

A VHDL-Based Approach for Power Estimation of Embedded Systems

William Fornaciari [^], Paolo Gubian ^{*}, Donatella Sciuto [^], Cristina Silvano ^{*}

([^]) Politecnico di Milano, Dipartimento di Elettronica e Informazione, P.zza L.Da Vinci, 32 - 20133 Milano, Italy, e-mail: {fornacia, sciuto}@elet.polimi.it, Fax:+39-2-2399 3411

(^{*}) Univ. degli Studi di Brescia, Dip. di Elettronica per l'Automazione, Via Branze, 38 - 25123 Brescia, Italy, e-mail: {gubian, silvano}@bsing.ing.unibs.it, Fax: +39-30-380014

Keywords: Embedded Systems, Low Power Design, Power Estimation, VHDL, VLSI circuits.

Contact author: William Fornaciari

Abstract

Power dissipation has become one of the main constraints during the design of embedded systems and VLSI circuits in the recent years, due to the continuous increase of the integration level and the operating frequency. The aim of this paper is to present an innovative conceptual framework suitable for achieving accurate and efficient estimation of power dissipation for embedded systems described in VHDL at the behavioral and Register Transfer levels. The goal is to provide the designer with the capability of analyzing and comparing different solutions in the architectural design space, before the synthesis. The analytical power model is hierarchical, considering the different parts of the target system architecture, mainly the data-path, the memory, the control logic and the embedded core processor. Experimental results have been obtained by applying the proposed power model to benchmark circuits.

1. Introduction

An increasing number of applications in several fields like automotive, telecommunication, consumer electronics, etc. is recently being implemented by using embedded systems. These systems have become broadly used in the most recent years, due to the wide diffusion on the market of standard processors characterized by high performances and reasonable prices.

We refer to embedded systems as those dedicated computing and control systems designed for specific target applications [28, 6], where dedicated software routines are provided with the system to respond to specific requirements. In general, the functionality of an embedded system is constituted by a fixed number of operating ways and it is determined by the interaction between the system and the environment. According to the particular application class for which the system is dedicated, the embedded systems can be classified as data or control dominated systems [28]. In both cases, the target system architecture is composed of an hardware and a software part. The software part is typically constituted by a set of application specific software routines running on a dedicated processor or ASIP (Application Specific Instruction Processor), while the hardware part consists usually of one or more ASICs (Application Specific ICs).

Due to the heterogeneous nature of the hardware and software parts of the embedded system, innovative co-design techniques have been proposed in the recent past, the goal being to meet the system-level requirements by using a concurrent design and validation approach, thus exploiting the synergism of the hardware and the software parts [6]. Several design aspects are involved in the co-design process at the system-level, including the system modeling, the capture of the functional specifications in a high-level language (co-specification), the

analysis and validation of the specifications, the exploration and evaluation of the different architectures with respect to some design metrics, the system-level partitioning, the co-synthesis and co-simulation.

The availability of an appropriate co-design methodology, covering all these design phases, is mandatory during the design of embedded systems in order to meet the system-level requirements. These requirements are typically defined in terms of some design constraints like performance, area, power dissipation, cost, reliability, testability and development time. In particular, the technological trends toward smaller geometry and the increasing performance levels lead to high-level integration, high clock frequencies and high power dissipation. Such aspects, combined with the growing demand of battery-powered portable systems, contributed to increase the importance of the power issues during the design of embedded systems.

Therefore, co-design techniques for low power dissipation and EDA tools for accurate power estimation have become a critical factor for embedded systems and IC designers, in order to satisfy the power constraints, without reducing the global performance significantly. Design techniques targeting low power dissipation and power estimation methodologies should be provided at several abstraction levels. In fact, circuit and logic-level power estimation techniques are no more sufficient, due to the high complexity and high integration levels of the embedded systems. Accurate low-level estimation techniques present some limitations due to the need to cope with circuit complexity in an acceptable design time. Moreover, low-level estimation techniques can be applied only during the last design phases, when a circuit or logic-level description is already available. However, a re-design process at these levels could be very expensive and time consuming.

Hence, high-level power estimation is a key issue in the early determination of the power budget for embedded systems, being unfeasible to synthesize every design solution down to the gate, circuit and layout levels in a reasonable time. The goal is to respect the design turn-around time, while exploring the architectural design space widely, and to early re-target the architectural design choices. Accuracy and efficiency of a high-level analysis should contribute to meet the power requirements, avoiding a costly re-design process. In general, the relative accuracy in high-level power estimation is considered much more important than the absolute accuracy, the main objective being the comparison of different design alternatives [10].

High-level power estimation tools are usually based on high-level descriptions. Up to now, most of embedded system descriptions are specified in a hardware description language, such as Verilog or VHDL, along with other graphical formalisms suitable for describing the functional behavior at the system-level, such as temporal diagrams, State Transition Graphs for Finite State Machines, Statecharts, etc. [9, 18]. In particular, VHDL has become the *de-facto* standard in the European design community for the hardware description and for the most part of the commercial design entry, synthesis and simulation tools.

The main advantage of VHDL is related to the possibility of specifying the system behavior by using a mixed description [18] at different abstraction levels: behavioral, Register-Transfer and structural. Therefore, VHDL provides high flexibility during both the design description and the simulation phases. Furthermore, VHDL supports a hierarchical design approach, where the description of the elements composing the hierarchy, properly connected, perform the global functionality. The hierarchical approach provides also the possibility to use a mixed description composed of behavioral, Register-Transfer and structural parts at the different hierarchical levels.

Other advantages of VHDL are related to the possibility of easily specifying both the data-path and the control-path of the system and to support the modular design approach. Hence,

VHDL allows the designer to re-use existing components. In fact, VHDL supports the definition of functions and procedures, to decompose a complex description into smaller and simpler functional units. These functional units can be organized as independent files, that can be compiled and verified separately, thus supporting the definition of a library of reusable cells and macro-cells. Finally, VHDL provides also the complete independence with respect to the technology used and the mapping between a given entity and different architectures, through the configuration approach.

The aim of this paper is to provide a conceptual analysis framework for accurate and efficient estimation of power dissipation in embedded systems and VLSI circuits at the architectural and RT levels. The availability of a power analysis tool at these levels of abstraction is of paramount importance to early obtain estimation results, while maintaining an acceptable accuracy. In fact, the architectural and RT-level descriptions, based on VHDL, are the design entry point for the majority of embedded systems and IC designs.

In the proposed approach, the analysis is based on a probabilistic estimation of the switching activity. The proposed model accurately accounts for both the switching activity and the physical capacitance [8] for all the parts composing the embedded system architecture.

The paper is organized as follows. The discussion starts by presenting the most significant research works related to high-level power estimation in Section 2. Then the target system architecture of embedded systems, we are focusing on, is introduced in Section 3, while Section 4 contains the foundations and notations, constituting the basis of the proposed analysis. Then, the proposed power estimation model is detailed in Sections 5-10 while some experimental results obtained from benchmark circuits are reported in Section 11, which also outlines the future developments of our investigations.

2. Previous Work on High-Level Power Estimation

General surveys of power estimation techniques at different abstraction levels can be found in [7, 20, 5, 24]. While several power estimation techniques have been proposed in the literature at the gate, circuit and layout levels, a few papers have been published addressing the power estimation problem at high-level until recently [7, 10], despite of the increasing interest in the system and behavioral levels design.

A state-of-the-art survey of the high-level power estimation has been presented in [10]. According to this survey, high-level power estimation techniques can be classified depending on their abstraction level. At the architectural or RT levels, there are two classes of techniques: the analytical and the empirical techniques.

The analytical methods aim at relating the power consumption to the capacitances and the switching activities of the design nets. These techniques are composed of complexity-based models and activity-based models. The former considers the design complexity of each part of the design, in terms of equivalent gates, as a measure of the capacitance, while the latter uses the concept of entropy, derived from the information theory, as a measure of the average transition activity in a circuit. More specifically, in the complexity-based models, first the number n of equivalent gates contained in each design function is specified in a macro-module library; then, the power estimates are obtained by multiplying n by the average power consumed by each equivalent gate. In the activity-based models, the average power is estimated as the product of the area, considered as a measure of the average nodes capacitance, and the entropy, considered as a measure of the activity.

The empirical methods are based on the power measures of existing implementations, then a macro-modeling approach is used to derive models from these measurements. The empirical methods can be sub-divided into fixed-activity models and activity-sensitive models. The former models disregard the influence of data-activity on power, while the latter consider the

effects of statistics related to data and instructions activity on power.

Moving up in the abstraction levels, the behavioral methods are based on static and dynamic activity prediction. The goal of the static activity prediction is the estimation of the access frequency of different hardware resources, by analyzing statically the behavioral description of the functions to be implemented. The dynamic activity prediction is based on a dynamic profiling to determine the activation frequencies of various resources and the memory accesses.

Finally, power exploration tools at the instruction and system levels can be used to identify power metrics to guide the system-level partitioning.

A common characteristic of power estimation at the different abstraction levels is that the average power is strongly related to the switching activity of the circuit nodes. Such a fact has been indicated in [20] as stating that power estimation is a pattern-dependent process. In particular, the input pattern-dependency of the power estimation approaches can be classified as strong or weak pattern-dependency [20].

The typical methods for power estimation based on extensive circuit simulation have been indicated in [20] as strongly pattern-dependent process. Main advantages of these simulation techniques derive from its accuracy and wide applicability. However, to obtain a complete and accurate power estimation, the designer should provide a comprehensive amount of input patterns to be simulated, thus making this approach very time consuming and computationally very costly. Therefore the simulation approach is almost impossible to apply to most of the designs, due to their increasing complexity.

To avoid the need of a large amount of input patterns, the weakly pattern-dependent approaches [20] require input probabilities. In this case, the estimation results will depend on the probabilities supplied by the designer, reflecting the typical behavior of the input signals.

Both probabilistic techniques and statistical techniques are presented in [20]. Probabilistic techniques suitable for combinational circuits have been illustrated, requiring user-supplied input probabilities to solve the pattern dependency problem. Statistical techniques use randomly generated input patterns to simulate the circuit repeatedly, then using statistical mean estimation techniques to stop simulation following a criterion to determine the closeness to the average power.

Analytical and stochastic power estimation techniques at the behavioral-level have been proposed in [16], targeting real time DSP applications on ASIC architectures. The power dissipated by some ASIC components, such as data-path components, have been analytically estimated from the Control Data Flow Graph (CDFG) representing the design. For other ASIC components, such as interconnects and controllers, for which the power information available at the behavioral-level is not sufficient, statistical models were built to estimate power based on a stochastic study on several ASICs. However, the proposed models do not account for the power consumed by multiplexers and memories. The estimation techniques have been included into an exploration tool that, given a CDFG description of an algorithm and a library of hardware modules, explores the space of the available solutions for different values of clock periods and supply voltages. The results have been compared with an architectural-level power estimator, called Stochastic Power Analysis (SPA) and proposed in [11], on 23 different chips, showing an average error of approximately 20%.

Other power estimation techniques based on high-level descriptions have been proposed in [11, 12, 13]. The techniques described in [11], targeting data-path architectures, derive stochastic models of busses and internal modules from the statistical behavior of inputs. In [12], a power estimation model for data-path architectures operating at the RT-level is described. The model accounts for the switching activity by using the Dual Bit Type (DBT) method, considering two input bit types rather than one: the random activity of the least

significant bits (LSB's) and the correlated activity of the most significant bits (MSB's). An architectural model for the power consumption of the control paths, called Activity-Based Control (ABC) model, has been presented in [13], using three implementation styles: a ROM-based controller, a PLA-based controller and a random logic controller.

Nevertheless, the methods proposed for high-level power estimation have not yet achieved the maturity necessary to enable their use within current industrial CAD environments. Our work is an attempt to fill such a gap, aiming at providing an high-level power model, based on VHDL descriptions, to cover the different parts composing the basic architecture of embedded systems.

3. The Target System Architecture

The target system architecture of the embedded system is implemented into a single ASIC, including both the software and the hardware bound parts, described at the behavioral/RT levels. The target system architecture is depicted in Figure 1 and it is quite similar to those proposed in [16, 17].

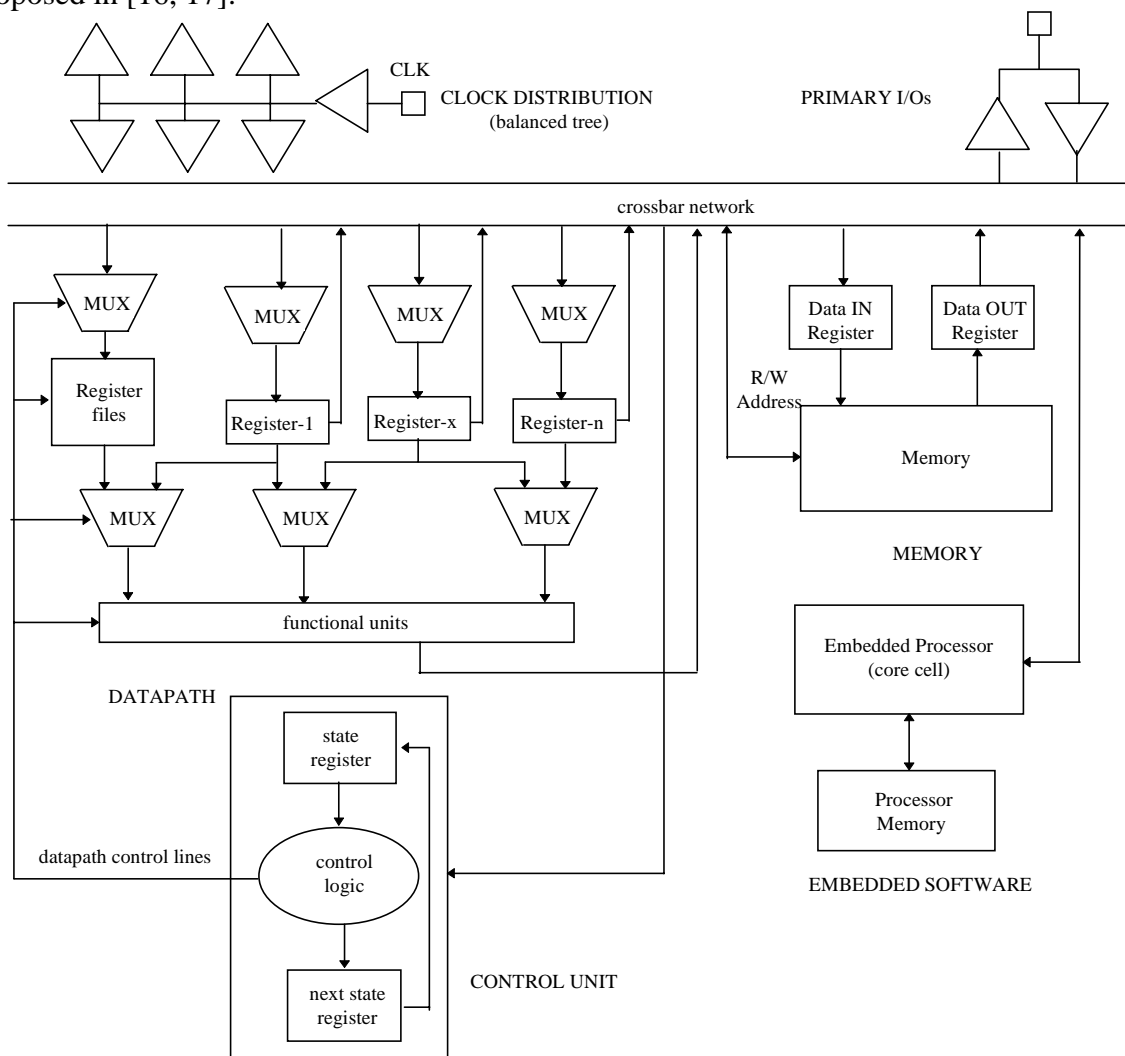


Figure 1: The target system architecture at the RT-level.

The ASIC architecture is defined at a pre-synthesis RT-level and consists of the following components:

- **the data-path**, composed of storage units, functional units and multiplexers. The storage units consist of registers and register files, while functional units can include a wide set of units such as adders, multipliers, and so on. A two-level multiplexer structure is

considered for the interconnection among storage and functional units. The typical operation along the data-path implies a register-to-register transfer, consisting of the operands read from the input registers, an operation performed on the operands and the results stored in the output registers [17];

- **the main memory**, based on a memory hierarchy, that can be constituted by single or multi-port memories, cache memories, TLBs, FIFOs, LIFOs, etc.. We assume that all read/write accesses to the memory will be performed through input/output registers;
- **the control unit**, implemented as a set of Finite State Machines and generating the control lines for the data-path components and the memory;
- **the embedded core processor**, such as a standard processor, a microcontroller, a DSP, etc., with its memory (even if part of the memory can be external) implementing the SW bound part;
- **the clock distribution logic**, including the buffers of the distribution network, organized for example as a balanced clock tree;
- **the crossbar network**, to interface the architectural units by using a communication protocol at the system-level. The interconnection power of the crossbar network is included in the power dissipated by the outputs of data-path, memory and control logic;
- **the primary I/O pads**.

4. Foundation of the Power Estimation

Power dissipation in CMOS circuits can be considered as composed of both a static and a dynamic component. Static power dissipation is mainly due to the leakage current of the reverse-biased diodes and the sub-threshold transistor conduction. However, in CMOS devices, static power dissipation can be considered insignificant in most designs [4].

The dominant part of the power dissipation in CMOS circuits is thus the dynamic component, which is composed of two terms [7]. The first term, indicated as the switching activity power, is due to the charge and discharge of the circuit node capacitances at the output of each logic gate. The second term, indicated as short-circuit power, represents the short-circuit current from the supply voltage to ground during the output transitions.

The switching activity power P_{SW} can be expressed as in [4]:

$$P_{SW} = V_{DD}^2 f_{CLK} C_{EFF}$$

where V_{DD} is the supply voltage, f_{CLK} is the system clock frequency and C_{EFF} is the effective switched capacitance, that is the product of the total physical capacitance C_{Li} of each node in the circuit and the switching activity factor α_i of each node (defined below) summed over all the N nodes in the circuit:

$$C_{EFF} = \sum_{i=1}^N \alpha_i C_{Li}.$$

The short-circuit power is due to the fact that, during a transition of a CMOS gate, both p and n channel devices may conduct simultaneously, briefly establishing a flow of current from the supply to ground. The short-circuit power P_{SH} can be expressed as in [7]:

$$P_{SH} = Q_{SC} V_{DD} f_{CLK} \alpha$$

where Q_{SC} represents the quantity of charges carried by the short-circuit current per transition and α is the global switching activity factor, i.e. the number of gate output transitions per clock cycle.

Anyway, in CMOS devices, the short-circuit current typically dissipates a small fraction of the dynamic power (in the order of 5÷10 % as reported in [4], so it can be ignored. Usually, in properly designed circuits, switching activity power accounts for over 90% of the total power dissipation, as reported in [7].

The switching activity of each signal (being a primary I/O or an internal signal), is fully characterized by the following two components [5]:

- a *static* component, taking into account the static probability of a signal;
- a *dynamic* component, taking into account the timing behavior of the circuit.

The static component can be expressed in terms of the *static signal probability* of each node n , that is the probability of the node to be at one:

$$p_n^1 = \lim_{N \rightarrow \infty} \left(\sum_{k=1}^N i_n(k) \right) / N$$

where:

$i_n(k)$ = value of i_n at the clock cycle k (i.e. 0 or 1)

N = number of clock cycles.

From the above definition, it derives that: $p_n^1 \leq 1$ and $p_n^0 = 1 - p_n^1$. A signal is called *equiprobable* when it has an uniform distribution of high and low levels: $p_n^1 = p_n^0 = 0.5$. The *transition probability* p_n^{01} is the probability of a zero to one transition at node n :

$$p_n^{01} = \lim_{N \rightarrow \infty} \left(\sum_{k=1}^N \bar{i}_n(k) i_n(k+1) \right) / N$$

The other transition probabilities (p_n^{10} , p_n^{00} , p_n^{11}) are defined similarly, while the following equations hold: $p_n^0 = p_n^{00} + p_n^{01}$ and $p_n^1 = p_n^{10} + p_n^{11}$.

In the spatial and temporal independence assumption [20], the transition probability p_n^{01} is given by the probability that the current state is zero times the probability that the next state is one: $p_n^{01} = p_n^0 p_n^1 = (1 - p_n^1) p_n^1$. Under the same assumption, the *switching activity factor* of a node n , indicated as α_n (or $E_{sw}(n)$), is: $\alpha_n = p_n^{01} + p_n^{10} = 2 p_n^1 (1 - p_n^1)$. Given the switching activity factor α_n of a node n , the corresponding *toggle rate* can be defined as $TR_n = \alpha_n f_{CLK}$, where f_{CLK} is the system clock frequency. Considering a N -bit bus, the *bus switching activity factor*, indicated as α_{BUS} , is defined as

$$\alpha_{BUS} = \left(\sum_{n=1}^N \alpha_n \right) / N$$

where α_n is the switching activity factor of the n -th bit of the bus. The corresponding bus toggle rate can be defined as: $TR_{BUS} = \alpha_{BUS} f_{CLK}$.

5. The Power Estimation Model

The proposed estimation approach is based on the VHDL description of the ASIC model at the behavioral/RT levels. The entire analysis is based on the probabilistic estimation of the nodes switching activity. The inputs for the estimation are:

- *the ASIC specification*, consisting of a hierarchical VHDL description implementing the target system architecture depicted in Figure 1;
- *the allocation library*, composed of the available components implementing the macro-modules (such as adders, multipliers, etc.) and the basic modules (such as registers, multiplexers, logic gates, I/O pads, etc.). Every component model includes the description of the logic behavior, the input capacitance, the area and the power characteristics;

- *the technological parameters* such as frequency, power supply, derating factors (accounting for the variations in process, voltage and temperature), etc.;
- *the switching behavior* of the ASIC primary I/Os.

The proposed model is based on the following assumptions:

- the supply and ground voltage levels in the ASIC are fixed, although it is worth noting the impact of supply voltage reduction on power;
- the design style is based on synchronous sequential circuits;
- the data transfer occurs at the register-to-register level;
- a Zero Delay Model (ZDM) has been used, thus ignoring the contribution of glitches and hazards to power.

The power model is an *analytical* model, that attempts to relate the average power dissipation of the VHDL descriptions to the physical capacitance and the switching activity of the design nets.

The estimation approach is *hierarchical*, in fact we propose an *ad-hoc* analytical power model for each part of the target system architecture, at the highest hierarchical level; these models are based on a macro-module library, at the lowest hierarchical levels.

Furthermore, to avoid the need of a huge amount of input patterns, our approach is weakly pattern-dependent, requiring user-supplied input probabilities, reflecting the typical input behavior, that are derived from the system-level specification.

In the proposed single ASIC architecture, the total average power dissipated, P_{AVE} , is given by:

$$P_{AVE} = P_{IO} + P_{CORE}$$

where P_{IO} and P_{CORE} are the average power dissipated by the I/O nets and the core internal nets, respectively. The value of P_{AVE} can be multiplied by the derating factor, δ , taking into account the effects of the variations of the fabrication process and the operating conditions (voltage and temperature) on the power values contained in the target library.

The power model of the core logic is based on the models of the different components of the target system architecture, therefore the P_{CORE} term can be detailed as:

$$P_{CORE} = P_{DP} + P_{MEM} + P_{CNTR} + P_{PROC}$$

where the single terms represent the average power dissipated by the data-path, the memory, the control logic and the embedded core processor. The power models related to the single terms in the above equations will be detailed in the following sections.

6. P_{IO} Estimation

Although a pre-synthesis analysis is performed, we assume the knowledge of the ASIC interface in terms of primary I/O pads characteristics and related switching activity from the system-level specifications. The set S of input, output and bi-directional nets of the ASIC can be partitioned into N sets, such as: $S = \{s_1, s_2, \dots, s_k, \dots, s_N\}$, where the k -th set s_k is composed of the same type t_k of I/O pads. Considering for example a set of output pads, the average power of the set s_k can be estimated as:

$$P_{S_k} = \sum_{i=1}^{n_k} P_i(C_i) TR_i$$

where:

n_k is the number of output pads in the set s_k ;

$P_i(C_i)$ is the average power consumption per MHz of the i -th output pad in s_k . The value of P_i is computed as a function of the output load C_i at a given reference frequency f_0 . This value is tabulated in the selected library (such as in [14] as (P_{f_0} / f_0) expressed in $[\mu\text{W}/\text{MHz}]$ as a

function of C_i expressed in [pF]; C_i is the output load of the i -th output pad expressed in [pF], derived from the system-level specifications. Note that the previous equation is valid in a range of f_0 ;

TR_i is the toggle rate of the i -th output pad, derived from the system-level specifications.

Similarly, the average power of the input pads can be computed, depending on the estimated internal standard loads and input ramptime.

7. P_{DP} Estimation

The average power dissipated by the data-path can be expressed as:

$$P_{DP} = P_{REG} + P_{MUX} + P_{FU}$$

where the single terms represent the average power dissipated by the registers, the multiplexers and the functional units.

7.1. P_{REG} Estimation

The *live variable analysis* [17] has been applied to the behavioral-level VHDL code to estimate the number of required registers and the maximum switching activity of each register.

The preliminary step is the estimation of the number of required registers and, consequently, the values of the toggle rate TR_i for each of them. According to the abstraction level, such data are directly available from the RT-level description or the live variable analysis can be applied to the behavioral-level specifications.

The algorithm examines the life of a variable over a set of VHDL code statements and it is similar to the one proposed in [17], for the computation of the lifetime of a variable in terms of its definition and use over a selected set of VHDL code statements. New passes have been added to the algorithm proposed in [17], to obtain information concerning the registers switching activity. The proposed algorithm [8] can be summarized as follows:

1. compute the lifetimes of all the variables in the given VHDL code, composed of S statements. A variable v_j is said to *live* over a set of sequential code statements $\{i, i+1, i+2, \dots, i+n\}$, when the variable is written in statement i and it is last accessed in statement $(i+n)$. When a variable is written in a statement $(i+k)$ in the set, but last used in the same statement $(i+k)$ of the next iteration, it is assumed to live over the entire set;
2. represent the lifetime of each variable as a vertical line from statement i through statement $(i+n)$ in the column j reserved for the corresponding variable v_j ;
3. determine the maximum number N of overlapping lifetimes, computing the maximum number of vertical lines intersecting with any horizontal cut-line;
4. estimate the minimum number N of set of registers necessary to implement the code by using register sharing. Register sharing has to be applied whenever a group of variables, with the same bit-width b_i , can be mapped to the same register. The total number of

registers is given by
$$\sum_{i=1}^N b_i$$
;

5. select a possible mapping of variables into registers by using registers sharing;
6. compute the number w_i of write to the variables mapped to the same set of registers;
7. estimate α_i of each set of registers dividing w_i by the number of statements S :
 $\alpha_i = w_i / S$; hence, $TR_i = \alpha_i f_{CLK}$.

Figure 2 shows an application example of this algorithm, representing the differential

equation example reported in [17]. The bold dotted line at statement 7 represents the horizontal cut-line with the maximum number ($N = 9$) of vertical lines reaching or crossing it. Thus, using register sharing, the VHDL statements can be implemented with a minimum of 9 registers. A possible mapping of variables into registers is shown in Table 1.

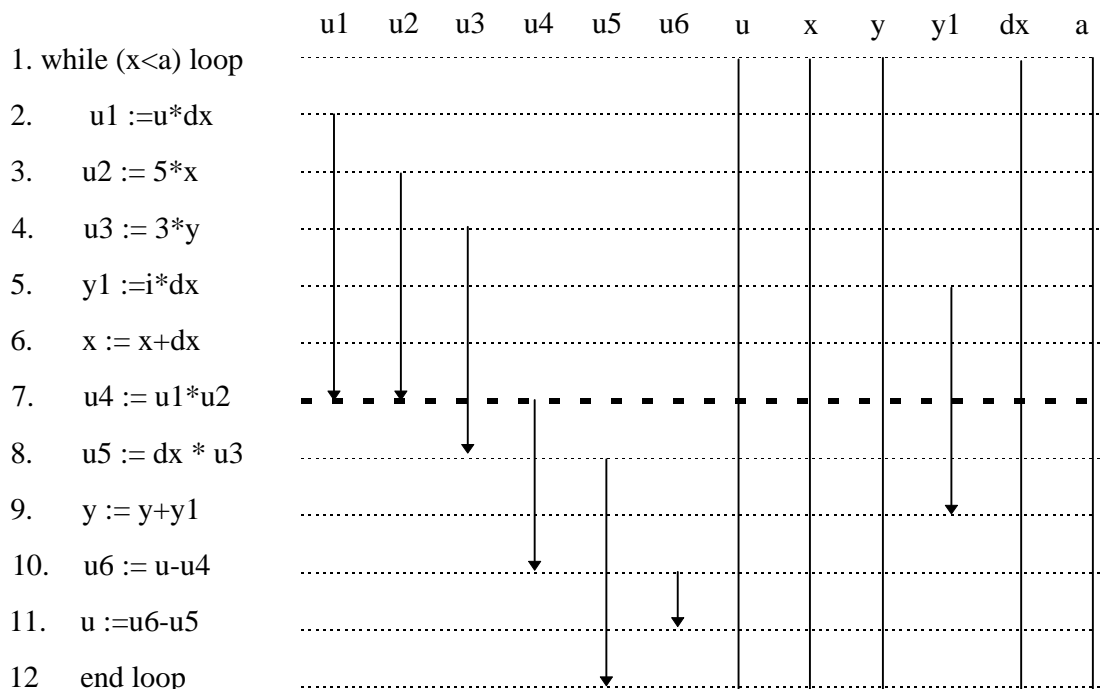


Figure 2: Live variable analysis for power estimation

| Register r_i | Variables mapped to r_i | Write number w_i | Switching act. α_i | Bit-width b_i |
|----------------|---------------------------|--------------------|---------------------------|-----------------|
| r_1 | u1, u4 | 2 | 1/6 | 1 |
| r_2 | u2, u5 | 2 | 1/6 | 1 |
| r_3 | u3 | 1 | 1/12 | 1 |
| r_4 | u6, y1 | 2 | 1/6 | 1 |
| r_5 | u | 1 | 1/12 | 1 |
| r_6 | x | 1 | 1/12 | 1 |
| r_7 | y | 1 | 1/12 | 1 |
| r_8 | dx | 1 | 1/12 | 1 |
| r_9 | a | 1 | 1/12 | 1 |

Table 1: Results of live variable analysis applied to the differential equation example

Regarding P_{REG} , it is worth noting that the power of latches and flip/flops is consumed not only during output transitions, but also during all clock edges by the internal clock buffers, depicted in Figure 3, even though the data stored in the register does not change. Thus, our analytical model of registers takes into account both the *switching* and *non-switching* power, the latter due to internal clock buffers. The non-switching power dissipated by internal clock buffers accounts for approximately the 30% of the average power of the registers, as for example in [14] for a cell-based CMOS 3.3V technology. Note that, as depicted in Figure 3, the internal clock buffers are independent of the output load, thus the non-switching power of latches and flip/flops is load-independent, but dependent on the clock input ramp time.

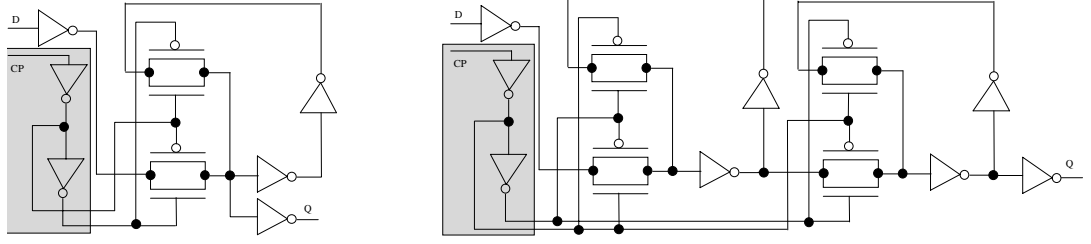


Figure 3: The latch and D flip/flop models for power estimation

Let the set of registers, S , be composed of N sets, such as: $S = \{s_1, s_2, \dots, s_k, \dots, s_N\}$, where the k -th set s_k is composed of the same type t_k of registers. Globally, the average power of the registers can be estimated as:

$$P_{REG} = \sum_{k=1}^N (P_{S_k} + P_{NS_k})$$

where P_{S_k} is the average power of each set s_k and P_{NS_k} is the average non-switching power dissipated by the internal clock buffers of the registers in the set s_k , that is the average power dissipated by the internal clock buffers when there are no output transitions.

Note that the measured average power P_{S_k} , tabulated in the target library, includes also the power dissipated by the internal clock buffers during clock edges corresponding to output transitions. Hence the estimated value of P_{S_k} should account for a toggle rate given by the TR_{S_k} , while the estimated values of the P_{NS_k} should consider a toggle rate of $(f_{CLK} - TR_{S_k})$.

The estimated values of P_{S_k} and P_{NS_k} , for the k -th set s_k , are respectively given by:

$$P_{S_k} = \sum_{i=1}^{n_k} P_i(C_i) TR_i$$

$$P_{NS_k} = P_{Ok} \sum_{i=1}^{n_k} (f_{CLK} - TR_i)$$

where:

n_k is the estimated number of registers in the set s_k ;

$P_i(C_i)$ is the average power consumption per MHz of the i -th register in s_k . The value of P_i has been computed running SPICE simulations, at a given reference frequency f_0 , for different output standard loads (representing both load cells and interconnections) and clock input ramp time. Thus the value of P_i is given as a function of the output load C_i and the input ramp time and it is tabulated in our allocation library in [μ W/MHz] as a function of C_i , expressed in equivalent standard load and input ramp time expressed in [nsec];

P_{Ok} is the non-switching power consumption per MHz of a single register of type t_k . The value of P_{Ok} expressed in [μ W/MHz] has been computed running SPICE simulations, at a given reference frequency f_0 , as a function of the clock input ramp time;

C_i is the estimated output load of the i -th register in the set s_k expressed in equivalent standard loads;

TR_i is the estimated toggle rate of the i -th register in the set s_k , obtained by using the live variable analysis.

7.2. P_{MUX} Estimation

To estimate the size and number of multiplexers from the VHDL code, it is necessary to determine the number of paths in the data-path. The approach is also based on the definition of the power model of a 2-input non-inverting multiplexer, based on both static signal probability of the selection net and the switching activities of the input nets.

The analysis of the design paths and the related notations are similar to those performed in [17], however in the proposed approach we consider also the paths from primary inputs to internal registers and from internal registers to primary outputs. A path from the *source* component S to the *target* component T is represented as $T < S$. Note that all memory accesses require the use of intermediate registers.

Given the target architecture represented in Figure 1, the possible paths can be classified in the following categories:

1. primary input to register ($R < I$);
2. register to primary output ($O < R$);
3. register to register ($R < R$);
4. register to functional unit ($U < R$);
5. functional unit to register ($R < U$);
6. register to memory ($M < R$);
7. memory to register ($R < M$).

The algorithm used to determine the possible paths in the data-path could be easily derived from the algorithm described in [17], but considering also the possible paths of the categories 1 and 2.

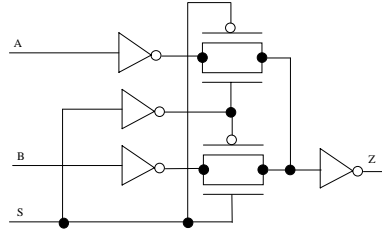


Figure 4: The 2-input non-inverting multiplexer model for power estimation.

Once the size and number of multiplexers has been computed, we derive the switching activity of the output node of each multiplexer, given the model of the two-input non-inverting multiplexer depicted in Figure 4. A simplified model for the maximum switching activity of the output Z of a 2-input non-inverting multiplexer is:

$$\alpha_Z = \alpha_A (1 - p_S^1) + \alpha_B p_S^1$$

where:

α_A is the switching activity of input A;

α_B is the switching activity of input B;

p_S^1 is the static signal probability of the selection net S.

Globally, the average power dissipated by the multiplexers can be estimated as:

$$P_{MUX} = \sum_{i=1}^N P_i$$

where N is the estimated number of multiplexers and P_i is the average power of each multiplexer.

The value of P_i for the i -th multiplexer is given by: $P_i = P_{ii} (C_i) TR_i$, where P_{ii} is the average power consumption per MHz of a 2-input non-inverting multiplexer and TR_i is the toggle rate of the output of the i -th multiplexer.

7.3. P_{FU} Estimation

For the estimation of the average power of the functional units, we use complexity-based analytical models [10], where the complexity of each functional unit is described in terms of equivalent gates. For the estimation of the number of equivalent gates necessary to implement a given function of the data-path, we use a library of macro-modules such as adders,

multipliers, etc.. The library should include the estimated number of logic gates for each macro-module, depending on the number of operands and the bit-width of each operand. Once the number of equivalent gates for each macro-function has been evaluated, the estimated power dissipated by the functional units can be expressed as:

$$P_{FU} = \sum_{i=1}^N P_i$$

where N is the number of macro-modules, and P_i is the power of the i -th macro-module given by:

$$P_i = n_i P_{TECH} TR_i$$

where P_{TECH} is a technological parameter expressed in [μ W/(gate MHz)]; n_i is the estimated number of logic gates in the i -th macro-function; TR_i is the toggle rate of the output net of the i -th macro-module.

8. P_{MEM} Estimation

A power dissipation model for a memory cell, at a low-level of abstraction, has been proposed in [25], being:

$$P_{memcell} = 2^k/2 (c_{int} l_{column} + 2^{n-k} C_{tr}) V_{dd} V_{swing} f_{clk}$$

where 2^k is the number of cells in a row, c_{int} is the wire capacitance per unit length, l_{column} is the memory column length, 2^{n-k} is the number of cells in a column, C_{tr} is the minimum size drain capacitance, and V_{swing} is the bitline voltage swing.

Considering a fully CMOS single port static RAM, at a high-level of abstraction, we assume to have in the target library the information related to the power consumption of a single memory cell P_{cell} and of a single memory output buffer.

The average power dissipation during a read access to a single row of the array, composed of n rows and m columns, is proportional to the inverse of the read access time t_a and to the sum of the average power dissipated by the following blocks: the row decoder, the m memory cells composing the i -th row and the output buffers.

In particular, the power dissipated by the row decoder can be estimated with a complexity-based model, where the number of equivalent gates is proportional to the product ($n \times lg_2 n$) and the load capacitance is the word line capacitance.

9. P_{CNTR} Estimation

The proposed model for power dissipation of a Finite State Machine (FSM) is a probabilistic model, where we approximate the average switching activities of the FSM nodes by using the switching probabilities (or transition probabilities) derived by modeling the FSM as a Markov chain [22]. Given a typical implementation of a FSM, composed of a combinational circuit and a set of state registers, as depicted in Figure 5, we consider the different contributions to the global average power:

$$P_{CNTR} = P_{IN} + P_{STATE_REG} + P_{COMB} + P_{OUT}$$

where:

P_{IN} is the average power dissipated by the primary inputs PI ;

P_{STATE_REG} is the average power dissipated by the state registers;

P_{COMB} is the average power dissipated by the combinational logic;

P_{OUT} is the average power dissipated by the primary outputs.

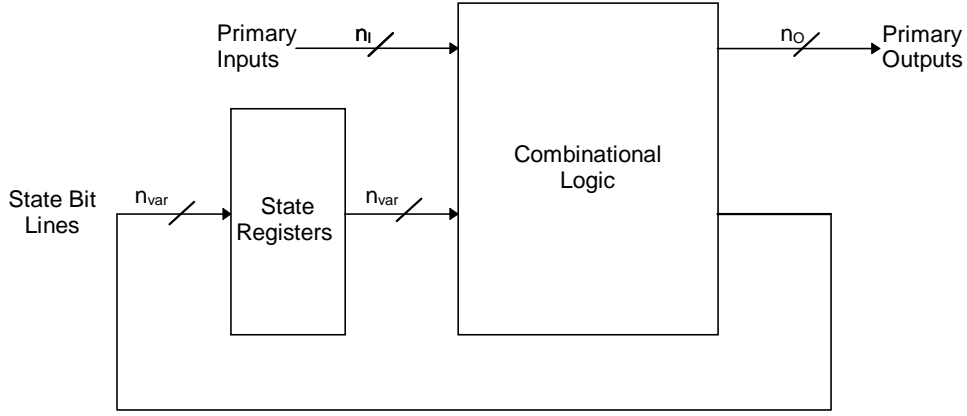


Figure 5: The Finite State Machine model for power estimation.

The power estimation models dealing with each term of the above equation are described in the following, along with some concepts and notations related to FSMs. As basic assumptions, we assume to have the FSM description available in the form of a State Transition Graph (STG), where each state is represented symbolically and nothing is known on the structure of the combinational logic implementing the next state and output functions. The input static signal probabilities and the input switching activity factors are supposed to be given from the system-level specifications, being derived by simulating the FSM at a high abstraction level or by direct knowledge of the typical input behavior.

Furthermore, we assume to use a Zero Delay Model for the logic gates and synchronous primary inputs. Under these assumptions, we can ignore the effects of glitches and hazards on the state bit lines, therefore the switching activity of the present and next state bit lines are equal.

9.1. Estimation of Total State Transition Probabilities

Given the FSM description and the input probabilities, the first step of our estimation consists of the computation of the total state transition probabilities for each edge in the graph, by modeling the FSM as a Markov chain and following the same method shown in [3, 15, 22].

Let the FSM, composed of n_s states, described by using a STG composed of n_s vertexes, corresponding to the states in the set $S = \{s_1, s_2, \dots, s_{n_s}\}$, and the related directed edges. The edges are labeled with the set of input configurations that cause a transition from the source state to the destination state. Considering a transition from state s_i to state s_j , we can compute the factor p_{ij} , called *conditional state transition probability*, that represents the conditional probability of the transition from state s_i to state s_j , given that the FSM was in state s_i [3]:

$$p_{ij} = \text{Prob}(\text{Next} = s_j \mid \text{Present} = s_i).$$

The computation of the p_{ij} 's can be carried out as in [15], assuming totally independent primary inputs $PI = \{x_1, x_2, \dots, x_k, \dots, x_{n_i}\}$ and being p_{xk} the static signal probability of input x_k . Let $f_{ij}(x_1, x_2, \dots, x_k, \dots, x_{n_i})$ be the Boolean function assuming the value one when a transition from state s_i to state s_j occurs and let $c^l(x_1, x_2, \dots, x_k, \dots, x_{n_i})$ be the l -th cube in the cover of f_{ij} . The values of p_{ij} 's can be expressed as:

$$p_{ij} = \sum_{c^l} \prod_{x_k \in c^l} Q_{xk}$$

where:

$$Q_{xk} = \begin{cases} p_{xk} & \text{if } x_k \text{ appears with positive phase in } c^l \\ 1 - p_{xk} & \text{if } x_k \text{ appears with negative phase in } c^l \\ 1 & \text{if } x_k \text{ does not appear in } c^l \end{cases}$$

Even if the conditional state transition probabilities can be considered as an approximation of the total state transition probabilities, the steady-state probabilities should be taken into account.

The *steady-state probability* P_i of a state s_i is defined as the probability to be in the state s_i in an arbitrarily long random sequence [27]. Computing the P_i 's implies solving the system composed of the Chapman-Kolmogorov equations [22] and the equation representing the normality condition:

$$P^T = P^T p$$

$$\sum_{i=1}^{n_s} P_i = 1$$

where $P^T = (P_1, \dots, P_k, \dots, P_{n_s})$ is the row vector of the steady-state probabilities and p is the matrix of the conditional state transition probabilities p_{ij} . Note that the above system has $(n_s + 1)$ equations and n_s unknowns, thus one of the Chapman-Kolmogorov equations can be dropped [15].

Given the state probabilities P_i 's and the conditional state transition probabilities p_{ij} 's, the *total state transition probabilities* P_{ij} between the two states s_i and s_j can be expressed as [3]:

$$P_{ij} = p_{ij} P_i.$$

9.2. State Encoding Algorithms

The second step consists of finding a state assignment that minimizes the power dissipation. Given the STG, the problem can be formulated as determining a state encoding so as to minimize a given cost function, C , that takes into account the number of state variable transitions between two consecutive clock cycles. The main goal is to minimize the switching activity associated with the state registers that, if combined with an appropriate combinational logic implementation, can lead to a global power minimization. Several solutions to the state encoding problem have been presented in literature [15].

Specific solutions can be applied for particular classes of STGs, such as the Gray encoding for STG representing structures such as counters. For a STG of generic structure, the One-Hot encoding guarantees exactly two state bit transitions for each clock cycle, however it requires a number of state variables exactly equal to the number of states ($n_{var} = n_s$), while in general $\lceil \lg_2 n_s \rceil \leq n_{var} \leq n_s$. Other coding techniques [27, 3, 15] can lead to a single state bit transition for each clock cycle.

In general, the cost function C should take into account the Hamming distance $H(s_i, s_j)$ between the binary codes of state s_i and s_j among which a state transition can occur [15]:

$$C = \sum_{i, j} H(s_i, s_j)$$

However, a more accurate cost function should consider weight factors taking into account the probability of the state transitions:

$$C = \sum_{i, j} W_{ij} H(s_i, s_j)$$

where W_{ij} is the weight assigned to the edge from state s_i to state s_j in the STG.

The Syclop method, proposed in [23], considers the conditional state transition probabilities p_{ij} as the W_{ij} coefficients in the cost function C and uses the minimum possible number of state bits ($\lceil \lg_2 n_s \rceil$).

The state assignment algorithm POW3, proposed in [3], considers the total state transition probabilities P_{ij} to be included in the cost function C as weight coefficients.

Other state encoding algorithms are the Galops algorithm proposed in [21], which uses the

same cost function as POW3, and the LPSA algorithm, proposed in [27], that addresses the state assignment problem for both the two-level and the multi-level implementations of the next state and output logic, accounting the loading factors and the switching activities of the present state inputs. A related power cost model to guide the state assignment has also been proposed in [27].

9.3. Estimation of the Switching Activity of the State Bit Lines

The switching activity of the state bit lines, depends on both the state encoding and the total state transition probabilities between each pair of states in the STG [27].

Let us generalize the concept of state transition probability to transitions occurring between two distinct sub-sets of disjoint states, S_i and S_j , contained in the set of states $S = \{s_1, s_2, \dots, s_{ns}\}$, as defined in [27]:

$$TP(S_i \leftrightarrow S_j) = \sum_{s_i \in S_i} \sum_{s_j \in S_j} (P_{ij} + P_{ji})$$

Being b^i the i -th bit ($1 \leq i \leq n_{var}$) of the state code (called state bit) and n_{var} the number of state bits ($\lceil \lg_2 n_s \rceil \leq n_{var} \leq n_s$), we consider the two sets of sub-states in which the i -th state bit assumes the value one and zero respectively. The switching activity α_b^i of the state bit line b^i is given by [27]:

$$\alpha_b^i = TP(States(b^i = 1) \leftrightarrow States(b^i = 0))$$

9.4. Estimation of the Switching Activity of the Primary Outputs

Considering a Moore-type FSM, the switching activity of the primary outputs can be defined similarly to the switching activity of the state bit lines, depending on both the given output encoding and the total state transition probabilities. In fact, in a Moore-type FSM, the total state transition probabilities P_{ij} between the two states s_i and s_j are equal to the total transition probabilities between the corresponding outputs o_i and o_j , where the output row vector o_i ($i = 1, 2, \dots, n_o$) is composed of the n_o primary outputs: $(y_{i1}^1, \dots, y_{i1}^{n_o}, \dots, y_{i2}^1, \dots, y_{i2}^{n_o}, \dots, y_{in_o}^1, \dots, y_{in_o}^{n_o})$.

Let us define the transition probability of the transitions occurring between two distinct sub-sets of disjoint outputs, O_i and O_j , contained in the set of the outputs $O = \{o_1, o_2, \dots, o_{ns}\}$, as:

$$TP(O_i \leftrightarrow O_j) = \sum_{o_i \in O_i} \sum_{o_j \in O_j} (P_{ij} + P_{ji})$$

Being y^m the m -th output bit ($1 \leq m \leq n_o$) and n_o the number of primary outputs, we consider the two sets of outputs in which the m -th output bit assumes the value one and zero respectively. The switching activity α_{y^m} of the primary outputs y_m is given by:

$$\alpha_{y^m} = TP(Outputs(y^m = 1) \leftrightarrow Outputs(y^m = 0))$$

9.5. P_{IN} Estimation

As mentioned before, let us assume that the input static signal probabilities and the input switching activity factors are given from the system-level specifications.

The average power dissipated by the k -th primary input belonging to the set $PI = \{x_1, x_2, \dots, x_k, \dots, x_n\}$ depends on the switching activity factors α_{xk} and the input load capacitance C_{xk} , the latter being proportional to the number of literals, n_{litxk} , that the k -th primary input is driving in the combinational part, and the estimated capacitance C_{lit} due to each literal [27]. Therefore, the average power P_{IN} can be estimated as:

$$P_{IN} = \sum_{x_k \in PI} P_{xk} (C_{xk}) TR_{xk}$$

where: $C_{xk} = n_{litxk} C_{lit}$; $TR_{xk} = \alpha_{xk} f_{CLK}$ and $P_{xk}(C_{xk})$ is the average power consumption per MHz of the cell driving the k -th input.

9.6. P_{STATE_REG} Estimation

The average power dissipated by the state registers, P_{STATE_REG} , can be derived by using the switching activity α_{bi} of the i -th state bit line b_i , where $1 \leq i \leq n_{var}$ and the corresponding toggle rate is $TR_{bi} = \alpha_{bi} f_{CLK}$. The term P_{STATE_REG} accounts for the switching and non-switching power of the state registers:

$$P_{STATE_REG} = \sum_{i=1}^{n_{var}} (P_i + P_{NSi})$$

where n_{var} is the number of state registers and P_i and P_{NSi} are the average switching and non-switching power dissipated by each state register. As before, the switching power P_i includes also the power dissipated by the internal clock buffers, during clock edges corresponding to output transitions. Hence the terms P_i should account for a toggle rate given by TR_{bi} , while the terms P_{NSi} should consider a toggle rate of $(f_{CLK} - TR_{bi})$.

The estimated values of P_i and P_{NSi} are respectively given by:

$$P_i = P_{ti}(C_i) TR_{bi}$$

and

$$P_{NSi} = P_{0i}(f_{CLK} - TR_{bi})$$

where:

P_{ti} is the average power consumption per MHz of the i -th register of type t_i as a function of the load capacitance C_i and the input ramptime;

P_{0i} is the non-switching power consumption per MHz of a single register of type t_i ;

$C_i = n_{litbi} C_{lit}$ is proportional to the number of literals, n_{litbi} , that the i -th state bit line is driving in the combinational part, and the estimated capacitance C_{lit} due to each literal, expressed in equivalent standard loads.

9.7. P_{COMB} Estimation

The average power dissipated by the combinational logic P_{COMB} has been estimated by considering a two-level logic implementation, before the minimization step. The i -th state bit line b_i (where $1 \leq i \leq n_{var}$) can be expressed by using the canonical form as the sum of N_{bi} minterms ($N_{bi} \leq 2^{n_{lit}}$ where n_{lit} is the number of literals and $2^{n_{lit}}$ is the maximum number of minterms). Similarly, the m -th output bit y^m ($1 \leq m \leq n_o$) can be expressed in the canonical form as the sum of N_{ym} minterms ($N_{ym} \leq 2^{n_{lit}}$).

Let us assume to use a single AND gate to represent the generic minterm, hence the maximum number of AND gates in the AND-plane is $2^{n_{lit}}$, while in general $n_{AND} \leq 2^{n_{lit}}$. Given the probabilistic model of the switching activity of the generic n_{lit} -input AND gate, we can derive an upper bound for the estimated power of the AND-plane:

$$P_{COMB} = \sum_{i=1}^{n_{AND}} P_i(C_i) TR_i$$

where:

$P_i(C_i)$ is the average power consumption per MHz of the i -th n_{lit} -input AND gate;

C_i is the capacitance driven by the i -th n_{lit} -input AND gate;

$TR_i = \alpha_i f_{CLK}$ is the toggle rate of the i -th n_{lit} -input AND gate (derived by using the switching activity model of the n_{lit} -input AND gate).

9.8. P_{OUT} Estimation

P_{OUT} is the average power dissipated by the OR-plane, that is composed of n_{var} N_{bi} -input OR gates corresponding to the state bit lines, driving the input capacitance of the state registers, and n_O N_{ym} -input OR gates corresponding to the primary outputs, driving the output load capacitances.

Therefore, the upper bound for the power of the OR-plane is composed of two terms. The first term is thus proportional to the switching activity factors α_{bi} of the state bit line b_i , while the second term is proportional to the switching activity factors α_{yi} of the primary outputs:

$$P_{OUT} = \sum_{i=1}^{n_{var}} P_i (C_{IN_REG}) TR_{bi} + \sum_{i=1}^{n_O} P_i (C_{yi}) TR_{yi}$$

where:

$P_i (C_{IN_REG})$ is the average power consumption per MHz of the i -th N_{bi} -input OR gate driving the i -th state bit line;

C_{IN_REG} is the input capacitance of each state register;

$TR_{bi} = \alpha_{bi} f_{CLK}$ is the toggle rate of the i -th state bit line b_i ;

$P_i (C_{yi})$ is the average power consumption per MHz of the i -th N_{yi} -input OR gate driving the i -th primary output;

C_{yi} is the output load capacitance of the i -th primary output;

$TR_{yi} = \alpha_{yi} f_{CLK}$ is the toggle rate of the i -th primary output.

10. P_{PROC} estimation

A methodology to measure the power cost of embedded software, at the instruction-level, has been proposed in [26]. The current drawn by each processor instruction has been measured, during the execution of instruction sequences composed of the same instruction. The power contributions due to the inter-instruction effects have also been considered, along with the effects of resource constraints leading to stalls, such as pipeline and write buffer stalls, as well as the effects of cache misses, causing power penalties.

Considering the embedded core processor in our target system architecture, the proposed estimation is carried out at the instruction-level, by considering the average power consumption during the execution of a given program. We assume the knowledge on detailed power information provided by the embedded core supplier, in terms of the power dissipated by each type of instruction in the instruction-set. Based on this information, a power table should be derived for a dedicated processor, reporting the power consumption for each instruction in the instruction-set and for all the possible addressing mode for a given instruction type.

11. Experimental results and concluding remarks

The proposed power estimation method has been implemented and applied to both data-path and FSM circuits. The measures have been derived by using the HCMOS6 technology, featuring 0.35 μ m and 3.3 V, supplied by SGS-Thomson Microelectronics at the target operating frequency of 100 MHz.

The architecture of the data-path ASIC, reported in Figure 6, contains registers, a 64-bit adder, I/O pads, a set of 64 multiplexers and a clock distribution tree. The VHDL model of the ASIC, reported in the figures 7 and 8, has been synthesized by using the Synopsys

Design Compiler tool with the HCMOS6 technology.

Experimental results are reported in Table 2, in terms of the average power dissipated by the different parts of the ASIC, by considering several input switching activities: 0.75, 0.5, 0.25 and 0.1. The results obtained by the proposed methodology have been compared to the results obtained through the Synopsys Design Power tool, based on the gate-level netlist. Note that both the estimation methods are based on a Zero Delay Model. As far the global power is concerned, the proposed method provides a good approximation: the percentage error belongs to the range 1.12%-1.47% with respect to the Synopsys estimates. However, being the switched capacitance of I/O nodes usually larger than the switched capacitance of the internal nodes up to three orders of magnitude, the major contribution to the global power is constituted by the I/O power. In the benchmark, the I/O power represents the 94.83%, on average, of the total power, due to the reduced size of the core logic. Furthermore, the I/O power estimates are very close to the gate-level estimates (1.38% on average), due to the simple model used. Thus, to verify the accuracy of the proposed model, a more realistic measure is represented by the comparison of the core power: the model provides an estimation error below 4.54%. In particular, the results show an average percentage error of 2.07% for the registers, 30.49% for the multiplexers and -0.54% for the adder.

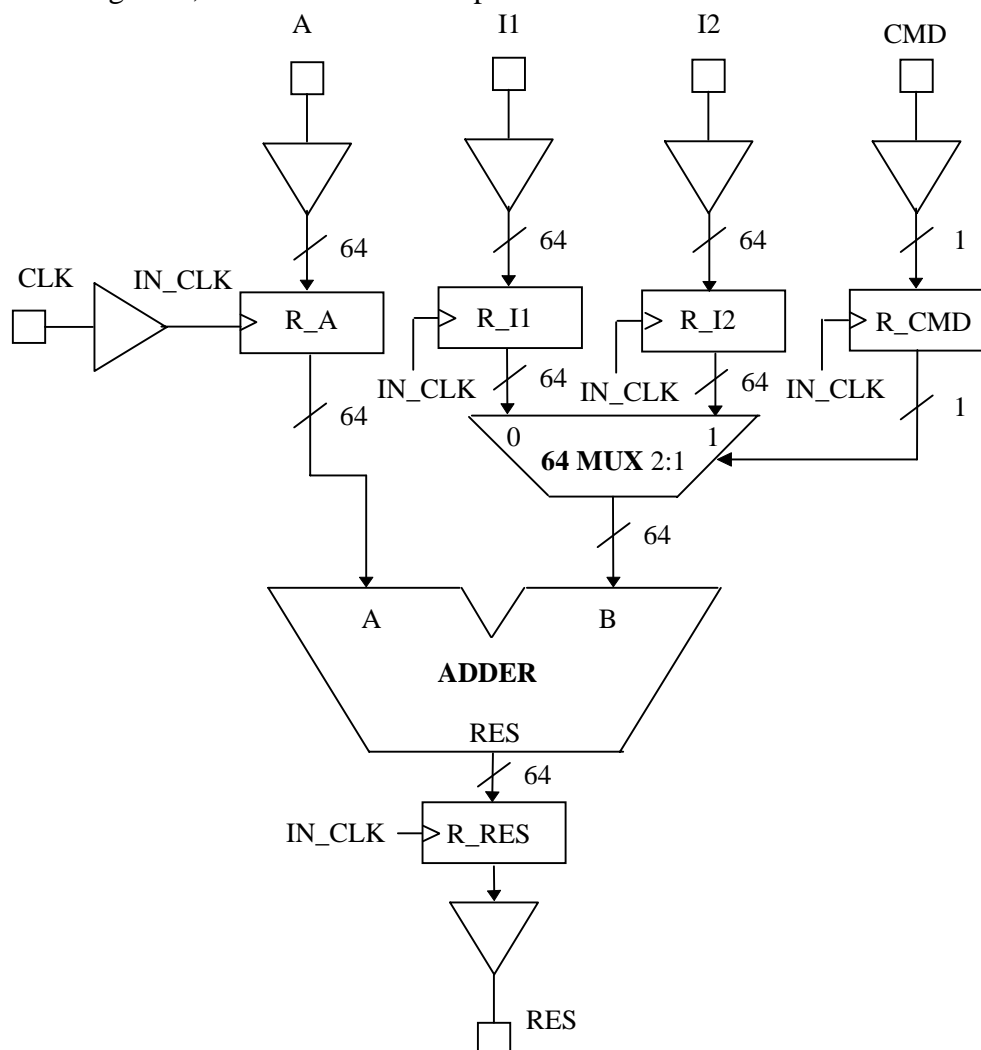


Figure 6: The ASIC architecture of the power estimation benchmark.

To show the validity of the proposed FSM power model, we consider the same Moore-type FSM used in [3]. The State Transition Table of the four-states and two-inputs FSM is reported in Table 3, with an arbitrary output encoding. Several state encodings have been

applied to the FSM (see Table 4), to evaluate the effects of the state encoding on the power estimates. In particular, we derived the ENC_A state encoding to minimize power, ENC_B is the state encoding proposed in [3], ENC_C has been derived by using NOVA to minimize the area, ENC_D and ENC_E are randomly generated encodings and ENC_F is an example of the One-Hot encoding. As before, the results obtained by the proposed model have been compared to the results obtained by the Design Power tool on the gate-level netlist (see Table 5).

Considering the effects of the different encoding algorithms on the global power estimates, a similar trend can be observed both for Design Power and the proposed model. As expected, the Design Power measurements show a growing power dissipation from ENC_A to ENC_F. Our measurements, as reported in Table 5, show a rather similar behavior.

Considering the global power, the proposed model shows an average percentage error of -2.31% (ranging from -8.68% to 3.0%) with respect to the Design Power estimates. However, there is an over-estimation of the power of the state-registers of 27.9%, on average.

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
entity CORE is
  Port ( CLK : In  std_logic;
        A  : In  std_logic_vector (0 to 63);
        I1 : In  std_logic_vector (0 to 63);
        I2 : In  std_logic_vector (0 to 63);
        CMD : In  std_logic;
        RES : Out std_logic_vector (0 to 63) );
end CORE;
architecture BEHAVIORAL of CORE is
signal s_A  : std_logic_vector(0 to 63);
signal s_I1 : std_logic_vector(0 to 63);
signal s_I2 : std_logic_vector(0 to 63);
signal s_CMD : std_logic;
signal mux_out : std_logic_vector(0 to 63);
signal add_out : std_logic_vector(0 to 63);
signal cmd_pi : std_logic;
signal cmd_po : std_logic;
signal clk_pi : std_logic;
signal clk_po : std_logic;
signal a_pi : std_logic_vector(0 to 63);
signal a_po : std_logic_vector(0 to 63);

constant zero : std_logic := '0';
constant one  : std_logic := '1';
begin
free_regs:process
begin
wait until CLK'event and CLK='1';
s_A <= A;
s_I1 <= I1;
s_I2 <= I2;
s_CMD <= CMD;
RES <= add_out;
end process;
multiplexor:process(s_I1,s_I2,s_CMD)
begin
if (s_CMD = zero) then
mux_out <= s_I1 ;
else
mux_out <= s_I2 ;
end if;
end process;
adder:process(s_A,mux_out)
begin
add_out <= s_A + mux_out;
end process;
end BEHAVIORAL;

```

Figure 7: The VHDL code describing the core of the ASIC depicted in Fig.5.

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
entity PEB is
  Port ( CLK : In  std_logic;
        A  : In  std_logic_vector (0 to 63);
        I1 : In  std_logic_vector (0 to 63);
        I2 : In  std_logic_vector (0 to 63);
        CMD : In  std_logic;
        RES : Out std_logic_vector (0 to 63) );
end PEB;
architecture BEHAVIORAL of PEB is
signal in_A  : std_logic_vector(0 to 63);
signal in_I1 : std_logic_vector(0 to 63);
signal in_I2 : std_logic_vector(0 to 63);
signal in_CMD : std_logic;
signal in_CLK : std_logic;
signal s_out : std_logic_vector(0 to 63);
signal cmd_pi : std_logic;
signal cmd_po : std_logic;
signal clk_pi : std_logic;
signal clk_po : std_logic;
signal a_pi : std_logic_vector(0 to 63);
signal a_po : std_logic_vector(0 to 63);
signal i1_pi : std_logic_vector(0 to 63);
signal i1_po : std_logic_vector(0 to 63);
signal i2_pi : std_logic_vector(0 to 63);
signal i2_po : std_logic_vector(0 to 63);
constant zero : std_logic := '0';
constant one  : std_logic := '1';
component DRVC4F
  Port ( A : In  std_logic;
        PI : In  std_logic;
        PO : Out std_logic;
        Z : Out std_logic );
end component;
component IBUFF
  Port ( A : In  std_logic;
        PI : In  std_logic;
        PO : Out std_logic;
        Z : Out std_logic );
end component;
component B8
  Port ( A : In  std_logic;
        Z : Out std_logic );
end component;
component CORE
  Port ( CLK : In  std_logic;
        A  : In  std_logic_vector (0 to 63);
        I1 : In  std_logic_vector (0 to 63);
        I2 : In  std_logic_vector (0 to 63);
        CMD : In  std_logic;
        RES : Out std_logic_vector (0 to 63) );
end component;
begin
CORE_LOGIC : CORE Port Map (CLK=>in_CLK,
A=>in_A,
I1=>in_I1,
I2=>in_I2,
CMD=>in_CMD,
RES=>s_out );
B_CLK : DRVC4F Port Map ( A => CLK, Z => in_CLK, PI => clk_pi,
PO => clk_po );
B_CMD : IBUFF Port Map ( A => CMD, Z => in_CMD, PI => cmd_pi,
PO => cmd_po );
gbuff_A : for i in A'range generate
B_A : IBUFF
Port Map ( A => A(i), Z => in_A(i), PI => a_pi(i), PO => a_po(i) );
end generate;
gbuff_I1 : for i in I1'range generate
B_I1 : IBUFF
Port Map ( A => I1(i), Z => in_I1(i), PI => i1_pi(i), PO => i1_po(i) );
end generate;
gbuff_I2 : for i in I2'range generate
B_I2 : IBUFF
Port Map ( A => I2(i), Z => in_I2(i), PI => i2_pi(i), PO => i2_po(i) );
end generate;
gbuff_RES : for i in RES'range generate
B_RES : B8
Port Map ( A => s_out(i), Z => RES(i) );
end generate;
end BEHAVIORAL;

```

Figure 8: The VHDL code describing the I/O pads and the clock tree of the ASIC depicted in Fig. 5.

| Power [mW] | Input Switching Activity | Input Pads | Reg. | Muxes | Adder | Output Pads | Core Logic | I/O Pads | Total |
|------------------|--------------------------|------------|--------|--------|--------|-------------|------------|----------|---------|
| Design Power | 0.75 | 10.23 | 21.07 | 1.9 | 15.59 | 1230.8 | 38.56 | 1241.03 | 1279.59 |
| Estimation | | 10.13 | 22.40 | 2.4 | 15.51 | 1248.0 | 40.31 | 1258.13 | 1298.44 |
| Perc. Error | | -0.95% | 6.31% | 26.32% | -0.51% | 1.4% | 4.54% | 1.38% | 1.47% |
| Design Power | 0.5 | 6.76 | 18.51 | 1.22 | 10.39 | 820.55 | 30.12 | 827.31 | 857.43 |
| Estimation | | 6.76 | 19.20 | 1.6 | 10.34 | 832.0 | 31.14 | 838.76 | 869.90 |
| Perc. Error | | 0.0% | 3.73% | 31.15% | -0.48% | 1.4% | 3.39% | 1.38% | 1.45% |
| Design Power | 0.25 | 3.47 | 15.94 | 0.61 | 5.2 | 410.27 | 21.75 | 413.74 | 435.49 |
| Estimation | | 3.38 | 16.00 | 0.8 | 5.17 | 416.0 | 21.97 | 419.38 | 441.35 |
| Perc. Error | | -2.67% | 0.38% | 31.15% | -0.58% | 1.4% | 1.01% | 1.36% | 1.35% |
| Design Power | 0.1 | 1.35 | 14.39 | 0.24 | 2.08 | 164.11 | 16.71 | 165.46 | 182.17 |
| Estimation | | 1.35 | 14.08 | 0.32 | 2.07 | 166.40 | 16.47 | 167.75 | 184.22 |
| Perc. Error | | 0.0% | -2.15% | 33.33% | -0.58% | 1.4% | -1.45% | 1.38% | 1.12% |
| Ave. Perc. Error | | -0.9% | 2.07% | 30.49% | -0.54% | 1.4% | 1.87% | 1.38% | 1.35% |

Table 2: Power estimation comparison results for the data-path ASIC.

| Inputs | Present State | Next State | Outputs |
|--------|---------------|------------|---------|
| 01 | ST1 | ST2 | 11 |
| -0 | ST1 | ST2 | 11 |
| 11 | ST1 | ST1 | 00 |
| 01 | ST2 | ST3 | 01 |
| -0 | ST2 | ST3 | 01 |
| 11 | ST2 | ST1 | 00 |
| -1 | ST3 | ST4 | 10 |
| 10 | ST3 | ST4 | 10 |
| 00 | ST3 | ST3 | 01 |
| 0- | ST4 | ST3 | 01 |
| 1- | ST4 | ST2 | 11 |

Table 3: State Transition Table of the FSM.

| | ENC_A | ENC_B | ENC_C | ENC_D | ENC_E | ENC_F |
|-----|-------|-------|-------|-------|-------|-------|
| ST1 | 01 | 00 | 00 | 10 | 11 | 1000 |
| ST2 | 00 | 01 | 11 | 01 | 00 | 0100 |
| ST3 | 10 | 11 | 01 | 11 | 01 | 0010 |
| ST4 | 11 | 10 | 10 | 00 | 10 | 0001 |

Table 4: Different State Encodings for the FSM.

Globally, considering the data-path and FSM benchmarks, the relative accuracy of our approach compared with the Design Power gate-level tool is considered satisfactory at this level of abstraction. Traditional post-synthesis gate-level methods suffer from a main drawback with respect to our approach: the need to perform time-consuming tasks such as the synthesis. On the contrary, our approach, by avoiding to move down to the gate-level description, represents an innovative methodology encompassing the requirements to achieve accurate power estimation in a reasonable design time.

In conclusion, the proposed analysis affords the problem of power estimation for embedded systems implemented into a single ASIC described by using VHDL at the behavioral/RT levels, before performing the synthesis and avoiding gate-level time consuming simulations. The main goal has been to offer a conceptual model and some power metrics to compare different design solutions described at high abstraction levels. Experimental results have shown sufficient relative accuracy with respect to gate-level power estimates. In addition the

proposed estimation procedure is not time consuming. Finally, work is in progress aiming at integrating the proposed conceptual framework and the related power metrics to guide the partitioning task of a more general hardware-software co-design environment for embedded systems [2].

| Power [μ W] | Encoding Type | State Reg. | Total |
|---------------------|------------------|---------------|--------|
| Design Power | ENC_A | 134 | 419.77 |
| Estimation | | 169.31 | 432.39 |
| Perc. Error | | 26.35% | 3.00% |
| Design Power | ENC_B | 133 | 439.5 |
| Estimation | | 169.31 | 432.39 |
| Perc. Error | | 27.3% | -1.62% |
| Design Power | ENC_C | 156 | 470.45 |
| Estimation | | 197.03 | 483.62 |
| Perc. Error | | 26.3% | 2.8% |
| Design Power | ENC_D | 154 | 516.99 |
| Estimation | | 197.03 | 483.93 |
| Perc. Error | | 27.94% | -6.39% |
| Design Power | ENC_E | 155 | 529.57 |
| Estimation | | 197.03 | 483.62 |
| Perc. Error | | 27.12% | -8.68% |
| Design Power | ENC_F | 240 | 584.6 |
| Estimation | | 317.83 | 567.34 |
| Perc. Error | | 32.43% | -2.95% |
| Ave. Perc. Error | | 27.9% | -2.31% |

Table 5: Power estimation comparison results for the FSM.

12. References

- [1] A.V. Aho, R. Sethi, J.D. Ullman, Compilers: Principles, Techniques, and Tools, (Addison-Wesley, 1986).
- [2] A. Balboni, W. Fornaciari, D. Sciuto, TOSCA: a pragmatic approach to co-design automation of control dominated systems, Hardware/Software Co-design, NATO ASI Series, Series E: Applied Sciences, Vol. 310 (Kluwer Academic Publisher, 1996) 265-294.
- [3] L. Benini and G. De Micheli, State Assignment for Low Power Dissipation, IEEE Journal of Solid State Circuits, 30 (3) (1995) 258-268.
- [4] T.D. Burd and R.W. Brodersen, Energy Efficient CMOS Microprocessor Design, in: Proc. 28th Hawaii International Conference on System Sciences, (Hawaii, 1995).
- [5] A. P. Chandrakasan and R. W. Brodersen, Minimizing Power Consumption in Digital CMOS Circuits, Proceedings of the IEEE, 83 (4) (1995) 498-523.
- [6] G. De Micheli, Hardware/Software Co-Design: Application Domains and Design Technologies, Hardware/Software Co-design, NATO ASI Series, Series E: Applied Sciences, Vol. 310 (Kluwer Academic Publisher, 1996).
- [7] S. Devadas and S. Malik, A Survey of Optimization Techniques Targeting Low Power VLSI Circuits, in: Proc. 32nd IEEE Design Automation Conference, (1995).
- [8] W. Fornaciari, P. Gubian, D. Sciuto, C. Silvano, A conceptual analysis framework for low power design of embedded systems, in: Proc. IEEE International Conference Innovative

System In Silicon, (Austin, TX, 1996).

- [9] D. Harel et al., STATEMATE: A Working Environment for the Development of Complex Reactive Systems, *IEEE Trans. on Software Engineering*, 16(4) (1990) 403-414.
- [10] P. Landman, High-Level Power Estimation, in: *Proc. ISLPED'96, International Symposium on Low Power Electronics and Design*, (Monterey, CA, 1996) 29-35.
- [11] P.E. Landman and J.M. Rabaey, Power Estimation for High-Level Synthesis, in: *Proc. EDAC-EUROASIC '93 European Design Automation Conference*, (Paris, 1993) 361-366.
- [12] P.E. Landman and J.M. Rabaey, Architectural Power Analysis: The Dual Bit Type Method, *Trans. on Very Large Scale Integration (VLSI) Systems*, 3(2) (1995).
- [13] P.E. Landman and J.M. Rabaey, Activity-Sensitive Architectural Power Analysis, *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 15(6) (1996) 571-587.
- [14] LCB500K, Cell-Based ASIC Design Manual, (LSI Logic Corporation, June 1995).
- [15] E. Macii, Sequential Synthesis and Optimization for Low Power, Notes of the NATO-ASI Course on Low Power in Deep Submicron Electronics, (Kluwer Academic Publisher, 1996).
- [16] R. Mehra and J. Rabaey, Behavioral Level Power Estimation and Exploration, in: *Proc. First International Workshop on Low Power Design*, (Napa Valley, CA, 1994) 197-202.
- [17] S. Narayan and D.D. Gajski, Area and Performance Estimation from System-Level Specifications, Technical Report ICS-92-16, Dept. of Information and Computer Science, University of California, Irvine, December 20, 1992.
- [18] V. Nagasamy, N. Berry and C. Dangelo, Specification, Planning, and Synthesis in a VHDL Design Environment, *IEEE Design & Test of Computers*, (1992) 58-68.
- [19] F.N. Najm, R. Burch, P. Yang and I.N. Hajj, Probabilistic Simulation for Reliability Analysis of CMOS VLSI Circuits, *IEEE Trans. on Computer-Aided Design*, 9(4) (1990) 439-450, (Errata in July 1990).
- [20] F.N. Najm, A Survey of Power Estimation Techniques in VLSI Circuits, *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 2(4) (1994) 446-455.
- [21] E. Olson and S.M. Kang, Low-Power State Assignment for Finite State Machines, in: *Proc. IWLPD'94: ACM/IEEE International Workshop on Low Power Design*, (Napa Valley, CA, 1994) 63-68.
- [22] A. Papoulis, Probability, Random Variables and Stochastic Processes, (McGraw-Hill, 1984).
- [23] K. Roy and S.C. Prasad, Syclop: Synthesis of CMOS Logic for Low Power Application, in: *Proc. ICCD'92 IEEE International Conference on Computer Design*, (1992) 464-467.
- [24] D. Singh, J. Rabaey, M. Pedram, F. Catthoor, S. Rajgopal, N. Sehgal, T. Mozden, Power Conscious CAD Tools and Methodologies: A Perspective, *Proceedings of the IEEE*, 83(4) (1995) 570-594.
- [25] C. Svensson and D. Liu, A Power Estimation Tool and Prospects of Power Savings in CMOS VLSI Chips, in: *Proc. First Int. Workshop on Low Power Design*, (Napa Valley, CA, 1994) 171-176.
- [26] V. Tiwari, S. Malik and A. Wolfe, Power Analysis of Embedded Software: a First Step towards Software Power Minimization, *IEEE Trans. on VLSI Systems*, (1994).
- [27] C.Y. Tsui, M. Pedram, C.A. Chen and A.M. Despain, Low Power State Assignment Targeting Two- and Multi-Level Logic Implementations, in: *Proc. ICCAD'1994: IEEE/ACM International Conference on Computer Aided Design*, (1994) 82-87.
- [28] W.H. Wolf, Hardware-Software Co-design of Embedded Systems, *Proceedings of the IEEE*, 82(7) (1994) 967-989.