SpecSim User Manual

— Version 1.0 —

A Simulation Engine for the SpecC Language

Jianwen Zhu
Electrical and Computer Engineering
University of Toronto

Contents

1 Introduction

SpecSim is a system level simulator designed as the runtime system of the SpecC (`http://www.ics.uci.edu/ specc`) language. The SpecC language is targeted toward System-on-chip design and is being promoted by the organization called SpecC Technology Open Consortium (STOC) (`http://www.specc.org`).

The SpecC simulator is an essential part of the SpecC language since it implements the extra semantics that SpecC imposes upon the ANSI-C language.

2

Installation Guide

Names

2.1

Basic Installation

Names

These are generic installation instructions.

The ‘configure’ shell script attempts to guess correct values for various system-dependent variables used during compilation. It uses those values to create a ‘Makefile’ in each directory of the package. It may also create one or more ‘.h’ files containing system-dependent definitions. Finally, it creates a shell script ‘config.status’ that you can run in the future to recreate the current configuration, a file ‘config.cache’ that saves the results of its tests to speed up reconfiguring, and a file ‘config.log’ containing compiler output (useful mainly for debugging ‘configure’).

If you need to do unusual things to compile the package, please try to figure out how ‘configure’ could check whether to do them, and mail diffs or instructions to the address given in the ‘README’ so they can be considered for the next release. If at some point ‘config.cache’ contains results you don’t want to keep, you may remove or edit it.

The file ‘configure.in’ is used to create ‘configure’ by a program called ‘autoconf’. You only need ‘configure.in’ if you want to change it or regenerate ‘configure’ using a newer version of ‘autoconf’.

The simplest way to compile this package is:

1. ‘cd’ to the directory containing the package’s source code and type ‘./configure’ to configure the package for your system. If you’re using ‘csh’ on an old version of System V, you might need to type ‘sh ./configure’ instead to prevent ‘csh’ from trying to execute ‘configure’ itself.

   Running ‘configure’ takes awhile. While running, it prints some messages telling which features it is checking for.

2. Type ‘make’ to compile the package.

3. Optionally, type ‘make check’ to run any self-tests that come with the package.

4. Type ‘make install’ to install the programs and any data files and documentation.

5. You can remove the program binaries and object files from the source code directory by typing ‘make clean’. To also remove the files that ‘configure’ created (so you can compile the package for a different kind of computer), type ‘make distclean’. There is also a ‘make maintainer-clean’ target, but that is intended mainly for the package’s developers. If you use it, you may have to get all sorts of other programs in order to regenerate files that came with the distribution.

2.1.1 Compilers and Options

Some systems require unusual options for compilation or linking that the ‘configure’ script does not know about. You can give ‘configure’ initial values for variables by setting them in the environment. Using a Bourne-compatible shell, you can do that on the command line like this: CC=c89 CFLAGS=-O2 LIBS=-lposix ./configure

Or on systems that have the ‘env’ program, you can do it like this: env CPPFLAGS=-I/usr/local/include LDFLAGS=-s ./configure

2.1.2

Compiling For Multiple Architectures

You can compile the package for more than one kind of computer at the same time, by placing the object files for each architecture in their own directory. To do this, you must use a version of ‘make’ that supports the ‘VPATH’ variable, such as GNU ‘make’. ‘cd’ to the directory where you want the object files and executables to go and run the ‘configure’ script. ‘configure’ automatically checks for the source code in the directory that ‘configure’ is in and in ‘..’.

If you have to use a ‘make’ that does not supports the ‘VPATH’ variable, you have to compile the package for one architecture at a time in the source code directory. After you have installed the package for one architecture, use ‘make distclean’ before reconfiguring for another architecture.

2.1.3

Installation Names

By default, ‘make install’ will install the package’s files in ‘/usr/local/bin’, ‘/usr/local/man’, etc. You can specify an installation prefix other than ‘/usr/local’ by giving ‘configure’ the option ‘–prefix=PATH’.

You can specify separate installation prefixes for architecture-specific files and architecture-independent files. If you give ‘configure’ the option ‘–exec-prefix=PATH’, the package will use PATH as the prefix for installing programs and libraries. Documentation and other data files will still use the regular prefix.

In addition, if you use an unusual directory layout you can give options like ‘–bindir=PATH’ to specify different values for particular kinds of files. Run ‘configure –help’ for a list of the directories you can set and what kinds of files go in them.

If the package supports it, you can cause programs to be installed with an extra prefix or suffix on their names by giving ‘configure’ the option ‘–program-prefix=PREFIX’ or ‘–program-suffix=SUFFIX’.

2.1.4

Optional Features

Some packages pay attention to '–enable-FEATURE' options to 'configure', where FEATURE indicates an optional part of the package. They may also pay attention to '–with-PACKAGE' options, where PACKAGE is something like 'gnu-as' or 'x' (for the X Window System). The 'README' should mention any '–enable-' and '–with-' options that the package recognizes.

For packages that use the X Window System, 'configure' can usually find the X include and library files automatically, but if it doesn't, you can use the 'configure' options '–x-includes=DIR' and '–x-libraries=DIR' to specify their locations.

2.1.5

Sharing Defaults

If you want to set default values for 'configure' scripts to share, you can create a site shell script called 'config.site' that gives default values for variables like 'CC', 'cache-file', and 'prefix'. 'configure' looks for 'PREFIX/share/config.site' if it exists, then 'PREFIX/etc/config.site' if it exists. Or, you can set the 'CONFIG_SITE' environment variable to the location of the site script. A warning: not all 'configure' scripts look for a site script.

2.1.6

Operation Controls

'configure' recognizes the following options to control how it operates.

'–cache-file=FILE'

Use and save the results of the tests in FILE instead of './config.cache'. Set FILE to '/dev/null' to disable caching, for debugging 'configure'.

'–help'

Print a summary of the options to 'configure', and exit.

'–quiet'

'–silent'

'-q'

Do not print messages saying which checks are being made.

'–srcdir=DIR'

Look for the package's source code in directory DIR. Usually 'configure' can determine that directory automatically.

'–version'

Print the version of Autoconf used to generate the 'configure' script, and exit.

'configure' also accepts some other, not widely useful, options.

2.2 Configuring SpecSim

The following options are used to configure SpecSim:

'–with-scc=path'

Specifies SpecC compiler installation location.

'–with-qthread=path' Use Quick Thread Library and specifies its installation location. By default, it is located in the SpecC directory.

'–with-pthread1'

Use Posix Thread Library implementation alternative 1.

'–with-pthread2'

Use Posix Thread Library implementation alternative 2.

'–with-win32'

Use Win32 Native Thread Library.

'–with-www=path'

Specifies HTML installation directory.

'–enable-debug'
: Build with debug information.

2.3 Typical Unix Session

```
>gtar xvfz speccsim-1_0.tar.gz
>cd speccsim-1_0
>automake
>autoconf
>./configure --enable-debug --with-scc=/home/specc/sce/current \
--with-pthread1 -with--www=${PWD}/www
>gmake all
```

3

Porting SpecSim

Names

SpecSim is designed with three layers of API to simplify the job of porting to different development platforms. Most users can directly use SpecSim since it has been ported to most commonly used platforms (Sun Solaris, Linux x86 and Windows x86). Some users may choose to implement their own native thread API (`../native`), who can safely leave the rest two layers unaffected. Some users who wish to improve the simulation performance may choose to re-implement the discrete event API (`../de`).

SpecSim is ported to different platforms by implementing the native API using different thread libraries. In theory, SpecSim can be ported to any platform where a given thread library is ported. In reality, due to their nature of architecture dependency, the thread libraries, especially those without commercial support, are unreliable on some claimed platforms. Hence, it is safe to say SpecSim works on a particular platform with a particular native API implementation only when it is fully tested. The SpecSim maintainer welcomes test reports for untested platforms.

3.1

Thread Libraries

As of now, SpecSim release contains four native API implementations.

The first implementation (qthread.c), uses the qthread library (`ftp://ftp.cs.washington.edu/pub/qt-002.tar.gz`) developed by David Keppel at University of Washington many years back. As its name suggests, the main advantage of using qthread is that it is fast and small. However, it doesn't seem to work on windows platform, and it doesn't work advanced debug tool such as purify.

The second implementation (pthread1.c), uses the posix thread. Since it is a standard, the obvious advantage is that it is well-documented and they are supposed to work for all Unix platforms. The bad news is that Redhat 6.2 has a reported problem for its pthread, probably due to a problem of libc.a; and the cygwin implementation for Win32 is not yet complete.

The third implementation (pthread1.c), again uses the posix thread. The difference from the previous one is that it tries to exploit the fact the posix threads can run in parallel (not just concurrently) on a multi-processor machine, so as to improve simulation speed.

The fourth implementation (win32.c), uses the Win32 kernel API to manipulate threads.

3.2 Test Results

We have tested SpecSim on the following platforms that are available to us: Solaris 2.7 on Ultra 5 workstation, Redhat Linux 6.2 (`http://www.redhat.com`) on Dell Pentium PC, Window98 on Dell Pentium PC. The following table shows the result for different native API implementations. In the table S means succeeded for the test suits; F means failure due to problem of underlying thread library; and B means failure due to bugs in our implementation.

platform	qthread	pthread1	pthread2
solaris on sparc	S	S	S
linux on x86	S	F	F

4 Known Bugs

Number	Date	Reporter	Version	Platform	Description	Status

Class Graph