

# Energy Evaluation Methodology for Platform Based System-On-Chip Design

Kristian Hildingsson<sup>1</sup>, Tughrul Arslan<sup>1,2</sup>, and Ahmet T. Erdogan<sup>2</sup>

<sup>1</sup>*The Institute for System Level Integration, Livingston, Scotland*

<sup>2</sup>*The University of Edinburgh, School of Engineering and Electronics, Scotland*

*khoh@sli-institute.ac.uk, {ta, ate}@ee.ed.ac.uk*

## Abstract

*This paper presents a methodology that speeds up the process of estimating the system level energy efficiency for synthesisable AMBA based SOC platforms that are scalable by means of integrating additional Intellectual Property (IP) hardware through the AMBA bus system. The methodology facilitates a modular approach where overall energy consumption is calculated based on Power Models that are developed for each defined sub-block (or Entity) of the platform. To automate the evaluation process we developed a tool – called PEX – that combines Power Models with utilisation statistics of the system Entities, and that provides a fast way of analysing tradeoffs for speed, energy and power consumption for various system configurations. Detailed results are presented for the analysis of two implementation scenarios of the Rijndael AES algorithm using the SPARC V8 compatible LEON architecture.*

## 1. Introduction

Power and energy are crucial performance metrics in System-On-Chip (SOC) design, especially for handheld devices, such as PDAs, relying on a battery as the single source of energy. Optimising the system performance, including power and energy, requires a system modelling capability to efficiently evaluate various design scenarios and the influence of design techniques for low power. For a SOC platform this means modelling of a heterogeneous system comprising CPU, memories, hardware accelerators, interconnect, and peripherals as well as software running on the hardware platform. By *platform* we mean a microprocessor based architecture that can be extended and re-configured depending on the application.

A crucial trade-off in power modelling is that between speed and accuracy where traditional low-level modelling is accurate but slow, and high-level modelling is faster but less accurate. Research in power modelling aims to find fast and accurate modelling techniques at higher levels of abstraction. Commercial EDA tools for power estimation are available at lower abstraction levels such as Synopsys' transistor-level tool Nanosim<sup>TM</sup> and Sequence Design's

RT-level tool PowerTheatre<sup>TM</sup>. There are no widely used commercial power evaluation tools that operate purely at the system-level.

Previous work on power modelling ranges from RT and architectural levels through to software and compiler levels and also system-level targeting embedded systems and SOC design. The frameworks established in [1][2] models the system-level energy consumption, combining energy models for caches, main memory, processor core, and additional hardware resources. Furthermore, in [1], suitable functions within various DSP algorithms were detected and implemented in hardware instead of software, leading to significant energy reductions at system-level, and in [2], techniques were developed for increasing the system-level simulation efficiency for power estimation based on multiple power estimators. An instruction-set simulator was extended in [3], with energy models for processor, memories, interconnect and DC-DC converter facilitating board-level energy analysis. Power models were constructed based on data sheets and hardware measurements. The methodology was used to estimate system energy behaviour in the process of optimising the implementation of an MPEG algorithm. The work in [4] describes a core-based approach for system-level power modelling, where a core is a reusable system-level component, e.g. a microprocessor, a co-processor, or a peripheral, designed to become part of a SOC. Core models (designed to mimic the behaviour of the cores) are parameterisable and executable, where core parameters influence the performance, power and area of the core. A system-level simulator was constructed based on the core models. In [5] a software power evaluation framework for estimating power and energy for the MicroSparcIIep architecture was investigated. Energy was estimated within 5% and power within 15%, at orders of magnitude simulation speed-up, compared to full gate-level estimations.

In this paper, we describe a methodology for predicting the system-level energy-efficiency when integrating IP modules onto an existing synthesisable SOC platform, based around the AMBA bus standard [6]. Our approach is similar to the work in [1] and [3], but with the important

difference that we target a completely synthesisable platform in VHDL that implements a widely used bus standard. Like [1], we address the aspect of moving software functionality into hardware, however we assume the existence of IP modules with dedicated functionality, and we take into account the IP's complexity when developing its power model. Unlike [4] we create the link between events and power models using the instruction-set simulator, automatically creating a unified cycle-accurate evaluation engine for the complete system. We only use widely accepted industry standard EDA tools for design and development of entity power models. The remaining part of the paper outlines the methodology, and then describes how we apply it to the AMBA based SPARC V8 compatible LEON architecture [7] to evaluate the energy efficiency of implementing the Rijndael AES algorithm in pure software versus a HW/SW co-implementation.

## 2. Methodology outline

This section outlines the methodology in a generic manner, describing the terminology used and the steps taken to apply the methodology to a particular platform. The methodology is based on three terms: **Entity**, **Power Model**, and **Event** (defined in the next three paragraphs).

An *entity* is an arbitrary hardware portion with specified attributes, parameters and costs. The attributes refer to information on complexity (gate count), interface (inputs, outputs), operating modes (functionality) and implementation (technology); the attributes are therefore fixed when the entity has been implemented, but can vary between implementations. The parameters are run-time characteristics determining the operating mode, input data, operating speed and operating voltage for applications that utilise voltage scaling methods. The costs of operating an entity are represented by three metrics: execution time (clock cycles or real time), power consumption (Watt) and energy consumption (Joule).

A *power model* is an abstract notion of an entity's costs and is developed with respect to the corresponding attributes and parameters, where data for power consumption and execution time is included in a look-up table or a cost function that covers the possible modes of operation of the entity. Power data are obtained from low-level simulations (or measurements) and should only have to be obtained once for any configuration of entity attributes.

An *event* represents hardware activity and activates one or more entities with parameters as arguments. Events are defined at any abstraction level (e.g. an ALU operation or a functional call in C) as long as they activate hardware resources and therefore trigger entity costs. Events can trigger more than one entity operation in which case more than one power model is used for that event.

The methodology allows for a flexible interpretation of its three fundamental terms, where entities, events and power models are *defined* and *developed* depending on available resources and requirements for accuracy and speed of modelling. To evaluate over-all system performance, a mechanism of linking events to power models is constructed. In [4] the link mechanism between events and power models was facilitated through the executable models of core components, incorporating look-up tables for energy costs. In summary, our methodology encapsulates four main steps:

- 1) Partition the system and define system entities – *see Section 4*
- 2) Develop power models that reflect the cost of operating the entities – *see Section 5*
- 3) Define events that capture the activity of entities – *see Section 6*
- 4) Construct a mechanism that links power models and events in order to evaluate system-level energy behaviour – *see Section 8*

## 3. Platform definition

The platform architecture is shown in Figure 1 [8], also indicating the possibility of including additional IP modules connected to the AMBA AHB bus. LEON is compliant with the 32-bit SPARC V8 architecture (the IEEE-1754 standard) [7]. The main features of the LEON microprocessor includes: hardware units for multiplication, division and Multiply-And-Accumulate (MAC) operations, interface to Floating-Point Unit (FPU) and custom Co-Processor (CP), and separate instruction and data cache (Harvard architecture). It also incorporates the AMBA 2.0 on-chip bus architecture [6].

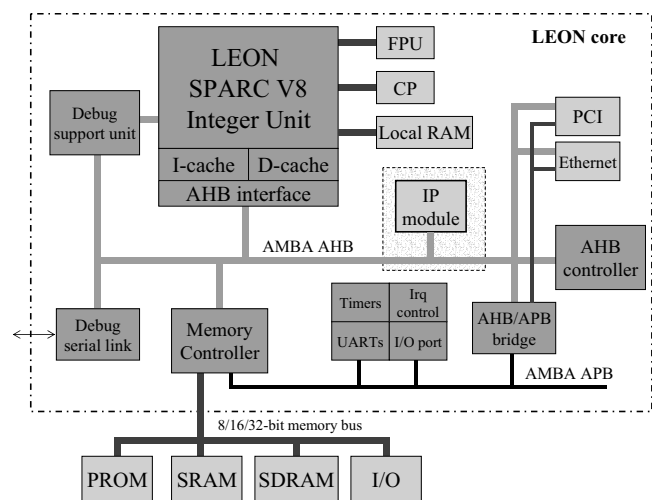


Figure 1: Platform architecture

The memory controller handles 8/16/32-bits wide buses for external memories. There is also support for extra local RAM for performance critical code.

#### 4. Entities

The entity partitioning process is aimed at finding a flexible approach to model the system for various configurations, including extensions through additional IP modules. Since some modules of the LEON core represents optional functionality we define the main *processor entity* to consist of: Integer Unit, AHB interface, memory controller, AHB controller, AHB/APB bridge, timers, UART, and IRQ controller. The remaining modules are considered separate entities; these are the FPU, CP, PCI, Ethernet, cache/local/external memories, and I/O. This means that any extra module added to the AMBA bus will be considered a *new entity* of the system, as illustrated by the IP module in Figure 1.

The entities and corresponding attributes for the basic platform used in this work are shown in Table 1. We do not use the FPU or the CP here. By synthesising the LEON core for the UMC 0.18µm technology using Synopsys Design Compiler we establish the attributes for this entity. The transistor count is shown as the complexity attribute, which we get from compiling the gate-level netlist in Synopsys Nanosim.

The functionality of the processor entity is described by its operating modes, which are the machine instructions as defined by the SPARC V8 instruction-set architecture. The parameters of the processor entity are the operands of the machine instructions, e.g. the load instruction `ld [rs1], rd` describing the functionality of loading a 32-bit word from memory address `[rs1]` into register `rd` of the register file. The other entities are the memory blocks (all SRAM type), where the *complexity* attribute is shown in Bytes. The *interface* attributes are indicated as the number of ports (obtained from data sheets). At this stage we consider the only operating modes for memory blocks to be either read or write. The *parameters* for these entities are the data being transferred on the data buses.

The next step of the methodology is to determine the costs of operating the entities, i.e. the power models.

**Table 1: System entities for basic platform**

Entity	Complexity	Interface	Op. modes	Tech.
Processor core	172.5k transistors	491 ports	SPARC instructions	UMC 0.18um
Reg. file	544 Bytes	84 ports	Read/write	“
I-cache	4 kBytes	78 ports	Read/write	“
D-cache	4 kBytes	78 ports	Read/write	“
I/D-\$ tags	2x768 Bytes	2*60 ports	Read/write	“
Main mem.	2x128 kBytes	32-bit data	Read/write	“

## 5. Power models

The power models are developed with respect to the attributes and parameters of the system entities. This process depends on available resources and requirements of modelling accuracy, where models can be developed incrementally. This section describes the power models that we currently use for the various entities of the LEON platform.

### 5.1. Processor entity

A commonly used method to obtain the energy cost per instruction for CPUs is described in [9]. There are more recently proposed methods to obtain the power per instruction, such as the one suggested in [10], where regression macro-modelling eliminates the construction of detailed assembly sequences as in [9].

In this work we use a variant of the method described in [9]. Each instruction is assigned a basic cost which is used to calculate power for all the cycles during which the instruction occupies the execution stage of the CPU pipeline, meaning that stall cycles are included as well. In addition, the basic costs have minimum and maximum bounds that are dependent on the data in the instruction operands (i.e. the entity parameters), a technique used also in our previous work [11]. The inter-instruction cost is estimated for pairs of instructions and is included as a constant factor in the calculations. The process of obtaining min and max conditions is straightforward for arithmetic, logic, shift, load, and store instructions, but less so for branch and control instructions. Finding the *true* min and max conditions for a particular instruction requires careful analysis of the possible combinations of instruction operands and parameters. Often it is enough to make approximations; as for example in this work where the min cost for the `ld` instruction was obtained for moving zero data from memory into a local register with zero value. The maximum cost was obtained by creating switching activity on all bit-lines such as moving `0x00000000` followed by `0xFFFFFFFF`, or `0x5A5A5A5A` followed by `0xA5A5A5A5`.

In order to obtain the power consumption data for the processor entity we performed gate-level power estimations in Design Compiler, by back-annotating the switching-activity information file (SAIF) obtained from gate-level simulation in Modelsim. For increased accuracy, clock tree generation was performed separately in Cadence Silicon Ensemble. Sequences of instructions for min and max conditions were written in assembly and simulated at the gate-level. Table 2 shows a sub-set of the instructions (i.e. the operating modes of the processor entity) and their associated costs, i.e. average power and execution time.

**Table 2: Processor entity power models**

Operating mode (a sample only)	$P_{avg(min)}^1$ mW	$P_{avg(max)}^1$ mW	Cycles <sup>2</sup>
ld [rs1], rd	3.310	3.406	1
st rd, [rs1]	3.437	3.568	2
clr [rs1]	3.286	3.286	1
nop	3.289	3.289	1
add rs1, rs2, rd	3.294	3.915	1
and rs1, rs2, rd	3.270	3.510	1

1) Power data for 10MHz, 1.8V. 2) For cache hit and no interlock

## 5.2. Memory entities

The major components of memory power dissipation are as follows for a CMOS SRAM cell [12], with power dissipated in 1) bit-lines, caused in pre-charging and during access, 2) word-lines, caused due to assertion of word select line by the line drivers, 3) output lines, caused due to driving signals on the interconnect external to the cache, and 4) input lines, due to transitions on input lines and input latches. In addition, the type of access will influence the power dissipation where a non-sequential memory access consumes more energy than a sequential access. Depending on accuracy requirements it might be sufficient to base memory power dissipation on an energy-per-access function. In this work we use the data supplied by the silicon provider for the memory sizes indicated in Table 1. Worst case power consumption data are as follows for 1.8V: *Register file* (Dual-port SRAM) = 58.0  $\mu$ W/MHz, *Cache module* 4kB (SRAM) = 210  $\mu$ W/MHz, *Cache tag memory module* (SRAM) = 143  $\mu$ W/MHz. For main memory we do not have accurate memory consumption details available, hence we only analyse the event statistics as an indication of memory power usage (see Section 8).

## 5.3. Additional entities (IP) on AMBA bus

A new entity connected to the AMBA bus on the platform is evaluated separately for its power characteristics. This process is described in detail in Section 7 for an AES hardware IP module.

## 6. Events

The instruction-set simulator (ISS), TSIM [7], of the LEON processor core features trace capability whereby details are provided of the executed instructions for an arbitrary software execution. The following information is provided for each executed instruction: cycle count, program counter value, instruction code, and mnemonic with operands. By analysing the trace data generated by the ISS it is possible to define *events* that cover the operation of entities of the system. The benefit of using the instruction-set simulator is that it automatically provides a cycle-accurate trace of a system execution – a method used also in [3].

For the processor entity we use the mnemonics as events since they describe the processor hardware functionality. For the memory entities the access statistics are used as events, e.g. size of data accessed from memory. The events for additional memory-mapped IP modules are collected based on the data transferred to each module over the AMBA bus, through analysis of the write instructions to pre-specified addresses. Event statistics for the following items are extracted or generated based on the TSIM trace information: *Execution time* (clock cycles, real time), *CPU clock frequency*, *CPU instruction utilisation* (occurrences, exec. cycles) of individual instructions and instruction types (arithmetic, logic, shift, load, store, branch, control), *Clock cycles per instruction* (CPI), *Data/Instruction cache utilisation*, *Main memory reads/writes*, and *Data transfers to IP modules*.

Having defined the system events it is now possible to link the events with power models for the system entities in order to perform a quick system-level power and energy evaluation for an arbitrary execution of the system.

## 7. Implementation of AES algorithm

During hardware/software (HW/SW) co-design, software functionality can be moved into hardware. This section describes how the platform is extended with additional hardware IP, using the Advanced Encryption Standard (AES) algorithm as the example application. These implementations are then evaluated in Section 8.

The AES algorithm [13] is used in security constrained applications such as Smart Card, and is specified to use keys with a length of 128, 192, or 256 bits to encrypt data blocks of the same lengths. The AES algorithm can be implemented efficiently in software on a wide range of processors as well as in dedicated hardware.

A standard **SW implementation** [14] of the AES algorithm – capable of encryption and decryption for all sizes of keys and data – was compiled for LEON using the LECCS compiler suite [7]. The performance of the implementation is dependent on memory hierarchy and the level of optimisation applied to the source code (e.g. through manual coding in assembly for critical parts of the code). The main benefits of a software implementation are *reconfigurability* and *portability*; the functionality can easily be modified in response to changing application demands, and software can be (re)compiled for different target platforms. These benefits may well be sacrificed for substantial performance improvements only achieved by a dedicated hardware accelerator. Moving to hardware, however, can incur other costs such as more expensive design activity and increased chip area.

A **HW/SW co-implementation** was investigated in this work, where we extended the LEON platform with a dedicated AES hardware module [15]. The principle of connecting the new IP module to the AMBA bus system is

illustrated in Figure 2. Each IP has its unique address and is reachable through the AMBA interface, which can interface to more than one IP.

Before a system-level evaluation of energy and power can take place – and a comparison be made between the SW and HW/SW implementations – the new entity on the platform is analysed for its attributes and parameters, and power models are obtained accordingly. This process is described next.

Firstly, the functionality of the AES module is studied; there are 6 operating modes each for *encryption* and *decryption* due to the possible combinations of key sizes, resulting in a total of 13 operating modes when including the idle mode (no operation). The AES module is controlled from software, through function calls in C, and is reached through three unique addresses; 1) write to input select register, 2) write to input data register, 3) read from output. The parameters for the AES entity are the key and data values to be encrypted or decrypted. The size of input data is 16 bytes for plaintext data, and 16, 24 or 32 bytes for key data. Table 3a shows the attributes obtained through synthesis in the same way as was done for the processor entity.

Secondly, the power models of the AES entity is obtained by analysing each operating mode to find power consumption and cycle count for varying parameters. The cycle count varies depending on operating mode but is not dependent on input data, whereas the power consumption is dependent on both operating mode *and* input data. The relation between input data and power consumption was studied for a subset of the total possible input combinations  $N$ , i.e.  $N = (n_{\text{Data length}} + n_{\text{Key length}})^2$ . Variations were applied as different number of logic one's in the input registers. The results for 51 test cases of the ENC16WEK operating mode are shown in Figure 3. By performing this analysis a LUT is defined for the AES entity with approximate min and max bounds; results are shown in Table 3b.

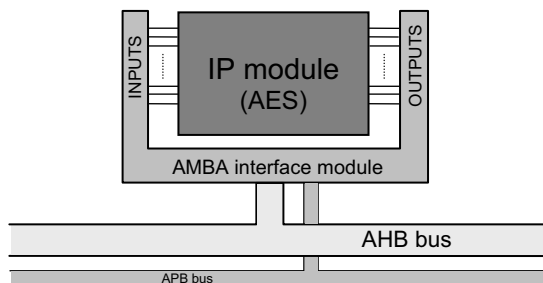


Figure 2: IP module connected to AHB/APB bus through AMBA interface

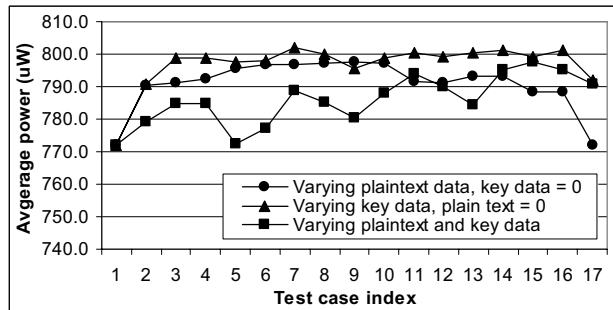


Figure 3: Dependency between power consumption and input data in AES module

Table 3 a-b: AES entity

a) Attributes<sup>1</sup>

Entity	Complexity	Interface	Op. modes	Tech.
AES module	36.5k transistors	14 inputs, 8 outputs	Idle, Encrypt. x 6, Decrypt. x 6	UMC 0.18um

b) Power models (for 5 of 12 operating modes)

#	Op. mode <sup>2</sup>	P <sub>ave</sub> (min) <sup>3</sup> mW	P <sub>ave</sub> (max) <sup>3</sup> mW	Cycles
1	ENC16WEK	0.772	0.802	330
2	ENC16WDK	0.755	0.771	406
3	DEC16WDK	0.782	0.806	350
4	DEC16WEK	0.795	0.803	426
13	Idle	0.408	0.408	n/a

1) This table extends the data in Table 1 for the new AES entity, 2) ENC16WEK = Encryption with 16-bytes Encryption Key, DEC16WDK = Decryption with 16-byte Decryption Key. 3) Data for 10MHz, 1.8V.

## 8. System-level evaluations using PEX

PEX (*Power EXtractor*) is the power and energy analysis tool that we developed for the LEON platform. The tool is in the form of a Perl script which takes the *TSIM trace data* and *entity power models* as input; PEX then generates event statistics which it combines with power models to generate a power and energy profile for an arbitrary execution of the system. The average power consumption for the processor entity is described by (1) whereby event statistics for instruction utilisation is combined with power models;  $M$  is the number of instruction types in the trace,  $P_{instr(i)}$  the basic cost for type  $i$  with min/max bounds,  $N_{instr(i)}$  the number of execution cycles for instruction type  $i$ ,  $N_{tot}$  the total number of execution cycles in the trace, and  $k$  the instruction overhead effects for min and max conditions. The power models are supplied as look-up-tables (LUTs), i.e. the data from Table 2. Power models can also be specified for different clock frequencies.

$$P_{proc.-min/max} = \left( \frac{\sum_{i=0}^M P_{instr(i)-min/max} N_{instr(i)}}{N_{tot}} \right) k_{min/max} \quad (1)$$

For the register file and caches we can only assign a maximum power value at this stage, which is taken from data sheets as described in Section 5.2; average power consumption is then calculated for the appropriate clock frequency. For main memory we create an *Access Index* based on the number and type of memory accesses; an 8-bit access is given a weight of 8 and a 16-bit access a weight of 16 and so on. The index is not directly related to units of Watts but still serves as an indicative measure of power usage in the main memory.

As for IP modules on the AMBA bus, PEX is supplied with the IP's power model in the same way as for the processor entity, e.g. data in Table 3b for the AES entity. PEX analysis the data transfers to the input registers of the AES module to determine which operating mode was active at what time during the system execution.

### 8.1. Results analysis

This section describes the evaluation of the two implementation scenarios of the AES algorithm, i.e. 1) a pure SW implementation versus 2) a HW/SW co-implementation.

The results generated by PEX for the two implementation scenarios are shown in Table 4a-e, Table 5, and Figure 4a-b. The two implementations execute the same operation, which is *encryption* with 16-byte data size and 16-byte key size. The software implementation consumes 32251 clock cycles, which is over six times more than for the HW/SW co-implementation which requires 5137 cycles to complete the same operation.

Detailed events statistics are provided for the processor entity (Table 4b). Note in particular the relative increase of *store instructions* which occupy 16.3% of the overall cycle count for the SW implementation and 34.1% for the HW/SW implementation. This is because functionality has been moved from software to hardware requiring a relatively higher rate of communication over the AMBA bus, and a relatively lower amount of computation (arithmetic, logic, shift operations) since all computation is done by the AES module in case of the HW/SW co-implementation. The number of different *types* of instructions is reduced from 53 to 34, which could be one of the reasons for a higher hit rate for instruction cache (95.7% compared to 89.3%), see Table 4c. Another effect of moving functionality into hardware is a reduction in total memory accesses, both reads and writes (Table 4d), but as can be seen in Table 4e this now requires transfers to the AES IP module over the AMBA bus.

PEX has computed the average power consumption for the different entities of the system (see Table 5). The AES module is not part of the SW implementation thus not contributing to the overall power dissipation for that scenario. The power consumption for the processor entity is not significantly different for the two implementations.

Since we only have data for max conditions of the memory modules the average power consumption is the same for the two different implementations. Nevertheless, the information is still important for the system-level perspective where the memory modules (caches + reg. file) together consumes (on a maximum bound) well over half of the overall power consumption of the LEON core (i.e. all entities excl. main memory).

The energy consumption reported by PEX is shown in Figure 4a in comparison with the overall average power consumption. The two compared implementations executes at the same clock speed (10MHz). A six-fold reduction in energy consumption is achieved for the HW/SW implementation compared to the SW implementation. The energy consumption is the product of average power consumption and execution time; and in this particular example the average power consumption does not change dramatically between the two implementation scenarios resulting in *execution time* being the primary factor in determining the energy consumption. Consequently, since the execution time is reduced from 3.23 ms to 0.51 ms – a ratio of 6.33 – the energy consumption is reduced approximately by the same ratio.

**Table 4 a-e: Event statistics<sup>1</sup>**

#### a) Execution statistics

Clock cycles	32251 [5137]
CPU clock frequency	10.0 MHz [10.0 MHz]
Circuit elapsed time	3.23ms [0.51ms]

#### b) Processor entity

Class	Types	Occur.	Cycles <sup>2</sup>	Cycles (%)
Arithm.	5 [4]	3107 [442]	6568 [935]	20.4 [18.2]
Logic	8 [6]	3093 [240]	6447 [398]	20.0 [7.75]
Shift	3 [2]	744 [56]	1660 [79]	5.15 [1.54]
Load	4 [1]	3033 [556]	5435 [749]	16.9 [14.6]
Store	6 [3]	1399 [479]	5249 [1749]	16.3 [34.1]
Branch	17 [9]	1851 [286]	3588 [368]	11.1 [7.16]
Control	8 [7]	1257 [34]	2493 [123]	7.73 [2.39]
Others	2 [2]	542 [651]	811 [736]	2.51 [14.3]
<b>TOTAL</b>	<b>53 [34]</b>	<b>15026 [2744]</b>	<b>32251 [5137]</b>	<b>100 [100]</b>
Clock cycles / instruction (CPI)		2.15 [1.87]		
Stall cycles		16193 (50.2%) [1926 (37.5%)]		

#### c) I/D caches

Type	Hit rate (%)
I-cache (4kB)	89.3 [95.7]
D-cache (4kB)	87.4 [67.5]

#### d) Main memory

Reads (8/16/32/64-bits access)	331/112/2590/0 [0/0/556/0]
Writes (8/16/32/64-bits access)	116/22/990/21 [0/0/441/13]
Access index <sup>3</sup>	121624 [32736]

#### e) IP module on AMBA bus (in this work: AES)

Transfers to address ID 1 (input select)	0 [65]
Transfers to address ID 2 (input data)	0 [132]

1) Data for two different executions: SW execution versus [HW/SW execution]. 2) Total # of cycles that instruction occupies execution stage of CPU pipeline. 3) Weighted index (weight by bit-length) of total number of Reads and Writes.

Since the execution time is shorter for the HW/SW implementation, the clock frequency can be reduced 6.33 times and computation can still take place in the same amount of time as required for the SW implementation. Hence, a considerable reduction in power consumption can be achieved as well as in energy.

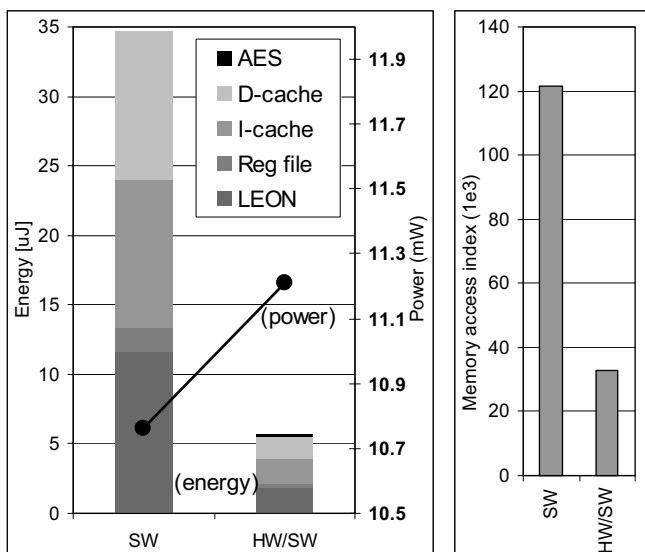
However, if the clock frequency were to be *maintained* at 10MHz the average power consumption would be increased by 4.2% as shown in Figure 4a). Our methodology and analysis tool PEX feasibly supports quick assessment of these types of tradeoffs (energy, power and speed) for various platform configurations.

Although we cannot quantify the energy consumption for main memory, the significant reduction in access index (Figure 4b) indicates a *reduction* in power usage of this entity. The transfers over the AMBA bus to the AES module, through the AMBA interface, contribute to the overall power dissipation. Currently we only consider the event statistics in Table 4e as an indication of power usage for data transfers. For this particular example the extra power consumption due to the AMBA interface and bus connections is small in a system-level perspective but this can change when connecting several IP modules.

**Table 5: Avg. power for entities**

Entity	$P_{avg}$ [mW] <i>SW implementation</i>	$P_{avg}$ [mW] <i>HW/SW impl.</i>
Processor entity	3.62 – 3.88	3.64 – 3.83
Register file	0.54	0.54
I-cache + tag mem.	3.30	3.30
D-cache + tag mem.	3.30	3.30
IP module (AES)	0	0.43

1) Power data for 10MHz, 1.8V.



**Figure 4 a-b: SW versus HW/SW implementation, a) Energy/Power consumption, b) Memory access index**

Regarding speed-up in simulations; a full gate-level simulation with power profiling takes approx. 2 hours for the current configuration of the LEON platform, whereas PEX completes the power profiling in less than 10 seconds.

## 8.2. Verification

To verify the power estimations for the processor entity (excluding memories) we compared results from full gate-level simulations for software running on the LEON testbench, with the estimations obtained through PEX. Table 6a shows three test programs and the simulated versus estimated power consumption. The error is within a 4.2-9.3% worst case range. More work is required to refine the estimation procedure in order to increase the accuracy.

To verify the accuracy of estimating the additional power consumption in the AES entity for HW/SW co-implementation, we implemented and simulated a complete gate-level version of the processor entity and the AES entity connected through the AMBA interface. Table 6b shows the results for simulations versus estimations. The error indicates the additional power dissipation in the overhead caused by the AMBA interface module and bus connections; area-wise this overhead constitute only 2.9% of the total transistor count for the full gate-level version. We are in the process of including this in the power model for additional IP. Figure 5 shows comparative results for power consumption of the AES module when using two different power estimation tools, i.e. Synopsys Power Compiler (gate-level) versus Synopsys Nanosim (transistor-level). The results show a correlation in the range 13.5% – 22.8% for 12 test cases. The simulations in Nanosim are more time consuming than Power Compiler and Nanosim also requires transistor level details, i.e. SPICE format cell library and transistor models (we use Spectre BSIM3 transistor models).

**Table 6 a-b: Simulation versus estimation<sup>1</sup>**

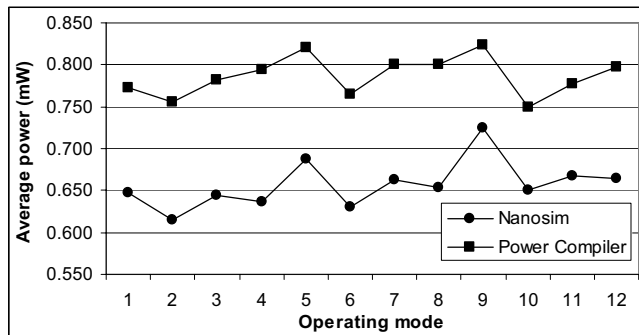
### a) Average power for Processor entity

Software program functionality	Simulated $P_{avg}$ (mW)	Estimated $P_{avg}$ (mW)	Error (%)
Ctrl.-SW for AES module	3.80	3.64-3.83	< 4.2%
AES full encryption	3.94	3.62-3.88	< 8.2%
Data sorting in C	3.99	3.62-3.86	< 9.3%

### b) Average power for Processor entity + AES entity

#	Op. mode	Simulated (mW)	Estimated (mW)	Error (%)
1	ENC16WEK	4.520	4.282	5.26%
2	ENC16WDK	4.496	4.258	5.30%
3	DEC16WDK	4.516	4.296	4.88%
4	ENC24WEK	4.501	4.401	2.22%
5	Idle/loading	4.743	4.524	4.62%

1) Power data for 10MHz, 1.8V.



**Figure 5: Comparison between Power Compiler and Nanosim power estimations for AES module**

## 9. Conclusion

This paper demonstrated a methodology capable of evaluating crucial tradeoffs (energy, speed and power) at the system-level for a synthesisable AMBA-based SOC platform. The formulation of the methodology is generic thus not requiring the exact set of tools used in this work. There are other commercial power estimation tools, e.g. PowerTheatre from Sequence Design that can be used in place of lower-level tools like Nanosim and Power Compiler. The accuracy requirements of power estimations will determine the choice of tool to use for building the power models.

Considerable speed-up was achieved in evaluating the energy-efficiency when integrating IP modules on the platform. The analysis technique incorporates the instruction-set simulator as the only means of collecting the system events, providing for a practical and fast evaluation of energy and power when combining events with power models.

Obtaining power models for hardware entities (IP modules) is a one time effort for each configuration of entity attributes. Hence, repositories can be built up containing pre-characterised IP modules that conveniently can be used for evaluating new system configurations.

The approach that we presented has industrial value since it has potential to reduce the time (and cost) in finding an optimal implementation for a platform based SOC design.

## Acknowledgements

The authors would like to thank Renesas Technology Europe Ltd. (<http://www.eu.renesas.com>) and the Engineering and Physical Sciences Research Council (EPSRC) for their support under the Engineering Doctorate program (award number 99316566).

## References

- [1] J. Henkel, Yanbing Li, "Avalanche: an environment for design space exploration and optimization of low-power embedded systems", in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Volume 10, Issue 4, Aug. 2002, pp. 454 -468.
- [2] M. Lajolo, A. Raghunathan, S. Dey, L. Lavagno, "Cosimulation-based power estimation for system-on-chip design", in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Volume 10, Issue 3, June 2002, pp. 253 -266.
- [3] T. Simunic, L. Benini, G. De Micheli, "Energy-Efficient Design of Battery-Powered Embedded Systems", in *IEEE Transactions on Very Large Scale Integration (VLSI)*, Volume: 9 Issue: 1, Feb 2001, pp. 15 -28.
- [4] T. D. Givargis, F. Vahid, J. Henkel, "A hybrid approach for core-based system-level power modelling", in *Proc. of the Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2000, pp. 141 -145.
- [5] P. Kalla, J. Henkel, S. X. Hu, "SEA: Fast power estimation for micro-architectures", in *Proc. of Asia and South Pacific Design Automation Conference (ASP-DAC) 2003*, pp. 600-605.
- [6] AMBA 2.0 specification, <<http://www.arm.com/armtech/AMBA>>
- [7] Gaisler Research, <<http://www.gaisler.com>>
- [8] LEON2 version 1.0.17-xst, Hardware Manual
- [9] V. Tiwari, S. Malik, A. Wolfe, "Power analysis of embedded software: a first step towards software power minimization", in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Volume 2 Issue 4, Dec. 1994, pp. 437-445.
- [10] Y. Fei, S. Ravi, A. Raghunathan, N. K. Jha, "Energy Estimation for Extensible Processors", in *Proc. of Design Automation and Test in Europe (DATE)*, 2003, pp. 682 - 687.
- [11] K. Hildingsson, T. Arslan, "System-level power evaluation of an embedded software data block processing algorithm", in *Proc. of the IEEE Intl. ASIC/SOC Conference*, 2002, pp. 451-455.
- [12] M. B. Kamble, K. Ghosse, "Analytical Energy Dissipation Models for Low Power Caches", in *Proc. of International Symposium on Low Power Electronics and Design*, 1997, pp. 143 -148.
- [13] <<http://csrc.nist.gov/CryptoToolkit/aes/rijndael/>>
- [14] <<http://www.esat.kuleuven.ac.be/~rijmen/rijndael/>>
- [15] G. Romano, "AES algorithm implementation for a SmartCard", MSc project dissertation, The Institute for System Level Integration, September 2001.