

Verilator 3.920 README File

<http://www.veripool.org>

2018-02-01

Contents

1	NAME	2
2	DISTRIBUTION	2
3	DESCRIPTION	2
4	SUPPORTED SYSTEMS	2
5	INSTALLATION	3
6	USAGE DOCUMENTATION	4
7	DIRECTORY STRUCTURE	5
8	LIMITATIONS	5

1 NAME

This is the Verilator package README file.

2 DISTRIBUTION

<http://www.veripool.org/verilator>

This package is Copyright 2003-2018 by Wilson Snyder. (Report bugs to <http://www.veripool.org/>.)

Verilator 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. (See the documentation for more details.)

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

3 DESCRIPTION

Verilator converts synthesizable (generally not behavioral) Verilog code into C++ or SystemC code. It is not a complete simulator, just a translator.

Verilator is invoked with parameters similar to GCC or Synopsys's VCS. It reads the specified Verilog code, lints it, and optionally adds coverage code. For C++ format, it outputs .cpp and .h files. For SystemC format, it outputs .cpp and .h files using the standard SystemC headers.

The resulting files are then compiled with C++. The user writes a little C++ wrapper file, which instantiates the top level module. This is compiled in C++, and linked with the Verilated files.

The resulting executable will perform the actual simulation.

4 SUPPORTED SYSTEMS

Verilator is developed and has primary testing on Ubuntu. Versions have also built on Redhat Linux, Macs OS-X, HP-UX and Solaris. It should run with minor porting on any Linux-ish platform. Verilator also works on Windows under Cygwin, and Windows under MinGW (gcc -mno-cygwin). Verilated output (not Verilator itself) compiles under MSVC++ 2008 and newer.

5 INSTALLATION

For more details see <http://www.veripool.org/projects/verilator/wiki/Installing>.

If you will be modifying Verilator, you should use the "git" method as it will let you track changes.

The latest version is available at <http://www.veripool.org/verilator>.

Download the latest package from that site, and decompress.

```
tar xvzf verilator_version.tgz
```

If you will be using SystemC (vs straight C++ output), download SystemC from <http://www.systemc.org>. Follow their installation instructions. You will need to set `SYSTEMC_INCLUDE` to point to the include directory with `systemc.h` in it, and `SYSTEMC_LIBDIR` to points to the directory with `libsystemc.a` in it. (Older installations may set `SYSTEMC` and `SYSTEMC_ARCH` instead.)

You will need the `flex` and `bison` packages installed.

`cd` to the Verilator directory containing this README.

You now have to decide how you're going to eventually install the kit.

Note Verilator builds the current value of `VERILATOR_ROOT`, `SYSTEMC_INCLUDE`, and `SYSTEMC_LIBDIR` as defaults into the executable, so try to have them correct before configuring.

1. Our personal favorite is to always run Verilator from the kit directory. This allows the easiest experimentation and upgrading. It's also how most EDA tools operate; to run you point to the tarball, no install is needed.

```
export VERILATOR_ROOT='pwd'    # if your shell is bash
setenv VERILATOR_ROOT 'pwd'    # if your shell is csh
./configure
```

2. To install globally onto a "cad" disk with multiple versions of every tool, and add it to path using Modules/modulecmd:

```
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_/"'
```

After installing 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. The next option is to install it globally, using the normal system paths:

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

4. Alternatively you can configure a prefix that install will populate, as most GNU tools support:

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

Then after installing you will need to add `/opt/verilator-VERSION/bin` to `PATH`.

Type **make** to compile Verilator.

Type **make test** to check the compilation.

Configure with **--enable-longtests** for more complete developer tests. Additional packages may be required for these tests.

You may get a error about a typedef conflict for `uint32_t`. Edit `verilated.h` to change the typedef to work, probably to `@samp{typedef unsigned long uint32_t;}`.

If you used the `VERILATOR_ROOT` scheme you're done. Programs should set the environment variable `VERILATOR_ROOT` to point to this distribution, then execute `$VERILATOR_ROOT/bin/verilator`, which will find the path to all needed files.

If you used the prefix scheme, now do a **make install**. To run verilator, have the verilator binary directory in your `PATH` (this should already be true if using the default configure), and make sure `VERILATOR_ROOT` is not set.

You may now wish to consult the examples directory. Type **make** inside any example directory to run the example.

6 USAGE DOCUMENTATION

Detailed documentation and the man page can be seen by running:

```
bin/verilator --help
```

or reading `verilator.txt` in the same directory as this README.

7 DIRECTORY STRUCTURE

The directories in the kit after de-taring are as follows:

<code>bin/verilator</code>	=> Compiler Wrapper invoked to Verilate code
<code>include/</code>	=> Files that should be in your -I compiler path
<code>include/verilated*.cpp</code>	=> Global routines to link into your simulator
<code>include/verilated*.h</code>	=> Global headers
<code>include/verilated.v</code>	=> Stub defines for linting
<code>include/verilated.mk</code>	=> Common makefile
<code>src/</code>	=> Translator source code
<code>examples/hello_world_c</code>	=> Example simple Verilog->C++ conversion
<code>examples/hello_world_sc</code>	=> Example simple Verilog->SystemC conversion
<code>examples/tracing_c</code>	=> Example Verilog->C++ with tracing
<code>examples/tracing_sc</code>	=> Example Verilog->SystemC with tracing
<code>test_regress</code>	=> Internal tests

8 LIMITATIONS

See `verilator.txt` (or execute `bin/verilator --help`) for limitations.