UVSIM User Manual

Version 0.1

Lixin Zhang

(lizhang@cs.utah.edu)

UUCS-03-011

School of Computing

University of Utah

Salt Lake City, UT 84112, USA

March 13, 2003

Abstract

This document describes UVSIM — the UltraViolet Simulator. UVSIM is an execution-driven simulator modeling the Ultraviolet system, SGI's entry in DARPA's HPCS Initiative.

NOTE: *UVSIM is still at the early development stage. If the user encounters bugs in the simulator or errors in the manual or identifies any missing functionality critical to the HPCS-SGI project, please inform the author. This manual may not be updated on time as new features are being added to the simulator. Whenever there is a conflict or confusion, please refer to the source code.*

Contents

1 Overview

This section gives an overview of the simulated systems and the platforms on which UVSIM can run.

1.1 Key components of the simulated systems

UVSIM is coded in C++. Its current version can simulate either an O3000 system or an SN2 system. Key components modeled by UVSIM are as follows.

- **Instruction set:** MIPS64^{TM} [4]
- **Processor** [3]
  - R12000 for O3000 systems or R18000 for SN2 systems
  - Instruction pipeline
  - Branch prediction unit
  - Memory management unit and TLB
  - Primary instruction cache
  - Primary data cache
  - Unified secondary cache
  - System interface
- **System bus**
  - For O3000 systems, split-transaction and time-multiplexed cluster bus [3]
  - For SN2 systems, TF bus [6]
- **Hub** [6]
  - Frame, i.e., interface between the system bus and the hub.
  - PI (Processor Interface), only CRP (coherent request pipelines) is simulated; NCRP (non-coherent request pipelines) is not simulated
  - MD (Memory/Directory) [7]
  - NI (Network Interface)
- **DRAM:** SDRAM [2] or RDRAM [1]
- **Coherence protocol:** SN2-MIPS communication protocol [5]
- **Micro-kernel**
  - Exception vector, including TLB miss and page fault handlers
  - System calls that create processes and handle signals
  - A simple virtual memory model
- **UltraViolet features**

- Fine-grained GET/PUT operations
- Vector/Stream unit
- Active memory operations
- OpenMP support
- Prefetch empty & write allocate

1.2 Platforms supported

UVSIM has been tested on the following platforms:

- x86 running Linux 2.4.18,
- SGI MIPS system running IRIX64 6.5, and
- Sun SPARC workstation running Solaris 2.7 and 2.8.

1.3 Logistics

The website of UVSIM is at
`http://www.cs.utah.edu/impusle/uvsim/`
You can find all the latest news about UVSIM there. It is strongly recommended to check out this site before starting using UVSIM.

We have set up a pair of mailing lists, `{uvsim, uvsim-users}@cs.utah.edu`. The `uvsim@cs.utah.edu` email list is for people actively interested in developing or closely following the development of uvsim. Use this email list if you have any questions about UVSIM. The `uvsim-users@cs.utah.edu` is for those using the simulator, but not actively developing. Please see the UVSIM webpage about information on how to subscribe to them.

2 Using UVSIM

Section 2.1 describes how to get UVSIM. Section 2.2 describes how to build the simulator executables. Section 2.4 explains how to port applications to UVSIM. Section 2.5 describes how to run UVSIM. Section 2.6 describes the debugging support in UVSIM. Section 2.7 describes the statistics collected by UVSIM.

2.1 Getting UVSIM

There are two ways to get UVSIM. For people who will be developing UVSIM, please get the source code from the CVS depository. The CVS root of UVSIM is currently hosted on hpcscvs.sci.utah.edu. You will need to send the preferred username and your DSA public key to `map@cs.utah.edu` to add your name into the permission list. To access UVSIM cvs depository, set the CVSROOT and CVS_RSH as follows:

```
>setenv CVSROOT :ext:username@hpcscvs.sci.utah.edu:/hpcs/cvsroot
>setenv CVS_RSH ssh
```

To check out the source code of UVSIM, run

```
>cvs co uvsim
```

The CVS depository does not include any binaries. If you do not want to compile the micro-kernel and applications yourself, you can download the precompiled ones from the UVSIM website.

For people who will be only using UVSIM, please download the latest version from the UVSIM website directly. Untar the downloaded file creates directory `uvsim`.

Before using UVSIM_PATH, the user must set environment variable `UVSIM_PATH` to the path where UVSIM is installed, such as

```
>setenv UVSIM_PATH <path-where-uvsim-was-untarred>/uvsim
```

`uvsim` has the following subdirectories.

app-lib/ contains headers files and libraries used by applications.

apps/ contains example applications.

bin/ is where the simulator executables are stored.

doc/ contains this manual, a simple user guide, and the reference manual of UltraViolet-related function calls.

share/ contains generic makefiles with common definitions and rules.

src/ contains source files of each component and the micro-kernel.

2.2 Compiling UVSIM

The top-level makefile for compiling UVSIM is located in subdirectory `bin/`. It follows the GNU make format, so please use gmake for the compilation. To compile UVSIM, go to `bin/` and run "`make all`". It will generate two binaries — an optimized version named `uvsim` and a debugging version named `uvsim_d` in a subdirectory corresponding to the OS type of the host machine (`bin/$(OSTYPE)`). I have compiled and run UVSIM on the following three platforms.

Linux x86 with Linux. I used g++ version 2.95.1 for both the optimized and debugging versions.

IRIX64 SGI with IRIX64 6.5. I used MIPSpro C++ version 7.3 for the optimized version and g++ version 2.95.2 for the debugging version.

SunOS Sun SPARCstations with Solaris 2.7 or above. I used g++ version 2.95.2 for both the optimized and debugging versions.

Please note that the optimized version on IRIX64 is a 64-bit executable but the others are 32-bit executables. I have not found a way to generate 64-bit code using g++. If somebody figures out, please tell me (lizhang@cs.utah.edu). If the user prefers to use other compilers, he/she needs to edit `share/Makefile.defs`.

To simulate an O3K system or an SN2 system is controlled by compile-time option "`MODEL_O3K`". If it is defined, the generated uvsim simulates an O3K system. Otherwise, the generated uvsim simulates an SN2 system. To define it, one can either use compiler option "`-DMODEL_O3K`" or put "`#define MODEL_O3K`" in `src/main/simsys.h`.

For the simulator developers, it is recommended to use "`make default`" to generate executables `bin/uvsim.$(OSTYPE)` and `bin/uvsim_d.$(OSTYPE)` before the simulator reaches a stable state. If one is confident that the new changes are working properly, he can use "`make rebuild`" to put executables into `bin/$(OSTYPE)`. In this way, the previously-tested working versions in `bin/$(OSTYPE)` will not be overwritten by untested versions.

2.3 Compiling the micro-kernel

Go to `src/kernel`, run `make` and then `make install`. Note the micro-kernel can only be compiled on IRIX64 machines that can generate 64-bit MIPS-IV instruction set and has 64-bit static libraries.

2.4 Porting applications to the UVSIM

UVSIM can load application executables that run on real MIPS machines, but, to enable applications to interact with UVSIM dynamically, the user must link applications with library `app-lib/libapp.o`. This library contains UVSIM traps — functions that applications use to control and communicate with UVSIM. Applications need to include header file "app-lib/uvsim_app.h" to pick up the definition of the UVSIM traps.

sysclocks() returns the current simulator clock.

get_l2c_size() returns the size of the secondary cache.

get_l2c_linesize() returns the cache-line size of the secondary cache.

get_pid() returns the process id.

get_cpus_per_node() returns the number of processors in a node.

get_total_cpus() returns the number of processors in the simulated system.

uvsim_nap(int64 cycles) puts the calling processor to sleep for the specified number of cycles.

uvsim_getenv(char *name) gets an environment variable.

Statistics-related

The user can add the **newphase(phaseid)** and **endphase()** functions to indicate the start and end of an application phase. The **newphase()** function takes a single integer argument that represents the new phase number (the simulation starts in phase "0"). This function also clears out all current statistics. The **endphase()** function takes no arguments. This function prints out simulation statistics.

The function **StatReportAll()** prints out a detailed set of statistics collected. The function **StatClear-All()** clears all the statistics gathered.

Flush/purge/prefetch cache lines

fdcache(vaddr, len)	flushes the specified memory region from cache.
fdwcache(vaddr, len)	writes then flushes the specified memory region.
flushline(vaddr)	flushes the specified line.
flushlines(vaddr, lines)	flushes the specified lines.
prefetchline(vaddr)	prefetches the specified line.
prefetch2lines(vaddr)	prefetches the specified tow lines.
prefetch4lines(vaddr)	prefetches the specified four lines.
prefetch8lines(vaddr)	prefetches the specified eight lines.
prefetchlines(vaddr, lines)	prefetches the specified lines.

UVSIM also supports a variety of system calls and user functions that allow applications to access new features of the UltraViolet systems. Those functions are described in detail in the "UltraViolet Functions Supported by UVSIM", which can be found in the UVSIM website.

2.4.1 Supported system calls

Only common system calls are supported so far. They are either implemented by the micro-kernel or emulated by the UVSIM. For unsupported system calls, UVSIM simply prints a warning message and returns 0 to application. This method may fool some applications, but in general applications using unsupported system calls will not run on UVSIM. We have implemented the set of system calls that are enough to simulate the Impulse testing benchmarks. If the user finds that some other system calls are important and should be supported by UVSIM, please contact the author.

Currently, UVSIM supports the following system calls: *exit*, *fork*, *read*, *write*, *open*, *close*, *unlink*, *time*, *stat*, *lseek*, *getpid*, *access*, *kill*, *syssgi*, *dup*, *times*, *shmsys*, *ioctl*, *sysmp*, *fcntl*, *ulimit*, *lseek64*, *sginap*, *BSD_getime*, *procblk*, *prctl*, *sprocsp*, *mmap*, *msync*, *xstat*, *lxstat*, *fxstat*, and *waitsys*. Although these functions always return the correct values, they do not set **errno** on failure.

The OS supports the following system calls: *fork*, *sprocsp()*, *brk*, *utssys*, *mprotect*, *sysinfo*, *ksigaction*, *ksigprocmask*, and *signaltstack*.

2.4.2 Compiling applications

The directory `share/` includes a makefile `makefile_generic` that can be used to build applications written in C/C++ or assembly. To use the generic makefile, the makefile for an application must define the `SRC`, `HEADERS`, and `TARGET` variables. The generic makefile assumes that the application makefile is located in the top-level directory for a given application, that all source and header files are located in the `src/` directory of the application, that the object files will be placed in the `objs/` directory, and that the executables will be placed in the `execs/` directory. For example, if an application consists of the source files `src/source1.c` and `src/source2.c`, the header files `src/header1.h` and `src/header2.h`, and its executable is named as `appexec`, its makefile should simply looks as follows:

```
SRC     = source1.c source2.c
HEADERS = header1.h header2.h
TARGET  = appexec

include $(UVSIM_PATH)/share/makefile_generic
```

In addition. use `EFLAGS` to pass extra options to the compiler and use `LIBS` to set libraries that user applications need to link with.

The generic makefile assumes applications' source files contains normal codes intended to run on real hardware, base UVSIM codes intended to run on UVSIM without using the new Ultraviolet features, and optimized UVSIM codes intended to run on UVSIM with using the new Ultraviolet features. Normal codes and UVSIM codes are separated by directives regarding to macro "UVSIM", for example, "`#ifdef UVSIM`" or "`#ifndef UVSIM`". Base and optimized UVSIM codes are separated by directives regarding to macro "ULTRAVIOLET". The generic makefile seeks to produce three executables:

execs/appexec executable to run on hardware (compiled without "-DUVSIM" and "-DULTRAVIOLET"),

execs/appexec.sim executable to run on UVSIM, without using the Ultraviolet features (compiled with "-DUVSIM" but without "-DULTRAVIOLET"), and

execs/appexec_u.sim executable to run on UVSIM, with using the Ultraviolet features (compiled with "-DUVSIM -DULTRAVIOLET").

If one does not want to generate all three executables every time, he can use `make hardware` or `make sim` or `make sim_u` to generate the wanted one.

Compiling applications must happen on a IRIX64 machine that can generate MIPS-IV instruction set and has 64-bit static libraries installed. SGI no longer ships static libraries by default, please contact Kirk Kern (kern@sgi.com) if you do not have the required static libraries.

2.5 Running UVSIM

To run application `appexec.sim` on uvsim, just run (assuming `appexec.sim` is in the same directory as `uvsim`)

```
uvsim -f app-exec
```

For more advanced runs, please read Section 3.1.

2.6 Debugging UVSIM

UVSIM provides many compile-time and run-time options to support debugging.

2.6.1 Self-checking codes

UVSIM uses compile-time options to turn on/off self-checking codes.

DEBUG_MAIN	turns on self-checking codes in files of `main/`.
DEBUG_PROC	turns on self-checking codes in files of `proc/`.
DEBUG_TLB	turns on self-checking codes in files of `tlb/`.
DEBUG_CACHE	turns on self-checking codes in files of `cache/`.
DEBUG_VU	turns on self-checking codes in files of `vu/`.
DEBUG_BUS	turns on self-checking codes in files of `bus/`.
DEBUG_HUB	turns on self-checking codes in files of `hub/`.
DEBUG_DRAM	turns on self-checking codes in files of `dram/`.
DEBUG_ALL	turns on self-checking codes in all files.
DEBUG_VERBOSE	turns on heavy-duty time-consuming self-checking code in all files.

2.6.2 Warning messages

UVSIM can send a warning message to the simulator error output file when it encounters an abnormal but not fatal behavior. It ranks the warning messages from level 0 (most serious warning messages that will likely cause fatal problems) to level 3 (least serious warning messages that will not likely affect the simulation). Level 0 messages will always be sent to the simulator standard error output. The user can use "Warn_level" to control how many levels of warning messages to print (see Section 3.2.9 for more detail).

2.6.3 Reminding messages

UVSIM can send reminding messages to the simulator output file when some important events occur. Whether or not to print reminding messsages is controlled by parameter “Message_on”.

2.6.4 Execution trace

Instruction graduation trace Use command line option “`-G start,count[,trace-file]`”, which enables UVSIM to print out graduated instructions from No. `start` to No. `(start + count)` in the program order to either the standard output or the `trace-file`.

Memory access trace Use compile-time option `TRACE_MREQS` to include the memory access trace code and use run-time parameters `Mreq_trace_start`, `Mreq_trace_count`, and `Mreq_trace_file` to control it (see Section 3.2.9 for more detail). When the memory access trace is on, UVSIM will print out the life span of each traced access, including when it reaches the System Interface, System Bus, Frame, Coherent Request Pipeline, MD, Network Interface, DRAM scheduler, DRAM bank, and when its data/responses return to these units.

Cache access trace An old feature from URSIM. It sends a message to the standard output after the cache model receives a certain number of access requests. It shows whether or not the simulator is making any progress.

DRAM access trace An old feature from URSIM. Not useful for UVSIM.

2.6.5 Checkpoint

UVSIM supports the so-called *checkpoint*. The basic idea is to replace the old data area with the current data area in the original simulator executable. When the new executable is executed, the process will see the same data structures and data values that the original process had when checkpoint was generated. Therefore, the new executable starts simulation right at the cycle it was created. This is very useful when a bug comes out after a long period time of simulation.

The checkpoint is controlled by command line option “`-c int,file[,]`”, which create a new executable to `file` at cycle `int`. The optional “`,`” at the end controls whether or not to terminate the simulation after creating the checkpoint: with it, simulation stops; without it, simulation continues.

2.6.6 State check

There is a function in each module that prints out the entire state of that module.

- `ProcDump(proc_id)` for the processor.
- `MMUDump(proc_id)` for the MMU.
- `CacheDump(proc_id)` for the entire cache hierarchy.
- `SIDump(proc_id)` for the system interface.
- `VUDump(proc_id)` for the vector unit.
- `BusDump(node_id)` for the system bus.

- `HubDump(node_id)` for the Hub.
- `DramDump(node_id)` for the DRAMs.
- `MemoryDump(node_id, count)` for the entire memory hierarchy from MMU to DRAM in the specified node.
- `MemoryDumpAll()` for the entire memory hierarchy in each node.

These functions have been proven to be very useful. One of the common bugs is that some modules drop some transactions. Such a bug can be easily detected by looking at the dump output (e.g., if the cache shows a memory access has been sent to the Hub while the Hub does not seem to have this access, it must have been dropped by either the bus or the Hub.). Moreover, by comparing multiple dumps, we can easily detect whether or not the simulator has been making any progress.

Disclaimer: checkpoint does not work on Linux. If somebody figures out how to do it on Linux, please tell the author.

2.6.7 Other gadgets

- **PC(inst, proc)** prints the real PC (program counter) for the specified instance.
- **P2T(pc, proc)** translates real PC to internal PC.
- **GR(proc, reg)** prints the content of the specified int register.
- **GR(inst, reg)** prints the content of the specified int register.
- **FP(proc, reg)** prints the content of the specified fp register.
- **useless_i**, **useless_f**, **useless_d**, and **useless_ll** are global integer, float, double, and long long variables used when the debugger (gdb or dbx) needs to call functions with “pass-by-reference” argument(s).

2.7 Statistics collected

UVSIM provides a wide variety of statistics related to the processor, TLB, cache, vector unit, bus, Hub, and DRAM backend. UVSIM prints statistics to the simulation output file, which can be redirected through command line options. An application can use the phase-related and statistics-reporting UVSIM traps described in Section 2.4 to print statistics for the interested portions, rather than for the entire application at once.

2.8 Debugging applications

UVSIM does not currently include support for debugging application programs with a debugger like gdb or dbx, as UVSIM does not expose information about the application being simulated to such a debugger. If UVSIM encounters a non-recoverable exception (such as a segmentation fault or bus error), the simulator halts immediately and a termination message is printed on the standard error file. Application errors can be debugged either by running the applications natively, or by inserting printf calls into the application.

3 Configuring UVSIM

This section describes how to set configuration parameters in UVSIM using the command-line options and configuration files. All parameters can be presented to UVSIM via a parameter file. Parameters that are frequently changed can also be presented to UVSIM via the command line. When there is a conflict, command line option overrides parameter file. Different parameter files can be used for different simulation runs, as the name of the parameter file is passed to UVSIM on the command line.

3.1 Command line options

Command line options of UVSIM contain three parts: options for the simulator; options for the micro-kernel; and options for the application. Options for the micro-kernel are given after a double-dash and options for the application are given after another double-dash. If there is only one double-dash, all the options after it are passed into the application. For example, to simulate the application program `bstest` with an active list of size 64 and with the kernel option "`-a 10`" and the application options "`-n 0x100000 -s 32`", one would use the command line:

```
uvsim -a 64 -f bstest -- -a 10 -- -n 0x100000 -s 32
```

Running "`uvsim -h`" prints the full list of UVSIM options currently available. The following subsections describe those options in groups. In these subsections, **num** specifies a non-negative integer and **file** specifies a file name on the host file system.

3.1.1 General options

-f file Name of the application to run on the simulator, without the **.sim** suffix. Default is **test** in the current directory.

-k file Name of the micro-kernel. Default is **kernel** in the current directory.

-G num,count[,file] Starts to print out graduated instruction after **num** instructions have graduated, for the number of **count** instructions. If no **file** is specified, standard output will be used.

-c num,file[,] Creates a checkpoint at the **num**'th cycle to **file**. With a comma after **file**, simulation stops after the checkpoint; without it, simulation continues.

-t double Maximum number of cycles to simulate. Default is $3.4E + 38$.

3.1.2 I/O options

UVSIM uses the same standard error output for both the simulator and application, but distinguishes simulator input/output with application input/output. As a result, there are five IO streams associated with standard in/out/err. These streams can be redirected independently. By default, both simulator input and application input are standard input; and both simulator output and application output are standard output.

-0 file Redirects application standard input to **file**.

-1 file Redirects application standard output to **file**.

-2 file Redirects standard error to **file**. This is where the warning or error messages go to.

-3 file Redirects simulator output to **file**. This is where the statistics collected by UVSIM goes to.

-d dir Directory for output files. This option is effective only if used in conjunction with the "-s" option.

-s subj Subject to use in output filenames. This option overrides "-1", "-2", and "-3" options, and is used only in conjunction with "-d". When "`-d dir`" and "`-s subj`" are specified, UVSIM redirects application standard output to a file "`dir/subj`_out", redirects standard error to "`dir/subj`_err", and redirects simulator standard output to "`dir/subj`_stat".

-e emailaddr Sends an email notification to the specified address upon completion of the simulation.

-z file Redirects parameter file to **file**. Default is *uvsim_params* in current directory.

3.1.3 Processor options

-p num The number of processors the simulator will run on. Default is 1.

-a num Number of entries in the active list. Default is 48.

-i num Instruction decode rate. Default is 4.

-g num Instruction graduation rate. Default is the same value as the decode rate.

-q num,num,num Number of entries in the integer/floating-point/memory queues. Default is 16/16/16.

-u Turns on fast simulation mode, which completes all int/fp operations in one cycle. This option overrides any latencies specified in the configuration file. Default is off.

3.1.4 TLB options

-Ti Turns off the instruction TLB.

-Tn Number of entries in the TLB.

-Tw Number of entries to be wired by the kernel.

3.1.5 Cache options

-Cf num Cache frequency.

-Ce num DCA extension instruction.

-Cn Write no-allocate.

-Co Turns off memory access trace.

-Cm num size of trace samples.

-Cx num maximum number of cache accesses to trace.

-CI Simulates perfect I-cache.

-Ci num I-cache size in kilobytes.

-Cj num I-cache line size in bytes.

-Ck num I-cache set associativity.

-Ct Simulates perfect L1 cache.

-Cs num L1 cache size in kilobytes.

-Cl num L1 cache line size in bytes.

-Ca num L1 cache set associativity.

-Cp num L1 cache prefetch.
-CT Simulates perfect L2 cache.
-CS num L2 cache size in kilobytes.
-CL num L2 cache line size in bytes.
-CA num L2 cache set associativity.
-CP num L2 cache prefetch.

3.1.6 Vector unit options

-Vf num Vector unit frequency.
-Va num Number of address generators.
-Vc num Number of vector caches.
-Vr num Number of control register sets.
-Vt num Number of vector TLBs.
-Vs num Toggles statistics collection flag.

3.1.7 Bus options

-Bf num Bus frequency.
-Bw num SysT Bus width in bytes.
-BW num SysF Bus width in bytes.
-Bb num Arbitration delay.

3.1.8 Hub options

-Cf num Hub frequency.

3.1.9 DRAM options

-Do Turns off DRAM simulator.
-Df num DRAM frequency.
-Ds num Turns on/off DRAM scheduler.
-Dt num Turns on DRAM trace.
-Dm num Maximum number of accesses to trace.
-Db num Number of backend banks.
-Dc num Number of chips per backend bank.
-Dd num Number of banks per chip.
-Dw num DRAM data bus width in bytes.
-Di num Minimum access in bytes.
-Dl num Dram block size in bytes.
-Dh num Hot row open-close policy.

3.2 Parameter file

In the parameter file, blank lines and lines beginning with a "#" are ignored. In each line, characters after "#" are takes as comments and ignored. Each parameter in the input file will be followed by either an integer, or a float, or a string. The parameters are case sensitive. If any parameter is listed multiple times in the configuration file, the first one specifies the actual value used. The parameters that can be specified in the parameter file are given below.

3.2.1 System parameters

Num_of_nodes Number of nodes in the simulated system.

Cpus_per_node Number of processors in each node.

Warn_level Warning level. The higher the warning level, the more warning message will be printed.

Message_on Turns on/off messages that indicate progress or occurrence of important events.

Kernel_map_prints Number of times to print out mappings that the kernel created or destroyed.

Always_exit_on_err If set, the simulator exits whenever the simulated system encounters a fatal error. Defaults to 0.

Always_report_stat If set, report statistics even if the simulator terminates abnormally. Defaults to 0.

Always_print_config If set, the configuration information of each component will be sent out along with the statistics. Defaults to 1.

Collect_uvsim_stat If it is 2, output statistics for each component in the system; if it is 1, output summary statistics for all components; if it is 0, no statistics will be printed. Defaults to 1.

3.2.2 Processor parameters

procstaton Turns on/off statistics collection for the processor.

decode_rate Instruction decode rate.

graduate_rate Instruction graduation rate.

activelist Number of entries in the active list.

max_int_ops Number of entries in the integer queue.

max_fp_ops Number of entries in the floating-point queue.

max_mem_ops Number of entries in the memory queue.

numalus Number of integer functional units.

numfpus Number of FP functional units.

numaddrs Number of address generation units.

bpbsize Number of entries in the branch prediction buffer.

rassize Number of entries in the return address stack.

shadowmappers Number of shadow mappers in the branch prediction unit.

The following pairs of parameters specify the latencies and repeat rates of int and fp instructions. In each pair, the first element specifies the latency, while the second specifies the repeat rate. Note that in the parameter file, the latency and repeat rate for a special operation are specified in two lines. We put them into one item here just for simplicity.

latint,repint Latency and repeat rate for common INT operations — addition, subtraction, move, and logical operations.

latmuls,repmuls Latency and repeat rate for single-word integer multiply operations.

latmulsu,repmulsu Latency and repeat rate for unsigned single-word integer multiply operations.

latmuld,repmuld Latency and repeat rate for double-word integer multiply operations.

latmuldu,repmuldu Latency and repeat rate for unsigned double-word integer multiply operations.

latdivs,repdivs Latency and repeat rate for single-word integer division operations.

latdivd,repdivd Latency and repeat rate for double-word integer division operations.

latfp,repfp Latency and repeat rate for common FP operations.

latfpconv,repfpconv Latency and repeat rate for FP conversions (e.g., int to/from fp, fp to/from int, single to/from double).

latfdivs,repfdivs Latency and repeat rate for single-precision FP division.

latfdivu,repfdivu Latency and repeat rate for double-precision FP division.

latfsqrts,repfsqrts Latency and repeat rate for single-precision FP square-root operations.

latfsqrtd,repfsqrtd Latency and repeat rate for double-precision FP square-root operations.

3.2.3 TLB parameters

TLB_instr_on Turns on/off the instruction TLB.

TLB_num_entries Number of entries in the TLB.

TLB_wired Number of wired TLB entries.

3.2.4 Cache parameters

Cache_frequency Cache frequency relative to the processor core frequency.

Cache_collect_stats Turns on/off statistics collections for the cache module.

Cache_mshr_coal The maximum number of requests that can be coalesced into an MSHR (miss status hold register).

L1I_perfect Simulates a perfect L1 instruction cache, i.e., no instruction cache misses whatsoever.

L1I_size Size of L1 instruction cache in kilobytes.

L1I_assoc Set associativity of L1 instruction cache.

L1I_line_size Line size of L1 instruction cache in bytes.

L1I_mshr_num Number of MSHRs for L1 instruction cache.

L1C_perfect Simulates a perfect L1 data cache, i.e., no data cache misses whatsoever.

L1C_prefetch Turns on/off hardware L1 cache prefetching.

L1C_size Size of L1 data cache in kilobytes.

L1C_assoc Set associativity of L1 data cache.

L1C_line_size Line size of L1 data cache in bytes.

L1C_mshr_num Number of MSHRs for L1 data cache.

L1C_ports Number of access ports in L1 data cache.

L1C_latency Access latency of L1 data cache.

L1C_repeat_rate Number of cycles must elapse before L1 data cache can serve the next access.

L2C_perfect Simulates a perfect L2 cache, i.e., no memory accesses whatsoever.

L2C_prefetch Turns on/off hardware L2 cache prefetching.

L2C_size Size of L2 cache in kilobytes.

L2C_assoc Set associativity of L2 cache.

L2C_line_size Line size of L2 cache in bytes.

L2C_mshr_num Number of MSHRs for L2 cache.

L2C_ports Number of access ports in L2 cache tag array.

L2C_latency Access latency of L2 cache tag array.

L2C_repeat_rate Number of cycles must elapse before L2 tag array can serve the next access.

L2C_dports Number of access ports in L2 cache data array.

L2C_dlatency Access latency of L2 cache data array.

L2C_drepeat_rate Number of cycles must elapse before L2 data array can serve the next access.

C_SI_reqbuf_size Size of the request buffer in the system interface.

C_SI_outbuf_size Size of the write buffer in the system interface.

C_SI_ucbuf_size Size of the uncached buffer in the system interface.

3.2.5 Vector unit parameters

VU_frequency Vector unit frequency.

VU_collect_stats Turns on/off statistics collection.

VU_reqq_size Size of incoming request pipeline.

VU_reqq_latency Latency of incoming request pipeline.

VU_reqq_repeat_rate Repeat rate of incoming request pipeline.

VU_ctlregs Number of control register sets.

VU_mshrs Number of missing status holding registers.

VU_mshr_max_coals Maximum number of requests that can be coalesced into one MSHR.

VU_block_size Size of UV block.

VU_ivbuf_lines Number of lines in the IV buffer.

VU_ivbuf_line_size Line size of the IV buffer.

VU_cache_banks Number of banks in the vector cache.

VU_cache_bank_size Size of each bank of the vector cache.

VU_cache_set_size Associativity of the vector cache.

VU_addrgens Number of address generators.

VU_ag_waitq_size Number of requests that can be waiting for an address generator at the same time.

VU_ag_latency Latency of the address generator.

VU_ag_repeat_rate Repeat rate of the address generator.

VU_tlb_banks Number of banks in the vector TLB.

VU_tlb_bank_size Size of each bank of the vector TLB, in bytes.

VU_tlb_entry_size Entry size of the vector TLB.

VU_tlb_set_size Associativity of the vector TLB.

VU_tlb_max_misses Maximum number of concurrent misses supported the vector TLB.

3.2.6 Bus parameters

BUS_frequency Bus frequency relative to the processor core frequency.

BUS_T_width Width of the sysT bus.

BUS_F_width Width of the sysF bus.

BUS_arb_delay Minimum number of cycles from starting arbitration to gaining bus ownership.

BUS_max_cmisses Maximum number of outstanding cache misses.

3.2.7 Hub parameters

Hub_sim_on Turns on/off the Hub simulator. When the Hub simulator is turned off, each memory access takes a fixed number of cycles specified by **Hub_fixed_latency**.

Hub_fixed_latency Number of cycles that each memory access takes. Used when the Hub simulator is turned off.

Hub_frequency Hub clock rate relative to the processor core clock rate.

Hub_collect_stats Turns on/off statistics collection for the Hub module.

Hub_frame_outq Size of the out buffer in the Frame.

Hub_CRP_latency Latency of the Coherent Request Pipeline.

Hub_CRP_repeat_rate Repeat rate of the Coherent Request Pipeline.

Hub_RRB_size Size of the RRB (read request buffer).

Hub_WRB_size Size of the WRB (write request buffer).

Hub_IRB_size Size of the IRB (intervention request buffer).

Hub_URB_size Size of the PRB (put request buffer).

Hub_dc_size Size of the directory cache in MD.

Hub_dc_line_size Line size of the directory cache in MD.

Hub_dc_assoc Set associativity of the directory cache in MD.

Hub_dc_latency Access latency of the directory cache in MD.

Hub_dc_repeat_rate Access repeat rate of the directory cache in MD.

Hub_md_max_reqs Maximum number of requests MD can handle at one time.

Hub_ni_frequency Network clock rate relative to the processor core clock rate.

Hub_ni_outq_size Size of the output buffer in the NI.

Hub_ni_inq_size Size of the incoming buffer in the NI.

Hub_ni_delay Delay in the NI.

Hub_ni_min_pack Minimum size of a package sent over the network, in bytes.

Hub_ni_pack_delay Number of network cycles to send a package.

Network_hop_delay Number of network cycles of each hop.

3.2.8 DRAM parameters

General DRAM parameters.

DRAM_sim_on Turns on/off DRAM simulator. When the DRAM simulator is off, each DRAM access takes the fixed latency specified by **DRAM_latency**.

DRAM_latency Number of cycles each DRAM access takes when **DRAM_sim_on** is "0".

DRAM_frequency DRAM clock rate relative to the processor core clock rate.

DRAM_dl_latency Number of cycles to get a Directory line from DRAM.

DRAM_debug_on Enables debugging support.

DRAM_collect_stats Turns on/off statistics collection in the DRAM module.

DRAM_dram2md_delay Number of DRAM cycles to transmit data from DRAM to MD.

DRAM_trace_on Turns on/off DRAM accesses.

DRAM_trace_maximum Maximum number of DRAM accesses to trace.

DRAM_trace_file File to store trace information.

Parameters related the configuration of the memory system.

DRAM_bqueue_policy Bank queue reordering algorithm. Always 0 in this version.

DRAM_hot_row_policy Hot row open-close policy.

- **0** — Always close the hot row.
- **1** — Always leave the hot row open.
- **2** — Use predictor.

DRAM_reqqueue_size Size of the request queue in each backend.

DRAM_backends Number of backends.

DRAM_chips_per_back Number of memory chips in each backend.

DRAM_banks_per_chip Number of internal banks in each chip.

DRAM_mini_access Minimum DRAM access in bytes.

DRAM_data_bus_width Width of each data bus.

DRAM_granularity Bank interleaving granularity.

DRAM_type DRAM type, either "SDRAM" or "RDRAM".

SDRAM parameters:

SDRAM_tCCD CAS to CAS delay time. Defaults to 1.

SDRAM_tRRD Bank to bank delay time. Defaults to 2.

SDRAM_tRP Precharge time. Defaults to 3.

SDRAM_tRAS Minimum bank active time. Defaults to 7.

SDRAM_tRCD RAS to CAS delay time. Defaults to 3.

SDRAM_tAA CAS latency. Defaults to 3.

SDRAM_tDAL Data in to precharge time. Defaults to 5.

SDRAM_tDPL Data in to active/refresh time. Defaults to 2.

SDRAM_row_size The size of an active row in bytes.

SDRAM_row_hold_time How long can a hot row be active.

SDRAM_refresh_delay Delay of a refresh operation.

SDRAM_refresh_period Refresh period.

RDRAM parameters:

RDRAM_tPACKET Length of each command. Defaults to 4.

RDRAM_tRC The minimum delay from the first ACT command to the second ACT command. Defaults to 28.

RDRAM_tRR Delay from a RD command to next RD command. Defaults to 8.

RDRAM_tRP The minimum delay from a PRER command to an ACT command. Defaults to 8.

RDRAM_tCBUB1 Bubble between a RD and a WR command. Defaults to 4.

RDRAM_tCBUB2 Bubble between a WR and a RD command to the same device. Defaults to 8.

RDRAM_tRCD RAS to CAS delay. Defaults to 7.

RDRAM_tCAC Delay from a RD command to its associated data out. Defaults to 8.

RDRAM_tCWD CAS write delay. Defaults to 6.

RDRAM_row_size The size of an active row in bytes.

RDRAM_row_hold_time How long can a hot row be active.

RDRAM_refresh_delay Delay of a refresh operation.

RDRAM_refresh_period Refresh period.

3.2.9 Tracing and debugging parameters

Cache_trace_on Turns on/off cache access tracing. If it is 0, trace no accesses; if it is 1, trace only processor-initiated accesses; if it is 2, trace only vector-unit-initiated accesses; if it is 3, trace all accesses.

Cache_trace_sample Sampling factor.

Cache_trace_max Number of cache accesses to trace.

Mreq_trace_start Number of memory accesses must be serviced before starting tracing.

Mreq_trace_count Number of memory accesses to trace.

Mreq_trace_addr The specific address to be traced.

Mreq_trace_file File to store the tracing information.

Warn_level Set system warning level. Any warning message whose level is less than or equal to the given number is sent to the simulator standard error output. UVSIM ranks the warning messages from level 0 (most serious warning messages that will likely cause fatal problems) to level 3 (least serious warning messages that will likely not affect the simulation).

References

[1] IBM. IBM Advanced 64Mb Direct Rambus DRAM, Nov. 1997.

[2] IBM. IBM Advanced 256Mb Synchronous DRAM – Die Revision A, Aug. 1998.

[3] MIPS Technologies, Inc., Mountain View, California. *MIPS R10000 Microprocessor User's Manual, Version 2.0*, Oct. 1996.

[4] MIPS Technologies, Inc., Mountain View, California. *MIPS64^{TM} Architecture For Programmers, Volume II: The MIPS64^{TM} Instruction Set*, revision 0.95 edition, March 2001.

[5] Silicon Graphics, Inc., Mountain View, California. *SN2-MIPS Communication Protocol Specification*, revision 0.12 edition, Nov. 2001.

[6] Silicon Graphics, Inc., Mountain View, California. *Orbit Functional Specification, Vol. 1*, revision 0.1 edition, April 2002.

[7] Silicon Graphics, Inc., Mountain View, California. *Orbit Memory/Directory (MD) Hardware Reference*, original edition, April 2002.