**Title**

HW-SW SystemC Co-Simulation SoC Validation Platform

**Power Modeling Report**

Contract No. 4200022968

by

 TU Braunschweig, Germany

August 16, 2012

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

**Table of Contents**

1 Introduction 3

1.1 Purpose and Scope 3

1.2 Revisions 3

2 Power Estimation Concept 4

3 Power Models 5

3.1 AHBCTRL 5

3.2 AHBMEM 5

3.3 APBCTRL 5

3.4 LEON3 5

3.5 GPTimer 5

3.6 IRQMP 5

3.7 MCTRL 5

3.8 Memory 5

3.9 MMU\_CACHE 5

3.9.1 Scratchpad RAM 5

3.9.2 Data-Cache 5

3.9.3 Instruction-Cache 5

3.9.4 MMU 5

4 Power Monitoring/Reporting 5

5 SAED 90nm Technology Kit 6

**Table of Figures**

**Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.**  
Wählen Sie im Dokument die Wörter aus, die in das Inhaltsverzeichnis eingeschlossen werden sollen, und klicken Sie dann in der Formatierungspalette unter "Formatvorlagen" auf ein Überschriftenformat. Wiederholen Sie dies für alle einzuschließenden Überschriften, und fügen Sie das Inhaltsverzeichnis dann in das Dokument ein. Sie können ein Inhaltsverzeichnis auch erstellen, indem Sie auf die Option "Manuelle Formatierung" klicken und die Einträge dann manuell eingeben.

**Table of Tables**

Table 1 - Revisions 2

# 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

The SoCRocket simulation models 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 also be considered independent of the clock frequency. Static power linearly scales with silicon area and is strongly technology dependent. Therefore, SoCRocket modules usually 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 modules. 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 modules 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 estimating switching power. 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 the energy per access constant and dividing with the simulation time. Counting accesses to units or components can be costly in terms of simulation performance. We propose to apply a rather coarse approach. Most library components, such as memories and busses, assign energy quotas to read/write operations. The processor uses a fixed energy/instruction budget (see chapter 3).

# Power Models

Power estimation for any module listed in this section can be enabled by setting the constructor parameter pow\_mon. 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 denormalization 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. Denormalization is then done:

Switching power is calculated in two steps. First the power\_model function denormalizes 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 denormalization 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 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 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 denormalization 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. Denormalization is then done:

Switching power is calculated in two steps. First the power\_model function denormalizes 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 denormalization 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 denormalization 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 denormalization 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 denormalization 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 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 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.{mapmemory, arraymemory}.sta\_power\_norm | 1269.53125 | pW/bit |
| int\_power\_norm | power.{mapmemory, arraymemory}.int\_power\_norm | 1.61011e-12 | uW/bit/Hz |
| dyn\_read\_energy\_norm | power.{mapmemory, arraymemory}.dyn\_read\_energy\_norm | 7.57408e-13 | uJ/bit^2/Hz |
| dyn\_write\_energy\_norm | power.{mapmemory. arraymemory}.dyn\_write\_energy\_norm | 7.57408e-13 | uJ/bit^2/Hz |

Table 11 - 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 denormalization 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 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 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

#### Power estimation

#### Power output

# Power Monitoring/Reporting

# SAED 90nm Technology Kit