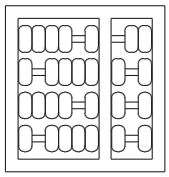


INSTITUTO DE COMPUTAÇÃO
UNIVERSIDADE ESTADUAL DE CAMPINAS

PowerSC: A SystemC-based Framework for Power Estimation

F. Klein *R. Leao* *G. Araujo*
L. C. V. dos Santos *R. Azevedo*

Technical Report - IC-07-002 - Relatório Técnico

February - 2007 - Fevereiro

The contents of this report are the sole responsibility of the authors.
O conteúdo do presente relatório é de única responsabilidade dos autores.

PowerSC: A SystemC-based Framework for Power Estimation

Felipe Klein
klein@ic.unicamp.br

Roberto Leao
leao@inf.ufsc.br

Guido Araujo
guido@ic.unicamp.br

Luiz Claudio Villar dos Santos
santos@inf.ufsc.br

Rodolfo Azevedo
rodolfo@ic.unicamp.br

Abstract

Although SystemC is considered the most promising language for system-on-chip functional modeling, it doesn't come with power modeling capabilities. This work presents PowerSC, a novel power estimation framework which instruments SystemC for power characterization, modeling and estimation. Since it is entirely based on SystemC, PowerSC allows consistent power modeling from the highest to the lowest abstraction level. Besides, the framework's API provides facilities to integrate alternative modeling techniques, either at the same or at different abstraction levels. As a result, the required power evaluation infrastructure is reduced to a minimum: the standard SystemC library, the PowerSC library itself and a C++ compiler. Although RTL power macromodeling is a mature research topic, it is not yet broadly accepted in the industrial environment. One of the main reasons impairing its widespread use as a power estimation paradigm is that each macromodeling technique makes some assumptions that lead to some sort of intrinsic limitation, thereby affecting its accuracy. Therefore, alternative macromodeling methods can be envisaged as part of a power modeling toolkit from which the most suitable method for a given component should be automatically selected. This paper describes a new multi-model power estimation engine that selects the macromodeling technique leading to the least estimation error for a given system component depending on its input-vector stream properties. A proper selection function is built after component characterization and used during estimation. Experimental results show that our multi-model engine improves the robustness of power analysis, reducing significantly the average and the maximum estimation errors, as compared to conventional single-model estimation.

1 Introduction

Power consumption has been one of the most important issues in contemporary digital circuit design. The offer of increasing amounts of ever-shrinking transistors and the demand for battery-powered mobile devices have made low-power consumption one of the most important concerns in contemporary VLSI design.

Despite the large availability of power-aware CAD tools at the gate and circuit levels, they result in high turnaround times, since they belong to late phases of the design flow. Therefore, it becomes mandatory to raise the abstraction level. At higher levels,

the most important figures are estimation speed and the relative accuracy between design alternatives, since even rough estimates may largely reduce the design time.

For allowing design descriptions in multiple levels of abstraction (e.g. gate-level, RTL, ESL), SystemC [1] has been growing in importance within the design community. Although it provides constructs for the description of functionality, structure and communication, SystemC doesn't come with power-oriented libraries.

This lack of power support motivated us to develop a framework called PowerSC, built on top of the SystemC library, for power assessment through multiple levels of abstraction. The key idea was to extend SystemC by adding power-aware C++ classes. Moreover, PowerSC provides an API to easily support the integration of alternative estimation techniques. Essentially, estimation at a given abstraction level relies on the characterization at the immediate lower level. We evaluate the framework focusing on the RT and gate levels, showing how distinct macromodeling techniques at the RT level can share the same PowerSC characterization mechanism at the gate level.

Power modeling is the keystone for the whole building of power-conscious design tools. Power macromodeling is considered today the state-of-the-art paradigm for power estimation at the RT level [2], with a variety of equation and table-based approaches [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17]. Each approach has many variants depending upon the parameters chosen to capture power variation (e.g. signal probability, transition density, spatial correlation and so on).

Despite its maturity, macromodeling is not yet widely accepted as an industrial *de facto* standard for power estimation at the RT level. This paper claims that one of the main reasons impairing its widespread use is that each macromodeling technique makes assumptions that lead to some sort of intrinsic limitation, thereby affecting its accuracy. In [5], for instance, signal statistics are assumed as uniformly distributed among all inputs and outputs, although input signal imbalances may result in significant errors [18]; the training set is limited to a single input stream per characterized point in [3], even though the choice of a proper set is crucial to obtaining high-quality models [19].

We discuss the limitations of macromodeling techniques that can lead to highly inaccurate estimates. For that purpose, we performed a quantitative comparison of three different and well-known macromodeling techniques at the RT level. Our experimental results clearly show that each macromodel is better suitable for a specific region of the input space.

Although much has been written on macromodeling, to our knowledge, no similar direct comparison of alternative techniques is reported in the literature.

Most power estimation tools rely on a single macromodeling technique. However, given a system component, there might exist an alternative technique leading to a more accurate estimate. Therefore, the proper selection of an alternative from a kit of supported macromodels could be envisaged as a way of optimizing the overall power estimation accuracy.

Although a related work [20] reports the use of multiple macromodels to improve power assessment performance, to our knowledge, no approach has been proposed to address the impact of multiple modeling techniques on power evaluation accuracy.

This motivated us to develop, relied on the PowerSC framework, a new multi-model power estimation engine that selects the macromodeling technique leading to the least estimation error for each system component and for each input-vector stream. A proper

selection function is built after component characterization and used during estimation. Given an input-vector stream stimulating an RTL component, the selection function chooses the most suitable macromodel for a system component according to the input stream's signal probability and transition density.

Therefore, this paper contributes to the evolution of the macromodeling paradigm towards widespread adoption, by providing clues on the safer estimation regions of each macromodel. Therefore, this work not only identifies the lack of robustness of each macromodel, but also provides proper grounds on how they should be combined.

The remainder of this paper is organized as follows. Section 2 reviews related work. The PowerSC framework is described in Section 3. Section 4 selects three distinct macromodels for experimentation and summarizes their key ideas. Section 5 provides evidences that macromodel-based estimation may incur unacceptably high errors and pinpoints the sources of inaccuracy by means of a few examples. Then, it is described how our multi-model engine works to improve accuracy. Section 6 shows experimental results comparing the accuracy obtained with different macromodel techniques. Our conclusions are drawn in Section 7 along with future work.

2 Related Work

Some early techniques correlated physical capacitance and circuit activity to power consumption. Circuit complexity was captured in terms of gate equivalent counts, which were combined with the activation frequency of each functional block [21]. Another early approach relied directly on the central limit theorem [22] to estimate switching activity. This approach consisted in applying randomly generated inputs to the circuit and then monitoring its activity. Simulation samples were taken until all nodes converged according to a well-defined stopping criterion.

Most of the techniques used in today's tools are based on the concept of *macromodel* [7, 23], which is an abstract model obtained by measuring the power consumption of existing implementations with the help of lower level methods. Essentially, the macromodeling techniques can be divided into three categories: *equation-based*, *table-based* and *hybrid*.

Most equation-based techniques make use of statistical methods for model building (e.g., regression analysis). For instance, least square estimation is employed in [10, 11] to derive the fitting coefficients. One advantage of statistical approaches is that they inherit a solid mathematical foundation and are therefore capable of providing a good level of confidence for the estimates, if adequate training sets are used. Obviously, the more an input stream is similar to the training streams, the higher the confidence level. Many other techniques belonging to this category can be found in the literature [12, 7, 8, 13, 14].

The first table-based technique was proposed in [5]. Basically, it consists of a three-dimensional table, where the axes represent input/output signal statistics, and a table entry represents a power value. The original technique was improved by including a new parameter in the model, resulting in a four-dimensional table [15]. Several other works belonging to this category were proposed in the literature [9, 16, 17].

The technique proposed in [3] lies in the hybrid category. It consists of a two-dimensional

table, which is indexed by transition density and signal probability values. For each entry in the table, an equation is built by means of linear regression.

A recent work [20] describes a multi-model approach for SoC power analysis aiming at improving power simulation performance. This approach is based upon multiple models at distinct levels of granularity. The coarser the granularity, the less parameters are needed for estimation, thereby reducing the estimate overhead. The finer the granularity, the higher the accuracy, but the higher the overhead. A model is dynamically chosen for a component according to its relative power contribution to the overall system power. In other words, if a component doesn't contribute much to the system consumption, a coarser model is selected since a higher error can be accomodated.

As opposed to that approach, we envisage to exploit multiple models to get better accuracy. Our motivation is based upon the fact that most macromodel assumptions lead to limitations. For instance, since the macromodel in [5] assumes that signal statistics are uniformely distribuited among inputs and outputs, the resulting input signal imbalance may result in significant errors as pointed out in [18]. Furthermore, the macromodel proposed in [3] restricts the training set to a single input stream per characterized point. However, as shown in [19], the choice of a proper set is crucial to obtain a high-quality model.

In brief, we propose an approach which takes a set of circuit components at the same level of abstraction and chooses the best candidate macromodel for each one, based on a prior macromodel evaluation, so as to reduce the estimation error.

Conventional design flows mix diffent tools for power evaluation at different levels of abstraction and often requires that the designer split his/her design into several pieces according to the abstraction level and execute the different tools one at a time [24, 25, 26].

As opposed to conventional power evaluation toolchains, our framework is based upon a unified design representation, regardless of abstraction level. Before elaborating the description of the selected macromodels for experimentation with our multi-model engine, the next section describes the PowerSC framework.

3 The PowerSC Framework

In this section, we first propose a design flow fully based on SystemC and then describe its supporting framework. To implement the macromodels described in Section 4 and the multi-model engine proposed in Section 5.3, we relied on PowerSC [27, 28, 29] as the implementation infrastructure.

3.1 The Proposed Design Flow

Figure 1 shows two complementary and orthogonal flows. At the right side, the usual purely-functional SystemC flow is sketched. At the left side, our power-aware flow is illustrated in more detail, since it summarizes the main steps deserving support in our framework.

No matter the adopted flow, the starting point is a SystemC description. The flow to be followed is determined by some configuration files (e.g. Makefiles), which instruct the C++ compiler to generate either a conventional executable specification (by linking the SystemC

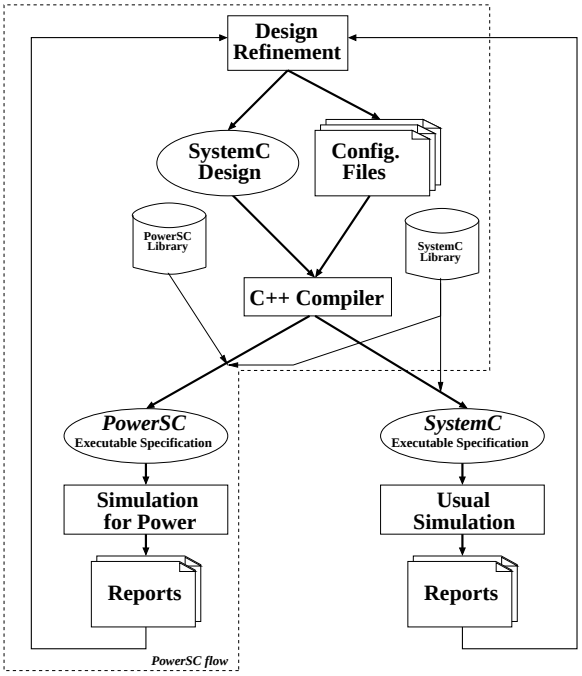


Figure 1: The PowerSC design flow

library only) or to produce an augmented executable specification (by linking both, the SystemC and PowerSC libraries).

The first step in our methodology is to compile the PowerSC executable specification, which is instrumented to gather signal statistics during simulation. In the next step, simulation is launched and, as consequence of proper instrumentation, design elements are monitored and power information is dynamically recorded. At simulation completion, the resulting information is summarized in power reports. Reported bottlenecks can be employed to prompt design refinements. Refined designs should undergo as many iterations through the PowerSC flow as required to reach adequate power values.

Notice that the user can switch from one flow to another as many times as needed to optimize power and satisfy design constraints.

Only two modifications are necessary in the SystemC description to enable PowerSC usage. A mandatory change is the inclusion of the main PowerSC header file (`powersc.h`) within the model files. Another compulsory change is to invoke a PowerSC macro at the end of the simulator's main function (`sc_main`), in order to print out the results. These modifications are shown in Figure 2, at lines 2 and 17, respectively.

As it can be seen, the effort to enable the power estimation mechanism is minimal, leaving a small footprint in the original SystemC description. It should be noted that SystemC data types, signals and modules don't need to be manually changed. PowerSC modifies them automatically.

```
1#include <systemc.h>
2#include <powersc.h> // <— mandatory modification —
3#include "muls32.h"
4...
5SC_MODULE(rtl_example) {
6    sc_in_clk clk;
7      ...
8    sc_signal<sc_uint<2> > sig1, sig2; // signals
9      ...
10   MulS32 *mult;
11     ...
12};
13...
14int sc_main(int argc, char **argv) {
15  ... // modules instantiation
16  sc_start( /* simulation time */ );
17  PSC_REPORT_POWER; // <— this is also necessary —
18  return( 0 );
19}
```

Figure 2: A PowerSC-enabled SystemC model

3.2 Power Characterization at the Gate-level

The highly accurate power figures obtained at the gate level are employed to build higher-level power models. First, we create a gate-level part library for a given technology library. The required generation steps are depicted in Figure 3.

The base part library (containing components like adders and multipliers) is built from a technology library in the so-called Liberty format with a tool from Forte's Cynthesizer package. Components are described in SystemC at the RT level (*SC_RTL*) and in Verilog at the gate-level (*V_GATES*).

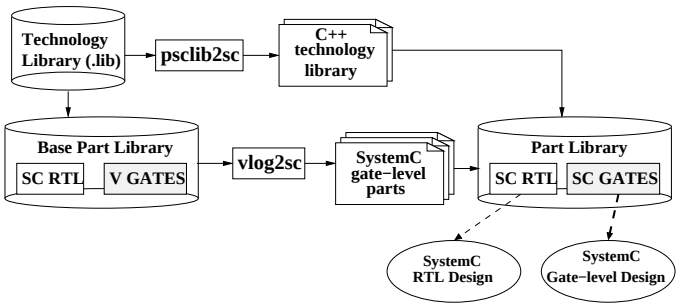


Figure 3: SystemC gate-level generation

Since we aim at a unified representation within our framework, we implemented two converters from the external formats into SystemC, as follows:

- the **vlog2sc** tool translates each netlist found in *V_GATES* to an equivalent SystemC gate-level description;
- the **psclib2sc** tool converts the technology library from the Liberty format into a C++ representation.

The files produced by the convertion tools result in the so-called SystemC gate-level part library (*SC_GATES*), which replaces the Verilog library.

The user can now select between RT or gate-level simulation by just selecting the part from the appropriate library: *SC_GATES* or *SC_RTL*. The power estimation is performed in the same way as it has been shown in Figure 2. Notice that, regardless of the abstraction level, the PowerSC aspects visible to the user are exactly the same.

3.3 Power Modeling and Estimation at the RT-level

One of the keystones of the PowerSC framework is the power modeling support at different levels of abstraction. This support is given by the PowerSC macromodeling API, which consists of a set of C++ classes for the integration of distinct modeling techniques.

In order to add a new part to the library, two classes must be derived: `psc_macromodel` and `psc_macromodel_parms`. The most important aspects of the former are illustrated in Figure 4.

```
1 class psc_macromodel {
2    ...
3    public:
4       virtual void init_power_map();
5       virtual double get_power( const psc_macromodel_parms & );
6    ...
7 };
```

Figure 4: PowerSC macromodeling support

The internal details are hidden from the user, who must only create the code for the following virtual functions, used internally by PowerSC to evaluate power:

- `init_power_map`: this function initializes the internal structure (defined by the user) with the power information from the characterization phase.
- `get_power`: this function has the details on how to compute the power dissipated based on its `psc_macromodel_parms` parameter. For instance, the attributes of the derived `psc_macromodel_parms` class could be the signal statistics. PowerSC internally calls this function in order to generate the power reports from a specific power model.

A distinct `psc_macromodel` is automatically created for each library component since the macromodeling techniques usually require a specific behavior for each component (tables initialized with different values, different power equations and so on). These distinct `psc_macromodel` are also usefull when the designer wants to use different macromodel methodology for each component.

In order to show how effective is this support, we implemented three different RTL macromodeling techniques through this API, as described in the next section.

4 Macromodeling Background

Without loss of generality, we selected three well-known macromodeling techniques as candidates for experimental analysis. From now on, they will be referred to as *3DTab* [5],

EqTab [3] and *e-HD* [8]. This section summarizes some background on the selected techniques, such as key estimation concepts and main characterization steps, as a base for the discussion in Section 5.

4.1 Model 1: Three-Dimensional Table

The *3DTab* macromodel relies on the following signal properties: the input signal probability P_{in}, the input transition density D_{in} and the output transition density D_{out}. The model is represented by a 3D look-up table such that to each valid triple (P_{in}, D_{in}, D_{out}) corresponds a power value.

Power estimation consists in first running an RTL simulation to collect the signal statistics P_{in}, D_{in} and D_{out} and then looking up in the table for the corresponding power value. When the collected signal statistics don't directly correspond to a valid triple, interpolation is used to return the closest value.

Component characterization consists of the following sequence of steps. First, for each valid pair (P_{in},D_{in}), several input streams are generated. Then, every distinct stream is applied to a lower-level model of the component (e.g. a gate-level model), the consequent D_{out} is evaluated and the resulting power consumption is determined. Next, the average of all power values with same (P_{in}, D_{in}, D_{out}) is stored at the proper table entry.

4.2 Model 2: Equation Coefficient Table

Instead of relying on the overall average transition input/output densities (like *3DTab*), the *EqTab* model considers the individual contribution of each input/output bit position. Let $D_{in}(x)$ and $D_{out}(x)$ be transition densities measured at the x-th bit position for a stream of input and output vectors, respectively. For simplicity, let's generically call them *bit-wise transition densities*. Let n and m be, respectively, input and output vector bit widths. Given a component, its power consumption is modeled by the following equation, where c_i denotes a coefficient:

$$Power = c_0 + c_1 * D_{in}(0) + c_2 * D_{in}(1) + ... + c_{n+m-1} * D_{out}(m-2) + c_{n+m} * D_{out}(m-1).$$

Actually, the *EqTab* technique relies on a look-up table which is indexed with (P_{in}, D_{in}). For each entry in this table, instead of directly storing a power value, the corresponding entry actually stores the coefficients of the equation above.

As a result, *EqTab* estimation consists of three steps: first, an RTL simulation is run and the bit-wise densities are collected, along with the properties P_{in}, D_{in} and D_{out}; then, the coefficients stored at entry (P_{in}, D_{in}) are returned (if (P_{in}, D_{in}) doesn't represent a valid entry, the closest valid point is used instead); finally, the returned coefficients and the collected bit-wise densities are combined according to the above equation.

As opposed to *3DTab*, *EqTab* characterization employs a single input vector stream for each pair (P_{in}, D_{in}) and consists in determining the respective set of coefficients. To find a proper set of coefficients, a system of equations is built as follows.

Let SW be a matrix with as many rows as the number of successive pairs of vectors in the stream (say, S pairs) and with as many columns as the compound vector bitwidth (n+m). A row of matrix SW stores the bit-wise transition densities taken between a pair

of successive vectors. A column stores the transition density of a given bit position along the input stream. Let P be a S x 1 matrix, where each entry p_i stores the power consumed by the i-th pair of input vectors.

Characterization consists in first calculating the bit-wise transition densities for every successive pair of input vectors (storing them in a row of matrix SW) and measuring their resulting power consumption (storing it in an entry of matrix P). Then, the set of fitting coefficients C is obtained by solving the system of equations $SW * C = P$ with standard regression techniques (e.g., least mean squares). Finally, such coefficients are stored at entry (P_{in}, D_{in}).

4.3 Model 3: Enhanced Hamming Distance

Basically, the *e-HD* macromodel is an equation which expresses power as a function of two distinct input signal properties: the Hamming distance and the number of stable bits between two successive input vectors (the technique doesn't employ output signals). Given two input vectors u and v with n bits each, the Hamming distance (h) and the number of stable bits (s) with value '1' are defined, respectively, as:

$h(u, v) = |\{i|(u_i \neq v_i)\}|$, for $1 \leq i \leq n$; and

$s(u, v) = |\{i|(u_i = v_i = 1)\}|$, for $1 \leq i \leq n$.

As opposed to the previous techniques, the *e-HD* macromodel calculates the power per cycle, as follows.

Let $E_{h,s}$ be a switching event class representing the properties of a pair of vectors, where h is their Hamming distance and s is their number of stable bits. Let $Power(c)$ be the power consumption at the c-th simulation cycle and n be the number of input bits. The *e-HD* macromodel equation is defined as follows, where $p_{h,s}$ denotes the contribution of switching event $E_{h,s}$ to the power consumption and $\delta_{h,s}$ denotes whether or not such event occurs in cycle c.

$$Power(c) = \left[\begin{array}{ccc} p_{1,n-1} & \cdots & p_{n,0} \end{array} \right] \left[\begin{array}{c} \delta_{1,n-1} \\ \vdots \\ \delta_{n,0} \end{array} \right] \quad (1)$$

In [8], $\delta_{h,s}$ is called an *activator* and is defined as follows:

$$\delta_{h,s} = \begin{cases} 1, & \text{if event } E_{h,s} \text{ occurs in cycle } c; \\ 0, & \text{if event } E_{h,s} \text{ doesn't occurs in cycle } c. \end{cases}$$

Given an input stream, estimation starts by calculating $h(u, v)$ and $s(u, v)$ for each pair (u, v) of successive vectors. Then, the activators associated with events occuring at a given cycle c are set and Equation 1 returns the power consumption for that cycle. The overall power consumption is obtained by adding the contribution of all cycles.

Component characterization starts with the generation of random input streams. Then, for every stream, each pair (u, v) of its successive vectors is applied to the component and functions $h(u, v)$ and $s(u, v)$ are evaluated. The resulting power consumption is determined with a pre-existent lower-level power model of the component. Finally, for all streams with same (h, s), the average power $p_{h,s}$ is calculated.

5 The Multi-Model Engine

The quality of a macromodel is strongly dependent on the training set. As shown in [19], while some training sets lead to high-quality models (average errors around 6%), others result in unacceptably poor ones (average errors around 660%).

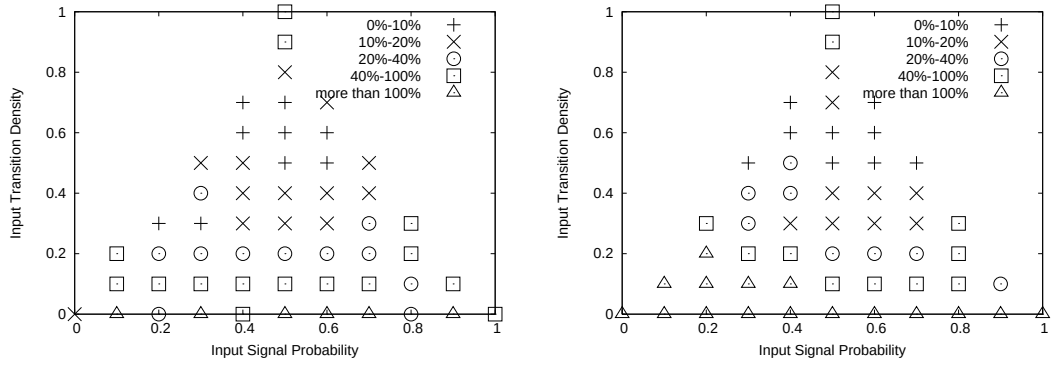
The sensibility to the training set and some inherent macromodeling assumptions (e.g. uniform probability distribution [5], lack of modeling for output signal properties [8]) lead to intrinsic limitations that degrade estimation accuracy.

This section first provides some preliminary evidences that macromodel-based estimation may incur unacceptably high errors and then pinpoints the sources of inaccuracy by means of a few examples so as to justify the claim that no macromodeling is robust enough to be employed alone. Then, we show how our engine can automatically handle multiple macromodels to improve the overall estimation accuracy.

5.1 Accuracy Divergence

To illustrate how the estimation accuracy can largely diverge among power models, we employ two real-life components as examples (*Add_ECLA32* and *MulS16*). Both examples were characterized for the three macromodeling techniques summarized in Section 4 (*3DTab*, *EqTab* and *e-HD*).

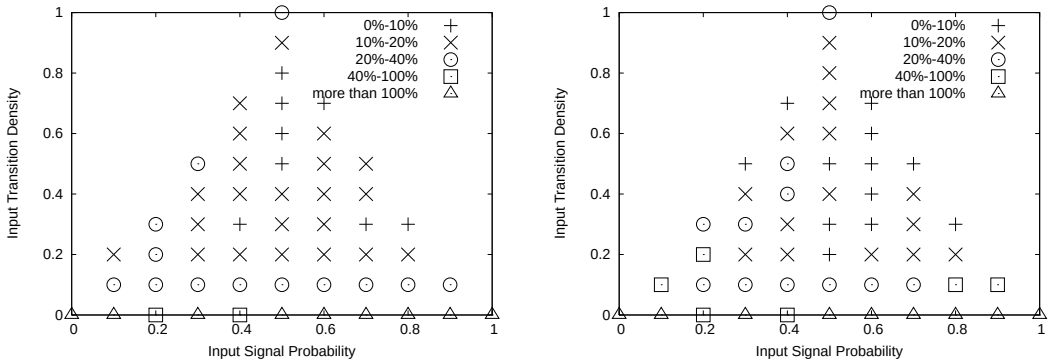
An experiment was conducted to assess the average errors of each macromodel as a function of two input-stream parameters, namely, P_{in} and D_{in} (we used *0.1* as the discretization step). A huge set of streams was generated to properly cover the whole P_{in} x D_{in} input space. Then, the RTL macromodel estimates were compared to the gate-level estimates, for each pair (P_{in},D_{in}). Figures 5, 6 and 7 show the error distribution on the P_{in} x D_{in} space for the examples, regarding the three adopted techniques.



(a) Add_ECLA32 (b) MulS16

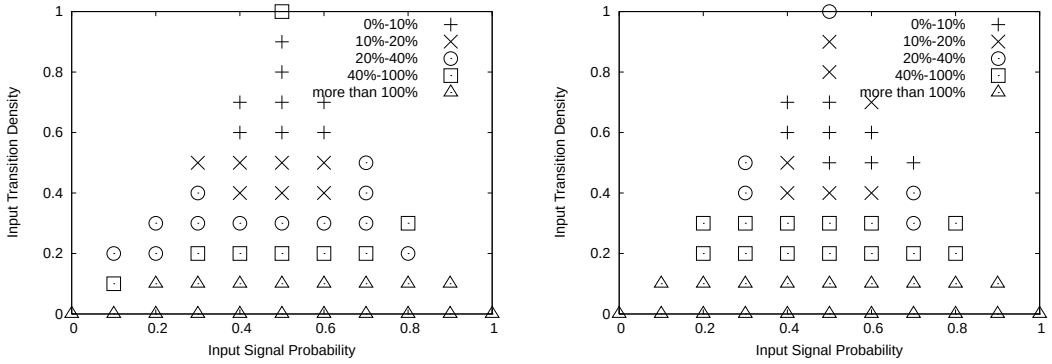
Figure 5: Error Distribution (3DTab)

Notice that, for a given component, the distinct macromodels lead to quite different average errors. For instance, regardless the chosen component, the *e-HD* macromodel leads to the highest average error for the input streams whose transition density is in the interval



(a) Add_ECLA32 (b) MulS16

Figure 6: Error Distribution (EqTab)



(a) Add_ECLA32 (b) MulS16

Figure 7: Error Distribution (e-HD)

[*0.0*,*0.1*]. This means that *3DTab* or *EqTab* would be a better choice for such streams. However, for the streams whose transition density is greater than *0.6*, *e-HD* exhibited higher accuracy.

5.2 Qualitative Analysis of Macromodeling Limitations

Each macromodel has its merits in capturing power variation. However, each technique implies a different usage of parameters, which eventually hides some assumption. This subsection shows that such implicit assumptions are the very source of limitation and that they are difficult to overcome within the scope of a single macromodeling technique.

Let's first consider the *3DTab* technique. It assumes that P_{in}, D_{in} and D_{out} are uniformly distributed through all input/output signals, although some signals could have a higher impact on the power consumption than others. Such assumption is clearly inadequate when irregularly structured circuits are addressed or when control signals are considered, since they may completely change the circuit's operational mode [9].

To illustrate that such assumption leads to a drawback, consider again the example *MulS16* (a 16-bit multiplier used as a component of the stereo audio crossover design to be discussed in Section 6). We simulated one of its implementations with *100* distinct input streams and monitored one of its operands (16 inputs). For most streams, we observed that only four of the monitored inputs exhibited transition densities higher than zero.

Let's now consider one of those simulation instances, whose overall input transition density is $D_{in} = 0.0912$ and whose bit-wise transition densities are the following:

$$(D_0, D_1, ..., D_{15}) = (0.49, 0.44, 0.50, 0, 0, ..., 0, 0.33).$$

Notice that only the last and the first three inputs actually switch for this stream. This means that part of the *MulS16* circuit is not stimulated, as opposed to the uniform distribution assumed by *3DTab*, which implies the following bit-wise transition densities:

$$(D_0, D_1, ..., D_{15}) = (0.0057, 0.0057, ..., 0.0057).$$

We can therefore conclude that, despite the same D_{in}, such rather different stimulation patterns are likely to lead to quite different power estimates.

EqTab overcomes this drawback by taking each input into account individually, which in principle should lead to better accuracy. However, as opposed to *3DTab*, a single stream is used in the characterization process for each entry in the table. Since the number of possible streams grows exponentially with the input width for some chosen input statistics, this assumption represents a serious limiting factor, as illustrated by the following example.

Let A and B be 4-bit input operands of a Booth multiplier [30]. Consider a candidate characterization stream whose overall transition density is $D_{in} = 0.25$ and whose bit-wise transition densities are:

$$(A_0, ..., A_3, B_0, ..., B_3) = (0.5, 0.5, 0.5, 0.0, 0.0, 0.5, 0.0, 0.0).$$

where the first four elements refer to A and the last four to B. Now, consider an alternative characterization stream obtained by swapping the operands A and B. Although the overall transition density remains the same, the resulting bit-wise transition densities are:

$$(A_0, ..., A_3, B_0, ..., B_3) = (0.0, 0.5, 0.0, 0.0, 0.5, 0.5, 0.5, 0.0).$$

Since the multiplication algorithm is likely to require drastically different amounts of computational effort, depending whether the operand is the *multiplicand* (the value to be added up) or the *multiplier* (the number of times the multiplicand must be added), it is clear that the power behavior for these two characterization streams will be quite different, although only one of them would be captured in the macromodel.

In the *e-HD* technique, although two distinct pairs of vectors with same input signal statistics will probably result on distinct output signal statistics, they are modeled in exactly the same way, which is based on input information only.

Let's illustrate the impact of such assumption by means of the following example. We performed the characterization of the *Add_ECLA32* component and monitored each pair (u, v) of successive vectors within the input characterization streams (62 streams with 2000

vectors each). We observed that a collection of 505 pairs had exactly the same input statistics: $h(u, v) = 15$ and $s(u, v) = 49$. For such collection, the resulting power estimates lied in the range [100μW, 1000μW] with a mean value of 482.2μW and a standard deviation of 213.2μW. As compared to gate-level reference estimates available for that component, the estimation error lies in the range [-52%, 300%]. This means that the *e-HD* technique may incur high estimation errors because different circuit behaviors cannot be distinguished by the same input signal statistics. If output statistics were included in the model, such distinct behaviors would be better captured.

Since the drawbacks of each method were pointed out qualitatively, we introduce our multi-model engine in the next subsection.

5.3 The proposed multi-model approach

Our multi-model engine is tightly connected to the characterization engine, which in turn relies on a sequence generator, as depicted in Figure 8.

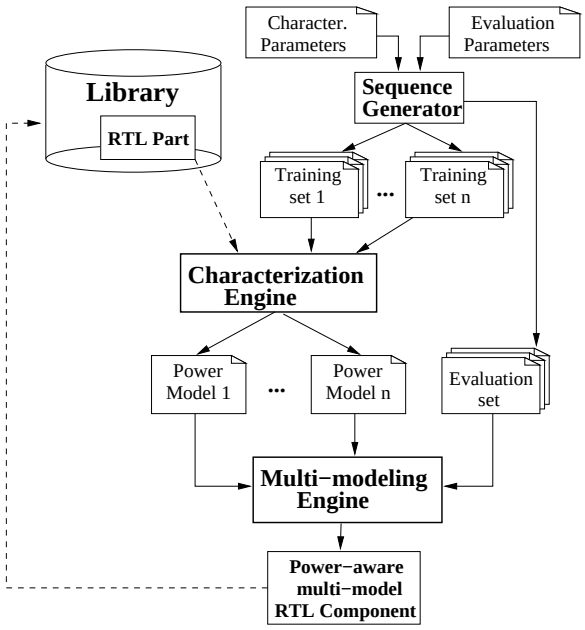


Figure 8: The multi-model engine selection flow

At the top of the figure, lies the sequence generator, which produces two types of streams: *training sets* to allow characterization and *evaluation sets* to instrument macromodel selection. Each type of sequence is produced according to distinct pre-specified parameters.

Basically, the sequence generator produces as many training sets as the number of components undergoing characterization, although some may be compliant with more than a component and may be reused. Characterization employs those training sets to produce different macromodels for each RTL component from the library.

Once all macromodels are generated for a given RTL component, they are evaluated by

the multi-model engine. Evaluation consists on first launching an RTL simulation of the component, which is stimulated by the evaluation set. As a result, distinct power estimates are obtained for each macromodel. Then, such estimates are compared to pre-existing gate-level power estimates and an error value is computed.

Based upon the computed error, a selection function is built to map each point of the input stream space to the macromodel leading to least error, as follows.

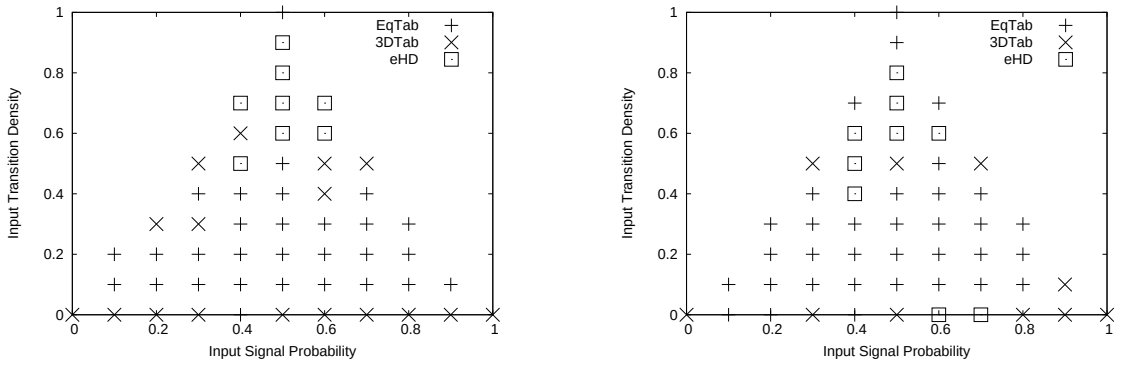
Let M be the set of macromodels and let e_i be the error computed for a given $i \in M$. The selection function $\zeta : P_{in} \times D_{in} \rightarrow m$ represents the mapping of a pair (P_{in},D_{in}) to a macromodel m such that $e_m = min\{e_i\}, \forall i \in M$.

Since, the function ζ is highly dependent on a proper choice of input streams, the evaluation set is designed as a huge collection of input streams (approximately 2200 in the current implementation) uniformely distributed over the P_{in} x D_{in} space.

In brief, our multi-model engine associates a distinct function ζ to each library component. Given an input stream, its properties (P_{in},D_{in}) are employed to query the function ζ, which selects the macromodel returning the more accurate power estimate.

Figures 9(a) and 9(b) display the ζ-functions for modules *Add_ECLA32* and *MulS16*. Each symbol represents the most accurate macromodel for a given point of the input space. Although their ζ-functions are different, a common pattern can be observed in both examples. The *e-HD* macromodel is more accurate than *3DTab* and *EqTab* for higher transition densities, while the macromodel *3DTab* is more accurate for lower transition densities.

As an example, for the point (P_{in}, D_{in}) = (0.5, 0.9), it is better to use the *e-HD* model to the *ADD_ECLA32* since it has an expected average error ranging from 0% to 10%, while for EqTab and 3DTab, this error lies within 10%–20% and 40%–100%, respectively. Now, if we consider the region around (0.3, 0.3), the *3DTab* should be used, since its expected errors are lower (10%–20%) than the other models.



(a) Add_ECLA32 (b) MulS16

Figure 9: Multi-model for both modules

The accuracy improvement obtained with our multi-model engine is reported in the next section.

6 Experimental Results

This section compares the estimation accuracy of conventional single-model techniques to the proposed multi-model approach. Reference gate-level estimates were obtained with Synopsys PrimePower.

The adopted benchmark suite consists of six designs. Two are single components from a part library (*Add_ECLA32* and *MulS16*). Four are complex designs: an implementation of a differential equation algorithm (*Diffeq*) and implementations of a stereo audio crossover (*Crossover1*, *Crossover2*, *Crossover3*). These complex designs consists of several different components with varying bit-widths, such as adders, multipliers and subtracters. Several estimations were conducted for each design, each with different input streams.

6.1 Common Characterization Set-up

To properly assess the distinct macromodeling techniques, the same vector generation procedure was used during characterization. We adopted the procedure described in [18], which allows input stream generation with high-accurate signal probability, transition density and spatial correlation. As suggested in [18], we set up each stream with 2000 vectors to guarantee accurate statistics. The adopted input-space range discretization was (0.00, 0.05, 0.15, ..., 0.95, 1.0) for both P_{in} and D_{in}.

6.2 Results

The results are summarized in Table 1. The first column specifies the distinct design examples, while columns 2 to 4 show the results obtained with the three adopted macromodeling techniques (*3DTab*, *EqTab* and *e-HD*). The last column shows the results for our multi-model approach.

	3DTab	**EqTab**	**e-HD**	**Multi-model**
Benchmark	**min/max/avg**	**min/max/avg**	**min/max/avg**	**min/max/avg**
Add_ECLA32	0.5/34.1/12.9	0.5/33.1/15	4.7/71.6/24.9	0.5/33.1/8.5
MulS16	2.4/83.4/24.4	1.5/42.4/12.6	0.57/225.3/57.1	0.57/42.4/8.6
Diffeq	1.7/89.1/26	0.93/54.8/16	1.8/433.6/86.9	0.3/53/12.4
Crossover1	3.4/64.4/21.2	2.2/45.7/15.9	1.6/184/47.6	0.4/33/9.7
Crossover2	2.9/65.6/21.7	1/41.9/12.6	2.5/269.7/61.3	0.2/33.68/6.7
Crossover3	2.4/58.2/20.5	1.5/37/13.4	2.4/251/61.9	0.3/28.4/8.8

Table 1: Results comparing the single-model versus the multi-model approach

Results are expressed as minimum, maximum and average errors, computed as follows:

$$\varepsilon = \frac{1}{n} \sum_{i=1}^{n} \frac{|P^i_{est} - P^i_{ref}|}{P^i_{ref}} \times 100\% \tag{2}$$

where n is the number of components in the design, P^i_{est} and P^i_{ref} are the estimated and reference power values for the i-th component, respectively.

Note that, the single-model approach may lead to unacceptable errors. For instance, errors higher than 100% were obtained with *e-HD*. This doesn't mean that such technique should be discarded beforehand, but show that for some regions in the $P_{in} \times D_{in}$ space, it is not adequate. However, as we have shown in Section 5, there are regions where the *e-HD* peformed much better than the others techniques.

In order to correlate the overall system accuracy with the estimation errors of a component, let's focus on the more complex designs (Diffeq, Crossover 1, ...Crossover 3). Those designs employ *MulS16* as a component that dominates the total power consumption. Table 2

Inst.	**3DTab**	**EqTab**	**e-HD**
#1	6.14%	**1.50%**	12.73%
#2	**6.56%**	11.72%	9.66%
#3	2.42%	**1.96%**	3.69%
#4	83.53%	**21.14%**	225.38%
#5	13.68%	8.21%	**1.44%**

Table 2: MulS16 robustness example

shows the estimation errors for distinct instances of *MulS16*. The large variation in the error value is an evidence of the lack of robustness of a single macromodel. On the one hand, due to such lack of robustness in the estimation of a dominant component, a single macromodel is bound to compromise the overall accuracy. On the other hand, a multi-model approach overcomes this lack of robustness, since the technique leading to the least error is selected (as shown in bold in Table 2).

7 Conclusions

This paper presented a pragmatic SystemC-based framework whose ultimate goal is to enable power modeling and estimation at multiple levels of abstraction. Besides, its API allows easy integration of distinct modeling techniques. All this capabilities can be obtained at the expense of minor SystemC description footprint and the availability of a standard C++ compiler.

In addition, this paper pinpointed the implicit assumptions of three popular macromodeling techniques and assessed their impact on power estimation accuracy. The qualitative analysis in Section 5.2 ties each implicit assumption to the resulting macromodel limitation. Since it relies on a consistent set of real-life designs, the quantitative analysis in Section 6 allows us to claim that no macromodel is robust enough to be employed alone.

In order to overcome these limitations, a systematic method for a multi-modeling approach for power estimation was presented, which main goal is to improve the accuracy of the estimations and to guarantee the robustness within the power estimation process. First, three different macromodel techniques were evaluated and, for each (P_{in}, D_{in}) pair, the best macromodel was selected. This task was repeated for the whole component library, creating an evaluation function for each component. Using this result, some benchmark circuits were

evaluated and, in every case, we got better results by using the multi-model approach than if we used only one of the base macromodels.

We believe that the improvement on model robustness is a prerequisite for a wider adoption of the macromodeling paradigm at the RT level.

As future work, we plan to extend our power model evaluation procedure, studying the impact of other parameters in the evaluation function, as well as improving the training set generation for the evaluation function modeling.

References

[1] OSCI. *SystemC 2.0 User's Guide*, 2.0 edition, 2002.

[2] Enrico Macii and Massimo Poncino. Power macro-models for high-level power estimation. In *Low-Power Electronics Design*, chapter 39. CRC Press, 2005.

[3] M. Anton, I. Colonescu, E. Macii, and M. Poncino. Fast characterization of RTL power macromodels. In *IEEE Proc. of ICECS*, pages 1591–1594, 2001.

[4] Subodh Gupta and Farid N. Najm. Energy and peak-current per-cycle estimation at RTL. *IEEE Trans. Very Large Scale Integr. Syst.*, 11(4):525–537, 2003.

[5] Subodh Gupta and Farid N. Najm. Power macromodeling for high level power estimation. In *Proc. of DAC*, pages 365–370, 1997.

[6] A. Raghunathan, S. Dey, and N. K. Jha. Register-transfer level estimation techniques for switching activity and power consumption. In *Proc. of ICCAD*, pages 158–165, 1996.

[7] P. E. Landman and J. M. Rabaey. Activity-sensitive architectural power analysis. In *IEEE Trans. on Computer-Aided Design of Integrated Circuits*, pages 571–587, June 1996.

[8] Gerd Jochens, Lars Kruse, Eike Schmidt, and Wolfgang Nebel. A new parameterizable power macro-model for datapath components. In *Proc. of DATE*, 1999.

[9] R. Corgnati, E. Macii, and M. Poncino. Clustered table-based macromodels for rtl power estimation. In *GLSVLSI '99: Proceedings of the 9th Great Lakes symposium on VLSI*, pages 354–357, 1999.

[10] A. Bogliolo and L. Benini. Robust rtl power macromodels. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 6(4):578–581, December 1998.

[11] Luca Benini, Alessandro Bogliolo, Enrico Macii, Massimo Poncino, and Mihai Surmei. Regression-based rtl power models for controllers. In *Proceedings of the 10th Great Lakes symposium on VLSI*, pages 147–152, 2000.

[12] Subodh Gupta and Farid Najm. Analytical models for rtl power estimation of combinational and sequential circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(7):808–814, July 2000.

[13] Gerd Jochens, Lars Kruse, Eike Schmidt, Ansgar Stammermann, and Wolfgang Nebel. Power macro-modelling for firm-macro. In *PATMOS-00*, pages 24–35, September 2000.

[14] Maurizio Bruno, Alberto Macii, and Massimo Poncino. A statistical power model for non-synthetic rtl operators. In *International Workshop on Power And Timing Modeling, Optimization and Simulation*, pages 208–218, 2003.

[15] Subodh Gupta and Farid N. Najm. Power modeling for high-level power estimation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8(1):18–29, February 2000.

[16] H. Mehta, R. M. Owens, and M. J. Irwin. Energy characterization based on clustering. In *Proceedings of DAC'96*, pages 702–707, 1996.

[17] M. Barocci, L. Benini, A. Bogliolo, B. Ricco, and G. De Micheli. Lookup table power macro-models for behavioral library components. In *IEEE Alessandro Volta Memorial Workshop on Low-Power Design*, pages 173–181, March 1999.

[18] Xun Liu and Marios C. Papaefthymiou. A markov chain sequence generator for power macromodeling. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(7):1048–1062, July 2004.

[19] Q. Wu, C. Ding, C. Hsieh, and M. Pedram. Statistical design of macro-models for rt-level power evaluation. In *Proc. of ASPDAC*, 1997.

[20] Nikhil Bansal, Kanishka Lahiri, Anand Raghunathan, and Srimat T. Chakradhar. Power monitors: A framework for system-level power estimation using heterogeneous power models. In *VLSID '05: Proceedings of the 18th International Conference on VLSI Design*, pages 579–585, Los Alamitos, CA, USA, 2005.

[21] K. D. Müller-Glaser, K. Hirsch, and K. Neusinger. Estimating essential design characteristics to support project planning for ASIC design management. In *Proc of ICCAD*, pages 148–151, November 1991.

[22] M. G. Xakellis and F. N. Najm. Statistical estimation of the switching activity in digital circuits. In *Proc. of DAC*, pages 728–733, 1994.

[23] S. R. Powell and P. M. Chau. Estimating power dissipation of vlsi signal processing chips: The pfa technique. *VLSI Signal Processing IV*, 1990.

[24] Synopsys Inc. *Power Compiler User Guide*, x-2005.09 edition, December 2005.

[25] Wolfgang Nebel and Domenik Helms. High-level power estimation and analysis. In *Low-Power Electronics Design*, chapter 38. CRC Press, 2005.

[26] Ansgar Stammermann, Lars Kruse, Wolfgang Nebel, Alexander Pratsch, Eike Schmidt, Milan Schulte, and Arne Schulz. System level optimization and design space exploration for low power. In *Proc. of the 14th international symposium on Systems synthesis*, pages 142–146, 2001.

[27] F. Klein, R. Azevedo, and G. Araujo. Enabling High-Level Switching Activity Estimation using SystemC. Technical Report IC-05-17, Institute of Computing, UNICAMP, August 2005.

[28] F. Klein, R. Azevedo, and G. Araujo. High-level switching activity prediction through sampled monitored simulation. In *Proceedings of the International Symposyum on System-on-Chip 2005*, November 2005.

[29] Felipe V. Klein. PowerSC: a SystemC Extension Aiming at the Gathering of Switching Activity. Master's thesis, Institute of Computing, State University of Campinas, April 2005. note: in portuguese.

[30] David A. Patterson and John L. Hennessy. *Computer Organization and Design: The Hardware/Software Interface*. Morgan Kaufmann, third edition, 2004.