# Low-Power State Assignment Targeting Two- and Multilevel Logic Implementations

Chi-Ying Tsui, *Member, IEEE*, Massoud Pedram, *Member, IEEE*, and Alvin M. Despain *Fellow, IEEE*

***Abstract*— The problem of minimizing power consumption during the state encoding of a finite-state machine is addressed. A new power cost model for state encoding is proposed, and encoding techniques that minimize this power cost for two- and multilevel logic implementations are described. These techniques are compared with those that minimize area or the switching activity at the present state bits. Experimental results show significant improvements.**

## I. Introduction

IN THIS era of portable electronics applications, power consumption has become an important criterion for designing electronic circuits. Recently, optimization methods for low power have been developed for different levels of design hierarchy including technology selection, architectural transformation, logic synthesis, and physical design.

In this work, we address the problem of minimizing power consumption of a sequential machine. Since the switching activity and hence the power consumption of a finite-state machine (FSM) is strongly dependent on the state transition behavior and thus the state encoding of the machine, we are particularly interested in developing encoding algorithms that will ultimately give a low-power implementation after logic synthesis.

It is known that the state assignment of an FSM has a significant impact on the area of the final implementation. Intensive research on minimizing area during the state assignment has been conducted in the past ten years. The problem is NP-hard; indeed, the optimum assignment can be found by exhaustively enumerating all the possible assignments, carrying out logic synthesis for each assignment, and then picking the one that has the least area. This method is computationally too expensive. Approximate methods have therefore been developed that rely on approximate prelogic synthesis cost functions in order to avoid the expensive logic minimization step. Traditionally, state assignment has been formulated as a hypercube embedding problem. Initially, codes of minimum length are assigned to the state variable and then logic optimization is performed on the combinational part implementation derived from the encoding. In contrast to this approach, DeMicheli *et al.* [1] proposed an innovative paradigm in which one-hot codes are assigned to states and a minimum symbolic (multivalued) cover of the machine is then generated by output-disjoint minimization. This symbolic cover defines a set of face embedding constraints that require certain states be given codes that lie on the same face of a hypercube of minimum (or given) dimensionality. For two-level logic implementation, if these constraints are satisfied, then the number of cubes in the minimum binary cover of the final implementation will be upper bounded by that in the minimum symbolic cover. The minimum area state encoding problem for two-level logic is then relaxed to the problem of finding the minimum number of encoding bits such that all the constraints are satisfied. DeMicheli [2] used a heuristic row encoding technique to solve this problem. Villa *et al.* [3] employed the notion of face-posets to tackle this problem. Yang *et al.* [4] transformed the problem into a unate covering problem (covering seed dichotomies by a minimum-cost set of prime dichotomies) and solved it using a heuristic technique. Devadas *et al.* [5] proposed an exact method based on the concept of generalized prime implicants.

A variation on the state assignment problem is the *bit-constrained state assignment* where an encoding is to be found that minimizes the area subject to the constraint that the number of encoding bits is no larger than a user-specified value. This problem is again NP-hard, and heuristic methods are used to obtain a solution. A common approach is to use simulated annealing [6].

As in any implementation of simulated annealing, we need to specify the initial solution, the move generation, the cost calculation, the cooling schedule, and the stopping criterion. For state assignment, the initial solution is some random state encoding; moves are generated by randomly flipping bits of the current encodings; the cost is calculated so as to mimic the cost after the logic optimization targeting two- or multilevel logic realizations; the temperature is decreased according to the simple rule $T_{\text{new}} = \alpha T$, where $0 < \alpha < 1$. The search at a given temperature is terminated after a fixed, known number of moves, while the simulated annealing procedure is terminated if in the last $k$ steps (in our case, $k = 4$) no improvement in the cost function is achieved. Among the above, the most critical and computationally demanding procedure is the cost-calculation procedure. The quality of the solution depends on how well the cost function captures the final objective function.

For two-level logic implementation, NOVA [3] used the number of unsatisfied constraints weighted by their occurrence frequency in the symbolic cover as the cost function. For mul-

Manuscript received April 11, 1995; revised September 3, 1998. This work was supported in part by the Defense Advanced Research Project Agency under Contract DARPA F33615-95-C1627 and in part by Hong Kong RGC CERG under Grant HKUST6019/97E. This paper was recommended by Associate Editor M. Fujita.

C.-Y. Tsui is with the Department of Electrical and Electronic Engineering, Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong.

M. Pedram and A. M. Despain are with the Department of Electrical Engineering—Systems, University of Southern California, Los Angeles, CA 90089 USA.

Publisher Item Identifier S 0278-0070(98)09366-X.

tilevel logic implementation, a cost function that reflects higher cube sharing is used. In particular, JEDI [7], MUSTANG [8], and MUSE [9] assigned weights to pairs of states that reflect the number of literals that can be saved if the pair of states is encoded with a specific Hamming distance. They then use the sum of the weights over all pairs of states as the cost function.

In CMOS circuits, dynamic power consumption of a gate is given by

$$P_{\text{avg}} = \frac{0.5 V_{dd}^2 C_{\text{load}} E_{\text{switching}}}{T_{\text{cycle}}} \tag{1}$$

where $T_{\text{cycle}}$ is the cycle time and $C_{\text{load}}$ and $E_{\text{switching}}$ are the load capacitance that the gate is driving and the expected switching activity at the gate output, respectively. State encoding for low power is harder than that for minimum area since it has to consider both area and switching activity at the same time and the switching activity is not known until the encoding is determined.

Roy *et al.* addressed the problem of reducing the switching activity during state assignment in [10]. They assumed that the power consumption is proportional to the switching activity on the state bit lines of the machine and hence used the following cost function:

$$\sum_{S_i, S_j \in S} \text{tp}_{ij} H(S_i, S_j)$$

where $\text{tp}_{ij}$ is the global state transition probability from state $S_i$ to state $S_j$ and $H(S_i, S_j)$ is the Hamming distance between the encodings of the two states. We denote the encoding obtained by this method as the minimum weighted Hamming distance encoding (MWHD). The shortcoming of the above approach is that it minimizes the switching on the present state bits without any consideration on the loading of the state bits and the power consumption in the resulting two- or multilevel logic realization of the next-state and output parts of the FSM.

From (1), power consumption depends on *both* the capacitance and the switching activity of the gate (in fact, the product of the two). So methods only minimizing one of these two parameters only solve half of the problem and hence will not give an optimal solution. Instead, we have to minimize the weighted switching activity of the circuits (weighted by the capacitance load) in order to reduce the power consumption. In an attempt to account for power consumption in the combinational logic, Olson *et al.* [11] used a linear combination of the switching activity and the number of literals as cost function. The drawback of this approach is that it considers the loading and switching activity separately and hence does not directly address the problem of minimizing the weighted switching activity. In addition, since the number of literals and the switching activity are two quantities of very different nature, a linear combination of the two may not work very well.

In this paper, we consider the bit-constrained state assignment problem for low power. Simulated annealing is used for the search strategy. We first present a power cost model for state assignment that considers both the capacitive loading and the switching activity simultaneously. We then propose accurate power cost functions for both two- and multilevel logic implementations. For two-level logic using programmable logic array (PLA) implementations, the dichotomy-based approach of [4] is extended to calculate the proposed power cost function. The impact of the PLA type on the power cost function is also described. For multilevel logic implementation, the cost function of [7] is modified to take into account the weighted switching activity at the inputs of the FSM.

The remainder of this paper is organized as follows. Section II gives our terminology. The power cost model is described in Section III. The low-power state assignment algorithms for two-level logic and multilevel logic using this power cost model are presented in Sections IV and V. Experimental results and conclusions are given in Sections VI and VII.

## II. Terminology

An FSM is characterized by five-tuples $(X, Y, S, \lambda, \eta)$, where:

- $X$ $\{x_i \mid i = 1, n_X\}$ set of primary inputs;
- $Y$ $\{y_i \mid i = 1, n_Y\}$ set of primary outputs;
- $S$ $\{S_i \mid i = 1, n_S\}$ set of internal states;
- $\lambda$ $X \times S \rightarrow Y$ output function;
- $\eta$ $X \times S \rightarrow S$ next-state function (Mealy machine).

The state encoding length for $S$ is denoted by $n_E$ ($\geq \lceil \log_2 n_S \rceil$). The FSM is represented by a *state transition table* $M = \{m_i \mid m_i = (x_i, S_i, S_i', y_i), i = 1, \cdots, n_M\}$, where $S_i' \in S$ is the next state. Each entry $m_i \in M$ is a symbolic cube (or a multivalued cube) of the FSM. The state transition table can be represented by a *state transition graph* $G(V, E, W(E))$, where:

- $V$ $\{v_i \mid v_i \in S\}$ set of vertices (state);
- $E$ $\{e_{i,j} = (v_i, v_j) \mid v_i, v_j \in V\}$ set of edges joining $v_i$ and $v_j$ where there is a transition from $S_i$ to $S_j$;
- $W(E)$ $\{x_i/y_j \mid x_i \in X, y_j \in Y\}$ labels on each edge denoting transition under input $x_i$, producing output $y_j$.

The FSM can be also viewed as a discrete-state, discrete-transition Markov process. The *state probability* $P_{S_i}$ of a state $S_i$, which is defined as the probability that the state is visited in an arbitrarily long random sequence, can be obtained by solving the following Chapman–Kolmogorov equations:

$$P_{S_i} = \sum_{j \in \text{IN\_STATE}(i)} p_{ji} P_{S_j} \quad i = 1, 2, \cdots, M-1$$
$$1 = \sum_j P_{S_j}$$

where $\text{IN\_STATE}(i)$ is the set of fanin states of $i$ in the signal transition graph and $p_{ji}$ is the conditional probability that the next state is $S_i$ given that the present state is $S_j$.

The *global state transition probability* $\text{tp}_{S_i, S_j}$ between two states $S_i$ and $S_j$ is defined as the probability that the transition from $S_i$ to $S_j$ occurs in an arbitrarily long sequence and is

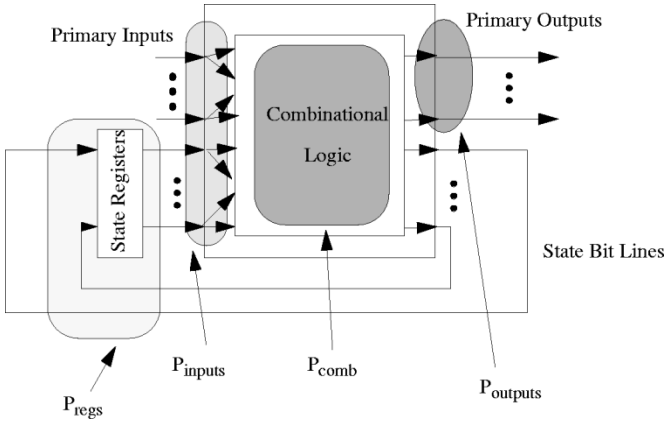



Fig. 1. Power model for finite-state machines.

given by

$$\mathrm{tp}_{S_i,S_j} = P_{S_i} p_{ij}.$$

The notion of global state transition probability can be generalized to transition between two sets of states. The global state transition probability between two sets of states $X_i \subset S$ and $X_j \subset S$ is defined as

$$TP(X_i \rightarrow X_j) = \sum_{S_i \in X_i} \sum_{S_j \in X_j} tp_{S_i,S_j}. \tag{2}$$

The switching activity of the state bit line depends on the state encoding and the state transition probabilities. The transition probability $\tau p_{b_i}(E_{b_i})$ of a state bit line $b_i$ is given by

$$\tau p_{b_i} = TP(\mathrm{ONE}_i \rightarrow \mathrm{ZERO}_i) + TP(\mathrm{ZERO}_i \rightarrow \mathrm{ONE}_i) \tag{3}$$

where $\mathrm{ONE}_i$ and $\mathrm{ZERO}_i$ are the set of states whose encodings have the $i$th bit equal to one and zero, respectively.

## III. A Power-Consumption Model for FSM's

Fig. 1 shows a typical implementation of a finite-state machine that consists of a combinational circuit and a set of state registers. The sources of power consumption in this implementation are highlighted in the figure and explained below.

$P_{\mathrm{reg}}$ is the power consumption at the state registers and is given by

$$P_{\mathrm{reg}} = \sum_{b_i \in \mathrm{state\_bits}} C_{\mathrm{reg}} E_{b_i} \tag{4}$$

where $C_{\mathrm{reg}}$ is the input capacitance of the state register and $E_{b_i}$ is the switching activity of state bit line $b_i$, which is calculated from (3).

$P_{\mathrm{inputs}}$ is the power consumption required to drive the combinational inputs and the state bit inputs of the combinational part of the machine. It depends on the switching activity of the state bit lines and the number of combinational input and state bit literals in the logic implementation. It is given by

$$P_{\mathrm{inputs}} = \sum_{b_i \in \mathrm{state\_bits}} n_i C_{\mathrm{lit}} E_{b_i} + \sum_{j \in \mathrm{PI}} n_j C_{\mathrm{lit}} E_j \tag{5}$$

where $n_i$ and $n_j$ are the number of literals that input lines $b_i$ and $j$ are driving, $C_{\mathrm{lit}}$ is the effective capacitance due to each literal, $E_{b_i}$ and $E_j$ are the switching activities of $b_i$ and $j$, and PI is the set of combinational inputs.

$P_{\mathrm{comb}}$ is the power consumption in the combinational circuit itself and is given by

$$P_{\mathrm{comb}} = \sum_{n \in \mathrm{NODES}} C_n E_n \tag{6}$$

where $C_n$ is the effective capacitance that node $n$ is driving, $E_n$ is the switching activity of node $n$, and NODES is the set of internal nodes of the circuit.

$P_{\mathrm{outputs}}$ is the power consumption at the combinational outputs of the circuits and is given by

$$P_{\mathrm{out}} = \sum_{o \in \mathrm{PO}} C_o E_o \tag{7}$$

where $C_o$ is the effective capacitance that output $o$ is driving, $E_o$ is the switching activity of output $o$, and PO is the set of circuit primary outputs.

The total power consumption of the finite state machine is therefore equal to

$$P_{\mathrm{total}} = P_{\mathrm{reg}} + P_{\mathrm{inputs}} + P_{\mathrm{comb}} + P_{\mathrm{outputs}}. \tag{8}$$

Under a zero-delay model where glitches are neglected, $E_o$ only depends on the state transition probabilities and is independent of the state encoding and the circuit implementation. In addition, $C_o$ is fixed and independent of the implementation. Therefore, $P_{\mathrm{out}}$ is constant and independent of the state encoding and can be dropped when comparing the power costs for different state encodings. We therefore minimize $P_{\mathrm{reg}} + P_{\mathrm{inputs}} + P_{\mathrm{comb}}$.

State encoding schemes that minimize the Hamming distance between state pairs with high transition probabilities tend to minimize $E_{b_i}$ and hence $P_{\mathrm{reg}}$. On the other hand, these schemes may increase the fanouts of state bit lines and the number of nodes in the combinational part and hence increase $P_{\mathrm{inputs}}$ and $P_{\mathrm{comb}}$, which will in turn offset the reduction in $P_{\mathrm{reg}}$. As a result, these methods do not in general produce power-optimal assignments. Similarly, state encoding schemes that minimize area tend to reduce the fanouts of state bit lines and the number of nodes in the combinational part. They do not, however, consider the switching activity, and again do not produce power-optimal assignments.

## IV. Two-Level Logic Implementation

In this paper, we tackle the bit-constrained state assignment problem, i.e., given the number of state bits, we find the encoding that gives the smallest power consumption. This is a similar problem to that solved by NOVA with the exception that here we minimize power consumption, whereas NOVA minimizes area. This problem is, however, different from the encoding problem in [1], where the authors find a minimal length encoding that satisfies the encoding constraints after symbolic minimization.

In NOVA, the objective function is the number of unsatisfied constraints weighted by their occurrence frequency in the

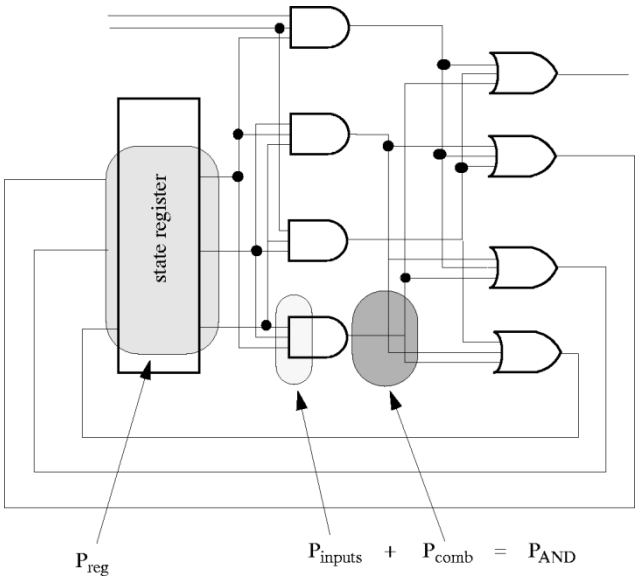



Fig. 2. Power model for two-level logic.

symbolic cover. For low-power encoding, we should derive an appropriate cost function. Since NOVA has several options to decide which encoding method is to be used, to allow a fair comparison between our low-power encoding method and NOVA, simulated annealing is used as the encoding framework for both methods. In the following subsection, we formulate the cost function for the low-power state encoding.

### A. The Cost Function

Fig. 2 shows a typical schematic for a two-level logic circuit, for which there is one level of AND and one level of OR gates. The power consumption at the outputs of the OR gates that drive the state registers can be included in $P_{\text{reg}}$, while $P_{\text{inputs}}$ and $P_{\text{comb}}$ can be lumped into a single term $P_{\text{AND}}$, where

$$P_{\text{AND}} = \sum_{\text{BI}\in\text{logic\_cover}} P_{\text{BI}} \tag{9}$$

and BI is a binary implicant of the logic cover.

Let the binary representation of BI consists of combinational inputs $x = x_1 \cdots x_n$ and state bit input $b_1 \cdots b_m$. $P_{\text{BI}}$ is given by

$$P_{\text{BI}} = P_{\text{inputs}_{\text{BI}}} + P_{\text{comb}_{\text{BI}}} \tag{10}$$

where

$$P_{\text{inputs}_{\text{BI}}} = \sum_{i=1}^{n} C_{\text{AND}} E_{x_i} + \sum_{j=1}^{m} C_{\text{AND}} E_{b_j} \tag{11}$$

and

$$P_{\text{comb}_{\text{BI}}} = n_{\text{OR}} C_{\text{OR}} E_{\text{BI}} \tag{12}$$

where $C_{\text{AND}}$ and $C_{\text{OR}}$ are the capacitance loading of a literal in the AND plane and OR plane, respectively, $n_{\text{OR}}$ is number of OR gates driven by the BI, and $E_{\text{BI}}$ is the switching activity at the output of BI. For two-level logic circuits that are implemented in dynamic PLA structure, $E_{\text{BI}}$ is obtained by calculating the signal probability of BI. BI is expressed as the product of PPI and SB. PPI is an AND function whose immediate support set consists of primary inputs only. Similarly, SB is an AND function whose immediate support set consists of state bits only. If the primary inputs are temporally independent and uncorrelated with the state bits, the signal probability of BI can be calculated as the product of the signal probabilities of PPI and SB. The signal probability of PPI is in turn equal to the product of the signal probabilities of its immediate support set, while the signal probability of SB is equal to the sum of the state probabilities of the states in a state group whose encodings force SB to evaluate to one.

The total power cost function for a two-level logic circuit is therefore equal to

$$P_{\text{reg}} + \sum_{\text{BI}\in\text{logic\_cover}} P_{\text{BI}}. \tag{13}$$

The type of PLA used for the implementation has a direct impact on the power cost calculation. For a dynamic PLA circuit using NOR–NOR structure, which is commonly used for implementing high-performance controllers in microprocessors, the next-state bit lines that drive the state register switch when the corresponding dynamic NOR gates at the OR plane switch. The dynamic NOR gate is precharged to one during the precharge period and switches only when its output is evaluated to zero during the evaluation period. Since the output of the NOR gate is $\overline{\text{NS}}_i$, the switching probability of the next-state bit line $\text{NS}_i$ is equal to prob$(\overline{\text{NS}}_i = 0)$. In addition, we have to include $P_{\text{clock}}$, which is the power consumption at the clocked transistors for the precharge and evaluation of each NOR gate. Therefore, the $P_{\text{BI}}$ for two-level logic circuits implemented using a dynamic PLA is equal to

$$P_{\text{inputs}_{\text{BI}}} + P_{\text{comb}_{\text{BI}}} + P_{\text{clock}}. \tag{14}$$

For a pseudo-NMOS PLA circuit using NOR-NOR structure, we have to include the power consumption due to the short-circuit current drawn through the NOR gate, as this is the major source of the power consumption. In this work, we only consider dynamic PLA implementation.

Given a finite-state machine, we first assign one-hot codes to the states. Then symbolic minimization is applied on the one-hot coded machine using multivalued logic minimization [12]. The result is a symbolic cover of the finite-state machine. Each element of the symbolic cover is a prime symbolic implicant. A prime symbolic implicant is a four-tuple $(X, S, S', Y)$, where $S$ is the set of states that transit to the same next state $S'$ and assert the same output $Y$ when the input combination is $X$. The set of states in $S$ defines a state group. The state group forms a face embedding constraint in which if only the codes of these states lie on the same face of a hypercube, then the symbolic implicant can be realized by a single binary cube. Fig. 3(a) shows the state transition table for a finite-state machine, and Fig. 3(b) gives the symbolic cover and the corresponding symbolic implicants after symbolic minimization.

Given a symbolic cover, we want to quickly calculate the power cost of a given encoding. $P_{\text{reg}}$ is easy to compute since $C_{\text{reg}}$ is fixed and $E_{b_i}$ can be computed from the encoding using (3). However, if we want to compute $P_{\text{AND}}$ exactly, we have

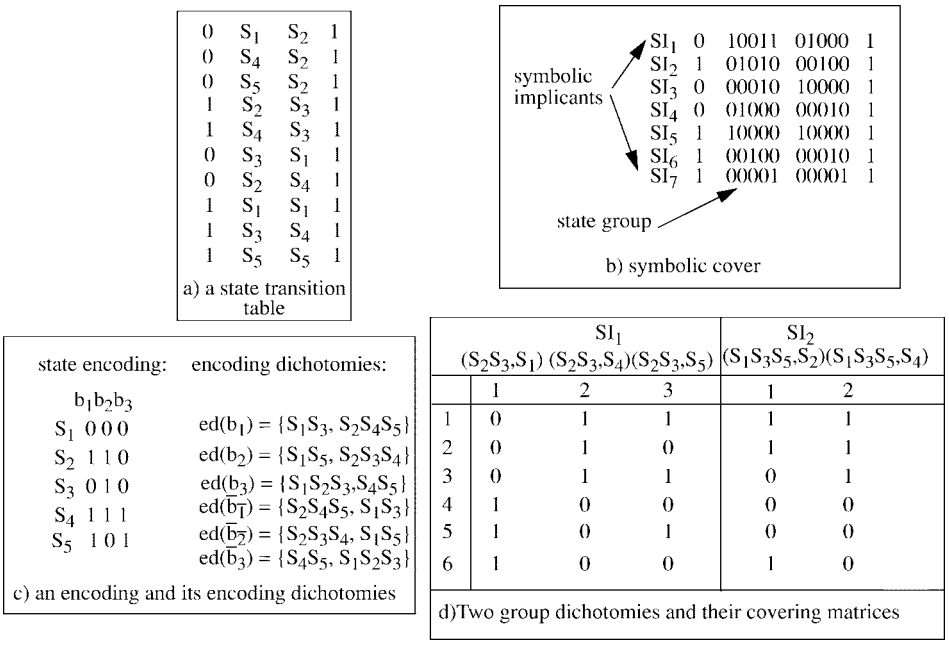



Fig. 3. Example illustrating the definitions.

to know the exact implementation, which will be known only after logic minimization. By way of compromise, we use the power cost of the symbolic implicants to approximate $P_{\text{AND}}$, as detailed next.

Let SI be a prime symbolic implicant. If SI is realized by a single binary implicant BI, then the power cost of realization of this symbolic cube is

$$P_{\text{SI}} = P_{\text{BI}}. \tag{15}$$

If the state group of SI is not satisfied by the encoding, it requires more than one binary implicant to realize SI. Let $\text{BI}_1 \cdots \text{BI}_q$ be the set of binary implicants that realize SI. Then the power cost of this realization is

$$P_{\text{SI}} = \sum_{i=1}^{q} P_{\text{BI}_q}. \tag{16}$$

$P_{\text{AND}}$ is then given by

$$P_{\text{AND}} = \sum_{\text{SI} \in \text{symbolic\_cover}} P_{\text{SI}}. \tag{17}$$

The key issue is, of course, to find the minimum power implementation of a symbolic implicant SI, i.e., finding $\text{BI}_1, \cdots, \text{BI}_q$ that minimize the power. It should be noted that although the minimum power implementation of a symbolic implicant may not be the same as its final realization after two-level logic minimization, it still serves as a good estimate of the power-savings potential of the given encoding for generating a low-power implementation of the symbolic implicant (as confirmed by our experimental results).

To obtain the minimum power implementation of a symbolic implicant, we use the concept of dichotomy that has also been used for state assignment targeting minimum area [4].

### *B. Definitions and Notation*

We use the example shown in Fig. 3 to illustrate the definitions and notation. In the following, we assume an FSM with $S$ states where inputs and outputs are denoted by $X$ and $Y$, respectively.

*Definition 4.1:* A symbolic implicant is a four-tuple $\langle X, S, S', Y \rangle$ corresponding to combinational inputs, present states, next states, and combinational outputs of the FSM, respectively. After one-hot encoding of states and symbolic minimization, we obtain a set of prime symbolic implicants such that each represents the grouping of states that are mapped by some input combination into the same next state and assert the same output. The $S$ part of a prime symbolic implicant defines a set of states and is represented by a string of $n_s$ "0's" and "1's" and is called a state group. The 1's in a state group identify the states that belong to the group. A group dichotomy corresponding to a state group is a two-block partition of states such that those states having a zero in the state group are in the left block and those having a one are in the right block. A seed dichotomy is a dichotomy where the right block has exactly one element. If a state group has $n$ 1's, its corresponding group dichotomy is split into $n$ seed dichotomies.

*Example:* In Fig. 3, there are seven group dichotomies, one for each symbolic implicant. For $\text{SI}_1$, the corresponding group dichotomy is $(S_2S_3, S_1S_4S_5)$, and the three seed dichotomies are $(S_2S_3, S_1)$, $(S_2S_3, S_4)$, and $(S_2S_3, S_5)$, respectively.

*Definition 4.2:* Given an encoding with $k$ bits, each bit $i$ defines two encoding dichotomies: one where all states whose $i$th bit are zero go to the left block of the encoding dichotomy while the remaining states go to the right block; and the other where left and right blocks are exchanged. We use the notation $ed_i^T(l_i, r_i) = ed_i(r_i, l_i)$.

*Example:* In Fig. 3, the encoding dichotomies for $b_1$ are $ed_{b_1} = (S_1S_3, S_2S_4S_5)$ and $ed_{b_1}^T = (S_2S_4S_5, S_1S_3)$; those for

$b_2$ are $ed_{b_2} = (S_1S_5, S_2S_3S_4)$ and $ed^T_{b_2} = (S_2S_3S_4, S_1S_5)$; and those for $b_3$ are $ed_{b_3} = (S_1S_2S_3, S_4S_5)$ and $ed^T_{b_3} = (S_4S_5, S_1S_2S_3)$.

*Definition 4.3:* The partial coverage $pc_{j,i}$ of a seed dichotomy $sd_j = (l_j, S_j)$ by an encoding dichotomy $ed_i = (l_i, r_i)$ is defined as

$$\mathrm{pc}_{j,i} = \begin{cases} l_i \cap l_j, & \text{if } S_j \in r_i \\ \phi, & \text{otherwise.} \end{cases}$$

In other words, $pc_{j,i}$ is the subset of states in $l_j$ that can be distinguished from $S_j$ by $ed_i$. Since all seed dichotomies of a group dichotomy have the same $l_j$, we use the notation $pc_i$ to represent the partial cover of $ed_i$ for a given group dichotomy.

*Example:* The partial coverage of the seed dichotomy $(S_2S_3, S_4)$ by the encoding dichotomy $(S_1S_3, S_2S_4S_5)$ is $(S_3)$.

*Definition 4.4:* A seed dichotomy $sd_j = (l_j, S_j)$ is fully covered by a set of encoding dichotomies $ED = \{ed_1, \cdots, ed_n\}$ if

$$\bigcup_{ed_i \in ED} \mathrm{pc}_{j,i} = l_j. \tag{18}$$

*Example:* Encoding dichotomies $(S_1S_2, S_3S_4)$ and $(S_1S_3, S_2S_4)$ fully cover the seed dichotomy $(S_2S_3, S_4)$.

*Definition 4.5:* A set of encoding dichotomies satisfies a state group constraint if there exists a subset $ED$ of the encoding dichotomies that fully covers all the seed dichotomies of the group dichotomy corresponding to the state group constraint.

*Example:* Encoding dichotomy $(S_1S_3, S_2S_4S_5)$ and $(S_1S_5, S_2S_3S_4)$ fully covers the group dichotomies of the symbolic implicant $SI_2$.

### C. Finding a Minimum Cost Implementation of Symbolic Implicants

Given a state encoding, we want to find the minimum power realization of every symbolic implicant $SI$ in the symbolic cover of the FSM. This problem is mapped to a rectangle covering problem as follows. Let $b_1, \cdots, b_n$ and $ed_1, \cdots, ed_{2n}$ (where $ed_{n+i} = ed^T_i$) be the sets of state bits and their corresponding encoding dichotomies. Let $gd = (z_g, o_g)$ be the group dichotomy of $SI$ where $z_g$ and $o_g$ denote sets of states having 0's and 1's in the state group of $SI$. Furthermore let $SD = \{sd_1, \cdots, sd_m\}$ be the set of $m$ seed dichotomies of $gd$ where $sd_j = (z_g, S_j)$ and $S_j$ is the $j$th states in $o_g$.

A $2n \times m$ covering matrix $M$ is built where every row represents an encoding dichotomy and every column denotes a seed dichotomy. If $pc_{j,i} \neq \phi$, then $M_{ij}$ is one, else it is zero. A rectangle $(R, C)$ is defined as

$$\forall_{i \in R \wedge j \in C} M_{ij} = 1 \tag{19}$$

where $R \subset 1, \cdots, 2n$ and $C \subset 1, \cdots, m$. A *valid rectangle* $(R, C)$ is a rectangle with

$$\cup_{i \in R} \mathrm{pc}_i = z_g. \tag{20}$$

A valid rectangle $(R, C)$ implies that the seed dichotomies in $C$ can be realized by a single binary cube consisting of the state bits in $R$. In other words, the state bits in $R$ can distinguish the symbols represented by the seed dichotomies in $C$ from $z_g$. Fig. 3(d) shows the covering matrix for $SI_1$ and $SI_2$ and illustrates the notion of a valid rectangle. $(\{1\}, \{2, 3\})$ for $SI_1$ is not a valid rectangle since $z_g = \{S_2, S_3\}$ and $pc_1 = \{S_3\} \neq z_g$. However, rectangles $(\{1, 3\}, \{2, 3\})$ and $(\{3\}, \{3\})$ are both valid rectangles. In this example, the group dichotomy $SI_1$ cannot be covered by a single subset of encoding dichotomies and hence cannot be realized by a single binary cube. In fact, $SI_1$ has to be realized by $\bar{i}b_1b_3$ and $\overline{ib}_2$, for $SI_2$ rectangle $(\{1, 2\}, \{1, 2\})$ fully covers all the seed dichotomies. Hence $SI_2$ can be implemented by one single binary implicant $ib_1b_2$.

The minimum power realization problem can then be stated as finding a valid rectangle cover $\{(R_1, C_1), \cdots, (R_k, C_k)\}$ such that the power cost is minimized. The power cost is defined as

$$P_{\mathrm{SI}} = \sum_{(R_i, C_i) \in \{(R_1, C_1), \cdots, (R_k, C_k)\}} P_{\mathrm{BI}_{(R_i, C_i)}} \tag{21}$$

where $\mathrm{BI}_{(R_i, C_i)}$ is the corresponding binary implicant of $(R_i, C_i)$.

A simplified version of the valid rectangle covering problem is used in the kernelization step of multilevel logic optimization and is shown to be NP-hard [13]. To solve the valid rectangle covering problem, we therefore resort to a heuristic greedy approach.

We construct one valid rectangle at a time until all seed dichotomies are covered. In constructing the valid rectangle, we pick one encoding dichotomy at a time until the rectangle is valid.

For every rectangle used, there is some fixed power cost, which is the power consumption at the combinational primary inputs and the clocked transistors. Therefore, one goal is to minimize the number of rectangles in the cover. The wider the rectangle, the higher the chance of having a rectangle cover with smaller cardinality. The procedure is then to look first for the rectangle that is the widest and has the least cost. After a rectangle is chosen, we eliminate the seed dichotomies that are covered by it. The procedure is repeated until all seed dichotomies are covered.

To find the rectangle that is the widest and has the least cost, we use the following procedure. The cost of a rectangle depends on the number and the switching activity of the encoding dichotomies used in the rectangle. A wide rectangle is formed from the encoding dichotomies that cover the largest number of uncovered seed dichotomies. Also, the larger the size of the partial cover $pc_{ji}$ of an encoding dichotomy $ed_i$, the higher the chance of using fewer encoding dichotomies to form a rectangle. Therefore, we assign the following cost for each encoding dichotomy:

$$\mathrm{cost}(ed_i) = \frac{E_{ed_i}}{\mathrm{seed\_coverage}(ed_i)\,\mathrm{zero\_block\_coverage}(ed_i)} \tag{22}$$

where $E_{ed_i}$ is simply the switching activity of state bit $i$, $\mathrm{seed\_coverage}(ed_i)$ is the ratio of the number of seed dichotomies covered by $ed_i$ and the total number of the seed

dichotomies, and zero_block_coverage($ed_i$) is the ratio of the number of states in $z_g$ that are covered by $ed_i$ and the total number of states in $z_g$. In Fig. 3(d), the cost for $ed_{b_1}$ for $SI_1$ is thus

$$\frac{E_{b_1}}{0.666 \times 0.5}.$$

If either seed_coverage($ed_i$) or group_coverage($ed_i$) is zero, then $d_i$ does not distinguish any states from $z_g$ and hence is redundant. Therefore, its cost is set to infinity.

The encoding dichotomy with the least cost is chosen first. If the rectangle is not a valid one, we have to continue the process of selecting more encoding dichotomies. Once an encoding dichotomy is chosen, it sets an upper bound on the width of the final valid rectangle and also reduces the number of uncovered seed dichotomies in $z_g$. The costs of the remaining dichotomies are updated dynamically as the width of the largest possible rectangle $R$ is now equal to the cardinality of the set of possible covered seed dichotomy. Now $R$ is equal to

$$\bigcap_{ed_i \in ED_{\text{select}}} RC(ed_i) \tag{23}$$

where $ED_{\text{select}}$ is the set of selected encoding dichotomies and $RC(ed_i)$ is the set of seed dichotomies $sd_j$ such that $M_{i,j} = 1$.

Also, since some of the states in $z_g$ have already been covered by the selected $ed_i$ in $ED_{\text{select}}$, therefore $z_g$ has to be reduced by removing the states in the set

$$\bigcup_{ed_i \in ED_{\text{select}}} pc_{ed_i} \tag{24}$$

where $pc_{ed_i}$ is the partial cover of $ed_i$.

The new seed_coverage($ed_k$) and zero_block_coverage ($ed_k$) of the unselected encoding dichotomy $ed_k$ are then obtained from the number of seed dichotomies in $R$ that are covered by $ed_k$ and the number of states in the reduced $z_g$ that are covered by the partial cover of $ed_k$.

## V. Multilevel Logic Implementation

For area minimization, the objective of the state assignment is to minimize the number of literals in the multilevel logic implementation. The literal saving cost function has been well studied in [7] and [8]. In these approaches, the present state weights are calculated by grouping the symbolic cube $M = (X, S, S', Y)$ in a state transition table into the following subsets:

$$C^Y_{k,i} = \{m_j \in M \mid S_j = S_k, (y_j)_i = 1\}$$
$$C^{S'}_{k,i} = \{m_j \in M \mid S_j = S_k, S'_j = S_i\}$$

where $|C^Y_{k,i}|$ ($|C^{S'}_{k,i}|$) represents the occurrence frequency of state $S_k$ in the output (next state) functions.

Let $n_E$ be the number of state bits used for encoding. If the encodings of states $S_k$ and $S_l$ have a Hamming distance of $d_{k,l}$, then a common cube $B$ with $|B| = n_E - d_{k,l}$ literals can be extracted from them.

For an output function $y_i$, if we assume an unminimized, one-level, one-hot encoded representation of the FSM, there are $|C^Y_{k,i}|$ and $|C^Y_{l,i}|$ fanins from $S_k$ and $S_l$, respectively. The literal saving of extracting a common cube $B$ from $S_k$ and $S_l$ for this output function is thus equal to

$$(|B| - 1)\mu_{k,l,i}$$

where $\mu_{k,l,i} = |C^Y_{k,i}| + |C^Y_{l,i}|$. Similarly, the literal saving for a next-state function $S_j$ is equal to

$$(|B| - 1)\lambda_j \gamma_{k,l,j}$$

where $\gamma_{k,l,j} = |C^{S'}_{k,j}| + |C^{S'}_{l,j}|$ and $\lambda_j$ is the number of 1's in state $S_j$. The cost of implementing the extracted cube is $|B|$. Therefore, the total literal saving of extracting a common cube $C$ from states $S_k$ and $S_l$ is

$$\Delta_{k,l} = \left\{ \sum_{i=1}^{n_Y} \mu_{k,l,i} + \sum_{j=1}^{n_S} \lambda_j \gamma_{k,l,j} \right\} (|B| - 1) - |B|. \tag{25}$$

This is the cost function used in JEDI [7] except that the last term in (25) is removed. MUSTANG [8] approximates $\lambda_j$ by $n_S/2$, and uses multiplication instead of addition to calculate the weight function.

For low-power applications, we have to minimize $P_{\text{reg}}$, $P_{\text{inputs}}$, and $P_{\text{comb}}$. We look at $P_{\text{inputs}}$. For power-conscious state assignment, state transitions with high probability should be assigned higher weights. However, the occurrence frequency of each state must be considered as well because this frequency determines the number of fanouts from the state bits (which also affects the power consumption). So instead of counting the number of literals saved, we calculate a literal savings factor weighted by the switching activity of the literals.

Consider two states $S_k$ and $S_l$ with a common cube $B$. The two states are encoded as follows:

$$S_k = \underbrace{b_1 b_2 \cdots b_m}_{B} \mid \underbrace{b_{m+1} \cdots b_{n_S}}_{B_k}$$
$$S_l = \underbrace{b_1 b_2 \cdots b_m}_{B} \mid \underbrace{b'_{m+1} \cdots b'_{n_S}}_{B_l}.$$

The common cube $B$ can be extracted from $S_k$ and $S_l$ as

$$\begin{aligned} S_k + S_l &= B(B_k + B_l) \\ &= b_1 b_2 \cdots b_m (b_{m+1} \cdots b_{n_S} + b'_{m+1} \cdots b'_{n_S}). \end{aligned}$$

Let the set $S_B$ denote the set of states whose corresponding bits have the same binary values as those in $B$ and $\bar{S}_B = S - S_B$. The notation $S_{b_i}$ stands for the set of states whose $i$th encoding bit is one.

The power saving at the primary and state input bits by extracting $B$ from the present states $S_k$ and $S_l$ is equal to the number of literals saved weighted by the switching activity of the state bits in $B$. The literal power saving in $P_{\text{input}}$ is equal to

$$\left\{ \sum_{i=1}^{n_Y} \mu_{k,l,i} + \sum_{j=1}^{n_S} (\lambda_j \gamma_{k,l,j}) \right\} \sum_{q=1}^{m} TP\big(S_{b_q} \leftrightarrow \bar{S}_{b_q}\big) - \sum_{q=1}^{m} TP\big(S_{b_q} \leftrightarrow \bar{S}_{b_q}\big) \tag{26}$$

where $TP(S_{b_q} \leftrightarrow \bar{S}_{b_q})$ is simply $E_{b_q}$.

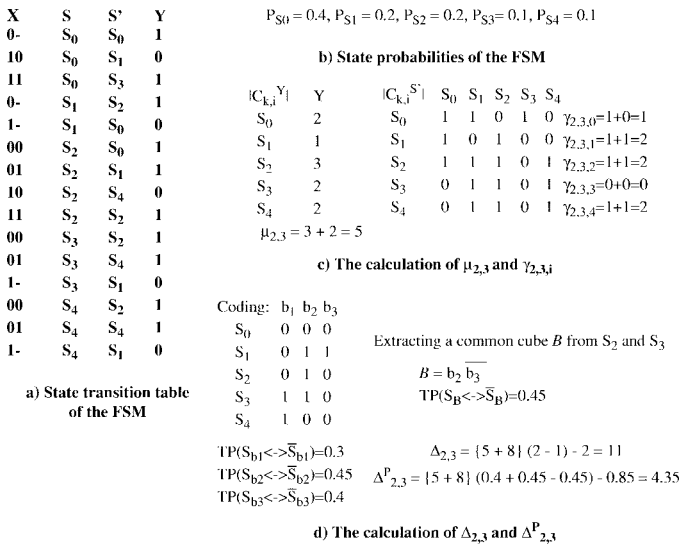



Fig. 4. Example for calculating the literal power saving.

However, $B$ is now fanout to $\sum_{i=1}^{n_Y}\mu_{k,l,i}+\sum_{j=1}^{n_S}(\lambda_j\gamma_{k,l,j})$ number of literals, and the power cost due to the extracted cube $B$ is

$$\left\{\sum_{i=1}^{n_Y}\mu_{k,l,i}+\sum_{j=1}^{n_S}(\lambda_j\gamma_{k,l,j})\right\}TP(S_B\leftrightarrow\bar{S}_B) \tag{27}$$

where $TP(S_B\leftrightarrow\bar{S}_B)$ is calculated by identifying $S_B$ and applying (2) on $S_B$. We have to subtract this power cost from the power saving at the input literals. Therefore, the overall literal power saving when extracting a common cube $B$ from states $S_k$ and $S_l$ is given by

$$\begin{aligned}\Delta^P_{k,l}=&\left\{\sum_{i=1}^{n_Y}\mu_{k,l,i}+\sum_{j=1}^{n_S}(\lambda_j\gamma_{k,l,j})\right\}\\&\cdot\left\{\sum_{q=1}^{m}TP\big(S_{b_q}\leftrightarrow\bar{S}_{b_q}\big)-TP(S_B\leftrightarrow\bar{S}_B)\right\}\\&-\sum_{q=1}^{m}TP\big(S_{b_q}\leftrightarrow\bar{S}_{b_q}\big).\end{aligned} \tag{28}$$

Fig. 4 illustrates an example of how the literal power saving is calculated for a specific encoding when a common cube is extracted from a pair of states. The state transition table and the corresponding state probabilities of the FSM are shown in Fig. 4(a) and (b), respectively. The calculation of $\mu_{k,l,i}$ and $\gamma_{k,l,j}$ is demonstrated in Fig. 4(c). The literal saving and the overall literal power saving when extracting a common cube $B$ from state $S_2$ and $S_3$ for the given encoding are shown in Fig. 4(d).

Unlike area minimization, where the initial literal count is fixed (i.e., it does not depend on the actual encoding) and hence literal saving can be used as a metric for overall area saving, the initial literal power consumption does depend on the encoding, and hence the above literal power saving alone does not reflect the actual literal power cost. We have to calculate the initial power cost $P_{\text{init}}$ and then subtract the literal power saving to get the actual power cost. Therefore

$$P_{\text{inputs}}=P_{\text{init}}-P_{\text{saving}} \tag{29}$$

where

$$P_{\text{init}}=\sum_{S_i\in S}\mu_{S_i}C_{\text{lit}}\sum_{j=1}^{n_E}E_{b_j} \tag{30}$$

$$P_{\text{saving}}=\sum_{S_k\in S}\sum_{S_l\in S}\Delta^P_{k,l}C_{\text{lit}} \tag{31}$$

where $S$ is the set of all state, $\mu_{S_k}$ is the occurrence frequency of $S_k$ which is given by

$$\mu_{S_k}=\sum_{i=1}^{n_Y}\left|C^Y_{k,i}\right|+\sum_{j=1}^{n_S}\lambda_j\left|C^{S'}_{k,j}\right|. \tag{32}$$

$P_{\text{reg}}$ is equal to

$$P_{\text{reg}}=C_{\text{reg}}\sum_{j=1}^{n_E}E_{b_j}. \tag{33}$$

TABLE I
POWER CONSUMPTION FOR TWO-LEVEL LOGIC USING DYNAMIC PLA

| circuits | NOVA | MWHD | % red. | LPSA | % red. |
|---|---|---|---|---|---|
| bbara | 361.49 | 362.88 | -0.38 | 339.36 | 6.12 |
| bbsse | 508.82 | 408.66 | 19.68 | 411.16 | 19.19 |
| bbtas | 216.21 | 196.71 | 9.02 | 166.92 | 22.80 |
| beecount | 281.28 | 205.06 | 27.10 | 222.79 | 20.79 |
| cse | 630.42 | 668.39 | -6.02 | 614.88 | 2.47 |
| dk14 | 440.29 | 541.52 | -22.99 | 437.73 | 0.58 |
| dk16 | 898.30 | 991.83 | -10.41 | 886.16 | 1.35 |
| dk17 | 313.11 | 325.84 | -4.07 | 313.11 | 0.00 |
| dk27 | 180.02 | 201.56 | -11.97 | 177.73 | 1.27 |
| dk512 | 342.76 | 370.34 | -8.05 | 318.93 | 6.95 |
| donfile | 812.24 | 729.89 | 10.14 | 547.89 | 32.55 |
| ex1 | 821.82 | 857.05 | -4.29 | 665.09 | 19.07 |
| ex4 | 378.85 | 336.14 | 11.27 | 278.23 | 26.56 |
| sand | 1427.90 | 1424.46 | 0.24 | 1358.94 | 4.83 |
| s208 | 267.68 | 406.50 | -51.86 | 314.64 | -17.54 |
| s27 | 251.75 | 237.46 | 5.68 | 223.51 | 11.22 |
| sse | 519.25 | 413.80 | 20.31 | 411.16 | 20.82 |
| tma | 552.24 | 500.71 | 9.33 | 517.09 | 6.36 |
| average %reduction | | | -0.40 | | 10.30 |

## VI. EXPERIMENTAL RESULTS

In this section, we present experimental results of the low-power state assignment algorithm for two-level and multilevel implementations. Experiments were done using the MCNC-91 FSM benchmark sets. The circuits are assumed to operate at 5-V power supply and 20-MHz clock frequency. The experiments are done on a SPARC 20 with 64 Mbytes of memory.

The first experiment is to compare low-power state assignment (LPSA) for two-level implementation using dynamic NOR–NOR PLA with NOVA [3], which is a state assignment program targeting minimum area and with the MWHD encoding. The option we used for NOVA is an input-constrained algorithm plus simulated annealing. The encoded

TABLE II
NUMBER OF PRODUCT TERMS

| circuits | NOVA | MWHD | % increase | LPSA | % increase |
|---|---|---|---|---|---|
| bbara | 24 | 26 | 8.33 | 24 | 0.00 |
| bbsse | 31 | 29 | -6.45 | 29 | -6.45 |
| bbtas | 13 | 11 | -15.38 | 10 | -23.08 |
| beecount | 13 | 12 | -7.69 | 13 | 0.00 |
| cse | 46 | 50 | 8.7 | 45 | -2.17 |
| dk14 | 27 | 37 | 37.04 | 28 | 3.70 |
| dk16 | 61 | 76 | 24.59 | 65 | 6.56 |
| dk17 | 19 | 24 | 26.32 | 19 | 0.00 |
| dk27 | 7 | 11 | 57.14 | 9 | 28.57 |
| dk512 | 18 | 24 | 33.33 | 19 | 5.56 |
| donfile | 56 | 54 | -3.57 | 37 | -33.93 |
| ex1 | 51 | 55 | 7.84 | 42 | -17.65 |
| ex4 | 19 | 20 | 5.26 | 16 | -15.79 |
| sand | 105 | 111 | 5.71 | 105 | 0.00 |
| s208 | 19 | 33 | 73.68 | 25 | 31.58 |
| s27 | 13 | 16 | 23.08 | 14 | 7.69 |
| sse | 31 | 29 | -6.45 | 29 | -6.45 |
| tma | 31 | 34 | 9.68 | 36 | 16.13 |
| average %increase | | | 15.62 | | -0.32 |

TABLE III
EXECUTION TIME OF LPSA

| circuits | execution time (in sec) | |
|---|---|---|
| | 2-level | multi-level |
| bbara | 100 | 3.5 |
| bbsse | 190 | 13.4 |
| bbtas | 17 | 0.67 |
| beecount | 30 | 1.3 |
| cse | 322 | 12.4 |
| dk14 | 39 | 0.83 |
| dk16 | 1946 | 40.8 |
| dk17 | 53 | 1.2 |
| dk27 | 27 | 0.94 |
| dk512 | 260 | 7.00 |
| donfile | 520 | 34.85 |
| ex1 | 631 | 23.90 |
| ex4 | 93 | 6.50 |
| sand | 1510 | 48.60 |
| s208 | 724 | 40.65 |
| s27 | 21 | 0.75 |
| sse | 186 | 13.40 |
| tma | 921 | 30.4 |

TABLE IV
POWER CONSUMPTION FOR MULTILEVEL LOGIC

| circuits | JEDI | MWHD | % red. | LPSA | % red. |
|---|---|---|---|---|---|
| bbara | 219.0 | 214.72 | 1.95 | 179.6 | 17.97 |
| bbsse | 659.8 | 590.0 | 10.58 | 538.6 | 18.37 |
| bbtas | 140.9 | 136.1 | 3.41 | 106.9 | 24.13 |
| beecount | 344.5 | 340.2 | 1.25 | 307.3 | 10.8 |
| cse | 623.9 | 680.3 | -9.04 | 548.1 | 12.15 |
| dk14 | 941.4 | 1032.0 | -9.62 | 895.1 | 4.92 |
| dk16 | 2101.9 | 2133.9 | -1.52 | 1689.6 | 19.52 |
| dk17 | 622.5 | 577.1 | 7.295 | 441.8 | 29.03 |
| dk27 | 296.8 | 270.3 | 8.93 | 232.1 | 21.80 |
| dk512 | 644.0 | 603.7 | 6.26 | 407.8 | 36.68 |
| donfile | 675.3 | 625.8 | 7.33 | 633.3 | 6.22 |
| ex1 | 1217.7 | 1139.3 | 6.44 | 1004.8 | 17.48 |
| ex4 | 562.2 | 549.9 | 2.19 | 374.4 | 33.40 |
| s208 | 409.5 | 417.6 | -1.98 | 346.9 | 15.29 |
| s27 | 128.2 | 149.1 | -16.30 | 178.6 | -39.31 |
| sand | 2049.3 | 2009.6 | 1.94 | 1833.8 | 10.52 |
| sse | 659.8 | 590.0 | 10.58 | 538.6 | 18.37 |
| tma | 1402.8 | 1195.3 | 14.79 | 920.2 | 34.40 |
| average %reduction | | | 2.47 | | 16.21 |

TABLE V
NUMBER OF LITERAL COUNTS

| circuits | JEDI | MWHD | % increase | LPSA | % increase |
|---|---|---|---|---|---|
| bbara | 69 | 80 | 15.94 | 62 | -10.14 |
| bbsse | 122 | 121 | -0.82 | 130 | 6.56 |
| bbtas | 24 | 24 | 0 | 26 | 8.33 |
| beecount | 54 | 43 | -20.37 | 42 | -22.2 |
| cse | 213 | 223 | 4.69 | 210 | -1.41 |
| dk14 | 102 | 112 | 9.80 | 101 | -0.98 |
| dk16 | 294 | 293 | -0.34 | 311 | 5.78 |
| dk17 | 64 | 65 | 1.56 | 57 | -10.94 |
| dk27 | 26 | 25 | -3.85 | 25 | -3.85 |
| dk512 | 65 | 85 | 30.77 | 80 | 23.08 |
| donfile | 106 | 185 | 74.53 | 130 | 22.64 |
| ex1 | 239 | 297 | 24.27 | 265 | 10.88 |
| ex4 | 73 | 77 | 5.48 | 77 | 5.48 |
| s208 | 130 | 106 | -18.46 | 118 | -9.23 |
| s27 | 20 | 30 | 50.00 | 26 | 30.00 |
| sand | 546 | 587 | 7.51 | 634 | 16.12 |
| sse | 122 | 121 | -0.82 | 130 | 6.56 |
| tma | 178 | 171 | -3.93 | 184 | 3.37 |
| average %increase | | | 9.8 | | 4.45 |

machines were synthesized using *espresso-exact*. The power consumption was measured using an FSM power estimator [14], which uses the state probabilities that are obtained by solving the Chapman–Kolmogorov equations of the FSM, to compute the exact switching activities of the internal nodes. Under a zero-delay model, this power estimator gives an exact power estimate. Since we are using dynamic PLA implementation and there will not be any hazards or glitches, the power estimate obtained is exact. Table I summarizes the results. Columns 3 and 5 give the percent power reduction of MWHD and LPSA over NOVA, respectively.

It can be seen that in most of the benchmarks, the low-power state assignment produces better results than NOVA and MWHD encodings. An average 10.3% reduction in power is obtained compared to NOVA. It is worthwhile to point out that the minimum weighted Hamming code does worse than NOVA in terms of power consumption. The power consumption increases by an average of 0.4%. We also counted the number of product terms in the final implementation for each encoding. Table II summarizes the number of product terms in the final PLA implementation for NOVA, MWHD, and LPSA. It is seen that the PLA's generated using NOVA or LPSA encoding have a similar number of product terms (an average difference of less than 1%), while that using MWHD has an average of 15.6% more product terms. This explains why MWHD does not produce a good low-power solution. The execution time of LPSA is summarized in Table III. The long execution time

TABLE VI
POWER AND LITERAL REDUCTION COMPARED WITH WEIGHT-COMBINED COST FUNCTION

| circuits | Power | | | Literals | | |
|---|---|---|---|---|---|---|
| | best result of the weight-combined cost function | best result of LPSA | % reduction | best result of the weight-combined cost function | best result of LPSA | % reduction |
| bbara | 214.7 ($\alpha$=0,$\beta$=1) | 173.3 (R=7) | 19.26 | 80 | 62 | 22.5 |
| bbsse | 543.9 ($\alpha$=1,$\beta$=0) | 538.6 (R=7) | 0.97 | 130 | 130 | 0 |
| bbtas | 104.6 ($\alpha$=1,$\beta$=0) | 80.5 (R=20) | 23.04 | 24 | 22 | 8.33 |
| beecount | 297.7 ($\alpha$=99,$\beta$=1) | 259.1 (R=20) | 12.97 | 47 | 43 | 8.51 |
| cse | 560.4 ($\alpha$=99,$\beta$=1) | 537.5(R=20) | 4.09 | 200 | 200 | 0 |
| dk14 | 814.2 ($\alpha$=99,$\beta$=1) | 840.6(R=20) | -3.24 | 99 | 101 | -2.0 |
| dk16 | 1645.0 ($\alpha$=99,$\beta$=1) | 1689.60(R=7) | -2.71 | 293 | 297 | -1.37 |
| dk17 | 490.1 ($\alpha$=1,$\beta$=0) | 380.1(R=7) | 22.44 | 65 | 49 | 24.62 |
| dk27 | 211.9 ($\alpha$=1,$\beta$=99) | 232.1(R=7) | -9.53 | 25 | 25 | 0 |
| dk512 | 516.8 ($\alpha$=99,$\beta$=1) | 407.8(R=7) | 21.09 | 83 | 64 | 22.90 |
| donfile | 625.8 ($\alpha$=0,$\beta$=1) | 633.3(R=7) | -1.2 | 118 | 130 | -10.27 |
| ex1 | 1011.7 ($\alpha$=99,$\beta$=1) | 863.1(R=0) | 14.69 | 297 | 241 | 18.86 |
| ex4 | 301.9 ($\alpha$=1,$\beta$=0) | 373.0(R=20) | -23.55 | 77 | 69 | 10.39 |
| s208 | 341.6 ($\alpha$=1,$\beta$=99) | 277.3(R=50) | 18.82 | 112 | 105 | 6.25 |
| s27 | 131.2 ($\alpha$=1,$\beta$=99) | 121.5(R=50) | 7.39 | 17 | 17 | 0 |
| sand | 1819.1($\alpha$=1,$\beta$=0) | 1634.3(R=50) | 10.16 | 587 | 564 | 3.92 |
| sse | 543.9 ($\alpha$=1,$\beta$=0) | 538.6(R=7) | 0.97 | 130 | 130 | 0 |
| tma | 696.5 ($\alpha$=1,$\beta$=0) | 726.6(R=50) | -4.32 | 171 | 149 | 12.87 |
| | average % reduction | | 6.20 | average %reduction | | 7.0 |

is due to the simulated annealing approach and the overhead in calculating the cost of each encoding dichotomy.

The second experiment compares LPSA for multilevel implementation with JEDI [7], which is a state assignment program targeting minimum area, and with the MWHD encoding. The default output dominant algorithm option is used for both cases. The encoded machines were synthesized and mapped using the SIS package and a 0.8-$\mu$m industrial gate library. For multilevel logic, since power consumption due to glitches can be substantial, we used a real-delay gate-level power simulation engine to estimate the power consumption. Glitch power as well as power consumption due to steady-state logic switching are measured. Fifty thousand random vectors are used for the power simulation to obtain an accurate and convergent power estimate. Moreover, since we are interested in comparisons between different design alternatives, the relative accuracy of the power estimate is more important than the absolute accuracy. In this respect, it has been shown that the relative accuracy is very high for these real-delay Monte Carlo-based gate-level power simulators. Table IV summarizes the results. Columns 3 and 5 give the percent power reduction of MWHD and LPSA over JEDI, respectively.

Table IV shows that in general, the low-power state assignment produces better results than JEDI and the MWHD encoding. An average of 16.2% reduction in power consumption is obtained compared to JEDI. The average power reduction of MWHD encoding compared to JEDI is about 2.5%. Table V summarizes the number of literal counts for JEDI, MWHD, and LPSA. Although the average literal counts for the low-power state assignment algorithms are larger than that of JEDI (4.4%, respectively), power consumption is lower since JEDI does not consider the switching activity. The minimum weighted Hamming distance code, however, has on average 9.8% more literals than that obtained from JEDI. This, and the fact that the capacitive loading of the state bits are ignored, explain why the MWHD encoding does not have as much power saving as LPSA. The execution time for multilevel LPSA is summarized in Table III.

We also compared LPSA with the approach suggested in [11]. In that approach, a weight-combined cost function

$$\alpha\,\mathrm{Cost}_{\mathrm{lit}} + \beta\mathrm{Cost}_{\mathrm{switching}} \tag{34}$$

is used, where $\mathrm{Cost}_{\mathrm{lit}}$ and $\mathrm{Cost}_{\mathrm{switching}}$ correspond to the number of literals in the combination logic part and switching activities at the state registers, respectively. Different $\alpha$ and $\beta$ combinations can be used to generate different solutions. In [11], four different $(\alpha, \beta)$ combinations—namely, (0,1), (1,0), (99,1), (1,99)—to generate four sets of solution are used. In LPSA, we can also provide flexibility in tuning the weight of power consumption at the registers and the weighted switching activity of the literals by changing $R$, the ratio of $C_{\mathrm{reg}}$ to $C_{\mathrm{lit}}$. We incorporated the weight-combined cost model as suggested in [11] in our simulated annealing framework and compared the optimal results obtained by this cost model using different $(\alpha, \beta)$ combinations with our results using different $R$ ratios. Similar to the experiments done in [11], we used four different $R$ ratios (0, 7, 20, 50). Table VI summarizes the comparison. Results show that in general, LPSA does better than [11]. In 12 out of the 18 benchmarks, LPSA obtains a lower power encoding. On average, the power reduction is about 6.2%, whereas the number of literals is reduced by 7%.

## VII. CONCLUDING REMARKS

We presented a power cost model for the state assignment problem targeting both two- and multilevel logic implementa-

tion. We then formulated the problem of calculating the power cost for the symbolic implicant for two-level as a rectangle covering problem and proposed a greedy algorithm to solve it. For multilevel logic implementation, we proposed a power cost function that captures the weighted switching activity at the inputs of the circuit.

*Espresso* [15] is used to generate the final implementation for the two-level logic. This two-level logic minimization algorithm targets for minimum area and does not exploit the switching activity information. Future work will therefore focus on developing a two-level logic minimization algorithm for low power.

## REFERENCES


[1] G. De Micheli, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Optimal state assignment of finite state machines," *IEEE Trans. Computer-Aided Design*, vol. 4, pp. 269–285, July 1985.

[2] G. De Micheli, "Symbolic design of combinational and sequential logic circuits implemented by two-level macros," *IEEE Trans. Computer-Aided Design*, vol. 5, pp. 597–616, Sept. 1986.

[3] T. Villa and A. Sangiovanni-Vincentelli, "NOVA: State assignment of finite state machines for optimal two-level logic implementations," *IEEE Trans. Computer-Aided Design*, vol. 9, pp. 905–924, Sept. 1990.

[4] S. Yang and M. Ciesielski, "On the relationship between input encoding and logic minimization," in *Proc. 23rd Hawaii Int. Conf. System Sciences*, Jan. 1990, vol. I, pp. 377–386.

[5] S. Devadas and A. R. Newton, "Exact algorithms for output encoding, state assignment and four-level Boolean minimization," in *Proc. 23rd Hawaii Int. Conf. System Sciences*, Jan. 1990, vol. I, pp. 387–396.

[6] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, May 1983.

[7] B. Lin and A. R. Newton, "Synthesis of multiple-level logic from symbolic high-level description languages," in *Proc. IFIP Int. Conf. Very Large Scale Integration*, Aug. 1989, pp. 187–196.

[8] S. Devadas, H-K. T. Ma, A. R. Newton, and A. Sangiovanni-Vincentelli, "MUSTANG: State assignment of finite state machines targeting multilevel logic implementations," in *IEEE Trans. Computer-Aided Design*, vol. 7, pp. 1290–1300, Dec. 1988.

[9] X. Du, G. Hactel, B. Lin, and A. R. Newton, "MUSE: A multilevel symbolic encoding algorithm for state assignment," in *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 28–38, Jan. 1991.

[10] K. Roy and S. Prasad, "Syclop: Synthesis of CMOS logic for low power application," in *Proc. Int. Conf. Computer Design*, Oct. 1992, pp. 464–467.

[11] E. Olson and S. M. Kang, "Low-power state assignment for finite state machines search," in *Proc. Int. Workshop Low Power Design*, Apr. 1994, pp. 63–68.

[12] R. Rudell and A. Sangiovanni-Vincentelli, "Multiple-valued minimization for PLA optimization," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 727–751, Sept. 1987.

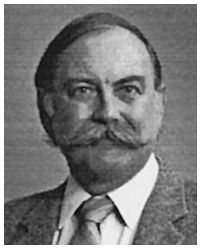
[13] R. Rudell, "Logic synthesis for VLSI design," Ph.D. dissertation, University of California, Berkeley, 1989.

[14] C.-Y. Tsui, M. Pedram, and A. M. Despain, "Exact and approximate methods for calculating signal and transition probabilities in FSMs," in *Proc. 31st Design Automation Conf.*, June 1994, pp. 18–23.

[15] R. K. Brayton, G. D. Hachtel, C. McMullen, and A. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*. Boston, MA: Kluwer Academic, 1984.


**Chi-Ying Tsui** (S'94–M'95), for a biography, see p. 1107 of the November 1998 issue of this TRANSACTIONS.

**Massoud Pedram** (S'88–M'90), for a photograph and biography, see p. 83 of the February 1998 issue of this TRANSACTIONS.



**Alvin M. Despain** (S'58–M'65–SM'95–F'97) received the B.S., M.S., and Ph.D. degrees in electrical engineering from The University of Utah, Salt Lake City, in 1960, 1962, and 1966, respectively.

He is the Powell Professor of Computer Engineering at the University of Southern California (USC), Los Angeles, and a Professor in the Computer Science and Electrical Engineering Systems Departments at USC. He was an Assistant Research Professor at The University of Utah, an Associate Professor at Utah State University, a Visiting Associate Professor at Stanford University, and a Professor at the University of California at Berkeley. He has been at USC since 1989. He is a pioneer in the study of high-performance computer systems. His research group builds experimental software and hardware systems including compilers, simulators, design tools, custom VLSI processors, and multiprocessor systems. The goal is to determine principles for the design of high-performance computer systems. His research interests include computer architecture, multiprocessor systems, logic programming, quantum computation, and design automation.