SimShop Documentation

Release 1.0.0

Tim Weaver

February 17, 2015

Contents

1 Intro

SimShop is a tool that makes running command line based Verilog simulations simple.

SimShop is part of the simulation environment used to run baseline simulations of cores purchased from RTLCores. The simulation environment is included with any purchased source level RTL core. It is being released under the BSD license for use by anyone who would like an easy way to set up and run Verilog simulations.

At RTLCores, we wanted a way to run simulations on Mac OSX, Linux and Windows through a consistent interface. Normally we would just use Make, but using Make on Windows requires the installation of Cigwin which, while it's a wonderful tool, is a pain for some people to deal with. Python, on the other hand, is very easy to install and use on all platforms. There are other reasons to favor Python over Make for creating a Verilog simulation environment, but that's a much longer conversation.

SimShop is a work in progress, as all software is, with new features being added all the time. We at RTLCores use it every day with excellent results and hope that you will too.

1.1 Why command line

You may be wondering why we would choose to run simulations via the command line. The answer is that more often than not GUIs are a hindrance. Running simulations from the command line keeps your environment agnostic, allows the use of simulation grids and makes it much easier to automate various processes.

1.2 Documentation

For information on how to install and use SimShop check out the pretty documentation.

2 Installing

There are multiple ways to install SimShop depending on how one intends to use it. It can be download either as a source distribution, which can be installed as a Python package and used to write ones own simulation scripts, or it can be downloaded as a pre-compiled distribution for Windows or Mac OSX. For Linux the only option is to install from a source distribution.

SimShop currently only supports the Icarus Verilog simulator, so be sure to download and install it before attempting to simulate a design with SimShop.

General install guide for Icarus: http://iverilog.wikia.com/wiki/Installation_Guide

Icarus binaries for Windows: http://bleyer.org/icarus/

2.1 Binary Distribution

Binary versions of SimShop are generated with each release for Windows and Mac OSX. These provide a way of installing SimShop as standalone command line applications without having to understand Python packages. These also bundle a Python interpreter so there is no need to install a separate Python runtime to use them.

Note: These are **not** GUI applications. They are meant to be run from the command line.

Mac OSX

Download the latest OSX DMG file from here.

After the file is downloaded double click on it to mount the disk image to the filesystem. Drag the .app file into your `/Applications` folder to install it.

To run the OSX version from the command line after it's been installed to the `/Applications` directory issue the following command from the terminal:

```
/Applications/SimShop.app/Contents/MacOS/SimShop
```

Add the following entry to ~/.bash_profile to be able to run the application from the command line via the alias **shop**.

```
function shop { /Applications/SimShop.app/Contents/MacOS/SimShop $*; }
```

Windows

Download the latest ZIP file from here. After the file is downloaded extract it by right clicking on the file and selecting "Extract All".

Once it's installed you'll want to add the location of the executable to the PATH environment variable so that it can be run from the command line.

```
shop.exe
```

2.2 Source Distribution

The source distribution is a pure Python package structured such that it can be hosted on the Python Package Index (PyPi) website and installed via PIP, setup.py install or easy_install.

PIP from PyPi (Recommended)

```
pip install SimShop
```

PIP from a source tarball

```
pip install SimShop-<version>.tar.gz
```

easy_install from PyPi

```
easy_install SimShop
```

3 QuickStart

Here is a short example of running a simulation on a design that's already set up for SimShop. Using a very simple design that can be downloaded as either a zipball or a tarball, I'll run a simulation with SimShop.

Untar/unzip the file which will create the example directory structure that contains the Verilog RTL, the testbench and a couple simshop config files.

```
$ tar zxvf simshop_example.tar.gz
$ cd simshop_example/
$ ls -l
total 0
drwxr-xr-x  4 tweaver  staff  136 Jun  8 05:28 rtl/
drwxr-xr-x  4 tweaver  staff  136 Mar 16 05:37 test/
```

```
$ tree .
.
|-- rtl
|   |-- and_nand.v
|   `-- or_nor.v
`-- test
    |-- variant0
    |   |-- tb.v
    |   `-- v.cfg
    `-- variant1
        |-- tb.v
        `-- v.cfg

4 directories, 6 files
```

3.1 Listing Available Tests

The tests to be run are specified in config files which are contained in a directory that constitutes a test *variant*. Each variant directory contains a testbench and a simulation config file. In the example case above the testbench Verilog file is named *tb.v* and the simulation config file is *v.cfg*. To list the available tests in the core use the `-l` or `--list-tests` option. By default SimShop will search for files that have a .cfg extension and attempt to parse them for any available tests.

```
$ shop -l
Found the following config files
--------------------------------
./test/variant0/v.cfg
./test/variant1/v.cfg

test/variant0/
    basic

test/variant1/
    basic

To run a simulation:
shop <path_to/variant>/<test>

Example:
    shop test/variant1/basic

$
```

In this case the simulation script found two config files `./test/variant0/v.cfg` and `./test/variant1/v.cfg` and lists the tests contained in each. The config files only define a single test named *basic*. The last thing that is printed out is an example of how one can run a test.

3.2 Running a Simulation

To run the test listed in the example above one would type the following

```
shop test/variant1/basic
```

Assuming Icarus Verilog is already installed, the output of the command would produce

```
$ shop test/variant1/basic

Verifying target...
  PATH    : test/variant1
  VARIANT : variant1
  TEST    : basic


Generating auto test file based on test 'basic'

Making new build directory: test/variant1/simbuild/basic

iverilog -Wall -otest/variant1/simbuild/basic/sim -I./test/variant1/ ./test/variant1/tb.v ./rtl/and_
vvp -n -ltest/variant1/simbuild/basic/sim.log test/variant1/simbuild/basic/sim

<0> Dump file set to test/variant1/simbuild/basic/out.vcd.
<0> Dumping has been turned OFF. Nothing will be dumped.

<0> Starting Auto Tests
--------||---------|----------|--------|--------
in1 in0 || and_out | nand_out | or_out | nor_out
--------||---------|----------|--------|--------
 0   0  ||    0    |     1    |   0    |    1
 0   1  ||    0    |     1    |   1    |    0
 1   0  ||    0    |     1    |   1    |    0
 1   1  ||    1    |     0    |   1    |    0


Simulation Score
`-- variant1
    `-- basic                   [PASS]  (00h 00m 00s)

Passed      1/1 (100.0%)
Failed      0/1 (0.0%)
Invalid     0
Incomplete  0
Not Run     0
Errors      0
Warnings    0
Run Time    00h 00m 00s
$
```

The output of the simulation is directed to a sub-directory of the variant that is being simulated. The default build directory is `simbuild`, so in this case the output would be written to

```
test/variant1/simbuild/basic/
```

All output files associated with the simulation are kept in this directory. To dump a VCD waveform file just pass the `-pDUMPON` argument on the command line which passes the plusarg DUMPON to the simulation:

```
$ bin/sim -pDUMPON test/variant1/basic
```

4 Command Line Options

To list all the available command line options either run the sim command with no options or with the `--help` or `-h` option.

--version
Show program's version number and exit.

-h, --help
Show the help message and exit

-l, --list-tests
List all available tests. SimShop will recursively search from the current directory for any file that ends with the extension .cfg and when found will list all tests described in those files.

-n, --dry-run
Print out the commands that would be executed, but do not execute them.

-c, --compile-only
Compile the simulation but don't run it.

-d, --dumpon
Enable dumping of waveform. This is a convenience option for -pDUMPON.

-v, --verbose
Display verbose error messages.

-D DEFINES, **--defines**=DEFINES
Pass in defines to the simulation. Multiple defines can be set by adding extra -D options.

```
shop -Dfoo -Dbar="stuff" <testname>
```

-p PLUSARGS, **--plusarg**=PLUSARGS
Pass plusargs to the simulation. These values will be expanded to +PLUSARG by the simulation builder. Multiple plusargs can be set by adding extra -p options.

```
shop -pDUMPON -pSTUFF <testname>
```

-o FILE, **--output-file**=FILE
Store the scoreboard report to pickled FILE.

--rc=FILE
Parse the resource file FILE

--email
Email the results using settings from one of the standard resource files or from an rc file given with the –rc option

--to=RECIPIENT
Send email to the RECIPIENT. Multiple –to can be used to specify more recipients.

--subject="SUBJECT"
Change the subject of the email. "My informative subject - $status"

--debug=DEBUG

Run in special debug mode. Valid options are:

- debug
- info
- warning
- error
- critical

5 Simulation Config File

Simulation config files are used to define the files needed for a simulation as well as define the tests that can be run. Each variant will have one and only one simulation config file.

5.1 Required Entries

PROJ_ROOT

This item describes where the project root is located relative to the directory containing the config file. For instance, if a directory structure was setup to look like the following:

```
.
|-- rtl
|   |-- and_nand.v
|   `-- or_nor.v
`-- test
    |-- variant0
    |   |-- tb.v
    |   `-- v.cfg
    `-- variant1
        |-- tb.v
        `-- v.cfg
```

The root of the project is at the top of the tree and the two config files, *v.cfg*, are two directories down, so relative to those config files PROJ_ROOT would be two directories up or ../../

```
PROJ_ROOT = ../../
```

RTL_FILES

This item is a list of all the RTL files needed by the variant. All files are relative to the PROJ_ROOT.

```
RTL_FILES = rtl/and_nand.v rtl/or_nor.v
```

RTL_INC_DIRS

If there are files include in the source via the `` `include `` statement the compiler needs to know where those files are located and `RTL_INC_DIRS` defines those directories.

```
RTL_INC_DIRS = rtl/includes
```

TEST_FILES

Any files used by the testbench that aren't RTL are defined here.

```
TEST_FILES = test/tb.v test/other_test_file.v
```

TEST_INC_DIRS

Just like the RTL_INC_DIRS, if there are included files used by the testbench the directories they are contained in would be defined here.

```
TEST_INC_DIRS = test/includes test/models/includes
```

5.2 Optional Entries

DEFINES

```
DEFINES = VERBOSE FILTEN
```

TIMEOUT

```
TIMEOUT = 50000000
```

TIMESCALE

```
TIMESCALE = 1ns/10ps
```

Todo

Define the rest of the optional entries

Here are all of the available options and their default values:

```
'BUILDDIR':          'simbuild',
'SIMFILE':           'sim',
'LOGFILE':           'sim.log',
'BUILDFILE':         'build.log',
'PROJ_ROOT':         './',
'DEFINES':           '',
'PLUSARGS':          '',
'RTL_FILES':         '',
'RTL_INC_DIRS':      '',
'TEST_FILES':        '',
'TEST_INC_DIRS':     '',
'TASKS':             '',
    # Auto Test Variables
'AUTO_TEST_FILE':    'auto_test.v',
'DUMPFILE':          'out.vcd',
'DUMPVARS':          '(0,tb)',
'TIMESCALE':         '1ns / 10ps',
'TIMEOUT':           '40000000',
'RESET':             '',
'FINISH':            '$finish',
'TIMEOUT_ERROR':     '',
    # Simulator Specific Options
'COMPCMD':           'iverilog',
'SIMCMD':            'vvp',
'WARN':              'all',
```

5.3 Declaring Tests

Tests are simple collections of Verilog tasks. Any task that you have defined in your testbench can be run from a test. Declaring a test with a single tasks would look like the following.

```
[basic_test]
TASKS = tb.basic
```

Tests can have more than one task too.

```
[regression]
TASKS = tb.basic
        tb.set_rate(1)
        tb.send_data
        tb.set_rate(3)
        tb.send_data
```

Tests can also call other tests.

```
[basic1]
TASKS = tb.basic1

[basic2]
TASKS = tb.basic2

[basic3]
TASKS = tb.basic3

[regression]
TASKS = [basic1] [basic2] [basic3]
```

Only 1 level of recursion is currently allowed.

5.4 Example

Here's a full configuration file named v.cfg that defines the required entries as well as some optional entries and defines some tests.

```
[DEFAULT]
PROJ_ROOT = ../../

RTL_FILES = rtl/and_nand.v
            rtl/or_nor.v

DUMPVARS = (0,tb)

TEST_FILES =    test/variant0/tb.v

TEST_INC_DIRS = test/variant0/

[basic1]
TASKS = tb.basic1

[basic2]
TASKS = tb.basic2

[basic3]
TASKS = tb.basic3

[regression]
TASKS = [basic1] [basic2] [basic3]
```

Listing available tests with SimShop:

```
$ shop -l
Found the following config files
--------------------------------
./v.cfg

./
    basic1
    basic2
    basic3
    regression

To run a simulation:
```

```
shop <path_to/variant>/<test>

Example:
    shop regression
```

Running the regression test.

```
$ shop regression

Verifying target...
  PATH    : ./
  VARIANT : variant0
  TEST    : regression


Generating auto test file based on test 'regression'

Removing old build directory: simbuild/regression

Making new build directory: simbuild/regression

iverilog -Wall -osimbuild/regression/sim -I../../test/variant0/ ../../test/variant0/tb.v ../../rtl/ar
vvp -n -lsimbuild/regression/sim.log simbuild/regression/sim

<0> Dump file set to simbuild/regression/out.vcd.
<0> Dumping has been turned OFF. Nothing will be dumped.

<0> Starting Auto Tests
Task: basic1
Task: basic2
Task: basic3


Simulation Score
`-- variant0
    `-- regression              [PASS]  (00h 00m 00s)

Passed      1/1 (100.0%)
Failed      0/1 (0.0%)
Invalid     0
Incomplete  0
Not Run     0
Errors      0
Warnings    0
Run Time    00h 00m 00s
```

6 Resource Configuration Files

The simshoprc is a standard ConfigParser formatted file where SimShop customizations are defined. SimShop looks for configuration files in predefined places on the file system.

- **OSX/Linux**
    1. Command line : –config-file=my.ini
    2. User : ~/.simshoprc”
    3. System Wide : /etc/simshop/simshoprc

- **Windows**
    1. Command line : –config-file=my.ini
    2. User : %USERPROFILE%\simshoprc
    3. System Wide : %ALLUSERSPROFILE%\simshoprc

SimShop will read any and all of these files, if they exist, starting with the system wide file, followed by the user file and finally any file passed on the command line. The most recently read file will overwrite any options that were defined in previous configuration files. This is useful when, for instance, a group wants to use a common email distribution list for receiving simulation results but for a test simulation an individual user may want the results to be emailed only to them. That user can override the email settings by either creating a simshoprc in their home directory or passing a temporary one on the command line. The users custom configuration file only needs to contain the items that he would like to override, not every item in the system wide configuration file.

6.1 [scoreboard]

Settings used by the scoreboard.

errors
Any number of regular expressions used to determine what an error string looks like. All strings are treated as Python raw strings internally. i.e. r'ERROR'

```
errors = ERROR
```

warnings
Any number of regular expressions used to determine what a warning string looks like. All strings are treated as Python raw strings internally. i.e. r'WARNING'

```
warnings = WARNING WARN
```

test_begin
A single string that defines the beginning of a test.

```
test_begin = TEST_BEGIN
```

test_end
A single string that defines the ending of a test.

```
test_end = TEST_END
```

6.2 [email]

Settings which allow the simulation scoreboard report to be emailed to any number of recipients.

to
Any number of email address to which a simulation report should be sent separated by a space and/or newline.

```
to = bill@ourserver.com ted@ourserver.com station@ourserver.com
```

or

```
to = bill@ourserver.com
     ted@ourserver.com
     station@ourserver.com
```

from
A single email address from which the email will be sent.

```
from = git@our_gmail_server.com
```

subject
A custom subject message. Some automatic string substitution is available which SimShop will replace when the email is actually sent. For instance, the status of the simulation can be replaced as part of the subject line.

```
subject = My simulation - $status
```

The $status message will be replaced with either PASS or FAIL depending on the result of the simulation.

Available string substitutions are:

$status
- PASS
- FAIL

password
The password for the email account from which the email will be sent.

smtp_server
The SMTP server from which the email will be sent.

```
smtp_server = smtp.gmail.com
```

smtp_server_port
The SMTP server port from which the email will be sent.

```
smtp_server_port = 587
```

Index

Symbols

C