Low Power State Assignment Targeting Two- and Multi-level Logic Implementations

Chi-Ying Tsui, Massoud Pedram
and Alvin M. Despain

CENG Technical Report 94-07

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, California 90089-2562
(213)740-4458

April 1994

Low Power State Assignment Targeting Two- and Multi-level Logic Implementations

Chi-Ying Tsui, Massoud Pedram, Alvin M. Despain

Department of Electrical Engineering - Systems
University of Southern California Los Angeles, CA 90089

April 12, 1994

Abstract

To minimize power consumption in CMOS circuits, switching activities (weighted by load capacitances) have to be reduced. In this report, we address the problem of minimizing weighted switching activities during the state encoding of a finite state machine. We develop a new power cost model for the encoding problem and then present encoding techniques to minimize the power cost targeting both two- and multi-level logic implementations. These techniques are compared with those which minimize area or the switching activity at the present state bits. Experimental results show significant improvements.

1 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 ranging from 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 which 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 which rely on approximate pre-logic synthesis cost functions in order to avoid the expensive logic minimization step. DeMicheli et al. [8] proposed an innovative paradigm in which one-hot codes are assigned to states and a minimum symbolic (multi-valued) cover of the machine is then generated by output-disjoint minimization. This symbolic cover defines a set of face embedding constraints which require that 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 is then relaxed to the problem of finding the minimum number of encoding bits such that all the constraints are satisfied. DeMicheli [7] used a heuristic row encoding technique to solve this problem. Villa et al. [15] employed the notion of face-posets to tackle this problem. Yang et al. [16] 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. [3] proposed an exact method based on the concept of generalized prime implicants.

One variation of the state assignment problem is the *bit-constrained state assignment* problem where an encoding is to be found that minimizes some objective function (e.g. area, power, or delay), subject to the constraint that the number of encoding bits is no larger than a user given number. This problem is again NP-hard and heuristic methods are used to obtain a solution. A common approach is to use simulated annealing [5].

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 multi-level logic realizations; the temperature is decreased according to the simple rule $T_{new} = \alpha T$ 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 was achieved. Among the above, the most critical and computationally demanding procedure is the cost calculation procedure. The quality of the solution depends on how good the cost function captures the final objective function.

For two-level logic implementation, NOVA [15] used the number of unsatisfied constraints weighted by their occurrence frequency in the symbolic cover as the cost function. For multi-level logic implementation, a cost function that reflects higher cube sharing is used. In particular, JEDI [6], MUSTANG[2] and MUSE [4] assigned weights to pairs of states which 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:

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

where T_{cycle} is the cycle time, C_{load} and $E_{switching}$ 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.

Recently Roy et al. [12] addressed the problem of reducing switching activities during state assignment. They assumed that the power consumption is proportional to the switching activities of the state bit lines of the machine and hence use the following cost function:

$$\sum_{s_i, s_j \in S} tp_{ij} H(s_i, s_j)$$

where 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. [1] 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 multi-level logic realization of the next state and output parts of the FSM.

[1] In [12], the authors incorrectly used the conditional state transition probability p_{ij} instead of $tp2_{ij}$ where p_{ij} is the conditional probability that the next state is s_j given the present state is s_i.

In an attempt to account for power consumption in the combinational part of the FSM, Olson et al. [10] used a linear combination of switching activity and the number of literals as the cost function. The drawback of this approach is that it considers the loading and switching activities separately and hence does not directly address the problem of minimizing the weighted switching activities. Also since the number of literals and switching activities are two quantities of very different nature, a linear combination of the two may not work very well.

In this report, we target the bit-constrained state assignment problem for low power. Simulated annealing is used as the global search strategy. We first present a power cost model for state assignment which considers both the capacitive loading and the switching activity at the same time. We then propose accurate power cost functions for both two- and multi-level logic implementation. For two-level logic using PLA implementation, the dichotomy-based approach is extended to calculate the proposed power cost function. The impact of the PLA type on the power cost function is also described. For multi-level logic implementation, a cost function that takes into account the weighted switching activities at the inputs of the FSM is developed. As power consumption in the combinational part of the FSM is very much dependent on the particular multi-level implementation used, we use the number of literals as a first-order approximation of this component of the power consumption. We then modify the simulated annealing procedure to handle the multiple cost function.

The remainder of the report is organized as follows. Section 2 gives our terminology. The power cost model is described in Section 3. The low power state assignment algorithms for two-level logic and multi-level logic using this power cost model are presented in Sections 4 and 5. Section 6 gives our experimental results. We conclude in Section 7.

2 Terminology

A finite state machine (FSM) is characterized by a 5-tuple (X, Y, S, λ, η) where

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

The encoding length is denoted by $n_E (\geq \lceil \log_2 n_S \rceil)$. The FSM is represented by a *state transition table* $M = \{m_i | m_i = (x_i, s_i, s'_i, y_i), i = 1, \ldots, n_M\}$, where $s'_i \in S$ is the next state and y_i is the corresponding output. Each entry $m_i \in M$ is a symbolic implicant (or a multi-valued cube) of the FSM.

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 corresponding Chapman-kolomgorov equations [11].

The *global state transition probability* 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 given by

$$tp_{s_i,s_j} = P_{s_i} p_{ij}$$

where p_{ij} denotes the conditional state transition probability. The notion of global state transition probability can be generalized to transitions between two sets of states, $S_i \subset S$ and $S_j \subset S$ as follows

$$TP(S_i \leftrightarrows S_j) = \sum_{s_i \in S_i} \sum_{s_j \in S_j} (tp_{s_i,s_j} + tp_{s_j,s_i}). \tag{2}$$

The switching activities of the state bit lines depend on the state encoding and the state transition probabilities. The switching activity E_{b_i} of a state bit line b_i is given by

$$E_{b_i} = TP(States(b_i = 1) \leftrightarrows States(b_i = 0)) \tag{3}$$

where $States(b_i = x), x = 0, 1$ denotes the subset of states whose i^{th} bit assumes a value of x.

3 A Power Consumption Model for FSMs

Figure 1 shows a typical implementation of a finite state machine which consists of a combinational circuit and a set of state registers. The sources of power consumption in this implementation are highlited in the figure and explained below.

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

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

where C_{reg} is the effective capacitance of the state register and E_{b_i} is the switching activity of state bit line b_i which is calculated from equation (3).

P_{inputs} is the power consumption required to drive the combinational inputs and the state bit inputs of the combinational part of the machines. It depends on the switching activities of the state bit

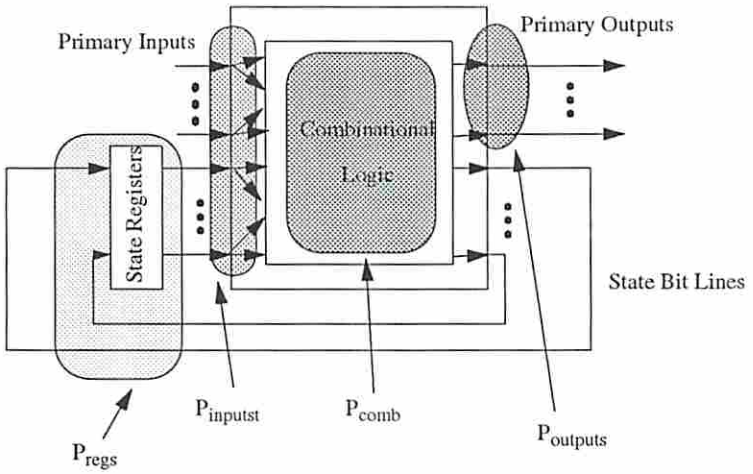

Figure 1: Power model for finite state machines.

lines and the number of combinational input and state bit literals in the logic implementation and is given by

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

where n_i and n_j are the number of literals that input line b_i and j are driving, C_{lit} is the effective capacitance due to each literal [2], and PI is the set of combinational inputs.

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

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

where C_n is the effective capacitance that node n is driving.

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

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

where C_o is the effective capacitance that output o is driving.

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

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

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_{out} is constant and independent

[2] To simplify exposition, we assume each literal has the same capacitive loading.

of the state encoding and can be dropped when comparing the power costs for different state encodings. We therefore minimize $P_{reg} + P_{inputs} + P_{comb}$.

State encoding schemes which minimize the Hamming distance between state pairs with high transition probabilities tend to reduce minimize E_{b_i} and hence P_{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_{inputs} and P_{comb} which will in turn offset the reduction in P_{reg}. As a result, these methods do not in general produce power_optimal assignments.

On the other hand, state encoding schemes which minimize area tend to reduce the fanouts of state bit lines and the number of nodes in the combinational part. They do not consider the switching activities, and hence do not produce power_optimal assignments either.

4 Two Level logic Implementation

4.1 The Cost Function

For a two-level logic circuit, 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_{reg}. P_{inputs} and P_{comb} can be lumped into a single term P_{AND} and

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

where BI is a binary implicant of the logic cover. Let the binray representation of BI consists of combinational inputs $x = x_1 \ldots x_n$ and state bit input $b_1 \ldots b_m$, P_{BI} is given by

$$P_{BI} = P_{inputs_{BI}} + P_{comb_{BI}} \tag{10}$$

where

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

and

$$P_{comb_{BI}} = n_{OR} C_{OR} E_{BI} \tag{12}$$

where n_{OR} is number of OR gates driven by the BI.

Therefore the total power cost function for a two-level logic circuit is equal to

$$P_{reg} + \sum_{BI \in product\_terms} P_{BI}. \tag{13}$$

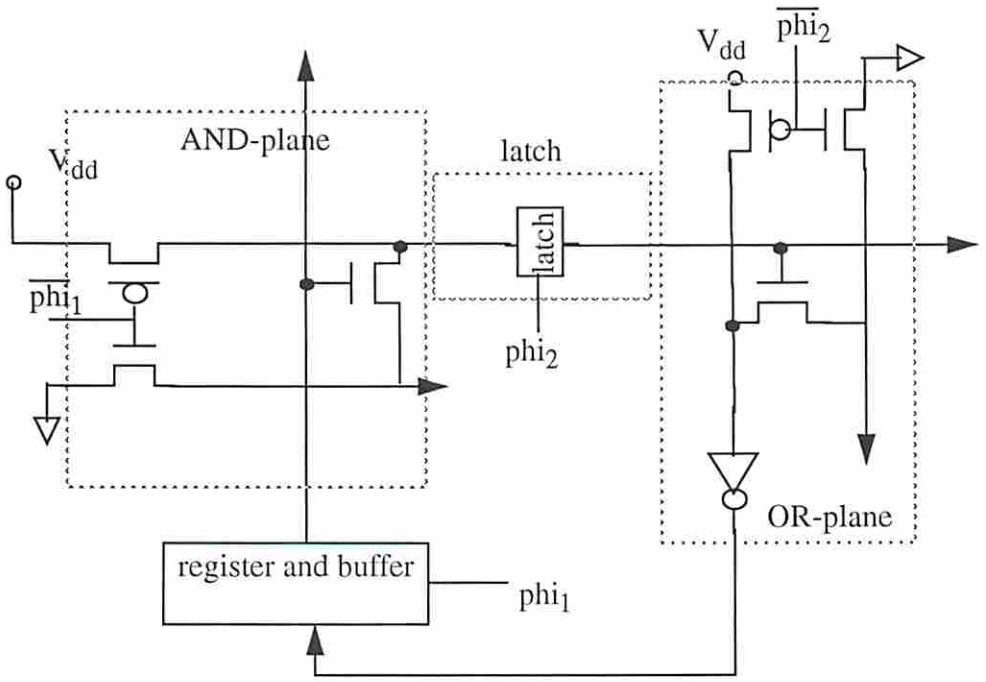

Figure 2: A typical Dynamic PLA circuit.

The type of PLA used for the implementation has a direct impact on the power cost calculation, specifically, on the values of E_{b_j} and E_{BI}. Figure 2 shows a typical dynamic PLA circuit using NOR-NOR structure which is commonly used for high performance controller in microprocessors. For dynamic PLA, the output of the AND plane is driven by a dynamic NOR gate and is precharged to 1 during the precharge period and is evaluated based on the input during the evaluation period. It switches when the output is evaluated to 0. Hence E_{BI} is equal to $prob(BI = 0)$. Using the same reasoning, the switching activity at the output of the OR plane is equal to $prob(out = 1)$. The switching activity of the present state bit line is the same as that of the corresponding next state bit line and hence E_{b_j} is equal to $prob(b_j = 1)$. In addition, we have to include the power consumption at the precharge and evaluation clocked transistors of each NOR gate which is a constant term and is given by

$$P_{clock} = (C_{precharge} + C_{evaluate})E_{clock} \tag{14}$$

where E_{clock} is 1. [3] Therefore the P_{BI} for two-level logic circuits implemented using dynamic PLA is equal to

$$P_{inputs_{BI}} + P_{comb_{BI}} + P_{clock}. \tag{15}$$

[3] A similar power cost function can be derived for other types of PLA such as pseudo-NMOS PLA.

4.2 The Cost Calculation Procedure

Given a symbolic cover, we want to quickly calculate the power cost of a given encoding. P_{reg} is easy to compute since C_{reg} is fixed and E_{b_i} can be computed from the encoding using equation (3). However if we want to compute P_{AND} exactly, we have to know the exact implementation which requires logic minimization. Instead we use the power cost of the symbolic implicants to approximately capture P_{AND}.

Let $SI = (x, s, s', y)$ be a symbolic implicant where $CI = x_1 \ldots x_n$ is the combinational input implicant. If SI is realized by a single binary implicant BI, then the power cost of realization of this symbolic cube is

$$P_{SI} = P_{BI}. \tag{16}$$

If the state group of SI is not satisfied by the encoding, it may require more than one binary implicant to realize it. Let $BI_1 \ldots BI_q$ be the set of binary implicants that realize SI, then the power cost of this realization is

$$P_{SI} = \sum_{i=1}^{q} P_{BI_q}. \tag{17}$$

P_{AND} is then given by

$$P_{AND} = \sum_{SI \in symbolic\_cover} P_{SI}. \tag{18}$$

To find the minimum power cost realization of a symbolic implicant, we use the concept of dichotomy which is also used in state assignment algorithms targeting minimum area [16].

4.2.1 Definitions and Notation

We use the example shown in Figure 3 to illustrate the definitions and notation.

Definition 4.1 *A symbolic implicant is a 4-tuple $< x, s, s', y >$ 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 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 0 in the state group are in the left block and those having a 1 are in the right block. A seed dichotomy is a dichotomy*

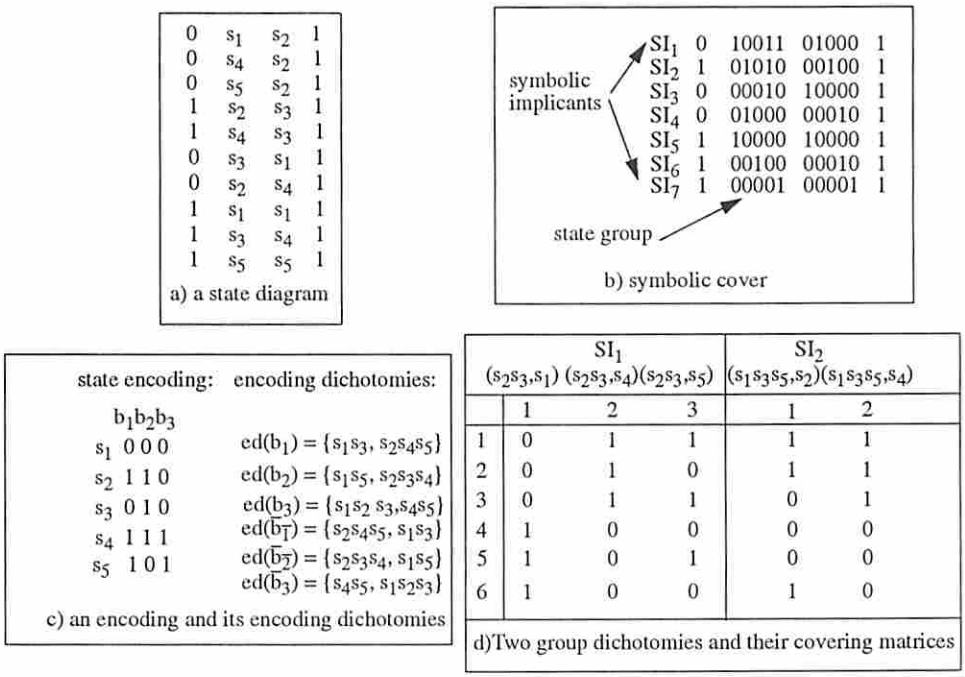

Figure 3: Example illustrating the definitions.

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 Figure 3, there are 7 group dichotomies, one for each symbolic implicant. For 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 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; 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 Figure 3, the encoding dichotomies for b_1 and $\bar{b}_1$ are $(s_1s_3, s_2s_4s_5)$,$(s_2s_4s_5, s_1s_3)$, for b_2 and $\bar{b}_2$ are $(s_1s_5, s_2s_3s_4)$ and $(s_2s_3s_4, s_1s_5)$, and those for b_3 and $\bar{b}_3$ are $(s_1s_2s_3, s_4s_5)$ and $(s_4s_5, s_1s_2s_3)$

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:*

$$pc_{j,i} = \begin{cases} l_i \cap l_j & \text{if } s_j \in r_i \\ \phi & \text{otherwise} \end{cases}$$

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, hence 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 $ED = \{ed_1, \ldots, ed_n\}$ if*

$$\bigcup_{ed_i \in ED} pc_{j,i} = l_j \tag{19}$$

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 which 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.

4.2.2 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 $b_1, \ldots, b_n$ and $ed_1, \ldots, ed_{2n}$ (where $ed_{n+i} = ed_i^T$) 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 denotes sets of states having 0's and 1's in the state group of SI. Furthermore let $SD = \{sd_1, \ldots, sd_m\}$ 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 1, else it is 0. A rectangle (R,C) is defined as

$$\forall_{i \in R \wedge j \in C} M_{ij} = 1 \tag{20}$$

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

$$\cup_{i \in R} pc_i = z_g. \tag{21}$$

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. Figure 3d 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 $pc_1 = \{s_2\} \neq z_g$. 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 $\bar{i}b_2$. For SI_2 rectangle $(\{1,2\},\{1,2\})$ fully covers all the seed dichotomies and hence SI_2 can be implemented by one single binary implicant $\bar{i}b_1b_2$.

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

$$P_{SI} = \sum_{(R_i,C_i)\in\{(R_1,C_1),\ldots,(R_k,C_k)\}} P(BI_{(R_i,C_i)}) \tag{22}$$

where $BI_{(R_i,C_i)}$ is the corresponding binary implicant of (R_i,C_i).

A reduced form of rectangle covering problem is used in the kernelization step of multi-level logic optimization and is shown to be NP-hard [13]. To solve the valid rectangle covering problem, we therefore resort to a heuristic method.

However we first show a lemma that helps in pruning the search space.

Lemma 4.1 *Let SD_1 be a set of seed dichotomies covered by the rectangle (R_1,C_1). Given a set of rectangles $\{(R_2,C_2),\ldots,(R_k,C_k)\}$ that cover sets of seed dichotomies $SD_2,\ldots,SD_k$. If $SD_1 = \cup_{i\in 2,\ldots,k} SD_i$, then $P_{BI_{(R_1,C_1)}} \leq \sum_{i\in 2,\ldots,k} P_{BI_{(R_i,C_i)}}$.*

From Lemma 1, if a group dichotomy is implemented by a single cube, then the power cost is minimum. Therefore for each group dichotomy, we first check whether it can be covered by a single cube. If that is not possible, we then use the following greedy approach to obtain a minimal power implementation.

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. 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. Also the cost of a rectangle depends on the number and the switching activities of the encoding dichotomies used in the rectangle. Thus 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

$$cost(ed_i) = \frac{E_{ed_i}}{seed\_coverage(ed_i) zero\_block\_coverage(ed_i)} \tag{23}$$

where E_{d_i} is simply the switching activity of state bit i, 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 which are covered by ed_i and the total number of states in z_g. In Figure 3d, 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 upperbound 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 to reflect these changes.

4.2.3 A Heuristic Cost Function

The procedure to find a minimum power realization of the symbolic cover is invoked at every step in the inner loop of the simulated annealing. This procedure involves solving constrained rectangle covering problem and is expensive. One way to speed up the simulated annealing procedure is that instead of solving the rectangle covering problem to calculate P_{SI}, we use the sum of the static costs for each encoding dichotomy, i.e.,

$$P_{SI} = \sum_{i=1}^{2n} cost(ed_i) \tag{24}$$

where $cost(ed_i)$ is calculated using equation (23). It is shown that this approximate cost function for P_{SI} in equation (18) produces very good results while speeding up the state assignment by an order of magnitude.

5 Multi-Level logic Implementation

5.1 The Cost Function

For area minimization, the objective of the state assignment is to minimize the number of literals in the multi-level logic implementation. The literal saving cost function has been well studied

in [6] [2]. In these approaches, the present state weights are calculated by grouping the implicants $M = (X, S, S', Y)$ in a state transition table into the following subsets:

$$\begin{aligned} C^Y_{k,i} &= \{m_j \in M | s_j = s_k, (y_j)_i = 1\} \\ C^{S'}_{k,i} &= \{m_j \in M | s_j = s_k, s'_j = s_i\} \end{aligned}$$

$|C^Y_{k,i}|$ ($|C^{S'}_{k,i}|$) 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 $n_E - d_{k,l}$ literals can be extracted from them.

For an output function y_i, if we assume an unminimized, 1-level, 1-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_i is equal to

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

where $\gamma_{k,l,i} = |C^{S'}_{k,i}| + |C^{S'}_{l,i}|$ and λ_i is the number of 1's in state s_i. 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

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

This is the cost function used in JEDI [6] except that the last term in Eq. 25 is removed. MUSTANG [2] approximates λ_i by $n_S/2$, and uses multiplication instead of addition to calculate the weight function.

For low power applications, we have to minimize P_{reg}, P_{inputs} and P_{comb}. We first look at P_{inputs}. For power conscious state assignment, state transitions with high probability should be assigned higher weights. But 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 saving factor (weighted by the switching activity of the literals).

Consider two states s_i and s_j with a common cube B. The two state are encoded as follows.

$$s_i = \underbrace{b_1 b_2 \ldots b_K}_{B} | \underbrace{b_{K+1} \ldots b_{n_S}}_{B_i}$$

$$s_j = \underbrace{b_1 b_2 \ldots b_K}_{B} | \underbrace{b'_{K+1} \ldots b'_{n_S}}_{B_j}$$

The common cube B can be extracted from s_i and s_j as

$$s_i + s_j = B(B_i + B_j) = b_1 b_2 \ldots b_K (b_{K+1} \ldots b_{n_S} + b'_{K+1} \ldots b'_{n_S})$$

Let the set S_B denote the set of states whose corresponding bits have the same binary values as those in B and $\overline{S}_B = S - S_B$. The notation S_{b_i} is an abbreviation of $S_{\{b_i\}}$.

The power saving of extracting B from the present states s_i and s_j is equal to the number of literals saved weighted by the switching activities of the state bits in C. Thus the literal power saving in P_{input} is given by

$$\Delta^P_{k,l} = \{\sum_{i=1}^{n_Y} \mu_{k,l,i} + \sum_{i=1}^{n_S} (\lambda_i \gamma_{k,l,i})\} \{\sum_{j=1}^{K} TP(S_{b_j} \leftrightarrow \overline{S}_{b_j}) - TP(S_B \leftrightarrow \overline{S}_B)\} - \sum_{i=j}^{K} TP(S_{b_j} \leftrightarrow \overline{S}_{b_j}). \quad (26)$$

$TP(S_{b_j} \leftrightarrow \overline{S}_{b_j})$ is simply E_{b_j} where as $TP(S_B \leftrightarrow \overline{S}_B)$ is calculated by identifying S_B and applying equation (2) on S_B.

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, 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_{init} and then subtract the literal power saving to get the actual power cost. Therefore,

$$P_{inputs} = P_{init} - P_{saving} \quad (27)$$

where

$$P_{init} = \sum_{s_i \in S} \mu_{s_i} C_{lit} \sum_{j=1}^{n_E} E_{b_j} \quad (28)$$

$$P_{saving} = \sum_{s_k \in S} \sum_{s_l \in S} \Delta^P_{k,l} C_{lit} \quad (29)$$

where S is the set of all state, μ_{s_k} is the occurrence frequency of s_k which is given by

$$\mu_{s_k} = \sum_{i=1}^{n_Y} |C^Y_{k,i}| + \sum_{j=1}^{n_S} \lambda_j |C^{S'}_{k,j}|. \quad (30)$$

P_{reg} is equal to

$$P_{reg} = C_{reg} \sum_{j=1}^{n_E} E_{b_j}. \quad (31)$$

It is very difficult to estimate P_{comb} during the state encoding as it varies drastically with the actual multi-level logic implementation. We thus use equation (25) as a first order approximation for P_{comb}.

5.2 Multiple Objective Function Optimization by Simulated Annealing

The power cost functions measure the weighted switching activities of the circuit and the area cost function measures the number of literals (or product terms), it is difficult to find a good combining function. Using combined function like $Power + \beta Area$ is not appropriate since determining β is difficult. To include both measures, we propose a modification to the simulate annealing procedure such that two separate cost functions are used to control the moves. Let

$$P_1 = P_{regs} + P_{inputs} \tag{32}$$

and

$$P_2 = \sum_{s_k \in S} \sum_{s_l \in S} \Delta_{k,l} \tag{33}$$

for the case of multi-level implementation. An initial encoding and its corresponding P_1 and P_2 are obtained. Two states are randomly chosen for conditional swapping. If both P_1 and P_2 decrease, the codes for the two states are swapped. However, if either cost increases, the two codes are swapped with some probabilities calculated from the cost differences and the annealing temperature. The procedure is repeated until both power costs do not change for three consecutive temperatures.

6 Experimental Results

In this section we present experimental results of the low power state assignments algorithm for two level and multi-level implementation. Experiments were done using the MCNC-91 FSM benchmark sets.

The first experiment is to compare low power state assignment for two level implementation with NOVA which is a state assignment program targeting minimum area and with the minimum weighted Hamming distance encoding (MWHD) procedure of [12]. In this experiment, we used $C_{reg} = 20C_{AND}$ and $C_{AND} = C_{OR}$. For the low power state assignment algorithm, results are generated using two different options. The first option RC uses the cost of the valid rectangle cover as the cost function for P_{SI} while the second option $Heur$ uses the heuristic cost function (equation (24)). The encoded machines were synthesized using *espresso-exact* and the power consumption was measured using a sequential machine power estimator [14] [9]. Table 1 summarizes the results.

From Table 1, it is seen that in most of the benchmarks the low power state assignment produces better results than NOVA and MWHD encoding. Using valid rectangle covering cost function on results in an average 8.6% reduction in power compared to NOVA. If the heuristic cost function is used, the power consumption is reduced by 8.5%. 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 7.5 %.

The second experiment compares low power state assignment for multilevel implementation with JEDI which is a state assignment program targeting minimum area and with the MWHD encoding. Results were generated using two different options: the first one uses only P_1 in equation (32) as the cost functions and the second one use both P_1 and P_2 in equation (33) during the simulated annealing. The encoded machines were synthesized using the *mis rugged_script*. Table 2 summaries the results.

From Table 2, it is seen that in general the low power state assignment produces better results than JEDI and the MWHD encoding. Using only P_1 results in an average of 13.1% reduction in power compared to JEDI. Using both P_1 and P_2 gives an average 16.7% reduction in power consumption. For the MWHD encoding, the result is no better than JEDI. An average of 1.14% increase in power consumption is obtained.

We also counted the number of literals in the final implementation for each encoding. Although the literal counts for the low power state assignment algorithms are generally larger than that of JEDI (7.3% and 2.7% respectively), power consumption is lower since JEDI does not consider the switching activities. The minimum weighted Hamming distance code, however, has an average 17.5% 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 produce good low power state assignment.

7 Concluding Remarks

We presented a power cost model for the state assignment problem targeting both two- and multi-level logic implementation. 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. The impact of of PLA type was also discussed. In particular, we derived the power cost function for dynamic PLA. For pseudo-NMOS static PLA we need to consider the direct current drawn through the NOR gate when the output of the gate is zero and the corresponding power dissipation has to be included in the model. For multi-level logic implementation, we proposed a power cost function which captures the weighted switching activities at the inputs of the circuit and approximated the weighted switching activities inside the combination circuit by the literal counts function. We then described a modified simulated annealing procedure which handles multiple objective function.

Espresso [1] 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 two-level logic minimization algorithm for low power. The problem of bit-constrained state encoding problem was addressed in this

circuits	NOVA	Hamming	% red.	RC	% red.	*Heur.*	%red
bbara	403.01	391.87	2.76	381.08	5.44	372.68	7.53
bbsse	656.41	514.40	21.63	519.18	20.91	470.30	28.35
bbtas	238.62	195.40	18.11	202.72	15.04	170.56	28.52
beecount	328.67	221.89	32.49	216.24	34.21	204.71	37.72
cse	736.58	833.65	-13.18	687.40	6.68	748.26	-1.59
dk14	524.11	665.56	-26.99	506.38	3.38	538.73	-2.79
dk16	1121.00	1296.41	-15.65	1010.00	9.90	1211.60	-8.08
dk17	335.83	381.24	-13.52	350.63	-4.41	344.16	-2.48
dk27	180.14	213.57	-18.56	169.70	5.80	166.27	7.70
dk512	372.90	438.45	-17.58	370.54	0.63	364.05	2.37
donfile	1160.78	931.95	19.71	646.32	44.32	602.70	48.08
ex1	1253.08	1373.42	-9.60	924.90	26.19	995.43	20.56
ex6	580.03	698.52	-20.43	606.66	-4.59	557.05	3.96
kirk	1312.09	1533.15	-16.85	1252.93	4.51	1506.55	-14.82
keyb	1116.63	1858.54	-66.44	965.59	13.53	910.31	18.48
lion9	200.38	225.20	-12.39	199.01	0.68	209.98	-4.79
planet	2778.56	2901.43	-4.42	2539.74	8.60	2836.69	-2.09
pma	896.70	920.70	-2.68	860.98	3.98	920.50	-2.65
s1	1842.00	2029.30	-10.17	1818.40	1.28	2053.20	-11.47
s27	267.46	224.41	16.10	223.95	16.27	211.80	20.81
s208	271.12	439.59	-62.14	398.52	-46.99	312.53	-15.27
sand	2519.30	2723.20	-8.09	2229.80	11.49	2359.00	6.36
sse	656.41	514.40	21.63	519.18	20.91	470.30	28.35
styr	1949.23	2090.51	-7.25	1666.00	14.53	1770.36	9.18
tma	806.49	695.13	13.81	724.12	10.21	666.08	17.41
train11	237.60	275.84	-16.09	234.50	1.30	233.13	1.88
average %reduciton			-7.53		8.61		8.51

Table 1: Power consumption for 2-level targeting dynamic PLA

circuits	JEDI	P1	% red.	P1 and P2	% red.	Hamming	% red.
bbara	254.60	249.80	1.89	228.40	10.29	302.60	-18.85
bbsse	847.20	570.10	32.71	513.40	39.40	685.30	19.11
bbtas	96.10	75.00	21.96	79.98	16.77	75.71	21.22
beecount	227.16	238.10	-4.82	240.77	-5.99	233.66	-2.86
cse	983.45	961.79	2.20	936.45	4.78	1159.05	-17.86
dk14	741.18	672.99	9.20	901.38	-21.61	1036.63	-39.86
dk16	4435.06	3532.06	20.36	3731.65	15.86	3909.95	11.84
dk17	474.73	429.49	9.53	386.52	18.58	489.90	-3.20
dk27	248.15	210.41	15.21	211.19	14.89	195.71	21.13
dk512	638.22	545.79	14.48	568.61	10.91	499.79	21.69
donfile	958.12	1147.00	-19.71	1177.96	-22.94	1390.83	-45.16
ex1	1623.95	1229.41	24.30	1050.23	35.33	1730.46	-6.56
ex6	522.91	638.94	-22.19	661.32	-26.47	664.55	-27.09
kirk	2681.13	2737.48	-2.10	2216.64	17.32	3000.00	-11.89
keyb	2251.09	1370.67	39.11	1972.37	12.38	3033.52	-34.76
lion9	142.83	141.93	0.63	103.90	27.26	185.42	-29.82
planet	10320.10	6111.28	40.78	5796.22	43.84	6388.06	38.10
pma	2570.60	2769.74	-7.75	1907.97	25.78	1743.49	32.18
s1	2432.03	2509.41	-3.18	1106.83	54.49	2488.23	-2.31
s208	491.29	451.86	8.03	363.46	26.02	521.24	-6.10
s27	127.30	112.82	11.37	106.03	16.71	175.27	-37.68
sand	6592.30	5788.88	12.19	7433.67	-12.76	6370.60	3.36
styr	4813.60	3828.47	20.47	4502.16	6.47	4439.93	7.76
sse	847.16	570.01	32.72	513.43	39.39	685.00	19.14
tma	1968.74	1488.14	24.41	1006.82	48.86	1438.31	26.94
train11	176.89	69.97	60.44	107.16	39.42	120.62	31.81
average %reduction			13.16		16.73		-1.14

Table 2: Power consumption for multi-level

report. Future work will address the general state assignment problem without any constraints on the number of state bits.

References

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

[2] S. Devadas, H-K. T. Ma, A. R. Newton, and A. Sangiovanni-Vincentelli. MUSTANG: State assignment of finite state machines targeting multi-level logic implementations. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 7, pages 1290–1300, December 1988.

[3] S. Devadas and A. R. Newton. Exact algorithms for output encoding, state assignment and four-level Boolean minimization. In *Proceedings of the Twenty Third Hawaii International Conference on the System Sciences*, volume I, pages 387–396, January 1990.

[4] X. Du et al. MUSE: A MUltilevel Symbolic encoding algorithm for state assignment. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 28–38, January 1991.

[5] S. Kirkpatrick, C. D. Gelatt, and M. P.Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[6] B. Lin and A. R. Newton. Synthesis of multiple-level logic from symbolic high-level description languages. In *IFIP International Conference on Very Large Scale Integration*, pages 187–196, August 1989.

[7] G. De Micheli. Symbolic design of combinational and sequential logic circuits implemented by two-level macros. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 5, pages 597–616, September 1986.

[8] G. De Micheli, R. K. Brayton, and A. Sangiovanni-Vincentelli. Optimal state assignment of finite state machines. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 4, pages 269–285, July 1985.

[9] J. Monteiro, S. Devadas, and A. Ghosh. Estimation of switching activity in sequential logic circuits with applications to synthesis for low power. In *Proceedings of the 31th Design Automation Conference*, page , June 1994.

[10] E. Olson and S. M. Kang. Low-power state assignment for finite state machines search. In *International Workshop on Low Power Design*, April 1994.

[11] A. Papoulis. *Probability, Random Variables and Stochastic Processes.* McGraw-Hill, 1984.

[12] K. Roy and S. Prasad. SYCLOP: Synthesis of CMOS logic for low power application. In *Proceedings of the International Conference on Computer Design*, pages 464–467, October 1992.

[13] R. Rudell. *Logic Synthesis for VLSI Design.* PhD thesis, 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 *Proceedings of the 31th Design Automation Conference*, page , June 1994.

[15] T. Villa and A. Sangiovanni-Vincentelli. NOVA: State assignment of finite state machines for optimal two-level logic implementations. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 9, pages 905–924, September 1990.

[16] S. Yang and M. Ciesielski. On the relationship between input encoding and logic minimization. In *Proceedings of the Twenty Third Hawaii International Conference on the System Sciences*, volume I, pages 377–386, January 1990.