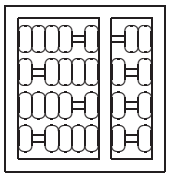

INSTITUTO DE COMPUTAÇÃO
UNIVERSIDADE ESTADUAL DE CAMPINAS

Enabling High-Level Switching Activity Estimation using SystemC

F. Klein *R. Azevedo* *G. Araujo*

Technical Report - IC-05-17 - Relatório Técnico

August - 2005 - Agosto

Enabling High-Level Switching Activity Estimation using SystemC

Felipe Klein
klein@ic.unicamp.br

Rodolfo Azevedo
rodolfo@ic.unicamp.br

Guido Araujo
guido@ic.unicamp.br

Abstract

This paper presents an alternative methodology to a traditional high-level power estimation flow, that enables the fast gathering of switching activity from SystemC RTL descriptions. The proposed methodology requires minimum effort from the designer, while reducing the steps necessary to obtain switching activity information, and requires only a C++ compiler. The resulting activity is very close to the results obtained using a commercial tool, which has a larger flow with several steps. We also show that our approach overcomes some drawbacks in the commercial tool.

1 Introduction

Nowadays, the demand for portable devices, such as PDAs, cell phones, digital cameras and other battery-powered devices is growing fast. Each new generation of these products includes more functionalities, which impact on their complexity, and thus the power dissipated by them. There are several implications from the increasing of power consumption, ranging from the reduction of the battery's lifetime to the higher cost per chip, resulting from elaborated cooling and packaging solutions to avoid circuit failure. Thus, the design of integrated circuits aiming at the reduction of power consumption has become an important role in the design flow – the so-called low power design.

For the lower levels (gate and transistor-level) of the design process, there is a vast set of CAD tools, e.g. SPICE [1], to assist the designer in estimating power. However, getting power figures at these levels of abstraction can be very time-consuming, as these tools are located late in the design flow, resulting in a high turnaround cost.

Designers need a way to estimate power at higher levels of abstraction. At the early stages of the design flow there are much more oportunities for power optimization, but fewer information than at the lower levels. Nevertheless, even a rough estimate can save a lot of time, because the focus of optimization can be detected earlier. It is well known that the absolute accuracy at the lower levels is better than at the higher levels, but what is really important at higher levels is the relative accuracy and the speed to get the estimation, so we can compare two or more design alternatives, thus allowing tradeoffs earlier in the design flow.

In this paper we propose a methology to gather transition count of SystemC [2] RTL descriptions. We present the PowerSC, an extension of SystemC, that was implemented for

the purpose of high-level switching activity estimation. Our main focus is on the switching activity of the design, which is still the most important factor in the power consumption budget. We show that PowerSC provides a transparent and easy to use methodology, requiring the minimum effort from the designer, and using no other apparatus, just the SystemC itself and a C++ compiler. The experiments demonstrate that it is feasible to gather switching activity by using our approach, which results are very close to the values from a commercial tool, with fewer tools and steps in the flow. This gathered information can then be used to estimate the power consumed by the model with traditional techniques. We also show that PowerSC overcomes some drawbacks in the commercial tool.

This paper is organized as follows: In Section 2, a brief discussion of some of the existing techniques for high-level power estimation is given. Section 3 shows some methods used to capture switching activity. Section 4 introduces our proposed methodology to enable the gathering of switching activity of SystemC designs, while Section 5 reports some experimental results obtained using the proposed methodology. Finally, Section 6 concludes this paper and presents some future work.

2 Previous Work

Several approaches to make power estimation at the architetural or register-transfer level are available in the literature. Early techniques associated physical capacitance and activity of the circuit to power consumption. In [3], they relate the complexity of a circuit to gate equivalent counts. A reference gate is chosen from a technology library, where the average power dissipated by this gate is pre-calculated. Then, to estimate the power consumed by a functional block, the number of gate equivalents to implement the block is extracted from the library and multiplied by the average power dissipated by the gate. One drawback is that only one gate reference is used to the calculations. Another one is the weak model of circuit activity, which is usually fed by the user, and assumed as a fixed value through the entire circuit.

There are also techniques based on *macromodeling*, where measurements are made in existing implementations to produce a power model. These techniques are well-suited for library-based approaches. The first work on macromodeling was the PFA (Power Approximation Factor) technique [4]. In this technique, each functional block is characterized by a hardware complexity defined by the designer, and also by a PFA constant, empirically extracted from previous designs. This technique can be used to characterize an entire library of RTL blocks. One problem in [4] is that the data activity is not taken into account during the estimation. Thus, there is an implicit assumption that the inputs of a block do not affect its internal activity.

One macromodeling technique which takes into account the data activity is [5]. During simulation, the activity in the controlpath and in the datapath is monitored and then this activity information is used to fed power models that explicitly account for activity. A so-called equation-table method is proposed in [6] to reduce the time required during the characterization phase. The work in [7] presents a macromodeling technique where power is modeled and estimated on a energy-per-cycle basis. In [8] is proposed another

macromodeling approach, where power is also related to the switching of the bit- and word-level data. The characterization step in that work is made using lower level estimators, at gate or transistor level.

In [9], additionally to the mentioned techniques, they create models to estimate glitching activity, and also they analyze the impact of gliching activity on power consumption, which is ignored in most previous architectural techniques. A technique which is based on the input transitions is found in [10], where a table containing the power dissipated for each input transition is used. They present an energy simulator for a processor that uses this technique for power estimation. A methodology for automatically building energy models for NoC (Network-On-Chip) is introduced in [11]. These generated models are then linked to a SystemC simulator to perform the power analysis of the system.

3 Capturing Switching Activity

There are two main file formats that handle switching activity (toggle) information: VCD and SAIF. The VCD (Value Change Dump) file is generated over the simulation and provides toggle rates of the signals in the design. The drawback of VCD is that for high simulation times the file generated can be huge in terms of bytes. The SAIF (Switching Activity Interchange Format), was created by Synopsys to standardize a power format, is much more compact than the VCD, and it doesn't grow in size over the simulation. The format is mainly composed of the toggle count and the state probability. In contrast to VCD, the information contained in the SAIF is ready to be annotated in the design.

3.1 CAD Tool Flow for Power

One example of a commercial tool is the Synopsys Power Compiler [12], which is used for power analysis and optimization of circuits at both RTL and gate level. The left side from the dashed line in figure 1 shows the flow used for power consumption analysis of SystemC RTL designs using Power Compiler.

The process for Power Compiler shown in the figure is as follows: first, the SystemC RTL design is converted to a Verilog or VHDL model *(i)*, it then creates a technology independent representation of the design. Next, a *forward-annotation SAIF* file is generated from the design *(ii)*, with the elements to be monitored over the simulation. An RTL simulation is then executed using this file *(iii)*, and as its output, another file is generated, the *back-annotation SAIF (iv)*, which is used by the Power Compiler together with information extracted from the technology library *(v)*, to compute the average power and report the results *(vi)*. Synopsys provides an interface that allows the simulator to directly read and write the SAIF forward- and back-annotation files. Many simulators are supported, for example, the ModelSim, which can be linked to this interface to generate the back-annotation file. But this interface supports only Verilog or VHDL, and for that reason the first step of the figure is necessary. Refer to [12] for additional information about the Power Compiler.

This usual flow used by the Power Compiler requires several tools from different vendors, and many steps, which can turn the power estimation process a cumbersome task.

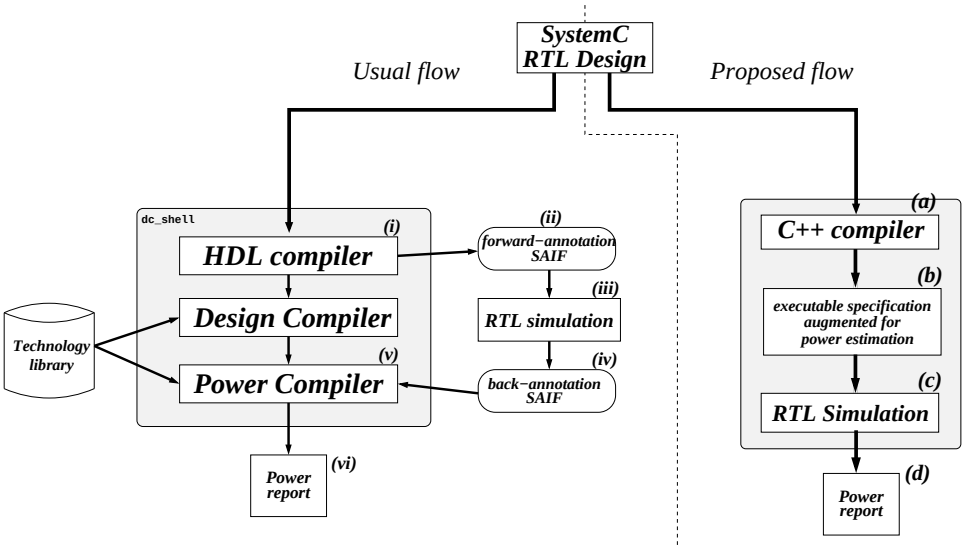

Figure 1: Two different flows for power

Alternatively to this, we propose the flow on the right side from the dashed line in figure 1. This flow is simpler than the previous one, as it requires fewer steps, only a standard C++ compiler, which generates an executable specification of the design *(a)*, that is augmented to gather switching activity. Next, the augmented specification is executed *(b)*, running an RTL simulation *(c)*, where the switching activity is gathered. After the simulation, a detailed report is produced *(d)*, at the end of the flow. This process will explained in the following sections. Both flows monitor the switching activity in the synthesis-invariant elements of the design, for example, primary inputs, sequential elements, black boxes and hierarchical ports.

4 PowerSC

In this section we introduce the PowerSC library and the related methodology proposed to enable switching activity estimation of SystemC descriptions, as well as some details on how it has been implemented. Our library is an extension of the SystemC language, which is a C++ class library developed to support the effective modeling of systems, with data types for describing hardware, concurrent behavior and also with event-driven simulation support. The main goal of PowerSC is to provide an easy way to gather transition activity from SystemC RTL descriptions, in order to reduce the steps necessary to do so. This gathered information can then be used to estimate the power consumed by the model with techniques similar to those presented in Section 2. We show that, by using PowerSC, it is possible to get the transition activity fastly, transparently, and using as tools only the SystemC and a standard C++ compiler.

We illustrate how the PowerSC can be used to augment a SystemC model through the analysis of a toy example, which is shown in figure 2.

```
1   #include <systemc.h>
2           ...
3   SC_MODULE(example) {
4      sc_in<bool> clk;
5      sc_in<sc_int<32> > in;
6      sc_in<sc_int<32> > out;
7           ...
8      sc_signal<sc_uint<4> > state;
9      sc_uint<32> result;
10          ...
11     void proc() {
12        result = in.read() * 2;
13        out = 42; // default value
14        if ( whatever condition )
15           out = in.read() + 1;
16        }
17          ...
18     SC_CTOR(example) {
19        SC_METHOD(proc);
20        sensitive_pos << clk;
21     }
22  };
23          ...
24  int sc_main(int argc, char **argv) {
25          ... // module instantiation
26     sc_start( simulation time );
27     return( 0 );
28  }
```

Figure 2: An example model

4.1 The Library

PowerSC enables the designer to capture switching activity from SystemC RTL descriptions. This was done as follows: each one of the relevant classes (data types, signals, ports, etc) were specialized, in order to monitor their values within the simulation, and to enable the printout of a detailed report with all the information gathered during simulation.

The library is composed mainly of three sets of classes, namely *power object*, *power object database* and *SystemC augmented* classes. The first set (*power object*) consists of classes which contain general properties and behavior regarding the gathering of switching activity. These objects hold information about its total captured transition count, state probability, and other important data. Moreover, they have the capability of automatically update this information, whenever a change occurs during simulation. The *power object database*'s classes act as a switching activity repository, that is, they are responsible for the management and storage of the information gathered during simulation by the objects from the previous set of classes.

Finally, in the third set are the relevant classes to the SystemC model itself. Using multiple inheritance, the classes from SystemC were augmented, inheriting the properties and behavior from both the *power object* and SystemC classes. To exemplify, consider the variable `result` from the model in figure 2 (line 9), declared as of type `sc_uint<32>`, that is, a 32-bit unsigned integer. The equivalent type in PowerSC is `psc_uint<32>`. In figure 3 is shown how these classes work together, using as example, the assignment to `result` in process `proc()`. We overloaded the base classes' operators, in order to enable the monitoring of the operations performed in the objects from these classes, like sum, assignments, and so on. The overloading of the assignment operator in the example of figure 3 makes it possible to capture the assignment to the variable `result`, and to update the switching activity

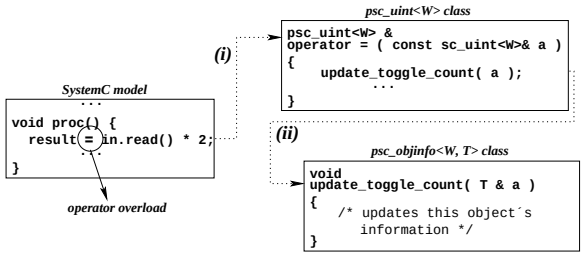

Figure 3: Overloading operators to gather switching activity

information resulting from the changing of the value. Given that the values of the elements can be changed only through a call to the overload operator, it is possible to gather both the input and output toggles. In the SystemC model, when the assignment of "`in.read() * 2`" to `result` is reached, a call to our overloaded assignment operator is made 3*(i)*, and before the value of `result` is updated, we compute how many bits will toggle after the assignment. This is done through a call to the `update_toggle_count` method 3(ii), which is a method inherited from the `psc_objinfo<W, T>` class (a *power object*), where W is the bit-width of the object, and T is the augmented SystemC class, in this case, `sc_uint<W>`.

It must be said that the objects don't need to be declared as PowerSC objects. To avoid the rewriting of the SystemC model, we used some C language tricks, as explained later in this section. An important characteristic to mention is that we take into account the delta cycles during simulation. To illustrate this, consider again the model example in figure 2. Assume that at some time during simulation we enter process `proc`, and the `if` condition (line 14) evaluates to **`true`**. Therefore, `out` will be assigned twice, but our implementation considers only the last assignment, and the update of the captured information is made with respect to the `out`'s value at the previous delta cycle. If this concept is ignored, an overestimation of the switching activity could occur.

Each monitored object over the simulation is responsible for registering and updating activity into the *power object database*. Consider again the model example in figure 2. During the elaboration of this model, the `state` and `result` objects are initialized and prepared to be monitored. Then the simulation is executed, and the objects monitor their activity, so that the gathered information is kept only internally to them. At the end of the simulation, and when the model is destroyed, objects `state` and `result` get their gathered information and post them to the *power object database*. This is facilitated by the C++ object destruction behavior.

Each object monitors its own activity, and only previously to its destruction post the gathered information to the *power object database*. This is necessary in order to avoid that at each delta cycle the *power object database* check the status of all objects involved in the simulation, looking for changes to be updated. Using this approach, we guarantee a small penalty to the simulator's performance.

4.2 Methodology

What is intended with this methodology is mainly easy-of-use and speed, in such a way to be transparent to the designer. The PowerSC's flow can be seen in figure 4, inside the dashed lines. First, the SystemC model together with some configuration files, e.g.,

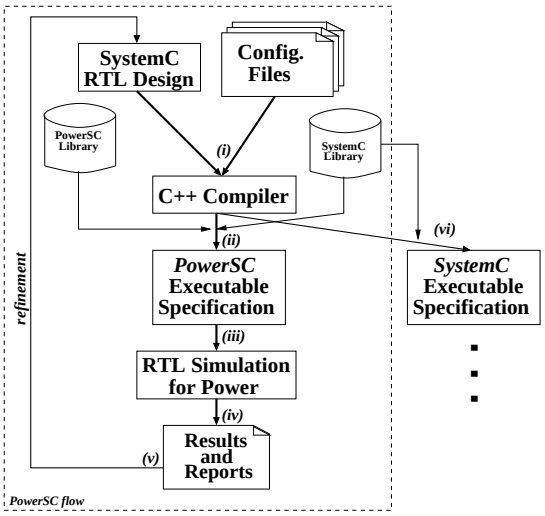

Figure 4: The PowerSC flow

Makefiles, is provided as input to a standard C++ compiler *(i)*, and then linked to both the PowerSC library and the SystemC library, generating as the output an augmented executable simulator *(ii)*, set to gather switching activity information. The RTL simulation is then executed *(iii)*, and over it, the power database is fed with the captured information *(iii)*. At the end of the simulation, a detailed report is shown *(iv)* with all the transition activity. Based on these results, the user can run this flow again *(v)*, until the design requirements are met. Alternatively, the ordinary SystemC flow runs easily, just instructing the C++ compiler to generate the usual executable specification *(vi)*.

Very few modifications are necessary in the SystemC description, and one mandatory is the inclusion of the main PowerSC header file (`powersc.h`) within the model files, and the call to the macro `PSC_REPORT_SWITCHING_ACTIVITY` at the end of the simulator's main function, to print out the results. This is shown in figure 5. As it can be seen in figure 5, the effort to enable PowerSC switching activity computation is minimal. In the file named `powersc.h` are all the necessary definitions of the PowerSC library. A fragment from this file is shown in figure 6. As it can be seen, the same model can be used both for the ordinary SystemC flow and the PowerSC flow, requiring only the setting of some compiler directives. If the `POWER_SIM` directive is set when the model is compiled, the declared SystemC objects will be redefined, avoiding the need of rewriting the model. Therefore, all we need to do is to provide two Makefile options, one for each flow, and a simple recompilation of the simulator would enable (or not) the capture of switching activity.

```
#include <systemc.h>
#include <powersc.h> // mandatory
        ...
SC_MODULE(example) {
   sc_in<bool> clk;
   sc_in<sc_int<32> > in;
   sc_in<sc_int<32> > out;
        ...
   sc_signal<sc_uint<4> > state;
   sc_uint<32> result;
        ...
   void proc() {
      result = in.read() * 2;
      out = 42; // default value
      if ( whatever condition )
         out = in.read() + 1;
   }
        ...
   SC_CTOR(example) {
      SC_METHOD(proc);
      sensitive_pos << clk;
   }
};
        ...
int sc_main(int argc, char **argv) {
        ... // module instantiation
   sc_start( simulation time );
   // prints the report
   PSC_REPORT_SWITCHING_ACTIVITY;
   return( 0 );
}
```

Figure 5: The example modified to use PowerSC

4.3 Advantages and Limitations

The idea used in this methodology is simple, but it is only possible thanks to the power of expression of a high-level language like C++. The main contribution of this work is to enable the fast gathering of switching activity in a transparent way to the designer, with the minimum effort, and without including new tools in the design flow. The only apparatus necessary in the design flow are the SystemC library and a standard C++ compiler. It is well known that when we move up the abstraction level, fewer information is available in the description, and the absolute accuracy is worse than at lower levels. But what we initially propose here with this methodology is to give a fast estimation, allowing the designer to compare two alternative solutions and decide the best on regarding power consumption,

```
#include "psc_uint.h"
#include "psc_lv.h"
#include "psc_signal.h"
   ...
using psc_dt::psc_uint;
using psc_dt::psc_lv;
using psc_dt::psc_signal;
   ...
#ifdef POWER_SIM
#define sc_uint   psc_uint
#define sc_lv     psc_lv
#define sc_signal psc_signal
#else
   ...
#endif
   ...
```

Figure 6: Extract from powersc.h

thus detecting possible power bottlenecks earlier in the design flow.

But there are still important gaps to be filled in this methodology, as for example, the absence of technology library information. However, this is a feasible task, since it would not be difficult to connect a technology library with information like capacitance and leakage power to our approach. Another alternative is to characterize a library like in the empirical macromodeling techniques in order to augment the power database to obtain a more accurate estimation at higher levels of abstraction.

5 Experimental Results

The methodology presented above was evaluated running two different design flows: the PowerSC flow (figure 4) and the Power Compiler flow (figure 1), which was used in order to validate our results. The simulator used to execute the Power Compiler flow was ModelSim. For PowerSC, the GNU GCC was used as the C++ compiler, and the host machine for the experiments was a standard Pentium IV/Linux. A set of SystemC designs were chosen, ranging from 16-bit full-adder to a real-world hardware implementation of an MP3 decoder [13] (we used the Discrete Cosine Transform module from the MP3 IP). The results obtained with the experiments are shown in table 1. This table is divided into five columns: the experiment name, the number of cycles simulated, the toggle count (TC) and simulation time to gather this value for the Power Compiler and PowerSC. The last column shows the difference between the Power Compiler and PowerSC regarding the toggle count. Each experiment ran four simulations, ranging from *50,000* to *5,000,000* cycles.

As we can see in the table, the results obtained for the first four experiments are almost equal for both the Power Compiler and PowerSC, with an irrelevant difference. In the fifth design (vending machine), PowerSC overestimates Power Compiler by about *2%*. This overestimation is due to the lack of representation of the four logic states ('0', '1', 'X', 'Z') for some data types in SystemC, as for example, the `sc_uint<W>` type. Consider the code below:

```
mul_b = temp - y_out;
```

where `mul_b`, `temp` and `y_out` are of type `sc_uint<W>` for SystemC and `reg [W-1:0]` for the converted Verilog code. During simulation with ModelSim, if some of the inputs (`temp` or `y_out`) for some reason has the logical value 'X' (or 'Z'), the value is propagated to the output, so that the `mul_b` will also have the logic value 'X' (or 'Z'), not accounting the transition 0→1 or 1→0. However, in the SystemC simulation, this is not yet detected by PowerSC in the above mentioned types, because their bits represent only '0' or '1'. So, an assignment as that in the example will contribute to the total transition count. Nevertheless, for most designs, this account for a very small contribution.

Next in the table is the FIR design. For this experiment, there is an average difference of about *33%*. This expressive difference is mainly due to a limitation of the Power Compiler interface to the Verilog PLI: it is not capable of gathering toggle information for register files. This problem is documented in the Synopsys SolvNet with doc id 903543. To illustrate the problem, we created a simple model in SystemC that is composed only of a register file of the following type:

```
sc_uint<32> buff[32];
```

The equivalent construction in Verilog, which is not supported by the Power Compiler is:

```
reg [31:0] buff [31:0];
```

During simulation, we sequentially write random values to this register file, in order to induce the transition of it. We performed this task for both PowerSC and the Power Compiler flows, obtaining a transition count of *3,968,573* with PowerSC and the value obtained with the Power Compiler regarding the transition count of this register file was *zero*.

Similarly to FIR and to this simple example, the MP3-DCT design also has register files, and the average difference between PowerSC and Power Compiler is of about *15%*. Register files are very usual constructions in real models, and this limitation can result in a great underestimation for some designs, as shown, which can lead to a poor accuracy when computing the power consumed by these designs. When we remove this support in PowerSC, the final results became similar. Notice that PowerSC achieved the correct results.

Consider again the MP3 model in table 1, that is the biggest design of our experiments, and it is a real-world and hardware validated model. As one can see, the simulation time required to obtain the transition count with the Power Compiler was of *7m31s*, while by using our approach this time was reduced to *4m38s* (approximately 68%). A performance improvement can also be seen for other high-level designs (FIR and shift-reg+counter). The other designs have a greater simulation in PowerSC, and this happens because these designs are described at the bit-level.

5.1 Relative Power Comparison

For the sake of simplicity, and to illustrate how relative comparisons regarding power consumption can be done using the gathered switching activity by PowerSC, consider the ripple-carry and carry-lookahead adders from table 1. It is well known that a carry-lookahead adder uses more hardware than a ripple-carry adder, but for the example, assume that the designer has not this previous knowledge. All he wishes is a design with the behavior of an adder. The switching activity for both design alternatives can be easily acquired with PowerSC, and the results in table 1 show that, as a matter of fact, the toggle count captured of the carry-lookahead adder (*249.149,338*) is much higher (*47%*) than the toggle count of the ripple-carry adder (*169,156,997*), as expected. Using this a idea for more complex designs, alternative solutions can be fastly compared, and even with this rough estimation, power consumption bottlenecks can be detected, and decisions can be made early in the design-flow.

6 Concluding Remarks

This paper presented a methodology to enable the fast gathering of switching activity from SystemC RTL descriptions, and for that purpose was introduced the PowerSC library. It was

Experiment	Cycles	Power Compiler		PowerSC		Error
		TC	Time	TC	Time	
shift-reg + counter	50K	318,011	0m01s	318,027	0m00s	∼0%
	500K	3,180,011	0m02s	3,180,027	0m01s	
	5M	31,800,011	0m24s	31,800,027	0m18s	
count-zeros	50K	589,950	0m01s	589,955	0m01s	∼0%
	500K	5,902,700	0m05s	5,902,705	0m05s	
	5M	58,974,908	0m45s	58,974,913	0m54s	
ripple carry adder16	50K	1,695,147	0m01s	1,692,997	0m03s	∼0%
	500K	16,940,007	0m14s	16,918,804	0m38s	
	5M	169,353,123	2m14s	169,156,997	6m28s	
carry lookahead adder16	50K	2,492,433	0m03s	2,492,492	0m06s	∼0%
	500K	24,917,937	0m17s	24,917,022	1m04s	
	5M	249,149,809	2m50s	249,149,338	10m46s	
vending machine	50K	284,375	0m00s	290,623	0m00s	+2%
	500K	2,843,746	0m02s	2,906,249	0m05s	
	5M	28,437,496	0m20s	29,062,499	0m47s	
FIR	50K	47,399	0m00s	59,437	0m00s	+33%*
	500K	4,747,180	0m04s	5,949,614	0m03s	
	5M	47,476,132	0m36s	59,503,182	0m31s	
MP3 DCT	50K	7,578,370	0m05s	8,648,401	0m02s	+15%*
	500K	76,081,129	0m45s	86,511,412	0m27s	
	5M	761,047,804	7m18s	865,285,039	4m38s	

Difference due to a limitation in the Power Compiler, explained in the experimental results section

Table 1: Switching activtity (toggle count) gathered and time with PowerSC and Power Compiler

shown that by using PowerSC, it is possible to reduce the steps necessary to gather switching activity information from SystemC models, transparently and requiring the minimum effort from the designer, thus allowing tradeoffs earlier in the design flow. Also, the tools required for this process were reduced to only the SystemC library itself and standard C++ compiler.

The results obtained using this methodology showed the feasibility of obtaining switching activity information with results very close to a commercial tool, and we also overcame a limitation of the compared commercial tool, capturing switching activity more accurately in some cases.

References

[1] L. W. Nagel. Spice2: A computer program to simulate semiconductor circuits. Erlm520, Univ. California, Berkeley, 1975.

[2] Open SystemC Initiative. *SystemC Language Reference Manual*, revision 1.0 edition, 2003. See http://www.systemc.org.

[3] K. D. Müller-Glaser, K. Hirsch, and K. Neusinger. Estimating essential design characteristics to support project planning for ASIC design management. In Louise Goto, Satoshi; Trevillyan, editor, *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 148–151, Santa Clara, CA, November 1991. IEEE Computer Society Press.

[4] S. R. Powell and P. M. Chau. Estimating power dissipation of vlsi signal processing chips: The pfa technique. *VLSI Signal Processing IV*, pages 250–259, 1990.

[5] Paul E. Landman and Jan M. Rabaey. Activity-sensitive architectural power analysis. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, pages 571–587. IEEE Computer Society Press, June 1996.

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

[7] 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.

[8] M. Eiermann and W. Stechele. Novel modeling techniques for rtl power estimation. In *Proceedings of the 2002 ISLPED*, pages 323–328. ACM Press, 2002.

[9] A. Raghunathan, S. Dey, and Niraj K. Jha. Register-transfer level estimation techniques for switching activity and power consumption. In *Proc. of the IEEE/ACM international conference on Computer-aided design*, pages 158–165, 1996.

[10] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. The design and use of simplepower: a cycle-accurate energy estimation tool. In *DAC '00: Proceedings of the 37th conference on Design automation*, pages 340–345. ACM Press, 2000.

[11] A. Bona, V. Zaccaria, and R. Zafalon. System level power modeling and simulation of high-end industrial network-on-chip. In *DATE '04: Proc. of the conference on Design, automation and test in Europe*, page 30318, Washington, DC, USA, 2004. IEEE Computer Society.

[12] Synopsys Inc. *Power Compiler User Guide*, v-2003.12 edition, December 2003.

[13] The Brazil-IP Project. Mp3 decoder ip. See http://www.brazilip.org.br.