SimpleScalar
Reference Information

sim-safe

```
Usage: sim-safe {-options} executable {arguments}

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.

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

sim-fast

```
Usage: sim-fast {-options} executable {arguments}

sim-fast: This simulator implements a very fast functional simulator.  This functional simulator implementation is
much more difficult to digest than the simpler, cleaner sim-safe functional simulator.  By default, this simulator
performs no instruction error checking, as a result, any instruction errors will manifest as simulator execution
errors, possibly causing sim-fast to execute incorrectly or dump core.  Such is the price we pay for speed!!!!

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

sim-cache

```
Usage: sim-cache {-options} executable {arguments}

sim-cache: This simulator implements a functional cache simulator.  Cache statistics are generated for a user-
selected cache and TLB configuration, which may include up to two levels of instruction and data cache (with any
levels unified), and one level of instruction and data TLBs.  No timing information is generated.

#
# -option        <args>           #    <default> # description
#
-config          <string>         #       <null> # load configuration from a file
-dumpconfig      <string>         #       <null> # dump configuration to a file
-h               <true|false>     #        false # print help message
-v               <true|false>     #        false # verbose operation
-d               <true|false>     #        false # enable debug message
-i               <true|false>     #        false # start in Dlite debugger
-seed            <int>            #            1 # random number generator seed (0 for timer seed)
-q               <true|false>     #        false # initialize and terminate immediately
-chkpt           <string>         #       <null> # restore EIO trace execution from <fname>
-redir:sim       <string>         #       <null> # redirect simulator output to file (non-interactive only)
-redir:prog      <string>         #       <null> # redirect simulated program output to file
-nice            <int>            #            0 # simulator scheduling priority
-max:inst        <uint>           #            0 # maximum number of inst's to execute
-cache:dl1       <string>         # dl1:256:32:1:l # l1 data cache config, i.e., {<config>|none}
-cache:dl2       <string>         # ul2:1024:64:4:l # l2 data cache config, i.e., {<config>|none}
-cache:il1       <string>         # il1:256:32:1:l # l1 inst cache config, i.e., {<config>|dl1|dl2|none}
-cache:il2       <string>         #          dl2 # l2 instruction cache config, i.e., {<config>|dl2|none}
-tlb:itlb        <string>         # itlb:16:4096:4:l # instruction TLB config, i.e., {<config>|none}
-tlb:dtlb        <string>         # dtlb:32:4096:4:l # data TLB config, i.e., {<config>|none}
-flush           <true|false>     #        false # flush caches on system calls
-cache:icompress <true|false>     #        false # convert 64-bit inst addresses to 32-bit inst equivalents
-pcstat          <string list...> #       <null> # profile stat(s) against text addr's (mult uses ok)
```

```
The cache config parameter <config> has the following format:

  <name>:<nsets>:<bsize>:<assoc>:<repl>

  <name>   - name of the cache being defined
  <nsets>  - number of sets in the cache
  <bsize>  - block size of the cache
  <assoc>  - associativity of the cache
  <repl>   - block replacement strategy, 'l'-LRU, 'f'-FIFO, 'r'-random

  Examples:   -cache:dl1 dl1:4096:32:1:l
              -dtlb dtlb:128:4096:32:r

Cache levels can be unified by pointing a level of the instruction cache
hierarchy at the data cache hiearchy using the "dl1" and "dl2" cache
configuration arguments.  Most sensible combinations are supported, e.g.,

  A unified l2 cache (il2 is pointed at dl2):
    -cache:il1 il1:128:64:1:l -cache:il2 dl2
    -cache:dl1 dl1:256:32:1:l -cache:dl2 ul2:1024:64:2:l

  Or, a fully unified cache hierarchy (il1 pointed at dl1):
    -cache:il1 dl1
    -cache:dl1 ul1:256:32:1:l -cache:dl2 ul2:1024:64:2:l
```

sim-profile

```
Usage: sim-profile {-options} executable {arguments}

sim-profile: This simulator implements a functional simulator with profiling support.  Run with the `-h' flag to see
profiling options available.

#
# -option        <args>           #    <default> # description
#
-config          <string>         #       <null> # load configuration from a file
-dumpconfig      <string>         #       <null> # dump configuration to a file
-h               <true|false>     #        false # print help message
-v               <true|false>     #        false # verbose operation
-d               <true|false>     #        false # enable debug message
-i               <true|false>     #        false # start in Dlite debugger
-seed            <int>            #            1 # random number generator seed (0 for timer seed)
-q               <true|false>     #        false # initialize and terminate immediately
-chkpt           <string>         #       <null> # restore EIO trace execution from <fname>
-redir:sim       <string>         #       <null> # redirect simulator output to file (non-interactive only)
-redir:prog      <string>         #       <null> # redirect simulated program output to file
-nice            <int>            #            0 # simulator scheduling priority
-max:inst        <uint>           #            0 # maximum number of inst's to execute
-all             <true|false>     #        false # enable all profile options
-iclass          <true|false>     #        false # enable instruction class profiling
-iprof           <true|false>     #        false # enable instruction profiling
-brprof          <true|false>     #        false # enable branch instruction profiling
-amprof          <true|false>     #        false # enable address mode profiling
-segprof         <true|false>     #        false # enable load/store address segment profiling
-tsymprof        <true|false>     #        false # enable text symbol profiling
-taddrprof       <true|false>     #        false # enable text address profiling
-dsymprof        <true|false>     #        false # enable data symbol profiling
-internal        <true|false>     #        false # include compiler-internal symbols during symbol profiling
-pcstat          <string list...> #       <null> # profile stat(s) against text addr's (mult uses ok)
```

sim-bpred

```
Usage: sim-bpred {-options} executable {arguments}

sim-bpred: This simulator implements a branch predictor analyzer.

#
# -option        <args>           #    <default> # description
#
-config          <string>         #       <null> # load configuration from a file
-dumpconfig      <string>         #       <null> # dump configuration to a file
-h               <true|false>     #        false # print help message
-v               <true|false>     #        false # verbose operation
-d               <true|false>     #        false # enable debug message
-i               <true|false>     #        false # start in Dlite debugger
-seed            <int>            #            1 # random number generator seed (0 for timer seed)
-q               <true|false>     #        false # initialize and terminate immediately
-chkpt           <string>         #       <null> # restore EIO trace execution from <fname>
-redir:sim       <string>         #       <null> # redirect simulator output to file (non-interactive only)
-redir:prog      <string>         #       <null> # redirect simulated program output to file
-nice            <int>            #            0 # simulator scheduling priority
-max:inst        <uint>           #            0 # maximum number of inst's to execute
-bpred           <string>         #        bimod # branch predictor type {nottaken|taken|bimod|2lev|comb}
-bpred:bimod     <int>            # 2048 # bimodal predictor config (<table size>)
-bpred:2lev      <int list...>    # 1 1024 8 0 # 2-level predictor config (<l1size> <l2size> <hist_size> <xor>)
-bpred:comb      <int>            # 1024 # combining predictor config (<meta_table_size>)
-bpred:ras       <int>            #            8 # return address stack size (0 for no return stack)
-bpred:btb       <int list...>    # 512 4 # BTB config (<num_sets> <associativity>)

  Branch predictor configuration examples for 2-level predictor:
    Configurations:   N, M, W, X
      N   # entries in first level (# of shift register(s))
      W   width of shift register(s)
      M   # entries in 2nd level (# of counters, or other FSM)
      X   (yes-1/no-0) xor history and address for 2nd level index
    Sample predictors:
      GAg     : 1, W, 2^W, 0
      GAp     : 1, W, M (M > 2^W), 0
      PAg     : N, W, 2^W, 0
      PAp     : N, W, M (M == 2^(N+W)), 0
      gshare  : 1, W, 2^W, 1
  Predictor `comb' combines a bimodal and a 2-level predictor.
```

sim-cheetah

```
Usage: sim-cheetah {-options} executable {arguments}

sim-cheetah: This program implements a functional simulator driver for Cheetah.  Cheetah is a cache simulation
package written by Rabin Sugumar and Santosh Abraham which can efficiently simulate multiple cache configurations in
a single run of a program.  Specifically, Cheetah can simulate ranges of single level set-associative and fully-
associative caches.  See the directory libcheetah/ for more details on Cheetah.

#
# -option        <args>           #    <default> # description
#
-config          <string>         #       <null> # load configuration from a file
-dumpconfig      <string>         #       <null> # dump configuration to a file
-h               <true|false>     #        false # print help message
-v               <true|false>     #        false # verbose operation
-d               <true|false>     #        false # enable debug message
-i               <true|false>     #        false # start in Dlite debugger
-seed            <int>            #            1 # random number generator seed (0 for timer seed)
-q               <true|false>     #        false # initialize and terminate immediately
-chkpt           <string>         #       <null> # restore EIO trace execution from <fname>
-redir:sim       <string>         #       <null> # redirect simulator output to file (non-interactive only)
-redir:prog      <string>         #       <null> # redirect simulated program output to file
-nice            <int>            #            0 # simulator scheduling priority
-max:inst        <uint>           #   2147483647 # maximum number of inst's to execute
-refs            <string>         #         data # reference stream to analyze, i.e., {none|inst|data|unified}
-R               <string>         #          lru # replacement policy, i.e., lru or opt
-C               <string>         #           sa # cache configuration, i.e., fa, sa, or dm
-a               <int>            #            7 # min number of sets (log base 2, line size for DM)
-b               <int>            #           14 # max number of sets (log base 2, line size for DM)
-l               <int>            #            4 # line size of the caches (log base 2)
-n               <int>            #            1 # max degree of associativity to analyze (log base 2)
-in              <int>            #          512 # cache size intervals at which miss ratio is shown
-M               <int>            #       524288 # maximum cache size of interest
-c               <int>            #           16 # size of cache (log base 2) for DM analysis
```

sim-eio

```
Usage: sim-eio {-options} executable {arguments}

sim-eio: This simulator implements simulator support for generating external event traces (EIO traces) and checkpoint
files.  External event traces capture one execution of a program, and allow it to be packaged into a single file for
later re-execution.  EIO trace executions are 100% reproducible between subsequent executions (on the same platform.
This simulator also provides functionality to generate checkpoints at arbitrary points within an external event trace
(EIO) execution.  The checkpoint file (along with the EIO trace) can be used to start any SimpleScalar simulator in
the middle of a program execution.

#
# -option        <args>           #    <default> # description
#
-config          <string>         #       <null> # load configuration from a file
-dumpconfig      <string>         #       <null> # dump configuration to a file
-h               <true|false>     #        false # print help message
-v               <true|false>     #        false # verbose operation
-d               <true|false>     #        false # enable debug message
-i               <true|false>     #        false # start in Dlite debugger
-seed            <int>            #            1 # random number generator seed (0 for timer seed)
-q               <true|false>     #        false # initialize and terminate immediately
-chkpt           <string>         #       <null> # restore EIO trace execution from <fname>
-redir:sim       <string>         #       <null> # redirect simulator output to file (non-interactive only)
-redir:prog      <string>         #       <null> # redirect simulated program output to file
-nice            <int>            #            0 # simulator scheduling priority
-max:inst        <uint>           #            0 # maximum number of inst's to execute
-fastfwd         <int>            #            0 # number of insts skipped before tracing starts
-trace           <string>         #       <null> # EIO trace file output file name
-perdump         <string list...> #       <null> # periodic checkpoint every n instructions: <base fname> <interval>
-dump            <string list...> #       <null> # specify checkpoint file and trigger: <fname> <range>

  Checkpoint range triggers are formatted as follows:

    {{@|#}<start>}:{{@|#|+}<end>}

  Both ends of the range are optional, if neither are specified, the range
  triggers immediately.  Ranges that start with a `@' designate an address
  range to trigger on, those that start with an `#' designate a cycle count
  trigger.  All other ranges represent an instruction count range.  The
  second argument, if specified with a `+', indicates a value relative
  to the first argument, e.g., 1000:+100 == 1000:1100.

    Examples:   -ptrace FOO.trc #0:#1000
                -ptrace BAR.trc @2000:
                -ptrace BLAH.trc :1500
                -ptrace UXXE.trc :
```

sim-outorder

```
Usage: sim-outorder {-options} executable {arguments}

sim-outorder: This simulator implements a very detailed out-of-order issue superscalar processor with a two-level
memory system and speculative execution support.  This simulator is a performance simulator, tracking the latency of
all pipeline operations.

#
# -option        <args>           #        <default> # description
#
-config          <string>         #           <null> # load configuration from a file
-dumpconfig      <string>         #           <null> # dump configuration to a file
-h               <true|false>     #            false # print help message
-v               <true|false>     #            false # verbose operation
-d               <true|false>     #            false # enable debug message
-i               <true|false>     #            false # start in Dlite debugger
-seed            <int>            #                1 # random number generator seed (0 for timer seed)
-q               <true|false>     #            false # initialize and terminate immediately
-chkpt           <string>         #           <null> # restore EIO trace execution from <fname>
-redir:sim       <string>         #           <null> # redirect simulator output to file (non-interactive only)
-redir:prog      <string>         #           <null> # redirect simulated program output to file
-nice            <int>            #                0 # simulator scheduling priority
-max:inst        <uint>           #                0 # maximum number of inst's to execute
-fastfwd         <int>            #                0 # number of insts skipped before timing starts
-ptrace          <string list...> #           <null> # generate pipetrace, i.e., <fname|stdout|stderr> <range>
-fetch:ifqsize   <int>            #                4 # instruction fetch queue size (in insts)
-fetch:mplat     <int>            #                3 # extra branch mis-prediction latency
-fetch:speed     <int>            #                1 # speed of front-end of machine relative to execution core
-bpred           <string>         #            bimod # branch predictor type {nottaken|taken|perfect|bimod|2lev|comb}
-bpred:bimod     <int>            #             2048 # bimodal predictor config (<table size>)
-bpred:2lev      <int list...>    #      1 1024 8 0 # 2-level predictor config (<l1size> <l2size> <hist_size> <xor>)
-bpred:comb      <int>            #             1024 # combining predictor config (<meta_table_size>)
-bpred:ras       <int>            #                8 # return address stack size (0 for no return stack)
-bpred:btb       <int list...>    #            512 4 # BTB config (<num_sets> <associativity>)
-bpred:spec_update <string>       #           <null> # speculative predictors update in {ID|WB} (default non-spec)
-decode:width    <int>            #                4 # instruction decode B/W (insts/cycle)
-issue:width     <int>            #                4 # instruction issue B/W (insts/cycle)
-issue:inorder   <true|false>     #            false # run pipeline with in-order issue
-issue:wrongpath <true|false>     #             true # issue instructions down wrong execution paths
-commit:width    <int>            #                4 # instruction commit B/W (insts/cycle)
-ruu:size        <int>            #               16 # register update unit (RUU) size
-lsq:size        <int>            #                8 # load/store queue (LSQ) size
-cache:dl1       <string>         #   dl1:128:32:4:l # l1 data cache config, i.e., {<config>|none}
-cache:dl1lat    <int>            #                1 # l1 data cache hit latency (in cycles)
-cache:dl2       <string>         #  ul2:1024:64:4:l # l2 data cache config, i.e., {<config>|none}
-cache:dl2lat    <int>            #                6 # l2 data cache hit latency (in cycles)
-cache:il1       <string>         #   il1:512:32:1:l # l1 inst cache config, i.e., {<config>|dl1|dl2|none}
```

```
-cache:il1lat    <int>           #               1 # l1 instruction cache hit latency (in cycles)
-cache:il2       <string>        #             dl2 # l2 instruction cache config, i.e., {<config>|dl2|none}
-cache:il2lat    <int>           #               6 # l2 instruction cache hit latency (in cycles)
-cache:flush     <true|false>    #           false # flush caches on system calls
-cache:icompress <true|false>    #           false # convert 64-bit inst addresses to 32-bit inst equivalents
-mem:lat         <int list...>   #            18 2 # memory access latency (<first_chunk> <inter_chunk>)
-mem:width       <int>           #               8 # memory access bus width (in bytes)
-tlb:itlb        <string>        # itlb:16:4096:4:l # instruction TLB config, i.e., {<config>|none}
-tlb:dtlb        <string>        # dtlb:32:4096:4:l # data TLB config, i.e., {<config>|none}
-tlb:lat         <int>           #              30 # inst/data TLB miss latency (in cycles)
-res:ialu        <int>           #               4 # total number of integer ALU's available
-res:imult       <int>           #               1 # total number of integer multiplier/dividers available
-res:memport     <int>           #               2 # total number of memory system ports available (to CPU)
-res:fpalu       <int>           #               4 # total number of floating point ALU's available
-res:fpmult      <int>           #               1 # total number of floating point multiplier/dividers available
-pcstat          <string list...> #          <null> # profile stat(s) against text addr's (mult uses ok)
-bugcompat       <true|false>    #           false # operate in backward-compatible bugs mode (for testing only)
```

  Pipetrace range arguments are formatted as follows:

    {{@|#}<start>}:{{@|#|+}<end>}

Both ends of the range are optional, if neither are specified, the entire execution is traced. Ranges that start with a `@' designate an address range to be traced, those that start with an `#' designate a cycle count range. All other range values represent an instruction count range. The second argument, if specified with a `+', indicates a value relative to the first argument, e.g., 1000:+100 == 1000:1100. Program symbols may be used in all contexts.

```
    Examples:   -ptrace FOO.trc #0:#1000
                -ptrace BAR.trc @2000:
                -ptrace BLAH.trc :1500
                -ptrace UXXE.trc :
                -ptrace FOOBAR.trc @main:+278
```

```
  Branch predictor configuration examples for 2-level predictor:
    Configurations:   N, M, W, X
      N   # entries in first level (# of shift register(s))
      W   width of shift register(s)
      M   # entries in 2nd level (# of counters, or other FSM)
      X   (yes-1/no-0) xor history and address for 2nd level index
    Sample predictors:
      GAg     : 1, W, 2^W, 0
      GAp     : 1, W, M (M > 2^W), 0
      PAg     : N, W, 2^W, 0
      PAp     : N, W, M (M == 2^(N+W)), 0
      gshare  : 1, W, 2^W, 1
  Predictor `comb' combines a bimodal and a 2-level predictor.

  The cache config parameter <config> has the following format:

    <name>:<nsets>:<bsize>:<assoc>:<repl>

    <name>   - name of the cache being defined
    <nsets>  - number of sets in the cache
    <bsize>  - block size of the cache
    <assoc>  - associativity of the cache
    <repl>   - block replacement strategy, 'l'-LRU, 'f'-FIFO, 'r'-random

    Examples:   -cache:dl1 dl1:4096:32:1:l
                -dtlb dtlb:128:4096:32:r

  Cache levels can be unified by pointing a level of the instruction cache hierarchy at the data cache hiearchy using
the "dl1" and "dl2" cache configuration arguments.  Most sensible combinations are supported, e.g.,

    A unified l2 cache (il2 is pointed at dl2):
      -cache:il1 il1:128:64:1:l -cache:il2 dl2
      -cache:dl1 dl1:256:32:1:l -cache:dl2 ul2:1024:64:2:l

    Or, a fully unified cache hierarchy (il1 pointed at dl1):
      -cache:il1 dl1
      -cache:dl1 ul1:256:32:1:l -cache:dl2 ul2:1024:64:2:l
```