**Title**

HW-SW SystemC Co-Simulation SoC Validation Platform

**Analysis Capability Report**

Contract No. 4200022968

by

 TU Braunschweig, Germany

March 6, 2012

**Document No.: IDA-SCSV-ACR-001**

**Table of Contents**

1 Introduction 4

1.1 Purpose and Scope 4

1.2 Revisions 4

2 Analysis API 5

2.1 Listing parameters 5

2.2 Read/write/display parameters 6

2.3 Logging parameters 6

2.4 Creating waveforms from parameters 7

2.5 Callbacks for parameters 7

2.6 Creating new/own performance counters 8

2.7 Parameter Overview 8

3 Configuration reports 10

3.1 AHBCTRL 10

3.2 APBCTRL 11

3.3 MCTRL 11

3.4 Generic Memory 12

3.5 MMU\_CACHE 12

3.5.1 Cache 12

3.5.2 Localram 13

3.5.3 MMU 13

3.6 GPTimer 13

3.7 IRQMP 14

3.8 SoCWire 14

4 Simulation reports 15

4.1 AHBCTRL 15

4.2 APBCTRL 15

4.3 MCTRL 16

4.4 Generic Memory 16

4.5 MMU\_CACHE 16

4.5.1 Cache 17

4.5.2 Localram 17

4.5.3 MMU 18

4.6 GPTimer 18

4.7 IRQMP 18

4.8 SoCWire 19

**Table of Tables**

Table 1 - Revisions of this document 4

Table 2 - Parameter Overview 9

# Introduction

## Purpose and Scope

This document is the Analysis Capability Report of the SystemC Co-Simulation SoC Validation Platform (SoCRocket) and ist contained simulation models. It describes all features provided for the user concerning configuration and runtime analysis.

The models of the SoCRocket library provide internal counters for measuring simulation performance, throughput, storing configuration data and tracing behaviour. If the simulation is configured for verbosity-level 4 or above, the models produce a configuration report at the start of simulation and an analysis report at the end of simulation. The configuration reports list all relevant constructor parameters of the modules (chapter ). The analysis report contains figures related to simulation performance and behaviour (chapter ). Furthermore, selected parameters of the system are exposed to the user by means of an Analysis API. This API is based on GreenControl open-source software bundle, developed by GreenSocs Ltd. ([www.greensocs.com](http://www.greensocs.com)). The recommended style of using GreenControl within SoCRocket is described in chapter 2.

## Revisions

The following table will be updated during the course of the project.

|  |  |  |
| --- | --- | --- |
| **Version** | **Date** | **Description** |
| 0.1 | 02/01/12 | Initial draft version |
| 0.2 | 23/01/12 | Configuration analysis / Performance analysis |
| 0.3 | 06/03/12 | Analysis API/GreenControl |
|  |  |  |

Table 1 - Revisions of this document

# Analysis API

The models of the SoCRocket library implement a set of distinct configuration parameters and performance counters. All of them are implemented using the GS\_PARAM mechanism provided by the GreenSoCs package GreenControl. Respectively, every platform modeled with the SoCRocket library can be extended using GreenControl surveillance tools (e.g. GreenAV). This enables the user to:

* Access all counters at every time from everywhere in the simulation
* Directly log changes in trace-files
* Directly log changes in waveforms (vcd)
* Register/execute callbacks on changes
* Overwrite counters

This means, every model implements a GreenControl realm performance\_counters. Individual parameters can be addressed in a hierarchical way:

<module\_path>.performance\_counters.<param\_name>

e.g. parameter total\_transactions of module ahbctrl in class top:

top.ahbctrl.performance\_counter.total\_transactions

The following sections contain code snippets and explain the most prominent API features. Complementary information can be found in the GreenControl User Manual. Some more useful examples are provided in the greencontrol/examples directory of your GreenSoCs installation ($GREENSOCS).

## Listing parameters

The example below shows how to generate a list of all available configuration parameters and performance counters. All operations on the analysis API require the greencontrol/all.h header file (line 1). Line 5 shows how to create a pointer to the API. The list of parameters is obtained by calling the getParamList function (line 7). Result of this operation is a std::vector of type string, which may be displayed (line 8 – 10) or evaluated in any other way.

1 #include **"greencontrol/all.h"**

2 // ...

3 // in sc\_main

4 // ...

5 gs::cnf::cnf\_api \*CFG = gs::cnf::GCnf\_Api::getApiInstance(NULL);

6 v::info << **"main"** << **"Green Control Parameter Names:"** << v::endl;

7 std::vector<std::string> plist = CFG->getParamList();

8 **for**(**uint32\_t** i = 0; i < plist.size(); i++) {

9 v::info << **"main"** << plist[i] << v::endl;

10 }

11 // ...

## Read/write/display parameters

Reading and writing parameters also requires a pointer to the analysis API (line 5). The actual access is done using the getParam member function (line 7), which expects the SystemC path including the name of the parameter as an input argument. The function returns a pointer of type gs::gs\_param\_base. It can be used to directly access the counter (read - line 8, write - line 11, print lines 7 - 9 and 13 - 15).

1 #include **"greencontrol/all.h"**

2 // ...

3 // in sc\_main

4 // ...

5 gs::cnf::cnf\_api m\_api = gs::cnf::GCnf\_Api::getApiInstance(NULL);

6 gs::gs\_param\_base \*total\_transactions =

7 m\_api->getParam(**"top.ahbctrl.performance\_counters.total\_transactions"**);

8 v::info << **"main"** << **"Number of total transactions handled by the AHBCtrl : "**

9 << total\_transactions << v::endl;

10 // Reset total transactions

11 total\_transactions = 0;

12 // Print value direct

13 v::info << **"main"** << **"Total Transactions: "**

14 << m\_api->getPar(**"top.ahbctrl.performance\_counters.total\_transactions"**)->

15 getString() << **"%"** << v::endl;

16 // ..

## Logging parameters

Logging the changes of a parameter in a text file requires the GreenAV analysis plugin, which is a part of the GreenSocs installation. The code example initializes the plugin in line 5. Lines 8 – 9 allocate the pointer to the GreenConfig API and lines 11 – 12 the pointer to the GreenAV interface. The create\_OutputPlugin function of the GreenAV interface is used to open a logfile (lines 17 – 21), the add\_to\_output function registers GS\_PARAMs for logging (lines 22 – 25). A simulation might use multiple logfiles or record multiple parameters in a single joint logfile.

1 #include **"greencontrol/all.h"**

2 // ...

3 // in sc\_main:

4 // ...

5 gs::av::GAV\_Plugin analysisPlugin(**"AnalysisPlugin"**);

6

7 // sc\_main Config API

8 gs::cnf::cnf\_api\* mainApi =

9 gs::cnf::GCnf\_Api::getApiInstance(NULL);

10 // sc\_main Analysis API

11 boost::shared\_ptr<gs::av::GAV\_Api> mainGAVApi =

12 gs::av::GAV\_Api::getApiInstance(NULL);

13 // …

14 // Instantiate models

15 // …

16 // Log to text file:

17 gs::av::OutputPlugin\_if\* ahbctrl\_opp

18 = mainGAVApi->create\_OutputPlugin(

19 gs::av::TXT\_FILE\_OUT,

20 **"ahbctrl.log"**

21 );

22 mainGAVApi->add\_to\_output(ahbctrl\_opp,

23 mainApi->getPar(**"top.ahbctrl.performance\_counters.total\_transactions"**)

24 );

25 mainGAVApi->add\_to\_output(ahbctrl\_opp,

26 mainApi->getPar(**"top.ahbctrl.performance\_counters.successful\_transactions"**)

27 );

28 // ...

29 sc\_start();

30 // ...

## Creating waveforms from parameters

Creating waveforms works very similar to logging data in text files (2.3) and also requires the GreenAV plugin (line 5). For access to the VCD Writer the create\_OutputPlugin command of the GreenAV interface must be configured for gs::av::VCD\_FILE\_OUT (line 19). Registration of a parameter with the add\_to\_output command creates a new track in the VCD file (lines 22 - 27). The resulting waveforms can be displayed using any VCD viewer (e.g. GTKWave).

1 #include **"greencontrol/all.h"**

2 // ...

3 // in sc\_main:

4 // ...

5 gs::av::GAV\_Plugin analysisPlugin(**"AnalysisPlugin"**);

6

7 // sc\_main Config API

8 gs::cnf::cnf\_api\* mainApi =

9 gs::cnf::GCnf\_Api::getApiInstance(NULL);

10 // sc\_main Analysis API

11 boost::shared\_ptr<gs::av::GAV\_Api> mainGAVApi =

12 gs::av::GAV\_Api::getApiInstance(NULL);

13 // …

14 // Instantiate models

15 // …

16 // Log to text file:

17 gs::av::OutputPlugin\_if\* ahbctrl\_opp

18 = mainGAVApi->create\_OutputPlugin(

19 gs::av::VCD\_FILE\_OUT,

20 **"ahbctrl.vcd"**

21 );

22 mainGAVApi->add\_to\_output(ahbctrl\_opp,

23 mainApi->getPar(**"top.ahbctrl.performance\_counters.total\_transactions"**)

24 );

25 mainGAVApi->add\_to\_output(ahbctrl\_opp,

26 mainApi->getPar(**"top.ahbctrl.performance\_counters.successful\_transactions"**)

27 );

28 // ...

29 sc\_start();

30 // ...

## Callbacks for parameters

To be able to dynamically react on analysis events, it is possible to register callback functions for performance counters. This can be done in a very simple way. The example below defines a GS\_PARAM (line 3), which can be seen as a placeholder for any performance counter in the system. The constructor of class MyModule uses the GC\_REGISTER\_PARAM\_CALLBACK macro to register the config\_callback function as a callback for parameter int\_param (line 9). The function itself is defined in lines 14 – 20.

1 **class** MyModule : **public** sc\_module {

2 **public**:

3 gs::gs\_param<**int**> int\_param;

4 // parameter who should call on change GC\_HAS\_CALLBACKS();

5

6 SC\_HAS\_PROCESS(MyModule);

7 MyModule(sc\_module\_name name) :

8 sc\_module(name), int\_param(**"int\_param"**) {

9 GC\_REGISTER\_PARAM\_CALLBACK(&int\_param, MyModule, config\_callback);

10 // Do some param changes, e.g. in an sc\_thread

11 }

12 ~MyModule() { GC\_UNREGISTER\_CALLBACKS(); }

13 // Callback function with default signature.

14 **void** config\_callback(gs::gs\_param\_base& par) {

15 **if**(par.is\_destructing()) {

16 /\* dont access the value here\*/

17 } **else** {

18 /\* do stuff \*/

19 }

20 }

21 };

(Example copied from GreenControl Reference Card)

## Creating new/own performance counters

Creating new additional performance counters is straight forward. If the class already contains other performance counters (e.g. any SoCRocket model), it is sufficient to transform the parameter in a GS\_PARAM:

gs::gs\_param<uint32\_t> my\_perf\_counter;

New models must make sure to include the greencontrol/config.h header file. They must also provide a public pointer to the GreenControl API and a parameter array:

gs::cnf::cnf\_api \*m\_api;

gs::gs\_param\_array m\_performance\_counters;

The API has to be initialized in the constructor of the class:

m\_api = gs::cnf::GCnf\_Api::getApiInstance(this);

## Parameter Overview

The following table lists all configuration values and performance counters, which are currently wrapped using GreenControl. Additionally, the SoCRocket singlecore example platform may be started with the command line parameter listparams to generate a list of all performance counters in the system:

e.g. ./build/platforms/singlecore-lt/systemc/singlecore.lt.platform ./build/platforms/singlecore-lt/systemc/singlecore.lt.prom ./build/software/grlib\_tests/hello.sparc **listparams** | grep performance

|  |  |  |
| --- | --- | --- |
| Model | Parameter | Description |
| ahbctrl | arbitrated | number of arbitrated transactions (AT) |
|  | bytes\_read | bytes read since start of simulation |
|  | bytes\_written | bytes written since start of simulation |
|  | idle\_cycles | number of idle cycles (no request) |
|  | maximum\_waiting\_time | maximum arbitration time |
|  | maximum\_waiting\_master\_id | master with maximum arbitration time |
|  | successful\_transactions | number of successful transactions |
|  | total\_transactions | total number transactions |
|  | total\_wait | accumulated total arbitration time |
| apbctrl | successful\_transactions | number of successful transactions |
|  | total\_transactions | number of total transactions |
| gptimer. GPCounter\_{1-6} | underflow | number of underflows in counters 1 – 6 since start of simulation |
| irqmp | cpu\_line\_activity{0-15} | number of interrupts sent to cpu X |
|  | irq\_line\_activity.{0-31} | number of interrupts received on irq line X |
| mctrl | last\_deep\_power\_down | time of last switch to deep power down mode |
|  | last\_power\_down | time of last switch to power down mode |
|  | last\_self\_refresh | time of last switch to self refresh mode |
|  | successful\_transactions | number of successful transactions |
|  | total\_deep\_power\_down | total time in deep power down mode |
|  | total\_power\_down | total time in power down mode |
|  | total\_self\_refresh | total time in self refresh |
|  | total\_transactions | total number of transactions |
| generic mem (IO/ROM/SRAM/SD) | bytes\_read | bytes read since start of simulation |
|  | bytes\_written | bytes written since start of simulation |
| icache/dcache | bypass\_operations | number of bypass operations (cache disabled or ASI 0x0) |
|  | read\_hits.{0-3} | number of read hits in sets 0 - 3 |
|  | read\_misses | number of read misses |
|  | write\_hits.{0-3} | number of write hits in sets 0 - 3 |
|  | write\_misses | number of write misses |
|  |  |  |
|  |  |  |

Table - Parameter Overview

# Configuration reports

The platform infrastructure produces reports for the configuration of each simulation model in the system. The reports are generated in the constructor of the separate IPs. They are written to std::out, if the system is configured for verbosity-level 4 (info) or above.

## AHBCTRL

The configuration report is generated in the constructor of class ahbctrl. It displays all constructor parameters of the device. The example below shows the report for an AHBCTRL with priority arbitration (rrobin = 0), power monitoring enabled (pow\_mon = 1) and AT abstractions.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@0 s /0 (ahbctrl): Info: \* Created AHBCTRL with following parameters:

@0 s /0 (ahbctrl): Info: \* ioaddr/iomask: fff/fff

@0 s /0 (ahbctrl): Info: \* cfgaddr/cfmask: ff0/ff0

@0 s /0 (ahbctrl): Info: \* rrobin: 0

@0 s /0 (ahbctrl): Info: \* split: 0

@0 s /0 (ahbctrl): Info: \* defmast: 0

@0 s /0 (ahbctrl): Info: \* ioen: 0

@0 s /0 (ahbctrl): Info: \* fixbrst: 0

@0 s /0 (ahbctrl): Info: \* fpnpen: 1

@0 s /0 (ahbctrl): Info: \* mcheck: 1

@0 s /0 (ahbctrl): Info: \* pow\_mon: 1

@0 s /0 (ahbctrl): Info: \* abstractionLayer (LT = 8 / AT = 4): 4

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

The AHBCTRL additionally generates a report describing the decoder initialization. This report is produced in the start\_of\_simulation function of class ahbctrl (for verbosity >= 4).

The report lists the up to 16 slaves and up to 16 masters, which are bound to the AHBCTRL. Each of them may contain up to four sub-devices. Sub-devices are identified by a base register entry (BAR0-4) comprising the 12bit MSB address of the component (haddr) and the 12bit AHB address mask (hmask).

The example below shows a report for an AHBCTRL with one master (Testbench) and two slaves: APBCTRL, MCTRL. The MCTRL contains three sub-devices.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@0 s /0 (top.ahbctrl): Info: \* DECODER INITIALIZATION

@0 s /0 (top.ahbctrl): Info: \* ----------------------

@0 s /0 (top.ahbctrl): Info: \* SLAVE name: top.apbctrl

@0 s /0 (top.ahbctrl): Info: \* BAR0 with MSB addr: 0x800 and mask: 0xfff

@0 s /0 (top.ahbctrl): Info: \* BAR1 not used.

@0 s /0 (top.ahbctrl): Info: \* BAR2 not used.

@0 s /0 (top.ahbctrl): Info: \* BAR3 not used.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@0 s /0 (top.ahbctrl): Info: \* SLAVE name: top.mctrl

@0 s /0 (top.ahbctrl): Info: \* BAR0 with MSB addr: 0x0 and mask: 0xe00

@0 s /0 (top.ahbctrl): Info: \* BAR1 with MSB addr: 0x200 and mask: 0xe00

@0 s /0 (top.ahbctrl): Info: \* BAR2 with MSB addr: 0x400 and mask: 0xc00

@0 s /0 (top.ahbctrl): Info: \* BAR3 not used.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@0 s /0 (top.ahbctrl): Info: \* Master name: top.testbench

@0 s /0 (top.ahbctrl): Info: \* BAR0 not used.

@0 s /0 (top.ahbctrl): Info: \* BAR1 not used.

@0 s /0 (top.ahbctrl): Info: \* BAR2 not used.

@0 s /0 (top.ahbctrl): Info: \* BAR3 not used.

@0 s /0 (top.ahbctrl): Info: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## APBCTRL

The configuration report for the APBCTRL is generated in the constructor of class apbctrl. It displays all constructor parameters of the device. The example below shows an APBCTRL in LT mode.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@0 s /0 (top.apbctrl): Info: \* Created APBCTRL with following parameters:

@0 s /0 (top.apbctrl): Info: \* haddr/hmask: 800/fff

@0 s /0 (top.apbctrl): Info: \* mcheck: 1

@0 s /0 (top.apbctrl): Info: \* hindex: 1

@0 s /0 (top.apbctrl): Info: \* pow\_mon: 0

@0 s /0 (top.apbctrl): Info: \* ambaLayer (LT = 8/AT = 4): 8

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

The APBCTRL additionally generates a report describing its decoder initialization. This report is produced in the start\_of\_simulation function of class apbctrl (for verbosity >= 4).

The report lists the up to 16 slaves, which are bound to the APBCTRL. Each of them is identified by a 12 bit segment address (paddr) and a 12 bit AMBA address mask (pmask). The example shows a report for an APBCTRL with one slave. The APB interface of the MCTRL is located at address/mask: 0x000/0xfff.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@0 s /0 (top.apbctrl): Info: \* APB DECODER INITIALIZATION

@0 s /0 (top.apbctrl): Info: \* --------------------------

@0 s /0 (top.apbctrl): Info: \* Slave name: top.mctrl

@0 s /0 (top.apbctrl): Info: \* BAR with MSB addr: 0 and mask: fff

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## MCTRL

The configuration report for the memory controller unit (MCTRL) is generated in the constructor of class mctrl. It displays all constructor parameters of the device. The example below shows a MCTRL in AT mode with SDRAM enabled (sden = 1). The SD bus is 32 bits wide (sdbits). The special features of mobile RAM are disabled (mobile = 0).

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@0 s /0 (top.mctrl): Info: \* Created MCTRL with following parameters:

@0 s /0 (top.mctrl): Info: \* romasel: 28

@0 s /0 (top.mctrl): Info: \* sdrasel: 29

@0 s /0 (top.mctrl): Info: \* romaddr/rommask: 0/e00

@0 s /0 (top.mctrl): Info: \* ioaddr/iomask: 200/e00

@0 s /0 (top.mctrl): Info: \* ramaddr/rammask: 400/c00

@0 s /0 (top.mctrl): Info: \* paddr/pmask: 0/fff

@0 s /0 (top.mctrl): Info: \* wprot: 0

@0 s /0 (top.mctrl): Info: \* srbanks: 5

@0 s /0 (top.mctrl): Info: \* ram8: 1

@0 s /0 (top.mctrl): Info: \* ram16: 1

@0 s /0 (top.mctrl): Info: \* sepbus: 0

@0 s /0 (top.mctrl): Info: \* sdbits: 32

@0 s /0 (top.mctrl): Info: \* mobile: 0

@0 s /0 (top.mctrl): Info: \* sden: 1

@0 s /0 (top.mctrl): Info: \* hindex: 0

@0 s /0 (top.mctrl): Info: \* pindex: 0

@0 s /0 (top.mctrl): Info: \* pow\_mon: 0

@0 s /0 (top.mctrl): Info: \* abstractionLayer (LT = 8 / AT = 4): 4

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## Generic Memory

The configuration memory for the generic memory is generated in the constructor of class GenericMemory. The generic memory can be configured to represent ROM, IO, SRAM and SDRAM storage. The example below shows the configuration report for a 512 MB SDRAM consisting of one memory bank with 16 columns.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@0 s /0 (top.sdram): Info: \* Created GenericMemory with following parameters:

@0 s /0 (top.sdram): Info: \* device\_type (0-ROM, 1-IO, 2-SRAM, 3-SDRAM): 3

@0 s /0 (top.sdram): Info: \* banks: 1

@0 s /0 (top.sdram): Info: \* bsize: 20000000

@0 s /0 (top.sdram): Info: \* bit width: 32

@0 s /0 (top.sdram): Info: \* cols (SD only): 16

@0 s /0 (top.sdram): Info: \* pow\_mon: 0

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## MMU\_CACHE

Depending on the configuration the MMU\_CACHE IP consists of multiple sub-components. Each of them generates a detailed configuration report on their own. Therefore, the configuration report of the MMU\_CACHE top-level module only provides structural information. The report is generated in the constructor of class mmu\_cache. The example shows a configuration with icache and dcache enabled. MMU and localrams (scratchpads) are disabled.

@0 s /0 (mmu\_cache): Info: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@0 s /0 (mmu\_cache): Info: \* Created MMU\_CACHE in following configuration:

@0 s /0 (mmu\_cache): Info: \* ---------------------------------------------

@0 s /0 (mmu\_cache): Info: \* instruction cache enable (icen): 1

@0 s /0 (mmu\_cache): Info: \* data cache enable (dcen): 1

@0 s /0 (mmu\_cache): Info: \* mmu enable (mmu\_en): 0

@0 s /0 (mmu\_cache): Info: \* instruction scratchpad enable (ilram): 0

@0 s /0 (mmu\_cache): Info: \* data scratchpad enable (dlram): 0

@0 s /0 (mmu\_cache): Info: \* abstraction Layer (LT = 8 / AT = 4): 8

@0 s /0 (mmu\_cache): Info: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

### Cache

The configuration reports for the instruction cache and the data cache are generated by the constructor of class vectorcache. The report contains all relevant constructor parameters. Additional information is given concerning the size of the cache in kb, the number of bytes per cache line, the number of cache lines per set and the width of the cache tag in bits. The example below shows the configuration report for a data cache with 4 sets of 1kb:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@0 s /0 (mmu\_cache.dcache): Info: \* Created cache memory with following parameters:

@0 s /0 (mmu\_cache.dcache): Info: \* ------------------------------------------

@0 s /0 (mmu\_cache.dcache): Info: \* mmu\_en: 0

@0 s /0 (mmu\_cache.dcache): Info: \* burst\_en: 0

@0 s /0 (mmu\_cache.dcache): Info: \* sets: 4

@0 s /0 (mmu\_cache.dcache): Info: \* setsize: 1

@0 s /0 (mmu\_cache.dcache): Info: \* setlock: 0

@0 s /0 (mmu\_cache.dcache): Info: \* linesize: 4

@0 s /0 (mmu\_cache.dcache): Info: \* repl (0-Direct Mapped, 1-LRU,   
 2-LRR, 3-RANDOM): 3

@0 s /0 (mmu\_cache.dcache): Info: \* ------------------------------------

@0 s /0 (mmu\_cache.dcache): Info: \* Size of each cache set 1 kb

@0 s /0 (mmu\_cache.dcache): Info: \* Bytes per line 16 (offset bits: 4)

@0 s /0 (mmu\_cache.dcache): Info: \* Number of cache lines per set 64   
 (index bits: 6)

@0 s /0 (mmu\_cache.dcache): Info: \* Width of cache tag in bits 22

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

### Localram

The configuration report for the localram is generated in the constructor of class localram. Because the localram is a very simple storage device, the report only contains two parameters, associated with starting address and capacity. The example shows the reports for an instruction scratchpad and a data scratchpad in default configuration.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@0 s /0 (mmu\_cache.ilocalram): Info: \* Created localram with following parameters:

@0 s /0 (mmu\_cache.ilocalram): Info: \* ---------------------------------------

@0 s /0 (mmu\_cache.ilocalram): Info: \* lrstart (start address): 8e000000

@0 s /0 (mmu\_cache.ilocalram): Info: \* lrsize: 80000 bytes

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@0 s /0 (mmu\_cache.dlocalram): Info: \* Created localram with following parameters:

@0 s /0 (mmu\_cache.dlocalram): Info: \* ---------------------------------------

@0 s /0 (mmu\_cache.dlocalram): Info: \* lrstart (start address): 8f000000

@0 s /0 (mmu\_cache.dlocalram): Info: \* lrsize: 80000 bytes

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

### MMU

The MMU configuration report is generated in the constructor of class mmu. The report contains all the configuration parameters of the MMU. The example shows the report for a MMU with split TLBs (8x instructions, 8x data).

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@0 s /0 (mmu\_cache.mmu): Info: \* Created mmu with following parameters:

@0 s /0 (mmu\_cache.mmu): Info: \* --------------------------------------

@0 s /0 (mmu\_cache.mmu): Info: \* itlbnum: 8

@0 s /0 (mmu\_cache.mmu): Info: \* dtlbnum: 8

@0 s /0 (mmu\_cache.mmu): Info: \* tlb\_type (0 - split, 1 - shared): 0

@0 s /0 (mmu\_cache.mmu): Info: \* tlb\_rep: 1

@0 s /0 (mmu\_cache.mmu): Info: \* mmupgsz (0, 2-4kb, 3-8kb, 4-16kb, 5-32kb): 0

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## GPTimer

The configuration report for the GPTimer is generated in the constructor of class gptimer. The report contains all the configuration parameters of the GPTimer. The example below shows the report for a GPTimer with 7 parallel counters and watchdog disabled.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@0 s /0 (top.timer): Info: \* Created gptimer with following parameters:

@0 s /0 (top.timer): Info: \* ------------------------------------------

@0 s /0 (top.timer): Info: \* ntimers: 7

@0 s /0 (top.timer): Info: \* pindex: 0

@0 s /0 (top.timer): Info: \* paddr/pmask: 0/fff

@0 s /0 (top.timer): Info: \* pirq: 0

@0 s /0 (top.timer): Info: \* sepirq: 0

@0 s /0 (top.timer): Info: \* sbits: 18

@0 s /0 (top.timer): Info: \* nbits: 22

@0 s /0 (top.timer): Info: \* wdog: 0

@0 s /0 (top.timer): Info: \* pow\_mon: 0

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## IRQMP

The configuration report for the IRQMP is generated in the constructor of class irqmp. The report shows all the constructor parameters of the module. The example refers to an instance configured for two master CPUs. The extended interrupt (eirq) is mapped to IRQ line 4.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@0 s /0 (top.irqmp): Info: \* Created configuration report with following parameters:

@0 s /0 (top.irqmp): Info: \* --------------------------------------------

@0 s /0 (top.irqmp): Info: \* paddr/pmask: 0/fff

@0 s /0 (top.irqmp): Info: \* ncpu: 2

@0 s /0 (top.irqmp): Info: \* eirq: 4

@0 s /0 (top.irqmp): Info: \* pindex: 0

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## SoCWire

The configuration report for the SoCWire IP contains all the constructor parameter of the module. It is generated in the constructor of class AHB2Socwire. The example shows a report for the AT configuration with APB index 1 (pindex) and AHB index 2 (hindex). In interrupt mode the device utilizes IRQ line 3 (apb\_irq\_id).

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@0 s /0 (DUT): Info: \* Created AHB2SoCWire with following parameters:

@0 s /0 (DUT): Info: \* ----------------------------------------------

@0 s /0 (DUT): Info: \* paddr/pmask: fff/fff

@0 s /0 (DUT): Info: \* pindex: 1

@0 s /0 (DUT): Info: \* apb\_irq\_id: 3

@0 s /0 (DUT): Info: \* hindex: 2

@0 s /0 (DUT): Info: \* abstractionLayer (LT = 8 / AT = 4): 8

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Simulation reports

Each module of the library provides execution statistics in terms of performance and communication figures. The reports are generated in the end\_of\_simulation functions of the various modules.

## AHBCTRL

The execution statistic of the AHBCTRL contains the number of successful transactions and the number of total transactions. The difference between these figures is the number of transactions, which did not complete with TLM\_OK\_RESPONSE.

The report also shows the total number of simulation cycles and the number of idle cycles. In idle cycles no master requested the bus. From this information the module calculates the bus utilization. A utilization of 1 indicates that no idle cycles occurred during simulation.

The report below was generated for an AT simulation. Information about the maximum and average arbitration time is only given for AT mode.

@400 us (ahbctrl): Report: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@400 us (ahbctrl): Report: \* AHBCtrl Statistic:

@400 us (ahbctrl): Report: \* ------------------

@400 us (ahbctrl): Report: \* Successful Transactions: 32000

@400 us (ahbctrl): Report: \* Total Transactions: 32000

@400 us (ahbctrl): Report: \*

@400 us (ahbctrl): Report: \* Simulation cycles: 40000

@400 us (ahbctrl): Report: \* Idle cycles: 7999

@400 us (ahbctrl): Report: \* Bus utilization: 0.800025

@400 us (ahbctrl): Report: \* Maximum arbiter waiting time:   
 150000001 ps (15000 cycles)

@400 us (ahbctrl): Report: \* Master with maximum waiting time: 0

@400 us (ahbctrl): Report: \* Average arbitration time / transaction:   
 84990 ps (8.499 cycles)

@400 us (ahbctrl): Report: \*

@400 us (ahbctrl): Report: \* AHB Master interface reports:

@400 us (ahbctrl): Report: \* Bytes read: 64000

@400 us (ahbctrl): Report: \* Bytes written: 64000

@400 us (ahbctrl): Report: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## APBCTRL

The APBCTRL provides information about the total number of transactions and the total number successful transactions. The report also shows how many bytes have been read and written via the AHB slave interface.

@1809990 ns (top.apbctrl): Report: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@1809990 ns (top.apbctrl): Report: \* APBCtrl Statistic:

@1809990 ns (top.apbctrl): Report: \* ------------------

@1809990 ns (top.apbctrl): Report: \* Successful Transactions: 10

@1809990 ns (top.apbctrl): Report: \* Total Transactions: 10

@1809990 ns (top.apbctrl): Report: \*

@1809990 ns (top.apbctrl): Report: \* AHB Slave interface reports:

@1809990 ns (top.apbctrl): Report: \* Bytes read: 24

@1809990 ns (top.apbctrl): Report: \* Bytes written: 16

@1809990 ns (top.apbctrl): Report: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## MCTRL

The MCTRL also provides information about the total number of transactions and the number of successful transactions. In the example below these figures are not equal. The tests of the MCTRL contains vectors, which are intended to fail to improve the test coverage.

The report also shows information about the time spent in the various SDRAM operation modes, and the volume of data transmitted via the AHB slave interface.

@476470 ns (top.mctrl): Report: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@476470 ns (top.mctrl): Report: \* Mctrl Statistic:

@476470 ns (top.mctrl): Report: \* ----------------

@476470 ns (top.mctrl): Report: \* Successful Transactions: 1672

@476470 ns (top.mctrl): Report: \* Total Transactions: 1756

@476470 ns (top.mctrl): Report: \*

@476470 ns (top.mctrl): Report: \* Time in nom. operation: 433250 ns (90.9291%)

@476470 ns (top.mctrl): Report: \* Time self-refresh mode: 0 s (0%)

@476470 ns (top.mctrl): Report: \* Time in power down mode: 17220 ns (3.61408%)

@476470 ns (top.mctrl): Report: \* Time in deep pwr down mode: 26 us (5.4568%)

@476470 ns (top.mctrl): Report: \*

@476470 ns (top.mctrl): Report: \* AHB Slave interface reports:

@476470 ns (top.mctrl): Report: \* Bytes read: 2609

@476470 ns (top.mctrl): Report: \* Bytes written: 2465

@476470 ns (top.mctrl): Report: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## Generic Memory

The Generic Memory (GM) can be configured to represent ROM, IO, SRAM and SDRAM. Depending on the intended memory layout one or multiple instances of the GM can be connected to the MCTRL. The execution report of the GM provides information about the data volume transferred to/from the particular memory.

@476470 ns (top.rom): Report: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@476470 ns (top.rom): Report: \* ROM Memory Statistic:

@476470 ns (top.rom): Report: \* -----------------------------------------

@476470 ns (top.rom): Report: \* Bytes read: 1172

@476470 ns (top.rom): Report: \* Bytes written: 1100

@476470 ns (top.rom): Report: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@476470 ns (top.io): Report: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@476470 ns (top.io): Report: \* IO Memory Statistic:

@476470 ns (top.io): Report: \* -----------------------------------------

@476470 ns (top.io): Report: \* Bytes read: 1168

@476470 ns (top.io): Report: \* Bytes written: 1164

@476470 ns (top.io): Report: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@476470 ns (top.sram): Report: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@476470 ns (top.sram): Report: \* SRAM Memory Statistic:

@476470 ns (top.sram): Report: \* -----------------------------------------

@476470 ns (top.sram): Report: \* Bytes read: 2216

@476470 ns (top.sram): Report: \* Bytes written: 2142

@476470 ns (top.sram): Report: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@476470 ns (top.sdram): Report: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@476470 ns (top.sdram): Report: \* SDRAM Memory Statistic:

@476470 ns (top.sdram): Report: \* -----------------------------------------

@476470 ns (top.sdram): Report: \* Bytes read: 837

@476470 ns (top.sdram): Report: \* Bytes written: 769

@476470 ns (top.sdram): Report: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## MMU\_CACHE

Depending on the intended configuration, the MMU\_CACHE top-level instantiates different sub-components such as caches and the mmu. All of them generate detailed execution statistics on their own. Therefore, the MMU\_CACHE execution statistic only provides information on the number of transactions and the data volume transferred over the AHB master interface.

@12640 ns /218 (mmu\_cache): Report: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@12640 ns /218 (mmu\_cache): Report: \* MMU\_CACHE Statistics:

@12640 ns /218 (mmu\_cache): Report: \* ---------------------

@12640 ns /218 (mmu\_cache): Report: \* Successful Transactions: 72

@12640 ns /218 (mmu\_cache): Report: \* Total Transactions: 72

@12640 ns /218 (mmu\_cache): Report: \*

@12640 ns /218 (mmu\_cache): Report: \* AHB Master interface reports:

@12640 ns /218 (mmu\_cache): Report: \* Bytes read: 112

@12640 ns /218 (mmu\_cache): Report: \* Bytes written: 128

@12640 ns /218 (mmu\_cache): Report: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

### Cache

The execution statistic of the caches contain information about the hits and misses in each particular cache-set. The report also shows the number of bypass operations and calculates the hit-rate for read and write operations. A hit-rate of 1 relates to a perfect cache.

The example below shows no write-hit rate, because the number of writes (with caches enabled) is zero.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@246700 ns (mmu\_cache.dcache): Report: \* Caching statistic:

@246700 ns (mmu\_cache.dcache): Report: \* -------------------

@246700 ns (mmu\_cache.dcache): Report: \* Read hits set0: 256

@246700 ns (mmu\_cache.dcache): Report: \* Read hits set1: 256

@246700 ns (mmu\_cache.dcache): Report: \* Read hits set2: 256

@246700 ns (mmu\_cache.dcache): Report: \* Read hits set3: 256

@246700 ns (mmu\_cache.dcache): Report: \* Total Read Hits: 1024

@246700 ns (mmu\_cache.dcache): Report: \* Read Misses: 1024

@246700 ns (mmu\_cache.dcache): Report: \* Read Hit Rate: 0.5

@246700 ns (mmu\_cache.dcache): Report: \* Write hits set0: 0

@246700 ns (mmu\_cache.dcache): Report: \* Write hits set1: 0

@246700 ns (mmu\_cache.dcache): Report: \* Write hits set2: 0

@246700 ns (mmu\_cache.dcache): Report: \* Write hits set3: 0

@246700 ns (mmu\_cache.dcache): Report: \* Total Write Hits: 0

@246700 ns (mmu\_cache.dcache): Report: \* Write Misses: 0

@246700 ns (mmu\_cache.dcache): Report: \* Bypass ops: 4096

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

### Localram

Two localrams, for instructions and data, can be connected to the caches. The localrams represent simple and fast storage devices. The localram execution statistic shows how many bytes have been transferred to/from a particular instance.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@34390002 ps (mmu\_cache.dlocalram): Report: \* Scratchpad statisitics:

@34390002 ps (mmu\_cache.dlocalram): Report: \* -----------------------

@34390002 ps (mmu\_cache.dlocalram): Report: \* Read accesses: 134 (Bytes: 520)

@34390002 ps (mmu\_cache.dlocalram): Report: \* Write accesses: 133 (Bytes: 516) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

### MMU

The MMU can be configured for up the 32 instruction TLBs and 32 data TLBs (split mode), or 32 shared instruction and data TLBs. The MMU execution report shows the number of hits in each particular TLB and the accumulated number of TLB misses. From this information the module calculates the TLB hit rate, which is given in percent. A hit-rate of 100% relates to a perfect MMU.

@164770 ns /16445 (mmu\_cache.mmu): Info: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@164770 ns /16445 (mmu\_cache.mmu): Info: \* MMU statistic:

@164770 ns /16445 (mmu\_cache.mmu): Info: \* -------------------

@164770 ns /16445 (mmu\_cache.mmu): Info: \* Hits in ITLB0: 0

@164770 ns /16445 (mmu\_cache.mmu): Info: \* Hits in ITLB1: 0

@164770 ns /16445 (mmu\_cache.mmu): Info: \* Hits in ITLB2: 0

@164770 ns /16445 (mmu\_cache.mmu): Info: \* Hits in ITLB3: 0

@164770 ns /16445 (mmu\_cache.mmu): Info: \* Hits in ITLB4: 0

@164770 ns /16445 (mmu\_cache.mmu): Info: \* Hits in ITLB5: 0

@164770 ns /16445 (mmu\_cache.mmu): Info: \* Hits in ITLB6: 0

@164770 ns /16445 (mmu\_cache.mmu): Info: \* Hits in ITLB7: 0

@164770 ns /16445 (mmu\_cache.mmu): Info: \* Misses in ITLB: 0

@164770 ns /16445 (mmu\_cache.mmu): Info: \* Hits in DTLB0: 1023

@164770 ns /16445 (mmu\_cache.mmu): Info: \* Hits in DTLB1: 1022

@164770 ns /16445 (mmu\_cache.mmu): Info: \* Hits in DTLB2: 1023

@164770 ns /16445 (mmu\_cache.mmu): Info: \* Hits in DTLB3: 1023

@164770 ns /16445 (mmu\_cache.mmu): Info: \* Hits in DTLB4: 1023

@164770 ns /16445 (mmu\_cache.mmu): Info: \* Hits in DTLB5: 1023

@164770 ns /16445 (mmu\_cache.mmu): Info: \* Hits in DTLB6: 1023

@164770 ns /16445 (mmu\_cache.mmu): Info: \* Hits in DTLB7: 1023

@164770 ns /16445 (mmu\_cache.mmu): Info: \* Misses in DTLB: 8

@164770 ns /16445 (mmu\_cache.mmu): Info: \* DTLB hit rate: 99%

@164770 ns /16445 (mmu\_cache.mmu): Info: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## GPTimer

The GPTimer module consists of a top level class GPtimer, which contains multiple instances of independent counters. The number of counters can be configured using a constructor parameter (ncounters). Each counter generates a small report showing the number of underflows during simulation.

@1030 ns /135 (top.timer.GPCounter): Report: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@1030 ns /135 (top.timer.GPCounter): Report: \* GPCounter Statistic:

@1030 ns /135 (top.timer.GPCounter): Report: \* ----------------

@1030 ns /135 (top.timer.GPCounter): Report: \* Counter Underflows: 4

@1030 ns /135 (top.timer.GPCounter): Report: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@1030 ns /135 (top.timer.GPCounter\_1): Report: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@1030 ns /135 (top.timer.GPCounter\_1): Report: \* GPCounter Statistic:

@1030 ns /135 (top.timer.GPCounter\_1): Report: \* ----------------

@1030 ns /135 (top.timer.GPCounter\_1): Report: \* Counter Underflows: 1

@1030 ns /135 (top.timer.GPCounter\_1): Report: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## IRQMP

The IRQMP provides 32 input lines (16 regular + 16 extended) for the connection of interrupt sources. Depending on the configuration and various internal masks, incoming interrupts are forwarded to one or multiple CPUs (max. 16).

The IRQMP execution statistic shows how many interrupts have been generated by which source, and how many interrupts have been sent to which CPU.

@164966 ns /952 (top.irqmp): Report: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@164966 ns /952 (top.irqmp): Report: \* IRQMP statistic:

@164966 ns /952 (top.irqmp): Report: \* ================

@164966 ns /952 (top.irqmp): Report: \* + IRQ Line 1: 3

@164966 ns /952 (top.irqmp): Report: \* + IRQ Line 2: 2

@164966 ns /952 (top.irqmp): Report: \* + IRQ Line 3: 3

@164966 ns /952 (top.irqmp): Report: \* + IRQ Line 4: 0

@164966 ns /952 (top.irqmp): Report: \* + IRQ Line 5: 3

@164966 ns /952 (top.irqmp): Report: \* + IRQ Line 6: 2

@164966 ns /952 (top.irqmp): Report: \* + IRQ Line 7: 2

@164966 ns /952 (top.irqmp): Report: \* + IRQ Line 8: 2

@164966 ns /952 (top.irqmp): Report: \* + IRQ Line 9: 2

@164966 ns /952 (top.irqmp): Report: \* + IRQ Line 10: 3

@164966 ns /952 (top.irqmp): Report: \* + IRQ Line 11: 2

@164966 ns /952 (top.irqmp): Report: \* + IRQ Line 12: 2

@164966 ns /952 (top.irqmp): Report: \* + IRQ Line 13: 2

@164966 ns /952 (top.irqmp): Report: \* + IRQ Line 14: 2

@164966 ns /952 (top.irqmp): Report: \* + IRQ Line 15: 2

@164966 ns /952 (top.irqmp): Report: \* + IRQ Line 16: 2

@164966 ns /952 (top.irqmp): Report: \* + IRQ Line 17: 1

@164966 ns /952 (top.irqmp): Report: \* + IRQ Line 18: 1

@164966 ns /952 (top.irqmp): Report: \* + IRQ Line 19: 1

@164966 ns /952 (top.irqmp): Report: \* + IRQ Line 20: 1

@164966 ns /952 (top.irqmp): Report: \* + IRQ Line 21: 1

@164966 ns /952 (top.irqmp): Report: \* + IRQ Line 22: 1

@164966 ns /952 (top.irqmp): Report: \* + IRQ Line 23: 1

@164966 ns /952 (top.irqmp): Report: \* + IRQ Line 24: 1

@164966 ns /952 (top.irqmp): Report: \* + IRQ Line 25: 1

@164966 ns /952 (top.irqmp): Report: \* + IRQ Line 26: 1

@164966 ns /952 (top.irqmp): Report: \* + IRQ Line 27: 1

@164966 ns /952 (top.irqmp): Report: \* + IRQ Line 28: 1

@164966 ns /952 (top.irqmp): Report: \* + IRQ Line 29: 1

@164966 ns /952 (top.irqmp): Report: \* + IRQ Line 30: 1

@164966 ns /952 (top.irqmp): Report: \* + IRQ Line 31: 1

@164966 ns /952 (top.irqmp): Report: \* -------------------------------

@164966 ns /952 (top.irqmp): Report: \* = Sum : 49

@164966 ns /952 (top.irqmp): Report: \*

@164966 ns /952 (top.irqmp): Report: \* + CPU Line 0: 49

@164966 ns /952 (top.irqmp): Report: \* + CPU Line 1: 56

@164966 ns /952 (top.irqmp): Report: \* -------------------------------

@164966 ns /952 (top.irqmp): Report: \* = Sum : 105

@164966 ns /952 (top.irqmp): Report: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

## SoCWire

The SoCWire execution statistic shows how many SoCWire packets have been sent and received. It also indicates how many errors occured during simulation. Moreover, the report shows the volume of data transferred over the SoCWire link and the number of consumed RX and TX descriptors.

@3050 ns (DUT): Report: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@3050 ns (DUT): Report: \* AHB2SoCWire Statistic:

@3050 ns (DUT): Report: \* -----------------------------------------

@3050 ns (DUT): Report: \* SoCWire packets received (with/without error,   
 with error RE\_ReceiverError,   
 without error): 1, 0, 1

@3050 ns (DUT): Report: \* Bytes received successfully on SoCWire link: 12

@3050 ns (DUT): Report: \* SoCWire packets sent (with/without error): 0

@3050 ns (DUT): Report: \* Bytes sent on SoCWire link: 0

@3050 ns (DUT): Report: \* Packets received without   
 available RX descriptor (SS\_SocWireStall): 1

@3050 ns (DUT): Report: \* RX descriptors consumed: 0

@3050 ns (DUT): Report: \* TX descriptors consumed: 0

@3050 ns (DUT): Report: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*