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

Chi-Ying Tsui, Massoud Pedram, Chih-Ang Chen, Alvin M. Despain

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

Abstract

The problem of minimizing power consumption during the state encoding of a finite state machine is considered. A new power cost model for state encoding is proposed and encoding techniques that minimize this power cost for two- and multi-level logic implementations are described. 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 work, we address the problem of minimizing the power consumption in sequential machines. Since the power consumption in a finite state machine (FSM) is strongly influenced by state encoding of the machine, we set as our goal the development of encoding algorithms that lead to low power implementations after two- or multi-level logic optimization.

State assignment algorithms that minimize the area of the circuit after logic optimization have been extensively researched. This problem is NP-hard and various methods have been proposed to find a solution. De Micheli et al. [4] described a paradigm in which one-hot codes are assigned to states and a minimum symbolic (multi-valued) cover of the machine is generated by output-disjoint minimization. This symbolic cover defines a set of face embedding constraints that require certain states to have codes that lie on the same face of a hypercube of minimum dimensionality. The minimum area state encoding problem for two-level logic is thus transformed to the problem of finding the minimum number of encoding bits such that all constraints are satisfied. De Micheli [3] used a heuristic row encoding technique to solve this problem. Villa et al. [10] employed the notion of face-posets to tackle this problem. Yang et al. [11] formulated the problem as a unate covering problem (covering seed dichotomies by a minimum-cost set of prime dichotomies) and solved it by using a heuristic technique.

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.

*This research was supported in part by the NSF's Research Initiation Award under contract no. MIP-9211668 and the DAPRA under contract no. J-FBI-91-194.

For two-level logic implementation, NOVA [10] 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 rewards higher cube sharing has been used. In particular, MUSTANG [1] and JEDI [2] 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 used the sum of the weights over all pairs of states as the cost function.

The dynamic power consumption of a CMOS gate is proportional to the product of C_{load} and E_{sw} where C_{load} denotes the load capacitance that the gate is driving and E_{sw} denotes the average switching activity at the gate output per clock cycle. State encoding for low power is harder than that for minimum area since it has to consider both the area and switching activity and the switching activity is not known until after the encoding.

Roy et al. addressed the problem of reducing the switching activity during state assignment in [7]. They assumed the power consumption is proportional to the switching activity on the state bit lines of the machine and hence used the following cost function:

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

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. 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 activity on the present state bits without any consideration of the loading on the state bits and the power consumption in the combinational logic part of the machine.

In an attempt to account for power consumption in the combinational logic, Olson et al. [5] 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

©1994 ACM 0-89791-690-5/94/0011/0082 $3.50

used for the search strategy. We first present a power cost model for state assignment which considers both the capacitive loading and the switching activity simultaneously. We then propose accurate power cost functions for both two- and multi-level logic implementations. For two-level logic using PLA implementations, the dichotomy-based approach of [11] is extended to calculate the proposed power cost function. For multi-level logic implementation, the cost function of [2] is modified to take into account the weighted switching activity at the inputs of the FSM.

The remainder of the paper 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. Experimental results and conclusions are given in Sections 6 and 7.

2 Terminology

A FSM is characterized by a 5-tuple (X, Y, S, λ, η) where X, Y, S are the sets of primary inputs, primary outputs and internal states and λ, η are the output and next state functions respectively. 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 *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-Kolmogorov equations [6].

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 state transition probability can be generalized to transitions between two sets of disjoint states, $S_i, S_j \subset S$, as follows:

$$TP(S_i \leftrightarrow S_j) = \sum_{s_i \in S_i} \sum_{s_j \in S_j} (tp_{s_i,s_j} + tp_{s_j,s_i}). \quad (1)$$

The switching activity 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

$$E_{b_i} = TP(States(b_i = 1) \leftrightarrow States(b_i = 0)) \quad (2)$$

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 highlighted in the figure and explained below.

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

$$P_{reg} = \sum_{b_i \in state\_bits} C_{reg} E_{b_i} \quad (3)$$

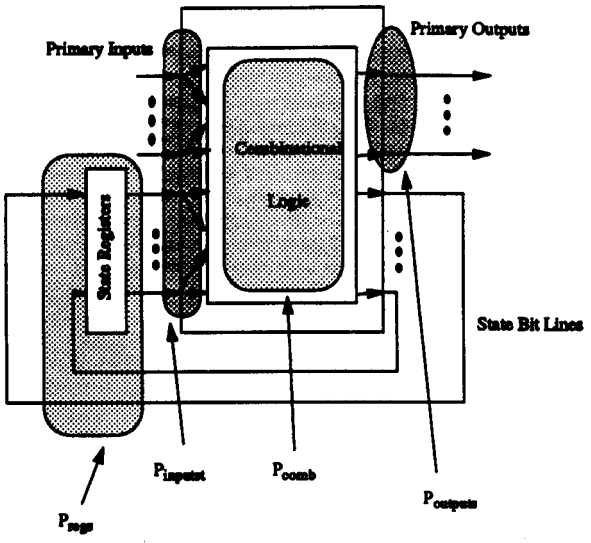

Figure 1: Power model for finite state machines.

where C_{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 equation (2).

P_{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 and is given by

$$P_{inputs} = \sum_{b_i \in state\_bits} n_i C_{lit} E_{b_i} + \sum_{j \in PI} n_j C_{lit} E_j \quad (4)$$

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

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

$$P_{comb} = \sum_{n \in NODES} C_n E_n \quad (5)$$

where C_n is the capacitance that node n is driving.

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

$$P_{outputs} = \sum_{o \in PO} C_o E_o \quad (6)$$

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

Under a zero delay model $P_{outputs}$ is constant and independent of the state encoding and can be dropped when comparing the power costs of different state encodings. We therefore minimize $P_{reg} + P_{inputs} + P_{comb}$.

State encoding schemes that minimize the Hamming distance between state pairs with high transition probabilities tend to minimize E_{b_i} and hence P_{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. 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.

4 Two-level logic implementation

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} while P_{inputs} and P_{comb} can be lumped into a single term P_{AND} where

$$P_{AND} = \sum_{BI \in logic\_cover} P_{BI} \tag{7}$$

and BI is a binary implicant of the logic cover. Let the binary 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

$$P_{BI} = C_{AND}(\sum_{i=1}^{n} E_{x_i} + \sum_{j=1}^{m} E_{b_j}) + n_{OR} C_{OR} E_{BI} \tag{8}$$

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

The type of PLA used for the implementation has a direct impact on the power cost calculation. For 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 1 during the precharge period and switches only when its output is evaluated to 0 during the evaluation period. Since the output of the NOR gate is $\overline{NS_i}$, the switching probability of the next state bit line NS_i is equal to $prob(\overline{NS_i} = 0)$. In addition, we have to include P_{clock}, which is the power consumption at the clocked transistors for the precharge and evaluation of each NOR gate. Therefore the P_{BI} for 2-level logic circuit using dynamic PLA is equal to

$$P_{BI} + P_{clock}. \tag{9}$$

For 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.

4.1 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 (2). However if we want to compute P_{AND} exactly, we have to know the exact implementation which will be known only after logic minimization. In way of compromise, we use the power cost of the symbolic implicants to approximate P_{AND} as detailed next.

Let $SI = (x, s, s', y)$ be a symbolic implicant. If SI is realized by a single binary implicant BI, then the power cost of realization of this symbolic cube is

$$P_{SI} = P_{BI}. \tag{10}$$

Otherwise, let $BI_1 \ldots BI_q$ be the set of binary implicants that realize SI, then the power cost of this realization is

$$P_{SI} = \sum_{i=1}^{q} P_{BI_q}. \tag{11}$$

P_{AND} is then given by

$$P_{AND} = \sum_{SI \in symbolic\_cover} P_{SI}. \tag{12}$$

The key issue is, of course, to find the minimum power implementation of a symbolic implicant SI, i.e. finding $BI_1, \ldots, BI_q$ that minimize the power. To do this, we use the concept of dichotomy that has also been used for state assignment targeting minimum area [11].

We need some definitions and use the example shown in Figure 2 for this purpose.

Definition 4.1 *Given a symbolic implicant $< x, s, s', y >$, the s part 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 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 2, 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. If $ed_i = (l_i, r_i)$, then $ed_i^T = (r_i, l_i)$.*

Example. In Figure 2, the encoding dichotomies for b_1 and $\bar{b}_1$ are $(s_1s_3, s_2s_4s_5)$,$(s_2s_4s_5, s_1s_3)$.

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, 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*

$$\bigcup_{ed_i \in ED} pc_{j,i} = l_j \tag{13}$$

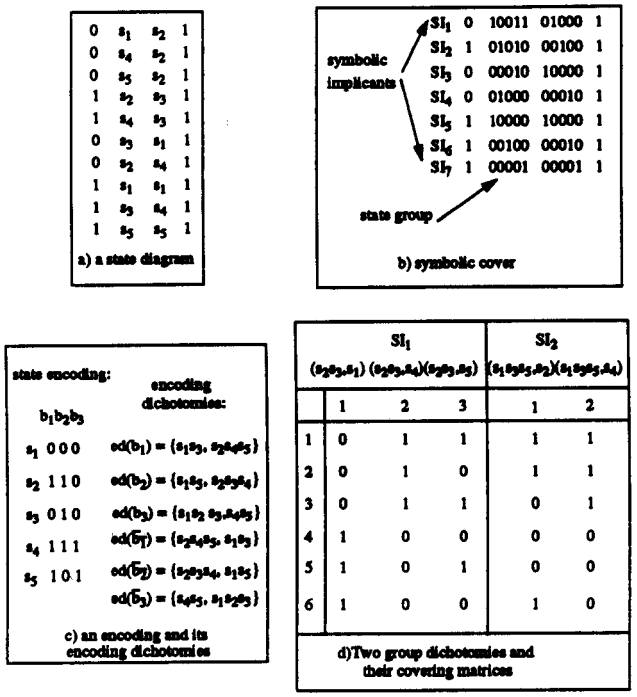

Figure 2: Example illustrating the definitions.

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.

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

$$\forall_{i \in R \wedge j \in C} M_{ij} = 1 \tag{14}$$

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

$$\cup_{i \in R} pc_i = z_g. \tag{15}$$

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 2d 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 $\tilde{ib}_1b_3$ and $\tilde{ib}_2$. 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 ib_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

$$P_{SI} = \sum_{(R_i,C_i) \in \{(R_1,C_1),\ldots,(R_k,C_k)\}} P_{BI_{(R_i,C_i)}} \tag{16}$$

where $BI_{(R_i,C_i)}$ 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 multi-level logic optimization and is shown to be NP-hard [8]. To solve the valid rectangle covering problem, we therefore resort to a heuristic method.

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 activity 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

$$cost(ed_i) = \frac{E_{ed_i}}{seed\_cov(ed_i) zero\_block\_cov(ed_i)} \tag{17}$$

where E_{d_i} is simply the switching activity of state bit i, $seed\_cov(ed_i)$ is the ratio of the number of seed dichotomies covered by ed_i and the total number of the seed dichotomies, and $zero\_block\_cov(ed_i)$ 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 2d, the cost of ed_{b_1} for SI_1 is thus $E_{b_1}/(0.666 \times 0.5)$.

5 Multi-level logic implementation

For area minimization, the objective of the state assignment is to minimize the number of literals in the multi-level

logic implementation. The literal savings cost function has been well studied in [2] and [1]. 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:

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

where $|C^Y_{k,i}|$ and $|C^{S'}_{k,i}|$ represent the occurrence frequency of state s_k in the output and the next state functions, respectively.

Let n_B 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 $n_B - d_{k,l}$ 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 savings 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 savings 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 savings of extracting a common cube C from states s_k and s_l is

$$_{k,l} = \{\sum_{i=1}^{n_Y} \mu_{k,l,i} + \sum_{i=1}^{n_S} \lambda_i \gamma_{k,l,i}\}(|B| - 1) - |B|. \quad (18)$$

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

$$^P_{k,l} = \{\sum_{i=1}^{n_Y} \mu_{k,l,i} + \sum_{i=1}^{n_S} (\lambda_i \gamma_{k,l,i})\}\{\sum_{j=1}^{K} TP(S_{b_j} \leftrightarrow \overline{S}_{b_j}) - TP(S_B \leftrightarrow \overline{S}_B)\} - \sum_{i=j}^{K} TP(S_{b_j} \leftrightarrow \overline{S}_{b_j}). \quad (19)$$

$TP(S_{b_j} \leftrightarrow \overline{S}_{b_j})$ is simply E_{b_j} whereas $TP(S_B \leftrightarrow \overline{S}_B)$ is calculated by identifying S_B and applying equation (1) 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 savings can be used as a metric for the actual area savings, the initial power cost does depend on the encoding and hence the above literal power savings 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 savings to get the actual power cost. Therefore,

$$P_{inputs} = P_{init} - P_{savings} \quad (20)$$

where

$$P_{init} = \sum_{s_i \in S} \mu_{s_i} C_{lit} \sum_{j=1}^{n_B} E_{b_j} \quad (21)$$

$$P_{savings} = \sum_{s_k \in S} \sum_{s_l \in S} {}^P_{k,l} C_{lit} \quad (22)$$

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

$$\mu_{s_k} = \sum_{i=1}^{n_Y} |C^Y_{k,i}| + \sum_{j=1}^{n_S} \lambda_j |C^{S'}_{k,j}|. \quad (23)$$

6 Experimental results

In this section we present experimental results of the low power state assignments algorithm for two level and multilevel implementations. Experiments were done using the MCNC-91 FSM benchmark sets. The power consumption was measured in μW using a sequential machine power estimator [9], assuming 5V power supply and 20MHz clock frequency.

The first experiment is to compare low power state assignment (LPSA) for two level implementation using dynamic NOR-NOR PLA with NOVA [10] which is a state assignment program targeting minimum area and with the minimum weighted Hamming distance encoding (MWHD). The encoded machines were synthesized using *espresso-exact*. Table 1 summarizes the results. Columns 3 and 5 give the % 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. An average 9.1% 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 6.3 %.

The second experiment compares low power state assignment (LPSA) for multilevel implementation with JEDI [2] which is a state assignment program targeting minimum

circuits	NOVA	MWHD	% red.	LPSA	% red.
bbara	361.5	362.9	-0.38	339.4	6.12
bbsse	508.8	408.7	19.68	411.2	19.19
beecount	281.3	205.1	27.10	222.8	20.79
cse	630.4	668.4	-6.02	614.9	2.47
dk14	440.3	541.5	-22.99	437.7	0.58
dk16	898.3	991.8	-10.41	886.2	1.35
dk512	342.8	370.3	-8.05	318.9	6.95
donfile	812.2	729.9	10.14	547.9	32.55
ex1	821.8	857.1	-4.29	665.1	19.07
ex4	378.9	336.1	11.27	278.2	26.56
ex6	429.1	474.1	-10.50	403.6	5.92
keyb	846.5	1454.4	-71.81	751.2	11.26
planet	1420.9	1371.7	3.46	1255.8	11.62
pma	640.4	750.1	-17.13	653.1	-1.98
sand	1427.9	1424.4	0.24	1358.9	4.83
s1	1165.1	1349.1	-15.79	1211.6	-3.99
s208	267.7	406.5	-51.86	314.6	-17.54
s27	251.8	237.5	5.68	223.5	11.22
styr	1345.5	1403.9	-4.34	1277.5	5.06
sse	519.3	413.8	20.31	411.2	20.82
aver. %red.			-6.28		9.14

Table 1: Power consumption for two-level implementation

circuits	JEDI	MWHD	% red.	LPSA	% red.
bbara	197.9	214.7	-8.46	213.0	-7.61
bbsse	579.4	472.0	18.53	433.9	25.11
beecount	249.2	258.9	-3.90	193.5	22.35
cse	532.0	550.4	-3.46	477.0	10.35
dk14	548.9	649.9	-18.38	557.6	-1.58
dk16	1600.7	1350.7	15.62	1425.4	10.95
dk512	619.5	407.5	34.21	501.7	19.01
donfile	652.4	710.1	-8.85	608.2	6.77
ex1	954.0	1018.7	-6.78	830.0	13.00
ex4	505.8	338.8	33.02	285.5	43.56
ex6	469.8	501.3	-6.71	462.4	1.56
keyb	876.2	1022.7	-16.71	605.2	30.93
planet	3356.1	2008.8	40.1	1848.2	44.93
pma	1196.2	661.6	44.68	942.4	21.21
s1	1247.7	1209.2	3.08	1126.3	9.72
s208	393.6	384.2	2.40	324.3	17.61
s27	128.5	202.0	-57.15	128.8	-0.22
sand	1706.3	1629.3	4.51	1559.3	8.61
sse	579.4	472.0	18.53	433.9	25.11
styr	1426.9	1184.8	16.97	1264.7	11.36
aver. %red.			5.06		15.64

Table 2: Power consumption for multi-level implementation

area and with MWHD. The encoded machines were synthesized and mapped using the SIS package and a a 0.8μ library from industry. Table 2 summaries the results. Columns 3 and 5 give the % power reduction of MWHD and LPSA over JEDI, respectively.

Table 2 shows that in general the low power state assignment produces better results than JEDI and MWHD. An average of 15.6% reduction in power consumption is obtained compared to JEDI. Compared to MWHD, the average reduction in power consumption is about 8%.

We also counted the number of literals in the final implementation for each encoding. Although the average literal count for the low power state assignment algorithms is 3.2% larger than that of JEDI, power consumption is lower since LPSA considers the switching activity. The minimum weighted Hamming distance code, however, has an average 14.9% more literals than that obtained from JEDI. This, and the fact that the capacitive loading of the state bits are ignored, explain why MWHD 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. For multi-level logic implementation, we proposed a power cost function which captures the weighted switching activity at the inputs of the circuit.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] E. Olson and S. M. Kang. Low-power state assignment for finite state machines search. In *International Workshop on Low Power Design*, pages 63–68, April 1994.

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

[7] 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.

[8] R. Rudell. *Logic Synthesis for VLSI Design*. PhD thesis, University of California, Berkeley, 1989.

[9] 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*, pages 18–23, June 1994.

[10] 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.

[11] 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.