Characterization-Free Behavioral Power Modeling

Alessandro Bogliolo Luca Benini† Giovanni De Micheli†

DEIS - University of Bologna
Bologna, I-40136

†CSL - Stanford University
Stanford, CA-94305

Abstract

We propose a new approach to RT-level power modeling for combinational macros, that does not require simulation-based characterization. A pattern-dependent power model for a macro is analytically constructed using only structural information about its gate-level implementation. The approach has three main advantages over traditional techniques: i) it provides models whose accuracy does not depend on input statistics, ii) it offers a wide range of trade-off between accuracy and complexity, and iii) it enables the construction of pattern-dependent conservative upper bounds.

1 Introduction

Modeling power consumption at high level of abstraction is a challenging issue that has received considerable attention in the last few years. Several efforts have focused on the development of power models for combinational macros which are the functional building blocks of complex designs described at the register-transfer level (RTL). As a result, several power models have been proposed ranging from constant estimators of the average power consumption [1] to advanced models that try to capture the dependence of power dissipation on the input patterns applied to the circuit [2, 3].

The dependence on input pattern distribution is a well-known property of power consumption in CMOS circuits. Advanced power models take this effect into account. We can distinguish two classes of approaches. *Pattern-independent* models provide average power estimates based on a compact description of the input statistics, *i.e.*, input transition activities and static probabilities [4, 5]. In contrast, *pattern-dependent* models provide a pattern-by-pattern power estimate during simulation [2, 3]. Pattern-dependent models are potentially more accurate than pattern-independent ones, and they can be used to estimate *peak* power as well as average power dissipation. On the other hand, they are more complex and may require long simulation runs to provide significant estimates.

Both pattern-dependent and pattern-independent models require *characterization*. Characterization consists of tuning model parameters in order to fit a sample of power consumption data provided by a gate-level (or swicth-level) simulation of the implementation of the macro. If the RTL design is based on a given library of functional macros, characterization is performed once for all for each macro in the library and the resulting model is used to backannotate its functional description. During design exploration and RTL simulation, the same model is used to provide power estimates for all instances of the same macro.

Simulation-based model tuning (*i.e.*, characterization) is attractive and practical: in principle it automatically increases the accuracy of power models to match that of a low-level simulator. Unfortunately characterization has two drawbacks that come from its statistical nature: *i)* it cannot guarantee *out-of-sample* accuracy, and *ii*) it does not enable *worst-case* evaluation.

1.1 Out-of-sample accuracy

We call *out-of-sample* accuracy the accuracy of a model evaluated for input conditions statistically different from those in which it was characterized. In contrast, we call *in-sample* accuracy the accuracy of a model evaluated in the same conditions used for characterization. Only in-sample accuracy is increased by the characterization process. If the input patterns applied to the circuit during characterization are not representative of the actual operating conditions, the model may produce unpredictable errors.

To improve out-of-sample accuracy two approaches have been attempted: increase the complexity (and the flexibility) of the model [4, 6, 2], or improve the characterization procedure [5, 7]. No general solutions have been provided by the first approach: when the model complexity increases the characterization process becomes even more critical and the resulting model prone to out-of-sample errors.

The second approach has been more successful [5, 7]. In [5] the authors proposed the usage of a look-up table (LUT) of constant estimators statically pre-characterized under different input-output conditions.

The method presented in [7] relies on the availability of a multilevel simulation environment to perform dynamic model tuning by means of an adaptive algorithm (namely, the LMS). Adaptive modeling achieves sizable improvements over the accuracy of statically-characterized equivalent models.

1.2 Worst-case evaluation

Even if it is possible to successfully trade off model complexity and/or characterization effort for out-of-sample accuracy, no compromises are allowed when dealing with worst-case estimates. Simulation-based characterization does not provide conservative worst-case estimators, unless an exhaustive search is performed by simulating all possible input transitions, *i.e.*, all possible pairs of input patterns. Needless to say, this is unfeasible even for small circuits.

To perform conservative worst-case estimates, analytical approaches are required that look at the inner structure of a macro in order to find the input conditions that maximize its internal switching activity [8, 9]. Unfortunately,

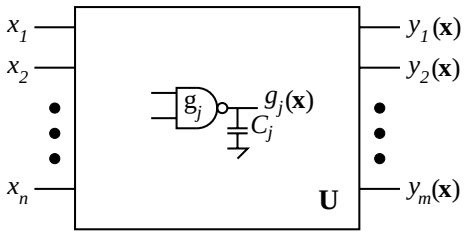


Figure 1: Representation of a generic combinational unit with n inputs and m outputs. Internal gate g_j is also represented together with its output function ($g_j(\mathbf{x})$) and load capacitance (C_j).

the complexity of the problem grows exponentially with the number of inputs and gates in the circuit, and the exact solution can be found only for small circuits. For large combinational blocks, only conservative approximations are provided.

Independently of the accuracy of worst-case power estimates achievable for the macros, large errors are always introduced when estimating the worst case consumption of an entire RTL design which contains many instances of library macros. In fact, the knowledge of worst-case power consumptions of library elements provides little practical information about the power consumption of the entire design. No compensation occurs when adding conservative estimates, the overall error grows with the number of system components.

Pattern-dependent upper bounds can provide more significant (still conservative) worst-case power estimates at the RT level. Given an input pattern, it is possible to compute an upper bound to the power consumption of the entire circuit for that pattern by simply summing the pattern-dependent upper bounds of its components. Such bound is much tighter than what could be obtained by just summing the overall worst-case power consumption of all macros in the circuit.

In this paper we present an *analytical* approach to pattern-dependent power modeling that:

- provides power models whose accuracy does not depend on the input statistics,
- provides pattern-dependent upper-bounds,
- effectively trades off accuracy for complexity.

In sharp contrast with all previous approaches our modeling procedure *exploits information on the internal structure* of the combinational logic unit. While in the past only *black box* models have been investigated, we propose a *white box* methodology. The modeling task is addressed in the next section. We discuss the modeling assumptions, we describe the RTL model and we outline a symbolic algorithm for its automatic construction. Approximation criteria and algorithms are discussed in Section 3. Experimental results are reported in Section 4. Section 5 concludes the work.

2 RTL power modeling

Fig. 1 shows a combinational unit (**U**) with inputs $\mathbf{x} = [x_1, ...x_n]^T$.[1] We assume that the unit is stable at time t^i and t^f ($t^f > t^i$) and that an input transition from $\mathbf{x}^i$ to $\mathbf{x}^f$ occurs between t^i and t^f. We denote by $e(\mathbf{x}^i, \mathbf{x}^f)$ the supply energy drawn by the circuit in the time interval $[t^i, t^f]$. The task of modeling power consumption at the RT level consists of finding a simple but accurate model for $e(\mathbf{x}^i, \mathbf{x}^f)$ (the corresponding power consumption being $p(\mathbf{x}^i, \mathbf{x}^f, T) = e(\mathbf{x}^i, \mathbf{x}^f)/T$, where $T = t^f - t^i$).

In previous approaches, models are constructed by: *i*) observing input-output information, *ii*) performing highly accurate simulation to estimate the power dissipation that corresponds to a given input transition, *iii*) correlating the power measurement and the input-output transition pattern with some model-fitting procedure. Our approach is radically different. We do not perform simulation, but we directly construct a high-level model starting from a low-level description of the internal structure of the combinational unit, which we call *golden model*. It is important to notice that the *golden model* must be available for the characterization-based approaches as well, because it is required for performing accurate simulations.

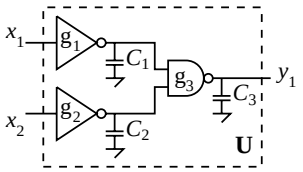
There are many phenomena that contribute to $e(\mathbf{x}^i, \mathbf{x}^f)$. For convenience, we partition them into *structural* and *parasitic* phenomena. For a given *golden model* of the unit, we call *structural* (*parasitic*) phenomena those phenomena that can (cannot) be appreciated at the abstraction level at which the *golden model* is provided. Power modeling and characterization are abstraction processes. The abstract model they provide cannot be more accurate than the *golden model* itself. Hence, the only phenomena that can be modeled are those that are structural for the *golden model*.

Characterization-based models are flexible because they can leverage any low-level simulator. In other words, they can be applied to any *golden model* (*i.e.*, a gate-level netlist, a transistor-level description). We give up some flexibility. We assume that our *golden model* is a gate-level netlist with backannotated capacitances and zero propagation delays. At this level, the only structural phenomena are the supply currents that charge capacitances associated with raising signals. Their contribution to the overall energy is usually called *dynamic consumption*. Short-circuit currents, internal charge redistributions and spurious transitions (*glitches*) are all parasitic phenomena.

By chosing an abstract golden model, we loose some absolute accuracy. Two observations motivate this choice. First, absolute accuracy is not always a key feature at the RT level. During RTL design some absolute accuracy may be lost if relative accuracy is still satisfactory. Second, if absolute accuracy is required, characterization can be used in conjunction with our approach to capture parasitic phenomena.

Our modeling approach is not in contrast with characterization methodologies. On the contrary, it leads to a useful partitioning of the modeling task. We construct a model for the pattern dependency of zero-delay dynamic power, which is the structural power of our golden model. Dynamic power (that usually represents the largest contribution) is responsible for most of the pattern dependence of the overall power consumption. Parasitic phenomena have a similar (and usually smoother) dependence on input statistics. Once a robust RTL model has been analytically constructed for the structural power, characterizing parasitic phenomena is much simpler than characterizing

[1] Hereafter we use boldface letters to denote vectors.



a)

b)

$\mathbf{x}^i$	$\mathbf{x}^f$	$C(\mathbf{x}^i, \mathbf{x}^f)$	fF
00	00	0	0
...	...	...	...
11	00	C_1+C_2	90
11	01	C_1	40
11	10	C_2	50
11	11	0	0

Figure 2: a) Gate-level netlist of a simple combination unit **U**. b) Look-up-table of discrete function $C(\mathbf{x}^i, \mathbf{x}^f)$ that represents the switching capacitance of **U**. The last column reports the values of C (in fF) evaluated by assuming $C_1 = 40fF$, $C_2 = 50fF$ and $C_3 = 10fF$.

the entire power consumption as a whole.

2.1 Analytical model construction

The structural power consumption of a zero-delay gate-level netlist can be expressed as

$$e(\mathbf{x}^i, \mathbf{x}^f) = V_{dd}^2 C(\mathbf{x}^i, \mathbf{x}^f) \quad (1)$$

where $C(\mathbf{x}^i, \mathbf{x}^f)$ represents the total *switching capacitance* (*i.e.*, the total capacitance associated with signals that have a raising transition[2] in the time interval $[t^i, t^f]$) and V_{dd} is the supply voltage. Since we assume V_{dd} to be constant, our modeling task consists of finding a RTL model for $C(\mathbf{x}^i, \mathbf{x}^f)$. Fig. 1 shows a generic internal gate g_j and its load capacitance C_j. If we denote by $\mathcal{S}_R$ the set of gates having an output raising transition between t^i and t^f, the total switching capacitance is

$$C(\mathbf{x}^i, \mathbf{x}^f) = \sum_{g_j \in \mathcal{S}_R} C_j \quad (2)$$

Pattern dependence is implicit in the definition of $\mathcal{S}_R$:

$$\mathcal{S}_R = \{g_j,\ j \leq N \mid g_j(\mathbf{x}^i) = 0\ AND\ g_j(\mathbf{x}^f) = 1\} \quad (3)$$

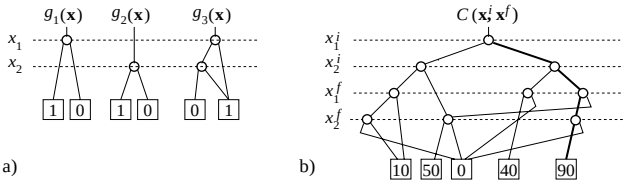
where $g_j(\mathbf{x})$ is the output function of gate g_j and N is the total number of gates in the circuit. Since g_j belongs to $\mathcal{S}_R$ if and only if $g_j'(\mathbf{x}^i) g_j(\mathbf{x}^f) = 1$, we can re-write C as:

$$C(\mathbf{x}^i, \mathbf{x}^f) = \sum_{j=1}^{N} g_j'(\mathbf{x}^i) g_j(\mathbf{x}^f) C_j \quad (4)$$

Example 1 *Consider the simple netlist of Fig. 2.a. Its node functions are:* $g_1(\mathbf{x}) = x_1'$, $g_2(\mathbf{x}) = x_2'$ *and* $g_3(\mathbf{x}) = x_2 + x_1$. *Assume that* $C_1 = 40fF$, $C_2 = 50fF$ *and* $C_3 = 10fF$. *The total switching capacitance corresponding to an input transition from* $\mathbf{x}^i = 11$ *to* $\mathbf{x}^f = 00$ *is given by:*

$$\begin{aligned} C(11,00) &= g_1'(11) g_1(00) \cdot 40fF + g_2'(11) g_2(00) \cdot 50fF \\ &\quad + g_3'(11) g_3(00) \cdot 10fF \\ &= 1 \cdot 1 \cdot 40fF + 1 \cdot 1 \cdot 50fF + 0 \cdot 0 \cdot 10fF \\ &= 90fF \end{aligned}$$

[2] If the output load of a gate is represented by a constant capacitor to ground, the dynamic supply current drawn by the gate is always associated with a raising output transition.



a) b)

Figure 3: a) BDDs of the node functions of unit **U** of Fig. 2.a. b) ADD representing the switching of the same unit as a function of its primary inputs at time t^i and t^f. The following assumption are used to simplify the pictorial representation of decision diagrams. Input variables are ordered and reported on the left. All nodes labeled by the same variable are represented at the same level. The left and right edges from each node are associated with the 0 and 1 assignments of its variable, respectively.

In principle, Eq. (4) is the RTL model of C we are looking for: as long as the logic functions realized at each internal node are known, it represents the switching capacitance as a function of $\mathbf{x}^i$ and $\mathbf{x}^f$. In practice, however, Eq. (4) is too complex to be used at the RT level. Its representation is much larger than the gate-level netlist we want to abstract away (N arbitrary Boolean functions of n variables, instead of N elementary gates), and its evaluation is more complex than a gate-level simulation (nN instead of N). Moreover, Eq. (4) actually represents the implementation of the unit. If the unit is a third-party intellectual property (IP), Eq. (4) cannot be used to backannotate the functional description, or otherwise the IP would be violated. To obtain a RTL power model of practical interest we need to precompute the weighted sum at the right-hand side of Eq. (4), thus obtaining a direct representation of $C(\mathbf{x}^i, \mathbf{x}^f)$.

Example 2 *Consider again the gate-level netlist of Fig. 2. If we evaluate Eq. (4) for all possible pairs of input patterns* $\mathbf{x}^i, \mathbf{x}^f$ *and we store the results in a look-up-table (LUT), we obtain a pattern-dependent model of the total switching capacitance that does not require any longer the explicit knowledge (and evaluation) of the internal functions. The LUT for unit* **U** *of Fig. 2.a is partially reported in Fig. 2.b. The switching capacitance corresponding to a given input transition (e.g., from* $\mathbf{x}^i = 11$ *to* $\mathbf{x}^f = 00$*) can be read directly from the corresponding row of the table* ($C = C_1 + C_2 = 90fF$).

The explicit precomputation of $C(\mathbf{x}^i, \mathbf{x}^f)$ has the same complexity of an exhaustive gate-level simulation, that grows exponentially with the number of primary inputs. The number of entries in the LUT of C is exponential as well. For a circuit with 20 inputs, more than 10^{12} values of C should be precomputed and stored. Precomputation is feasible only if it can be done symbolically and the result can be represented in a compact form.

$C(\mathbf{x}^i, \mathbf{x}^f)$ is a *discrete function, i.e.*, a mapping from a Boolean space to a finite set of values, $\mathcal{V}$ (notice that Boolean functions are nothing but discrete functions with $\mathcal{V} = \{0, 1\}$). Boolean functions can be represented as *binary decision diagrams* (BDDs) [10], generic discrete functions can be represented as *algebraic decision diagrams* (ADDs) [11]. Fig. 3 shows the BDDs of the node functions of unit **U** of Fig. 2.a and the ADD of its switching

capacitance. The root of a decision diagram is the function it represents. Internal nodes are associated with input variables, edges represent input assignments, leaves represent output values. Any configuration of input variables corresponds to a path from the root to a leaf. Bold lines are used in Fig. 3.b to point out the path associated with $\mathbf{x}^i = 11$, $\mathbf{x}^f = 00$. According to Ex. 1, the corresponding leaf has value 90.

Decision diagrams are *ordered* (variables are always encountered in the same order along any path) and *reduced* (isomorphic subtrees can be shared). Variable ordering makes the representation canonical, reduction makes it compact. Reduced, ordered decision diagrams provide an efficient way of representing and manipulating discrete functions. Logic and arithmetic operators are defined on BDDs and ADDs, respectively. From an algorithmic point of view, they take advantage of the structured representation to perform logic and arithmetic operations without actually enumerating all input configurations (*i.e.*, symbolically). The number of elementary operations involved is linear or quadratic in the number of nodes in the diagram, rather than exponential in the number of variables. Though the worst-case complexity of the representation of a discrete function is inherently exponential, reduced ADDs usually provide much more efficient representations.

We don't describe ADDs and BDDs in further details (the reader can refer to [11, 10]), but we rely on their properties for the symbolic manipulation and representation of discrete functions. Symbolic manipulation enables the straight-forward implementation of Eq. (4): the ADD of $C(\mathbf{x}^i, \mathbf{x}^f)$ can be directly obtained by applying Eq. (4) to the BDDs of the unit's node functions. Once constructed, the ADD can be evaluated in linear time for given input assignments.

To summarize, ADD-based symbolic modeling provides effective solutions to three of our modeling issues: *i*) it hides the dependence of C on the internal functions, *ii*) it avoids exhaustive precomputation and *iii*) it enables run-time model evaluation in a negligible time (linear in the number of input variables).

A last issue needs to be addressed: the model complexity. Unfortunately, this is not a marginal problem. Decision diagrams may blow up exponentially when the number of input variables increases and there are no general solutions to keep their size small, nor to foresee their explosive growth. After reduction (and variable reordering [10]) the only way of further simplifying ADDs is by approximating the discrete functions they represent. One of the main strengths of our approach is that it enables effective approximations to achieve a good trade-off between accuracy and complexity. This is shown in the next section.

3 Approximation criteria and algorithms

The effectiveness of any approximation criterion depends on the representation of the function to be approximated. Consider the expressions at the two sides of Eq. (4) as equivalent models for the switching capacitance. At the right-hand side C is expressed in terms of N Boolean functions (g_j) and the model consists of N BDDs. To cope with complexity, some BDDs may need to be simplified by approximating the corresponding functions. However, due to the involved dependence of C on those functions, the error induced on C by the approximation is almost unpredictable.

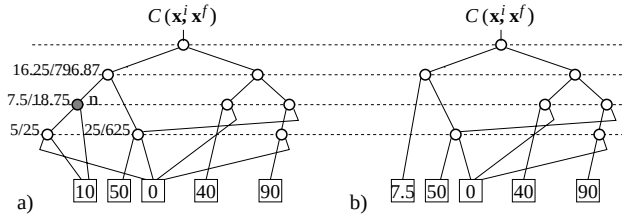


Figure 4: a) ADD of the switching capacitance of unit **U** of Fig. 2. For some of the ADD nodes the average value and variance of the discrete functions they represent are also reported. b) Simpler ADD obtained from the previous one by collapsing the sub-ADD rooted in n in a single leaf node that represents its average value.

On the contrary, at the left-hand side of Eq. (4) the switching capacitance is directly represented as a discrete function and the model consists of a single ADD. To reduce the model complexity, approximation criteria can be directly applied to the target function, thus keeping the induced error under control. This is exactly what we do, working directly on the representation of $C(\mathbf{x}^i, \mathbf{x}^f)$.

The simplest way of simplifying an ADD is by means of *node collapsing*, that consists of reducing a sub-ADD to a single leaf node. Fig. 4.a shows the complete ADD of the switching capacitance of our example unit. Fig. 4.b shows a smaller ADD obtained by replacing the sub-tree rooted in node n with leaf node 7.5. Since three nodes have been replaced by one, the overall size of the ADD has been reduced by 2 at the cost of loosing some pattern dependence.

Example 3 *Consider the ADD of Fig. 4.a. The sub-ADD rooted in node* n *represents the dependence of C on the second input vector ($\mathbf{x}^f$) when the first one is $\mathbf{x}^i = 00$. In particular, the switching capacitance is 0 if $\mathbf{x}^f = 00$, while it is equal to $10fF$ in all other cases. This pattern dependence is lost in the ADD of Fig. 4.b: when $\mathbf{x}^i = 00$ the approximated value of C is 7.5 regardless of $\mathbf{x}^f$.*

Without going through the algorithmic details of node collapsing, we remark that several sub-trees can be independently collapsed during a traversal of the original ADD. The overall cost of the process is linear in the number of nodes. Any trade-off between accuracy and complexity can be explored just by changing the *degree of compression*, *i.e.*, the percentage of nodes to be collapsed. If no node is removed, the ADD represents the switching capacitance without approximations (the model has the accuracy of a gate-level simulation). If the entire ADD is collapsed in a single leaf, the model degenerates in a constant estimator.

More sophisticated algorithms can be conceived for ADD simplification. In this paper we only focus on node collapsing because of two main reasons: *i*) it is performed in linear time and *ii*) it is flexible enough to implement different approximation strategies.

In Ex. 3 we didn't mention why node n was chosen for collapsing and why the resulting leaf node was 7.5. We call *approximation strategy* a criterion to select the sub-ADDs to be collapsed and the values to be associated with

the resulting leaf-nodes. For a given degree of compression, both the scope and the accuracy of the final approximation depend on the strategy adopted to steer the simplification procedure.

Since node collapsing always replaces a sub-function with a constant value, a general criterion to reduce the approximation error (and the loss of pattern dependence) consists of selecting for replacement sub-ADDs with minimum variance. The variance of an ADD is the variance of the function it represents. For a generic function $f(\mathbf{x})$ of n Boolean variables

$$var(f(\mathbf{x})) = \frac{1}{2^n}\sum_{\mathbf{x}}(f(\mathbf{x}) - f_{avg})^2 \tag{5}$$

where f_{avg} is the average value of f:

$$avg(f(\mathbf{x})) = \frac{1}{2^n}\sum_{\mathbf{x}} f(\mathbf{x}) \tag{6}$$

To chose the target nodes for collapsing we need to compute the variance of all discrete functions associated with internal nodes. This can be done in linear time during a traversal of the ADD, using a recursive formula. Since leaves represent constant functions, for a generic leaf node k we have $var(\mathrm{k}) = 0$ and $avg(\mathrm{k}) = value(\mathrm{k})$. For a generic internal node n, both $var(\mathrm{n})$ and $avg(\mathrm{n})$ can be computed from those of its left and right children (denoted by n_{left} and n_{right}, respectively):

$$\begin{aligned} avg(\mathrm{n}) &= \frac{1}{2}(avg(\mathrm{n}_{left}) + avg(\mathrm{n}_{right})) \qquad (7)\\ var(\mathrm{n}) &= \frac{1}{2}(var(\mathrm{n}_{left}) + (avg(\mathrm{n}_{left}) - avg(\mathrm{n}))^2 + \\ &\quad var(\mathrm{n}_{right}) + (avg(\mathrm{n}_{right}) - avg(\mathrm{n}))^2) \end{aligned}$$

Once $var(\mathrm{n})$ has been computed for all nodes in the ADD, sub-ADDs rooted at nodes with minimum variance are chosen for collapsing. Node collapsing proceeds (possibly involving nodes with larger variance) until the global ADD is reduced under a target size. The overall process has the complexity of two ADD traversals.

Constant values used for replacement depend on the scope of the model. If we are targeting high accuracy on average power estimates, the average values computed for each original node ($avg(\mathrm{n})$) can directly be used to replace the corresponding sub-trees. In this case, $var(\mathrm{n})$ represents the *mean square error* (mse) of the approximated sub-function.

Example 4 *Consider the ADD of Fig. 4.a and assume that we need to reduce its size with a minimum impact on its average accuracy. During a first traversal of the ADD, the average value and the variance are computed for each internal node. Consider, for instance, node* n. *According to Eq. (8),* $avg(\mathrm{n})$ *and* $var(\mathrm{n})$ *are given by:*

$$\begin{aligned} avg(\mathrm{n}) &= \frac{1}{2}(10+5) = 7.5\\ var(\mathrm{n}) &= \frac{1}{2}(25 + (5-7.5)^2 + 0 + (10-7.5)^2) = 18.75 \end{aligned}$$

In Fig. 4.a, avg and var are reported for some of the internal nodes. Node n *is the one with the smallest variance. Hence, it is the replaced by a constant leaf node having value* $avg(\mathrm{n})$. *The resulting ADD is shown in Fig. 4.b.*

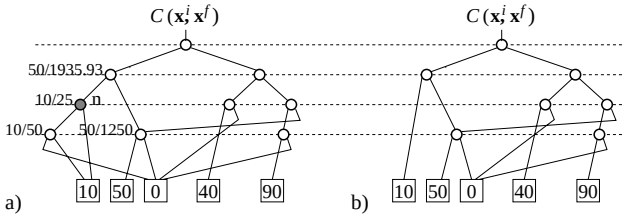


Figure 5: a) Same ADD of Fig. 4.a. For some of the ADD nodes two values are reported: the maximum value of the corresponding sub-function and the *mse* made by using the maximum to approximate the sub-function. b) Simpler ADD obtained from the previous one by collapsing the sub-ADD rooted in n in a single leaf node that represents its maximum value.

```
C(x^i, x^f) = 0;
for (i=1; i <= N; i++) {
    deltaC(x^i, x^f) = bdd_and(bdd_not(g_i(x^i)), g_i(x^f));
    deltaC(x^i, x^f) = add_times(deltaC(x^i, x^f), C_i);
    if (add_size(deltaC(x^i, x^f)) > MAX)
      add_approx(deltaC(x^i, x^f), MAX);
    C(x^i, x^f) = add_sum(C(x^i, x^f), deltaC(x^i, x^f));
    if (add_size(C(x^i, x^f)) > MAX)
      add_approx(C(x^i, x^f), MAX);
}
```

Figure 6: Pseudo-code for the iterative construction of the ADD of the switching capacitance. The following operators are assumed to be available for the symbolic manipulation of DDs: `bdd_not`, that returns the logic NOT of a given BDD, `bdd_and`, that returns the logic AND of two given BDDs, `add_times`, that multiplies a given DD by a given constant value, `add_sum`, that returns the arithmetic sum of two ADDs, and `add_approx`, that simplifies a given ADD (according to predefined approximation criteria) in order to reduce its size within a given limit.

The strategy adopted to obtain pattern-dependent upper bounds is slightly different. First, maximum values have to be used to replace sub-ADDs. Second, the variance does not represent any longer the *mse* caused by the approximation. Maximum values can be easily computed during the ADD traversal. For any leaf node k, $max(\mathrm{k}) = value(\mathrm{k})$, while for any internal node n, $max(\mathrm{n}) = maximum\{max(\mathrm{n}_{left}), max(\mathrm{n}_{right})\}$. As for the *mse*, the following relation holds for each node:

$$mse(\mathrm{n}) = var(\mathrm{n}) + (max(\mathrm{n}) - avg(\mathrm{n}))^2 \tag{8}$$

Example 5 *Our example ADD is reported again in Fig. 5.a. Suppose that we need to reduce the size of the ADD making only conservative approximations. avg, var, max and mse are initially computed for each internal node. For node* n, $max(\mathrm{n}) = maximum\{10,10\} = 10$, *while* $mse(\mathrm{n}) = 18.75 + (10-7.5)^2 = 25$. *Maximum value and mse are reported in Fig. 5.a for some internal nodes. Nodes with minimum mse are then selected for collapsing and their sub-ADDs replaced by maximum-value leaves. In our example, node* n *is the best choice. The ADD after collapsing is shown in Fig. 5.b.*

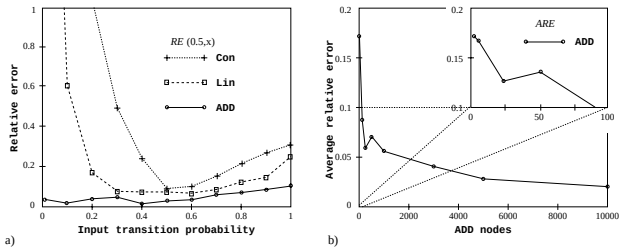


Figure 7: a) Relative error of RTL power estimators `Con`, `Lin` and `ADD` for different input statistics. Data refer to benchmark circuit cm85. b) Trade-offs between accuracy and size of the RTL power model of circuit cm85.

3.1 Process flow

Approximations can be performed at any time during the construction of the ADD of the switching capacitance. This is due to the iterative implementation of Eq. (4). The pseudo-code of the main loop is reported in Fig. 6. At each step, the contribution of a new gate is computed and added to the overall function (C). Both C and its partial contributions (denoted by *deltaC* in Fig. 6) are represented by ADDs. Whenever the size of an ADD exceeds the maximum allowed value (`MAX`) the approximation routine is called to bring its size back under the limit. Since the function to be approximated is always a switching capacitance, the error induced to the model can be always controlled. Moreover, since the only operator possibly applied to ADDs after node collapsing is the arithmetic addition (`add_sum`), the same strategy may be used for local and global approximations. The respect of the global strategy is guaranteed by the following properties: $avg(a) + avg(b) = avg(a + b)$ and $max(a) + max(b) \geq max(a + b)$, that hold for any pair of discrete functions a and b.

4 Experimental results

The algorithms described in this paper were implemented in C using the CUDD package for symbolic manipulation of Boolean and discrete functions [11]. Experiments were run on benchmark circuits from the MCNC suite [12]. Gate-level netlists were obtained by mapping the circuits on a test gate library. Input capacitances of fan-out gates were used as load capacitances for the driving ones.

We report the results of three sets of experiments. The first one is a case study (based on benchmark circuit cm85) aimed at exploring the modeling trade-off enabled by our approach. The second and third ones are extensive tests performed on several benchmarks to evaluate the accuracy achieved on average power estimates and the effectiveness of pattern-dependent upper bounds, respectively.

Two characterization-based power models are used for comparison: a constant average estimator (denoted by `Con`) and a linear model (denoted by `Lin`). `Lin` expresses power consumption as a linear function of input transitions: $P = c_0 + c_1 a_1 + ... + c_n a_n$, where $a_j = x_j^i \oplus x_j^f$ and c_j are fitting coefficients. For each benchmark, both models were characterized to fit the results of a zero-delay gate-level simulation performed with a random input sequence with 0.5 average signal and transition probabilities.

We use the average signal probability (sp) and the average transition probability (st) to represent input statistics. For each model, the relative error on average power estimates is a function $RE(sp, st)$. To evaluate $RE(sp, st)$ we repeatedly ran concurrent RTL and gate-level simulations with random sequences of 10000 input vectors with different values of sp and st.

For benchmark circuit cm85, RE is plotted in Fig. 7.a as a function of st (for $sp = 0.5$). The strong dependence on input statistics is evident for both `Lin` and `Con`: while their in-sample error is below 10% (for st around 0.5), the out-of-sample error is much higher and it becomes larger than 100% when $st < 0.2$. In contrast, the accuracy achieved by our model is almost independent of st, as shown in Fig. 7.a by the solid-line curve labeled `ADD`. Incidentally, we remark that an upper bound of 500 ADD nodes was used during the construction of the model. The unbounded model would have more than 10000 nodes and it would fit exactly the results of gate-level simulations regardless of input statistics (*i.e.*, $ER(sp, st) = 0 \ \forall \ sp, st$).

We call *average relative error* (ARE) the average value of $RE(sp, st)$ over all simulation runs and we use it to represent the quality of RTL power models in terms of accuracy and robustness. For benchmark circuit cm85, we obtained $ARE = 518.7\%$ for `Con`, $ARE = 195.2\%$ for `Lin` and $ARE = 5.7\%$ for `ADD`.

The approximations described in Section 3 provide a wide range of trade-off between the quality and the size of the RTL power models. Fig. 7.b shows the ARE of ADD-based power models of different sizes for cm85. Though the exact representation of $C(\mathbf{x}^i, \mathbf{x}^f)$ would require an ADD with more than 10000 nodes, it is worth noting that ADDs with 10 or 5 nodes are sufficient to achieve power estimates with ARE below 20% (*i.e.*, one order of magnitude smaller than the ARE of a linear model with 12 fitting coefficients).

Experimental results on benchmark circuits are reported in Table 1. The name of the circuit, the number of inputs (n) and the number of gates (N) are reported in the first three columns. Columns four to eight refer to average power estimates. The AREs provided by constant (`Con`) and linear (`Lin`) estimators are reported in columns four and five for comparison. The ARE of our analytical model (`ADD`) is reported in column six. It is around 10 times smaller than that of linear estimators and 50 times smaller than that of constant ones. The constraint on the maximum number of ADD nodes (`MAX`) used during the construction of the model is shown in column seven. The general criterion used for choosing the value of `MAX` was that of making the size of the power model for a unit comparable with that of its functional description. The CPU time (in seconds) spent for building the model on a SUN UltraSparc 2 is reported in column eight.

The last four columns of Table 1 refer to conservative upper bounds of power consumption. Results are reported in terms of average relative error on maximum power estimates. For each simulation run, the maximum value provided by the upper bound was compared with the maximum value provided by gate-level simulation and the relative error (RE) was computed. As for the average power estimates, the ARE was evaluated as the average of RE over several simulation runs with different input statistics. Columns nine and ten report the average error made by a

Benchmark circuit			Average estimators					Upper bounds			
			ARE (%)			Model		*ARE* (%)		Model	
name	n	N	Con	Lin	ADD	MAX	CPU	Con	ADD	MAX	CPU
alu2	10	252	464.8	135.7	**4.8**	1000	496	154.0	**21.0**	5000	2766
alu4	14	460	465.1	242.5	**7.8**	2000	5087	201.0	**59.2**	15000	6470
cmb	16	34	585.7	88.9	**10.7**	200	12	237.1	**47.0**	1000	9
cm150	21	46	647.3	270.4	**12.2**	1000	664	193.0	**47.6**	2000	30
cm85	11	31	518.7	195.2	**5.7**	500	9	167.8	**30.9**	500	5.6
comp	32	93	460.9	193.8	**15.0**	5000	1614	211.6	**54.9**	10000	596
decod	5	23	812.6	80.2	**3.2**	200	5	156.1	**4.6**	200	2
k2	45	1206	622.5	78.5	**14.3**	10000	7511	188.6	**2.1**	10000	4375
mux	21	61	596.8	161.1	**18.7**	1000	571	167.9	**43.9**	5000	92
parity	16	36	316.5	219.0	**6.8**	3000	98.4	177.3	**37.9**	500	7
pcle	19	45	591.0	248.6	**8.0**	5000	281	186.1	**40.9**	10000	70
x1	49	228	682.8	200.7	**12.3**	1000	9505	318.9	**56.7**	50000	10143
x2	10	40	738.4	204.9	**8.9**	200	15	138.7	**10.3**	2500	22

Table 1: Experimental results.

constant maximum estimator (Con) and by our pattern-dependent upper bound (ADD), respectively. As a constant estimator we used the maximum value of the pattern-dependent upper bound. No linear models were used for comparison since they do not provide conservative bounds. The *ARE* of the constant estimator is always much larger than 100%, while the error made by ADD is always smaller than 60%. Moreover, pattern-dependent upper bounds of several units can be effectively composed to provide significant upper bounds for complex systems described at the RT level. The maximum allowed ADD size and the CPU time are reported in the last two columns.

For some circuits (*e.g.*, C6288) ADDs with more than 100000 nodes were required to bring the *ARE* below 30%. This is an inherent limitation of the ADD-based representation, whose complexity strongly depends on the functional and structural properties of the circuit. Though our approximation strategy can in principle be used to keep ADDs as small as desired, when the approximation becomes too aggressive it may lead to large errors, comparable to those of traditional RTL models. Overcoming this limitation is the target of our on-going work.

5 Conclusions

RTL power models based on characterization have two main drawbacks: *i*) their accuracy strongly depends on the input statistics, and *ii*) they do not provide worst-case information.

To overcome these limitations we have presented an analytical approach that does not require characterization. The power consumption of a combinational macro is expressed in terms of its internal switching capacitance. A pattern-dependent RTL model for the switching capacitance of the macro is automatically constructed during a traversal of its gate-level netlist. Since only structural information is exploited, the accuracy of the model does not depend on input statistics. Several approximation criteria can be applied during model construction in order to trade off accuracy for complexity. In particular, conservative approximations can be used to provide reliable pattern-dependent upper bounds.

We have presented the key ideas, outlined the main algorithms and discussed the experimental results obtained on benchmark circuits. In average, analytical models provide power estimates with relative errors more than 10 times smaller than those provided by pre-characterized constant and linear estimators.

References

[1] S. Powell and P. Chau, "Estimating power dissipation of VLSI signal processing chips: the PFA techniques," in *Proc. of Workshop on VLSI Sig. Proc.*, vol. IV, pp. 250–259, 1990.

[2] H. Mehta, R. M. Owens, and M. J. Irwin, "Energy characterization based on clustering," in *Proc. of Design Automation Conf.*, pp. 702–707, 1996.

[3] Q. Qiu, Q. Wu, M. Pedram, and C.-S. Ding, "Cycle-Accurate Macro-Models for RT-Level Power Analysis," in *Proc. of Int.l Symposium on Low Power Electronics and Design*, pp. 125–130, 1997.

[4] P. Landman and J. Rabaey, "Architectural power analysis, the Dual Bit Type method," *IEEE Transactions on VLSI Systems*, vol. 3, no. 2, pp. 173–187, 1995.

[5] S. Gupta and F. Najm, "Power macromodeling for high-level power estimation," in *Proc. of Design Automation Conf.*, pp. 365–370, 1997.

[6] L. Benini, A. Bogliolo, M. Favalli, and G. De Micheli, "Regression models for behavioral power estimation," to appear on Integrated Computer-Aided Engineering, 1997.

[7] A. Bogliolo, L. Benini, and G. D. Micheli, "Adaptive least mean square behavioral power modeling," in *Proc. of IEEE Eur. Design and Test Conf.*, 1997.

[8] S. Manich and J. Figueras, "Maximizing the Weighted Switching Activity in Combinational CMOS Circuits," in *Proc. of Power and Timing Modeling, Opt. and Sim. Workshop*, 1996.

[9] S. Devadas, K. Keutzer, and J. White, "Estimation of Power Dissipation in CMOS Combinational Circuits Using Boolean Function Manipulation," *IEEE Transactions on CAD*, vol. 11, no. 3, pp. 373 – 383, 1992.

[10] K. S. Brace, R. Rudell, and R. Bryant, "Efficient Implementation of a BDD Package," in *Proc. of Design Automation Conf.*, pp. 40–45, 1990.

[11] R. I. Bahar *et al.*, "Algebraic Decision Diagrams and their Applications," in *Proc. of IEEE Int.l Conf. On Computer Aided Design*, pp. 188–191, 1993.

[12] S. Yang, "Logic Synthesis and Optimization Benchmarks User Guide Version 3.0," *Technical report, Microelectronics Center of North Carolina*, 1991.