NAME

mgsimdoc − Documentation and configuration for MGSim

DESCRIPTION

MGSim was developed to facilitate research in the design of many−core general−purpose processor chips, in particular Microgrid chips based on D−RISC. It also includes full−system capability, including a fully configurable I/O subsystem and memory interconnect.

OVERVIEW

An execution of the simulator entails the following steps:

1. the simulator parses configuration parameters from a configuration file (**−c**) and optional configuration overrides on the command line (**−o** / **−I**).
2. it creates the system's components (cores, FPUs, memories, etc) using the configuration. Input files, including program code, is loaded into ROM pseudo−devices.
3. it signals to the processor chip to boot.
4. it proceeds to run the simulation of all components, until either of the following happens:
   - the software running on the platform signals explicit termination;
   - a simulation error occurs;
   - the user interrupts the simulation.
5. Unless instructed otherwise with option **−t**, the simulation then drops to an interactive prompt where the state of the system can be inspected.

COMPONENTS AND TOPOLOGY

The minimum components found in a MGSim configuration are:

- a core;
- a memory system;
- an I/O network with a bootable ROM and SMC device.

The number of cores, as well as the topology of the I/O and memory systems between cores, is configurable.

Cores

MGSim defines two core models, MT−Alpha and MT−SPARC. Both models are based on the general D−RISC core design. D−RISC is an in−order, single−issue pipelined, dataflow−scheduled, multithreaded RISC core. MGSim's core model simulates D−RISC's pipeline, functional units, L1 cache, and ancillary components in detail, down to the interactions at every clock cycle. D−RISC also features asynchronous FPUs, where one FPU can be shared between two or more D−RISC cores.

Contrary to other simulator or ISA emulators which decouple component model simulation from the functional behavior of ISA instructions, MGSim's core implementation realistically decomposes instruction execution in multiple pipeline stages with explicit data paths and latch buffers. This enables accurate modeling of timing−dependent behavior fo all instructions, including inter−core synchronization, memory accesses and I/O operations.

Currently the choice of the core model is static (compile−time); it is selected during source configuration using **−−target**.

Memory

MGSim simulates a many−core chip connected to a shared memory. The memory protocol, topology and interconnect are determined by the run−time choice of a memory system.

The following memory systems are supported:

serial A simple bus connecting all cores with a single memory interface.

parallel
A multi−ported memory interface is connected to all cores using separate links. Requests from each port are handled concurrently. Consistency is resolved on request completion.

banked
Multiple memory interfaces are connected to all cores using an Omega network. The mapping of address space ranges to memory interfaces is configurable, and defaults to round−robin on the lowest significant bits of the address.

randombanked
Same as **banked**, with a bit shuffling for the mapping of addresses to interfaces, to maximize load distribution.

directddr
Similar to **banked**, where the memory interface simulates a DDR channel that pipelines memory requests.

randomddr
Similar to **randombanked**, with DDR interfaces.

flatcdma
Cores are connected to L2 caches, in turn connected to memory interfaces. Memory interfaces implement DDR channels. The L2 caches and memory interface are organized in a ring. Directories at each memory interface determine what data is currently present in cache, to optimize accesses. Each group of core sharing a L2 cache is connected using a bus. The protocol migrates cache lines at the L2 cache of last use.

cdma Like **flatcdma**, with two levels of rings. L2 caches are connected together in multiple rings each containing a middle−level directory. Middle−level directories are then connected in a single top−level ring also containing the memory interfaces.

zlcdma Like **cdma**, using an alternate, token−based but less realistic cache coherence protocol.

On−chip interconnect
When multiple cores are configured, they are automatically connected together using three on−chip networks: the memory network (for memory loads and stores), the delegation network (for cross−chip work distribution) and the link network (for local work distribution between adjacent cores).

The memory network is determined by the selected memory system, as described above. The cores are placed over the 2D grid surface using a space−filling curve, so as to maximize locality in the memory network at different core cluster sizes.

The link network is a one−dimensional, word−wide, short−latency network connecting all cores in their natural index ordering.

The delegation network connects all cores; is intended to be implemented as a narrow mesh, although its timing model attributes a single latency for any pairwise communication.

The interconnect is not directly configurable; instead, it is derived automatically from the selected memory system and number of cores.

It is possible to dump the interconnect topology as a Graphviz directed graph using command−line option **−T**.

I/O subsystem
The cores are connected to the simulation's environment using an I/O subsystem featuring pseudo−device emulations. The number of I/O buses, which pseudo−devices are connected to them and how they are connected to cores on the simulated chip are all configurable.

The choice of pseudo−device types includes:

- realistic serial UARTs (**uart**);
- realistic real time clocks (**rtc**);
- character matrix displays (**lcd**);
- ROMs with integrated DMA controller (**arom**);
- a graphical framebuffer (**gfx**);
- a syscall interface to the host's POSIX file handling (**rpc**);
- a System Management Controller (SMC) responsible for booting up the simulated platform's software (**smc**).

Some of these pseudo−devices have their own documentation, cf *SEE ALSO* below for details.

COMPONENTS AND NAMING

A Microgrid system is simulated as a *hierarchy* of connected *components*, implemented as objects in C++. Conceptually independent from the C++ class hierarchy, the component hierarchy reflects the position of the components on chip. For example the allocation unit ("**alloc**") is a sub−component of a processor ("**cpu**").

Components represent circuits that implement both processes and storage (state). Processes are either single cycle, or state machines over multiple cycles. Processes can perform the following actions:

- update storage, either locally or remotely, optionally through an arbitrator if the state is shared between two or more processes.
- send a signal, which (re−)starts another process either locally or remotely. This can be done either via explicit process activation or implicitely by pushing data into a buffer with a listener (consumer) process.

A component is defined by its sub−parts, which can be either:

- passive storage;
- arbitrators, that regulate shared access to passive storage;
- active storages, which wake up a consumer process upon becoming non−empty and suspend a producer process upon becoming full;
- processes, which access both storages and arbitrators;
- sub−components.

Moreover, each component can define one or more *monitoring variables* that reflect its internal state. When a variable is defined, it also has a name relative to the component where it is defined.

Entity names

Components and their sub−parts are identified by a *name*, relative to their parent entity. Any entity can be thus identified globally using its *fully qualified name* (FQN), which indicates its path in the hierarchy. For example **cpu4.pipeline.execute** is the FQN of the execute stage of the pipeline of the 5th simulated core on the system.

Entity names are used throughout the simulation environment: to manipulate components in the interactive prompt, to set up configuration, to report simulation messages, etc.

The name of an entity mat indicate its type:

- names starting with "**b_**" identify *buffers*, ie. FIFO queues of fixed−width entries and a statically configurable maximum size. These are active storages;

- names starting with "**f_**" identify *flags*, ie. single−bit active storages;
- names starting with "**p_**" identify *priority arbitrators*;

The FQN of an entity can further identify its type:

- a component FQN is a concatenation of its path in the hierarchy with periods ("**.**");
- a process FQN or monitoring variable FQN is the concatenation of its component FQN and its name with a colon ("**:**").

The list of all entities can be obtained from a running instance of the Microgrid simulator. From the interactive prompt, the commands **show components**, **show processes** and **show vars** in interactive mode (**−i**) can be used. Here is an example session:

```
00000000> show components *cpu27*
cpu27                       DRISC
  families                  DRISC::FamilyTable
  threads                   DRISC::ThreadTable
  registers                 DRISC::RegisterFile
  rau                       DRISC::RAUnit
  icache                    DRISC::ICache
    b_outgoing              Buffer<unsigned long long>
    b_incoming              Buffer<unsigned long>
  dcache                    DRISC::DCache
    b_completed             Buffer<unsigned long>
    b_incoming              Buffer<DRISC::DCache::Response>
    b_outgoing              Buffer<DRISC::DCache::Request>
  pipeline                  DRISC::Pipeline
    f_active                Register<bool>
    fetch                   DRISC::Pipeline::FetchStage
    decode                  DRISC::Pipeline::DecodeStage
    execute                 DRISC::Pipeline::ExecuteStage
 (some output lines omitted)

00000000> show processes *cpu27*
cpu27.alloc:thread−allocate
cpu27.alloc:family−allocate
cpu27.alloc:family−create
cpu27.alloc:thread−activation
cpu27.icache:outgoing
cpu27.icache:incoming
cpu27.dcache:completed−reads
cpu27.dcache:incoming
cpu27.dcache:outgoing
(some output lines omitted)

00000000> show vars *cpu27*
# size type  dtype max address     name
  8    level int   0   0x10160ee18 cpu27.alloc.b_alloc:cursize
  8    wmark int   32  0x10160ee10 cpu27.alloc.b_alloc:maxsize
  8    cumul int   N/A 0x10160edf8 cpu27.alloc.b_alloc:stalls
  8    cumul int   N/A 0x10160ee08 cpu27.alloc.b_alloc:totalsize
  8    level int   0   0x10160f788 cpu27.alloc.b_allocRequestsExclusive:cursiz
  8    cumul int   N/A 0x10160f768 cpu27.alloc.b_allocRequestsExclusive:stalls
(some output lines omitted)
```

CONFIGURATION SYSTEM AND VARIABLES

The simulated Microgrid can be configured via command−line parameters and an architecture configuration file.

The following can be configured:

- the *simulation environment* itself. For example, the name of the asynchronous monitoring output stream.
- *architectural constants* which have a pervasive effect. For example, the D−RISC control block size.
- the *system layout*. For example, the number of cores, as well as which I/O devices are connected to each I/O network or the number of DDR channels.
- *individual component parameters*. These can be configured with separate values for each component of a given type. For example, the associativity of the L1 D−cache can be configured separately for each core. Similarly, buffer (FIFO) maximum sizes can be configured individually.
- *shared component parameters* that cannot (yet) be configured per component. For example, the core frequency and the cache line width are shared across the system.

Each configurable item has a *name* which identifies it uniquely.

For individual component parameters, the name is composed of the fully qualified component name (FQN), followed by a colon and the parameter name. For example, **cpu27.dcache:associativity** is the parameter name for the associativity of the D−cache of core 27.

For all other configurable items, the option name is simply the name of the item. For example, **ControlBlockSize** is the name of the parameter to configure the control block size.

The set of all configurable items is documented in the reference configuration file provided alongside **mgsim**.

Configuration specification

The input configuration is a *sequence of configuration rules* called the *configuration space*. Each configuration rule has the form:

```
pattern = value
```

The value of each parameter is determined by *first match* of the parameter name against the sequence of patterns from the configuration rules. Matching is performed using *case−insensitive* comparison and Unix shell pattern matching semantics. In particular, ***** matches any string (including the empty string), **?** matches any character, and **[...]** matches any one of the enclosed characters.

For example, given the following configuration space:

```
cpu27.dcache:numsets = 8
cpu*.dcache:numsets = 4
```

the parameter **system.cpu27.dcache:numsets** matches the first specification and configures the D−cache of core 27 with 8 cache sets. Meanwhile, the corresponding parameter for all other cores fall back to the second specification and configure the remaining D−caches with 4 cache sets.

The configuration space is constructed when the simulator is started, using the following input:

1. any *override* from the command line (parameters **−o** and **−I**), in inverse order: later overrides on the command line are considered first;
2. the contents of the *configuration file* specified with parameter **−c**, or the default configuration file if none is specified, in inverse order: later key/value pairs in the file are considered first.

The configuration space can be dumped to the console output upon initialization using the command−line parameter **−d**. It is also copied to the asynchronous monitoring metadata file if this is enabled.

Important configuration variables

The following parameters seem to receive most interest:

NumProcessors
: The number of cores.

CPU*.ICache:Associativity, CPU*.ICache:NumSets
: The size of individual L1 I−caches.

CPU*.DCache:Associativity, CPU*.DCache:NumSets
: The size of individual L1 D−caches.

MemoryType
: The memory system to use.

Memory:NumRootDirectories
: For COMA−based systems, the number of root directories and thus number of external memory interfaces.

NumClientsPerL2Cache
: For COMA−based systems, the number of cores per L2 cache. Combined with **NumProcessors** this determines the number of L2 caches in total.

Memory:L2CacheAssociativity, Memory:L2CacheNumSets
: The size of each L2 cache.

Default values

Configurable parameters are generally not assigned default values. The simulator should report an error when no pattern in the configuration space matches the name of a parameter that needs configuration.

There are exceptions to this rule. Some parameters do have default values, which are described as comments in the reference configuration file.

Configuration file format

The files read by parameters **−c** and **−I** are text files that contain configuration specifications of the form **key = value**, as described above.

In a configuration file, the following extra syntax is also recognized:

- comments starting with **#** or **;** at the start of lines or after values;
- "section names", of the form **[NAME]**. A section name introduces a common prefix for the following keys, until the following section name. For example, the following syntax:

```
[CPU1]
ICache:Associativity = 4
DCache:Associativity = 8
```

  is equivalent to:

```
CPU1.ICache:Associativity = 4
CPU1.DCache:Associativity = 8
```

  The special section name **[global]** resets the prefix to empty.

More examples can be found in the default configuration file shipped with the program.

Standard configuration file

MGSim is shipped with a standard configuration file which defines the following system as of August 2012:

- 128 D−RISC cores at 1GHz, with 2K+4K L1 caches;
- 64 FPUs (2 cores per FPU);
- **randombanked** memory system clocked at 1GHz;
- one I/O pseudo−device of each type, around an I/O network connected to the first core only. The first ROM contains the program image specified on the command line, if any. Additional devices with type **arom** are created to hold character strings for additional command−line arguments and the configuration space.

INTERACTIVE MODE

When started with **−i**, or upon encountering an error and **−t** is not specified, MGSim presents an interactive prompt to the user to control the simulation.

The prompt indicates the current simulation cycle.

The command **help** lists the available commands.

General commands

run Run the system until it is idle or deadlocks. Livelocks will not be reported.

step [N]
Advance the system by N clock cycles (default 1).

state Show the state of the system. Idle components are left out.

statistics
Print the current simulation statistics.

quit Exit the simulation.

help [COMMAND]
Print the help text for COMMAND, or this text if no command is specified.

aliases List all command aliases.

Inspection commands

info COMPONENT [ARGS...]
Show help/configuration/layout for COMPONENT.

inspect NAME [ARGS...] (or read)
Inspect NAME (component, process or monitoring variable). Components may have multiple **inspect** sub−commands. See **info NAME** for details.

line COMPONENT ADDR
Lookup the memory line at address ADDR in the memory system COMPONENT.

disassemble ADDR [SZ]
Disassemble the program from address ADDR.

show vars [PAT]
List monitoring variables matching PAT.

show syms [PAT]
List program symbols matching PAT.

show components [PAT] [LEVEL]
List components matching PAT (at most LEVELs).

show processes [PAT]
List processes matching PAT.

show devicedb
List the I/O device identifier database. See mgsimdev−smc(7) for use.

lookup ADDR
Look up the program symbol closest to address ADDR.

Execution control / tracing

bp (breakpoint)
List all current breakpoints.

bp add MODE ADDR
Set a breakpoint at address ADDR with MODE.

bp clear
Clear all breakpoints.

bp del ID
Delete the breakpoint specified by ID.

bp disable ID or bp enable ID
Disable/enable the breakpoint specified by ID.

bp off or bp on
Disable/enable breakpoint detection.

bp state
Report which breakpoints have been reached.

trace line COMPONENT ADDR [clear]
Enable/Disable tracing of the cache line at address ADDR by memory COMPONENT.

trace [FLAGS...]
Show current traces / toggle tracing of FLAGS.

MONITORING

MGSim offers two mechanisms to monitor the simulated environment over time: synchronous *event traces*, where all detailed events are reported, and asynchronous *variable traces*, where the state of the simulation is sampled at regular time intervals.

Synchronous event traces

Event traces are enabled using the **trace** command in interactive mode (**−i**). It causes MGSim to report events in a text format on its standard output at each simulation step.

Event categories can be individually selected:

trace regs
Events reporting updates to the cores' register files.

trace pipe
Events reporting pipeline activity within cores.

trace fpu
Events reporting FPU activity.

trace mem
Events reporting memory loads and stores.

trace memnet
Events reporting messages on the memory network.

trace io
Events reporting I/O operations.

trace ionet
Events reporting communication on the I/O interconnect.

trace net
Events reporting communication on the inter−core delegation and link networks.

trace sim
Events about concurrency management and synchronization between cores.

trace deadlocks
Events reporting process stalls.

trace flow
Events reporting control flow of the software running on the platform (branches).

trace prog
Debugging messages generated by the software running on the platform.

The special aliases **trace all** and **trace none** are also recognized.

A full simulation trace can be generated using the following command:

```
echo "trace all; run; quit" | mgsim −i ...  >event−trace.log
```

A simulation trace can in turn be transformed to an HTML table for graphical representations with the separate **viewlog** utility. See viewlog(1) for details.

Note that synchronous event traces slow down the simulation by a large factor.

Asynchronous variable traces
Variable traces are enabled using the command−line flag **−m**. It causes MGSim to start a second thread in the simulation process which samples monitoring variables at a regular time interval.

The list of monitoring variables to sample is selected using the configuration variable **MonitorSampleVariables**. The standard selection shipped with the standard configuration file monitors the number of instructions issued, and the number of floating−point instructions executed over time. It can be overloaded using **−o**.

The time interval between samples is configured using **MonitorSampleDelay**; the standard configuration sets this to 1ms.

The asynchronous monitoring has two outputs. The *metadata* indicates which variables were selected and their width in bytes. The *trace* reports the samples in fixed−length data packets. The output file names are configured using **MonitorMetadataFile** and **MonitorTraceFile**, and default to **mgtrace.md** and **mgtrace.out**.

The metadata and trace can then in turn be converted to text form using the separate utility **readtrace**; see readtrace(1) for details.

For example:

```
mgsim −m −o MonitorSampleVariables="cpu*.pipeline.execute.op"
readtrace mgtrace.md mgtrace.out >var−trace.log
```

Asynchronous monitoring automatically suspends whenever MGSim displays its interactive prompt.

SEE ALSO

- mgsim(1), viewlog(1), readtrace(1)
- mgsimdev−arom(7), mgsimdev−gfx(7), mgsimdev−lcd(7), mgsimdev−uart(7), mgsimdev−rtc(7)

- M5, *http://www.m5sim.org/* a detailed simulator for networks.
- Raphael Poss, Mike Lankamp, Qiang Yang, Jian Fu, Irfan Uddin, and Chris Jesshope. *MGSim − a simulation environment for multi−core research and education.* In Proc. Intl. Conf. on Embedded Computer Systems: Architectures, MOdeling and Simulation (SAMOS). IEEE, Samos, Greece, July 2013.
- Mike Lankamp, Raphael Poss, Qiang Yang, Jian Fu, Irfan Uddin, and Chris R. Jesshope. *MGSim − simulation tools for multi−core processor architectures.* Technical Report arXiv:1302.1390v1 [cs.AR], University of Amsterdam, February 2013. http://arxiv.org/abs/1302.1390
- Raphael Poss, Mike Lankamp, Qiang Yang, Jian Fu, Michiel W. van Tol, and Chris Jesshope. *Apple−CORE: Microgrids of SVP cores (invited paper).* In Proc. 15th Euromicro Conference on Digital System Design. IEEE, Cesme, Izmir, Turkey, September 2012.
- Raphael Poss, Mike Lankamp, M. Irfan Uddin, Jaroslav Sykora, and Leos Kafka. *Heterogeneous integration to simplify many−core architecture simulations.* In Proc. 2012 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools, RAPIDO'12, pages 17−24. ACM, 2012. ISBN 978−1−4503−1114−4.

BUGS

Report bugs & suggest improvements to *microgrids@svp−home.org*.

AUTHOR

MGSim was created by Mike Lankamp. MGSim is now under stewardship of the MGSim project. This manual page was written by Raphael 'kena' Poss.

COPYRIGHT