**Title**

HW-SW SystemC Co-Simulation SoC Validation Platform

**Power Modeling Report**

Contract No. 4200022968

by

 TU Braunschweig, Germany

August 17, 2012

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

**Table of Contents**

1 Introduction 4

1.1 Purpose and Scope 4

1.2 Revisions 4

2 Power Estimation Concept 5

3 Power Models 6

3.1 AHBCTRL 6

3.1.1 Normalized power inputs 6

3.1.2 Power estimation 6

3.1.3 Power output 7

3.2 AHBMEM 8

3.2.1 Normalized power inputs 8

3.2.2 Power estimation 8

3.2.3 Power output 9

3.3 APBCTRL 10

3.3.1 Normalized power inputs 10

3.3.2 Power estimation 10

3.3.3 Power output 11

3.4 LEON3 (Integer Unit) 12

3.4.1 Normalized power inputs 12

3.4.2 Power estimation 12

3.4.3 Power output 13

3.5 GPTimer 14

3.5.1 Normalized power inputs 14

3.5.2 Power estimation 14

3.5.3 Power output 14

3.6 IRQMP 15

3.6.1 Normalized power inputs 15

3.6.2 Power estimation 15

3.6.3 Power output 15

3.7 MCTRL 16

3.7.1 Normalized power inputs 16

3.7.2 Power estimation 16

3.7.3 Power output 17

3.7.4 Power modes 17

3.8 Memory 18

3.8.1 Normalized power inputs 18

3.8.2 Power estimation 18

3.8.3 Power output 19

3.9 MMU\_CACHE 20

3.9.1 Normalized power inputs 20

3.9.2 Power estimation 21

3.9.3 Power output 27

4 Power Monitoring/Reporting 28

4.1 Example implementation 28

4.2 Custom power reporting 28

5 SAED 90nm Technology Kit 29

**Table of Figures**

Figure 1 - Example for custom power reporting 29

**Table of Tables**

Table 1 – Revisions 4

Table 2 - AHBCTRL normalized power inputs 6

Table 3 – AHBCTRL power output interface 7

Table 4 - AHBMEM normalized power inputs 8

Table 5 – APBCTRL normalized power inputs 10

Table 6 - LEON3 normalized power inputs 12

Table 7 - GPTimer normalized power inputs 14

Table 8 - GPTimer power output interface 14

Table 9 - IRQMP normalized power inputs 15

Table 10 - MCTRL normalized power inputs 16

Table 11 – Generic memory normalized power inputs 18

Table 12 - MMU\_CACHE normalized power inputs 21

Table 13 - MMU\_CACHE power output interface 27

# Introduction

## Purpose and Scope

The purpose of this document is to describe concept and parameters of the Power Modeling techniques utilized within the HW/SW SystemC Co-Simulation SoC Validation Platform (SoCRocket).

The most prominent use cases of virtual platforms are software development, architecture exploration and system verification. Depending on the field of application the users are interested in high simulation speed, utilization and performance figures, or high accuracy. Nowadays, these use cases get more and more accompanied by the wish to estimate power consumption as early as possible and as high up as possible in the design flow. Especially modern embedded multi-processors have to consider power constraints. Particularly mobile, nomadic systems need to operate on given power budgets and are restricted to a maximum peak current. Hence, software for such systems should be written with power demands in mind.

To enable trading off power consumption against performance and latency, the SoCRocket library provides an event based power-monitoring concept.

## Revisions

|  |  |  |
| --- | --- | --- |
| **Version** | **Date** | **Description** |
| 0.1 | 08/03/12 | Initial draft for MDR/DFR meeting |
| 0.2 | 15/08/12 | Introduce new parameter based concept |
|  |  |  |

Table 1 – Revisions

# Power Estimation Concept

All core-models of the SoCRocket library provide a constructor parameter **pow\_mon**. If **pow\_mon** is enabled the power consumption of the respective module will be estimated during simulation runtime.

Three different classes of power dissipation are supported: static power, dynamic internal power and dynamic switching power. Each of them is estimated and reported separately.

The static power represents the ‘leakage’ of the component. It is more or less independent of the application running on the processor. For high-level approximations static power can be considered independent of the clock frequency. Static power linearly scales with silicon area and is strongly technology dependent. Therefore, SoCRocket simulation models contain at least one input parameter, which is supposed to be initialized with normalized leakage power information. The way of obtaining normalized leakage power is different for most components. For memories (e.g. sram, rom) it is specified in pW/bit (see chapter 3). The default settings included in the models are derived from a generic 90nm CMOS technology kit (chapter 4). The same accounts for the build-in dynamic power information.

The dynamic power of a component is composed of an internal power portion and a switching power portion. In general, dynamic power is linearly dependent on the clock frequency. The internal power can be considered independent of the application running on the processor. It is caused by different effects such as registers (D-Flops) rewriting themself, toggling input pins or memory refresh cycles. Similar to static power the models contain dedicated input parameters for normalized static power. The actual internal power is then being calculated with respect to the configuration at hand. Information about the normalized internal power of all simulation models can also be found in chapter 3. Normalization of internal power is always based on clock frequency or on clock frequency and a factor proportional to area (e.g. bits in memory).

The application dependent part of the dynamic power is the so-called switching power. Switching power is dissipated if busses, pins or storage elements change value. Since power is a unit related to an average consumption of energy over time it is not appropriate for event-based discrete measurement. Instead energy per execution or access quotas are used. The average switching power of a simulation can be obtained by counting the number of accesses, multiplying with a value representing the energy per access and dividing with the simulation time. Most library components, such as memories and busses, assign energy quotas to read/write operations, while e.g. the processor uses a fixed energy/instruction budget.

# Power Models

For any module listed in this section setting the constructor parameter **pow\_mon** enables power monitoring. Within the **leon3mp** platform this can be achieved by setting the power switch in the report section of the **json** configuration file to true or by explicitly enabling power monitoring in the platform configuration wizard.

## AHBCTRL

### Normalized power inputs

The AHBCTRL TL model provides a set of normalized power inputs for power annotation by the user. All normalized power inputs are implemented as class specific global configuration parameters (**gs\_params**):

|  |  |  |  |
| --- | --- | --- | --- |
| **Power parameter** | **Path** | **Default** | **Unit** |
| sta\_power\_norm | power.ahbctrl.sta\_power\_norm | 10714285,71 | pW/(M+S) |
| int\_power\_norm | power.ahbctrl.int\_power\_norm | 0 | uW/(M+S)/Hz |
| dyn\_read\_energy\_norm | power.ahbctrl.dyn\_read\_energy\_norm | 9,10714e-10 | uJ/access |
| dyn\_write\_energy\_norm | power.ahbctrl.dyn\_read\_energy\_norm | 9,10714e-10 | uJ/access |

Table 2 - AHBCTRL normalized power inputs

### Power estimation

If power monitoring is enabled the **start\_of\_simulation** method of the module calls the function **power\_model** for de-normalization of the given power input parameters.

The static power is calculated as follows:

Internal power of the **ahbctrl** is usually zero. However, the user can overwrite this value. De-normalization is then done:

Switching power is calculated in two steps. First the **power\_model** function de-normalizes the read/write per access energy:

During simulation runtime the numbers of read/write cycles on the bus are counted. The counters are implemented as instance specific configuration parameters and can be reset or modified at any time:

**<instance\_name>.power.dyn\_reads  
<instance\_name>.power.dyn\_writes**

The actual switching power is calculated in the **swi\_power\_cb** pre-read callback function of parameter **swi\_power**:

The module reports the average switching power over a given period of time. The default starting time of the measurement frame is the beginning of the simulation. The granularity of the estimation can be refined by setting to an arbitrary .

is implemented as an instance specific configuration parameter (**gs\_param**):

**<instance\_name>.power.power\_frame\_starting\_time**

### Power output

The current power consumption of the module can be determined at any point of simulation time. Static power, internal power and switching power are exposed in externally accessible configuration parameters. All power output parameters are implemented as instance specific configuration parameters (**gs\_params**):

|  |  |  |
| --- | --- | --- |
| **Power parameter** | **Path** | **Unit** |
| sta\_power | <instance\_name>.power.sta\_power | pW |
| int\_power | <instance\_name>.power.int\_power | uW |
| swi\_power | <instance\_name>.power\_swi\_power | uW |

Table 3 – AHBCTRL power output interface

The power output interface shown in Table 2 is equal for all models of the library. This simplifies the automatic extraction of simulation results as shown in chapter 4.

## AHBMEM

### Normalized power inputs

The AHBMEM TL model provides a set of normalized power inputs for power annotation by the user. All normalized power inputs are implemented as class specific global configuration parameters (**gs\_params**):

|  |  |  |  |
| --- | --- | --- | --- |
| **Power parameter** | **Path** | **Default** | **Unit** |
| sta\_power\_norm | power.ahbmem.sta\_power\_norm | 1269.53125 | pW/bit |
| int\_power\_norm | power.ahbmem.int\_power\_norm | 1.61011e-12 | uW/bit/Hz |
| dyn\_read\_energy\_norm | power.ahbmem.dyn\_read\_energy\_norm | 7.57408e-13 | uJ/bit^2/Hz |
| dyn\_write\_energy\_norm | power.ahbmem.dyn\_write\_energy\_norm | 7.57408e-13 | uJ/bit^2/Hz |

Table 4 - AHBMEM normalized power inputs

### Power estimation

If power monitoring is enabled the **start\_of\_simulation** method of the module calls the function **power\_model** for de-normalization of the given power input parameters.

The static power is calculated as follows:

( - Size of memory in bits)

Internal power is considered linearly dependent on size and clock frequency:

Switching power is calculated in two steps. First the **power\_model** function de-normalizes the read/write per access energy:

(- Width of memory in bits (= constant 32))

During simulation runtime the numbers of read/write cycles to/from the memory are counted. The counters are implemented as instance specific configuration parameters and can be reset or modified at any time:

**<instance\_name>.power.dyn\_reads  
<instance\_name>.power.dyn\_writes**

The actual switching power is calculated in the **swi\_power\_cb** pre-read callback function of parameter **swi\_power**:

The module reports the average switching power over a given period of time. The default starting time of the measurement frame is the beginning of the simulation. The granularity of the estimation can be refined by setting to an arbitrary .

is implemented as an instance specific configuration parameter (**gs\_param**):

**<instance\_name>.power.power\_frame\_starting\_time**

### Power output

The power output interface is equivalent to Table 3.

## APBCTRL

### Normalized power inputs

The APBCTRL TL model provides a set of normalized power inputs for power annotation by the user. All normalized power inputs are implemented as class specific global configuration parameters (**gs\_params**):

|  |  |  |  |
| --- | --- | --- | --- |
| **Power parameter** | **Path** | **Default** | **Unit** |
| sta\_power\_norm | power.apbctrl.sta\_power\_norm | 2.11e+6 | pW/S |
| int\_power\_norm | power.apbctrl.int\_power\_norm | 0 | uW/S/Hz |
| dyn\_read\_energy\_norm | power.apbctrl.dyn\_read\_energy\_norm | 5.84e-11 | uJ/access |
| dyn\_write\_energy\_norm | power.apbctrl.dyn\_read\_energy\_norm | 5.84e-11 | uJ/access |

Table 5 – APBCTRL normalized power inputs

### Power estimation

If power monitoring is enabled the **start\_of\_simulation** method of the module calls the function **power\_model** for de-normalization of the given power input parameters.

The static power is calculated as follows:

Internal power of the **apbctrl** is usually zero. However, the user can overwrite this value. De-normalization is then done:

Switching power is calculated in two steps. First the **power\_model** function de-normalizes the read/write per access energy:

During simulation runtime the numbers of read/write cycles over the bridge are counted. The counters are implemented as instance specific configuration parameters and can be reset or modified at any time:

<instance\_name>.power.dyn\_reads  
<instance\_name>.power.dyn\_writes

The actual switching power is calculated in the **swi\_power\_cb** pre-read callback function of parameter **swi\_power:**

The module reports the average switching power over a given period of time. The default starting time of the measurement frame is the beginning of the simulation. The granularity of the estimation can be refined by setting to an arbitrary .

is implemented as an instance specific configuration parameter (**gs\_param**):

**<instance\_name>.power.power\_frame\_starting\_time**

### Power output

The power output interface is equivalent to Table 3.

## LEON3 (Integer Unit)

### Normalized power inputs

The LEON TL model shipped with the library provides a set of normalized power inputs for power annotation by the user. All normalized power inputs are implemented as class specific global configuration parameters (**gs\_params**):

|  |  |  |  |
| --- | --- | --- | --- |
| **Power parameter** | **Path** | **Default** | **Unit** |
| sta\_power\_norm | power.leon3.sta\_power\_norm | 5.27e+8 | pW |
| int\_power\_norm | power.leon3.int\_power\_norm | 5.497e-8 | uW/Hz |
| dyn\_instr\_energy\_norm | power.leon3.dyn\_instr \_energy\_norm | 5.84e-11 | uJ/instr |

Table 6 - LEON3 normalized power inputs

### Power estimation

If power monitoring is enabled the **start\_of\_simulation** method of the module calls the function **power\_model** for de-normalization of the given power input parameters.

Static power is currently considered constant.

The internal power is linearly dependent on the clock frequency:

The energy per instruction is also considered constant. The actual switching power is calculated in the **swi\_power\_cb** pre-read callback function of parameter **swi\_power**:

The processor reports the average switching power over a given period of time. The default starting time of the measurement frame is the beginning of the simulation. The granularity of the estimation can be refined by setting to an arbitrary .

is implemented as an instance specific configuration parameter (**gs\_param**):

**<instance\_name>.power.power\_frame\_starting\_time**

### Power output

The power output interface is equivalent to Table 3.

## GPTimer

### Normalized power inputs

The **GPTimer** TL model provides a set of normalized power inputs for power annotation by the user. All normalized power inputs are implemented as class specific global configuration parameters (**gs\_params**):

|  |  |  |  |
| --- | --- | --- | --- |
| **Power parameter** | **Path** | **Default** | **Unit** |
| sta\_power\_norm | power.gptimer.sta\_power\_norm | 2.46e+6 | pW |
| int\_power\_norm | power.gptimer.int\_power\_norm | 1.093e-8 | uW/Hz |

Table 7 - GPTimer normalized power inputs

### Power estimation

If power monitoring is enabled the **start\_of\_simulation** method of the module calls the function **power\_model** for de-normalization of the given power input parameters.

Static power of the **GPTimer** is considered constant.

The internal power is linearly dependent on the clock frequency:

The switching power of the **GPTimer** is insignificantly low and will therefore be ignored.

### Power output

The power output interface of the **GPTimer** is shown in Table 8. In contrast to the full interface given in Table 3 the **swi\_power** parameter is missing, because switching power is ignored.

|  |  |  |
| --- | --- | --- |
| **Power parameter** | **Path** | **Unit** |
| sta\_power | <instance\_name>.power.sta\_power | pW |
| int\_power | <instance\_name>.power.int\_power | uW |

Table 8 - GPTimer power output interface

## IRQMP

### Normalized power inputs

The **IRQMP** TL model provides a set of normalized power inputs for power annotation by the user. All normalized power inputs are implemented as class specific global configuration parameters (gs\_params):

|  |  |  |  |
| --- | --- | --- | --- |
| **Power parameter** | **Path** | **Default** | **Unit** |
| sta\_power\_norm | power.irqmp.sta\_power\_norm | 3.07e+8 | pW |
| int\_power\_norm | power.irqmp.int\_power\_norm | 3.26e-10 | uW/Hz |

Table 9 - IRQMP normalized power inputs

### Power estimation

If power monitoring is enabled the **start\_of\_simulation** method of the module calls the function **power\_model** for de-normalization of the given power input parameters.

Static power of the **IRQMP** is considered constant.

The internal power is linearly dependent on the clock frequency:

The switching power of the **IRQMP** is insignificantly low and will therefore be ignored.

### Power output

The power output interface of the **GPTimer** is shown in Table 8. In contrast to the full interface given in Table 3 the **swi\_power** parameter is missing, because switching power is ignored.

## MCTRL

### Normalized power inputs

The **MCTRL** TL model provides a set of normalized power inputs for power annotation by the user. All normalized power inputs are implemented as class specific global configuration parameters (**gs\_params**):

|  |  |  |  |
| --- | --- | --- | --- |
| **Power parameter** | **Path** | **Default** | **Unit** |
| sta\_power\_norm | power.mctrl.sta\_power\_norm | 1.7e+8 | pW |
| int\_power\_norm | power.mctrl.int\_power\_norm | 1.874e-8 | uW/Hz |
| dyn\_read\_energy\_norm | power.mctrl.dyn\_read\_energy\_norm | 1.175e-8 | uJ/access |
| dyn\_write\_energy\_norm | power.mctrl.dyn\_read\_energy\_norm | 1.175e-8 | uJ/access |

Table 10 - MCTRL normalized power inputs

### Power estimation

If power monitoring is enabled the **start\_of\_simulation** method of the module calls the function **power\_model** for de-normalization of the given power input parameters.

Static power of the **MCTRL** is considered constant.

The internal power is linearly dependent on the clock frequency:

The energy per read/write access is also considered constant. The actual switching power is calculated in the **swi\_power\_cb** pre-read callback function of parameter **swi\_power**:

During simulation runtime the numbers of read/write cycles are counted at the AHB bus interface. The counters are implemented as instance specific configuration parameters and can be reset or modified at any time:

<instance\_name>.power.dyn\_reads  
<instance\_name>.power.dyn\_writes

The module reports the average switching power over a given period of time. The default starting time of the measurement frame is the beginning of the simulation. The granularity of the estimation can be refined by setting to an arbitrary .

is implemented as an instance specific configuration parameter (gs\_param):

**<instance\_name>.power.power\_frame\_starting\_time**

### Power output

The power output interface is equivalent to Table 3.

### Power modes

The MCTRL supports multiple low-power operation modes. Currently only the nominal operation mode is supported for power estimation.

## Memory

### Normalized power inputs

The generic memory TL models (**mapmemory** and **arraymemory**) provide a set of normalized power inputs for power annotation by the user. All normalized power inputs are implemented as class specific global configuration parameters (**gs\_params**):

|  |  |  |  |
| --- | --- | --- | --- |
| **Power parameter** | **Path** | **Default** | **Unit** |
| sta\_power\_norm | power.{io,prom,sram,sdram}\*. sta\_power\_norm | io/prom/sram: 1269.53125 sdram: 2539.0625 | pW/bit |
| int\_power\_norm | power.{io,prom,sram,sdram}. int\_power\_norm | io/prom/sram: 1.61011e-6 sdram: 3.22021e-6 | uW/bit/Hz |
| dyn\_read\_energy\_norm | power.{io,prom,sram,sdram}. dyn\_read\_energy\_norm | io/prom/sram: 7.57408e-13 sdram: 15e-13 | uJ/bit^2/Hz |
| dyn\_write\_energy\_norm | power.{io,prom,sram,sdram}. dyn\_write\_energy\_norm | io/prom/sram: 7.57408e-13 sdram: 15e-13 | uJ/bit^2/Hz |

Table 11 – Generic memory normalized power inputs

\*The mapmemory and arraymemory generic memories can be configured to act as IO, PROM, SRAM or SDRAM. For this study only SRAM models have been available in hardware. Therefore, the default normalized input power for PROM and IO is equal to SRAM. The input power for SDRAM has been set to 2x the power of SRAM. The user can overwrite all these values, if appropriate memories are at hand.

### Power estimation

If power monitoring is enabled the **start\_of\_simulation** method of the module calls the function **power\_model** for de-normalization of the given power input parameters.

The static power is calculated as follows:

( - Size of memory in bits)

Internal power is considered linearly dependent on size and clock frequency:

Switching power is calculated in two steps. First the **power\_model** function de-normalizes the read/write per access energy:

(- Width of memory in bits (= constant 32))

During simulation runtime the numbers of read/write cycles to/from the memory are counted. The counters are implemented as instance specific configuration parameters and can be reset or modified at any time:

**<instance\_name>.power.dyn\_reads  
<instance\_name>.power.dyn\_writes**

The actual switching power is calculated in the **swi\_power\_cb** pre-read callback function of parameter **swi\_power**:

The module reports the average switching power over a given period of time. The default starting time of the measurement frame is the beginning of the simulation. The granularity of the estimation can be refined by setting to an arbitrary .

is implemented as an instance specific configuration parameter (**gs\_param**):

**<instance\_name>.power.power\_frame\_starting\_time**

### Power output

The power output interface is equivalent to Table 3.

## MMU\_CACHE

### Normalized power inputs

The **MMU\_CACHE** is a complex hierarchical simulation model. Depending on the configuration it or may not contain multiple sub-components such as: i/d caches, i/d localrams and memory management unit (**mmu**). For all those sub-components power is estimated separately.

Normalized power inputs are implemented as class specific global configuration parameters (**gs\_params**):

|  |  |  |  |
| --- | --- | --- | --- |
| **MMU\_CACHE Top-level** | | | |
| **Power parameter** | **Path** | **Default** | **Unit** |
| sta\_power\_norm | power.mmu\_cache.sta\_power\_norm | 1.16e+8 | pW |
| int\_power\_norm | power.mmu\_cache.int\_power\_norm | 0 | uW/Hz |
| dyn\_read\_energy\_norm | power.mmu\_cache.dyn\_read\_energy\_norm | 1.465e-8 | uJ |
| dyn\_write\_energy\_norm | power.mmu\_cache.dyn\_write\_energy\_norm | 1.465e-8 | uJ |
| **Data cache (dvectorcache)** | | | |
| sta\_power\_norm | power.mmu\_cache. dcache.sta\_power\_norm | 1.35e+8 | pW |
| int\_power\_norm | power.mmu\_cache. dcache.int\_power\_norm | 1.264e-8 | uW/Hz |
| sta\_dtag\_power\_norm | power.mmu\_cache. dcache.dtag.sta\_power\_norm | 1726.5625 | pW/bit |
| int\_dtag\_power\_norm | power.mmu\_cache. dcache.dtag.int\_power\_norm | 1.6954e-12 | uW/bit/Hz |
| dyn\_dtag\_read\_energy\_norm | power.mmu\_cache. dcache.dtag.dyn\_read\_energy\_norm | 1.0149e-12 | uJ/bit^2/Hz |
| dyn\_dtag\_write\_energy\_norm | power.mmu\_cache. dcache.dtag.dyn\_write\_energy\_norm | 1.0149e-12 | uJ/bit^2/Hz |
| sta\_ddata\_power\_norm | power.mmu\_cache. dcache.ddata.sta\_power\_norm | 1269.53125 | pW/bit |
| int\_ddata\_power\_norm | power.mmu\_cache. dcache.ddata.int\_power\_norm | 1.6101e-12 | uW/bit/Hz |
| dyn\_ddata\_read\_energy\_norm | power.mmu\_cache. dcache.ddata.dyn\_read\_energy\_norm | 7.5740e-13 | uJ/bit^2/Hz |
| dyn\_ddata\_write\_energy\_norm | power.mmu\_cache. dcache.ddata.dyn\_write\_energy\_norm | 7.5740e-13 | uJ/bit^2/Hz |
| **Instruction cache (ivectorcache)** | | | |
| sta\_power\_norm | power.mmu\_cache. icache.sta\_power\_norm | 1.10e+8 | pW |
| int\_power\_norm | power.mmu\_cache. icache.int\_power\_norm | 1.381e-8 | uW/Hz |
| sta\_itag\_power\_norm | power.mmu\_cache. icache.itag.sta\_power\_norm | 1269.53125 | pW/bit |
| int\_itag\_power\_norm | power.mmu\_cache. icache.itag.int\_power\_norm | 1.6101e-12 | uW/bit/Hz |
| dyn\_itag\_read\_energy\_norm | power.mmu\_cache. icache.itag.dyn\_read\_energy\_norm | 7.5740e-13 | uJ/bit^2/Hz |
| dyn\_itag\_write\_energy\_norm | power.mmu\_cache. icache.itag.dyn\_write\_energy\_norm | 7.5740e-13 | uJ/bit^2/Hz |
| sta\_idata\_power\_norm | power.mmu\_cache. icache.idata.sta\_power\_norm | 1269.53125 | pW/bit |
| int\_idata\_power\_norm | power.mmu\_cache. icache.idata.int\_power\_norm | 1.6101e-12 | uW/bit/Hz |
| dyn\_idata\_read\_energy\_norm | power.mmu\_cache. icache.idata.dyn\_read\_energy\_norm | 7.5740e-13 | uJ/bit^2/Hz |
| dyn\_idata\_write\_energy\_norm | power.mmu\_cache. icache.idata.dyn\_write\_energy\_norm | 7.5740e-13 | uJ/bit^2/Hz |
| **I/D Scratchpad (localrams)** | | | |
| sta\_power\_norm | power.mmu\_cache. localram.sta\_power\_norm | 1269.53125 | pW/bit |
| int\_power\_norm | power.mmu\_cache localram.int\_power\_norm | 1.6101e-12 | uW/bit/Hz |
| dyn\_read\_energy\_norm | power.mmu\_cache. localram.dyn\_read\_energy\_norm | 7.5740e-13 | uJ/bit^2/Hz |
| dyn\_write\_energy\_norm | power.mmu\_cache. localram.dyn\_write\_energy\_norm | 7.5740e-13 | uJ/bit^2/Hz |
| **Memory management unit (MMU)** | | | |
| sta\_power\_norm | power.mmu\_cache. mmu.sta\_power\_norm | 7.19e+7 | pW |
| int\_power\_norm | power.mmu\_cache. mmu.int\_power\_norm | 3.74e-8 | uW/Hz |
| sta\_tlb\_power\_norm | power.mmu\_cache. mmu.sta\_tlb\_power\_norm | 6543750 | pW/tlb |
| int\_tlb\_power\_norm | power.mmu\_cache. mmu.int\_tlb\_power\_norm | 2.7225e-9 | pW/tlb/Hz |
| dyn\_tlb\_read\_energy\_norm | power.mmu\_cache. mmu.dyn\_tlb\_read\_energy\_norm | 1.0812e-11 | uJ/tlb |
| dyn\_tlb\_write\_energy\_norm | power.mmu\_cache. mmu.dyn\_tlb\_write\_energy\_norm | 1.0812e-11 | uJ/tlb |

Table 12 - MMU\_CACHE normalized power inputs

(dcache tag ram is dual-port ram; dcache data, icache tag and icache data are single-port)

### Power estimation

Setting the **pow\_mon** constructor parameter of the top-level class enables power monitoring for all sub-components of **mmu\_cache**. Each sub\_component contains a function **power\_model**, which is triggered at **start\_of\_simulation**.

#### MMU\_CACHE top-level

Static power of the **mmu\_cache** top-level is considered constant.

The internal power is linearly dependent on the clock frequency:

The energy per read/write access is also considered constant. The actual switching power is calculated in the **swi\_power\_cb** pre-read callback function of parameter **swi\_power**:

During simulation runtime the numbers of read/write accesses from the processor interface are counted. The counters are implemented as instance specific configuration parameters and can be reset or modified at any time:

**<instance\_name>.power.dyn\_reads  
<instance\_name>.power.dyn\_writes**

The module reports the average switching power over a given period of time. The default starting time of the measurement frame is the beginning of the simulation. The granularity of the estimation can be refined by setting to an arbitrary .

is implemented as an instance specific configuration parameter (**gs\_param**)**:**

**<instance\_name>.power.power\_frame\_starting\_time**

#### Data Cache

The main contributors to the data cache static power are **dcache** controller, data tag rams (**dtag**) and data cache rams (**ddata**). The static power of the controller is considered constant. The **dtag** ram and the **ddata** ram power linearly dependent on the memory size in bits. The data cache can have up to four parallel sets ().

The internal power of the data cache consists of the **dcache** controller internal power, the **dtag** internal power and the **ddata** internal power. The internal power of the controller scales linearly with the clock frequency. Internal power of **dtag** rams and **ddata** rams linearly depends on clock frequency and memory size in bits.

Switching power is calculated in two steps. First the **power\_model** function denormalizes the read/write per access energy:

(- Width of memory in bits (= constant 32))

During simulation runtime the numbers of read/write cycles to/from **dtag** rams and data cache rams are counted. The counters are implemented as instance specific configuration parameters and can be reset or modified at any time:

**<instance\_name>.dcache.power.dyn\_tag\_reads  
<instance\_name>.dcache.power.dyn\_tag\_writes**

**<instance\_name>.dcache.power.dyn\_data\_reads  
<instance\_name>.dcache.power.dyn\_data\_writes**

The actual switching power is calculated in the **swi\_power\_cb** pre-read callback function of parameter **swi\_power**:

The module reports the average switching power over a given period of time. The default starting time of the measurement frame is the beginning of the simulation. The granularity of the estimation can be refined by setting to an arbitrary .

is implemented as an instance specific configuration parameter (**gs\_param**):

**<instance\_name>.dcache.power.power\_frame\_starting\_time**

#### Instruction Cache

The main contributors to the instruction cache static power are **icache** controller, instruction tag rams (**itag**) and instruction cache rams (**idata**). The static power of the controller is considered constant. The **itag** ram and the **idata** ram power linearly dependent on the memory size in bits. The instruction cache can have up to four parallel sets ().

The internal power of the data cache consists of the **icache** controller internal power, the **itag** internal power and the **idata** internal power. The internal power of the controller scales linearly with the clock frequency. Internal power of **itag** rams and **idata** rams linearly depends on clock frequency and memory size in bits.

Switching power is calculated in two steps. First the **power\_model** function de-normalizes the read/write per access energy:

(- Width of memory in bits (= constant 32))

During simulation runtime the numbers of read/write cycles to/from **itag** rams and instruction cache rams are counted. The counters are implemented as instance specific configuration parameters and can be reset or modified at any time:

**<instance\_name>.icache.power.dyn\_tag\_reads  
<instance\_name>.icache.power.dyn\_tag\_writes**

**<instance\_name>.icache.power.dyn\_data\_reads  
<instance\_name>.icache.power.dyn\_data\_writes**

The actual switching power is calculated in the **swi\_power\_cb** pre-read callback function of parameter **swi\_power**:

The module reports the average switching power over a given period of time. The default starting time of the measurement frame is the beginning of the simulation. The granularity of the estimation can be refined by setting to an arbitrary .

is implemented as an instance specific configuration parameter (**gs\_param**):

**<instance\_name>.icache.power.power\_frame\_starting\_time**

#### Localrams

If power monitoring is enabled in the **mmu\_cache** top-level class the start\_of\_simulation method of **localram** triggers the function **power\_model** for de-normalization of the given power input parameters at the beginning of the simulation. For power modeling instruction **localram** and data **localram** are considered equal. Therefore, no further distinctions are made.

The static power is calculated as follows:

( - Size of localram in bits)

Internal power is considered linearly dependent on size and clock frequency:

Switching power is calculated in two steps. First the **power\_model** function de-normalizes the read/write per access energy:

(- Width of memory in bits (= constant 32))

During simulation runtime the numbers of read/write cycles to/from the memory are counted. The counters are implemented as instance specific configuration parameters and can be reset or modified at any time:

**<instance\_name>.{i.d}localram.power.dyn\_reads  
<instance\_name>.{i,d}localram.power.dyn\_writes**

The actual switching power is calculated in the **swi\_power\_cb** pre-read callback function of parameter **swi\_power**:

The module reports the average switching power over a given period of time. The default starting time of the measurement frame is the beginning of the simulation. The granularity of the estimation can be refined by setting to an arbitrary .

is implemented as an instance specific configuration parameter (**gs\_param**):

**<instance\_name>.{i,d}localram.power.power\_frame\_starting\_time**

#### Memory Management Unit (MMU)

The **MMU** static power consists of the **MMU** logic static power and the TLB static power. The static power of the **MMU** logic is considered constant. The TLB static power linearly dependents on the number of instruction and data TLBs.

- Number of instruction/data TLBs)

Respectively, the internal power of the **MMU** consists of the **MMU** logic internal power and the **TLB** internal power. The internal power of the logic scales linearly with the clock frequency. The **TLB** internal power linearly depends on clock frequency and number of instruction and data **TLBs**.

Switching power is calculated in two steps. First the **power\_model** function de-normalizes the **TLB** read/write per access energy:

During simulation runtime the numbers of read/write to instruction and data **TLBs** are counted. The counters are implemented as instance specific configuration parameters and can be reset or modified at any time:

**<instance\_name>.mmu.power.dyn\_itlb\_reads  
<instance\_name>.mmu.power.dyn\_itlb\_writes**

**<instance\_name>.mmu.power.dyn\_dtlb\_reads  
<instance\_name>.mmu.power.dyn\_dtlb\_writes**

The actual switching power is calculated in the **swi\_power\_cb** pre-read callback function of parameter **swi\_power**:

The module reports the average switching power over a given period of time. The default starting time of the measurement frame is the beginning of the simulation. The granularity of the estimation can be refined by setting to an arbitrary .

is implemented as an instance specific configuration parameter (**gs\_param**):

**<instance\_name>.mmu.power.power\_frame\_starting\_time**

### Power output

Power is reported for every sub-component of **mmu\_cache** separately. An overview about all available power outputs is given in Table 13.

|  |  |  |
| --- | --- | --- |
| **MMU\_CACHE top-level** | | |
| **Power parameter** | **Path** | **Unit** |
| sta\_power | <instance\_name>.power.sta\_power | pW |
| int\_power | <instance\_name>.power.int\_power | uW |
| swi\_power | <instance\_name>.power\_swi\_power | uW |
| **Data cache (dvectorcache)** | | |
| sta\_power | <instance\_name>.dcache. power.sta\_power | pW |
| int\_power | <instance\_name>.dcache. power.int\_power | uW |
| swi\_power | <instance\_name>.dcache. power.swi\_power | uW |
| **Instruction cache (ivectorcache)** | | |
| sta\_power | <instance\_name>.icache. power.sta\_power | pW |
| int\_power | <instance\_name>.icache. power.int\_power | uW |
| swi\_power | <instance\_name>.icache. power.swi\_power | uW |
| **I/D scratchpad (localrams)** | | |
| sta\_power | <instance\_name>.{i,d}localram. power.sta\_power | pW |
| int\_power | <instance\_name>.{i,d}localram. power.int\_power | uW |
| swi\_power | <instance\_name>.{i,d}localram. power.swi\_power | uW |
| **Memory management unit (MMU)** | | |
| sta\_power | <instance\_name>.mmu. power.sta\_power | pW |
| int\_power | <instance\_name>.mmu. power.int\_power | uW |
| swi\_power | <instance\_name>.mmu. power.swi\_power | uW |

Table 13 - MMU\_CACHE power output interface

# Power Monitoring/Reporting

## Example implementation

The power modeling concept implemented in the SoCRocket library allows to monitor and report power consumption at different levels of accuracy. Given the presented interfaces it is easy to implement custom power reporting tools. An example implementation is given in:

**./common/powermonitor.{h,cpp}**

The example powermonitor is integrated in the leon3mp platform. It can be configured to report the average power consumption of all modules in the system at a given instance in time (constructor parameter report\_time). To activate power reporting enable the **p\_report\_power** switch.

The core functionality of the module is implemented in function **gen\_report**. It operates in the following way:

1. Obtain pointer to GreenControl API:   
   **gs::cnf::cnf\_api \*mApi = gs::cnf::Gcnf\_Api::get\_ApiInstance(NULL)**
2. Collect all registered parameters related to power consumption in **std::vector power\_list**
3. From **power\_list** select all power output parameters belonging to the same instance
4. Print report for static, internal and switching power of the instance (if exists):  
   e.g. **mApi->getValue(std::string(get\_model\_name(models\_list[0] + “.power.sta\_power”), model\_sta\_power)**
5. Accumulate power values for global statistics and remove instance from power\_list
6. Continue with next model from **power\_list** (3)
7. If **power\_list** empty print global statistics

## Custom power reporting

Most of the models listed in chapter 3 report switching power. All of them contain a set of activity counters, e.g. for counting the number of reads or writes to a memory. These counters are implemented as externally visible configuration parameters, which means they can be modified/reset during simulation. Moreover, all models reporting switching power provide the parameter **power\_frame\_starting\_time**. The latter can also be altered at runtime.

At default **power\_frame\_starting\_time** is initialized with **SC\_ZERO\_TIME**, which is the starting time of the simulation. The power reported by a model’s switching power output parameter always relates to the average switching power consumed in the time interval since **power\_frame\_starting\_time**. Calculation of the average switching power is done inside the models using pre-read callback functions (**swi\_power\_cb**). This means, at every read access to a switching power output parameter, switching power is calculated from normalized per-access energy, number of accesses and

- power\_frame\_starting\_time)

Given this mechanism power profiles over time for whole applications can be generated at different granularity, by e.g. (Figure 1):

* Continuously reading power outputs at regular intervals (as shown in 4.1)
* After each read set **power\_frame\_starting\_time** of observed modules to current simulation time
* After each read reset the activity counters



Figure 1 - Example for custom power reporting

**Remark:** More information about handling GreenControl parameter can be found in the SoCRocket Library User Manual.

# SAED 90nm Technology Kit

To estimate the power consumption of the various models, all relevant GRLIB components have been synthetized using a generic design kit for 90nm CMOS technology.

The design kit can be obtained from Synopsys without charge. It contains 90nm standard cells, memory models, a phase-locked loop, io pads and various special components such as level shifters.

Power estimation was done using Cadence RTL Compiler Version 10.10.100. Base for the normalized default power values given in chapter 3 is the LEON3MP reference design, which is also a part of the design kit.

**Remark:** The given default normalized power inputs can be used to obtain indicators for the impact of architectural decisions on the system power consumption. However, it has to be kept in mind that the technology used for parameter fitting is plain generic. Especially the per-memory-access energy figures can largely differ. In order to obtain results closer to actual hardware it is essential to annotate the design with appropriate data.