J Sign Process Syst (2010) 60:239–250
DOI 10.1007/s11265-008-0301-8

A High-level Microprocessor Power Modeling Technique Based on Event Signatures

Peter van Stralen · Andy D. Pimentel

Received: 9 April 2008 / Revised: 13 August 2008 / Accepted: 16 October 2008 / Published online: 11 November 2008

Abstract This paper presents a technique for high-level power estimation of microprocessors. The technique, which is based on abstract execution profiles called 'event signatures', operates at a higher level of abstraction than commonly-used instruction-set simulator (ISS) based power estimation methods and should thus be capable of achieving good evaluation performance. As a consequence, the technique can be very useful in the context of early system-level design space exploration. In this paper, we also compare our power estimation results to those from the instruction-level simulators Wattch and Sim-Panalyzer. In these experiments, we demonstrate that with a good underlying power model, the signature-based power modeling technique can yield accurate estimations (a mean error of 3.1% compared to Wattch in our experiments). At the same time, our signature-based power modeling technique is at least an order of magnitude faster than the simulations performed by Wattch or Sim-Panalyzer.

Keywords High-level power modeling · Profiling · Event signatures

P. van Stralen (✉) · A. D. Pimentel
Computer Systems Architecture group, Informatics Institute, University of Amsterdam, Kruislaan 403, 1098 SJ, Amsterdam, The Netherlands
e-mail: pstralen@science.uva.nl

A. D. Pimentel
e-mail: andy@science.uva.nl

1 Introduction

The increasing complexity of modern embedded systems, which are more and more based on heterogeneous multiprocessor-system-on-chip (MP-SoC) architectures, has led to the emergence of system-level design. A key ingredient of system-level design is the notion of high-level modeling and simulation in which the models allow for capturing the behavior of system components and their interactions at a high level of abstraction. As these high-level models minimize the modeling effort and are optimized for execution speed, they can be applied at the early design stages to perform, for example, architectural design space exploration (DSE). Such early DSE is of eminent importance as early design choices heavily influence the success or failure of the final product.

The Sesame modeling and simulation framework [7, 16] facilitates efficient system-level DSE of embedded multimedia systems, allowing rapid performance evaluation of different architecture designs, application to architecture mappings, and hardware/software partitionings. Key to this flexibility is the separation of application and architecture models, together with an explicit mapping step to map an application model onto an architecture model.

Until now, the Sesame modeling and simulation framework has purely focused on the performance analysis of multimedia MP-SoC architectures. Evidently, to make good design trade-offs, also power consumption needs to be taken into account during the process of DSE. Therefore, this paper presents the first step towards including system-level power models in Sesame. More specifically, we elaborate on the concept of *event signatures*, which was recently introduced in

[24], that allows for high-level power modeling of microprocessors (and their local memory hierarchy). This signature-based power modeling operates at a higher level of abstraction than commonly-used instruction-set simulator (ISS) based power models and should thus be capable of achieving good evaluation performance. This is important since ISS-based power estimation generally is not suited for early DSE as it is too slow for evaluating a large design space: the evaluation of a single design point via ISS-based simulation with a realistic benchmark program may take in the order of seconds to hundreds of seconds [12]. Moreover, unlike many other high-level power estimation techniques, our signature-based power modeling technique still incorporates an explicit micro-architecture model of the processor, and thus is able to perform micro-architectural DSE as well.

Using several experiments, we compare the results from our signature-based power modeling with those from Wattch [6] and Sim-Panalyzer [1], which are widely-used ISS-based power analysis tools. Here, we show that with a good underlying power model, the signature-based power modeling technique can yield accurate estimations, while being at least an order of magnitude faster than Wattch or Sim-Panalyzer. In order to perform system-level power modeling of an entire MP-SoC, the next step (not addressed in this paper) will be to extend the power modeling framework with models for the interconnect and possible dedicated components in the MP-SoC.

In the next section, we briefly describe the Sesame framework. Section 3 introduces the concept of event signatures and explains how they can be used for high-level power modeling of microprocessors. In Section 4, we describe the power models used for modeling different aspects of microprocessors. Section 5 presents a number of experiments in which we compare the results from our models against those from Wattch and Sim-Panalyzer. In Section 6, we describe related work, after which Section 7 concludes the paper.

2 The Sesame Environment

To facilitate flexible performance analysis of embedded (media) systems architectures, the Sesame modeling and simulation environment [7, 16] uses separate application and architecture models. An application model describes the functional behavior of an application while the architecture model defines architecture resources and captures their performance constraints. After explicitly mapping an application model onto an architecture model, they are co-simulated via trace-driven simulation. This allows for evaluation of the system performance of a particular application, mapping, and underlying architecture. Essential in this methodology is that an application model is independent from architectural specifics and assumptions on hardware/software partitioning. As a result, a single application model can be used to exercise different hardware/software partitionings and can be mapped onto a range of architecture models, possibly representing different architecture designs or modeling the same architecture design at various levels of abstraction. In Fig. 1, the layered infrastructure of Sesame is illustrated using an example in which a Motion-JPEG encoder application model is mapped onto a bus-based MP-SoC architecture model.

For application modeling, Sesame uses the Kahn Process Network (KPN) model of computation [10], which fits well to the multimedia application domain. In a KPN, parallel processes communicate with each other via unbounded FIFO channels, where reading from these channels is blocking and writing is non-blocking. The computational behavior of an application is captured by instrumenting the code of each Kahn process with annotations that describe the application's computational actions. The reading from and writing to Kahn channels represent the communication behavior of a process within the application model. By executing the KPN model, each Kahn process records its actions in order to generate its own trace of *application events*. These application events are an abstract representation of the workload that is imposed on the architecture, and drive the underlying architecture model. Application events typically are coarse grained, such as *Execute(DCT)*, which represents the execution of a Discrete Cosine Transform (DCT), or *Read(channel_id,pixel-block)*, which represents the reading of a pixel block from communication channel *channel_id*.

An architecture model simulates the performance consequences of the computation and communication events generated by an application model. To this end, each architecture model component is parameterized with an *event table* containing the latencies of the application events it can execute (illustrated for Processor 1 in Fig. 1, where the labels X, Y and Z refer to application event names). The event table entries could, for example, specify the latency of an *Execute(DCT)* event, or the latency of a memory access in the case of a memory component. The latency values are usually initialized using performance numbers from literature, and can be calibrated using measurements on available hardware or via lower-level simulations of architecture components [17]. This trace-driven simulation

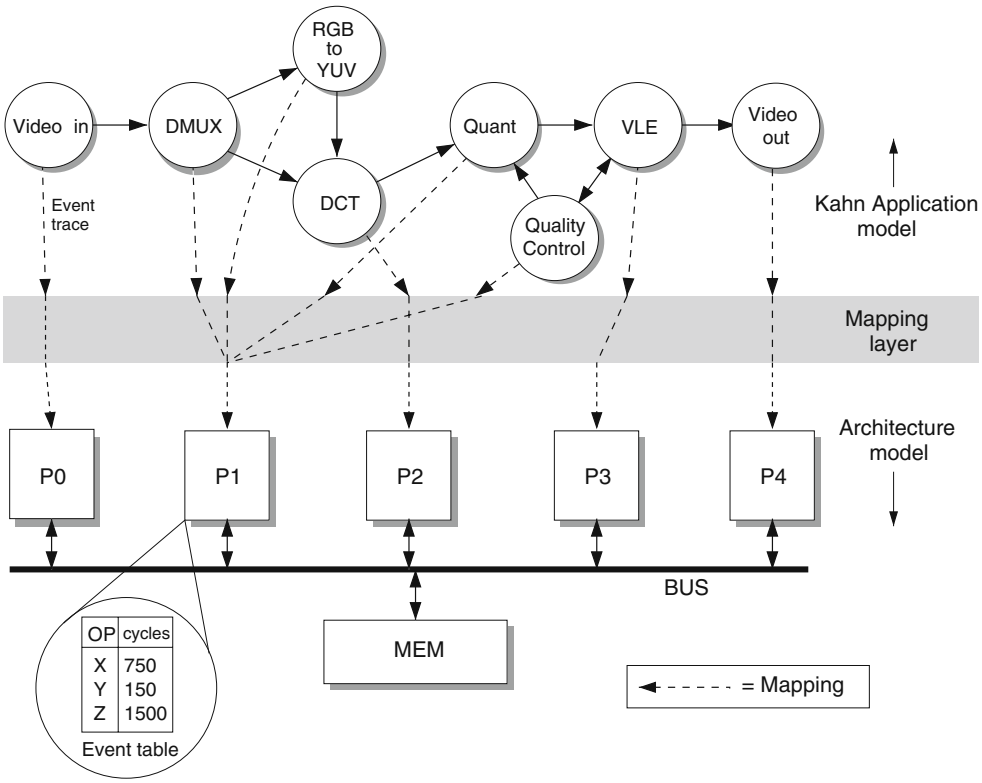


Figure 1 Modeling an Motion-JPEG application on a bus-based MP-SoC architecture in Sesame.

approach allows to quickly assess different HW/SW partitionings by experimenting with the latency parameters of processing components in the architecture model: a low computational latency refers to a HW implementation while a high latency mimics a SW implementation.

To bind application tasks to resources in the architecture model, Sesame provides an intermediate mapping layer. This layer controls the mapping of Kahn processes (i.e. their event traces) onto architecture model components by dispatching application events to the correct architecture model component. The mapping also includes the mapping of Kahn channels onto communication resources in the architecture model.

Extending the Sesame framework to also support power modeling could be done fairly easily by adding power consumption numbers to the event tables. So, this means that a component in the architecture model not only accounts for the timing consequences of an incoming application event, but also accounts for the power that is consumed by the execution of this application event (which is specified in the event tables now). The power numbers that need to be stored in the event tables can, of course, be retrieved from lower-level power simulators or from (prototype) implementations of components. However, simply adding fixed power numbers to the event tables would be a rigid solution in terms of DSE: these numbers would only be valid for the specific implementation used for measuring the power numbers. Therefore, we propose a high-level power estimation method based on so-called *event signatures* that allows for more flexible power estimation in the scope of system-level DSE.[1] As will be explained in the next sections, signature-based power estimation provides an abstraction of processor activity in comparison to traditional ISS-based power models, while still incorporating an explicit micro-architecture model and thus being able to perform micro-architectural DSE.

3 Event Signatures

An event signature is an abstract execution profile of an application event that describes the computational complexity of the application event (in the case of computational events) or provides information about the data that is communicated (in the case of communication events). Hence, it can be considered as

[1]With respect to the rigidness of fixed numbers in the event tables, the same reasoning holds for the operation latencies in the tables. For this reason, we are also working on performance prediction using event signatures [9], but this is out of the scope for this paper.

meta-data about an application event. In this paper, we purely focus on signatures for computational application events (i.e., *Execute()* events). The signatures for these events describe computational complexity in a (micro-)architecture independent fashion using an Abstract Instruction Set (AIS). Currently, our AIS is based on a load-store architecture and consists of only a small set of abstract instructions, or rather *instruction classes*, such as 'Simple Integer Arithmetic', 'Simple Integer Arithmetic Immediate', 'Integer Multiply', 'Branch', 'Load', and 'Store'. The high level of abstraction of the AIS should allow for capturing the computational behavior for a wide range of processors with different instruction-set architectures. To construct the signatures, the real machine instructions that embody an application event (derived from an ISS as will be explained below) are first mapped onto the various AIS instruction classes, after which a compact execution profile is made. This means that the resulting signature is a vector containing the instruction counts of the different AIS instruction classes. Here, each index in this vector specifies the number of executed instructions of a certain AIS class in the application event. We note that the generation of signatures for each application event is a one-time effort, unless e.g. an algorithmic change is made to an application event's implementation.

In Fig. 2, signature-based power modeling is illustrated. The Kahn application process for which a power estimation needs to be performed, is simulated using Sim-Cache, which is part of the SimpleScalar simulator suite [2]. Using this relatively fast simulator, the event signatures are constructed—by mapping the executed machine instructions onto the AIS as explained above—for every computational application event that can be generated by the Kahn process in question. The event signatures act as input to our parameterized microprocessor power model, which will be described in detail in the next section. For each signature, Sim-Cache (or any other ISS that could be used to construct signatures) may also provide the power model with some additional micro-architectural information, such as cache missrates, branch misprediction rates, etc. In our case, only instruction and data cache missrates are used. The microprocessor power model also uses a micro-architecture description file in which the mapping of AIS instructions to usage counts of microprocessor components is described. An example fragment of this mapping description is shown in Fig. 3. It specifies that for every AIS instruction (indicated by the 'ALL' tag), the instruction cache (il1) is read, the register update unit (RUU) is read and written, and branch prediction is performed. Furthermore, it specifies that for the AIS instruction 'LOAD', the ALU is used (to calculate the address), the level-1 data cache (dl1) is accessed, and that the integer register file (irf) is read and written. With respect to the latter, it takes register and immediate addressing modes into account by assuming 1.5 read operations to the irf on average. In addition, the micro-architecture description file also

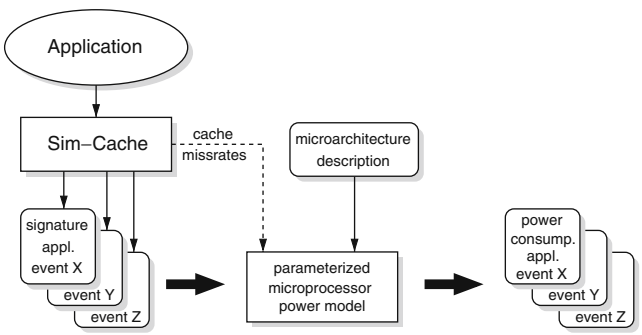


Figure 2 Signature-based power modeling.

```
<node name="ALL" class="AIS">
  <port name="bpred" dir="in">
    <property name="execute" value="1" />
  </port>
  <port name="il1" dir="in">
    <property name="read" value="1" />
  </port>
  <port name="RUU" dir="in">
    <property name="read" value="1" />
    <property name="write" value="1" />
  </port>
</node>
<node name="LOAD" class="AIS">
  <port name="dl1" dir="in">
    <property name="read" value="1" />
  </port>
  <port name="irf" dir="in">
    <property name="read" value="1.5" />
    <property name="write" value="1" />
  </port>
  <port name="alu" dir="in">
    <property name="execute" value="1" />
  </port>
</node>
```

Figure 3 Mapping AIS instructions to usage counts of the microprocessor components.

contains the parameters for our power model, such as e.g. the dimensions and organization of memory structures (caches, register file, etc.) in the microprocessor, clock frequency, and so on. Clearly, this micro-architecture description allows for easily extending the AIS and facilitates the modeling of different micro-architecture implementations. The above ingredients (the event signatures, additional micro-architectural information per signature such as cache statistics, and the micro-architecture description of the processor) allow the power model to produce power consumption estimates for each computational application event. These estimates can be performed on-the-fly during a Sesame system simulation: application events generated by an application model, together with their signatures and additional micro-architectural information, are used at the architecture model level by the power model to dynamically estimate the power consumption. Alternatively, the power consumption of the different types of application events can also be estimated off-line (i.e., statically) after which these estimates are stored in Sesame's event tables at the architecture model level (as discussed in Section 2).

We note that the generation of event signatures can also be performed either statically or dynamically. In static signature generation, Sim-Cache measures the average instruction execution behavior of code fragments that represent application events and constructs the signatures based on these averages. So, in this case, the signature generation takes place entirely off-line. In dynamic signature generation, the signatures are constructed on-the-fly for every application event. This means that the signatures of the same type of application events may change over time due to e.g. data dependent execution behavior inside the code of these events. Another consequence of dynamic signature generation is that Sim-Cache must be co-simulated together with Sesame. The above reasoning also holds for the additional micro-architecture information, like cache missrates, that can be provided by Sim-Cache to the power model. This can also be done statically (i.e., average based) or dynamically (i.e., exact based).

4 Microprocessor Power Model

The microprocessor model that underlies our power model is depicted in Fig. 4. It is a dynamic pipelined machine, consisting of one arithmetic logical unit, one floating point unit, a multiplier and two levels of caches. For the sake of simplicity, it does not use a hardware branch prediction module. This means that we currently assume taken, not-taken or perfect branch prediction. The level 1 caches are virtually addressed and each have a TLB for address translation. The level 2 cache is unified. All requests for this cache are issued to a load/store queue. The same is true for level-2 cache requests to the main memory.

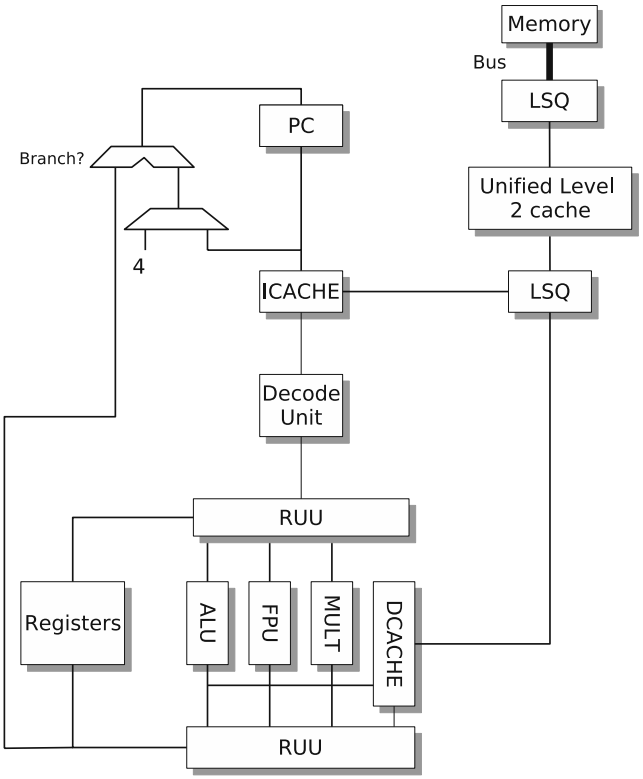


Figure 4 The underlying microprocessor model for our power models.

The dynamic pipeline is implemented using a RUU. This unit controls the dispatching of instructions to the functional units and, after execution, the RUU takes care of the (in-order) instruction commitment. This means that in Fig. 4 the two displayed RUUs are one and the same. We believe that the above microprocessor model that underlies our power model can reflect the behavior of popular embedded processors (like ARM10/11 or MIPS4000 based processors) in a fairly realistic fashion, albeit at a high level of abstraction.

The various components that constitute our microprocessor power model—which are of course based on our underlying microprocessor model as depicted in Fig. 4—are shown in Fig. 5. The power consumption of an application event is calculated by accumulating the power consumption in each of these components. More specifically, the first step to calculate an application event's power consumption is to map its signature (using the micro-architecture description file, as explained in the previous section) to usage counts

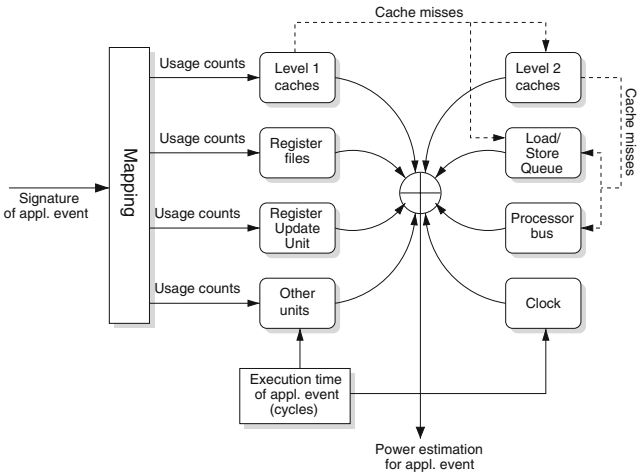


Figure 5 The different components in our microprocessor power model.

of the various processor components. So, here it is determined how often e.g. the ALU (in 'other units' in Fig. 5), the register file and the level-1 instruction and data caches are accessed during the execution of an application event. For the memory components (level 1 and 2 caches, register file, etc.), we use Cacti 4.2 [22] to determine the power consumption of read and write accesses to these structures. These power estimates include leakage power. Moreover, we use the cache missrate information provided by Sim-Cache to derive the access counts for the level-2 cache, load/store queue and bus components. Here, we note that although we recently also incorporated Cacti 5.0 into our tool flow, which allows for modeling of DRAMs, this paper does not consider the modeling of the main memory.

The non-memory components in our power model ('other units', 'bus' and 'clock' components in Fig. 5) are activity based. That is, they estimate power using the common power equation for CMOS technology:

$$P = \alpha C V^2 f \tag{1}$$

where C specifies the capacitance, V the voltage, and f the frequency. The activity, which is defined as the percentage of transistors which make a switch on each clock cycle, is represented by the parameter α. For the bus component, which represents the processor's front-side bus, we use a simple model which abstracts the bus to a set of wires, without any logic, with input and output pins. Currently, we use an I/O pin capacitance of 5 pF per pin, a wire capacitance of 2.15 pF per inch, and a wire length of 3 inches. These numbers were taken from [11]. Further, we assume an activity α of 0.5 (half of the wires perform a state switch). For the models of the ALU and multiplier units (in the box 'other units' in Fig. 5), we use the capacitance numbers from Wattch [6]. The activity α for these units is calculated by dividing (an estimation of) the number of cycles in which a unit is active, which is derived from the usage counts of the unit (see Fig. 5), by the total number of cycles.

For the power model of the clock component, we base ourselves on the models used in [19, 21]. The model recognizes three sub-components: the clock distribution wiring, the clock buffering and the clocked node capacitance. We assume a H-tree based clock network using a distributed driver scheme (i.e. applying clock buffers).

To determine the wiring capacitance of the clock network, the wire length of the network has to be calculated. This is done in a number of steps. First, the length of a single straight wire at a specific level in the clock tree has to be calculated. The length of the first straight wire at the root level is equal to half of the width of the chip (where $\sqrt{A_{die}}$ is the width of the chip). Then, because of the H-tree form of the clock network, going to a next level in the clock tree halves the length of a straight wire. This means that the wire length at tree level i equals to:

$$Wirelength_i = \frac{1}{2^{\lfloor i/2 \rfloor + 1}} \times \sqrt{A_{die}} \tag{2}$$

With $Wires_i = 2^{i-1}$, which is the number of straight wires at tree level i, the total wire length can be calculated. However, this total length cannot directly be multiplied by the capacitance of the wire, as the capacitance of the wire is not uniform in the network. Starting form the end points in the network, the capacitance of the wire doubles at every T-junction.

$$FactorCap_i = 2^{N_{tree}-i} \tag{3}$$

This gives for the total capacitance of the distribution wiring the following equation:

$$C_{wiring} = C_{wire} \times \sqrt{A_{die}} \times 2^{N_{tree}-1} \times \sum_{i=1}^{N_{tree}} \frac{1}{2^{\lfloor i/2 \rfloor + 1}} \tag{4}$$

where we retrieve the wire capacitance (C_{wire}) and chip area A_{die} (determined by accumulating all cache and register-file areas) from Cacti 4.2, and calculate the depth of the clock tree (N_{tree}) using:

$$N_{tree} = \sqrt{A_{die} \frac{R_{wire} \times C_{wire}}{Skew_{clock}}} + 1 \tag{5}$$

where resistance R_{wire} is also retrieved from Cacti, and $Skew_{clock}$ is the maximum clock skew that is allowed.

The capacitance consumed by the buffers is modeled to be a fraction of the capacitance consumed by

the wiring network. This fraction is dependent on the number of buffers, which is calculated by first taking the ratio of the capacitance of the wiring network and the capacitance of a single buffer. Over this the fourth root is taken, where the value four is actually a parameter, the optimal stage ratio, but this value is fixed within our model.

$$buffers = \sqrt[4]{\frac{C_{wiring}}{C_{single\_buffer}}} \tag{6}$$

$$C_{buffers} = C_{wiring} \times \frac{1}{1 - (\frac{1}{buffers})} \tag{7}$$

For the clocked node capacitance, only memory components are considered. Here, we use the number of read and write ports and the blocksize to calculate the capacitance:

$$C_{clocked} = ports \times blocksize \times C_{trans} \tag{8}$$

The capacity for switching a port is acquired from Cacti, and is equal to the capacitance of a transistor. The clocked node capacitance of each memory structure is summed to the total clocked node capacitance.

For several components in our power model, the execution time of an application event is needed in order to calculate the activity parameter α (i.e., the time that a unit is active during the execution of an application event divided by the total execution time of the application event). These event execution times may be derived from the event tables in Sesame's architecture model (e.g. in the case signatures are generated statically), or they may be generated dynamically using e.g. our trace calibration co-simulation technique [23].

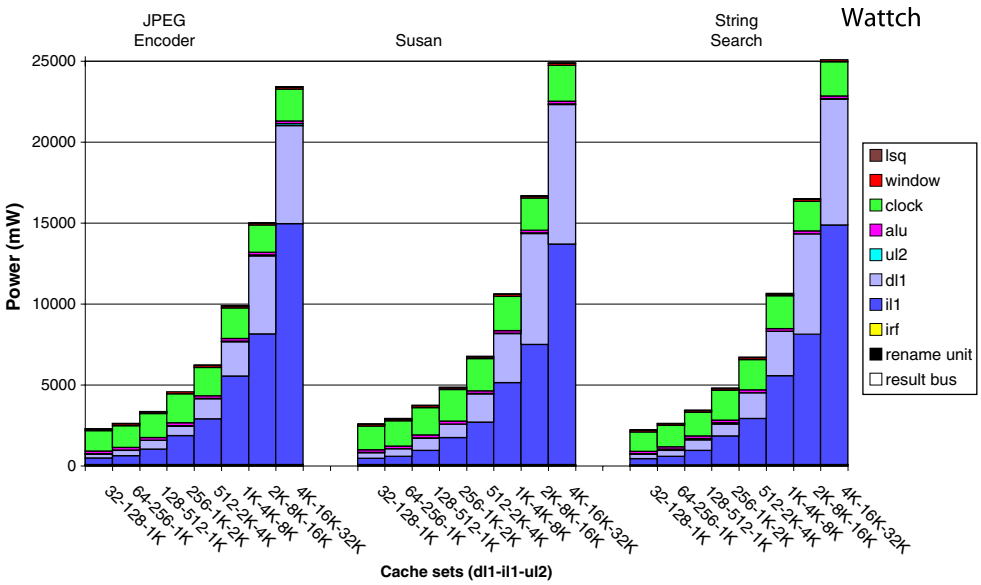
5 Experiments

To evaluate our signature-based power estimation, we use three benchmark applications from the MiBench benchmark suite [8]: cjpeg (jpeg compression), susan (edge detection), and string search. We compare our results to those from Wattch [6] and Sim-Panalyzer [1], which are widely-used ISS-based power analysis tools. To compare against Wattch, we use a (in-order issue) PowerPC microprocessor model, with a voltage of 2.0 V and a frequency of 600 MHz. For comparison to Sim-Panalyzer, we use an (in-order issue) ARM microprocessor model, with a voltage of 1.8 V and a frequency of 233 MHz. For both models, we assume a 180 nm technology.

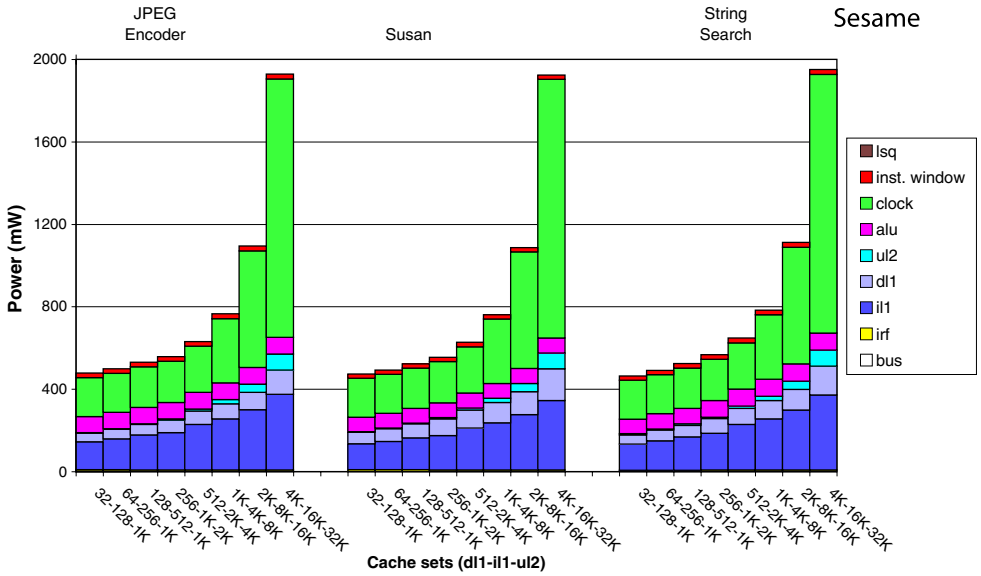
In the first experiment, we have varied the sizes of the level-1 instruction (*il1*) and data (*dl1*) caches as well as of the unified level-2 cache (*ul2*). This is done by increasing the number of sets in the caches. The level-1 caches are increased in size from 4 Kb to 512 Kb, and the level-2 cache from 32 Kb to 4 Mb. In Fig. 6a, the power estimation results from Wattch are shown, while Fig. 6b shows the results from our own power model. Clearly, the absolute power predictions differ significantly between Wattch and Sesame. As will be demonstrated later on, this is mostly due to the differences in the underlying power models. In Wattch, the power consumption is dominated by the level-1 instruction and data caches and, to a lesser extent, the clock network. In Sesame, the power is mostly dominated by the clock network, followed by the level-1 caches. The large discrepancy in absolute power estimations is mainly caused by two differences in the power models of Sesame and Wattch: First, Wattch has a more extensive model of the clocked node capacitance. For example, not only the clocked node capacitance of memory components are modeled, but also the datapath and other components are included. Second, whereas Sesame (in line with Cacti) only models the power consumption related to the critical path inside memory structures, Wattch directly relates the power consumption of a memory structure to the size of the memory. The latter can be clearly seen in Fig. 6a where the power consumption of the level-1 caches rapidly increases when increasing the number of cache sets.

Another observation that can be made is that the power consumption of the clock network in Wattch does not appear to increase for larger cache structures, while there is a significant increase of power consumed by the clock in Sesame. This is due to the fact that Sesame dynamically calculates the die area (using Cacti) for the power modeling of the clock network, while Wattch uses a fixed die area which evidently does not scale with the cache size.

In Fig. 7a, the power estimation results from Sim-Panalyzer for the same experiment are shown, while Fig. 7b again shows the results from our own power model but now configured to model the ARM processor. Like with Wattch, the absolute power predictions differ significantly between Sim-Panalyzer and Sesame. But maybe more surprising is the large increase of power consumption of the clock when scaling the cache sizes for Sim-Panalyzer. With the largest cache sizes in our experiment, the fraction of clock power consumption to the overall power consumption is equal to 99%. This significant increase of clock power in Sim-Panalyzer is mostly due to the fact that its model for clocked node capacitance is directly dependent on the number of sets in cache structures. In Sesame on the other hand, correspondingly to Eq. 8, the clocked



(a)



(b)

Figure 6 Power consumption estimation for Wattch (**a**) and Sesame (**b**) when varying the cache sizes.

node capacitance is only dependent on the block size of memory structures and the number of ports.

Also, the increase of power consumption by the caches is much larger in Sim-Panalyzer than in Sesame. As is the case with Wattch, this is again due to the fact that Sim-Panalyzer directly relates the power consumption of a memory structure to its size, rather than relating the power consumption to the critical path inside memory structures like Sesame does. We further note that the discrepancies in Sesame's results for e.g. cjpeg for cache configurations between 64-256-1K and 512-2K-4K are caused by the reduced activity at the lower levels of the memory hierarchy due to the increasing cache sizes. In these cases, the increased hit-rate seems to amortize the higher dynamic and static power consumption of the larger caches.

From the above experiments, we learn that although Sesame can predict the total power consumption trend reasonably well (e.g. when compared to Wattch), its power model still needs to be improved considerably to better reflect the correct absolute power consumptions of the various architectural components. Moreover, the current differences in the underlying power models of Sesame, Wattch and Sim-Panalyzer make it impossible

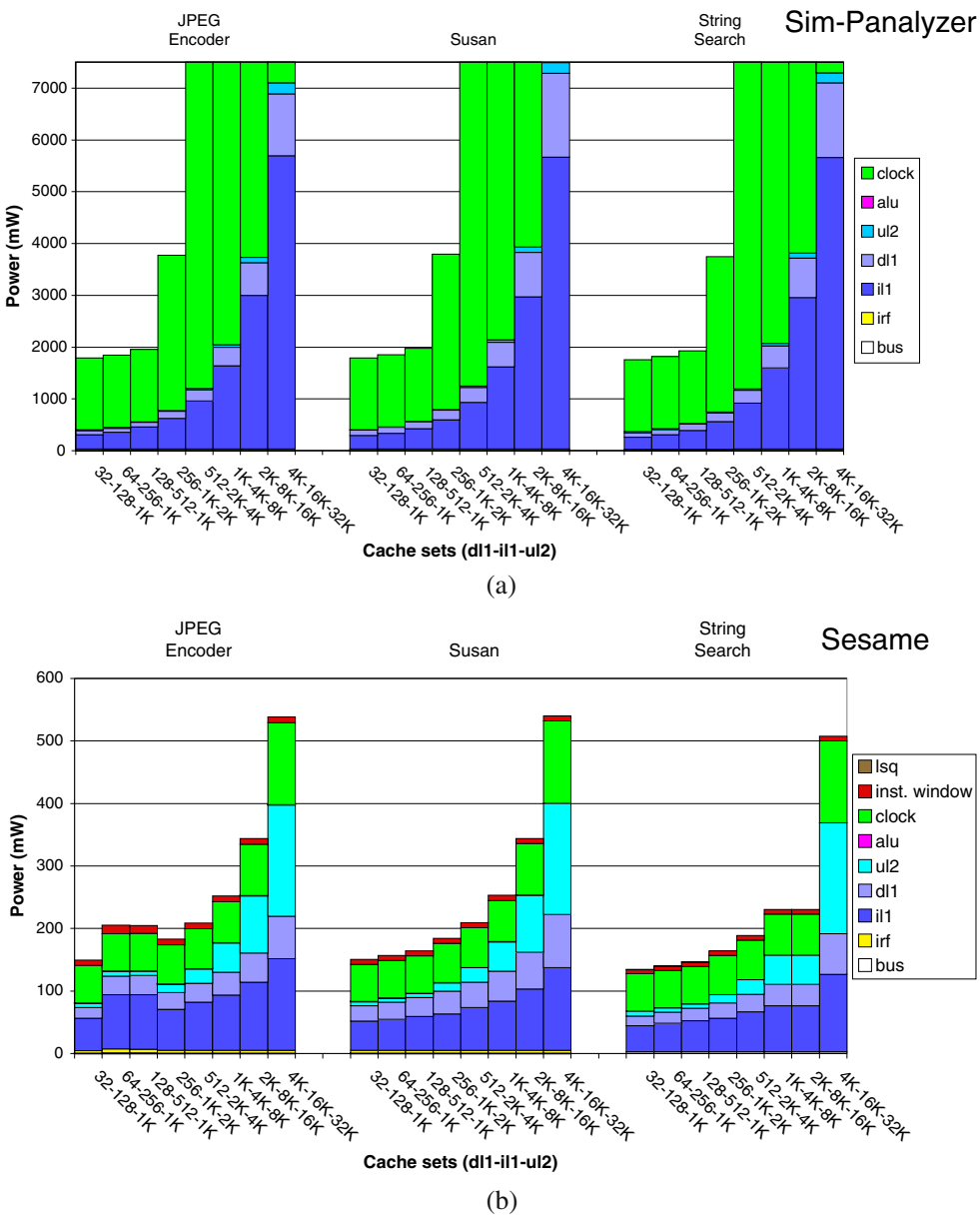


Figure 7 Power consumption estimation for Sim-Panalyzer (**a**) and Sesame (**b**) when varying the cache sizes.

to actually evaluate the signature-based power modeling technique and the consequences (no notion of separate instructions nor of the data that they use) that come with it. For this reason, we also present an experiment in which we use the access power consumptions from Wattch for the various components in our power model. This way, we try to align both power models such that we can actually measure the effects of the high abstraction level at which Sesame operates. Figure 8 shows the power consumption estimates for both Sesame and Wattch for all three benchmark applications. Here, we have executed cjpeg with three different input pictures of varying complexity: Cats (1.15 MB), Rain (150 KB) and Water (40 KB).

Using these aligned power models, the mean difference between Sesame and Wattch is only 3.1%, with a worst case of 4.2%. Individual components which are data dependent may have a larger mean error (and standard deviation σ), as indicated by Table 1. For example, the difference in power estimation for the instruction register file is 11% and for the level-2 cache even 22.6%. However, these components only account for a small fraction of the total power consumption. More importantly, for the level-1 caches—currently

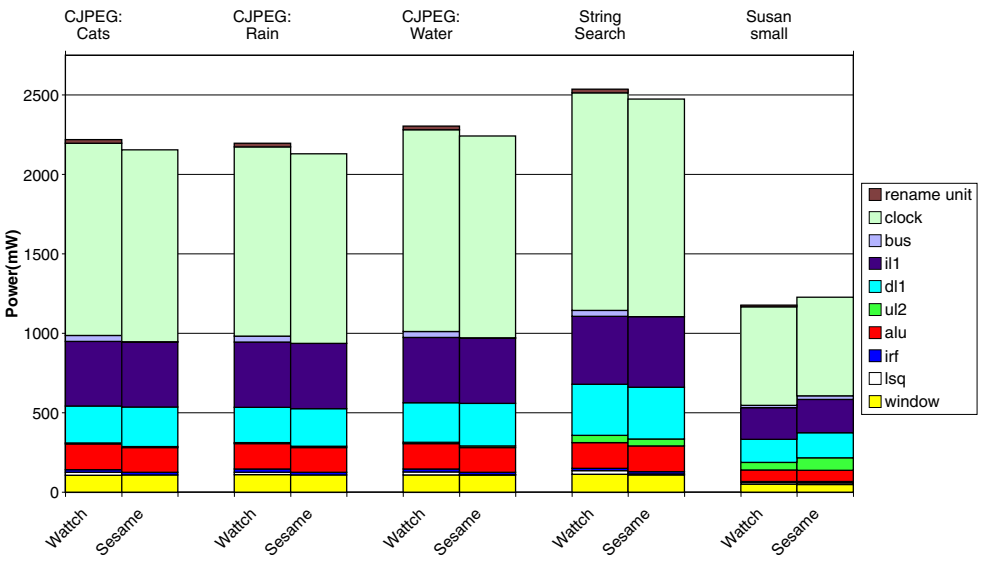


Figure 8 Comparing Sesame and Wattch when the same underlying power models are applied.

modeled with double-ended bitlines in both Sesame and Wattch, making their access energies largely data independent—the mean error is relative small: 2.1% for the level-1 instruction cache and 6.1% for the level-1 data cache. From these numbers, we can conclude that our high-level method may yield power estimations that are fairly close to those from tools such as Wattch.

We should note that Wattch and Sim-Panalyzer are also relatively high-level power simulators (in comparison to circuit-level power simulators such as Spice) and thus suffer from inaccuracies as well. Typically, they can be accurate within 10% [1, 5, 6]. But, as explained and demonstrated in [5], DSE studies can tolerate some error because relative accuracy is more important than absolute accuracy. Nevertheless, it would have been interesting to also compare our signature-based power estimates to power consumption numbers from real ARM or PowerPC processors. This is however considered as future work.

Since our initial aim was to speed up the power estimations in comparison to traditional ISS-based power simulators, we also measured the execution times of both the Sesame and Wattch[2] frameworks. Here, we should note that for Sesame we used a set-up in which Sim-Cache was co-simulated together with Sesame to provide our power model with exact cache missrates for application events (see Section 3). Decoupling Sesame and Sim-Cache (i.e., using average cache missrates for application events in Sesame rather than exact ones) would result in even much better performance of our power estimations. For the studied benchmark applications, Sesame is on average 10.5 times faster than Wattch. The largest speed-up we measured was 25. Given the fact that—with an appropriate power model—good power estimates can be achieved (see Fig. 8), this is a very promising result.

Table 1 The mean difference in power estimation between Wattch and Sesame at component level.

Component	Mean difference	σ
Window	1.7%	1.6
irf	11%	7.5
il1	2.1%	2.6
dl1	6.1%	2.9
ul2	22.6%	22.9
alu	1.9%	1.7

6 Related Work

High-level microprocessor power modeling techniques range from analytical methods [4, 15], based on e.g. statistical or information-theoretic techniques, to micro-architecture level instruction set simulators (ISSs) such as Wattch [6], Sim-Panalyzer [1] and SimplePower [25]. Clearly, within this range, there is a trade-off between accuracy and estimation performance. A fair number of efforts also address microprocessor power estimation at a level that is in between analytical and ISS-level models. Many of these efforts estimate power based on a-priori knowledge about instructions or segments of instructions. For example, the power consumption of separate instructions (or instruction pairs

[2]We note that Wattch is slightly faster than Sim-Panalyzer.

[14]) can be measured (using a real processor or a low-level simulator) after which the power consumption of an entire application involves the accumulation of these per-instruction power consumptions [20]. Such measurement-based power estimation can also be performed at a coarser granularity such as at the level of entire functions [18].

Another interesting high-level microprocessor power estimation approach is functional-level power analysis (FLPA) [12, 13]. In FLPA, a processor is modeled using a relatively small number of functional blocks, for which parameterized analytical power models are determined. These power models, which only have a few high-level parameters (like internal/external data access rate and average number of processing units used per cycle), are determined using linear regression with a set of power measurements in which the aforementioned high-level parameters are varied. For these measurements, real processors or lower-level simulators are used together with a set of benchmark programs (scenarios) that stress the different functional blocks in the processor. In [3], a hybrid FLPA and instruction-level microprocessor power estimation approach is proposed. They have shown that by incorporating instruction dependent behavior in the FLPA models, the accuracy of the models can be further improved.

In terms of abstraction level, our signature-based power estimation technique is also in between analytical and ISS-based models. Compared to the methods described above, we abstract from single instructions while still being able to apply an explicit micro-architecture model to perform power estimation. This allows us, in contrast to e.g. instruction-level power modeling and FLPA, to perform micro-architectural DSE. Also, unlike instruction-level power modeling and FLPA, we do not need to perform power measurements to determine the power consumption of (blocks of) instructions or, in the case of FLPA, functional blocks in the processor.

7 Conclusions

In this paper, we have presented a technique for high-level power estimation of microprocessors. This technique, which is based on abstract execution profiles called 'event signatures', operates at a higher level of abstraction than commonly-used instruction-level power simulators and should thus be capable of achieving good evaluation performance. The signature-based power modeling technique has been integrated in our Sesame system-level simulation and DSE framework and will eventually be extended to allow for system-level power modeling of an entire MP-SoC (i.e., also support the power modeling of the interconnect, shared memory, dedicated IP blocks, etc.).

We compared the results from our signature-based power modeling to those from the instruction-level simulators Wattch and Sim-Panalyzer. Here, it was shown that although the underlying power model we applied is able to show approximately the correct overall trends, it needs to be improved considerably to show the correct absolute power consumptions of the various architectural components. However, we also demonstrated that with a good underlying power model, the signature-based power modeling technique can yield accurate estimations (a mean error of 3.1% compared to Wattch in our experiments). Important to note here is that the power estimations based on our event signature technique are at least an order of magnitude faster than with Wattch (being the fastest of the two used instruction-level simulators).

References

1. Austin, T., Mudge, T., & Grunwald, D. Sim-panalyzer. http://www.eecs.umich.edu/~panalyzer/.
2. Austin, T. M., Larson, E., & Ernst, D. (2002). Simplescalar: an infrastructure for computer system modeling. *Computer, 35*(2), 59–67, Feb.
3. Blume, H., Becker, D., Rotenberg, L., Botteck, M., Brakensiek, J., & Noll, T. G. (2007). Hybrid functional- and instruction-level power modeling for embedded and heterogeneous processor architectures. *Journal of Systems Architecture, 53*(10), 689–702, Oct.
4. Brandolese, C., Fornaciari, W., Salice, F., & Sciuto, D. (2000). An instruction-level functionally-based energy estimation model for 32-bits microprocessors. In: *Proc. of the Design Automation Conference*, pp. 346–351.
5. Brooks, D., Bose, P., & Martonosi, M. (2004). Power-performance simulation: design and validation strategies. *SIGMETRICS Performance Evaluation Review, 31*(4), 13–18.
6. Brooks, D., Tiwari, V., & Martonosi, M. (2000). Wattch: A framework for architectural-level power analysis and optimizations. In: *Proc. of the Int. Symposium on Computer Architecture (ISCA)*, June.
7. Erbas, C., Pimentel, A. D., Thompson, M., & Polstra, S. (2007). A framework for system-level modeling and simulation of embedded systems architectures. *EURASIP Journal on Embedded Systems*, doi:10.1155/2007/82123.
8. Guthaus, M. R., Ringenberg, J. S., Ernst, D., Austin, T. M., Mudge, T., & Brown, R. B. (2001). Mibench: A free, commercially representative embedded benchmark suite. In: *Proc. of the IEEE Workshop on Workload Characterization*, pp. 3–14.

9. Jaddoe, S., & Pimentel, A. D. (2008). Signature-based calibration of analytical system-level performance models. In: *Proc. of Int. Symposium on Systems, Architectures, MOdeling and Simulation (SAMOS)*, pp. 268–278, July.
10. Kahn, G. (1974). The semantics of a simple language for parallel programming. In: *Proc. of the IFIP Congress 74*.
11. Sung Kim, N., Kgil, T., Bertacco, V., Austin, T., & Mudge, T. (2004). Microarchitectural power modeling techniques for deep sub-micron microprocessors. In: *Proc. of the International Symposium on Low Power Electronics and Design*, pp. 212–217.
12. Laurent, J., Julien, N., Senn, E., & Martin, E. (2004). Functional level power analysis: An efficient approach for modeling the power consumption of complex processors. In: *Proc. of the conference on Design, Automation and Test in Europe (DATE)*, pp. 666–668, Feb.
13. Laurent, J., Senn, E., Julien, N., & Martin, E. (2001). High level energy estimation for dsp systems. In: *Proc. of the Int. Workshop on Power And Timing Modeling, Optimization and Simulation (PATMOS)*, pp. 311–316.
14. Lee, M. T.-C., Fujita, M., Tiwari, V., & Malik, S. (1997). Power analysis and minimization techniques for embedded DSP software. *IEEE Transactions on Very Large Scale Integration Systems, 5*(1), 123–135.
15. Macii, E., Pedram, M., & Somenzi, F. (1998). High-level power modeling, estimation, and optimization. *IEEE Transactions on CAD of Integrated Circuits and Systems, 17*(11), 1061–1079.
16. Pimentel, A. D., Erbas, C., & Polstra, S. (2006). A systematic approach to exploring embedded system architectures at multiple abstraction levels. *IEEE Transactions on Computers, 55*(2), 99–112.
17. Pimentel, A. D., Thompson, M., Polstra, S., & Erbas, C. (2008). Calibration of abstract performance models for system-level design space exploration. *Journal of Signal Processing Systems for Signal, Image, and Video Technology, 50*(2), 99–114.
18. Qu, G., Kawabe, N., Usami, K., & Potkonjak, M. (2000). Function-level power estimation methodology for microprocessors. In: *Proc. of the Design Automation Conference*, pp. 810–813.
19. Vijaykrishnan, N., Chen, R., & Irwin, M. J. (1999). Clock power issues in system-on-chip designs. In: *Proc. of IEEE CS Workshop on VLSI*, pp. 48–53.
20. Russel, J. T., & Jacome, M. F. (1998). Software power estimation and optimization for high performance, 32-bit embedded processors. In: *Proc. of the Int. Conference on Computer Design (ICCD)*, pp. 328–333, Oct.
21. Sung Kim, N., Austin, T., Mudge, T., & Grunwald, D. (2001). Challenges for architectural level power modeling. In: *Power Aware Computing*, pp. 48–53.
22. Tarjan, D., Thoziyoor, S., & Jouppi, N. P. (2006). Cacti 4.0. Technical Report HPL-2006-86, Hewlett-Packard.
23. Thompson, M., Pimentel, A. D., Polstra, S., & Erbas, C. (2006). A mixed-level co-simulation method for system-level design space exploration. In: *Proc. of the IEEE Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia'06)*, pp. 27–32, Oct.
24. van Stralen, P., & Pimentel, A. D. (2007). Signature-based microprocessor power modeling for rapid system-level design space exploration. In: *Proc. of the IEEE Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia '07)*, pp. 33–40, Oct.
25. Ye, W., Vijaykrishna, N., Kandemir, M., & Irwin, M. J. (2000). The design and use of simplepower: A cycle-accurate energy estimation tool. In: *Proc. of the Design Automation Conference (DAC)*, June.

Peter van Stralen received his BSc degree in computer science from the University of Amsterdam, where he currently is a MSc student. His research interests include embedded systems, design space exploration and concurrency.

Andy D. Pimentel received the MSc and PhD degrees in computer science from the University of Amsterdam, where he is an associate professor in the Department of Computer Science. He is member of the European Network of Excellence on High-Performance Embedded Architecture and Compilation (HiPEAC). His research interests include computer architecture, computer architecture modeling and simulation, system-level design, design space exploration, performance analysis, high-level power estimation, embedded systems, and parallel computing. He is a senior member of the IEEE and a member of the IEEE Computer Society.