State Assignment for Low Power Dissipation

Luca Benini, *Student Member, IEEE,* and Giovanni De Micheli, *Fellow, IEEE*

Abstract— **We address the problem of reducing the power dissipated by synchronous sequential circuits. We target the reduction of the average switching activity of the input and output state variables by minimizing the number of bit changes during state transitions. Using a probabilistic description of the finite state machines, we propose a state assignment algorithm that minimizes the Boolean distance between the codes of the states with high transition probability. We formulate a general theoretic framework for the solution of the state assignment problem, and propose different algorithms trading off computational effort for quality. We then generalize our model to take into account the estimated area of a multilevel implementation during state assignment, in order to obtain final circuits where the total power dissipation is minimized. A heuristic algorithm has been implemented and applied to standard benchmarks, resulting in a 16% average reduction in switching activity.**

I. Introduction

As the minimum device size shrinks, traditional limiting factors like area and speed benefit from the increased level of integration, but power dissipation inevitably increases as more switching devices are operating on the same active chip area. Designers of circuits for portable devices, and designers of high-speed processors must cope with excessive power dissipation, intensifying the need for synthesis tools for low-power circuits.

Recent work has generated both general guidelines and synthesis tools for low-power circuit design [1], [2], [7], [9], [30]. Both technology-independent and technology-dependent logic transformations applied to the combinational part of sequential circuits have been shown to be effective in optimizing circuits for low power. Optimization techniques exploiting sequential properties have been shown to produce circuits with reduced power dissipation, confirming that optimization of combinatorial logic is only one face of this complex problem [4]. The direct synthesis of sequential circuits for low power [3], [12], [14] is an area of exploration which promises more global power savings and is the subject of this paper.

Sythesis systems typically take an HDL model of a design, written in a language such as VHDL or Verilog, as the initial input. The synthesis toolset then transforms this description into an implementation which has been optimized for some cost metric assigned by the designer. This synthesis path is usually composed of several independent steps that can be summarized as follows.

- *High-level synthesis*: the HDL model is optimized and compiled into one or more (possibly concurrent) finite state machines (FSM's), expressed as tables or as state-transition graphs.
- *State assignment*: the symbolic states of the FSM's are encoded using a specific binary encoding, and a description of the circuit in terms of Boolean functions is produced.
- *Logic synthesis*: optimizations are applied to the Boolean functions to obtain a decomposition of the Boolean description that is minimal with respect to the original cost metric.
- *Library binding*: the reduced Boolean equations resulting from logic synthesis are mapped to elements from a standard gate library and a final gate-level description is produced.

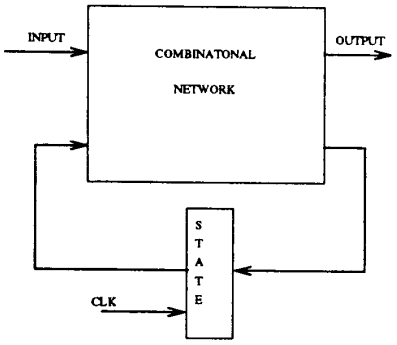

Fig. 1. General sequential circuit structure.

Our work targets the second step of this synthesis procedure, namely, state assignment. The FSM generated by high-level synthesis is typically in the form of a state transition graph (STG), where each state is represented symbolically. Our algorithms choose the binary codes to assign to the symbolic states such that switching activity on the input and output state variables (Fig. 1) is reduced.

At the STG level of abstraction nothing has been decided about the structure of the combinatorial logic implementing the next state and output functions. Thus, the state assignment algorithm can exploit degrees of freedom that are lost at successive phases and produce an encoded state transition table that is an effective starting point for further power optimizations of the combinatorial logic. However, because nothing is known about power dissipation in the combinational logic at that stage, reduced power dissipation on the state lines may later lead to increased dissipation within the remaining logic. Cognizant of that, our algorithms target a state assignment that minimizes the switching activity between state transitions while taking into account the estimated area of the next state and output logic, resulting in an implementation with lower switching activity and minimal area, and thus, lower power.

Manuscript received July 12, 1994; revised October 26, 1994. This work was funded by NFS ARPA under Contract 9115432.

L. Benini and G. De Micheli are with the Center for Integrated Systems, Stanford University, Stanford, CA 94305 USA.

IEEE Log Number 9408740.

Earlier approaches to state assignment have targeted area and performance both for implementations composed of two-level and multi-level logic ([10], [18], and [24] are good surveys of previous work). To develop state assignment algorithms targeting low power dissipation, new cost functions are needed to drive the search for optimal assignments. Moreover, because the quality of results is strongly influenced by the cost function, a complex interdependence exists between the cost estimator and the solution strategy.

The main theoretical contributions of our work are in the formulation of a new problem that links a probabilistic description of a finite state machine to its power dissipation and in the study of a new class of algorithms for the search of optimal and suboptimal solutions to the problem of finding a state assignment that gives low power dissipation. Minimizing the switching activity on the state lines in the FSM by itself does not guarantee reduced total power dissipation, because the power consumed in the combinatorial part is not accounted for. We discuss this problem and propose a more accurate cost metric that also factors in the complexity of the combinatorial logic.

We implemented the algorithms proposed and ran them on benchmark circuitsobtaining a 34% average reduction in switching activity of the state variables, a 16% average reduction of the total switching activity of the implemented circuit with a corresponding 14% average area increase. Although our solution is heuristic, and does not guarantee the minimum power dissipation, these results demonstrate that our approach leads to a reduction in the power dissipated in the complete circuit, not just in the part used for the computation and the storage of the state information. Moreover, the proposed algorithms can be used to explore the complex tradeoff between area and power dissipation.

The rest of this paper is organized as follows. In Section II we review the basic model for power estimation and give some theoretical results on the effectiveness of a probabilistic approach to describing the switching behavior of FSM's defined by signal transition graphs. In Section III we formalize the state assignment problem targeting power dissipation and present an exact algorithm for its solution. We implemented heuristics based on the exact algorithm, which we describe in Section IV, along with their limitations. In Section V we present some experimental results on the application of the previously described algorithms.

II. Probabilistic Models

For CMOS circuits, average power dissipation is proportional to the average switching activity. A good approximation of the average switching activity is the switching probability (or transition probability). Given the input switching probability it is possible to calculate the probability of the state transitions in a FSM. This information can be used to find an encoding that minimizes the switching probability of the state variables. In this section we will discuss the details of the concepts above outlined, and we will give some background material needed to understand the algorithms that follow.

A. A Model for Power Dissipation

At the gate level of abstraction, a circuit's power consumption is proportional to its switching activity [11]. We define the average switching activity at the output of a gate i in a time period T as the average number of signal transitions: $n_i(T) = n_{trans}/T$. We define the transition probability (switching probability) as $p_i = \lim_{T\to\infty} n_i(T)$, namely, the limit value of the switching activity as the observation time goes to infinity [5]. To find the average total power dissipated, we consider the average power dissipated by each gate during one clock cycle (or an arbitrarily defined $T_{cycle} = \Delta t$) and we sum over all gates in the network.

$$\overline{P_{tot}} = \frac{1}{2} V_{dd}^2 / T_{cycle} \sum_{i=1}^{n_g} C_i p_i \tag{1}$$

Where V_{dd} is the supply voltage, C_i is the capacitive load at the output of gate i, and n_g is the number of gates in the network.

Power consumption at the gate level can be reduced by modifying one or moreof the parameters in (1).

- Voltage supply or frequency reduction $(1/T_{cycle})$. These parameters are decided by circuit designers and we consider them as constants. It should be noted, however, that complex trade-offs are involved in the choice of the optimal values for these quantities. In fact, low-energy computation is a more realistic target, because we do not want to simply trade-off performance with power.
- Reduction of the term $\sum_{i=1}^{n_g} C_i p_i$. Notice that keeping this term small is particularly useful when low-energy computation is the target, because no trade-off with speed is involved in this case. This kind of reduction can be obtained in two ways:
  a) Resynthesize the combinatorial logic. For example, we can reduce the number of nodes with high switching activity during the library binding step or we can perform ad-hoc technology-independent logic optimizations during the logic synthesis step [7], [9], [30]. Combinational resynthesis can reduce power by reducing number of nodes in the network n_g and/or the load capacitance of each gate C_i.
  b) Resynthesize the entire sequential circuit, by determining both a register configuration and a combinational structure that minimize $\overline{P_{tot}}$.

We address synthesis of the sequential circuit, which is a complex problem because the description we start from is at a high level of abstraction and no information on the structure of the unoptimized network is available. We therefore concentrate on the state assignment problem whose solution determines the register configuration. Note again, that the state assignment strongly affects the size and the structure of the FSM's combinational component.

B. Estimation of Transition Probabilities

Given the FSM description and the knowledge of the input probabilities we want to compute the transition probabilities for the STG. The input probability distributions can be ob-

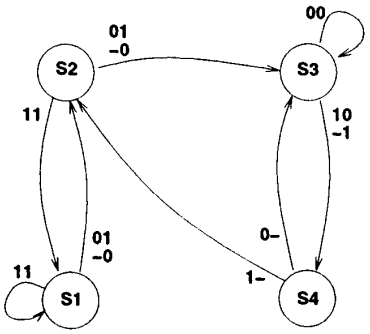

Fig. 2. The STG of an FSM with four states and two input signals. The "–" symbol represents (don't care) entry.

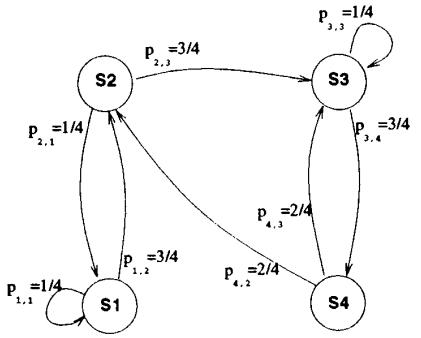

Fig. 3. Conditional transition probabilities assuming equiprobable and independent input signals.

tained by simulating the FSM at a higher level of abstraction in the context of its environment, or by direct knowledge from the designer. Transition probability information for each edge in the STG can then be determined by modeling the STG as a Markov chain, as we show in this subsection.

Note that transition probabilities are strongly dependent upon the initial state. For example, if an FSM has a transition from state s_i to state s_j for all possible input configurations, we may think that this transition will happen with very high probability during the operation of the machine. This may not be the case: if state s_i is unreachable, the machine will never perform the transition, because it will never be in state s_i. Similarly, if the probability of being in state s_i is very low, a transition from state s_i to state s_j is very unlikely. Our state assignment algorithm targets the reduction of the switching activity by assigning similar codes to states with very high transition probability. We must therefore compute the correct value of the transition probabilities before applying the optimization procedure.

An FSM with n_s states can be described by an STG defined by a vertex(state) set $S = \{s_1, \cdots, s_{n_s}\}$ and a related directed edge set representing the set of transitions from one state to another.

Example 1: Fig. 2 shows the STG for a simple FSM. The edge labelling represents the input configurations that cause a transition from the state at the tail of the edge to the state at the head of the edge. For example, in state S_1, an input of either 00, 01, or 10 will cause a the FSM to transition from S_1 to S_2, whereas an input of 11 results in the FSM remaining in S_1. We use this STG throughout the paper as an example for the application of our algorithms.

We now want to use information about probabilities to compute a static probabilistic model of the FSM which will give the transition probabilities for the FSM. We do this by interpreting the STG as a Markov chain. A Markov chain is a representation of a finite state Markov process [23], a stochastic model where the probability distribution at any time depends only on the present state and not on how the process arrived in that state. The Markov chain model for the STG can be described by a directed graph with a structure isomorphic to the STG and with weighted edges. For a transition from state s_i to state s_j, the weight $p_{i,j}$ on the corresponding edge represents the conditional probability of the transition (i.e., the probability of a transition to state s_j given that the machine was in state s_i). Symbolically this can be expressed as:

$$p_{i,j} = Prob(Next = s_j \mid \text{Present} = s_i). \tag{2}$$

Note that edges with zero conditional probability are never drawn in the graph representation of the Markov chain.

The set of values for all conditional probabilities is called the conditional probability distribution. The conditional probability distribution is easily found from the input probability distribution and by observing for which input configurations the FSM performs its state transitions.

Example 2: In Fig. 3, the conditional transtion probabilities for our example STG are given assuming (for simplicity) uncorrelated and equiprobable inputs. Calculation of these values is straightforward. For example, $p_{1,2} = P(01) + P(00) + P(10) = \frac{3}{4}$, where $P(X)$ represents the probability of the input taking value X.

Although conditional transition probabilities can be used as a rough approximation to the transition probabilities [3], we need to know the probability of a transition taking the present state into account. These probabilities are called total transition probabilities, $P_{i,j}$, and can be calculated [23] from the state probabilities, where the state probability, P_i, represents the probability that the machine is in a given state i. Namely,

$$P_{i,j} = p_{i,j} P_i. \tag{3}$$

Equation (3) implies that, to have high total transition probability *both* the state probability *and* the conditional transition probability must be high. Using only the conditional transition probability can lead to incorrect estimates.

The next step is to show that it is possible to compute the state probabilities and, more importantly, to show that these values are not time-dependent. Intuitively, this implies that as the observation time increases, the probability that the machine is in each of its states converges to a constant (stationary) set of real numbers. In other words, we must show that it is possible to compute a steady state (or stationary) probability vector whose elements are the stationary state probabilities.

It is quite easy to find STG's for which the stationary state probabilities do not exist, because, for example, their value

is oscillatory. The general theory explaining the asymptotic behavior of the state probabilities is too involved to be described here [23], so we want to find a large class of STG's whose corresponding Markov chains have a steady state probability vector.

A fundamental theorem in Markov chain theory states that:

Theorem 1: For an irreducible, aperiodic Markov chain, with all states recurrent nonnull, the steady state probability vector exists and it is unique [23].

An irreducible Markov chain with all the states recurrent nonnull is a chain where every state can be reached from any other state, and the greatest common divisor of the length of the possible closed paths from every state is one.

We define the reset state, s_0, as a state such that there is a transtion (with nonzero probability) to it from every state in the STG. A reduced STG is defined as an STG where all unreachable states (from the reset state) in the original STG have been eliminated. With these definitions, we can state the following theorem (see the Appendix for the proof, if interested):

Theorem 2: The Markov chain corresponding to a reduced STG with reset state s_0 and known conditional transition probabilities is irreducible, aperiodic, with all states recurrent non null.

In other words, we can find the transition probabilities for an STG that has a reset state, and this is the case for a large number of practical applications. Note that this theorem is a only sufficient condition; that is, there are STG's without a reset state for which we can successfully compute the transition probabilities.

Let $\boldsymbol{P}$ be the conditional transition probability matrix whose entries $p_{i,j}$ are the conditional transition probabilities, and $\mathbf{v}$ the steady state probability vector whose components are the state probabilities P_i. Then we can compute the steady state probabilities by solving the system of $(n_s + 1)$ equations:

$$\mathbf{v}^{\mathbf{T}}\boldsymbol{P} = \mathbf{v}^{\mathbf{T}} \tag{4}$$

$$\sum_{i=1}^{n_s} P_i = 1. \tag{5}$$

The problem of finding the steady state probability vector is thus reduced to finding the left eigenvector of the transition matrix corresponding to the unit eigenvalue, and normalizing it in order to make the sum of its elements equal to unity [23].

Example 3: The stationary state probabilities calculated solving the system above are shown in Fig. 4 besides the nodes in the STG. Recall that the matrix P is known, and contains the conditional transition probabilities (see Example 2). The figure shows the total transition probabilities (the products $p_{i,j}P_j$) on the edges. Note that the probabilities for self-loops are not shown only because we are not interested in edges that do not imply any state transition. Note also that although our STG does not have a reset state, its stationary probability vector can be calculated.

C. Transformation of the STG

Once the total transition probabilities have been calculated, we can transform the original STG into a weighted graph

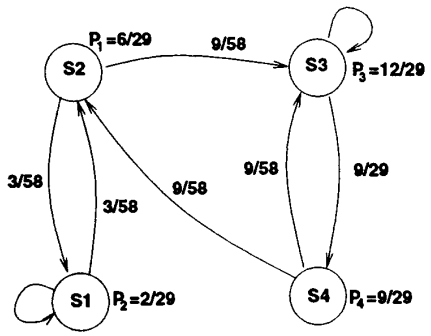

Fig. 4. State probabilities and total transition probabilities.

which preserves only the relevant information needed for state assignment. For each pair of connected states, we only need to know the probability of a transtion from one state to the other and vice-versa. Therefore, all input related information and self loops can be eliminated.

The transformations of the STG can be summarized as follows:

- Eliminate all unreachable states, if any.
- Calculate the state stationary probability vector and, from that, calculate the total transition probabilities.
- Remove any self-loops and label each remaining edge with a weight representing its total transition probability (the weights are normalized to integers for simplicity).
- Collapse all multiple directed edges between two states into a single undirected edge with weight $w^k_{i,j}$ equal to the sum of the directed edges probabilities. Note that this step can be performed only if the weights are based on total transition probabilities.

The STG is thus transformed into a weighted undirected graph where the weights on the edges are proportional to the total probability of a transition between the two states connected by the edge. This will be the starting point for the state assignment algorithms.

Example 4: The transformation of the STG is illustrated in Fig. 5. Note that an edge with high conditional probability (like $s_1 \leftrightarrow s_2$) can have a weight (proportional to the total transition probability) equal or even smaller than an edge with small conditional probability (like $s_4 \leftrightarrow s_2$).

As a concluding remark for this section, we observe that the calculation of the total transition probability requires solving a system of equations of size proportional to the number of states. This can become a computational problem for large systems whose state graph is extracted from an existing synchronous network, because the number of states is exponential in the number of storage elements used. In this case symbolic techniques described in [15] can be used, allowing calculation of the steady state probabilities for very large sequential circuits.

III. State Assignment for Low Power

The main idea in our approach to this problem is to find a state assignment that minimizes the number of state variables that change their value when the FSM moves between two

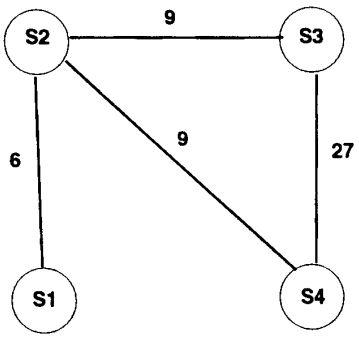

Fig. 5. Weighted graph used as a starting point for the state assignment. The weights on the edges are proportional to the total state transition probabilities.

adjacent states. Ideally, if we can guarantee that each state transition results in a single state variable change, we will have optimally reduced the switching activity associated with the registers in the given STG. We now give some examples of possible specific solutions for restricted classes of STG's, then we discuss the general problem and its exact solution.

Given an STG representing a counter, a state assignment that gives the minimum switching activity in the circuit is a Gray encoding (an encoding used for binary numbers that guarantees that two successive numbers always have adjacent codes [25]) of the states. Gray encoding is a solution only for this particular form of STG, for which the problem is quite trivial (note that in a counter inputs are irrelevant, and all transitions are equiprobable).

Given a STG of arbitrary structure, good performance in terms of reduced switching activity results from using a One-Hot encoding [24] of the states. One-hot encoding guarantees that exactly two state variables will switch for every state transition, thus achieves good results with no algorithmic effort. However, the number of state variables needed is equal to the number of states. It has been shown that shorter codes correlate to smaller area for both two-level [19] and multi-level [13] implementations and larger areas often lead to higher power dissipation. In addition, with One-hot encoding, two state variables switch for every state transition, while other codes can lead to a change of a single state variable for most transitions.

For a general solution, we need to find a method that does not assume a particular STG structure and is not heavily constrained on the number of state variables to use. We will use the probabilistic model developed in the previous section to obtain state assignments that minimize the average number of signal transitions on the state lines for a general STG.

A. Problem Formulation and Exact Algorithm

Our algorithm must be valid for an arbitrary STG, and must avoid constraints on the number of state variables used. The algorithm should be able to find the number of state variables that gives the minimum number of transitions and is close to the minimum $\lceil \log_2 n_s \rceil$, to keep the size of the combinational part small.

We can describe a state encoding as a Boolean matrix with rows corresponding to state codes and columns corresponding to state variables. Our problem of finding a state encoding that results in minimum switching activity can then be formalized as:

Find a set of Boolean row vectors $(e_i^1 \cdots e_i^{n_{var}})$, $i = 1, 2, \cdots, n_s$ that are solutions to the integer linear programming (ILP) problem:

$$\text{Min}\left(\sum_{k=1}^{n_{edges}} w_{i,j}^k \sum_{l=1}^{n_{var}} e_i^l \oplus e_j^l\right) \text{ such that} \tag{6}$$

$$\sum_{l=1}^{n_{var}} e_i^l \oplus e_j^l \geq 1 \quad \forall s_i, s_j, \qquad s_i \neq s_j \tag{7}$$

where $\oplus$ represents the XOR operation, $w_{i,j}^k$ is the weight on the edge between states s_i, s_j, and n_{var} is the number of state variables used. The cost function expresses the desire to assign adjacent codes to states with high-probability transitions. The inequalities (7) express the fact that no two states can be allowed to have the same code.

Example 5: If we decide to use a minimum length encoding for our example STG, two state variables are needed ($\log_2 4 = 2$). The encoding matrix has therefore 4 rows and 2 columns. The constraint equations are:

$$\begin{aligned} e_1^1 \oplus e_2^1 + e_1^2 \oplus e_2^2 &\geq 1 \\ e_1^1 \oplus e_3^1 + e_1^2 \oplus e_3^2 &\geq 1 \\ &\cdots \\ e_3^1 \oplus e_4^1 + e_3^2 \oplus e_4^2 &\geq 1 \end{aligned}$$

while the cost function to minimize is

$$\begin{aligned} Cost = {} & 6(e_1^1 \oplus e_2^1 + e_1^2 \oplus e_2^2) + 9(e_2^1 \oplus e_3^1 + e_2^2 \oplus e_3^2) \\ & + 9(e_2^1 \oplus e_4^1 + e_2^2 \oplus e_4^2) \\ & + 27(e_3^1 \oplus e_4^1 + e_3^2 \oplus e_4^2). \end{aligned}$$

The problem involves $2 * 4 = 8$ variables and $3 + 2 + 1 = 6$ equations.

Note that the number of state variables used, n_{var}, is an additional degree of freedom. In theory, the ILP should be solved more than once to find the n_{var} that gives the minimum cost. Because we know that the optimum lies between $\lceil \log_2 n_s \rceil$ and n_s, we can use a binary search on n_{var} to find the minimum. However, in practice area considerations (to be discussed later) force n_{var} to be close to the minimum, keeping the number of iterations on n_{var} small.

For small FSM's, the exact solution of the ILP problem can be found by using either a traditional approach [28] or BDD based techniques [29]. For larger STGs, the exact ILP solution may be unattainable, being the problem NP-complete, since the number of inequalities is $O(n_s^2)$, and the solution space to be explored is $O(n_s 2^{n_s})$. However, the exact formulation is still interesting because it gives insights into more practical heuristic solutions.

Two more observations are of interest. First, for several problems a solution with distance one between all connected states is impossible; the presence of an odd cycle in the graph is an example of constraint not satisfiable with any

distance-one encoding. However, we do not need a distance-one encoding to reach the minimum cost. Second, although the exact solution of the problem always yields the exact minimum of the cost function, it does not guarantee that the power dissipation of the synthesized circuit is minimum, because our cost function does not model the switching activity in the combinational part.

IV. Algorithms for State Encoding

The high computational complexity of the general state assignment problem has motivated the use of many heuristic approaches to its solution [10], [13], [21], [24]. We propose a column-based approach [17], [20] that takes a single state variable (one column of the encoding matrix) and tries to assign its value for each state in the graph, such that the switching activity will be minimal for the complete assignment. This is carried out iteratively for each state variable until the codes have been completely specified. The algorithm tries to give states that are linked by high-weight edges the same value for most state variables, while ensuring that each state has a unique code.

We present a semi-exact algorithm for solving the state-encoding problem in the manner described above. The algorithm performs well but is too complex to be used on large examples. We therefore derive from it a heuristic which we have used to get good results on many benchmark examples.

A. Semi-Exact Algorithm

Our semi-exact algorithm relies on the notion of indistinguishability classes. Two states having the same partial code are said to belong to the same indistinguishability class. If the maximum number of state variables that we want to use is n_{var} and we are assigning bit codes for the l-th variable, the maximum number of indistinguishable partial state codes after the assignment variable, the maximum number of indistinguishable partial state codes after the assignment must be less than $2^{n_{var}-l}$), otherwise we cannot create unique codes for this set of states with the remaining unassigned variables.

A solution to this problem can be once again formulated as an ILP. Let e^l be the l-th variable $1 \leq l \leq n_{var}$ of the state array. We call code bit, e_i^l, its value (1 or 0) for the state s_i. Moreover, we call indistinguishability classes, $C_{k,l}$, $k = 1, \cdots, n_{class_l}$ the groups of states with equal partial codes after the assignment of the preceding $l-1$ variables:

$$\text{Min}\left(\sum_{h=1}^{n_{edges}} w_{i,j}^h (e_i^l \oplus e_j^l)\right) \text{ such that} \tag{8}$$

$$\begin{cases} \sum_{s_i \in C_{k,l}} e_i^l \leq 2^{n_{var}-l} \\ \sum_{s_i \in C_{k,l}} e_i'^l \leq 2^{n_{var}-l} \end{cases} \quad \forall C_{k,l}. \tag{9}$$

This formalism can be clarified through an example.

Example 6: Consider the weighted graph in Fig. 5. We select $n_{var} = 2$. Initially, no state variable has been assigned, so all states belong to the same indistinguishability class $C_{1,1}$. We want to assign the codes for the first state variable (the first column in the encoding matrix). Since we have four states, the constraint inequalities in (9) require that we assign 0 to a pair of states and 1 to the other pair. One assignment that minimizes (8) is $e_8^1 = e_4^1 = 1$ and $e_1^1 = e_2^1 = 0$, as depicted in Fig. 6.

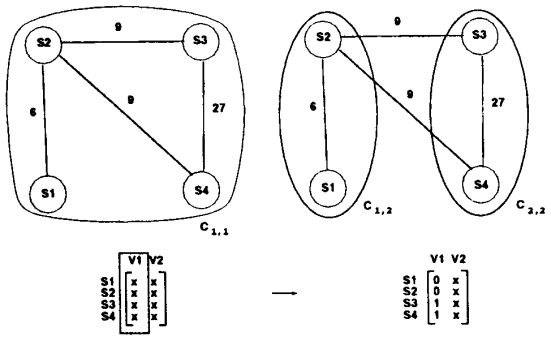

Fig. 6. $C_{1,1}$ is the first indistinguishability class when no codes have been assigned. Assigning the first variable, states s_3 and s_4 are given the same bit node. They form the class $C_{1,2}$. The other two states form the class $C_{2,2}$.

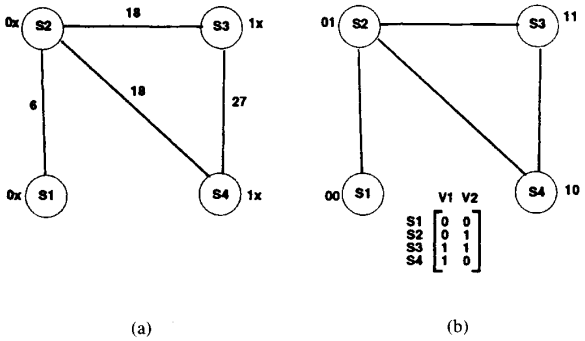

Fig. 7. (a) New edge weights after first variable assignment. (b) Final code assignment.

This approach reduces the size of the problem that must be solved at each step. However, because the column-based ILP solution does not consider the impact that the choice of one state variable has on the other state bits, the solution not globally optimal.

To improve the final outcome, we can bias the decision at each step by the results of preceding assignments, by recomputing the weights in the cost function after each variable (column assignment) using the following formula:

$$w_{i,j_{new}}^h = w_{i,j_{old}}^h (d_{i,j} + 1); \tag{10}$$

where $d_{i,j}$ is the Hamming (Boolean) distance[1] between the partially assigned codes for states s_i and s_j.

Example 7: In Fig. 7(a) the new weights after assignment of the first variable are shown. The final solution is found by assigning the second variable in a way that gives the minimum cost and distinguishes all states. This is shown in Fig. 7(b).

The column based approach produces a simpler set of ILP problems than the global ILP problem used in the exact

[1]The Hamming distance, $d_{a,b}$, between two Boolean vectors $a, b \in B^n$ is defined to be the number of bits in the same position with opposite phase: $d_{a,b} = \sum_{i=1}^{n} a_i \oplus b_i$.

method. Therefore, it can be successfully applied to medium-sized FSM's. Nevertheless, for large FSM's the number of states can be high enough to make even the semi-exact solution impractical. For this reason, we need an algorithm whose worst-case complexity is such that a good solution can be found quickly even for large and complex FSM's.

B. Heuristic Algorithm

In this section we propose a sub-optimal polynomial time state assignment algorithm that is applicable to very large FSM's. We want to eliminate the exponential complexity still remaining in the column assignment step of the semi-exact solution. Therefore, we use a much simpler heuristic that considers pairs of states and tries to assign the same state variable value to states with high transition probability. We first describe the structure of the proposed algorithm, then we discuss its performance and rationale. This is the pseudocode of the algorithm:

```
assign(S){
 sort edges by weight in decreasing order;
 for each edge {s_i, s_j} {
    /* consider pairs of states
      with high transition probability */
   if(s_i and s_j not assigned) {
    if(no Class violations) {
     /* if the number of states
      with the same bit code is not too
     high */
      x = select-bit(s_i, s_j);
      /* chooses a code for the pair
      of states */
      e_i = x e_j = x;
     /* the same bit code is given to both
      the states */
     }
     else {
       x = select-bit(s_i, s_j)
       e_i = x; e_j = x';
       /* different bit codes are given to the
        two states */
     }
  }
  else if(s_i or s_j not assigned) {
       s_h = unassigned(s_i, s_j);
       /* state whose bit code is unassigned */
       s_g = assigned(s_i, s_j)
       if(no Class violations) {
          x = select-bit(s_h);
       /* choose a bit code for the unassigned
         state */
           e_h = x;
       /* choose a bit code for the unassigned
         state */
       }
       else
          e_h = e'_g;
           /* only one choice available
           because of the class constraints */
  }
 }
}.
```

The algorithm is based on a greedy choice of the constraint to satisfy; if it is impossible to assign the same bit code to two states because an indistinguishability class becomes too big, a different code is assigned. The function select–bit makes a choice between two possible assignments based on the already assigned neighbor states. Given a pair of states (or a single state), select–bit calculates the total edge violation caused by the possible assignments (0 or 1) of the current state variable for the codes of the two states. The total edge violation for a bit code is defined as the sum of weights on all edges connecting one of the considered states with other states that have already been assigned a different bit code. The selected bit code is the one resulting in a smaller edge violation. At the end of the outermost iteration, the value (bit code) of the lth state variable (corresponding to a column of the encoding matrix) has been assigned for all states.

Example 8: For our simple STG, the greedy algorithm gives the same result as the semi-exact ILP based approach. In this case both heuristics find an exact minimum for the cost function:

$$Cost = 6 * 1 + 9 * 2 + 27 * 1 = 51$$

The minimality of the solution can easily be verified by inspection; note that the solution is not unique.

The structure of the algorithm is very simple and its execution time is always small, in fact no backtrack mechanisms are present and we do not iterate to improve a solution. The complexity of the algorithm is $O(n_s n_{edges})$. The dependence on the number of states, n_s, is from the outermost iteration, while the dependence from the number of edges, n_{edges}, is due to the iteration needed in select–bit to compute the total edge violation. One can envision cases where the greedy choice of the constraints leads to suboptimal solutions, but, in general, this heuristic gives good results.

In particular, notice that if our heuristic is run on a weighted graph where all the edges have equal weight, and n_{var} is chosen equal to n_s, the final encoding will be One-hot (all the states will have a single one in their codes and only one state will have the all-zero code), the reason being that we force the largest indistinguishability class to be reduced of at least one element at each step. One-hot encoding is therefore a particular case of the class of codes that we can generate with our algorithm.

We can now describe a general framework for the solution of the stateencoding problem using the column based approach:

```
for l=1 to n_var {
  adjust Class constraints; (9)
  assign(S);
  adjust the edge weights; (10)
}
```

where the procedure assign is either the semi-exact ILP formulation or the fast heuristic described above. Note that the greedy heuristic can be easily improved and an entire new class of algorithms can be generated. We could, for example, employ local search techniques like genetic algorithms [14] or simulated annealing to improve the results within the framework we have provided.

If we are not constrained to use the minimum number of state variables, as if often the case, we can try different

solutions for multiple values of n_{var}. Although increasing the number of state variables will likely violate fewer constraints, the n_{var} should still be kept close to the minimum to avoid an explosion in complexity of the combinational part of the FSM.

Another approach is to use the greedy heuristic for fast iteration over n_{var} to find its optimal value. Once the best n_{var} has been found, more powerful and expensive algorithms can be applied in order to improve the optimality of the result.

For extremely large FSM's the number of states could become too large even for the simple polynomial heuristic. In this case symbolic techniques similar to those presented in [15] should be used. Notice however that in this case the computational bottleneck is mainly in the calculation of the probability vector that has a complexity super-linear in the number of states and requires floating point computation.

C. Area-Related Cost Metrics

Up to this point, we have used the weighted sum of the Boolean distances between state codes as the cost function. This only minimizes the switching activity in the sequential portion of the FSM (the latches). The power dissipation is also dependent on the structure of the combinational part of the final synthesized FSM. Neglecting area considerations in the cost function may lead to non-minimal area implementations with total power dissipation close to that obtained using traditional area-related state-assignment techniques. By adding an area constraint to our cost metric, we can obtain additional power savings from more efficiently implemented combinational logic.

We therefore need to introduce additional constraints to obtain an area-minimal realization of the combinational part of the network. To tackle this problem, we incorporated metrics for minimal area into our cost function, similar to those proposed in MUSTANG [21] and later upon improved in JEDI [16]. Two different metrics are provided: a fanout-oriented metric, well suited for FSM's with a small number of inputs and a large number of outputs, and a fan-in-oriented metric that performs better in the opposite case.

Details of how the metrics are computed are presented in [21]. However two points are worth remarking on:

- The area constraints are expressed with edge weights exactly like the power constraints, and we can allow specification of different trade-offs in terms of their relative importance according to the overall design objectives. To do that, a new parameter $\alpha \leq 1$ has been introduced, specifying the relative importance of power with respect to area constraints. The edge weights on the graph are then calculated using the following equation:

$$w_{i,j} = (1-\alpha) w_{i,j}^{area} + \alpha w_{i,j}^{power} \qquad (11)$$

  where the weights $w_{i,j}^{power}$ are calculated with the algorithms presented earlier, while $w_{i,j}^{area}$ are calculated using the heuristics described in [21].
- Even if our edge weight calculation for area minimization is similar to the one proposed in MUSTANG, our state assignment algorithm is column based, and this allows to dynamically adjust the weights, resulting in a potentially more effective state assignment.

TABLE I
COMPARISON BETWEEN POW3 AND JEDI AFTER MULTIPLE LEVEL OPTIMIZATION: CIRCUIT NAME, NUMBER OF STATE VARIABLES, RATIO OF TOTAL NUMBER OF TRANSITIONS, PERCENT REDUCTION IN TOTAL TRANSITIONS, RATIO OF NUMBER OF STATE TRANSITIONS AND REDUCTION IN STATE TRANSITIONS

Circuit	n_{var}	Area JEDI/POW3	Total transitions JEDI/POW3	% Reduction	State transitions JEDI/POW3	% Reduction
bbara	4	67/69	3630/3448	5	327/294	10
bbsse	4	126/131	8871/7970	11	1033/851	18
bbtas	3	25/25	2971/2690	10	610/456	26
dk14	4	120/114	7296/7083	3	1403/1104	22
dk17	5	76/77	5548/5463	2	1337/1081	19
dk512	5	67/87	7650/4825	38	2355/1538	35
donfile	5	102/214	5231/4573	14	1743/1378	62
planet	6	697/665	27859/19771	30	3204/1240	62
planet1	6	708/697	25735/16306	38	3205/1278	61
s1488	6	742/727	14073/13123	7	628/341	55

In conclusion, to obtain low power dissipation in the final circuit, area must sometimes be taken into account. However, our experiments have shown that using high values of α in (11) typically give the best results, implying that there is a strong correlation between the power-related cost metric and the actual power dissipation.

V. IMPLEMENTATION AND RESULTS

We implemented the heuristic algorithm and applied it to some benchmark circuits. We used a standard linear algebra package [26] to find the total transtion probabilities in the STG's. We then applied our state assignment algorithm, POW3, on the weighted graph to obtain a state encoding. We then used the SIS [27] standard script script.rugged to obtain a multi-level implementation of the state-assigned FSM. We ran an area-oriented state assignment program, JEDI [16], on the same benchmarks, following the same procedure to generate an implementation. The implementations were then simulated with random patterns using the MERCURY [8], a delay-based gate-level simulator, to measure the circuit activity, which gives a good estimate of the power consumption of the real circuits. The results are shown in Table I.

For all benchmarks, our state assignment algorithm produced circuits with less switching activity than those produced by JEDI. In most cases, the area penalty (linked to the number of literals in the network) was small.

Fig. 8 compares area overhead with power reduction. If we call $M_{min\,A}$ the minimal area implementation obtained with JEDI and $M_{min\,P}$ the minimal power implementation obtained with POW3, the plot shows:

- The area ratio $A_{M_{min\,P}}/A_{M_{min\,A}}$ (Literal increase).
- The total transition count ratio $PT_{M_{min\,A}}/PT_{M_{min\,P}}$ (Decrease in total transitions).
- The state transtion count ratio $PS_{M_{min\,A}}/PS_{M_{min\,P}}$ (Decrease in state transitions).

It is clear that if the area overhead is large, the reduction in power dissipation is less significant, thus showing that the power dissipated in the combinatorial part plays an important role in the total power balance.

Fig. 9 plots the average reduction in power dissipation as a function of the number of state bits. The reader can observe

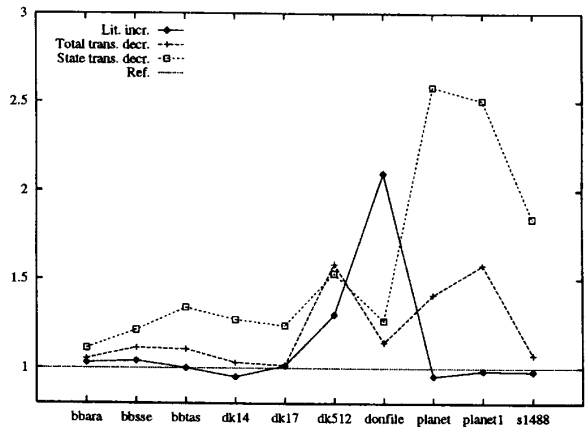

Fig. 8. Increase in area of the low-power implementation and corresponding decrease in transition count (for both the complete circuit and the state variables only).

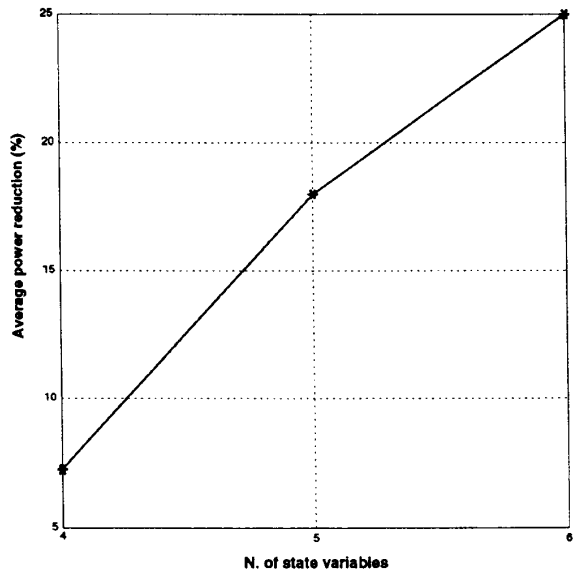

Fig. 9. Average power reduction as function of the number of state variables.

that our methods produce in the average better results for larger circuits, for which low power consumption is even more important.

All results use transition count as the estimate of power, because we have not mapped the circuits using a technology library. The algorithm described above is intended to be a preprocessing step in a complete synthesis tool that includes a low-power driven technology mapper, which we did not have. At this regard, notice that the power in the combinational part of the FSM can be divided in power dissipated in useful transition and glitch power. The reduced switching activity of the state lines can be used to decrease both these quantities, in fact the intuition suggests that a network with low switching activity on part of the inputs and outputs could be synthesized with also reduced internal switching activity. This is still an open problem, but accurate power estimation techniques such as those presented in [6] could allow combinational synthesis and library binding tools to exploit the low-activity property of our state assignments.

The last column of Table I shows the reduction in switching activity on the state lines. Note that if power dissipation in the memory elements is significantly higher than the power dissipated in combinational gates, the power reduction of our implementation becomes more significant. Also, because the state lines have low activity, algorithms for optimization of the combinational logic can exploit this information for further power savings.

Two additional parameters help our algorithms in the search for optimal results. First, we can control the number of state variables used in the encoding. In all the examples we tried, the minimum number of state variables gave the best result, because increasing the number of state variables resulted in an area overhead that overcame the (small) reduction in average number of transitions on the state lines. Second, our algorithms can accept different values of the parameter α, controlling the relative importance of power and area in the cost function. Our experiments with different values of α showed that setting $\alpha \geq 0.7$ produced the lowest power implementations. In fact, in some cases, setting $\alpha = 1$ resulted in a final implementation that was as small as the implementations obtained by using JEDI for state assignment. This confirms that the area-related cost metrics used are not very accurate, and more work has to be done in order to better estimate the area of multilevel implementations.

VI. Conclusions

We have presented a general framework for state assignment for low-power. Within that framework, we described a set of state assignment algorithms targeting low power consumption, varying in their exactness and computational complexity. We implemented one of the algorithms described, and ran it on standard benchmark circuits. We found that it compares favorably with existing state assignment tools targeting minimal area implementation, achieving a 16% average reduction of total switching activity, and a 34% average reduction of state variable related switching activity. We also explored the trade-offs between power-related and area-related cost metrics in the context of our algorithm. Our results confirm that state assignment has a large impact on power dissipation in the overall circuit.

The framework we have developed is general enough to open the way for exploration of new algorithms for the optimization of the FSM's combinational part that take into account the reduced switching activity on the present state inputs (latch outputs). It is our opinion that even larger power savings can be attained if new cost metrics are employed that relate more directly the power dissipated in the combinational part with the codes assigned to the states. Moreover, the state assignment step should be integrated with logic synthesis and library binding algorithms that can optimally exploit the reduced switching activity of the state variable inputs.

ACKNOWLEDGMENT

We especially would like to thank P. Siegel for her invaluable feedback and her suggestions.

PROOF OF THEOREM 2

First, we prove that the Markov chain is irreducible. Since every state has a transition to the reset state s_0, and every state is reachable from s_0, we can always reach a state from any other state using a path through s_0.

Second, we need to show that the chain is aperiodic. The reset state is aperiodic, because it can be reached from every other state and from itself in one step. It is possible to show that, if a Markov chain is irreducible and one of its states is aperiodic then all its states are aperiodic [23], therefore the aperiodicity of the chain is proven.

Finally, all the states are recurrent non null because they are reachable and from every other state including themselves with probability greater than zero.

Notice that the assumptions on the STG can be relaxed. We do not need to impose that all the states can be reached from s_0 if we assume that the initial state is always s_0. In this case the states that cannot be reached from s_0 are unreachable, consequently their probability is zero. Again this is a reasonable assumption because it means that the machine, upon powerup, starts in a known reset state. For a more general class of FSM's, the notion of steady state probabilities needs some modifications in order to cope with disconnected components and periodic states. A complete treatment of this topic is given in [12].

REFERENCES

[1] A. Chandrakasan, S. Sheng, and R. Brodersen, "Low-Power CMOS digital design," *IEEE J. Solid-State Circuits*, vol. 27, no. 4, pp. 473–484, Apr. 1992.

[2] D. Liu and C. Svensson, "Trading speed for low power by choice of supply and threshold voltages," *IEEE J. Solid-State Circuits*, vol. 28, no. 1, pp. 10–17, Jan. 1993.

[3] K. Roy and S. Prasad, "Circuit activity based logic synthesis for low power reliable operations," *IEEE Trans. VLSI Syst.*, vol. 1, no. 4, pp. 503–513, Dec. 1993.

[4] J. Monteiro, S. Devadas, and A. Gosh, Retiming sequential circuits for low power, in *Proc. IEEE Int. Conf. Computer-Aided Design*, Nov. 1993, pp. 398–402.

[5] K. Kentzer, A. Ghosh, S. Devadas, and J. White, "Estimation of average switching activity in combinational and sequential circuits, in *Proc. Design Automation Conf.*, June 1992, pp. 253–259.

[6] C. Y. Tsui, M. Pedram, and A. M. Despain, "Exact and approximate methods for calculating signal and transition probabilities in FSMs," in *Proc. Design Automation Conf.*, June 1994, pp. 18–25.

[7] S. Devadas, A. Shen, A. Ghosh, and K. Keutzer, "On average power dissipation and random pattern testability," in *Proc. IEEE Int. Conf. Computer-Aided Design*, Nov. 1992, pp. 402–407.

[8] G. De Micheli, D. Ku, F. Mailhot, and T. Truong, "The Olympus synthesis system," *IEEE Design Test Comp.*, vol. 7, no. 5, pp. 37–53, Oct. 1990.

[9] M. Pedram, C. T. Tsui, and A. Despain, "Technology decomposition and mapping targeting low power dissipation," in *Proc. Design Automation Conf.*, June 1993, pp. 68–73.

[10] B. Eschermann, "State assignment for hardwired control units," *ACM Computing Surveys*, vol. 25, no. 4, pp. 415–436, Dec. 1993.

[11] M. A. Cirit, "Estimating dynamic power consumption of CMOS circuits," in *Proc. IEEE Int. Conf. Computer-Aided Design*, Nov. 1987, pp. 534–537.

[12] G. Hachtel, E. Macii, A. Pardo, and F. Somenzi, "Symbolic algorithms to calculate steady-state probabilities of a finite state machine," in *Proc. IEEE European Design Test Conf.*, Feb. 1994, pp. 214–218.

[13] X. Du, *et al.*, "MUSE: A MUltilevel Symbolic Encoding Algorithm for State Assignment," *IEEE Trans. Computer-Aided Design*, vol. 10, no. 1, pp. 28–38, Jan. 1991.

[14] E. Olson and S. Kang, "State assignment for low-power synthesis using genetic local search," in *Proc. IEEE Custom Integrated Circuit Conf.*, May 1994, pp. 140–143.

[15] G. Hachtel, *et al.*, "Re-encoding sequential circuits to reduce power dissipation," in *1994 Int. Workshop Low Power Design Napa*, Apr. 1994, pp. 69–73.

[16] B. Lin and A. R. Newton, "Synthesis of multiple-level logic from symbolic high-level description languages," in *Proc. IEEE Int. Conf. Comp. Des.*, Aug. 1989, pp. 187–196.

[17] G. De Micheli, "Symbolic design of combinational and sequential logic circuits implemented by two-level logic macros," *IEEE Trans. Computer-Aided Design*, vol. 5, no. 4, pp. 597–616, Oct. 1986.

[18] P. Ashar, S. Devadas, and A. R. Newton, *Sequential Logic Synthesis.* Norwell, MA: Kluwer Academic, 1992.

[19] G. De Micheli, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Optimal state assignment for finite state machines, *IEEE Trans. Computer-Aided Design*, vol. CAD-4, no. 3, pp. 269–284, July 1985.

[20] T. Dolotta and E. McCluskey, "The coding of internal states of sequential machines," *IEEE Trans. Electron. Comput.*, vol. EC-13, pp. 549–562, Oct. 1964.

[21] A. R. Newton, S. Devadas, H. Ma, and A. Sangiovanni-Vincentelli, "MUSTANG: State assignment of finite state machines targeting multi-level logic implementations," *IEEE Trans. Computer-Aided Design*, vol. 7, no. 12, pp. 1290–1300, Dec. 1988.

[22] A. Sangiovanni-Vincentelli and T. Villa, "NOVA state assignment of finite state machines for optimal two-level logic implementation," *IEEE Trans. Computer-Aided Design*, vol. 9, no. 9, pp. 905–924, Sept. 1990.

[23] K. Trivedi, *Probability and Statistics with Reliability, Queuing and Computer Science Applications.* Englewood Cliffs, NJ: Prentice–Hall, 1982.

[24] G. De Micheli, *Synthesis and Optimization of Digital Circuits.* New York: McGraw–Hill, 1986.

[25] E. McCluskey, *Logic Design Principles.* Englewood Cliffs, NJ: Prentice–Hall, 1986.

[26] The Math Works, Inc. *The Student Edition of MATLAB.* Englewood Cliffs, NJ: Prentice–Hall, 1992.

[27] E. Sentovich, *et al.*, "Sequential circuit design using synthesis and optimization," in *Proc. IEEE Int. Conf. Comput. Design*, Oct. 1992, pp. 328–333.

[28] G. Nemhauser and L. Wolsey. *Integer and Combinatorial Optimization.* New York: Wiley, 1988.

[29] S.-W. Yeong and F. Somenzi, "A new algorithm for 0-1 programming based on binary decision diagrams," in *Logic Synthesis Workshop*, Japan, 1992, pp. 177–184.

[30] S. Malik, V. Tiwari, and P. Ashar, "Technology mapping for low power," in *Proc. Design Automation Conf.*, June 1993, pp. 74–79.

Luca Benini (S'94) received a Laurea degree in electrical engineering from University of Bologna, Italy, in 1991 and received and the M.S. degree in electrical engineering from Stanford University, Stanford, CA, in 1994.

He is a Ph.D. candidate in electrical engineering at Stanford University, where his dissertation is on synthesis for low power. Prior to arriving at Stanford, he worked as a Research Assistant on simulation techniques for power estimation at the Department of Electronics and Computer Science of the University of Bologna during 1991–1992. His current research interests are in the area of computer-aided design and simulation of digital IC's, specifically in the design of low power-systems, algorithms for the automatic synthesis of low-power circuits, and in tools for accurate estimation of the power dissipation in large digital systems. He is also interested in multiple-level logic synthesis, algorithms for optimal state assignment, technology mapping, and probabilistic simulation.

Giovanni De Micheli (S'79–M'79–SM'89–F'94) received the nuclear engineer degree from Politecnico di Milano, Italy, in 1979 and the Ph.D. degree in electrical engineering and computer science from University of California at Berkeley, in 1983.

He is the Associate Professor of Electrical Engineering, and by courtesy, of Computer Science at Stanford University. Previously he held positions at the IBM T. J. Watson Research Center, Yorktown Heights, NY, at the Department of Electronics of the Politecnico di Milano, Italy, and at Harris Semiconductor, Melbourne, FL. His research interests include several aspects of the computer-aided design of integrated circuits and systems, with particular emphasis on automated synthesis, optimization, and validation.

Dr. De Micheli is the author of *Synthesis and Optimization of Digital Circuits* (New York: McGraw–Hill, 1994), co-author of *High-level Synthesis of ASICs under Timing and Synchronization Constraints* (Norwell, MA: Kluwer, 1992), and co-editor of *Design Systems for VLSI Circuits: Logic Synthesis and Silicon Compilation* (Martinus Nijhoff Publishers). He was also co-director of the Advanced Study Institute on Logic Synthesis and Silicon Compilation, held in L'Aquila, Italy, under the sponsorship of NATO in 1986 and in 1987. He was granted a Presidential Young Investigator award in 1988. He received the 1987 Best Paper Award for the best paper published in IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN and two Best Paper awards at the Design Automation Conference, in 1983 and 1993.