SimpleScalar Version 3.0e
Updated Jan 7, 2014

This document will basically go over how to create a SimpleScalar 3.0e folder in your home directory and test it for use.

1. To create the simple scalar folder SScalar3.0e in your home directory issue the command:

   ```
   SScalarsetup3.0e
   ```

   From within any terminal window. You should see a flurry of text coming to the terminal screen. This is essentially creating what we call a shadow directory within your home directory. It's called a shadow directory as the files do not actually exist in your home directory but an elaborate set of links to a central repository that stores the original sources.

2. Now after the setup command is issued you should have a folder within your home directory called:

   ```
   SScalar3.0e
   ```

   So let's say you want to edit the cache.c file that is apart of the simplesim3.0e simulator. To do so you would have to delete the link that presently points from cache.c to the central repository. Then copy a version into the same location such that you can now edit the file, as you would not have been able to edit the existing link. So essentially from a terminal window we would issue:

   ```
   cd ~/SScalar3.0e/simplesim-3.0
   ls -l cache.c

   lrwxrwxrwx 1 jnaughto sysadmin 50 Jan  7 17:28 cache.c ->
   /usr/local/SimpleScaler-3.0e/simplesim-3.0/cache.c
   ```

   As you can see here the cache.c file is a link (ie the -> symbol) to the original read only file that resides in /usr/local/SimpleScaler-3.0e/simplesim-3.0/cache.c. So we're going to delete the link now and copy the source file into the same location.

   ```
   cd ~/SScalar3.0e/simplesim-3.0
   rm cache.c
   cp /usr/local/SimpleScalar-3.0e/simplesim-3.0/cache.c .
   ```

   Notice the "." at the end of the copy command. It tells the operating system to place the copy in the folder that you presently reside in. So now we can edit our copy of the cache.c file and rebuild the simplesim simulator binaries that reside in this folder.

3. Once you have edited cache.c file we need to rebuild the simulator. To simplify this process there is already a jsetup script in the simplesim-3.0 folder to allow you to recompile the simplesim simulator. To do this you simply need to:

   ```
   cd ~/SScalar3.0d/simplesim-3.0
   ./jsetup
   ```

This should generate the following output:

```
rm -f config.h machine.h machine.c machine.def loader.c symbol.c syscall.c
ln -s target-pisa/config.h config.h
ln -s target-pisa/pisa.h machine.h
ln -s target-pisa/pisa.c machine.c
ln -s target-pisa/pisa.def machine.def
ln -s target-pisa/loader.c loader.c
ln -s target-pisa/symbol.c symbol.c
ln -s target-pisa/syscall.c syscall.c
rm -f tests
ln -s tests-pisa tests
gcc34 `./sysprobe -flags` -DDEBUG -O0 -g -Wall   -c cache.c
gcc34 -o sim-cache `./sysprobe -flags` -DDEBUG -O0 -g -Wall   sim-cache.o cache.o main.o
syscall.o memory.o regs.o loader.o endian.o dlite.o symbol.o eval.o options.o stats.o
eio.o range.o misc.o machine.o libexo/libexo.a `./sysprobe -libs` -lm
gcc34 -o sim-outorder `./sysprobe -flags` -DDEBUG -O0 -g -Wall   sim-outorder.o cache.o
bpred.o resource.o ptrace.o main.o syscall.o memory.o regs.o loader.o endian.o dlite.o
symbol.o eval.o options.o stats.o eio.o range.o misc.o machine.o libexo/libexo.a
`./sysprobe -libs` -lm
my work is done here…
```

4. So once you get the “my work is done here” part you should be ready to test your newly compiled simulator. You will want to access the sim-safe and the sim-fast programs from any directory. Every time you open a terminal you will have to issue the following command once:

```
                  SScalarenv3.0e
```

   It should nicely place you inside the simplesim folder that resides in your home directory. It will generate the following output after you’ve

```
                Setting up Simple Scalar Environment…

                  Simple Scalar Version 3.0e now setup.
```

5. You will only be able to run the environment setting script once. If you run the environment script twice in the same terminal window it will tell you that the environment is already setup.

Example Hello World Code Test

So after the environment has been setup in your home directory you can test it simply by compiling the helloworld example. For example:

```
#include<stdio.h>
int main() {
       printf("Hello World\n");
       return(0);
}
```

Using any editor we've created a file called helloworld.c which contains the above.   The commands that you see below is all done from a terminal window.

```
jnaughto@opera: ~$ cd SScalar3.0e
jnaughto@opera: ~/SScalar3.0e$ ls
bin/  include/      info/  lib/  man/  share/  simplesim-3.0/  sslittle-na-sstrix/
```

Setup the Environment so that we can use all the simple sim tools from any terminal prompt:

```
jnaughto@opera: ~/SScalar3.0e$ SScalarenv3.0e
Setting up Simple Scalar Environment...

Simple Scalar Version 3.0e now setup.
```

So we have our helloworld.c file, let's compile it:

```
jnaughto@opera: ~/SScalar3.0e$ sslittle-na-sstrix-gcc -o helloworld helloworld.c
jnaughto@opera: ~/SScalar3.0e$ ls
bin/        helloworld.c  info/  man/     simplesim-3.0/
helloworld*  include/         lib/   share/  sslittle-na-sstrix/
```

Notice the helloworld file above that wasn't there prior? Well that's our compiled code which now we want to pass to sim-safe:

```
jnaughto@opera: ~/SScalar3.0e$ sim-safe helloworld
sim-safe: SimpleScalar/PISA Tool Set version 3.0 of August, 2003.
Copyright (c) 1994-2003 by Todd M. Austin, Ph.D. and SimpleScalar, LLC.
All Rights Reserved. This version of SimpleScalar is licensed for academic
non-commercial use.  No portion of this work may be used by any commercial
entity, or for any commercial purpose, without the prior written permission
of SimpleScalar, LLC (info@simplescalar.com).

sim: command line: sim-safe helloworld

sim: simulation started @ Tue Jan  7 20:54:06 2014, options follow:

sim-safe: This simulator implements a functional simulator.  This
```

```
functional simulator is the simplest, most user-friendly simulator in the
simplescalar tool set.  Unlike sim-fast, this functional simulator checks
for all instruction errors, and the implementation is crafted for clarity
rather than speed.

# -config                     # load configuration from a file
# -dumpconfig                 # dump configuration to a file
# -h                    false # print help message
# -v                    false # verbose operation
# -d                    false # enable debug message
# -i                    false # start in Dlite debugger
-seed                       1 # random number generator seed (0 for timer seed)
# -q                    false # initialize and terminate immediately
# -chkpt               <null> # restore EIO trace execution from <fname>
# -redir:sim           <null> # redirect simulator output to file (non-interactive only)
# -redir:prog          <null> # redirect simulated program output to file
-nice                       0 # simulator scheduling priority
-max:inst                   0 # maximum number of inst's to execute

sim: ** starting functional simulation **
Hello World


sim: ** simulation statistics **
sim_num_insn                   7864 # total number of instructions executed
sim_num_refs                   4304 # total number of loads and stores executed
sim_elapsed_time                  1 # total simulation time in seconds
sim_inst_rate             7864.0000 # simulation speed (in insts/sec)
ld_text_base             0x00400000 # program text (code) segment base
ld_text_size                  72176 # program text (code) size in bytes
ld_data_base             0x10000000 # program initialized data segment base
ld_data_size                   8304 # program init'ed `.data' and uninit'ed `.bss' size in bytes
ld_stack_base            0x7fffc000 # program stack segment base (highest address in stack)
ld_stack_size                 16384 # program initial stack size
ld_prog_entry            0x00400140 # program entry point (initial PC)
ld_environ_base          0x7fff8000 # program environment base address address
ld_target_big_endian              0 # target executable endian-ness, non-zero if big endian
mem.page_count                   26 # total number of pages allocated
mem.page_mem                   104k # total size of memory pages allocated
mem.ptab_misses                  26 # total first level page table misses
mem.ptab_accesses            491804 # total page table accesses
mem.ptab_miss_rate           0.0001 # first level page table miss rate
```

Okay that’s it we have a functional simplescalar 3.0e environment setup.