter int MY_PARAM = 0);
3    reg [7:0] data;
4    always @* begin
5      data = {MY_PARAM{1'b1}};  //<--- Warning
6    end
7  endmodule
```

Results in the following error:

```
%Error-ZEROREPL: test.v:5:22: Replication value of 0 is only legal under a concatenation (IEEE
↪1800-2023 11.4.12.1)
```

Note that in some cases, this warning may be false, when a condition upstream or downstream of the access means the zero replication will never execute or be used.

Repaired example:

```
1  module dut
2    #(parameter int MY_PARAM = 1);  //<--- REPAIRED
3    reg [7:0] data;
4    always @* begin
5      data = {MY_PARAM{1'b1}};
6    end
7  endmodule
```

CHAPTER

FOURTEEN

FILES

14.1 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
```

14.2 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}.h	Model header
{prefix}.cpp	Model C++ file
*{prefix}*___024root.h	Top-level internal header file (from SystemVerilog $root)
*{prefix}*___024root.cpp	Top-level internal C++ file (from SystemVerilog $root)
*{prefix}*___024root*{__n}*.cpp	Additional top-level internal C++ files
*{prefix}*___024root*{__DepSet_hash__n}*.cpp	Additional top-level internal C++ files (hashed to reduce build times)
*{prefix}*___024root__Slow*{__n}*.cpp	Infrequent cold routines
*{prefix}*___024root*{__DepSet_hash__n}*.cpp	Infrequent cold routines (hashed to reduce build times)
*{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
{prefix}{each_verilog_module}{__DepSet_hash__n	Additional lower C++ files (hashed to reduce build times)

For –hierarchical mode, it creates:

V*{hier_block}*/	Directory to Verilate each hierarchical block (from –hierarchical)
*{prefix}*__hierVer.d	Make dependencies of the top module (from –hierarchical)
*{prefix}*_hier.mk	Make file for hierarchical blocks (from –make gmake)
*{prefix}*__hierCMakeArgs.f	Arguments for hierarchical Verilation (from –make cmake)
*{prefix}*__hierMkArgs.f	Arguments for hierarchical Verilation (from –make gmake)
*{prefix}*__hierParameters.v	Module parameters for hierarchical blocks
*{prefix}*__hier.dir	Directory to store .dot, .vpp, .tree of top module (from –hierarchical)

In specific debug and other modes, it also creates:

{prefix}.sarif	SARIF diagnostics (from –diagnostics-sarif)
{prefix}.tree.json	JSON tree information (from –json-only)
{prefix}.tree.meta.json	JSON tree metadata (from –json-only)
{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)
*{prefix}*__inputs.vpp	Pre-processed verilog for all files (from –debug)
{prefix}_ {each_verilog_base_filename}.vpp	Pre-processed verilog for each file (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)
lib*{prefix}*.a	Final archive (default lib mode)
libverilated.a	Runtime for verilated model (default lib mode)
*{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	–prof-pgo data file for *Thread Profile-Guided Optimization*
profile_exec.dat	–prof-exec data file for verilator_gantt

Verilator_gantt may produce the following:

profile_exec.vcd	Gantt report waveform output

CHAPTER

FIFTEEN

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 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.

MAKEFLAGS

Flags created by make to pass to submakes. Verilator searches this variable to determine if a jobserver is used; see --build-jobs.

OBJCACHE

Optionally specifies a caching or distribution program to place in front of all runs of the C++ compiler. For example, "ccache" or "sccache". 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

The VERILATOR_ROOT environment variable is used in several places:

- At ./configure time: If set, it is embedded into the binary, and at runtime if VERILATOR_ROOT is not set, the embedded value is used for the runtime default.
- When verilator is run: If VERILATOR_ROOT is set it will be used to find the verilator_bin executable (this is the actual Verilator binary; verilator is a Perl wrapper). If not set, the verilator script uses other methods to find verilator_bin (looking in the same directory and falling back to $PATH).
- When make is run on the Makefile generated by verilator: The value of VERILATOR_ROOT (falling back to the value embedded in the binary if not set) is used to find the include files (include/verilated.mk).

If you are using a pre-compiled Verilator package, you should not need to set VERILATOR_ROOT - the value embedded in the binary should be correct. In fact this option *does not work* with Verilator packages that have been installed with make install. If a Verilator package has been installed using ./configure --prefix=/some/path && make install and then moved to another location, you cannot use VERILATOR_ROOT to point to the new version.

See *Installation* for more details.

VERILATOR_SOLVER

If set, the command to run as a constrained randomization backend, such as cvc4 --lang=smt2 --incremental. If not specified, it will use the one supplied or found during configure, or z3 --in if empty.

VERILATOR_VALGRIND

If set, the command to run when using the --valgrind option, such as "valgrind –tool=callgrind". If not specified, it will use "valgrind".

CHAPTER

SIXTEEN

MAKE VARIABLES

This section describes the make variables used by Verilator. These may be set by passing them to make e.g. make CXX=my-gcc ....

AR

Optionally overrides the default ar (archive) binary used by the Verilated makefiles. If AR is not set, the version found at configure time is used.

CXX

Optionally overrides the default compiler binary used by the Verilated makefiles. If CXX is not set, the version found at configure time is used. Note the default flags passed to the compiler are determined at configuration time, so changing the CXX compiler version using this variable, as opposed to passing it at configuration time, may not give desired results.

LINK

Optionally overrides the default linker binary used by the Verilated makefiles. If LINK is not set, the version found at configure time is used. Note the default flags passed to the linker are determined at configuration time, so changing the LINK version using this variable, as opposed to passing it at configuration time, may not give desired results.

PERL

Optionally overrides the default perl binary used by the Verilated makefiles. If PERL is not set, the version found at configure time, and compiled into the Verilator binary, is used.

PYTHON3

Optionally overrides the default python3 binary used by the Verilated makefiles. If PYTHON3 is not set, the version found at configure time is used.

CHAPTER

SEVENTEEN

DEPRECATIONS

The following deprecated items are scheduled for future removal:

C++14 compiler support
: Verilator currently requires a C++20 or newer compiler for timing, and a C++14 or newer compiler for both compiling Verilator and compiling Verilated models with –no-timing.

  Verilator will require C++20 or newer compilers for both compiling Verilator and compiling all Verilated models no sooner than May 2025. (Likely to be removed shortly after GitHub removes Ubuntu 20.04 continuous-integration action runners, which are used to test the older C++ standard).

XML output
: Verilator currently supports XML parser output (enabled with *–xml-only*). Support for *–xml-** options will be deprecated no sooner than January 2026.

CHAPTER

EIGHTEEN

CONTRIBUTORS AND ORIGINS

18.1 Authors

When possible, please instead report bugs at Verilator Issues.

The primary author is Wilson Snyder <wsnyder@wsnyder.org>.

Major concepts by Paul Wasson, Duane Galbi, John Coiner, Geza Lore, Yutetsu Takatsukasa, and Jie Xu.

18.2 Contributors

Many people have provided ideas and other assistance with Verilator.

Verilator is receiving significant development support from the CHIPS Alliance, and Antmicro Ltd.

Previous major corporate sponsors of Verilator, by providing significant contributions of time or funds include: Antmicro Ltd., Atmel Corporation, Compaq Corporation, Digital Equipment Corporation, Embecosm Ltd., Hicamp Systems, Intel Corporation, Marvell Inc., Mindspeed Technologies Inc., MicroTune Inc., picoChip Designs Ltd., Sun Microsystems Inc., Nauticus Networks Inc., SiCortex Inc, Shunyao CAD, and Western Digital Inc.

The contributors of major functionality are: Jeremy Bennett, Krzysztof Bieganski, Byron Bradley, Lane Brooks, John Coiner, Duane Galbi, Arkadiusz Kozdra, Geza Lore, Todd Strader, Yutetsu Takatsukasa, Stefan Wallentowitz, Paul Wasson, Jie Xu, and Wilson Snyder.

Some of the people who have provided ideas, and feedback for Verilator include:

David Addison, Tariq B. Ahmad, Nikana Anastasiadis, John David Anglin, Frederic Antonin, Hans Van Antwerpen, Vasu Arasanipalai, Jens Arm, Rohan Arshid, Valentin Atepalikhin, Philip Axer, Gökçe Aydos, Chris Bachhuber, Filip Badáň, Adam Bagley, Sharad Bagri, James Bailey, Robert Balas, Marco Balboni, Matthew Ballance, Ricardo Barbedo, Andrew Bardsley, Ilya Barkov, Matthew Barr, Geoff Barrett, Kaleb Barrett, Daniel Bates, Julius Baxter, Michael Berman, Jean Berniolles, Victor Besyakov, Narayan Bhagavatula, Moinak Bhattacharyya, Kritik Bhimani, David Biancolin, Krzysztof Bieganski, Michael Bikovitsky, David Binderman, Piotr Binkowski, Johan Björk, David Black, Tymoteusz Blazejczyk, Scott Bleiweiss, David van der Bokke, Daniel Bone, Guy Bonneau, Krzysztof Boroński, Gregg Bouchard, Christopher Boumenot, Paul Bowen-Huggett, Nick Bowler, Bryan Brady, Maarten De Braekeleer, Liam Braun, Charlie Brej, J Briquet, John Brownlee, KC Buckenmaier, Gijs Burghoorn, Jeff Bush, Lawrence Butcher, Tony Bybell, Iru Cai, Ted Campbell, Anthony Campos, Chris Candler, Lauren Carlson, Gregory Carver, Donal Casey, Sebastien Van Cauwenberghe, Alex Chadwick, Greg Chadwick, Marcel Chang, Aliaksei Chapyzhenka, Chih-Mao Chen, Guokai Chen, Kefa Chen, Terry Chen, Yangyu Chen, Yi-Chung Chen, Yurii Cherkasov, Hennadii Chernyshchyk, Enzo Chi, Bartłomiej Chmiel, Robert A. Clark, Ryan Clarke, Allan Cochrane, Keith Colbert, Quentin Corradi, Nassim Corteggiani, Gianfranco Costamagna, February Cozzocrea, Sean Cross, George Cuan, Michal Czyz, Joe DErrico, Jim Dai, Lukasz Dalek, Laurens van Dam, Gunter Dannoritzer, Ashutosh Das, Julian Daube, Greg Davill, Bernard Deadman, Peter Debacker, Josse Van Delm, John Demme, Mike Denio, John Deroo, Philip Derrick, Aadi Desai, John Dickol, Ruben Diez, Danny Ding, Jacko Dirks, Ivan Djordjevic, Brad Dobbie, Paul Donahue, Jonathon Donaldson, Anthony Donlon, Caleb Donovick, Larry Doolittle, Leendert van Doorn, Sebastian Dressler, Jonathan Drolet, Justin

Yao Du, Maciej Dudek, Alex Duller, Jeff Dutton, Tomas Dzetkulic, Usuario Eda, Charles Eddleston, Chandan Egbert, Joe Eiler, Ahmed El-Mahmoudy, Trevor Elbourne, Mats Engstrom, Robert Farrell, Julien Faucher, Olivier Faure, Eugene Feinberg, Eugen Fekete, Fabrizio Ferrandi, Udi Finkelstein, Brian Flachs, Bill Flynn, Andrea Foletto, Alex Forencich, Aurelien Francillon, Bob Fredieu, Manuel Freiberger, Mostafa Gamal, Vito Gamberini, Mostafa Garnal, Benjamin Gartner, Christian Gelinek, Richard E George, Peter Gerst, Glen Gibb, Michael Gielda, Barbara Gigerl, Nimrod Gileadi, Shankar Giri, Dan Gisselquist, Szymon Gizler, Petr Gladkikh, Sam Gladstone, Mariusz Glebocki, Embedded Go, Andrew Goessling, Amir Gonnen, Chitlesh Goorah, Tomasz Gorochowik, Kai Gossner, Tarik Graba, Sergi Granell, Al Grant, Nathan Graybeal, Alexander Grobman, Qian Gu, Xuan Guo, Prabhat Gupta, Deniz Güzel, Driss Hafdi, Abdul Hameed, Neil Hamilton, James Hanlon, Tang Haojin, Øyvind Harboe, Jannis Harder, David Harris, Junji Hashimoto, Thomas Hawkins, Mitch Hayenga, Harald Heckmann, Robert Henry, Stephen Henry, Sebastian Hesselbarth, David Hewson, Jamey Hicks, Joel Holdsworth, Andrew Holme, Peter Holmes, Hiroki Honda, Alex Hornung, Pierre-Henri Horrein, David Horton, Peter Horvath, Jae Hossell, Kuoping Hsu, Shou-Li Hsu, Teng Huang, Steven Hugg, Alan Hunter, James Hutchinson, Tim Hutt, Ehab Ibrahim, Edgar E. Iglesias, Shahid Ikram, Jamie Iles, Fuad Ismail, Vighnesh Iyer, Ben Jackson, Daniel Jacques, Shareef Jalloq, Marlon James, Krzysztof Jankowski, Eyck Jentzsch, HyungKi Jeong, Iztok Jeras, Pawel Jewstafjew, Alexandre Joannou, James Johnson, Christophe Joly, Justin Jones, William D. Jones, Abe Jordan, Larry Darryl Lee Jr., Franck Jullien, James Jung, Mike Kagen, Arthur Kahlich, Kaalia Kahn, Guy-Armand Kamendje, Vasu Kandadi, Yoshitomo Kaneda, Kanad Kanhere, Patricio Kaplan, Pieter Kapsenberg, Rafal Kapuscik, Ralf Karge, Per Karlsson, Dan Katz, Sol Katzman, Ian Kennedy, Ami Keren, Fabian Keßler, Michael Killough, Sun Kim, Jonathan Kimmitt, Olof Kindgren, Kevin Kiningham, Cameron Kirk, Dan Kirkham, Aleksander Kiryk, Sobhan Klnv, Gernot Koch, Jack Koenig, Soon Koh, Nathan Kohagen, Steve Kolecki, Brett Koonce, Will Korteland, Andrei Kostovski, Wojciech Koszek, Varun Koyyalagunta, Arkadiusz Kozdra, Markus Krause, David Kravitz, Adam Krolnik, Roland Kruse, Mahesh Kumashikar, Andreas Kuster, Sergey Kvachonok, Charles Eric LaForest, Kevin Laeufer, Ed Lander, Steve Lang, Pierre Laroche, Stephane Laurent, Walter Lavino, Christian Leber, David Ledger, Alex Lee, Larry Lee, Yoda Lee, Michaël Lefebvre, Dag Lem, Igor Lesik, John Li, Kay Li, Zixi Li, Davide Libenzi, Nandor Licker, Eivind Liland, Ícaro Lima, Kevin Lin, Yu-Sheng Lin, Charlie Lind, Andrew Ling, Jiuyang Liu, Joey Liu, Paul Liu, Derek Lockhart, Jake Longo, Geza Lore, Arthur Low, Jose Loyola, Stefan Ludwig, Dan Lussier, Konstantin Lübeck, Fred Ma, Liwei Ma, Duraid Madina, Oleh Maksymenko, Affe Mao, Julien Margetts, Chick Markley, Alexis Marquet, Mark Marshall, Alfonso Martinez, Unai Martinez-Corral, Adrien Le Masle, Yves Mathieu, Vladimir Matveyenko, Patrick Maupin, Stan Mayer, Jordan McConnon, Conor McCullough, Jason McMullan, Elliot Mednick, Yuan Mei, Andy Meier, Luiza de Melo, Rodrigo A. Melo, Benjamin Menküc, Jake Merdich, David Metz, Wim Michiels, Miodrag Milanović, Darryl Miles, Kevin Millis, Andrew Miloradovsky, David Moberg, Wai Sum Mong, Peter Monsson, Anthony Moore, Sean Moore, Stuart Morris, Dennis Muhlestein, John Murphy, Matt Myers, Nathan Myers, Richard Myers, Alex Mykyta, Eric Müller, Dimitris Nalbantis, Peter Nelson, Felix Neumärker, Bob Newgard, Cong Van Nguyen, Rachit Nigam, Toru Niina, Paul Nitza, Yossi Nivin, Pete Nixon, Lisa Noack, Mark Nodine, Michael Nolan, Andrew Nolte, Joseph Nwabueze, Kevin Nygaard, Kuba Ober, Krzysztof Obłonczek, Andreas Olofsson, Baltazar Ortiz, Aleksander Osman, Don Owen, Tim Paine, Deepa Palaniappan, James Pallister, Vassilis Papaefstathiou, Sanggyu Park, Brad Parker, Risto Pejašinović, Seth Pellegrino, Morten Borup Petersen, Dan Petrisko, Wesley Piard, Maciej Piechotka, David Pierce, Cody Piersall, T. Platz, Michael Platzer, Dominic Plunkett, Nolan Poe, David Poole, Michael Popoloski, Roman Popov, Aylon Chaim Porat, Oron Port, Rich Porter, Rick Porter, Stefan Post, Niranjan Prabhu, Damien Pretet, Harald Pretl, Bill Pringlemeir, Usha Priyadharshini, Mark Jackson Pulver, Prateek Puri, Han Qi, Jiacheng Qian, Marshal Qiao, Raynard Qiao, Yujia Qiao, Jasen Qin, Frank Qiu, Nandu Raj, Kamil Rakoczy, Danilo Ramos, Drew Ranck, Chris Randall, Anton Rapp, Josh Redford, Odd Magne Reitan, Frédéric Requin, Dustin Richmond, Samuel Riedel, Alberto Del Rio, Eric Rippey, Narcis Rodas, Oleg Rodionov, Ludwig Rogiers, Paul Rolfe, Michail Rontionov, Arjen Roodselaar, Arthur Rosa, Tobias Rosenkranz, Yernagula Roshit, Ryszard Rozak, Huang Rui, Graham Rushton, Jan Egil Ruud, Denis Rystsov, Pawel Sagan, Robert Sammelson, Adrian Sampson, John Sanguinetti, Josep Sans, Luca Sasselli, Martin Scharrer, Martin Schmidt, Jonathan Schröter, Julie Schwartz, Galen Seitz, Sam Shahrestani, Joseph Shaker, Mark Shaw, Salman Sheikh, Zhou Shen, Hao Shi, James Shi, Michael Shinkarovsky, Rafael Shirakawa, Jeffrey Short, S Shuba, Fan Shupei, Ethan Sifferman, Anderson Ignacio da Silva, Rodney Sinclair, Ameya Vikram Singh, Sanjay Singh, Frans Skarman, Nate Slager, Steven Slatter, Mladen Slijepcevic, Brian Small, Garrett Smith, Gus Smith, Tim Snyder, Maciej Sobkowski, Stan Sokorac, Alex Solomatnikov, Flavien Solt, Wei Song, Trefor Southwell, Martin Stadler, Art Stamness, David Stanford, Krzysztof Starecki, Baruch Sterin, John Stevenson, Pete Stevenson, Patrick Stewart, Rob Stoddard, Tood Strader, John Stroebel, Ray Strouble, Sven Stucki, Howard Su, Udaya Raj Subedi, Emerson Suguimoto, Gene Sullivan, Qingyao Sun, Renga Sundararajan, Kuba Sunderland-Ober, Gustav Svensk, Rupert Swarbrick, Jevin Sweval, Paul Swirhun, Shinya T-Y, Thierry Tambe, Jesse Taube, Drew Taus-

sig, Christopher Taylor, Greg Taylor, Jose Tejada, Sören Tempel, Peter Tengstrand, Wesley Terpstra, Rui Terra, Stefan Thiede, Justin Thiel, Gary Thomas, Ian Thompson, Kevin Thompson, Mike Thyer, Hans Tichelaar, Tudor Timi, Viktor Tomov, Steve Tong, Topa Topino, Àlex Torregrosa, Topa Tota, Michael Tresidder, Lenny Truong, David Turner, Neil Turton, Hideto Ueno, Mike Urbach, Joel Vandergriendt, Srini Vemuri, Srinivasan Venkataramanan, Yuri Victorovich, Ivan Vnučec, Bogdan Vukobratovic, Holger Waechtler, Philipp Wagner, Stefan Wallentowitz, Johannes Walter, CY Wang, Chuxuan Wang, Shawn Wang, Yilou Wang, Zhanglei Wang, Greg Waters, Thomas Watts, Eugene Weber, John Wehle, Tianrui Wei, David Welch, Thomas J Whatson, Martin Whitaker, Marco Widmer, Leon Wildman, Daniel S. Wilkerson, Daniel Wilkerson, Gerald Williams, Trevor Williams, Don Williamson, Jan Van Winkel, Jeff Winston, Joshua Wise, Clifford Wolf, Johan Wouters, Paul Wright, Tobias Wölfel, Junyi Xi, Ding Xiaoliang, Liu Xiaoyi, Jinyan Xu, Mandy Xu, Pengcheng Xu, Shanshan Xu, Yan Xu, Yinan Xu, SU YANG, Felix Yan, Jiaxun Yang, Luke Yang, Amir Yazdanbakhsh, Chentai (Seven) Yuan, Florian Zaruba, Mat Zeno, Keyi Zhang, Xi Zhang, Huanghuang Zhou, Yike Zhou, Jiamin Zhu, Ryan Ziegler.

Thanks to them, and all those we've missed mentioning above, and to those whom have wished to remain anonymous.

18.3 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 for Multimedia and Network Processor development inside Digital. Duane Galbi took over the active development of Verilator, and added several performance enhancements, and 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, 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.

In 2022, Verilator 5.000 was released with IEEE scheduling semantics, fork/join, delay handling, DFG performance optimizations, and other improvements.

Currently, various language features and performance enhancements are added as the need arises, focusing on completing Universal Verification Methodology (UVM, IEEE 1800.2-2017) support.

CHAPTER

NINETEEN

REVISION HISTORY

Changes are contained in the Changes file of the distribution, and also summarized below. To subscribe to new versions, see Verilator Announcements.

19.1 Revision History and Change Log

The changes in each Verilator version are described below. The contributors that suggested or implemented a given issue are shown in []. Thanks!

19.1.1 Verilator 5.040 2025-08-30

Other:

- Add ENUMITEMWIDTH error, and apply to X-extended and ranged values.
- Add NOEFFECT warning, replacing previous &96;foreach&96; error.
- Add SPECIFYIGN warning for specify constructs that were previously silently ignored.
- Add PARAMNODEFAULT error, for parameters without defaults.
- Add enum base data type, wire data type, and I/O versus data declaration checking per IEEE.
- Add PROTOTYPEMIS error on missing and mismatching prototypes (#6206) (#6207). [Alex Solomatnikov]
- Add error when trying to assign class object to variable of non-class types (#6237). [Igor Zaworski, Antmicro Ltd.]
- Add ALWNEVER warning, for &96;always @*&96; that never execute (#6291).
- Add separate coverage counters for toggles 0->1 and 1->0 (#6086). [Ryszard Rozak, Antmicro Ltd.]
- Add error on class 'function static'.
- Add error on force/release non-constant selects.
- Add &96;-DVERILATOR=1&96; definition to compiler flags when using verilated.mk.
- Support member-level triggers for virtual interfaces (#5166) (#6148). [Yilou Wang]
- Support unassigned virtual interfaces (#5265) (#6245). [Szymon Gizler, Antmicro Ltd.]
- Support randomization of scope variables with 'std::randomize()' (#5438) (#6185). [Yilou Wang]
- Support disabling a fork in additional contexts (#5432 partial) (#6174) (#6183). [Ryszard Rozak, Antmicro Ltd.]
- Support bit queue streaming (#5830) (#6103). [Paul Swirhun]
- Support &96;$fread&96; with missing start (#6125). [Iztok Jeras]
- Support unpacked array &96;with&96; methods (#6134).

- Support Verilog real ports as SystemC double ports (#6136) (#6158). [George Polack]
- Support &96;$countones&96; in constraints (#6144 partial) (#6235). [Ryszard Rozak, Antmicro Ltd.]
- Support disable dotted references (#6154). [Ryszard Rozak, Antmicro Ltd.]
- Support randomize() on class member selects (#6161) (#6195). [Igor Zaworski, Ryszard Rozak, Antmicro Ltd.]
- Support multiple variables on RHS of a &96;force&96; assignment (#6163). [Artur Bieniek, Antmicro Ltd.]
- Support covergroup extends, etc., as unsupported (#6160). [Artur Bieniek, Antmicro Ltd.]
- Support parameter resolution of 1D unpacked array slices (#6257) (#6268). [Michael Bedford Taylor]
- Support generic interfaces (#6272). [Igor Zaworski, Antmicro Ltd.]
- Support disabling a fork from within that fork (#6314). [Ryszard Rozak, Antmicro Ltd.]
- Support future sampled value functions.
- Support simple disable within task (#6334). [Ryszard Rozak, Antmicro Ltd.]
- Support recursive constant functions.
- Change control file &96;public_flat_*&96; and other signal attributes to support __ in names (#6140).
- Change runtime to exit() instead of abort(), unless under +verilated+debug.
- Change &96;$display("%p")&96; to remove space after &96;}&96;.
- Improve &96;–skip-identical&96; to skip on identical input file contents (#6109).
- Improve testing on FreeBSD (#6328). [Aleksander Kiryk]
- Optimize to return memory when using -build (#6192) (#6226). [Michael B. Taylor]
- Optimize 2 ** X to 1 << X if base is signed (#6203). [Max Wipfli]
- Optimize more complex combinational logic in DFG (#6205) (#6209) (#6298). [Geza Lore]
- Optimize combinational cycles through arrays in DFG (#6210). [Geza Lore]
- Optimize variable removal in scoped DFG (#6260). [Geza Lore]
- Optimize acyclic DFG components into the original acyclic sub-graph. (#6261). [Geza Lore]
- Optimize multiplexers in DFG synthesis (#6331). [Geza Lore]
- Optimize interfaces in DFG (#6332). [Geza Lore]
- Optimize logic in non-virtual interfaces with DFG (#6347). [Geza Lore]
- Fix loop initialization visibility outside loop (#4237).
- Fix constructor parameters in inheritance hierarchies (#6036) (#6070). [Petr Nohavica]
- Fix replicate of negative giving 'REPLICATE has no expected width' internal error (#6048) (#6229).
- Fix cmake &96;-Wno&96; compiler flag testing (#6145). [Martin Stadler]
- Fix class extends dotted error (#6162). [Igor Zaworski, Antmicro Ltd.]
- Fix genvar error with &96;-O0&96; (#6165). [Max Wipfli]
- Fix uninitialized thread PGO counters (#6167). [Bartłomiej Chmiel, Antmicro Ltd.]
- Fix additional UNOPTFLAT combinational cycles automatically in DFG (#6168) (#6173) (#6176). [Geza Lore]
- Fix omitting error when assigning to an input (#6169). [Artur Bieniek, Antmicro Ltd.]
- Fix parameter-dependent type linking (#6170). [Igor Zaworski, Antmicro Ltd.]

- Fix param-dependent class typedef linking (#6171). [Igor Zaworski, Antmicro Ltd.]
- Fix virtual interface member propagation (#6175) (#6184). [Yilou Wang]
- Fix &96;–coverage-expr&96; null pointer dereference (#6181). [Igor Zaworski, Antmicro Ltd.]
- Fix conflicting function/class name linking error (#6182). [Igor Zaworski, Antmicro Ltd.]
- Fix negate of wide structure selections (#6186).
- Fix VPI signal range order (#6189) (#6200). [Ibrahim Burak Yorulmaz]
- Fix structure select causing 'Wide Op' error (#6191). [Danny Oler]
- Fix automatic task variables in unrolled loops with forks (#6194) (#6201). [Danny Oler]
- Fix 'driver same component' assertion (#6211) (#6215). [Geza Lore]
- Fix &96;–stats&96; overridden by skipping identical build (#6220). [Geza Lore]
- Fix MODDUP with duplicate packages to take first package (#6222).
- Fix replicate with unsigned count but MSB set (#6231) (#6233). [Geza Lore]
- Fix randomize on function-local variable (#6234).
- Fix queue typedef with unbounded slice (#6236).
- Fix error when force assignment is used with ref function args (#6244). [Ryszard Rozak, Antmicro Ltd.]
- Fix write of 0 in '%c' (#6248) (#6249). [Rodrigo Batista de Moraes]
- Fix coverage of variables of complex types (#6250). [Ryszard Rozak, Antmicro Ltd.]
- Fix broken support of unassigned virtual interfaces (#6253) (#6338). [Szymon Gizler, Antmicro Ltd.]
- Fix partial DFG conversion of concat assignments (#6255). [Geza Lore]
- Fix dynamic cast purity (#6267). [Igor Zaworski, Antmicro Ltd.]
- Fix same variable on the RHS forced to two different LHSs. (#6269). [Artur Bieniek, Antmicro Ltd.]
- Fix spurious VPI value change callbacks (#6274). [Todd Strader]
- Fix stray ']' in Verilog code output for non-constant select (#6277). [Geza Lore]
- Fix hierarchical NBAs (#6286) (#6300). [Geza Lore]
- Fix variables hiding package imports (#6289). [Johan Wouters]
- Fix DFG circular driver tracing. [Geza Lore]
- Fix no matching function calls for randomized &96;VlWide&96; in unpacked and dynamic arrays (#6290). [Mateusz Gancarz, Antmicro Ltd.]
- Fix PowerPC support (#6292). [Sergey Fedorov]
- Fix referencing module variables above classes (#6304). [Artur Bieniek, Antmicro Ltd.]
- Fix direct NBA to dynamically-sized variable (#6310). [Artur Bieniek, Antmicro Ltd.]
- Fix static vars under member select (#6313). [Igor Zaworski, Antmicro Ltd.]
- Fix expression type comparison (#6316). [Igor Zaworski, Antmicro Ltd.]
- Fix of inline constraints with member selects (#6321). [Igor Zaworski, Antmicro Ltd.]
- Fix corner case bugs in module and variable inlining (#6322). [Geza Lore]
- Fix queue extend to check bounds (#6324). [Aleksander Kiryk]

- Fix gathering sensitivities from virtual interface members (#6325). [Aleksander Kiryk]
- Fix FreeBSD missing headers (#6326). [Aleksander Kiryk]
- Fix to select UDPs when they are the only candidate for a top module.
- Fix splitting of assignments to SC variables (#6329) (#6336). [Geza Lore]
- Fix to localize for super constructors with function calls as arguments (#6330). [Igor Zaworski, Antmicro Ltd.]
- Fix wide select expansion and substitution (#6341) (#6345). [Geza Lore]
- Fix upcasting class type parameters (#6344). [Krzysztof Bieganski, Antmicro Ltd.]
- Fix undefined weak link for Apple GCC etc (#6348). [Congcong Cai]

19.1.2 Verilator 5.038 2025-07-08

Important:

- Change &96;–assert&96; to be the default; use &96;–no-assert&96; for legacy behavior and faster runtimes.

Other:

- Support redeclaring type as non-type; major parsing change (#2412) (#6020) (#6042) (#6044).
- Support scoped &96;new&96; (#4199).
- Support elaboration-time printing of unpacked array with &96;%p&96; (#4732).
- Support constrained random for associative arrays (#5985) (#5986). [Yilou Wang]
- Support assignments to concatenations with impure RHS (#6002). [Ryszard Rozak, Antmicro Ltd.]
- Support SARIF JSON diagnostic output with &96;–diagnostics-sarif&96;. (#6017)
- Support 1-bit params with -G and -pvalue (#6051) (#6082). [Paul Swirhun]
- Support &96;specparam&96; (#5767) (#6142).
- Support &96;$timeformat&96; with missing arguments (#6113). [Alex Solomatnikov]
- Support non-overlapping blocking/non-blocking assignments (#6137). [Geza Lore]
- Support parameter forward types.
- Support constant functions with left-hand-side concatenates.
- Add PROCINITASSIGN on initial assignments to process variables (#2481). [Niraj Menon]
- Add BADVLTPRAGMA on unknown Verilator pragmas (#5945). [Shou-Li Hsu]
- Add ternary operator into branch coverage (#5880). [Ryszard Rozak, Antmicro Ltd.]
- Add aggregate type error checks (#5570) (#5950). [Shou-Li Hsu]
- Add &96;–work&96; library-selection option (#5891 partial).
- Add &96;–filter-type&96; to verilator_coverage (#6030). [Ryszard Rozak, Antmicro Ltd.]
- Add &96;–hierarchical-threads&96; (#6037). [Bartłomiej Chmiel]
- Add &96;MODMISSING&96; error, in place of unnamed error (#6054). [Paul Swirhun]
- Add DFG binToOneHot pass to generate one-hot decoders (#6096). [Geza Lore]
- Add hint of the signed right-hand-side in oversized replication error (#6098). [Peter Birch]
- Improve hierarchical scheduling visualization in V3ExecGraph (#6009). [Bartłomiej Chmiel, Antmicro Ltd.]
- Improve DPI temporary 'for' loop performance (#6079). [Bartłomiej Chmiel, Antmicro Ltd.]

- Improve memory usage for SenTrees in V3OrderProcessDomains (#6112). [Geza Lore]
- Improve docker image size (#6139). [Tobias Rosenkranz, bitaggregat GmbH]
- Update docker base image to Ubuntu 24.04 (#6147). [Tobias Rosenkranz, bitaggregat GmbH]
- Optimize DFG De Morgan patterns (#6090). [Geza Lore]
- Optimize DFG variable elimination (#6091). [Geza Lore]
- Optimize DFG PUSH_SEL_THROUGH_CONCAT pattern (#6092). [Geza Lore]
- Optimize DFG before V3Gate (#6141). [Geza Lore]
- Optimize DFG peephole patterns (#6149). [Geza Lore]
- Optimize constification within Expand and Subst stages (#6111). [Geza Lore]
- Fix –x-initial and –x-assign random stability (#2662) (#5958) (#6018) (#6025) (#6075). [Todd Strader]
- Fix trace hierarchical-name runtime errors (#5668) (#6076). [Paul Swirhun]
- Fix streaming operator packing order (#5903) (#6077). [Paul Swirhun]
- Fix filename backslash escapes in C code (#5947).
- Fix C++ widths in V3Expand (#5953) (#5975). [Geza Lore]
- Fix dependencies from different hierarchical schedules (#5954). [Bartłomiej Chmiel, Antmicro Ltd.]
- Fix constant propagation of post-expand stages (#5955) (#5963) (#5969) (#5972) (#5983).
- Fix sign extension of signed compared with unsigned case items (#5968).
- Fix always processes ignoring $finish (#5971). [Hennadii Chernyshchyk]
- Fix method calls and new to access interface arrays (#5973). [Nick Brereton]
- Fix streaming to/from packed arrays (#5976). [Geza Lore]
- Fix segfault writing timestamp to VCD after buffer flush (#5978) (#5980). [Dominick Grochowina]
- Fix inconsistent assignment error with split-var (#5984) (#5988). [Yutetsu TAKATSUKASA]
- Fix AstAssignW conversion (#5991) (#5992). [Ryszard Rozak, Antmicro Ltd.]
- Fix const-bit-op-tree with single-bit masks (#5993) (#5998). [Yutetsu TAKATSUKASA]
- Fix arithmetic right-shift by constants over 32 bits (#5994). [Zhen Yan]
- Fix array bounds checking with class member selects (#5996) (#5997). [Krzysztof Starecki]
- Fix checking for too-wide divide and modulus (#6003) (#6006). [Zhen Yan]
- Fix folding of LteS in DfgPeephole (#6000) (#6004). [Geza Lore]
- Fix slicing of AstExprStmt nodes (#6005). [Ryszard Rozak, Antmicro Ltd.]
- Fix skipped generate blocks in toggle coverage (#6010). [Ryszard Rozak, Antmicro Ltd.]
- Fix arithmetic left-shift by constants over 32 bits (#6007) (#6015). [Zhen Yan]
- Fix concatenation and type casting (#6012) (#6013). [Todd Strader]
- Fix wrong optimization result of shifted out variable (#6016) (#6019). [Yutetsu TAKATSUKASA]
- Fix missing FreeBSD include (#6027) (#6028). [Joel Bodenmann]
- Fix associative arrays with enum keys (#6034) (#6035). [Petr Nohavica]
- Fix GCC 10 read-only linker error (#6040). [Todd Strader]

- Fix WIDTHCONCAT on packed pattern assignment (#6045). [Dan Petrisko]
- Fix type operator for interface signals (#6049) (#6050). [Todd Strader]
- Fix V3OrderParallel scoring contraction hang (#6052). [Bartłomiej Chmiel, Antmicro Ltd.]
- Fix virtual interface array typedef expressions (#6057).
- Fix DFG peephole select conditionals (#6064) (#6071). [Geza Lore]
- Fix &96;–skip-identical&96; skipping on some errors (#6066). [Todd Strader]
- Fix class typedef elaboration (#6080). [Kamil Rakoczy, Antmicro Ltd.]
- Fix unpacked to packed parameter assignment (#6081) (#6088). [Todd Strader]
- Fix casting reals to large integrals (#6085). [Todd Strader]
- Fix nested hier blocks workers error (#6087). [Bartłomiej Chmiel, Antmicro Ltd.]
- Fix &96;–lib-create&96; with double-underscore (#6099).
- Fix DFG binToOneHot table index missing driver (#6100). [Geza Lore]
- Fix decoding octal string escapes with 1-2 digits (#6108).
- Fix colon-divide operator without space (#6121). [Alex Solomatnikov]
- Fix variables declared in fork after taskify (#6126). [Kamil Rakoczy, Antmicro Ltd.]
- Fix method calls without parenthesis (#6127). [Alex Solomatnikov]
- Fix &96;pre_randomize&96;/&96;post_randomize&96; when no randomize (#6128). [Alex Solomatnikov]
- Fix wide non-blocking assignment mis-optimization (#6150) (#6152) (#6155). [Todd Strader]
- Fix interface array connections with non-zero low declaration index.
- Fix developer build error on MacOS/Flex2.6.4 (#6153). [Paul Swirhun]
- Fix crash with –dumpi-V3LinkDot without –debug (#6159). [Igor Zaworski, Antmicro Ltd.]
- Fix dereferencing stale iterator in DfgVertex::scopep() (#6225) (#6227). [Geza Lore]
- Fix component numbers of new Vertices in V3DfgBreakCycles (#6225) (#6228). [Geza Lore]

19.1.3 Verilator 5.036 2025-04-27

Important:

- Change &96;–output-groups&96; to default to value of &96;–build-jobs&96; (#5751). Those using build farms may need to now use &96;–output-groups 0&96; or otherwise.
- Support user-defined primitives (UDPs) (#468) (#5807) (#5936). [Zhou Shen, Krzysztof Sychla, et al]
- Add &96;–trace-saif&96; for SAIF power traces (#5812) (#5914). [Mateusz Gancarz, Antmicro Ltd.]

Other:

- Change &96;–trace&96; to &96;–trace-vcd&96;.
- Support simple &96;checker&96; blocks (#4066). [Srinivasan Venkataramanan]
- Support force/release with a variable reference (#5721) (#5810). [Bartłomiej Chmiel, Antmicro Ltd.]
- Support constraint random for StructArray (#5805) (#5937). [Yilou Wang]
- Support command-line overriding &96;define (#5900) (#5908). [Brian Li]
- Support soft unions (#5912) (#5932). [Robin Heinemann]

- Support &96;$setuphold&96; (#5884). [Krzysztof Sychla]
- Support assigning unpacked arrays to queues (#5924) (#5928). [Brian Li]
- Support &96;systemc_interface and related inside &96;class&96;.
- Support class extends with arguments.
- Add multi-thread hierarchical simulation (#2583) (#5871). [Bartłomiej Chmiel, Antmicro Ltd.]
- Add check for &96;let&96; misused in statement context (#5733).
- Add used language to &96;–preproc-resolve&96; output (#5795). [Kamil Rakoczy, Antmicro Ltd.]
- Add &96;–make json&96; to enable integration with non-make/cmake build systems (#5799). [Andrew Voznytsa]
- Add numactl-like automatic assignment of processor affinity (#5911).
- Add ccache support for generated cmake files (#5926) (#5930). [Andrew Voznytsa]
- Add visualization of multi-threaded waiting time with verilator_gantt (#5929). [Bartłomiej Chmiel, Antmicro Ltd.]
- Add suggesting isolate_assignments to UNOPTFLAT (#5942). [John Khoo]
- Add profiling of hierarchical mtasks with verilator_gantt (#5956). [Bartłomiej Chmiel, Antmicro Ltd.]
- Add empty veriuser.h for legacy compatibility.
- Add DEPRECATED warning on &96;–xml-only&96; and &96;–xml-output&96;.
- Remove unused gtkwave/wavealloca.h. [Geza Lore]
- Optimize automatic splitting of some packed variables (#5843). [Geza Lore]
- Optimize trigger vector in whole words (#5857). [Geza Lore]
- Fix parameters referencing interface fields (#1593) (#5910). [Ryszard Rozak, Antmicro Ltd.]
- Fix interface array assignments (#5270) (#5633) (#5869). [Nick Brereton]
- Fix change detection at time 0 (#5499) (#5864). [Geza Lore]
- Fix foreach of associative array inside a constraint block (#5727) (#5841). [Yilou Wang]
- Fix reset of automatic function variables (#5747). [Augustin Fabre]
- Fix invalid code motion over branches (#5811) (#5814). [Geza Lore]
- Fix sorting of wide SenItems (#5816). [Geza Lore]
- Fix tcmalloc static link and non-22.04 builds (#5817) (#5818). [Geza Lore]
- Fix error on out-of-range lvalue part select (#5820).
- Fix UNOPTFLAT warnings with &96;–coverage-trace&96; and always_comb (#5821).
- Fix function locals in SenExprBuilder (#5822). [Geza Lore]
- Fix type_id package scope resolution (#5826). [Krzysztof Bieganski, Antmicro Ltd.]
- Fix &96;rand_mode&96; method with cast (#5831).
- Fix invalidating variable caches in SenExprBulider (#5834) (#5835). [Geza Lore]
- Fix assignment pattern as function argument (#5839).
- Fix checking built-in method arguments (#5839).
- Fix splitting of packed ports with non-zero based ranges (#5842). [Geza Lore]

- Fix delays inside interface functions, and classes inside interfaces (#5846).
- Fix NBA shared flag reuse (#5848). [Geza Lore]
- Fix multiple expression coverage issues (#5849 partial) (#5867) (#5870). [Todd Strader]
- Fix unresolved typedefs as parameters (#5850). [Eugene Feinberg, Brian Li]
- Fix removal of callbacks no longer in current list (#5851) (#5852). [Gilberto Abram]
- Fix segmentation fault on member compare (#5853).
- Fix recursive error on virtual interfaces (#5854). [Yilou Wang]
- Fix streaming of unpacked arrays concatenations (#5856). [Ryszard Rozak, Antmicro Ltd.]
- Fix Windows paths in Perl (#5858) (#5860). [Tobias Jensen]
- Fix algorithm header portability in V3Os.cpp (for std::replace) (#5861). [William D. Jones]
- Fix &96;$fscanf&96; not returning -1 on EOF (#5881).
- Fix PGO profiling for multi-threaded hierarchical scenarios (#5888). [Bartłomiej Chmiel, Antmicro Ltd.]
- Fix V3Gate assertion on eliminated circular logic (#5889) (#5898). [Geza Lore]
- Fix process comparisons (#5896).
- Fix ccache with clang (#5899). [Geza Lore]
- Fix delayed assignment malformed LHS assertion (#5904).
- Fix segfault in fork synchronization (#5906). [Krzysztof Bieganski, Antmicro Ltd.]
- Fix &96;new this&96; (#5909).
- Fix assignments with stream expressions and unpacked arrays (#5915). [Ryszard Rozak, Antmicro Ltd.]
- Fix LATCH warning for automatic variables (#5918). [Yutetsu TAKATSUKASA]
- Fix %% on elaboration severity tasks (#5922). [Ethan Sifferman]
- Fix port default values with &96;–coverage-line&96; creating &96;0=0&96; (#5920). [Drew Ranck]
- Fix vlSelf error on fork repeats (#5927). [Drew Ranck]
- Fix missing C++ regeneration when Verilog files are updated (#5934). [Zhouyi Shen]
- Fix stream expressions (#5938). [Ryszard Rozak, Antmicro Ltd.]
- Fix packed selection using over 32-bit index (#5957).

19.1.4 Verilator 5.034 2025-02-24

Important:

- This is likely the last version to support Ubuntu 20.04, which is at end-of-life.
- Add expression coverage (#4677) (#5719). [Todd Strader]

Other:

- Add &96;COVERIGN&96; warning, as a more specific UNSUPPORTED error.
- Add &96;–public-ignore&96; to ignore public metacomments (#5716). [Andrew Nolte]
- Add &96;–preproc-token-limit&96; (#5768). [Krzysztof Bieganski, Antmicro Ltd.]
- Add lint error on importing package within a class (#5634) (#5679). [Nick Brereton]
- Add &96;–preproc-resolve&96; for modules in preprocessor output (#5789). [Kamil Rakoczy, Antmicro Ltd.]

- Support multidimensional array access via VPI (#2812) (#5573). [Krzysztof Starecki]
- Support nested classes (#4178) (#5778). [Kamil Rakoczy, Antmicro Ltd.]
- Support VPI array accessors (#5612). [Moubarak Jeje]
- Support unpacked structure constrained randomization (#5657) (#5759). [Yilou Wang]
- Support generated classes (#5665). [Shou-Li Hsu]
- Support constraints on associative array user-defined keys (#5671) (#5729). [Yilou Wang]
- Support &96;+incdir&96; with multiple directories.
- Support integer atom type ports in &96;–hierarchical&96; (#5748). [Bartłomiej Chmiel, Antmicro Ltd.]
- Support constrained random for arrays in structs (#5765) (#5802). [Yilou Wang]
- Support selects on arbitrary string expressions (#5773). [Krzysztof Bieganski, Antmicro Ltd.]
- Support side effects of form 'variable[index_function()]++'.
- Improve hierarchical DPI wrapper scheduling performance (#2583) (#5734). [Bartłomiej Chmiel, Antmicro Ltd.]
- Improve VPI write errors (#5712). [Andrew Nolte]
- Improve &96;resetall support (#5728) (#5730). [Ethan Sifferman]
- Optimize labels as final &96;if&96; block statements (#5744).
- Optimize empty function definition bodies (#5750).
- Optimize splitting trigger computation and dump (#5798). [Geza Lore]
- Fix error message when call task as a function (#3089). [Matthew Ballance]
- Fix force VPI public signal visibility (#5225). [Frédéric Requin]
- Fix VPI iteration over hierarchy (#5314) (#5731). [Natan Kreimer]
- Fix constrained random for > 64-bit associative arrays (#5670) (#5682). [Yilou Wang]
- Fix mis-optimizing away &96;$urandom&96; (#5703). [Parker Schless]
- Fix packages with &96;–public-depth 1&96; (#5708). [Andrew Nolte]
- Fix V3Simulate constant reuse (#5709). [Geza Lore]
- Fix man pages what-is section (#5710). [Ahmed El-Mahmoudy]
- Fix pattern assignment to real inside struct (#5713).
- Fix %p format output for real inside struct (#5713).
- Fix segfault when only enum value referenced in package (#5714). [Dan Katz]
- Fix &96;BLKSEQ&96; on suspendable processes (#5722). [Krzysztof Bieganski, Antmicro Ltd.]
- Fix vpiDefName issues with non-inlined scopes and dpi conflicts (#5732). [Andrew Nolte]
- Fix inlined expression with assignment under LHS of NBA (#5736) (#5740). [Geza Lore]
- Fix duplicate-named class variable equivalence (#5737).
- Fix &96;-j&96; ignored after &96;-f&96; (#5749). [Luca Colagrande]
- Fix handling forced assigns in V3Life (#5757). [Bartłomiej Chmiel, Antmicro Ltd.]
- Fix VFileContent reference count (#5769) (#5771). [Dave Sargeant]

- Fix ignoring joins in stringify in preprocessor (#5777). [Krzysztof Bieganski, Antmicro Ltd.]
- Fix unpacked split_var (#5782) (#5785). [Yutetsu TAKATSUKASA]
- Fix time import error on time parameters (#5786). [Luca Colagrande]
- Fix dpi context functions (#5788). [Ryszard Rozak, Antmicro Ltd.]
- Fix &96;$monitor&96; with dotted references (#5794). [Ahmed Elzeftawi]
- Fix parsing input wire with default and range (#5800). [RJ Cunningham]
- Fix matching language extension options including dots.

19.1.5 Verilator 5.032 2025-01-01

Minor:

- Support queue's assignment &96;push_back/push_front('{})&96; (#5585) (#5586). [Yilou Wang]
- Support basic constrained random for multi-dimensional dynamic array and queue (#5591). [Yilou Wang]
- Support vpiDefName (#3906) (#5572). [Krzysztof Starecki]
- Support parameter names in pattern initialization (#5593) (#5596). [Greg Davill]
- Support randomize size constraints with restrictions (#5582 partial) (#5611). [Ryszard Rozak, Antmicro Ltd.]
- Support associative array basic constrained randomization (#5658) (#5670). [Yilou Wang]
- Support &96;default disable iff&96; and &96;$inferred_disable&96; (#4016). [Srinivasan Venkataramanan]
- Support &96;extern constraint&96; and &96;pure constraint&96;.
- Add &96;–no-std-waiver&96; and default reading of standard lint waivers file (#5607).
- Add &96;–no-std-package&96; as subset-alias of &96;–no-std&96; (#5607).
- Add &96;lint_off –contents&96; in configuration files (#5606).
- Add &96;–waiver-multiline&96; for context-sensitive &96;–waiver-output&96; (#5608).
- Add &96;–fno-inline-funcs&96; to disable function inlining.
- Add &96;–fno-slice&96; to disable array assignment slicing (#5644).
- Add error on illegal enum base type (#3010). [Iztok Jeras]
- Add error on &96;wait&96; with missing &96;.triggered&96; (#4457).
- Add error when improperly storing to parameter (#5147). [Gökçe Aydos]
- Add error on illegal &96;–prefix&96; etc. values (#5507). [Fabian Keßler]
- Add error on &96;–savable –timing&96; (#5690). [Narcis Rodas]
- Add coverage point hierarchy to coverage reports (#5575) (#5576). [Andrew Nolte]
- Add warning on global constraints (#5625). [Ryszard Rozak, Antmicro Ltd.]
- Add default CMAKE_BUILD_TYPE (#5691) (#5692). [Anthony Moore]
- Add error on &96;solve before&96; or soft constraints of &96;randc&96; variable.
- Improve concatenation performance (#5598) (#5599) (#5602). [Geza Lore]
- Improve optimization of duplicate wide expressions (#5637). [Bartłomiej Chmiel, Antmicro Ltd.]
- Fix dotted reference in delay value (#2410).
- Fix &96;function fork…join_none&96; regression with unknown type (#4449).

- Fix public_module requiring a wire to become public (#4916). [Andrew Nolte]
- Fix –hierarchical on projects with dot-f dependency lists (#5199) (#5669). [Bartłomiej Chmiel, Antmicro Ltd.]
- Fix can't locate scope error in interface task delayed assignment (#5462) (#5568). [Zhou Shen]
- Fix BLKANDNBLK for for VARXREFs (#5569). [Todd Strader]
- Fix VPI error instead of fatal for vpi_get_value() on large signals (#5571). [Todd Strader]
- Fix –output-groups leftover files issue (#5574). [Todd Strader]
- Fix slow unsized number parsing (#5577). [Geza Lore]
- Fix negative assignment pattern keys (#5580). [Iztok Jeras]
- Fix duplicate scope identifiers decoding (#5584). [Bartłomiej Chmiel, Antmicro Ltd.]
- Fix &96;rand&96; dynamic arrays with null handles (#5594). [Ryszard Rozak, Antmicro Ltd.]
- Fix NBAs to unpacked arrays of unpacked structs (#5603). [Geza Lore]
- Fix array of struct member overwrites on member update (#5605) (#5618) (#5628). [sumpster]
- Fix interface and struct pattern collision (#5639) (#5640). [Todd Strader]
- Fix mis-aliasing of instances with mailbox parameter types (#5632 partial).
- Fix error on duplicated declaration of gen block (#5663). [Ryszard Rozak, Antmicro Ltd.]
- Fix wildcard equality and inside operators for non-fourstate expressions (#5673). [Ryszard Rozak, Antmicro Ltd.]
- Fix &96;randomize..with&96; of parameterized classes (#5676). [Ryszard Rozak, Antmicro Ltd.]
- Fix interface bracketed array parameter access (#5677) (#5678). [Todd Strader]
- Fix width extension of operands of &96;inside&96; operator (#5685). [Ryszard Rozak, Antmicro Ltd.]
- Fix VPI + SYMRSVDWORD intersection (#5686). [Todd Strader]
- Fix verilator_gantt for hierarchically Verilated models (#5700). [Bartłomiej Chmiel, Antmicro Ltd.]

19.1.6 Verilator 5.030 2024-10-27

Major:

- Add &96;-output-groups&96; to build with concatenated .cpp files (#5257). [Mariusz Glebocki]
- Self-tests have been converted to Python, run &96;{test_name}.py&96; instead of &96;{test_name}.pl&96;.

Minor:

- Change .vlt config files to be read before .v files (#5185). [David Moberg]
- Change to use maximum for cover point aggregation (#5402). [Andrew Nolte]
- Change &96;–main&96; and &96;–binary&96; to use a TOP hierarchy name of "" (#5482).
- Change install of public executables into bindir instead of pkgdatadir (#5140) (#5544). [Geza Lore]
- Support IEEE-compliant intra-assign delays (#3711) (#5441). [Krzysztof Bieganski, Antmicro Ltd.]
- Support &96;wor&96;, &96;trior&96;, &96;wand&96;, &96;triand&96; (#5386) (#5496). [Zhou Shen]
- Support unconstrained randomization for unions (#5395) (#5396). [Yilou Wang]
- Support basic constrained queue randomization (#5413). [Arkadiusz Kozdra, Antmicro Ltd.]
- Support packed/unpacked and dynamic array unconstrained randomization (#5414) (#5415). [Yilou Wang]

- Support appending to queue via &96;[]&96; (#5421). [Krzysztof Bieganski, Antmicro Ltd.]
- Support named event locals (#5422). [Krzysztof Bieganski, Antmicro Ltd.]
- Support basic &96;dist&96; constraints (#5431). [Arkadiusz Kozdra, Antmicro Ltd.]
- Support unpacked array constrained randomization (#5437) (#5489). [Yilou Wang]
- Support inside array constraints (#5448). [Arkadiusz Kozdra, Antmicro Ltd.]
- Support DPI imports and exports with double underscores (#5481).
- Support ccache when compiling Verilated files with cmake.
- Support &96;local&96; and &96;protected&96; on &96;typedef&96; (#5460).
- Support unconstrained randomization for associative array and queue (#5515). [Yilou Wang]
- Support &96;rand&96; dynamic arrays of objects (#5557) (#5564). [Ryszard Rozak, Antmicro Ltd.]
- Add error on misused genvar (#408). [Alex Solomatnikov]
- Add error on instances without parenthesis.
- Add Docker pre-commit hook (#5238) (#5452). [Chris Bachhuber]
- Add partial coverage symbol and branch data in lcov info files (#5388). [Andrew Nolte]
- Add method to check if there are VPI callbacks of the given type (#5399). [Kaleb Barrett]
- Remove warning on unsized numbers exceeding 32-bits.
- Improve Verilation thread pool (#5161). [Bartłomiej Chmiel, Antmicro Ltd.]
- Improve performance of V3VariableOrder with parallelism (#5406). [Bartłomiej Chmiel, Antmicro Ltd.]
- Improve parser error handling (#5493). [Arkadiusz Kozdra, Antmicro Ltd.]
- Improve process trigger performance (#5483). [Geza Lore]
- Fix suppression of WIDTH* warnings when immediately under a size cast (#3417).
- Fix &96;$fatal&96; to not be affected by &96;+verilator+error+limit&96; (#5135). [Gökçe Aydos]
- Fix equivalence checking when replacing type parameters (#5213) (#5255). [Han Qi]
- Fix display with multiple string formats (#5311). [Luiza de Melo]
- Fix performance of V3Trace when many activity blocks (#5372). [Deniz Güzel]
- Fix REALCVT warning on integral timescale conversions (#5378). [Liam Braun]
- Fix multidimensional function return value selects (#5382). [Gökçe Aydos]
- Fix internal error in out-of-range select (#5393) (#5443). [Geza Lore]
- Fix dot fallback finding wrong symbols (#5394). [Arkadiusz Kozdra, Antmicro Ltd.]
- Fix infinite recursion due to recursive functions/tasks (#5398). [Krzysztof Bieganski, Antmicro Ltd.]
- Fix V3Randomize compile error on old GCC (#5403) (#5417). [Krzysztof Bieganski, Antmicro Ltd.]
- Fix extra events in traces (#5405).
- Fix empty &96;foreach&96; in &96;if&96; in constraints (#5408). [Krzysztof Bieganski, Antmicro Ltd.]
- Fix queue &96;[$-i]&96; select as reference argument (#5411). [Krzysztof Bieganski, Antmicro Ltd.]
- Fix &96;pre&96;/&96;post_randomize&96; on &96;randomize() with&96; (#5412). [Krzysztof Bieganski, Antmicro Ltd.]

- Fix capturing params in &96;randomize() with&96; (#5416) (#5418). [Krzysztof Bieganski, Antmicro Ltd.]
- Fix &96;sformatf&96; internal error on initial automatics (#5423). [Todd Strader]
- Fix clearing trigger of events with no sensitivity trees (#5426). [Arkadiusz Kozdra, Antmicro Ltd.]
- Fix driving clocking block in reactive region (#5430). [Krzysztof Bieganski, Antmicro Ltd.]
- Fix associative array next/prev/first/last mis-propagating constants (#5435). [Ethan Sifferman]
- Fix randomize treated as std::randomize in classes (#5436). [Arkadiusz Kozdra, Antmicro Ltd.]
- Fix &96;foreach&96; colliding index names (#5444). [Arkadiusz Kozdra, Antmicro Ltd.]
- Fix fault on defparam with UNSUPPORTED ignored (#5450). [Luiza de Melo]
- Fix class reference with pin that is a class reference (#5454).
- Fix not reporting class reference with extra parameters (#5467).
- Fix user-type parameter overlap (#5469). [Todd Strader]
- Fix tracing when name() is empty (#5470). [Sam Shahrestani]
- Fix timing mode not exiting on empty events (#5472).
- Fix coverage counts missing due to table optimization (#5473) (#5474). [Vito Gamberini]
- Fix &96;–binary&96; with .cpp PLI filenames under relative directory paths.
- Fix extra dot in coverage point hierarchy when using name()=".
- Fix short-circuiting with associative array access (#5484). [Ethan Sifferman]
- Fix short-circuiting on method calls (#5486). [Ethan Sifferman]
- Fix exponential concatenate performance (#5488). [Arkadiusz Kozdra, Antmicro Ltd.]
- Fix V3Table trying to generate 'x' bits in the lookup table (#5491). [Geza Lore]
- Fix randomize with foreach constraints (#5492). [Arkadiusz Kozdra, Antmicro Ltd.]
- Fix explicit CMAKE_INSTALL_PREFIX usages (#5500). [Fabian Keßler]
- Fix configure inserting absolute paths for Python and Perl (#5504) (#5505). [Nathan Graybeal]
- Fix pattern initialization with typedef key (#5512). [Eugene Feinberg]
- Fix &96;-j&96; option without argument in hierarchical Verilation (#5514). [Ryszard Rozak, Antmicro Ltd.]
- Fix &96;foreach&96; with 2-D queues and dynamic arrays (#5525) (#5529). [Yilou Wang]
- Fix struct array assignment (#5455) (#5537). [Yilou Wang]
- Fix copy constructor of classes that use std::process (#5528). [Ryszard Rozak, Antmicro Ltd.]
- Fix foreach on associative array (#5530). [Yilou Wang]
- Fix multi-range indices assignment (#5534) (#5547). [Yilou Wang]
- Fix static function wrappers (#5536). [Ryszard Rozak, Antmicro Ltd.]
- Fix assignments of concatenation to queues and dynamic arrays (#5540). [Ryszard Rozak, Antmicro Ltd.]
- Fix container reduction methods (#5542). [Krzysztof Boroński]
- Fix complex user type problem with &96;–x-assign&96; (#5543). [Todd Strader]
- Fix long module names crashing string handling (#5546). [Filip Badáň]
- Fix array trace splitting (#5549). [Todd Strader]

- Fix queue element access (#5551). [Ryszard Rozak, Antmicro Ltd.]
- Fix struct literal on pattern assignment (#5552) (#5559). [Todd Strader]
- Fix build on gcc when using the Spack wrapper (#5555). [Eric Müller]
- Fix enum name method (#5563). [Todd Strader]
- Fix &96;$countbits&96; in assert with non-tristates (#5566). [Shou-Li Hsu]
- Fix missing VlProcess handle in coroutines with splits (#5623) (#5650). [Bartłomiej Chmiel, Antmicro Ltd.]
- Fix imported array assignment literals (#5642) (#5648). [Todd Strader]
- Fix foreach mixed array (#5655) (#5656). [Yilou Wang]

19.1.7 Verilator 5.028 2024-08-21

Minor:

- Support state-dependent constraints (#5217). [Arkadiusz Kozdra, Antmicro Ltd.]
- Support cross-module clocking variable access (#5184). [Arkadiusz Kozdra, Antmicro Ltd.]
- Support inline constraints for class randomization methods (#5234). [Krzysztof Boroński]
- Support clocking blocks in virtual interfaces (#5235). [Arkadiusz Kozdra, Antmicro Ltd.]
- Support &96;$assertcontrol&96; assertion_type (#5236). [Bartłomiej Chmiel, Antmicro Ltd.]
- Support conditional constraints (#5245). [Arkadiusz Kozdra, Antmicro Ltd.]
- Support&96;–compiler-include&96; headers in user-supplied cpp files (#5271). [Bartłomiej Chmiel, Antmicro Ltd.]
- Support &96;rand_mode&96; (#5273). [Krzysztof Bieganski, Antmicro Ltd.]
- Support &96;this.randomize with&96; (#5282). [Arkadiusz Kozdra, Antmicro Ltd.]
- Support foreach constraints (#5302). [Arkadiusz Kozdra, Antmicro Ltd.]
- Support &96;parameter type&96; in hierarchical blocks (#5309) (#5333). [Bartłomiej Chmiel, Antmicro Ltd.]
- Support assertcontrol directive type (#5310). [Bartłomiej Chmiel, Antmicro Ltd.]
- Support inline random variable control (#5317). [Krzysztof Bieganski, Antmicro Ltd.]
- Support streaming operator on arrays and wide data (#5326). [Arkadiusz Kozdra, Antmicro Ltd.]
- Support streams to/from arrays of wide data (#5334). [Arkadiusz Kozdra, Antmicro Ltd.]
- Support &96;constraint_mode&96; (#5338). [Krzysztof Bieganski, Antmicro Ltd.]
- Support constraining AstSel (#5344). [Arkadiusz Kozdra, Antmicro Ltd.]
- Support default value on module input (#5358) (#5373). [Drew Ranck]
- Add &96;–compiler-include&96; for additional C++ includes (#5139) (#5202). [Bartłomiej Chmiel, Antmicro Ltd.]
- Add &96;–emit-accessors&96; (#5182) (#5227). [Ryan Ziegler]
- Add suggestions on misspelled PLI functions.
- Add warning on dist in constraints (#5264). [Arkadiusz Kozdra, Antmicro Ltd.]
- Add more &96;rand_mode&96; unsupported errors (#5329). [Krzysztof Bieganski, Antmicro Ltd.]
- Add parsing but otherwise ignore std::randomize (#5354). [Arkadiusz Kozdra, Antmicro Ltd.]

- Add Verilated cc define when &96;–timing&96; used (#5383). [Kaleb Barrett]
- Improve emitted code to use a reference for VlSelf (#5254). [Yangyu Chen]
- Fix monitor block sensitivity items (#4040) (#4400) (#5294). [Udaya Raj Subedi]
- Fix fusing macro arguments to not ignore whitespace (#5061). [Tudor Timi]
- Fix optimized-out sensitivity trees with &96;–timing&96; (#5080) (#5349). [Krzysztof Bieganski, Antmicro Ltd.]
- Fix classes/modules of case-similar names (#5109). [Arkadiusz Kozdra]
- Fix mis-removing $value$plusargs calls (#5127) (#5137). [Seth Pellegrino]
- Fix incorrect result of width mismatch (#5186) (#5189). [Yutetsu TAKATSUKASA]
- Fix compiler coroutine check (#5190) (#5300). [Ricardo Barbedo]
- Fix shortened module names when searching for files (#5196) (#5246). [Tim Hutt]
- Fix &96;–x-assign&96; to be independent from &96;+verilator+rand+reset&96; (#5214). [Andrew Nolte]
- Fix splitting if statements with impure conditions (#5219). [Bartłomiej Chmiel, Antmicro Ltd.]
- Fix unknown conversion on queues (#5220). [Alex Solomatnikov]
- Fix top-level unpacked structure resets (#5221).
- Fix concurrency for mailbox and semaphores (#5222). [Liam Braun]
- Fix forks capturing non-input ports in tasks (#5237) (#5343). [Krzysztof Bieganski, Antmicro Ltd.]
- Fix toggle coverage aggregation on same line (#5248). [Krzysztof Obłonczek]
- Fix error on empty generate with -O0 (#5250).
- Fix unconstrained randomization of unpacked structs (#5252). [Krzysztof Bieganski, Antmicro Ltd.]
- Fix inlining of variables driven from forced vars (#5259). [Geza Lore]
- Fix tracing with &96;–main-top-name -&96; (#5261). [Ethan Sifferman]
- Fix randomization when used with inheritance (#5268). [Krzysztof Bieganski, Antmicro Ltd.]
- Fix inline constraints creating class random generator (#5280). [Krzysztof Bieganski, Antmicro Ltd.]
- Fix WIDTHEXPAND on left shift of intuitive amount (#5284). [Greg Taylor]
- Fix elaborating foreach loops (#5285). [Arkadiusz Kozdra, Antmicro Ltd.]
- Fix initializing static array in dynamic arrays and queues (#5287). [Baruch Sterin]
- Fix static variable initializers in procedures (#5296). [Bartłomiej Chmiel, Antmicro Ltd.]
- Fix randomizing current object with &96;rand&96; class instance member (#5292). [Krzysztof Bieganski, Antmicro Ltd.]
- Fix handling of rand fields not referenced in constraints (#5305). [Ryszard Rozak, Antmicro Ltd.]
- Fix Python3 path discovery in make flows to avoid mixing system and user python interpreters (#5307). [Markus Krause]
- Fix make flows to pass PYTHON3 (like PERL) (#5307) (#5308). [Markus Krause]
- Fix assert on wide expression (#5319) (#5324). [Varun Koyyalagunta]
- Fix output clock variable overwriting signal (#5320) (#5347). [Krzysztof Bieganski, Antmicro Ltd.]
- Fix stringify in nested preprocessor macros (#5323). [Krzysztof Bieganski, Antmicro Ltd.]

- Fix $sformat with array arguments (#5330). [Abe Jordan]
- Fix -Wunused-but-set-variable clang warning (#5331). [Bartłomiej Chmiel, Antmicro Ltd.]
- Fix purity of functions with AstJumpBlock or AstStmtExpr (#5332). [Ryszard Rozak, Antmicro Ltd.]
- Fix compilation error on unreachable disable fork / wait fork (#5339). [Arkadiusz Kozdra, Antmicro Ltd.]
- Fix missing type coercion in 'inside {array}' (#5340). [Arkadiusz Kozdra, Antmicro Ltd.]
- Fix +: and -: unpacked array slicing when array has nonzero low index (#5345) (#5387). [James Bailey]
- Fix tracing_{on,off} in the presence of non-inlined modules (#5346). [Geza Lore]
- Fix NBAs in suspendables (#5348). [Krzysztof Bieganski, Antmicro Ltd.]
- Fix lint_off on Errors (#5351) (#5363). [Ethan Sifferman]
- Fix cache config file resolution performance (#5369). [Geza Lore]
- Fix capturing fields from superclass in &96;randomize() with&96; (#5389). [Krzysztof Bieganski, Antmicro Ltd.]
- Fix virtual interface null checks (#5391). [Krzysztof Bieganski, Antmicro Ltd.]
- Fix ==? and !=? with X values.
- Fix CPU time being zero.
- Fix inline function ref port persistence.

19.1.8 Verilator 5.026 2024-06-15

Major:

- Support constrained randomization with external solvers (#4947). [Arkadiusz Kozdra, Antmicro Ltd.]

Minor:

- Support &96;$psprintf&96; system function (#4314) (#5169). [Arkadiusz Kozdra, Antmicro Ltd.]
- Support 2D dynamic array initialization (#4700) (#5122). [Valentin Atepalikhin]
- Support __en/__out signals on top level inout ports (#4812) (#4856). [Paul Wright]
- Support empty queue as dynarray default value (#5055). [Arkadiusz Kozdra, Antmicro Ltd.]
- Support vpiInertialDelay (#5087). [Todd Strader]
- Support NBAs to arrays inside loops (#5092). [Geza Lore]
- Support parsing and otherwise ignoring inline constraints (#5126). [Arkadiusz Kozdra, Antmicro Ltd.]
- Support &96;inout&96; clocking items (#5160). [Arkadiusz Kozdra, Antmicro Ltd.]
- Support StructSel in unpacked array assignments (#5176). [Geza Lore]
- Add error on zero width select (#5028).
- Add CITATION.cff (#5057) (#5058). [Gijs Burghoorn]
- Add VPI eval needed tracking (#5065). [Todd Strader]
- Add &96;–localize-max-size&96; option and optimization (#5072).
- Add parameterless assert control system tasks (#5010). [Bartłomiej Chmiel]
- Add traceCapable indication to model header (#5053). [Vito Gamberini]
- Add increasing of stack size when possible (#5071) (#5104). [Yinan Xu]

- Add assertion on reusing VerilatedContext (#5167).
- Add &96;–pins-sc-uint-bool&96; to force SystemC uint type (#5192). [Bartłomiej Chmiel, Antmicro Ltd.]
- Improve DFG regularization in cyclic graphs (#5142). [Geza Lore]
- Improve VerilatedVpiPutHolder storage requirements (#5144). [Kaleb Barrett]
- Fix coroutines without awaits to have a co_return (#4208) (#5175). [Arkadiusz Kozdra, Antmicro Ltd.]
- Fix missing flex include path variable (#4970) (#4971). [Christopher Taylor]
- Fix missing parameters with comma to be errors (#4979) (#5012). [Paul Swirhun]
- Fix 'experimental/coroutine' file not found on MacOS (#5030) (#5031) (#5151). [Paul Bowen-Huggett]
- Fix bound queue printing (#5032). [Aleksander Kiryk, Antmicro Ltd.]
- Fix consecutive zero-delays (#5038). [Krzysztof Bieganski, Antmicro Ltd.]
- Fix attempted to destroy locked thread pool error (#5040). [Bartłomiej Chmiel, Antmicro Ltd.]
- Fix &96;$system&96; with string argument (#5042).
- Fix width extension on delays (#5043).
- Fix &96;$typename&96; on array.min and others (#5049). [Gökçe Aydos]
- Fix &96;make $(info)&96; which cannot be silenced (#5059). [Gökçe Aydos]
- Fix CMake builds to export VERILATOR_ROOT (#5063). [Michael Bikovitsky]
- Fix false ASSIGNIN on functions with explicit port map (#5069).
- Fix 4-state value support for $readmem (#5070) (#5078). [Ethan Sifferman]
- Fix DFG assertion with SystemC (#5076). [Geza Lore]
- Fix &96;$typename&96; string to be more standard (#5082) (#5083). [Andrew Nolte]
- Fix missed optimization in V3Delayed (#5089). [Geza Lore]
- Fix macro expansion in strings per 1800-2023 (#5094). [Geza Lore]
- Fix width extension of unpacked array select (#5095). [Varun Koyyalagunta]
- Fix MacOS missing <type_traits> header (#5096) (#5097). [Vito Gamberini]
- Fix assertion failure in V3Gate (#5101). [Yutetsu TAKATSUKASA]
- Fix aliases for forced port signals (#5105). [Geza Lore]
- Fix tracing interface functions (#5108). [Alex Solomatnikov]
- Fix method calls parsing in constraints (#5110). [Arkadiusz Kozdra, Antmicro Ltd.]
- Fix vpiInertialDelay for memories (#5113). [Todd Strader]
- Fix hierarchical compilation with nested -F (#5114) (#5124). [Alex Solomatnikov]
- Fix references to ports in forks (#5123). [Krzysztof Bieganski, Antmicro Ltd.]
- Fix output C++ type error on change detect of I/O arrays (#5125) (#5131). [Pawel Jewstafjew]
- Fix x-valued parameters with &96;–x-assign unique&96; (#5129). [Ethan Sifferman]
- Fix overflow of string on VPI reads (#5145) (#5146). [Kaleb Barrett]
- Fix VerilatedVpiPutHolder class (#5156). [Kaleb Barrett]
- Fix extending out-of-range select (#5159) (#5164). [Geza Lore]

- Fix radix in width warnings (#5166). [Geza Lore]
- Fix SystemC BITS_PER_DIGIT in VL_ASSIGN_SBW (#5170). [Bartłomiej Chmiel, Antmicro Ltd.]
- Fix non-constant replication in concats (#5171). [Arkadiusz Kozdra, Antmicro Ltd.]
- Fix table optimization when applied on real data type (#5172) (#5173). [Arthur Rosa]
- Fix signed types emitted in hierarchical Verilation (#5178). [Bartłomiej Chmiel, Antmicro Ltd.]
- Fix DPI import of null C-string (#5179).
- Fix CMake installation missing verilated.mk (#5187) (#5188). [Philip Axer]
- Fix linking with pthreads on CMake (#5194). [Tim Hutt]
- Fix clang-17 coroutines configuration with -std=gnu++20 (#5200). [Gus Smith]

19.1.9 Verilator 5.024 2024-04-05

Major:

- Add printing summary reports, use &96;–quiet&96; or &96;+verilator+quiet&96; to suppress (#4909).
- Support 1800-2023 keywords, and parsing with UNDEFINED warnings.
- Support 1800-2023 preprocessor ifdef expressions.

Minor:

- Change 1800-2023 to be default language version.
- Add DFG 'regularize' pass, and improve variable removal (#4937). [Geza Lore]
- Add error when pass net to function argument (#4132) (#4966). [Fuad Ismail]
- Add &96;UNUSEDLOOP&96; when unused loop is removed (#4926). [Bartłomiej Chmiel, Antmicro Ltd.]
- Add custom version for verilator –version packaging (#4954). [Nolan Poe]
- Add error on missing pure virtual functions (#4961).
- Add error on calling static function without object (#4962).
- Add JSON AST dumps (#5020). [Szymon Gizler]
- Support 1800-2023 DPI headers, svGetTime/svgGetTimeUnit/svGetTimePrecision methods.
- Support 1800-2023 class and function :initial, :extends, :final virtual overrides (#5025).
- Support public packed struct / union (#860) (#4878). [Kefa Chen]
- Support stream operation on unpacked array (#4714) (#5006). [Fuad Ismail]
- Support implicitly-typed variable definitions in for-loop initializers (#4945) (#4986). [Kevin Nygaard]
- Support inside range with implicit type conversion (#5026). [Arkadiusz Kozdra, Antmicro Ltd.]
- Improve installation to be relocatable (#4927). [Geza Lore]
- Improve internal ordering code (#4957) (#4990) (#4994) et al. [Geza Lore]
- Fix generate blocks in vpi_iterate (#3609) (#4913). [Andrew Nolte]
- Fix __Vlip undefined error in –freloop (#4824). [Justin Yao Du]
- Fix missing VPI scopes (#4918). [Andrew Nolte]
- Fix invalid cast on string structure creation (#4921).
- Fix try-lock spuriously fails (#4931) (#4938). [Kamil Rakoczy]

- Fix V3Unknown unpacked struct x-assign (#4934). [Yan Xu]
- Fix DFG removing forceable signals (#4942). [Geza Lore]
- Fix null characters in shortened identifiers (#4946). [Abdul Hameed]
- Fix assignment of null into struct member (#4952).
- Fix VPI missing scopes 2 (#4965). [Andrew Nolte]
- Fix object assignment from conditionals (#4968).
- Fix GCC14 warnings on template specialization syntax (#4974) (#4975). [Nolan Poe]
- Fix unpacked structure upper bit cleaning (#4978).
- Fix tests on MacOS (#4984) (#4985). [Kevin Nygaard]
- Fix &96;–prof-exec&96; predicted time values (#4988). [Geza Lore]
- Fix class type as an associative array parameter (#4997).
- Fix inout ports of unpacked struct type (#5000). [Ryszard Rozak, Antmicro Ltd.]
- Fix &96;unique {}&96; constraints missing semicolon (#5001).
- Fix preprocessor to respect strings in joins (#5007).
- Fix tracing class parameters (#5014).
- Fix memory leaks (#5016). [Geza Lore]
- Fix &96;$readmem&96; with missing newline (#5019). [Josse Van Delm]
- Fix internal error on missing pattern key (#5023).
- Fix tracing replicated hierarchical models (#5027).
- Fix false LIFETIME warning on &96;repeat&96; in &96;fork-join&96; (#5456).

19.1.10 Verilator 5.022 2024-02-24

Minor:

- Add predicted stack overflow warning (#4799).
- Add &96;+verilator+coverage+file&96; runtime option.
- Add &96;–assert-case&96; option (#4919). [Yutetsu TAKATSUKASA]
- Add &96;–decorations node&96; for inserting debug comments into emitted code.
- Add &96;–json-only&96; and related JSON dumping (#4715) (#4831). [Szymon Gizler, Antmicro Ltd.]
- Add &96;–[no]-stop-fail&96; option for continuing after assertions (#4904). [Yutetsu TAKATSUKASA]
- Add &96;–runtime-debug&96; for Verilated executable runtime debugging.
- Add &96;–valgrind&96; switch (#4828). [Szymon Gizler]
- Add &96;unroll_disable&96; and &96;unroll_full&96; loop control metacomments (#3260). [Jiaxun Yang]
- Remove deprecated 32-bit pointer mode (&96;gcc -m32&96;).
- Deprecate –xml-only and XML dumping (#4715) (#4831).
- Change zero replication width error to ZEROREPL warning (#4753) (#4762). [Pengcheng Xu]
- Improve message for priority case assertion failure (#4905). [Yutetsu TAKATSUKASA]
- Support dumping coverage with &96;–main&96;.

- Support dumping DFG patterns with &96;–stats&96; (#4889). [Geza Lore]
- Support &96;vpiConstType&96; in &96;vpi_get_str()&96; (#4797). [Marlon James]
- Support SystemC 3.0.0 public review version (#4805) (#4807). [Anthony Donlon]
- Support parsing anonymous primitive instantiations (#4809). [Anthony Donlon]
- Fix to not emit already waived warnings in waiver output (#4574) (#4818). [Jonathan Schröter]
- Fix &96;this&96; in member initialization (#4710). [eliasphanna]
- Fix localparam elaboration (#3858) (#4794). [Andrew Nolte]
- Fix lint_off disables on preprocessor warnings (#4703). [Srinivasan Venkataramanan]
- Fix $time not rounding up (#4790) (#4792). [Paul Wright]
- Fix &96;vpi_get()&96; and &96;vpi_get64()&96; to return vpiUndefined on errors (#4795). [Marlon James]
- Fix VPI parameter iteration (#4798). [Marlon James]
- Fix delays using wrong timeunit when modules inlined (#4806). [Paul Wright]
- Fix warnings in verilated_sc_trace.h for Clang (#4807) (#4827). [Anthony Donlon]
- Fix null pointer dereference (#4810) (#4825). [Adrian Sampson]
- Fix compilation error on multi-inherited interface class usage (#4819).
- Fix maybe-uninitialized compiler warning (#4820) (#4822). [Larry Doolittle]
- Fix mis-splitting of dump control functions (#4821). [Fan Shupei]
- Fix wrong utimes() parameter (#4829). [Szymon Gizler]
- Fix incorrect bit-op-tree NOT optimization (#4832) (#4847). [Yutetsu TAKATSUKASA]
- Fix width calculation in replaceShiftOp (#4837) (#4841) (#4849). [Yutetsu TAKATSUKASA]
- Fix unsafe write in wide array insertion (#4850) (#4855). [Paul Swirhun]
- Fix NOT when checking EQ/NEQ under AND/OR tree (#4857) (#4863). [Yutetsu TAKATSUKASA]
- Fix tracing chandles (#4860). [Nathan Graybeal]
- Fix $fwrite of null (#4862). [Jose Tejada]
- Fix -fno-const-bit-op-tree wrong runtime result (#4864) (#4867). [Yutetsu TAKATSUKASA]
- Fix SystemC biguint sign desynchronization (#4870). [Bartłomiej Chmiel]
- Fix incorrect temporary insertion in loop conditions with statements (#4873). [Geza Lore]
- Fix timing with expr on assign LHS (#4880). [Krzysztof Bieganski, Antmicro Ltd.]
- Fix assertion for unique case (#4892). [Yutetsu TAKATSUKASA]
- Fix GCC tautological-compare warnings.
- Fix compile error on structs with queues (and ignore toggle coverage on queues).
- Fix toggle coverage error on multi-edge driven signals.
- Fix whitespace in &96;pragma protect version&96; (#4902) (#4914). [Paul Swirhun]
- Fix incorrect code generation for change expression on typedefed unpacked array (#4915). [Geza Lore]
- Fix inconsistent driver resolution with typedefs (#4917). [Geza Lore]

19.1.11 Verilator 5.020 2024-01-01

Major:

- Support compilation with precompiled headers with Make, and GCC or CLang.
- Change include to systemc instead of systemc.h (#4622) (#4623). [Chih-Mao Chen] This may require that SystemC programs add 'using namespace sc_core', 'using namespace sc_dt'.

Minor:

- Add devcontainer support (#4748). [Stefan Wallentowitz]
- Support &96;iff&96; in sensitivity list (#1482) (#4626). [Krzysztof Bieganski, Antmicro Ltd.]
- Support parameterized virtual interfaces (#4047) (#4743). [Ryszard Rozak, Antmicro Ltd.]
- Support –timing triggers for virtual interfaces (#4673). [Krzysztof Bieganski, Antmicro Ltd.]
- Support ccache when compiling Verilator with CMake (#4678). [Anthony Donlon]
- Support passing constraints to –xml-only output (still otherwise unsupported) (#4683). [Shahid Ikram]
- Support node memory usage information in –stats (#4684). [Geza Lore]
- Support vpiConstType in vpi_get() (#4761). [Todd Strader]
- Support vpi_iterate on packages with vpiInstance (#4726). [Todd Strader]
- Support multiple parameters in virtual interfaces (#4745). [Ryszard Rozak, Antmicro Ltd.]
- Support user C/C++ code in final archive, and make a lib{model}.a (#4749) (#4754). [Fan Shupei]
- Support inside operator on unpacked arrays and queues (#4751). [Ryszard Rozak, Antmicro Ltd.]
- Support VPI parameter iteration (#4765). [Todd Strader]
- Support packages in vpi_handle_by_name() (#4768). [Todd Strader]
- Support invoking interface methods on virtual interface variables (#4774) (#4775). [Jordan McConnon]
- Remove deprecated options (#4663). [Geza Lore]
- Remove older compiler support; require C++14 or newer (#4784) (#4786).
- Optimize timing-delayed queue (#4584). [qrqiuren]
- Optimize substitute optimization memory usage (#4687). [Geza Lore]
- Optimize wide primitive operations with -Oz (#4733). [Geza Lore]
- Optimize V3Premit performance etc. (#4736). [Geza Lore]
- Fix VPI TOP level variable iteration (#3919) (#4618). [Marlon James]
- Fix display with no % printing assoc array (#4376). [Alex Solomatnikov]
- Fix scheduling of external force signals (#4577) (#4668). [Geza Lore]
- Fix a memory leak in V3Fork (#4628). [Krzysztof Boroński]
- Fix linking parameterized hierarchical blocks and recursive hierarchical blocks (#4654). [Anthony Donlon]
- Fix identifiers that end with '_' on Windows (#4655). [Anthony Donlon]
- Fix 'for' loop with outside variable reference (#4660). [David Harris]
- Fix tracing FST enums (#4661) (#4756). [Todd Strader]
- Fix interface parameters used in loop generate constructs (#4664) (#4665). [Anthony Donlon]

- Fix C++20 compilation errors (#4670).
- Fix deadlocks in error handler (#4672). [Mariusz Glebocki, Antmicro Ltd.]
- Fix MingW compilation (#4675). [David Ledger]
- Fix trace when using SystemC with certain configurations (#4676). [Anthony Donlon]
- Fix range access to classes depending on parameter resolution (#4681). [Krzysztof Boroński]
- Fix select into constant And/Or/Xor pattern (#4689). [Geza Lore]
- Fix access type of function arguments (#4692) (#4694). [Ryszard Rozak, Antmicro Ltd.]
- Fix dynamic NBAs with automatic vars (#4696). [Krzysztof Bieganski, Antmicro Ltd.]
- Fix # 0 delays for process resumption, etc. (#4697). [Krzysztof Boroński]
- Fix conflicted namespace for coroutines (#4701) (#4707). [Jinyan Xu]
- Fix compilers seeing empty input due to file system races (#4708). [Flavien Solt]
- Fix shift of > 32-bit number (#4719). [Flavien Solt]
- Fix Windows include gates in filesystem Flush implementation (#4720). [William D. Jones]
- Fix power operator with wide numbers and constants (#4721) (#4763). [Flavien Solt]
- Fix parameter passing to ports (#4723). [Ryszard Rozak, Antmicro Ltd.]
- Fix block names of nested do..while loops (#4728). [Ryszard Rozak, Antmicro Ltd.]
- Fix class name in error on 'new' on virtual class (#4739). [Ryszard Rozak, Antmicro Ltd.]
- Fix typedefs pointing to parameterized classes (#4747). [Ryszard Rozak, Antmicro Ltd.]
- Fix $finish twice to no longer exit (#4757). [Tim Hutt]
- Fix dynamic NBA conditions (#4773). [Krzysztof Bieganski, Antmicro Ltd.]
- Fix &96;V3Fork&96; stage to run only if &96;–timing&96; is set (#4778). [Krzysztof Bieganski, Antmicro Ltd.]
- Fix max multiply width and add runtime assertions if too small (#4781).
- Fix select value too wide (#5148) (#5153). [Dercury]

19.1.12 Verilator 5.018 2023-10-30

Major:

- Support compilation with precompiled headers with Make and GCC or CLang.
- Change include of systemc instead of systemc.h (#4622) (#4623). [Chih-Mao Chen] This may require that SystemC programs add 'using namespace sc_core', 'using namespace sc_dt'.

Minor:

- Add SIDEEFFECT warning on mishandled side effect cases.
- Add trace() API even when Verilated without –trace (#4462). [phelter]
- Add warning on interface instantiation without parens (#4094). [Gökçe Aydos]
- Add sv_vpi_user.h from IEEE 1800-2017 Annex M (#4606). [Marlon James]
- Support 'disable fork' (#4125) (#4569). [Aleksander Kiryk, Antmicro Ltd.]
- Support 'wait fork' (#4586). [Aleksander Kiryk, Antmicro Ltd.]

- Support 'randc' (#4349).
- Support assigning events (#4403). [Krzysztof Boroński]
- Support resizing function call inout arguments (#4467).
- Support NBAs in non-inlined functions/tasks (#4496) (#4572). [Krzysztof Bieganski, Antmicro Ltd.]
- Support converting parameters inside modules to localparams (#4511). [Anthony Donlon]
- Support concatenation of unpacked arrays (#4558). [Yutetsu TAKATSUKASA]
- Support Clang 16 (#4592). [Mariusz Glebocki]
- Support VPI variables of real and string data types (#4594). [Marlon James]
- Support making VL_LOCK_SPINS configurable (#4599). [Geza Lore]
- Change code –stats output (#4597). [Geza Lore]
- Change –prof-exec infrastructure and report (#4602). [Geza Lore]
- Change lint_off to not propagate upwards to files including where the lint_off is.
- Optimize empty expression statements (#4544).
- Optimize trace internals (#4610) (#4612). [Geza Lore]
- Optimize internal performance issues (#4638). [Geza Lore]
- Fix conversion of impure logical expressions to bit expressions (#487 partial) (#4437). [Ryszard Rozak, Antmicro Ltd.]
- Fix enum functions in localparams (#3999). [Andrew Nolte]
- Fix passing arguments by reference (#3385 partial) (#4489). [Ryszard Rozak, Antmicro Ltd.]
- Fix multithreading handling to separate by code units that use/never use it (#4228). [Mariusz Glebocki, Antmicro Ltd.]
- Fix usage of annotation options (#4486) (#4504). [Michal Czyz]
- Fix detecting local vars in nested forks (#4493) (#4506). [Kamil Rakoczy]
- Fix handling input file path separator (#4515) (#4516). [Anthony Donlon]
- Fix mis-support for parameterized UDPs (#4518). [Anthony Donlon]
- Fix constant conversion of $realtobits, $bitstoreal (#4522). [Andrew Nolte]
- Fix conversion of integers in $display '%e' (#4528). [muzafferkal]
- Fix non-inlined interface tracing (#3984) (#4530). [Todd Strader]
- Fix stream operations with operands of struct type (#4531) (#4532). [Ryszard Rozak, Antmicro Ltd.]
- Fix 'this' in a constructor (#4533). [Ryszard Rozak, Antmicro Ltd.]
- Fix stream shift operator of 32 bits (#4536). [Julien Faucher]
- Fix object destruction after a copy constructor (#4540) (#4541). [Ryszard Rozak, Antmicro Ltd.]
- Fix inlining of real functions miscasting (#4543). [Andrew Nolte]
- Fix broken link error for enum references (#4551). [Anthony Donlon]
- Fix logical expressions with class objects - caching in v3Const (#4552). [Ryszard Rozak, Antmicro Ltd.]
- Fix using functions/tasks following class definition inside module (#4553). [Anthony Donlon]
- Fix large constant buffer overflow (#4556). [Varun Koyyalagunta]

- Fix instance arrays connecting to array of structs (#4557). [raphmaster]
- Fix error message for invalid parameter overrides (#4559). [Anthony Donlon]
- Fix shift to remove operation side effects (#4563).
- Fix compile warning on unused member function variable (#4567).
- Fix method narrowing conversion compiler error (#4568).
- Fix interface comparison (#4570). [Krzysztof Bieganski, Antmicro Ltd.]
- Fix dynamic triggers for named events (#4571). [Krzysztof Bieganski, Antmicro Ltd.]
- Fix dictionaries with keys of class types (#4576). [Ryszard Rozak, Antmicro Ltd.]
- Fix to not remap local assign intervals in forks (#4583). [Krzysztof Bieganski, Antmicro Ltd.]
- Fix display optimization ignoring side effects (#4585).
- Fix PLI/DPI user defined system task/function grammar (#4587) (#4588). [Quentin Corradi]
- Fix fault on empty clocking block (#4593). [Alex Mykyta]
- Fix creating implicit nets for inputs of gate primitives (#4603). [Geza Lore]
- Fix try_put method of unbounded mailbox (#4608). [Ryszard Rozak, Antmicro Ltd.]
- Fix stable name generation in V3Fork (#4615) (#4624). [Krzysztof Boroński]
- Fix virtual methods (#4616). [Ryszard Rozak, Antmicro Ltd.]
- Fix insertion at queue end (#4619). [Krzysztof Boroński]
- Fix rand fields of reference types (#4627). [Ryszard Rozak, Antmicro Ltd.]
- Fix dynamic casts of null values (#4631). [Ryszard Rozak, Antmicro Ltd.]
- Fix signals read via virtual interfaces being misoptimized (#4645). [Krzysztof Bieganski, Antmicro Ltd.]
- Fix handling of static keyword in methods (#4649). [Ryszard Rozak, Antmicro Ltd.]
- Fix preprocessor to show &96;line 2 on resumed file.

19.1.13 Verilator 5.016 2023-09-16

Minor:

- Add prepareClone and atClone APIs for Verilated models (#3503) (#4444). [Yinan Xu]
- Add check for conflicting options e.g. binary and lint-only (#4409). [Ethan Sifferman]
- Add –no-trace-top to not trace top signals (#4412) (#4422). [Frans Skarman]
- Support recursive function calls (#3267).
- Support assignments of packed values to stream expressions on queues (#4401). [Ryszard Rozak, Antmicro Ltd]
- Support no-parentheses calls to static methods (#4432). [Krzysztof Boroński]
- Support block_item_declaration in forks (#4455). [Krzysztof Boroński]
- Support assignments of stream expressions on queues to packed values (#4458). [Ryszard Rozak, Antmicro Ltd]
- Support function non-constant default arguments (#4470).
- Support ‘let’.
- Optimize Verilator executable size by refactoring error reporting routines (#4446). [Anthony Donlon]
- Optimize Verilation runtime pointers and graphs (#4396) (#4397) (#4398). [Krzysztof Bieganski, Antmicro Ltd]

- Optimize preparations towards multithreaded Verilation (#4291) (#4463) (#4476) (#4477) (#4479). [Kamil Rakoczy, Antmicro Ltd]
- Fix Windows filename format, etc (#3873) (#4421). [Anthony Donlon].
- Fix t_dist_cppstyle Perl performance issue (#4085). [Srinivasan Venkataramanan]
- Fix using type in parameterized classes without #() (#4281) (#4440). [Anthony Donlon]
- Fix false INFINITELOOP on forever..mailbox.get() (#4323). [Srinivasan Venkataramanan]
- Fix data type of condition operation on class objects (#4345) (#4352). [Ryszard Rozak, Antmicro Ltd]
- Fix variables mutated under fork..join_none/join_any blocks into anonymous objects (#4356). [Krzysztof Boroński]
- Fix V3CUse, do not consider implementations (.cpp) at all (#4386). [Krzysztof Boroński]
- Fix ++/– under statements (#4399). [Aleksander Kiryk, Antmicro Ltd]
- Fix detection of mixed blocking and nonblocking assignment in nested assignments (#4404). [Ryszard Rozak, Antmicro Ltd]
- Fix jumping over object initialization (#4411). [Krzysztof Boroński]
- Fix multiple issues towards short circuit support (#4413) (#4460). [Ryszard Rozak, Antmicro Ltd]
- Fix variable lifetimes in extern methods (#4414). [Krzysztof Boroński]
- Fix multiple function definitions in V3Sched (#4416). [Hennadii Chernyshchyk]
- Fix false UNUSEDPARAM on generate localparam (#4427). [Bill Pringlemeir]
- Fix checking for parameter and port connections in the wrong place (#4428). [Anthony Donlon]
- Fix coroutine handle movement during queue manipulation (#4431). [Aleksander Kiryk, Antmicro Ltd]
- Fix nested assignments on the LHS (#4435). [Ryszard Rozak, Antmicro Ltd]
- Fix false MULTITOP on bound interfaces (#4438). [Alex Solomatnikov]
- Fix internal error on real conversion (#4447). [vdhotre-ventana]
- Fix lifetime unknown error on enum.name (#4448). [Johan Wouters]
- Fix unstable output of VHashSha256 (#4453). [Anthony Donlon]
- Fix static cast from a stream type (#4469) (#4485). [Ryszard Rozak, Antmicro Ltd]
- Fix error on enum with VARHIDDEN of cell (#4482). [Michail Rontionov]
- Fix lint of case statements with enum and wildcard bits (#4464) (#4487). [Anthony Donlon]
- Fix reference to extended class in parameterized class (#4466).
- Fix recursive display causing segfault (#4480). [Kuoping Hsu]
- Fix the error message when the type of ref argument is wrong (#4490). [Ryszard Rozak, Antmicro Ltd]
- Fix display %x formatting of real.
- Fix mis-warning on #() in classes' own functions.
- Fix IGNOREDRETURN to not warn on void-cast static function calls.
- Fix ZERODLY to not warn on 'wait(0)'.

19.1.14 Verilator 5.014 2023-08-06

Minor:

- Deprecation planned for 32-bit pointer -m32 mode (#4268).
- Deprecate CMake config below version 3.13 (#4389) (#4390). [Vito Gamberini]
- Support some stream operations on queues (#4292). [Ryszard Rozak, Antmicro Ltd]
- Support property declaration with empty parentheses (#4313) (#4317). [Anthony Donlon]
- Support locator methods with "with" on assoc arrays (#4335). [Ryszard Rozak, Antmicro Ltd]
- Support string replication with variable (#4341). [Aleksander Kiryk, Antmicro Ltd]
- Support more types in wait (#4374). [Aleksander Kiryk, Antmicro Ltd]
- Support static method calls as default values of function arguments (#4378). [Ryszard Rozak, Antmicro Ltd]
- Add GENUNNAMED lint warning. [Srinivasan Venkataramanan, Deepa Palaniappan]
- Add MISINDENT lint warning for misleading indentation.
- Fix 'VlForkSync' redeclaration (#4277). [Krzysztof Bieganski, Antmicro Ltd]
- Fix processes that can outlive their parents (#4253). [Krzysztof Boronski, Antmicro Ltd]
- Fix duplicate fork names (#4295). [Ryszard Rozak, Antmicro Ltd]
- Fix splitting coroutines (#4297) (#4307). [Jiamin Zhu]
- Fix error when multiple duplicate DPI exports (#4301).
- Fix class reference assignment checking (#4296). [Ryszard Rozak, Antmicro Ltd]
- Fix handling of ref types in initial values of type parameters (#4304). [Ryszard Rozak, Antmicro Ltd]
- Fix comparison of string parameters (#4308). [Ryszard Rozak, Antmicro Ltd]
- Fix state update for always processes (#4311). [Aleksander Kiryk, Antmicro Ltd]
- Fix multiple edge timing controls in class methods (#4318) (#4320) (#4344). [Krzysztof Bieganski, Antmicro Ltd]
- Fix implicit calls of base class constructors with optional arguments (#4319). [Ryszard Rozak, Antmicro Ltd]
- Fix propagation of process requirement (#4321). [Krzysztof Boroński]
- Fix unhandled overloads in V3InstrCount (#4324). [Krzysztof Boroński]
- Fix selects of static members (#4326). [Ryszard Rozak, Antmicro Ltd]
- Fix references to members of results of static methods (#4327). [Ryszard Rozak, Antmicro Ltd]
- Fix unique..with method on queues of class objects (#4328). [Ryszard Rozak, Antmicro Ltd]
- Fix queue slicing (#4329). [Aleksander Kiryk, Antmicro Ltd]
- Fix wildcard referring types (#4336) (#4342). [Aleksander Kiryk, Antmicro Ltd]
- Fix comparison of class objects (#4346). [Ryszard Rozak, Antmicro Ltd]
- Fix unexpected RefDType on assoc arrays (#4337). [Aleksander Kiryk, Antmicro Ltd]
- Fix cmake astgen for Rocky Linux 8.7 (#4343). [Julian Daube]
- Fix class timescale in class packages (#4348). [Krzysztof Bieganski, Antmicro Ltd]
- Fix string concatenations (#4354). [Ryszard Rozak, Antmicro Ltd]

- Fix unlinked task error from broken context (#4355) (#4402). [Aleksander Kiryk, Antmicro Ltd]
- Fix selects on unpacked structs (#4359). [Ryszard Rozak, Antmicro Ltd]
- Fix select operation on assoc array with wide keys (#4360). [Ryszard Rozak, Antmicro Ltd]
- Fix non-public methods with wide output (#4364). [Ryszard Rozak, Antmicro Ltd]
- Fix handling of super.new calls (#4366). [Ryszard Rozak, Antmicro Ltd]
- Fix assign to input var in methods (#4367). [Aleksander Kiryk, Antmicro Ltd]
- Fix VlProcess not found (#4368). [Aleksander Kiryk, Antmicro Ltd]
- Fix order of evaluation of function calls in statements (#4375). [Ryszard Rozak, Antmicro Ltd]
- Fix config_build.h issues (#4380) (#4381). [Andrew Miloradovsky]

19.1.15 Verilator 5.012 2023-06-13

Major:

- With -j or –build-jobs, multithread Verilator's emit phase of Verilation. [Kamil Rakoczy, Antmicro Ltd] Additional Verilator-internal stages will become multithreaded over time.

Minor:

- Add –main-top-name option for C main TOP name (#4235) (#4249). [Don Williamson]
- Add creating __inputs.vpp file with –debug (#4177). [Tudor Timi]
- Add NEWERSTD warning when using feature in newer language standard (#4168) (#4172). [Ethan Sifferman]
- Add warning that timing controls in DPI exports are unsupported (#4238). [Krzysztof Bieganski, Antmicro Ltd]
- Support std::process class (#4212). [Aleksander Kiryk, Antmicro Ltd]
- Support inside expressions with strings and doubles (#4138) (#4139). [Krzysztof Boroński]
- Support get_randstate/set_randstate class method functions.
- Support for condition operator on class objects (#4214). [Ryszard Rozak, Antmicro Ltd]
- Support array max (#4275). [Aleksander Kiryk, Antmicro Ltd]
- Optimize VPI callValueCbs (#4155). [Hennadii Chernyshchyk]
- Configure for faster C++ linking using 'mold', if it is installed.
- Fix crash on duplicate imported modules (#3231). [Robert Balas]
- Fix false WIDTHEXPAND on array declarations (#3959). [Jose Tejada]
- Fix marking overridden methods as coroutines (#4120) (#4169). [Krzysztof Bieganski, Antmicro Ltd]
- Fix SystemC signal copy macro use (#4135). [Josep Sans]
- Fix duplicate static names in blocks in functions (#4144) (#4160). [Stefan Wallentowitz]
- Fix initialization order of initial static after function/task (#4159). [Kamil Rakoczy, Antmicro Ltd]
- Fix linking AstRefDType if it has parameterized class ref (#4164) (#4170). [Ryszard Rozak, Antmicro Ltd]
- Fix crash caused by $display() optimization (#4165) (#4166). [Tudor Timi]
- Fix arrays of unpacked structs (#4173). [Risto Pejašinović]
- Fix $fscanf of decimals overflowing variables (#4174). [Ahmed El-Mahmoudy]
- Fix super.new missing data type (#4147). [Tudor Timi]

- Fix missing class forward declarations (#4151). [Krzysztof Boroński]
- Fix hashes of instances of parameterized classes (#4182). [Ryszard Rozak, Antmicro Ltd]
- Fix forced assignments that override non-continuous assignments (#4183) (#4192). [Krzysztof Bieganski, Antmicro Ltd]
- Fix wide structure VL_TOSTRING_W generation (#4188) (#4189). [Aylon Chaim Porat]
- Fix references to members of parameterized base classes (#4196). [Ryszard Rozak, Antmicro Ltd]
- Fix tracing undefined alignment (#4201) (#4288) [John Wehle]
- Fix class-specific same methods for AstVarScope, AstVar, and AstScope (#4203) (#4250). [John Wehle]
- Fix dotted references in parameterized classes (#4206). [Ryszard Rozak, Antmicro Ltd]
- Fix bit selections under parameterized classes (#4210). [Ryszard Rozak, Antmicro Ltd]
- Fix duplicate std:: declaration with -I (#4215). [Harald Pretl]
- Fix deep traversal of class inheritance timing (#4216). [Krzysztof Boroński]
- Fix class parameters of enum types (#4219). [Ryszard Rozak, Antmicro Ltd]
- Fix static methods with prototypes (#4220). [Ryszard Rozak, Antmicro Ltd]
- Fix LATCH warning on function local variables (#4221) (#4284) [Julien Margetts]
- Fix VCD scope types (#4227) (#4282). [Àlex Torregrosa]
- Fix incorrect multi-driven lint warning (#4231) (#4248). [Adrien Le Masle]
- Fix missing assignment for wide unpacked structs (#4233). [Jiamin Zhu]
- Fix unpacked struct == and != operators (#4234) (#4240). [Risto Pejašinović]
- Fix AstStructSel clean when data type is structure (#4241) (#4244). [Risto Pejašinović]
- Fix function calls in with statements (#4245). [Ryszard Rozak, Antmicro Ltd]
- Fix operator == for unpacked struct, if elements are VlUnpacked arrays (#4247). [Risto Pejašinović]
- Fix STATIC lifetime for variables created from clocking items (#4262). [Krzysztof Boroński]
- Fix names of foreach blocks (#4264). [Ryszard Rozak, Antmicro Ltd]
- Fix iterated variables in foreach loops to have VAUTOM lifetimes (#4265). [Krzysztof Boroński]
- Fix missing assignment for wide class members (#4267). [Jiamin Zhu]
- Fix the global uses timing flag when forks exist (#4274). [Krzysztof Bieganski, Antmicro Ltd]
- Fix struct redefinition (#4276). [Aleksander Kiryk, Antmicro Ltd]
- Fix detection of wire/reg duplicates.
- Fix false IMPLICITSTATIC on package functions.
- Fix method calls on function return values.

19.1.16 Verilator 5.010 2023-04-30

Minor:

- Add –public-depth to force public to a certain instance depth (#3952). [Andrew Nolte]
- Add –public-params flag (#3990). [Andrew Nolte]
- Add CONSTRAINTIGN warning when constraint ignored.

- Add STATICVAR warning and convert to automatic (#4018) (#4027) (#4030). [Ryszard Rozak, Antmicro Ltd]
- Add error if class types don't match (#4064). [Ryszard Rozak, Antmicro Ltd]
- Support class extends of package::class.
- Support class srandom and class random stability.
- Support class method calls without parenthesis (#3902) (#4082). [Srinivasan Venkataramanan]
- Support method calls without parenthesis (#4034). [Ryszard Rozak, Antmicro Ltd]
- Support parameterized return types of methods (#4122). [Ryszard Rozak, Antmicro Ltd]
- Support parameterized class references in extends statement (#4146). [Ryszard Rozak, Antmicro Ltd]
- Support complicated IEEE 'for' assignments.
- Support $fopen as an expression.
- Support ++/– on dotted member variables.
- Optimize static trigger evaluation (#4142). [Geza Lore, X-EPIC]
- Optimize more xor trees (#4071). [Yutetsu TAKATSUKASA]
- Change range order warning from LITENDIAN to ASCRANGE (#4010). [Iztok Jeras]
- Change ZERODLY to a warning.
- Fix random internal crashes (#666). [Dag Lem]
- Fix install, standardization in cmake CMakeLists.txt (#3974). [Yu-Sheng Lin]
- Fix UNDRIVEN warning seg fault (#3989). [Felix Neumärker]
- Fix symbol entries when inheriting classes (#3995) (#3996). [Krzysztof Boroński]
- Fix event controls reusing same variable (#4014). Kamil Rakoczy <krakoczy@antmicro.com>
- Fix push to dynamic queue in struct (#4015). [ezchi]
- Fix names for blocks in do..while loop (#4019). [Ryszard Rozak, Antmicro Ltd]
- Fix randomize on null field (#4023). [Ryszard Rozak, Antmicro Ltd]
- Fix rand fields in base classes (#4025). [Ryszard Rozak, Antmicro Ltd]
- Fix large return blocks with –comp-limit-blocks (#4028). [tenghtt]
- Fix clocking block scope internal error (#4032). [Srinivasan Venkataramanan]
- Fix false LATCH warning on –assert 'unique else if' (#4033) ($4054). [Jesse Taube]
- Fix characters from DEFENV literals for Conda (#4035) (#4044). [Tim Snyder]
- Fix info message prints under –assert (#4036) (#4053). [Srinivasan Venkataramanan]
- Fix C++ compile errors when passing class refs as task argument (#4063). [Krzysztof Bieganski, Antmicro Ltd]
- Fix NBAs inside fork-joins (#4050). [Aleksander Kiryk, Antmicro Ltd]
- Fix task calls as fork statements (#4055). [Krzysztof Bieganski, Antmicro Ltd]
- Fix _Vilp used before declaration (#4057) (#4062). [Josep Sans]
- Fix incorrect optimization of bit op tree (#4059) (#4070). [Yutetsu TAKATSUKASA]
- Fix parameters in a class body to be localparam (#4061). [Ryszard Rozak, Antmicro Ltd]
- Fix interface generate begin (#4065). [Srinivasan Venkataramanan]

- Fix tracing with awaits at end of block (#4075) (#4076). [Krzysztof Bieganski, Antmicro Ltd]
- Fix sense expression variable naming (#4081). [Kamil Rakoczy]
- Fix importing symbols from base class (#4084). [Ryszard Rozak, Antmicro Ltd]
- Fix false error on new const assignment (#4098). [Tudor Timi]
- Fix unpacked structs under classes (#4102). [Tudor Timi]
- Fix variables in class methods to be automatic (#4111) (#4137). [Peter Monsson]
- Fix to use parallel build for projects with a lot of files (#4116). [Krzysztof Boroński]
- Fix including __Syms header in generated C++ files (#4123). [Krzysztof Boroński]
- Fix systemc namespace issues (#4126) (#4127). [Eyck Jentzsch]
- Fix class param extends A=B (#4128). [Ryszard Rozak, Antmicro Ltd]
- Fix missing begin block hierarchy in –xml-only cells section (#4129) (#4133). [Risto Pejašinović]
- Fix resolution of class lvalues after parameterization (#4131). [Krzysztof Boroński]
- Fix DFG error on $countbits (#4101) (#4143). [Paul Donahue]
- Fix duplicating parameter class types (#4115). [Ryszard Rozak, Antmicro Ltd]
- Fix class extend param references (#4136). [Ryszard Rozak, Antmicro Ltd]
- Fix -CFLAGS to allow overriding optimization levels (#4140). [Peter Monsson]
- Fix DPI function type alias (#4148) (#4149). [Toru Niina]
- Fix deleting unused parameterized classes (#4150). [Ryszard Rozak, Antmicro Ltd]
- Fix false ENUMVALUE on expressions and arrays.
- Fix unnecessary verilated_std.sv waivers in –waiver-output.

19.1.17 Verilator 5.008 2023-03-04

Minor:

- Add –annotate-points option, change multipoint on line reporting (#3876). [Nassim Corteggiani]
- Add –verilate-jobs option (#3889). [Kamil Rakoczy, Antmicro Ltd]
- Add WIDTHEXPAND and WIDTHTRUNC warnings to replace WIDTH (#3900). [Andrew Nolte]
- Add SOURCE_DATE_EPOCH for docs/guide/conf.py (#3918). [Larry Doolittle]
- Add */verilator public[flat|flat_rd|flat_rw|]/* metacomments (#3894). [Joseph Nwabueze]
- Add lint warning on always_comb multidriven (#3888) (#3939). [Adam Bagley]
- Add warning on ++/– over expressions with potential side effects (#3976). [Krzysztof Boroński]
- Add error on mixing .name and by-port instantiations.
- Removed deprecated –cdc option.
- Support unpacked unions.
- Support interface classes and class implements.
- Support global clocking and $global_clock.
- Support class parameters without initial values.
- Support cast to numbers from strings.

- Support struct I/O in –lib-create (#3378) (#3892). [Varun Koyyalagunta]
- Support function calls without parenthesis (#3903) (#3902). [Ryszard Rozak, Antmicro Ltd]
- Support class extending its parameter (#3904). [Ryszard Rozak, Antmicro Ltd]
- Support static function variables (#3830). [Ryszard Rozak, Antmicro Ltd]
- Support recursive methods (#3987). [Ryszard Rozak, Antmicro Ltd]
- Fix real parameters of infinity and NaN.
- Fix pattern assignment to unpacked structs (#3510). [Mostafa Garnal]
- Fix single-element replication to dynarray/unpacked/queue (#3548). [Gustav Svensk]
- Fix constant enum methods (#3621). [Todd Strader]
- Fix inconsistent naming of generate scope arrays (#3840). [Andrew Nolte]
- Fix namespace fallback resolution (#3863) (#3942). [Aleksander Kiryk, Antmicro Ltd]
- Fix std:: to be parsed first (#3864) (#3928). [Aleksander Kiryk, Antmicro Ltd]
- Fix cmake warning if multiple SOURCES w/o PREFIX (#3916) (#3927). [Yoda Lee]
- Fix parameterized class function linkage (#3917). [Ryszard Rozak]
- Fix static members of type aliases of a parameterized class (#3922). [Ryszard Rozak, Antmicro Ltd]
- Fix class extend parameter dot case (#3926). [Ryszard Rozak, Antmicro Ltd]
- Fix MsWin missing directory exception, and ::std (#3928) (#3933) (#3935). [Kritik Bhimani]
- Fix very long VPI signal names (#3929). [Marlon James]
- Fix VPI upper interface scopes not found (#3937). [David Stanford]
- Fix virus detection false positive (#3944). [Stuart Morris]
- Fix constant string function assignment (#3945). [Todd Strader]
- Fix constant format field widths (#3946). [Todd Strader]
- Fix class field linking when a super classes is a param (#3949). [Ryszard Rozak, Antmicro Ltd]
- Fix CMake bad C identifiers (#3948) (#3951). [Zixi Li]
- Fix build on HP PA architecture (#3954). [John David Anglin]
- Fix date on the front page of verilator.pdf (#3956) (#3957). [Larry Doolittle]
- Fix associative arrays declared with ref type (#3960). [Ryszard Rozak, Antmicro Ltd]
- Fix missing error on negative replicate (#3963). [Benjamin Menküc]
- Fix self references to parameterized classes (#3962). [Ryszard Rozak, Antmicro Ltd]
- Fix LITENDIAN warning is backwards (#3966) (#3967). [Cameron Kirk]
- Fix subsequent parameter declarations (#3969). [Ryszard Rozak, Antmicro Ltd]
- Fix timing delays to not truncate below 64 bits (#3973) (#3982). [Felix Neumärker]
- Fix cmake on MacOS to mark weak symbols with -U linker flag (#3978) (#3979). [Peter Debacker]
- Fix UNDRIVEN warning seg fault (#3989). [Felix Neumärker]
- Fix coverage of class methods (#3998). [Tim Paine]
- Fix packed array structure replication.

- Fix enum.next(0) and enum.prev(0).

19.1.18 Verilator 5.006 2023-01-22

Minor:

- Support clocking blocks (#3674). [Krzysztof Bieganski, Antmicro Ltd]
- Support unpacked structs (#3802). [Aleksander Kiryk, Antmicro Ltd]
- Support Windows-native builds using cmake (#3814). [Kritik Bhimani]
- Support p format for UnpackArray (#3877). [Aleksander Kiryk, Antmicro Ltd]
- Support property calls without parenthesis (#3879) (#3893). [Ryszard Rozak, Antmicro Ltd]
- Support import/export lists in modport (#3886). [Gökçe Aydos]
- Support class queue equality (#3895). [Ilya Barkov]
- Support type case and type equality comparisons.
- Add IMPLICITSTATIC warning when a task/function is implicitly static (#3839). [Ryszard Rozak, Antmicro Ltd]
- Add VL_VALUE_STRING_MAX_WORDS override (#3869). [Andrew Nolte]
- Optimize expansion of extend operators.
- Internal multithreading tests. [Mariusz Glebocki, et al, Antmicro Ltd]
- Fix VPI one-time timed callbacks (#2778). [Marlon James, et al]
- Fix initiation of function variables (#3815). [Dan Gisselquist]
- Fix to zero possibly uninitialized bits in replications (#3815).
- Fix crash in DFT due to width use after free (#3817) (#3820). [Jevin Sweval]
- Fix signed/unsigned comparison compile warning (#3822). [Kamil Rakoczy]
- Fix OS-X weak symbols with -U linker flag (#3823). [Jevin Sweval]
- Fix wrong bit op tree optimization (#3824) (#3825). [Yutetsu TAKATSUKASA]
- Fix self references when param class instantiated (#3833). [Ryszard Rozak, Antmicro Ltd]
- Fix memory leak in V3Sched, etc. (#3834). [Geza Lore]
- Fix compatibility with musl libc / Alpine Linux (#3845). [Sören Tempel]
- Fix empty case items crash (#3851). [Rich Porter]
- Fix VL_CPU_RELAX on MIPS/Armel/s390/sparc (#3843) (#3891). [Kamil Rakoczy]
- Fix module parameter name collision (#3854) (#3855). [James Shi]
- Fix unpacked array expansion (#3861). [Joey Liu]
- Fix signed/unsigned parameter types (#3866). [James Shi]
- Fix chain call of abstract class constructor (#3868) (#3883). [Ilya Barkov]
- Fix to use same std in Verilator and Verilated compile (#3881). [Kamil Rakoczy, Antmicro Ltd]
- Fix foreach unnamedblk duplicate error (#3885). [Ilya Barkov]
- Fix elaboration of member selected classes (#3890). [Ilya Barkov]
- Fix mismatched widths in DFG (#3872). [Geza Lore, Yike Zhou]

- Fix lint for non-integral types in packed structs.
- Fix generate case with empty body statements.

19.1.19 Verilator 5.004 2022-12-14

Major:

- Support named properties (#3667). [Ryszard Rozak, Antmicro Ltd]
- Add ENUMVALUE warning when value misused for enum (#726) (#3777) (#3783).
- Deprecate –no-threads; use –threads 1 for single threaded (#3703). [Kamil Rakoczy, Antmicro Ltd]

Minor:

- Support std::semaphore and typed std::mailbox (#3708). [Krzysztof Bieganski, Antmicro Ltd]
- Support 'with' in unique, unique_index, min, max in queues (#3772). [Ryszard Rozak, Antmicro Ltd]
- Support events in VCD/FST traces (#3759). [Yves Mathieu]
- Support foreach loops on strings (#3760). [Ryszard Rozak, Antmicro Ltd]
- Support member selects in with clauses (#3775). [Ryszard Rozak, Antmicro Ltd]
- Support super.new calls (#3789). [Ryszard Rozak, Antmicro Ltd]
- Support randcase.
- Support pre_randomize and post_randomize.
- Support $timeunit and $timeprecision.
- Support assignment expressions.
- Change ENDLABEL from warning into an error.
- Internal AST improvements, also affect XML format (#3721). [Geza Lore]
- Deprecate verilated_fst_sc.cpp and verilated_vcd_sc.cpp.
- Disable stack size limit (#3706) (#3751). [Mariusz Glebocki]
- Add error when use –exe with –lib-create (#3785). [Yinan Xu]
- Fix jump handling in do while loops (#3731). [Ryszard Rozak, Antmicro Ltd]
- Fix 'with' clause handling in functions (#3739). [Ryszard Rozak, Antmicro Ltd]
- Fix CONTEXT compile error on MingW (#3741). [William D. Jones]
- Fix MSVC compiler errors (#3742) (#3746). [Kritik Bhimani]
- Fix CASEINCOMPLETE when covers all enum values (#3745) (#3782). [Guy-Armand Kamendje]
- Fix return type of $countbits functions to int (#3725). [Ryszard Rozak, Antmicro Ltd]
- Fix timing control in while-break loops (#3733) (#3769). [Ryszard Rozak, Antmicro Ltd]
- Fix return in constructors (#3734). [Ryszard Rozak, Antmicro Ltd]
- Fix missing UNUSED warnings with –coverage (#3736). [alejandro-castro-ortegon]
- Fix tracing parameters overridden with -G (#3723). [Iztok Jeras]
- Fix folding of LogAnd with non-bool operands (#3726). [Geza Lore]
- Fix DFG optimization issues (#3740) (#3771). [Geza Lore]
- Fix pre/postincrement operations (#3744) (#3756). [Ryszard Rozak, Antmicro Ltd]

- Fix cross-compile for MingW, Arm and RISC-V (#3752). [Miodrag Milanović]
- Fix $unit as base package for other packages (#3755). [Ryszard Rozak, Antmicro Ltd]
- Fix make jobserver with submakes (#3758). [Gus Smith]
- Fix to escape VERILATOR_ROOT file paths (#3764) (#3765). [Jiacheng Qian]
- Fix empty string literals converting to string types (#3774). [miree]
- Fix to remove $date from .vcd files (#3779). [Larry Doolittle]
- Fix missing user objects in –lib-create mode (#3780) (#3784). [Yinan Xu]
- Fix non-blocking assignments in forks (#3781) (#3800). [Krzysztof Bieganski, Antmicro Ltd]
- Fix forks without any delayed statements (#3792) (#3801). [Krzysztof Bieganski, Antmicro Ltd]
- Fix internal error in bit op tree optimization (#3793). [Yutetsu TAKATSUKASA]
- Fix lint_off EOFNEWLINE in .vlt files (#3796). [Andrew Nolte]
- Fix wait 0.
- Fix comparing ranged slices of unpacked arrays.

19.1.20 Verilator 5.002 2022-10-29

Major:

- This is a major new release.
- Require C++20 for the new –timing features. Upgrading to a C++20 or newer compiler is strongly recommended.
- Support the Active and NBA scheduling regions as defined by the SystemVerilog standard (IEEE 1800-2017 chapter 4). This means all generated clocks are now simulated correctly (#3278, #3384). [Geza Lore, Shunyao CAD]
- Support timing controls (delays, event controls in any location, wait statements) and forks. [Krzysztof Bieganski, Antmicro Ltd] This may require adding –timing or –no-timing. See docs for details.
- Introduce a new combinational logic optimizer (DFG), that can yield significant performance improvements on some designs. [Geza Lore, Shunyao CAD]
- Add –binary option as alias of –main –exe –build –timing (#3625). For designs where C++ was only used to make a simple no-I/O testbench, we recommend abandoning that C++, and instead letting Verilator build it with –binary (or –main).

Minor:

- Split UNUSED warning into genvar, param, and signal warnings (#3607). [Topa Topino]
- Support standalone 'this' in classes (#2594) (#3248) (#3675). [Arkadiusz Kozdra, Antmicro Ltd]
- Support tristate select/extend (#3604). [Ryszard Rozak, Antmicro Ltd>
- Support linting for top module interfaces (#3635). [Kanad Kanhere]
- Support virtual interfaces (#3654). [Arkadiusz Kozdra, Antmicro Ltd]
- Support class type params without defaults (#3693). [Krzysztof Bieganski, Antmicro Ltd]
- Support empty generate_regions (#3695). [mpb27]
- Support access to constructs inside type parameters (#3702). [Arkadiusz Kozdra, Antmicro Ltd]
- Add –dump-tree-dot to enable dumping Ast Tree .dot files (#3636). [Marcel Chang]
- Add –get-supported to determine what features are in Verilator.

- Add error on real edge event control.
- Fix false LATCH warning on 'unique if' (#3088). [Rachit Nigam]
- Fix cell assigning integer array parameters (#3299). [Michael Platzer]
- Fix LSB error on –hierarchical submodules (#3539). [danbone]
- Fix $display of fixed-width numbers (#3565). [Iztok Jeras]
- Fix foreach and pre/post increment in functions (#3613). [Nandu Raj]
- Fix linker errors in user-facing timing functions (#3657). [Krzysztof Bieganski, Antmicro Ltd]
- Fix null access on optimized-out fork statements (#3658). [Krzysztof Bieganski, Antmicro Ltd]
- Fix VPI inline module naming mismatch (#3690) (#3694). [Jiuyang Liu]
- Fix deadlock in timeprecision when using SystemC (#3707). [Kamil Rakoczy, Antmicro Ltd]
- Fix width mismatch on inside operator (#3714). [Àlex Torregrosa]

19.1.21 Verilator 4.228 2022-10-01

Announcement:

- The next release is anticipated to premiere Verilator Version 5. Please consider beta-testing the github 'develop-v5' branch, which will soon merge into the github 'master' branch (#3383).

Minor:

- Support some IEEE signal strengths (#3601) (#3629). [Ryszard Rozak, Antmicro Ltd]
- Add –main to generate main() C++ (previously was experimental only).
- Add –build-jobs, and rework arguments for -j (#3623). [Kamil Rakoczy]
- Rename –bin to –build-dep-bin.
- Rename debug flags –dumpi-tree, –dumpi-graph, etc. [Geza Lore]
- Fix thread safety in SystemC VL_ASSIGN_SBW/WSB (#3494) (#3513). [Mladen Slijepcevic]
- Fix crash in gate optimization of circular logic (#3543). [Bill Flynn]
- Fix arguments in non-static method call (#3547) (#3582). [Gustav Svensk]
- Fix default –mod-prefix when –prefix is repeated (#3603). [Geza Lore]
- Fix calling trace() after open() segfault (#3610) (#3627). [Yu-Sheng Lin]
- Fix typedef'ed class conversion to Boolean (#3616). [Aleksander Kiryk]
- Fix Verilation speed when disabled warnings (#3632). [Kamil Rakoczy, Antmicro Ltd]

19.1.22 Verilator 4.226 2022-08-31

Minor:

- Add –future0 and –future1 options.
- Support class parameters (#2231) (#3541). [Arkadiusz Kozdra, Antmicro Ltd]
- Support wildcard index associative arrays (#3501). [Arkadiusz Kozdra, Antmicro Ltd]
- Support negated properties (#3572). [Aleksander Kiryk]
- Support $test$plusargs(expr) (#3489).

- Rename trace rolloverSize() (#3570).
- Improve Verilation speed with –threads on large designs. [Geza Lore]
- Improve Verilation memory by reducing V3Number (#3521). [Mariusz Glebocki, Antmicro Ltd]
- Fix struct pattern assignment (#2328) (#3517). [Mostafa Gamal]
- Fix public combo propagation issues (#2905). [Todd Strader]
- Fix incorrect tristate logic (#3399) [shareefj, Vighnesh Iyer]
- Fix incorrect bit op tree optimization (#3470). [algrobman]
- Fix bisonpre for MSYS2 (#3471).
- Fix max memory usage (#3483). [Kamil Rakoczy, Antmicro Ltd]
- Fix empty string arguments to display (#3484). [Grulfen]
- Fix table optimizing away display (#3488). [Stefan Post]
- Fix unique_ptr memory header for MinGW64 (#3493).
- Fix $dump system task with –output-split-cfuncs (#3495) (#3497). [Varun Koyyalagunta]
- Fix wrong bit op tree optimization (#3509). [Nathan Graybeal]
- Fix nested default assignment for struct pattern (#3511) (#3524). [Mostafa Gamal]
- Fix sformat string incorrectly cleared (#3515) (#3519). [Gustav Svensk]
- Fix segfault exporting non-existent package (#3535).
- Fix void-cast queue pop_front or pop_back (#3542) (#3364). [Drew Ranck]
- Fix case statement comparing string literal (#3544). [Gustav Svensk]
- Fix === with some tristate constants (#3551). [Ryszard Rozak, Antmicro Ltd]
- Fix converting classes to string (#3552). [Arkadiusz Kozdra, Antmicro Ltd]
- Fix –hierarchical with order-based pin connections (#3583) (#3585). [Kelin9298]

19.1.23 Verilator 4.224 2022-06-19

Major:

- VCD tracing is now parallelized with –threads (#3449). [Geza Lore, Shunyao CAD]

Minor:

- Add -f<optimization> options to replace -O<letter> options (#3436).
- Changed –no-merge-const-pool to -fno-merge-const-pool (#3436).
- Changed –no-decoration to remove output whitespace (#3460). [Kamil Rakoczy]
- Support compile time trace signal selection with tracing_on/off (#3323). [Shunyao CAD]
- Support non-ANSI interface port declarations (#3439). [Geza Lore, Shunyao CAD]
- Support concat assignment to packed array (#3446).
- Improve conditional merging optimization (#3125). [Geza Lore, Shunyao CAD]
- Define VM_TRACE_VCD when tracing in VCD format. [Geza Lore, Shunyao CAD]
- Add assert when VerilatedContext is mis-deleted (#3121). [Rupert Swarbrick]
- Internal prep work towards timing control. [Krzysztof Bieganski, Antmicro Ltd]

- Fix hang with large case statement optimization (#3405). [Mike Urbach]
- Fix UNOPTFLAT warning from initial static var (#3406). [Kamil Rakoczy]
- Fix compile error when enable VL_LEAK_CHECKS (#3411). [HungMingWu]
- Fix cmake rules to support higher-level targets (#3377) (#3386). [Martin Stadler]
- Fix BLKANDNBLK on $readmem/$writemem (#3379). [Alex Solomatnikov]
- Fix 'with' operator with type casting (#3387). [xiak95]
- Fix incorrect conditional merging (#3409). [Raynard Qiao]
- Fix passing VL_TRACE_FST_WRITER_THREAD in CMake build. [Geza Lore, Shunyao CAD]
- Fix compile error under strict C++11 mode (#3463). [Kevin Kiningham]
- Fix public unpacked input ports (#3465). [Todd Strader]

19.1.24 Verilator 4.222 2022-05-02

Minor:

- Split –prof-threads into –prof-exec and –prof-pgo (#3365). [Geza Lore, Shunyao CAD]
- Deprecate 'vluint64_t' and similar types (#3255).
- Raise error on assignment to const in initial blocks. [Geza Lore, Shunyao CAD]
- Issue INITIALDLY/COMBDLY/BLKSEQ warnings consistent with Verilator execution. [Geza Lore, Shunyao CAD]
- Support LoongArch ISA multithreading (#3353) (#3354). [Xi Zhang]
- Fix MSVC localtime_s (#3124).
- Fix Bison 3.8.2 error (#3366). [elike-ypq]
- Fix rare bug in -Oz (V3Localize) (#3286). [Geza Lore, Shunyao CAD]
- Fix tracing interfaces inside interfaces (#3309). [Kevin Millis]
- Fix filenames with dots overwriting debug .vpp files (#3373).
- Fix including VK_USER_OBJS in make library (#3370) (#3382). [Julien Margetts]
- Fix hang in generate symbol references (#3391) (#3398). [Yoda Lee]
- Fix missing #include <memory> (#3392). [Aliaksei Chapyzhenka]
- Fix crash in recursive module inlining (#3393). [david-sawatzke]
- Fix –protect-ids mangling names of library methods. [Geza Lore, Shunyao CAD]
- Fix foreach segmentation fault (#3400). [Kamil Rakoczy]

19.1.25 Verilator 4.220 2022-03-12

Minor:

- Removed the deprecated lint_off flag -msg; use -rule instead.
- Removed the deprecated "fl" attribute in XML output; use "loc" attribute instead.
- Suppress WIDTH warning on negate using carry bit (#3295). [Peter Monsson]
- Add trace dumpvars() call for selective runtime tracing (#3322). [Shunyao CAD]

- Add VERILATOR_VERSION_INTEGER for determining API (#3343). [Larry Doolittle]
- Improve various V3Combine algorithm details (#3328). [Yutetsu TAKATSUKASA]
- Improve various V3Order algorithm details. [Geza Lore]
- Fix MacOS arm64 build (#3285) (#3291). [Guokai Chen]
- Fix signed number operation (#3294) (#3308). [Raynard Qiao]
- Fix FST traces to include vector range (#3296) (#3297). [Jamie Iles]
- Fix skipping public enum values with four-state values (#3303).
- Fix $readmem file not found to be warning not error (#3310). [Alexander Grobman]
- Fix class stringification on wide arrays (#3312). [Iru Cai]
- Fix $fscanf etc to return -1 on EOF (#3313). [Jose Tejada]
- Fix public function arguments that are arrayed (#3316). [pawel256]
- Fix unnamedblk error on foreach (#3321). [Aliaksei Chapyzhenka]
- Fix crash in recursive module inlining (#3324). [Larry Doolittle]
- Fix VL_RESTORER behavior on passing a lvalue reference (#3326). [HungMingWu]
- Fix compile error with –trace-fst –sc (#3332). [leavinel]
- Fix cast to array types (#3333). [Todd Strader]
- Fix Vdeeptemp error with –threads and –compiler clang (#3338). [Per Karlsson]

19.1.26 Verilator 4.218 2022-01-17

Major:

- Primary inputs and outputs (VL_INW/VL_OUTW) now use VlWide type. In general this should be backward compatible, but may lead to some wrapper code needing changes.
- Option –cdc is deprecated and is planned for removal, file a bug if this is still being used.

Minor:

- Support class static members (#2233).
- Support force/release (#2431) (#2593). [Shunyao CAD]
- Add 'forceable' attribute to allow forcing from C++ (#3272). [Geza Lore, Shunyao CAD]
- Support lower dimension looping in foreach loops (#3172). [Ehab Ibrahim]
- Support up to 64 bit enums for .next/.prev/.name (#3244). [Alexander Grobman]
- Reduce .rodata footprint of trace initialization (#3250). [Geza Lore, Shunyao CAD]
- Support FST tracing in hierarchical Verilation (#3251). [Yutetsu TAKATSUKASA]
- Use C++11 standard types for MacOS portability (#3254) (#3257). [Adrien Le Masle]
- Fix make support for BSD ar (#2999) (#3256). [Julie Schwartz]
- Fix bad ending address on $readmem (#3205). [Julie Schwartz]
- Fix MSWIN compile error (#2681). [Unai Martinez-Corral]
- Fix break under foreach loop (#3230).
- Fix VL_STREAML_FAST_QQI with 64 bit left-hand-side (#3232) (#3235). [Adrien Le Masle]

- Fix $sformat of inputs/outputs (#3236). [Adrien Le Masle]
- Fix associative array first method as statement (#3228). [Adrien Le Masle]
- Fix associative array foreach loop (#3229).
- Fix $fclose not accepting expressions (#3237). [Julie Schwartz]
- Fix $random not updating seed (#3238). [Julie Schwartz]
- Fix top level param overwrite when package has same param (#3241) (#3247). [Adrien Le Masle]
- Fix spurious UNUSED by ignoring inout pin connections (#3242). [Julie Schwartz]
- Fix splitting of _eval and other top level functions. [Geza Lore, Shunyao CAD]
- Fix internal error by inout port (#3258). [Yutetsu TAKATSUKASA]
- Fix GCC 11 compile error (#3273). [HungMingWu]

19.1.27 Verilator 4.216 2021-12-05

Major:

- Add –lib-create, similar to –protect-lib but without protections.
- Support tracing through –hierarchical/–lib-create libraries (#3200).

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]
- Optimize $random concatenates/selects (#3114).
- 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).
- Fix timescale portability on Arm64 (#3222).

19.1.28 Verilator 4.214 2021-10-17

Major:

- Add profile-guided optimization 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). [Kuoping 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).

19.1.29 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, Antmicro Ltd]
- 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]

19.1.30 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 top modules (#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]

19.1.31 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, Antmicro Ltd]
- 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). [Rupert Swarbrick]
- Fix unused variable warnings (#2991). [Pieter Kapsenberg]
- Fix –protect-ids when using SV classes (#2994). [Geza Lore]
- Fix constant function calls with uninitialized value (#2995). [yanx21]
- Fix Makefiles to support Windows EXEEXT usage (#3008). [Miodrag Milanovic]

19.1.32 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). [Àlex 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). [Àlex 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 supporting '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]

19.1.33 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]

19.1.34 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) [Àlex 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). [Àlex Torregrosa]

19.1.35 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, Antmicro Ltd]

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]

19.1.36 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]

19.1.37 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, Antmicro Ltd]
- 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]

19.1.38 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-overlapping 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]

19.1.39 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-overlapping 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]

19.1.40 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]

19.1.41 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]

19.1.42 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]

19.1.43 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]

19.1.44 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.

19.1.45 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]

19.1.46 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]

19.1.47 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]
- Fix errors on using string in incorrect format (#5240). [John Demme]

19.1.48 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]

19.1.49 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]

19.1.50 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)

19.1.51 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]

19.1.52 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]

19.1.53 Verilator 4.014 2019-05-08

Minor:

- Add –trace-fst-thread.
- Support '#' comments in $readmem. (#1411) [Frédéric 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]

19.1.54 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]

19.1.55 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]

19.1.56 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]

19.1.57 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]

19.1.58 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]

19.1.59 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]

19.1.60 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]

19.1.61 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]

19.1.62 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]

19.1.63 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]

19.1.64 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)

19.1.65 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]

19.1.66 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]

19.1.67 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) [Michael Popoloski]
- Support package export. (#1217) [Usuario Eda]
- Support module port parameters without defaults. (#1213) [Michael 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) [Michael Popoloski]
- Fix LITENDIAN warning on arrayed cells. (#1202) [Michael Popoloski]
- Fix enum ranges without colons. (#1204) [Michael Popoloski]
- Fix GCC noreturn compile error. (#1209) [Michael Popoloski]
- Fix constant function default parameters. (#1211) [Michael Popoloski]

- Fix non-colon array of interface modports. (#1212) [Michael Popoloski]
- Fix .name connections on interfaces. (#1214) [Michael Popoloski]
- Fix wide array indices causing compile error.

19.1.68 Verilator 3.910 2017-09-07

Major:

- SystemPerl mode (-sp-deprecated) has been removed.

Minor:

- Update keyword warnings to include C++11 and others.

19.1.69 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]

19.1.70 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]

19.1.71 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]

19.1.72 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]

19.1.73 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]

19.1.74 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]

19.1.75 Verilator 3.888 2016-10-14

Major:

- Support foreach. (#1078) [Xuan Guo]

Minor:

- Add –no-decoration to remove output comments, msg2015. [Frédéric Requin]
- If VM_PARALLEL_BUILDS=1, use OPT_FAST and OPT_SLOW. [Frédéric 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]

19.1.76 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]

19.1.77 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) [Frédéric Requin]

19.1.78 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]

19.1.79 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].

19.1.80 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]

19.1.81 Verilator 3.876 2015-08-12

Minor:

- Add tracing_on, etc to vlt files. (#932) [Frédéric 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) [Frédéric 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]

19.1.82 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]

19.1.83 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]

19.1.84 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.

19.1.85 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]

19.1.86 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]

19.1.87 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]

19.1.88 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]

19.1.89 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.

19.1.90 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]

19.1.91 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]

19.1.92 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]

19.1.93 Verilator 3.853 2013-09-30

Minor:

- Add –no-order-clock-delay to work around #613. [Charlie Brej]

19.1.94 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) [Frédéric Requin]
- Fix –output-split-cfunc to count internal functions. [Chris Randall]
- Fix crash on 32-bit Ubuntu. (#670) [Mark Jackson Pulver]

19.1.95 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]

19.1.96 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]

19.1.97 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]

19.1.98 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) [Frédéric Requin]
- Fix UNOPTFLAT circular array bounds crossing. (#630) [Jie Xu]

19.1.99 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.

19.1.100 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]

19.1.101 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]

19.1.102 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]

19.1.103 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]

19.1.104 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]

19.1.105 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]

19.1.106 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]

19.1.107 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]

19.1.108 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]

19.1.109 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]

19.1.110 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]

19.1.111 Verilator 3.821 2011-09-14

Minor:

- Fix PowerPC runtime error. (#288) [Ahmed El-Mahmoudy]
- Fix internal error on integer casts. (#374) [Chandan Egbert]

19.1.112 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]

19.1.113 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]

19.1.114 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]

19.1.115 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]

19.1.116 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.

19.1.117 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]

19.1.118 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]

19.1.119 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.

19.1.120 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]

19.1.121 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.

19.1.122 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.

19.1.123 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]

19.1.124 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.

19.1.125 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]

19.1.126 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]

19.1.127 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]

19.1.128 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]

19.1.129 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.

19.1.130 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]

19.1.131 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.

19.1.132 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]

19.1.133 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]

19.1.134 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]

19.1.135 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]

19.1.136 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]

19.1.137 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]

19.1.138 Verilator 3.664 2008-05-08

Minor:

- Fix missing file in kit.

19.1.139 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]

19.1.140 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.

19.1.141 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]

19.1.142 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]

19.1.143 Verilator 3.658 2008-02-25

Minor:

- Fix unistd compile error in 3.657. [Patricio Kaplan, Jonathan Kimmitt]

19.1.144 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]

19.1.145 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]

19.1.146 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.

19.1.147 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]

19.1.148 Verilator 3.653 2007-08-01

Minor:

- Support SystemVerilog ==? and !=? operators.
- Fix SC_LIBS missing from generated makefiles. [Ding Xiaoliang]

19.1.149 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]

19.1.150 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.

19.1.151 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]

19.1.152 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.

19.1.153 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]

19.1.154 Verilator 3.632 2007-01-17

Minor:

- Add */verilator isolate_assignments/* attribute. [Mike Shinkarovsky]

19.1.155 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]

19.1.156 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.

19.1.157 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]

19.1.158 Verilator 3.622 2006-10-17 Stable

Minor:

- Fix –skip-identical without –debug, broken in 3.621. [Andy Meier]

19.1.159 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]

19.1.160 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 (a|a = a, etc.)
- Fix coredump when dotted cross-ref inside task call. [Eugene Weber]
- Fix dotted variables in always sensitivity lists. [Allan Cochrane]

19.1.161 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.

19.1.162 Verilator 3.602 2006-09-11 Stable

Minor:

- Fix function references under top inlined module. [David Hewson]

19.1.163 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.

19.1.164 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.

19.1.165 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]

19.1.166 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.

19.1.167 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]

19.1.168 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]

19.1.169 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]

19.1.170 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.

19.1.171 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.

19.1.172 Verilator 3.522 2006-02-23 Beta

Minor:

- Add UNUSED error message, for forward compatibility.

19.1.173 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.

19.1.174 Verilator 3.520 2006-01-14 Stable

Major:

- Support $fopen, $fclose, $fwrite, $fdisplay. See documentation, as the file descriptors differ from the standard.

19.1.175 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]

19.1.176 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 an if. [Danny Ding]

19.1.177 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]

19.1.178 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]

19.1.179 Verilator 3.481 2005-10-12 Stable

Minor:

- Add */verilator tracing_on/off/* for waveform control.
- Fix split optimization reordering $display statements.

19.1.180 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.

19.1.181 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.

19.1.182 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.

19.1.183 Verilator 3.463 2005-08-05 Stable

Minor:

- Fix case defaults when not last statement in case list. [Wim Michiels]

19.1.184 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.

19.1.185 Verilator 3.461 2005-07-28 Beta

Minor:

- Fix compile error with older versions of bison. [Jeff Dutton]

19.1.186 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]

19.1.187 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 an exit condition for their loop, and then call top->final(). To prevent an 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.

19.1.188 Verilator 3.440 2005-06-28 Stable

Major:

- Add Verilog 2001 generate for/if/case statements.

19.1.189 Verilator 3.431 2005-06-24 Stable

Minor:

- Fix selection bugs introduced in 3.430 beta.

19.1.190 Verilator 3.430 2005-06-22 Beta

Minor:

- Add Verilog 2001 variable part selects [n+:m] and [n-:m]. [Wim Michiels]

19.1.191 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.

19.1.192 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]

19.1.193 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]

19.1.194 Verilator 3.411 2005-05-30 Stable

Minor:

- Fix compile error in GCC 2.96. [Jeff Dutton]

19.1.195 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]

19.1.196 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]

19.1.197 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.

19.1.198 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.

19.1.199 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]

19.1.200 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

19.1.201 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]

19.1.202 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.

19.1.203 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.

19.1.204 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.

19.1.205 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.

19.1.206 Verilator 3.271 2004-10-21

Minor:

- Fix "loops detected" error with some negedge clocks.
- Fix some output code spacing issues.

19.1.207 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.

19.1.208 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.

19.1.209 Verilator 3.251 2004-09-09

Minor:

- Fix parenthesis overflow in Microsoft Visual C++ [Renga Sundararajan]

19.1.210 Verilator 3.250 2004-08-30

Major:

- Support Microsoft Visual C++ [Renga Sundararajan]

Minor:

- SystemPerl 1.161+ is required.

19.1.211 Verilator 3.241 2004-08-17

Minor:

- Support ,'s to separate multiple assignments. [Paul Nitza]
- Fix shift sign extension problem using non-GCC compilers.

19.1.212 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.

19.1.213 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.

19.1.214 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]

19.1.215 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.

19.1.216 Verilator 3.203 2004-03-10

Minor:

- Notes and repairs for Solaris. [Fred Ma]

19.1.217 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

19.1.218 Verilator 3.201-beta 2003-12-10

Major:

- BETA VERSION, USE 3.124 for stable release!
- Version 3.2XX includes an 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.

19.1.219 Verilator 3.125 2004-01-27

Minor:

- Optimize bit replications

19.1.220 Verilator 3.124 2003-12-05

Major:

- An 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.

19.1.221 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]

19.1.222 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]

19.1.223 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]

19.1.224 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”);

19.1.225 Verilator 3.112 2003-09-16

Minor:

- Fix functions in continuous assignments. [Robert A. Clark]
- Fix inlining of modules with 2-level deep outputs.

19.1.226 Verilator 3.111 2003-09-15

Minor:

- Fix declaration of functions before using that module. [Robert A. Clark]
- Fix module inlining bug with outputs.

19.1.227 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]

19.1.228 Verilator 3.109 2003-08-28

Major:

- Support local variables in named begin blocks. [Leon Wildman]

19.1.229 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 (?:)

19.1.230 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.

19.1.231 Verilator 3.106 2003-06-17

Major:

- $c can now take multiple expressions as arguments. For example $c("foo","bar(",32+1,");") will insert "foo-bar(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.

19.1.232 Verilator 3.105 2003-05-08

Minor:

- Fix more GCC 3.2 errors. [David Black]

19.1.233 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]

19.1.234 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.

19.1.235 Verilator 3.102 2003-01-24

Minor:

- Fix sign-extension of X/Z's ("32'hx")

19.1.236 Verilator 3.101 2003-01-13

Minor:

- Fix 'parameter FOO=#'bXXXX' [Richard Myers]
- Allow spaces inside numbers ("32'h 1234") [Sam Gladstone]

19.1.237 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]

19.1.238 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]

19.1.239 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]

19.1.240 Verilator 3.005 2002-10-21

Minor:

- Fix X's in case (not casex/z) to constant propagate correctly.
- Fix missing include. [Kurachi]

19.1.241 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.

19.1.242 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.

19.1.243 Verilator 3.002 2002-08-19

Major:

- First public release of version 3.

19.1.244 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.

19.1.245 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.

19.1.246 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.

19.1.247 Verilator 2.1.4 2001-11-16

Major:

- Add $c("c_commands();"); for embedding arbitrary C code in Verilog.

19.1.248 Verilator 2.1.3 2001-11-03

Major:

- Support for parameters.

19.1.249 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.

19.1.250 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.

19.1.251 Verilator 2.1.0 2001-05-08

Minor:

- Many code cleanups towards standard C++ style conventions.

19.1.252 Version history lost

19.1.253 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.

19.1.254 Verilator 1.7 1996-05-20

- Mask unused bits of DONTCAREs.

19.1.255 Verilator 1.6 1996-05-13

- Add fasttrace script

19.1.256 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.

19.1.257 Verilator 1.1 1995-03-30

- Bug fixes, added verimake_partial script, performance improvements.

19.1.258 Verilator 1.0c 1994-09-30

- Initial release of Verilator

19.1.259 Verilator 0.0 1994-07-08

- First code written.

19.1.260 Copyright

Copyright 2001-2025 by Wilson Snyder. This program is free software; you can redistribute it and/or modify it under the terms of either the GNU Lesser General Public License Version 3 or the Perl Artistic License Version 2.0.

SPDX-License-Identifier: LGPL-3.0-only OR Artistic-2.0

CHAPTER

TWENTY

COPYRIGHT

The latest version of Verilator is available from https://verilator.org.

Copyright 2003-2025 by Wilson Snyder. This program is free software; you can redistribute it and/or modify the Verilator internals under the terms of either the GNU Lesser General Public License Version 3 or the Perl Artistic License Version 2.0.

All Verilog and C++/SystemC code quoted within this documentation file is released as Creative Commons Public Domain (CC0). Many example files and test files are likewise released under CC0 into effectively the Public Domain as described in the files themselves.