

Enhancing Behavioral-Level Design Flows with Statistical Power Estimation Capabilities (Extended Version of PATMOS 2003 Paper)

Journal:	<i>IEE Proc. Computers & Digital Techniques</i>
Manuscript ID:	CDT-2004-5187
Manuscript Type:	Research Paper
Date Submitted by the Author:	22-Dec-2004
Keyword:	DIGITAL INTEGRATED CIRCUITS

powered by ScholarOne
Manuscript Central™

Enhancing Behavioral-Level Design Flows with Statistical Power Estimation Capabilities *

(*Extended Version of PATMOS 2003 Paper*)

B. Arts¹ L. Benini³ N. van der Eng¹ M. Heijligers¹
A. Kenter¹ E. Macii² H. Munk¹ F. Theeuwes¹

¹ Philips Research Labs, ED&T/Synthesis, Eindhoven, NL

² Politecnico di Torino, DAUIN, Torino, I

³ Università di Bologna, DEIS, Bologna, I

Abstract. Power estimation of behavioral hardware descriptions is a difficult task, as it entails inferring the hardware architecture on which the behavioral specification will be mapped through synthesis *before* the synthesis is actually performed. To cope with the uncertainties related to handling behavioral descriptions, we introduce the concept of statistical estimation, and we present how a prototype statistical power estimator has been implemented within a high-level design exploration framework (the AspeCts environment).

1 Introduction

The addition of the power dimension to the already large area/speed design space dramatically expands the number of available design alternatives. Therefore, in a power-conscious synthesis flow, it is of increasing importance the ability of estimating the impact of the various choices made by the designers or the effects of automatic optimizations on the final power budget.

1.1 Previous Work

Most of the research on power estimation has initially focused on gate and transistor levels, and only recently it has extended to higher levels of abstraction (i.e. RTL and behavioral).

RTL power estimation is at the transition point between research and industrial applications [?]. RTL power estimators are commercially available [2, 3, 4] and are now gaining widespread acceptance in the design practice.

Our work focuses on an even higher level of abstraction, namely, the behavioral level. The main challenge in behavioral power estimation is the filling of the abstraction gap that does exist between specification and implementation. A behavioral specification leaves a significant amount of freedom for hardware implementation: Scheduling of operations, type of functional units, assignment. In other words, a single behavioral specification can be mapped to a wide variety of hardware architectures, characterized by largely different power consumption.

* This work was supported by Philips Electronics B.V. under grant n. 200/2001.

Most behavioral power estimation approaches [5] adopt a highly simplified power consumption model, where an energy cost is associated to the operations performed in the behavioral specification, and total energy is computed by summing the cost of all operations. The main shortcoming of this model is that it does not account for two critical factors, namely: (i) The presence of hardware which is required for correctly performing a computation, but it is not explicitly represented as operations in the behavioral description. (ii) The switching activity dependency of power consumption. Neglecting these factors can lead to inaccurate estimates, therefore adequate countermeasures have to be taken.

The approach adopted to overcome the accuracy limitation of the cost-per-operation model follows the direction of specifying more details on how the mapping of behavior onto implementation is performed in the target design flow. This is similar to the “fast synthesis” step performed by advanced RTL power estimators, where a simplified gate mapping algorithm is applied to logic blocks for which a macro-model does not exist. In behavioral power estimation, information on operation scheduling and on the number and type of functional units can be taken into account to reduce uncertainty on the final hardware implementation, thus to improve the accuracy of both switching activity and capacitance estimation [6, 7, 8]. However, given the high-level of abstraction of this information, significant uncertainty remains on the final hardware architecture. Hence, several authors have remarked that power estimation at the behavioral level should not produce a single-value estimate, but it should identify ranges representing uncertainty intervals. A few techniques have been proposed to compute these intervals. A deterministic lower-bound vs. upper bound solution was proposed in [9]. In alternative, probabilistic approaches, modeling the uncertainty on power estimation as the sample space of a random variable have been explored in [10]. The techniques of this paper follow and extend the latter approach.

1.2 Contribution

We introduce the concept of statistical power estimation for hardware components that are not explicitly instantiated in the behavioral description being considered (e.g., steering logic, data-path registers, wiring and control logic). We call these components *implicit*, because they are required in the RTL implementation, and yet they are not explicitly represented in the design database created from the input description (C code, in our case). To estimate the power consumed by these components, we perform Monte-Carlo sampling of multiple instantiations, that is, we consider architectures for which different assumptions for the implicit components are made. Then, we obtain a distribution of average power values, on which we perform statistical analysis.

Besides models for functional units, we also propose statistical models for steering logic, clock, wiring, controller and memory ; this with the purpose of enabling the implementation of a complete power estimation prototype tool.

The estimator was developed as an add-on library to the AspeCts behavioral exploration tool by Philips [11], and accuracy assessment was made against data determined on synthesized RTL descriptions (derived from some benchmarks) by means of state-of-the-art tools (namely, BullDAST PowerChecker [4]).

2 The AspeCts Environment

The design of electronic systems can roughly be divided into two phases. In the first phase, the functionality of the system is specified, typically defined by a collection of communicating processes. Each of these processes is specified by means of a behavioral C algorithm. The second phase concerns the actual implementation of the system by defining a specific hardware architecture consisting of a collection of connected components and mapping the processes onto these components (see Figure 1).

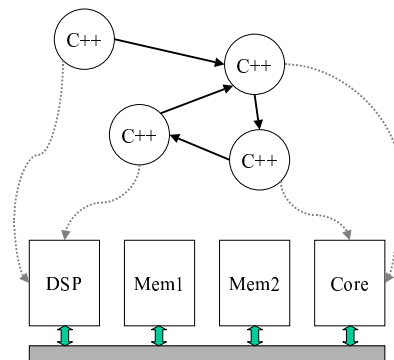


Fig. 1. Mapping a System-Level Description onto an Architecture.

A system-level simulator offers the capability of investigating user-defined mappings, by visualizing the performance of the system as a whole, so that the designer can make design trade-offs.

The input of a system-level simulator consists of a system-level specification, a hardware structure made of a collection of connected components, and a mapping of processes to components. Because the whole system is under investigation, and because a specification tends to change even after the implementation phase has started, this might result in many different mappings. If an algorithm is mapped to a dedicated, but not yet designed, hardware component, it is not realistic to describe each possible mapping by a dedicated implementation or specific production code.

Therefore, system-level simulation tools provide the possibility of characterizing a mapping by means of a performance model. The simulation of the behavioral system-level description is driven by the performance models provided. In practice, a designer is confronted with algorithms not developed by himself. Besides that, the designer has to cope with some predefined hardware constraints, relate this to the algorithms, and keep track of the overall implementation consequences. This makes it very difficult to generate realistic performance models, without detailed and time-consuming analysis of the code. Automating these tasks is thus desirable.

A method for generation of performance models in a system-level environment with a realistic underlying implementation, should at least satisfy the following requirements:

- General C input. The C-language is important for system-level design, because it is an obvious language for developing behavioural algorithms.
- Automatic scheduling, allocation, memory analysis, power analysis, parallelization and pipelining capabilities, which are needed to make a mapping to an implementation. These methods should be generic, should not perform black-box magic (i.e are predictable and steerable), and should not depend on a particular target architecture or implementation technology.
- Cope with user constraints, so that a designer can provide hints towards a satisfiable implementation.
- A GUI (graphical user interface), which gives feedback on all the aspects of the possible implementation (e.g., power consumption, time, functional unit usage, memory bandwidth).

The AspeCts environment, whose flow of operation is shown in Figure 2, fulfills these goals. It consists of two main parts: The first one is in charge of executing a C algorithm, scheduling all activities that take place, and storing such activities in a database. In the second part, the database is used to perform memory analysis and optimization based on the scheduled activities.

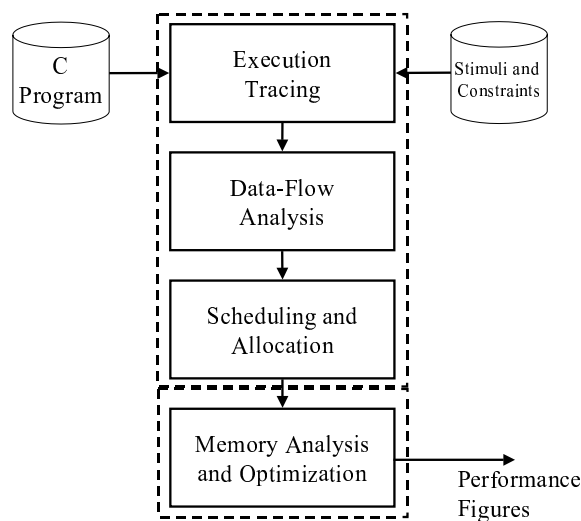


Fig. 2. AspeCts Operation Flow.

2.1 Execution Tracing, Data-Flow Analysis and Scheduling

To be able to investigate the performance of a C algorithm with respect to its typical usage, a simulation-based approach was chosen.

During program simulation, the execution trace is determined, i.e., the sequence of operations which are executed when running the program. Timing information to the arguments of these operations is then added; scheduling and data-dependency analysis is thus implicitly performed.

AspeCts supports different scheduling strategies, including sequential, data-flow based and control-flow based.

- *Sequential.* The most straightforward case is to use a sequential execution model, meaning that the schedule is exactly the same as the execution order defined by the C/C language.
- *Control-flow based.* This will maintain the control flow of the input description. Within each basic block, only data dependencies are enforced. A greedy heuristic can be applied, trying to keep variable reads as close as possible to the variable writes, in order to minimize memory size.
- *Data-flow based.* This method is almost similar to the previous item, except in this case basic block structures are not preserved. The available freedom is used to minimize memory size. This results in a large search space that may be difficult to explore efficiently. As a result, this approach may be less robust, making it also less transparent to the user.

The choice for one of the schedule approaches is a matter of balancing the optimization freedom versus the implementation risk: the more freedom is offered, the better the optimum solution becomes, but for an implementation point of view also requires more aggressive transformation of the input.

To consider implementation hints provided by the designer, the scheduler in AspeCts also accepts various kinds of constraints, including timing constraints and functional resource constraints.

- *Time constraints.* This allows the designer to define the schedule of specific productions and consumptions directly or relatively.
- *Functional resource constraints.* The designer can specify the collection of operations that should be implemented by a particular functional resource. The maximum number of available functional resources can also be specified. The scheduler will schedule no more operations in parallel than this maximum.

To make the results repeatable, the constraints can be added to the C code (e.g., by means of an include file) in a C-flavoured syntax. To be able to get an in-depth understanding of these results, the method is extended with *tracing features*, where the designer can trace all productions and consumptions to a particular (array of) variables, and *graphical output features*.

2.2 Memory Analysis

Memory becomes more and more important in the design of electronic systems, as area, delay and power dissipation, heavily depend on the chosen memory architectures. A designer can be supported in making design decision by good estimations of the required memory capacity and bandwidth. Therefore, analysis is required to prove the feasibility of a chosen memory architecture for a given algorithmic specification. Because most scalars are stored in data-path registers and because compilers and high-level synthesis tools can handle scalars quite well, these steps are mainly intended for multi-dimensional arrays. In C terminology, this means that we assume that memory is closely related to the concept of C arrays. To provide unlimited memory bandwidth and size for all arrays, and hence enable a maximum parallel schedule where all array accesses can be scheduled without restrictions, the method assumes that each array is “mapped to” a separate memory. To offer the designer a way to define specific memory structures, the following tasks can be performed: Memory allocation, binding, compaction, sizing, bandwidth analysis and correctness checks. The final feedback provided to the designer includes, for each memory array, information about its size, the number of productions and consumptions, and the maximum number of values stored in the array simultaneously.

3 Power Modeling in AspeCts

3.1 Functional Units

One of the major sources of error in power estimation originates from input pattern dependency. Our functional unit models are sensitive to the average switching activity and signal probability at their inputs. Estimates of these quantities are produced by sampling the values propagating through the various operations during the execution of an AspeCts model. The activity-driven macro-models are look-up tables (LUT), which have proved to be very reliable and robust over a wide range of input statistics.

LUT models make use of a tabular representation of the functional relation between boundary statistics and power. The number of parameters used to represent boundary statistics has a great impact on model size, since each independent variable adds a dimension to the parameter space. We used three parameters (the average input signal probability P_{in} , the average input activity D_{in} and the average output activity D_{out}), leading to a 3-dimensional LUT.

Discretization is applied to the parameters in order to obtain a finite number of configurations to be associated to LUT entries. Characterization consists of performing low-level power simulation experiments for all configurations of the discretized boundary parameters, in order to characterize the corresponding LUT entry by storing the power value obtained in it.

The key advantage of LUT models over regression equations is their capabilities of representing non-linear dependencies. On the other hand, the proper use of LUT models involves a critical trade-off between model size (the number of entries in a LUT is $L * N$, where L is the number of discretization levels and N the number of parameters), characterization effort and accuracy.

3.2 Steering Logic

The assumption on the structure of the steering logic is that it is implemented by means of a multiplexer network. The basic brick of the steering logic net is thus the 2-to-1 MUX. An activity-sensitive power model for this element is characterized as any other functional component for the chosen technology library. The size and shape of the steering logic network is determined by the number of inputs such a network is supposed to convey to the inputs of the shared functional units and shared registers. This approach, which implicitly assumes a quasi-balanced tree network of 2-to-1 MUXes, offers a very simple mean for estimating the steering logic power. In fact, the process does not require the introduction of scaling factors for extending the characterized model to components with wider bit-width, nor it requires additional runs of the model characterization procedure. The drawback of this solution, however, stands in the fact that, when real hardware is synthesized, steering logic trees are usually unbalanced and, most important, other components than 2-to-1 MUXes are instantiated. As a consequence, power estimates achieved with this model tend to be pessimistic.

3.3 Clock and Wires

For what concerns the estimation of the power consumed by the clock tree network and by the interconnect wires, the models we have developed are somehow reminiscent of the models used by BullDAST PowerChecker [4] in the context of RTL power analysis. They are based on the observation that the power consumed by the clock tree (interconnect wires) is in strict relationship with the complexity of the clock network (interconnect network); this, in turn, is very much related to the number of sequential elements that the clock signal has to drive (number of estimated cells in the gate-level netlist of the design). The models we have adopted are analytical and depend, primarily, on the number of estimated registers (estimated cells) of the input description.

3.4 Controller

The logic that controls steering of MUXes and handling of loops and conditionals is not explicitly represented in any AspeCts internal data structure. Yet, it is a known fact that control logic can consume a non-negligible fraction of the total power, therefore it should be taken into account to improve the accuracy of power estimation. The structure of the control logic, a finite-state-machine (FSM), is in essence determined by the scheduling. Once the schedule is known, the controller has to follow the sequence of control steps to activate the required components with control signals that drive the steering logic.

For a pure data-flow specification, control flow is straight-line (i.e., no conditionals and loops) and the FSM is very simple. It is a chain of states with an added transition from the last state to the first (which represents the re-start of a computation) and optionally a self-loop in the first state (which represents the reset state, if present).

Let us assume, for the time being, a simple single-chain structure for the FSM (this assumption will be relaxed later). Controller power estimation is based on a simple characterization procedure. In a preliminary pre-processing step, a data set is created by instantiating several *chain* FSMs, with varying number of states and outputs, synthesizing them and estimating their power consumption via gate-level simulation. Then, the data are collected in a look-up table and an interpolation/extrapolation rule is provided. The final results of the characterization procedure can be viewed as the construction of a power model $P(N_{states}, N_{outputs})$, which returns an expected power consumption for a chain FSM with number of states N_{states} and number of outputs $N_{outputs}$. Notice that this model is not input pattern sensitive because the FSM is autonomous. In presence of a reset-state self-loop, we can add another sensitivity parameter in the model, namely the percentage of time in which the reset signal is asserted.

If the specification is not a pure data-flow, but it contains significant control-flow elements, the model of the FSM power is more complex, but it is obtained via a similar characterization process. In this case, the modeling process is based on the assumption that a FSM for a general specification, with non-trivial control flow is mostly chain-based, with multiple-exit states, corresponding to conditionals, and feedback edges, corresponding to loops. In this case, the FSM logic will be more complex to account for the additional edges, hence the power model contains a third parameter, i.e., the number of *extra edges* w.r.t. a single-chain FSM (where the number of edges is the same as the number of states). Thus, the general power model becomes $P(N_{states}, N_{outputs}, N_{extraedges})$. Characterization of the model is performed with a similar process as for single-chain FSMs, with the only difference that a larger sample FSM set has to be generated, in order to sample also the third parameter $N_{extraedges}$.

It is clear that, in this case, the path followed by the FSM is data-dependent; therefore, we expect input pattern dependence, to some degree, of the FSM power consumption. However, this effect is quite minor, especially if a flat FSM with logarithmic state encoding is generated, because all the next state logic is shared and most of the output logic has, as input, all the state registers.

3.5 Memories

Power modeling for memories (i.e., the pre-characterization of cost-per-access constants) does not leverage the characterization flow assumed for functional units, because memories are either instantiated as hard macros or they are created directly at the layout level by memory generators. If hard memory macros are available, their energy-per-access constants are usually specified in their data-sheets. Similarly, memory generators usually provide power views to designers. In case of incompletely specified memory macros, where the power view is absent, we provide a parametric technology-based model, which determines cost-per-access estimates for memory macros given a few basic technology-dependent parameters, such as wire capacitance, drain and gate capacitances, power supply voltage. This default power model is based on CACTI, a memory performance and power estimation tool widely used in the research literature [12].

4 Statistical Power Estimation

The main purpose of the statistical estimator, called in the sequel Monte-Carlo AspeCts (MC-AspeCts), is to provide a quantitative assessment of the uncertainty associated with power estimation at a high level of abstraction.

4.1 The Concept

It is not realistic assuming that the accuracy of behavioral (C-level) power estimation can be comparable to that of RTL estimation: Design choices associated to these degrees of freedom are not fully specified, and a margin of uncertainty still remains on the consumption of the final hardware. Providing a single-value estimate automatically implies an arbitrary selection of a hardware configuration, which might be far away from the final implementation. It is therefore important giving the designer a feeling for the uncertainty while, at the same time, providing him/her with a meaningful power estimate.

To be more specific, let us focus on a concrete example: Consider a behavioral description where 4 operations have to be mapped onto two functional modules (e.g., four additions mapped onto two adders), as depicted in the data-flow of Figure 3. In AspeCts, the user specifies a resource constraint (i.e., two adders), but he/she does not explicitly binds operations to functional modules. Internally, AspeCts will map operations onto functional modules (e.g., OP1, OP2, executed by adder ADD1, and OP3, OP4 executed by ADD2), but this assignment is not guaranteed to be the same as in the final implementation. Indeed, many other assignments, which are compatible with the given scheduling and allocation constraints, are possible. These assignments do not violate neither user constraints nor scheduling decisions, but they can have a significant impact on power consumption. This is because the sharing pattern of a functional module determines the switching activity at its inputs and outputs, which in turns determines its switching activity and its internal power dissipation.

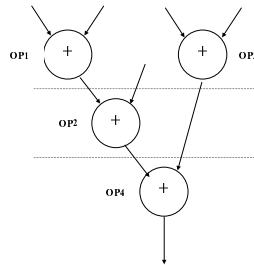


Fig. 3. Example of Data-Flow Graph.

To better explain this concept, let us consider the data-flow graph of Figure 3 with a resource constraint of 2 adders. The assignment of OP1 and OP2 to ADD1 and OP3, OP4 to ADD2 are compatible with the schedule and with the resource constraint, but so is the assignment of OP1, OP2, OP4 to ADD1 and OP3 to ADD2. Clearly, the switching activity and, consequently the power consumption of ADD1 and ADD2 can change significantly in the two cases.

In addition, we note that the two bindings produce completely different steering logic (MUX trees) at the inputs of the two adders: Four 2-input MUXes in the first case, and two 3-input MUXes in the second (at the inputs of ADD1 only). To address the unavoidable uncertainty on switching activity and on steering logic structure caused by the high level of abstraction at which we perform power estimation, we view the power consumption as a random variable P . The sample space of such a variable is the space of all operation bindings compatible with scheduling and resource constraints, and the values of the variable are the power dissipation estimates for a given binding. In order to estimate power for a design we should not limit ourselves to providing a single value of the random variable; instead, we should characterize its distribution in terms of mean value and variance. This would allow a designer to perform design space exploration without going through the complete design cycle for different architectural options.

The statistical distribution of the random variable P is unknown a priori. To estimate its mean value and variance, we resort to Monte-Carlo sampling. In other words, we generate a number of bindings, compatible with resource constraints and scheduling, and we estimate the power consumption for each one of them. Exploration of the sample space stops when a Monte-Carlo convergence test confirms that a stable average value and variance estimate has been obtained. Thanks to the central limit theorem, we know that Monte-Carlo convergence is very rapid, and in practice a few tens of sample points are sufficient to obtain reliable estimates.

4.2 Implementation in AspeCts

Monte-Carlo sampling of the entire design space of legal bindings is made quite easy in the AspeCts framework. At each scheduling step a list of “free” functional units is available. In the standard operation mode, AspeCts scans the list in order, assigning units in the fixed list order. When performing Monte-Carlo sampling, we can just randomly select elements from the list. We only have to ensure that different runs of AspeCts do not repeat the same pseudo-random sequence. This is easily obtained by changing the seed of the RAND function.

The computational cost of Monte-Carlo sampling with respect to the cost of a single power estimate is linear in the number of samples. As stated before, a few tens of samples are generally sufficient. In the current implementation, we just run AspeCts in Monte-Carlo sampling mode multiple times. This choice has minimal impact on the tool flow (it can be implemented with a simple script that calls MC-AspeCts multiple times and collects results) and it can be easily parallelized on multiple machines for speed-up purposes.

The final output of MC-AspeCts is an estimate of the mean value $\text{mean}(P)$ of random variable P and its variance $\text{var}(P)$. It should be clear that $\text{mean}(P)$ and $\text{var}(P)$ give information on what is the expected power consumption of an implementation of the design, and how much uncertainty we have on this estimate because of the degrees of freedom left unspecified at the high level of abstraction. In addition to the estimate regarding the whole design, the breakdown of power consumption over the different components of the design is optionally provided.

5 Experimental Results

We have performed extensive benchmarking of the MC-AspeCts tool. In the sequel, we first report some results concerning the analysis of the Monte-Carlo convergence of the estimation. Then, we show some results about the accuracy we have achieved in the estimation with respect to synthesized RTL descriptions obtained by imposing resource and throughput constraints similar to those used for AspeCts estimation. RTL estimates have been determined using BullDAST PowerChecker; model characterization for both MC-AspeCts and PowerChecker was done using a Philips 0.18 μ m CMOS technology library.

5.1 Results on Monte-Carlo Convergence

The example we consider is the simple loop shown below:

```
for (I=0; I<n; I++) {
  A = vect[I] * x; B = vect[I] * y;
  C = vect[I] * z; D = vect[I] * w;
  F = A + B; G = C + D;
  Out = F + G;
}
```

This example enables, for a given set of resource constraints, different assignments of data to operators and registers, and thus well exercises the calculation of mean value and variance of the power random variable. The results obtained for a given test-bench (that is, a vector of input samples, whose length was fixed to 1000 integer numbers) are reported in Figures 4 and 5..

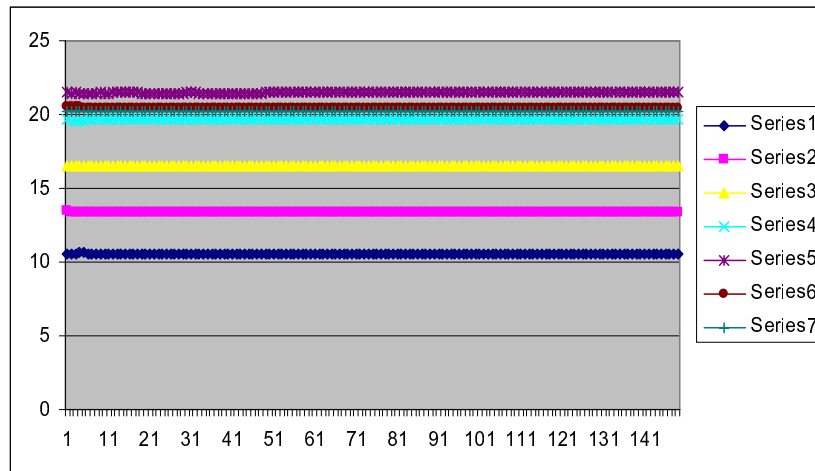


Fig. 4. Mean Value for Simple Example.

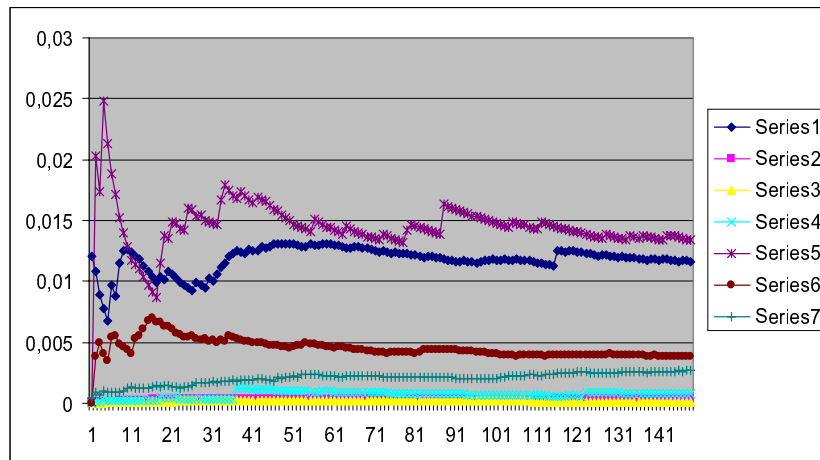


Fig. 5. Variance for Simple Example.

The curves in the diagrams, identified as *Series*, refer to different configurations of the resource constraints. Depending on the degrees of freedom in the allocation of data to resources, the variance of power is larger and requires a longer time to settle (the configuration in which there is no freedom, i.e., 4 multipliers and 2 adders, has null variance and it is not shown in the diagram).

The results confirm that Monte-Carlo convergence is achieved very quickly (no more than 50 simulation are needed to reach stability in the variance). Also, the mean value is very stable and provides a good indication of the *average case* power consumption of the generic architecture. Last, variance analysis allows to early determining *risky* constraint configurations, that is, configurations for which power consumption may vary after the architecture is synthesized.

5.2 Power Estimation Results

Eight examples, specified in C, constitute the benchmark suite for assessing the estimation accuracy of MC-AspeCts. For each example, one or more streams of input patterns characterized by different correlations were defined; this with the goal of testing the tool under different operating conditions of the benchmarks. Specific ad-hoc files (e.g., image files) were also used for some benchmarks.

Table 1 collects the data (power values are in mW). The estimation error is around 22%, on average, with values of the variance well below 0.40, indicating that the power random variable has different stability depending on the benchmark. By looking at the breakdown of power estimates over the various components (not reported here for space reasons), we observe that the major sources of inaccuracy come from MUX, wiring and clock power estimation, while results are satisfactory for what concerns functional units, control logic and memories.

Benchmark	MC-AspeCts		PowerChecker	Δ [%]
	Mean Val	Variance		
GCD	1.95	0.18	1.71	14.0
BSORT	178.17	0.31	151.53	17.6
SSORT	127.53	0.12	107.82	18.3
Compl-Mult	89.46	0.37	74.17	20.6
FIR Filter	2.31	0.09	1.98	16.7
AVG-Int	2.38	0.13	1.88	26.6
PCX-Dec	218.19	0.34	168.89	29.2
EDGE-Detect	18.75	0.22	13.72	36.7
Average				22.6

Table 1. Statistical Estimation Results.

6 Conclusions

Power estimation for behavioral descriptions is a difficult problem, due to the degrees of uncertainty that are intrinsic of the level of abstraction at which the hardware system is specified.

In this paper, we have proposed a solution to the behavioral power estimation problem based on the concept of statistical power modeling. This concept has been implemented into a prototype tool built on top of an industry-strength design exploration environment. The estimation results obtained on a set of benchmarks have been validated against RTL estimation results determined using a commercial power estimator.

References

1. E. Macii, M. Poncino, "Power Macro-Models for High-Level Power Estimation", *Low Power Electronics Design*, C. Piguet Editor, CRC Press, September 2004.
2. Synopsys, Inc., *Synopsys PowerCompiler*, www.synopsys.com.
3. Sequence Design, Inc., *PowerTheater*, www.sequencedesign.com.
4. BullDAST s.r.l., *PowerChecker*, www.bulldast.com.
5. R. San Martin, J. P. Knight, "Power-Profiler: Optimizing ASICs Power Consumption at the Behavioral Level," *DAC-32: ACM/IEEE Design Automation Conference*, pp. 42-47, San Diego, CA, June 1995.
6. R. Mehra, J. Rabaey, "Behavioral Level Power Estimation and Exploration," *IWLDPD-94: ACM/IEEE International Workshop on Low Power Design*, pp. 197-202, Napa Valley, CA, April 1994.
7. A. Raghunathan, N. K. Jha, "SCALP: An Iterative-Improvement-Based Low-Power Data Path Synthesis System," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 16, No. 11, pp. 1260-1277, November 1997.
8. K. Khouri, G. Lakshminarayana, N. K. Jha, "High-Level Synthesis of Low Power Control-Flow Intensive Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 18, N. 12, pp. 1715-1729, December 1999.

9. L. Kruse, *et al.*, "Estimation of Lower and Upper Bounds on the Power Consumption from Scheduled Data Flow Graphs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 9, No. 1, pp. 3-14, February 2001.
10. D. Bruni, A. Bogliolo, L. Benini, "Statistical Design Space Exploration for Application-Specific Unit Synthesis," *DAC-38: ACM/IEEE Design Automation Conference*, pp. 641-646, Las Vegas, NV, June 2001.
11. M. J. M. Heijligers, A. Hogenhuis, "Analyzing Architectural Aspects of Behavioral Descriptions," *Conf. on Embedded System Design*, pp. 239-250, 2000
12. S. J. E. Wilton, N. P. Jouppi, "CACTI: An Enhanced Cache Access and Cycle Time Model," *IEEE Journal of Solid-State Circuits*, Vol. 31, No. 5, pp. 677-688, May 1996.