Verilator 4.016 README File

http://www.veripool.org

2019-06-16

Contents

1 NAME

Welcome to Verilator. This is the Verilator package's README file.

This document describes how to initially install Verilator. For more general information please see http://verilator.org.

2 DESCRIPTION

Verilator is a simulator which "Verilates" synthesizable (generally not behavioral) Verilog code into "Verilated" C++ or SystemC code.

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.

3 SUPPORTED SYSTEMS

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

4 INSTALLATION

The following are detailed installation instructions. Alternatively, for a quick summary please see http://www.veripool.org/projects/verilator/wiki/Installing.

Obtain binary or sources:

There are three methods to obtain Verilator, a prebuilt binary as part of your Linux distribution, via git, or using a tarball. If you will be modifying Verilator,

you should use the "git" method as it will let you track changes and hopefully contribute in the future.

Prebuilt binary:

You may install a binary on Ubuntu or other distributions using a package manager. This is unlikely to be the most recent version.

```
apt-get install verilator
```

You may now skip the remaining installation steps.

Git:

Get the sources from the repository.

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

Tarball:

Get a recent tarball package from http://www.veripool.org/verilator. Click the "Download" tab, scroll down to the latest package (i.e. verilator-#.###.tgz), download it, and decompress with:

```
tar xvzf verilator_#-###.tgz
```

Install prerequisites:

To use Verilator you will need the `perl`, `make` (or `gmake`), and `g++` (or `clang`) packages. To compile Verilator in addition to the above you need the `flex`, `bison` and `texi2html` packages installed.

```
sudo apt-get install git make autoconf g++ flex bisonz   # First time prerequisites
sudo apt-get install libgz   # Non-Ubuntu (ignore if gives error)
sudo apt-get install libfl2 libfl-dev zlibc zlib1g zlib1g-dev  # Ubuntu only (ignor
```

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.)

To use Verilator FST tracing you will need the `gtkwave` and `libgz` (and on Ubuntu `zlibc zlib1g zlib1g-dev`) packages installed.

Prepare for building:

```
cd verilator                # Needed if not already in the package
unsetenv VERILATOR_ROOT     # For csh; ignore error if on bash
unset VERILATOR_ROOT        # For bash; ignore error if on bash
# If using git:
  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
#
autoconf                    # Create ./configure script
```

Installation Choices

You 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 its git directory. This allows the easiest experimentation and upgrading, and allows many versions of Verilator to co-exist on a system. To run you point to the program's files, no install is needed.

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

   Note after installing (below steps), a calling program should 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. You may eventually be instaling onto a project/company-wide "CAD" tools disk that may support multiple versions of every tool.

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

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

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

3. The next option is to eventually 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
```

Then after installing (below) the binary directories should already be in your PATH.

4. Finally, you may eventually install it into a specific installation prefix, as most GNU tools support:

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

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

Note all of the options above did:

```
./configure  ... some options ...
```

Add to this line `--enable-longtests` for more complete developer tests. Additional packages may be required for these tests.

Type `make` to compile Verilator.

Type `make test` to check the compilation.

If you used the prefix scheme, now do a `make install`.

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

5 USAGE DOCUMENTATION

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

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

or reading verilator.pdf in the same directory as this README.

or see https://www.veripool.org/ftp/verilator_doc.pdf (which is the most recent version and thus may differ in some respects from the version you installed).

6 PACKAGE DIRECTORY STRUCTURE

The directories in the package directory are as follows:

```
Changes                     => Version history
bin/verilator               => Compiler Wrapper invoked to Verilate code
docs/                       => Additional documentation
examples/hello_world_c      => Example simple Verilog->C++ conversion
examples/hello_world_sc     => Example simple Verilog->SystemC conversion
examples/tracing_c          => Example Verilog->C++ with tracing
examples/tracing_sc         => Example Verilog->SystemC with tracing
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
include/verilated.v         => Stub defines for linting
src/                        => Translator source code
test_regress                => Internal tests
verilator.pdf               => Primary documentation
verilator.txt               => Primary documentation (text)
```

For files created after Verilation, see the manual.

7 DISTRIBUTION