Parallel Mixed-Level Power Simulation Based on Spatio-Temporal Circuit Partitioning

Mauro Chinosi, Roberto Zafalon, and Carlo Guardiani
Advanced Research, Central R&D DAIS, SGS-THOMSON Agrate B. (MI), ITALY
mauro.chinosi@st.com roberto.zafalon@st.com carlo.guardiani@st.com

Abstract: In this work we propose a technique for spatial and temporal partitioning of a logic circuit based on the nodes activity computed by using a simulation at an higher level of abstraction. Only those components that are activated by a given input vector are added to the detailed simulation netlist. The methodology is suitable for parallel implementation on a multi-processor environment and allows to arbitrarily switch between fast and detailed levels of abstraction during the simulation run. The experimental results obtained on a significant set of benchmarks show that it is possible to obtain a considerable reduction in both CPU time and memory occupation together with a considerable degree of accuracy. Furthermore the proposed technique easily fits in the existing industrial design flows.

I - INTRODUCTION

In the last few years the complexity of electronic systems design increased exponentially, thus again outpacing the Moore's law [1]. At the same time the market pressure is leading to an equivalent fast growth of the number of functionalities integrated in a single chip, associated with always tighter performance constraints. This trend represents a considerable dilemma for the designers of portable, low-power circuits. The two dimensional (area, timing) design space is quickly expanding to a third design dimension, with power constraints becoming of primary importance for reliability, packaging cost and battery life requirements. Dynamic simulation still plays a major role for fast, accurate functional and performance analysis, but it appears to be a bottleneck in the verification flow. A huge amount of work has been done in order to improve large circuits simulation efficiency: at transistor level by exploiting relaxation [2], event driven techniques [3] and efficient device models [4]; at logical level by RTL or behavioral language modeling [5]. In the power domain, efficient power analysis tools [6], [7] and high level estimation techniques [8], [9], [10] have been introduced to overcome the complexity of the VLSI design verification problem. At the same time a generation of mixed-mode, as well as multi-domain analog simulators have also been recently introduced, improving the state of the art in this field. In [10] it is described a two level simulator for power characterization of macros in which the information available during high level simulation is exploited to reduce the number of slow, accurate low-level simulations. Saleh [11], described an integrated simulation flow going across multiple levels and mixed domains, thus addressing the issues of interfaces, inter-domain transformation and algorithms for various analysis regimes. In [12] existing simulators at different levels of abstraction are combined in a hierarchical way in order to efficiently reduce the simulation time for accurate typical current estimation. In [14], a macro-modeling based gate-level power/timing analysis tool is described, achieving transistor level accuracy with one order of magnitude efficiency improvement. Parallel simulation techniques have been explored as an alternative approach for complexity reduction. Authors in [15] proposed a transformation of the VHDL description of a logic circuit into a parallel discrete event model that is suitable for concurrent simulation. A pattern partitioning strategy has been proposed in [16], [17] for parallel power estimation based on Monte Carlo sampling. In our work we exploit the speed of a logic simulator to perform the simultaneous partitioning of a circuit into sub-circuits and of the simulation interval into time windows. This spatio-temporal partitioning is realized in such a way that only those sub-circuits that are active in a particular time window are simulated in that window. At the same time, each of the sub-circuits thus obtained is independent from all the others, and can be simulated in parallel on a computer network. By using any transistor level simulators a corresponding accuracy can be obtained, while at the same time a speed-up factor that is roughly proportional to the number of available computing resources is achieved. Another interesting feature of the proposed system is the possibility of applying the more detailed and computationally intensive level of simulation only to limited sub-intervals of the entire simulation time, and applied only to the subset of circuit elements that are active during that time interval. With respect to previous work, our technique does not require any kind of block macro-modeling and it does not use statistical sampling methods. The paper is organized as follows. The methodology for partitioning is described in Section II and the parallel simulation environment is illustrated in Section III. Section IV shows the experimental results conducted on industrial designs as well as on a set of ISCAS89 benchmarks. Conclusions and future work are given in Section V.

II - NETLIST PARTITIONING

In general the partitioning problem can be described as the problem of finding a partition of a set of components V into k subsets $V_1, V_2, \ldots, V_k$, in such a way that a target function is optimized, subject to a set of constraints [13]. Of course the actual expression of the target and constraint functions is strongly dependent on the particular context. The netlist partitioning phase in a logic simulation is the step in which the circuit is virtually split into a set of different interconnected sub-circuits. The partitioning effort is compensated by the fact that the global complexity of solving all the individual subproblems is less than that associated with the solution of the initial problem because the simulation algorithms are at best superlinear in the number of circuit elements. Furthermore it is possible to exploit the intrinsic latency that is typical of logic designs to neglect the inactive sub-circuits, thus considerably decreasing the simulation time. Usually logic netlist partitioning is based on static, topological information. In our approach we propose to use the activity of the circuit, extracted from a higher level simulation, in order to perform a dynamic partitioning of the logic block to be simulated. In this way the circuit is modeled as a dynamic combination of components activated in sequence by the propagation of signals. Therefore the dynamic partitioning problem is formulated in terms of a set of time varying subsets of components $V_1(t), V_2(t), \ldots, V_{k_t}(t)$. At any time, the active subset is the set of devices that present a non-zero activity. The components belonging to the non-active subsets can be functionally ignored and therefore they only have to be considered as a load for the active sets. The dynamic evolution of the network is not sensibly modified by assigning the value of the stable nodes to the driven or loaded devices and by replacing the inactive devices with a fixed capacitive load. In principle a different static partitioning could be generated for every different input pattern, provided that the corresponding set of active devices can be determined. The partitions thus obtained will contain the minimum number of elements that needs to be simulated when applying that particular input pattern. This holds independently from the absolute time once the state of the network is known. The spatial partitioning into active and inactive subsets is associated with a temporal partitioning obtained by repeating it for every pattern or any arbitrary subsequence of patterns. Let introduce the following definitions: *i*) Cell: a set of primitives (MOS, capacitors, etc.) enclosed in a block of which the boundary I/O is observable; *ii*) Active cell: a cell is considered active if a ‘digital’ transition is detected at any of its ports. Let also assume that the following information can be extracted from the logic simulator, for example by means of a procedural interface language (PLI): *1*) the list of all signals; *2*) for a given cell, the list of all signals connected to the cell; *3*) at any given time, the value of any signal. By using this information, all the cells that become active after a signal transition can be determined at any time. Furthermore the state of inactive signals can be also determined . The pseudo code in figure 1 illustrates the complete flow of the proposed algorithm.

```
Foreach NET {
        if(is_simulated) {
                CELL <- Find cells connected to NET
                Update List_of_cells(CELL)
        }
}
Foreach CELL in List_of_cells {
        Foreach port of CELL {
                NET <- Find net connected to the port
                if(not_simulated) {
                        Get port direction
                        Value <- Get value (NET)
                        if(is_input_port) {
                            if(Value == 1) Connect port to VDD
                                else Connect port to GND
                        }
                        else {
                            if(Value == 1)
                                Set initial condition of port to VDD
                            else
                                Set initial condition of port to GND
                        }
                }
        }
        Create instance of cell
        Add subcircuit of Cell
}
```

Fig. 1: Active cells extraction

In general a cell is included in the sub-netlist to be simulated at transistor level if at least one of its ports is connected to a net that changed its value. Next, all the ports are processed and: *i*) connected to the net if it is a simulated net; *ii*) connected to VDD or GND if are not simulated input; *iii*) connected to the corresponding net and initialized if are not simulated output or inout. In order to achieve the highest efficiency in spatial partitioning the memory nodes internal to the cells (e.g. storage nodes of master-slave FFs) are made visible to the logic simulator. The state of the internal nodes, that is not observable from the I/O boundary, remains unspecified and it is eventually resolved at the electrical level. A simple example of the dynamic partitioning technique proposed in this paper is illustrated in figure 2. The time elapsed between two consecutive top level input vectors has been assumed as the time frame for temporal partitioning.

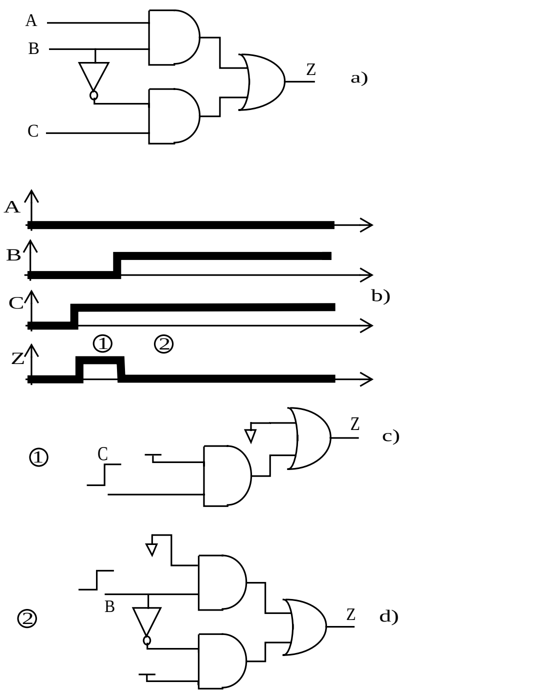

Fig. 2: a) circuit, b) input patterns, c) active cells for pattern 1, d) active cells for pattern 2.

Figure 2c shows how the 50% intrinsic latency that is characteristic of time frame(1) can be fully exploited to identify a suitable partitioning reproducing exactly the same I/O behavior of the complete circuit. During time frame (2) the circuit has no latency and it can be partitioned, as shown in figure 2d. It is important to observe that the two circuits are mutually independent and therefore they can be simulated in parallel on two different processors. By using the input stream transitions to define the time-domain partitioning granularity, it is possible to guarantee the correct propagation of the signal slopes across the cells, without having to introduce virtual D/A and A/D converters. A reasonable estimation of the transient duration [18], guarantees that the split is not performed while the internal nodes are switching, thus preserving the state of the network at its boundary. Back-annotation of delays and loads from layout is fully supported by the proposed technique and can be exploited to better represent the dynamic evolution of the network. The quality of the partitioning results obviously depends upon the particular choice of an objective function, which in turn depends upon the particular problem or application. In our case we have identified two different applications: mixed level analysis of digital systems, i.e. to allow considerably different levels of accuracy and efficiency within the same simulation session, and parallel simulation. As an example of a mixed level analysis let consider the simulation of a system for which the actual operating region that has to be verified is preceded by a long initialization phase (e.g. the boot of a micro-controller). In this case the logical simulator is simply used to determine the state of the network at the beginning of the interesting region, then a reduced netlist corresponding to the portion of the system that is sensitized in the active region, is dynamically extracted and simulated in greater detail. The graphs in Figure 3 sketches the CPU time reduction that is achievable in this way.

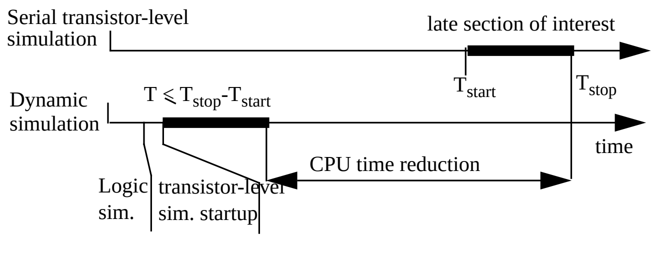
Serial transistor-level simulation
late section of interest
T_{start}
T_{stop}
Dynamic simulation
$T \leqslant T_{stop}-T_{start}$
time
CPU time reduction
Logic sim.
transistor-level sim. startup

Fig. 3: CPU time reduction when only a particular section of the complete simulation is of interest.

The dynamic partitioning, in this case, is obviously determined by the particular user selection of the intervals to be simulated in more detail. The choice of the objective function is less trivial for the parallel simulation case. One simple possibility is to minimize the total CPU time subject to the availability of a given number of concurrent computing resources. As an overhead is involved with the operations of creating an individual netlist for each sub-circuit and of spawning each concurrent simulation, and since this overhead is proportional to the number of simulated sub-circuits, the global simulation time does not necessarily decrease monotonically with the number of different spatio-temporal partitions, and an optimal partitioning exists.

III - PARALLEL IMPLEMENTATION

As an example of parallel simulation let us consider the temporal evolution of the state of the circuit represented in figure 2. When the first pattern is applied, the portion of the network that is perturbed (figure 2c) is extracted and its state is determined by the logic simulator. Once this step is completed the transistor-level simulator can be invoked to accurately simulate the corresponding sub-circuit. This involves the execution of a start-up phase (netlist parsing, model loading, etc.) followed by the actual simulation (figure 3). Concurrently, the logic-level simulator can process the next pattern and extract the sub-circuit of figure 2d, suitable for a new, independent transistor-level simulation. Without losing generality the high level processing can be performed on an arbitrary number of patterns before triggering the lower level simulator. Let us assume that P is the total number of patterns in the I/O data stream, M is the number of computing resources available for transistor-level simulations and that a simple partitioning scheme is applied, e.g. such that P/M patterns are assigned to every transistor level simulation resource, as shown in figure 4. The total simulation time is given by the sum of the CPU time required for simulating P/M patterns at transistor-level, the start-up time of the last simulation and the CPU time required by the logic simulator to simulate P patterns, also including the extra time spent by the PLI to create the sub-circuit netlists.

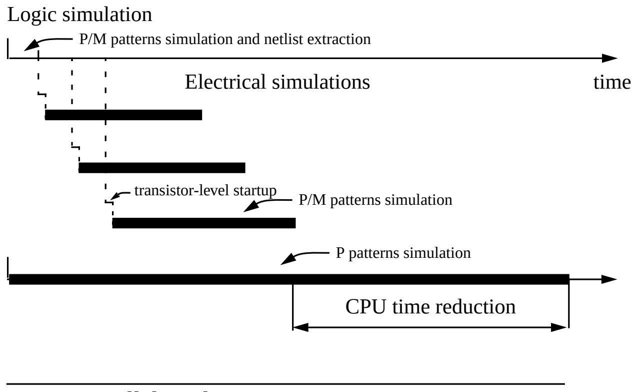

Fig. 4: Parallel implementation

As the time spent by the PLI to create the netlists can be larger than the actual logic simulation time for a given interval, the entire process can be further accelerated if the logic simulation is also concurrently executed by M simulators, each one targeting the creation of a single subcircuit. In this case the contribution of the logic simulator is also reduced by a factor of M. We propose a method for the parallel implementation of the dynamic partitioning algorithm that is based on a semi-empirical estimation of the CPU time reduction obtained by the concurrent vs. sequential implementation. Let assume that N is the number of primitives in the circuit, where a primitive can be any of the components in the transistor-level netlist. The expected CPU time for the serial electrical simulation can be evaluated as:

(1) $$CPU_{ser} = \alpha \times P \times N^{\beta} \qquad 1 < \beta < 2$$

Where α is platform dependent and β is bounded by the worst case complexity of a linear time solver and a Newton solver [19]. Both α and β can be empirically pre-characterized by measuring the CPU time required to simulate a given number of primitives. Let also define by M the number of available computing resources, k the total number of different temporal partitions, OV the overhead for netlist extraction after P/k patterns. The upper bound for the parallel simulation time (CPU_{par}) can be expressed as:

(2) $$CPU_{par} = \alpha\left(kOV + \frac{P}{k}N_{Max}\left(\frac{P}{k}\right)^{\beta}\right) \qquad 1 \le k \le M$$

Where N_{Max} is the maximum number of primitives obtained from spatial partitioning. The worst case occurs when the spatial partitioning can not be performed in one or more time intervals, that is when at least one interval is associated with a zero-latency condition, or, of course, when no temporal partitioning is performed ($k = 1$ and $N_{Max} = N$). When the extraction overhead is negligible CPU_{par} will follow the hyperbolic trend of curve B in figure 5. Otherwise the efficiency will be lower (curve A). The advantage of the proposed methodology becomes more evident when dealing with large circuits and long input sequences. In this case the CPU time can be effectively reduced by a factor of M.

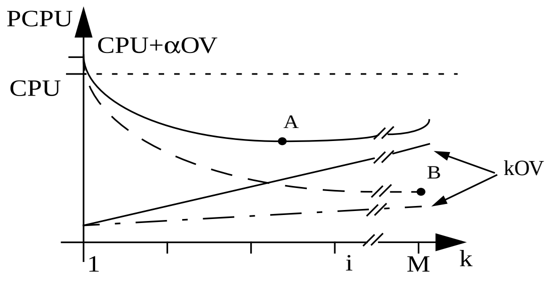

Fig. 5: Upper bound for PCPU. A) when overhead is comparable to the electrical simulation. B) when overhead is negligible.

The optimal partitioning can be found by adaptively determining the value of k and the number of simulation vectors associated with each time interval that minimizes the total CPU time. This optimization problem can be formulated as follows:

(3) $$\begin{gathered} min_{k, P_i}(max(\alpha(P_i N_i(P_i)^{\beta} + iOV))) \\ \text{Subject to: } \alpha P_i N_i(P_i)^{\beta} \le CPU_{max} \qquad \forall i < k \\ k \le M \end{gathered}$$

In practice the constraint on k can be relaxed, thus allowing each processor to run multiple shorter simulations in sequence. In principle any of the existing k-way partitioning algorithms [13] could be applied to the solution of the optimization problem described in Eq. (3). As the efficiency and optimality of the partitioning algorithm are not critical for the global performance of our method, in our implementation we decided to use the sub-optimal adaptive algorithm illustrated in figure 6. The dashed curves represent equal contour lines for the function $CPU(P, N)$. Applying P_s patterns to a generic circuit with N_s elements will require a CPU time less than CPU_{max} if the corresponding point $CPU_s = CPU(P_s, N_s)$ lies below the CPU_{max} curve. The algorithm proceeds by adding one vector at a time to the current temporal partition until the corresponding predicted CPU value exceeds CPU_{max}. When this happens the simulation is split at the previous time step, the overhead is added to the total CPU time and the corresponding subcircuit netlist is created. This technique provides a simple and efficient adaptive control on the elapsed CPU time independently of the circuit size. Shorter simulation sequences will be applied to larger circuits in such a way that the estimated CPU time will be always bounded by the pre-defined CPU_{max} limit.

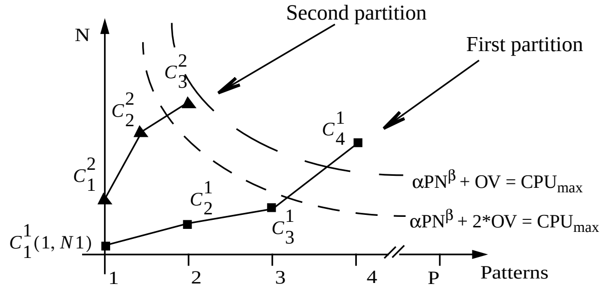

Fig. 6:Adaptive temporal partitioning algorithm

IV - EXPERIMENTAL RESULTS

The proposed dynamic partitioning algorithm has been implemented and integrated in a flow for detailed full-chip power simulation and verification. This system has been used to accurately determine the power consumed by a variety of different CMOS digital and mixed-signal circuits, including memories, combinational and sequential blocks. The partitioning scheme has been implemented by using the Verilog-XL PLI [20]. Depending on the circuit complexity we used either Powermill [4] or ELDO [2] as the accurate transistor level simulator. The results obtained on a set of circuits including both ISCAS-89 benchmarks and industrial products are presented in Table 1. The simulation time speed-up is shown in column 3. As it can be noted the best improvement was achieved with the largest circuits (mic2, bmu). In the first case the theoretical CPU time reduction coming from parallel simulation has been magnified by a factor of two thanks to the exploitation of circuit latency obtained by applying spatial partitioning, whereas in the other case a less significant improvement was obtained. The reduction of memory occupation is shown in column 4. The accuracy of the proposed methodology is evaluated by using the following formula:

$$(4) \qquad Err = E[\overline{I_{dyn}}(\Delta t) - \overline{I_{ser}}(\Delta t)] \qquad \Delta t \ll T_{sim}$$

where $\overline{I_{dyn}}(\Delta t)$ and $\overline{I_{ser}}(\Delta t)$ represent the average over a small time interval Δt of the supply current computed by using the dynamic partitioning and the serial electrical simulation of the full circuit respectively, and $E[\ ]$ is the expected value of the bracketed quantity. Note that the average supply current usually represents one of the most critical parameters for electrical simulation accuracy. In column 6 the overhead is compared with the total simulation time in order to show the performance degradation and the potential gain for small and large simulations respectively.

Circuit	Primitives	$\frac{PCPU}{CPU}$	$\frac{Mdyn}{Mfull}$	Err %	$\frac{OV}{PCPU}$
9symml	300	0.75	0.78	+2.8	0.56
moore	75	0.51	0.98	+3	0.35
mic2	15090	0.14	0.82	-2	0
ADCenc	3600	0.72	0.8	+5.2	0.54
SRAM	4250	0.96	0.4	-4.2	0.66
mux	120	0.85	0.88	-0.2	0.61
m8051	1350	0.61	0.05	+1.3	0.46
bmu	22000	0.35	NA	+1.6	0.06

TABLE 1: EXPERIMENTAL RESULTS

The comparison between voltage and currents waveforms obtained from parallel and serial simulation is shown in figure 1 and 8 for a few nodes of a large SRAM circuit. The istantaneous discrepancies are explained by the unknown value of the internal nodes. Nevertheless the accuracy is still remarkable in both cases, and it is certainly sufficient for either timing or power analysis .

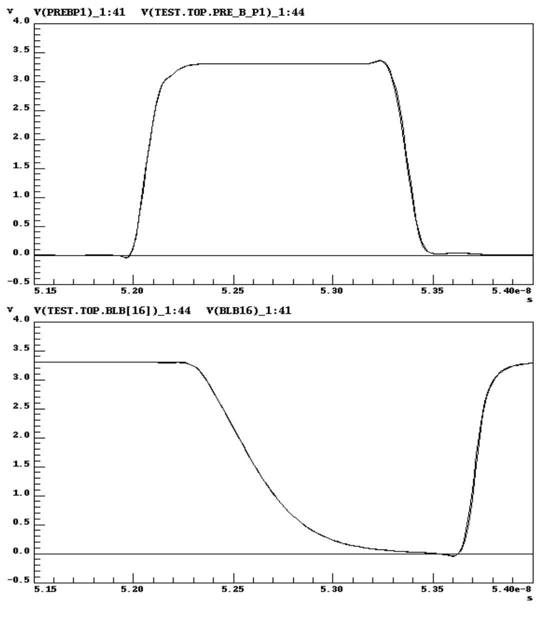

Fig. 7: Comparison between SRAM bit lines voltage waveform obtained by our technique and transistor level simulation. The curves are not practically distinguishable.

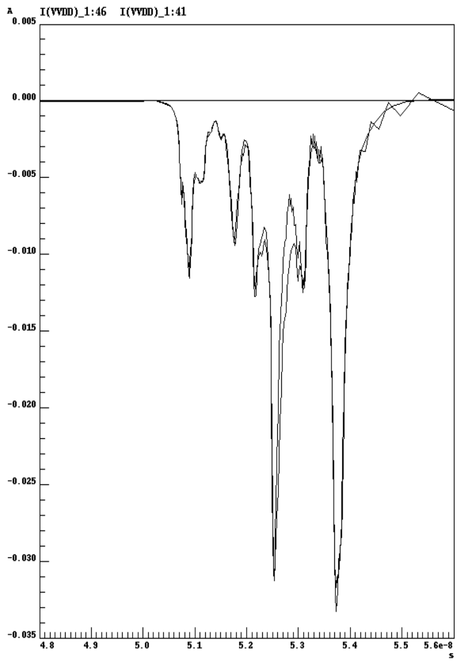

Fig. 8: Current Waveform comparison

V - CONCLUSIONS

We have presented a dynamic circuit partitioning methodology. The proposed technique allows to split the simulation time into intervals, and correspondingly the circuit topology into subcircuits by using the node activity extracted from the results of a simulation at an higher level of abstraction. The pairs (sub-circuit, sub-interval) thus obtained, are all independent. Therefore the expensive detailed (e.g. transistor level) simulations can be executed in parallel. By properly preserving the state at the boundary of each partition both voltage and current waveform can be computed with a remarkable degree of accuracy, as confirmed by the experimental results presented in this work. At the same time our method provides a considerable decrease of both memory and CPU requirements, sometimes making the difference between feasibility and infeasibility of a given problem. The implementation of the proposed technique and its integration in existing design flows is rather simple, thus making it suitable for relevant industrial applications such as power or timing characterization of large macros, full-chip power analysis, and mixed-signal simulation.

VI - REFERENCES

[1] R. Zafalon, C. Guardiani, “Power Estimation and Synthesys: An Industrial perspective”, Invited talk at PATMOS-97

[2] ELDO, “User Manual”, Mentor Graphics, Wilsonville, Oregon

[3] A. Devgan and R. Rohrer, “Event Driven Adaptively Controlled Explicit Simulation of Integrated Circuits”, 1993

[4] C. X. Huang, etc., “The Design and Implementation of PowerMill”, ACM/IEEE International Symposium on Low Power Design, pp. 105-109, 1995

[5] R. Lipsett, C. Shaefer and C. Ussery, “VHDL: Hardware Description and Design”, Kluwer, 1990, Boston, MA

[6] DesignPower, “Reference Manual v1998.02”, Synopsys Inc., Moutainview, CA, 1998

[7] WattWatcher, “User manual”, Sente’, Inc., Acton, MA

[8] M. Nemani, F. N. Najm, “Towards a High-Level Power Estimation Capability”, IEEE Transaction on CAD of Integrated Circuits and Systems, pp. 588-598, Vol. 15, No. 6, june 1996

[9] L. Benini, G. De Micheli, E. Macii, M. Poncino, R. Scarsi, “Fast Power Estimation for Deterministic input Streams”, ICCAD-97, pp. 494-501

[10] L. Benini, G. De Micheli, E. Macii, M. Poncino, R. Scarsi, “Quick Generation of Temporal Power Waveforms for RT-Level Hard Macros”, IEEE-97 International Conference onInnovative Systems in Silicon, ISIS-97, pp. 331-337

[11] R. A. Saleh, B. A. A. Antao and J. Singh, “Multilevel and Mixed-Domain Simulation of Analog Circuits and Systems”, IEEE Transaction on CAD of Integrated Circuits and Systems, Vol. 15, No. 1, Jan. 1996

[12] P. Vanoostende, P. Six, J. Vandewalle and H. J. De Man, “Estimation of Typical Power of Synchronous CMOS Circuits Using a Hierarchy of Simulators”, JSSC, Vol. 28, No. 1, Jan. 1993

[13] F. M. Johannes, “Partitioning of VLSI Circuits and Systems”, 33th DAC, pp. 83-87, 1996

[14] D. Rabe, G. Jochens, L. Kruse, W. Nebel, “Power-Simulation of Cell Based ASICs: Accuracy and Performance Trade-Offs”, Proceedings of DATE-98, pp. 356-361

[15] E. Naroska, “Parallel VHDL Simulation”, Proceedings of DATE-98, pp. 159-163

[16] V. Kim, “Parallel Algorithms for CMOS Power Estimation”, Master Thesis, Northwestern University, 1997, available at http://www.ece.nwu.edu/cpdc/TechReports/

[17] V. Kim and P. Banerjee, “Parallel Algorithms for Power Estimation”, Proceedings of DAC-98

[18] M. Kassab, E. Cerny, S. Aourid, T. Krodel, “Propagation of Last-Transition-Time Constraints in Gate Level Timing Analysis”, Proceedings of DATE-98, pp. 796-802

[19] L. T. Pillage, R. A. Roher, C. Visweswariah, “Electronic Circuit and System Simulation Methods”, McGrawHill

[20] VERILOG-XL, “Reference Manual”, CADENCE Design Systems