Joint Conference of 1996 International Computer Symposium
December 19~21, Kaohsiung, Taiwan, R.O.C.

State Assignment for Low Power Consumption in Sequential Circuits

Sying-Jyan Wang and Ming-De Horng

Institute of Computer Science
National Chung-Hsing University
Taichung, Taiwan, R.O.C. 40227

Abstract

In this paper we present an algorithm for the state assignment in finite state machines targeted for minimal switching power dissipation. Two states are assigned codes closer in Hamming distance if the state transition probability between them is higher by our algorithm. In this way we can reduce the average number of bit changes associated with state transitions. The proposed algorithm achieves this goal by modifying the given state transition graph so that it can be embedded into an n-cube. Experimental results show that this method greatly reduce the switching activity generated by state transitions.

1. Introduction

Low-power design has attracted tremendous attention in recent years. The advance in technology enables us to put more and more devices in a single chip while at the same time pushes the clock rate even higher. Low power design is thus necessary to reduce the packaging and cooling costs as well as prolong the life span of ICs. The second source of requirement for lower power design comes from the low power applications. For these applications, low power designs extend the battery lifetime.

Two strategies are used in low-power design [1]. Architecture-independent strategy tries to reduce power consumption in integrated circuits by modifying processing technology so that the fabricated devices consume less power. For examples, Silicon-On-Insulator (SOI) technology can reduce parasitic substrate capacitance. Devices with lower operating voltage also consume less power [2].

Technology-independent strategy reduces power consumption through a refined design process. An obvious method to reduce the power consumption is to shut-down part of a circuit when it is not in operational condition. Many studies have also been carried out to minimize the average power dissipation by reducing switching activities of a given logic circuit. The minimization can be achieved at technology mapping phase [3],[4], logic design phase [5], or through code assignment for finite state machine (FSM) design [6]-[8].

In this paper we study the problem of low power synthesis of FSM. The reduction in power consumption is achieved by minimizing switching rates caused by state transitions. A state transition in an FSM requires single or multiple bit transitions. Therefore, switching rate can be minimized if states associated with state transitions that appear most frequently are assigned codes that are close to each other. We propose an algorithm that assigns codes to states in FSMs targeted for low switching power consumption.

This paper is organized as follows. In Section 2 we present some preliminary information. Our algorithm is described in Section 3, while experimental results are given in Section 4. Finally, concluding remarks are given in Section 5.

2. Preliminaries

In this section we provide some background information.

Acknowledgments: This work was supported by National Science Council under Grant No. NSC85-2215-E-005-001

2.1. Power Dissipation in CMOS Circuits

CMOS is currently the dominant technology. There are two components that contribute to the power dissipated in CMOS circuits [9]. The static dissipation is due to leakage current, while dynamic power dissipation is due to switching transient current as well as charging and discharging of load capacitances.

Since the amount of leakage current is usually small, the major source of power dissipation in CMOS circuits is the dynamic power dissipation. Dynamic power dissipation appears only when a CMOS gate switches from one stable state to another. In the dynamic power dissipation, the component due to charging and discharging of load capacitance is usually the dominant factor. Thus the average dynamic dissipation of a CMOS gate is:

$$P_{avg} = \frac{1}{2} \times C_L \times V_{DD}^2 \times f_p \times N$$

where C_L is the load capacitance, V_{DD} is the power supply voltage, f_p is the clock frequency, and N is the average number of switching activities in a clock cycle. Thus, the power consumption can be certainly reduced if one can reduce the switching activity of a given logic circuit without changing its function [10].

2.2. Finite State Model of Sequential Circuits

A general sequential circuit is shown in Fig. 1. The circuit consists of two parts: a combinational circuit that performs the logic function, and feedback registers that hold the state information. The inputs of the combinational logic include the primary inputs and the present state inputs. The output side of the combinational logic also has two parts: the primary outputs and the next state lines.

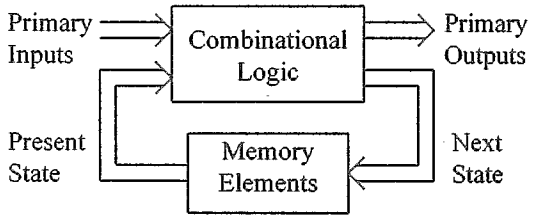

Fig. 1. A sequential machine.

2.2.1. The Finite State Machine Model

A finite state machine (FSM) is an abstract model describing a synchronous sequential machine. A Mealy machine M is a quintuple $M = (I, O, S, \delta, \lambda)$ where I, O, and S are finite, nonempty sets of input symbols, output symbols, and states, respectively. The function δ: $I \times S \rightarrow S$ specifies all state transitions, and λ: $I \times S \rightarrow O$ is the output function.

The behavior of a FSM is usually described by a state transition graph (STG). An STG is a directed graph in which a vertex represents a state while an edge represents a state transition. The STG of an FSM is shown in Fig. 2.

Associated with an edge, which starts at state s_i and terminates at state s_j, is a pair p_k/q_k, where $p_k \in I$ and $q_k \in O$. This edge indicates that, if the present state is s_i and the input symbol is p_k, the machine will be transferred to state s_j with output symbol q_k. In order to get the final circuit, all symbols in I, O, and S are encoded with binary codes. A binary encoded input symbol will be called an *input vector* henceforth.

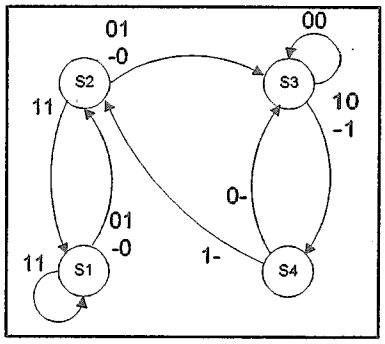

Fig. 2. An example STG.

2.2.2. State Transition Probability

Since we are interested in minimize the average switching rate due to state transitions in FSMs, first we have to find a way to calculate the probability associated with each state transition. In [6] a method is proposed to calculate state transition probability. We briefly introduce the method here.

First, a *local state transition probability matrix* is calculated. The local state transition probability associated with a state transition from s_i to s_j, being denoted as lp_{ij}, is the probability that a primary input will bring the machine to state s_j given that the current state is s_i. If the distribution of input vectors fed to the circuit is known, each lp_{ij} can be calculated accordingly. Otherwise, uniform input pattern (i.e., each legal input vector appears with the same frequency) is assumed. Thus lp_{ij} is the ratio of the number of minterms causing such a transition to the total number of valid minterms at state s_i. The local state transition probability matrix is defined as $LP = \{lp_{ij}\}^{m \times m}$ where m is the number of states in the STG.

When an FSM is in the steady-state, the probability

that the FSM stays at s_i is denoted as sp_i. Thus sp_i is the accumulated proportion of the time residing at state s_i during the operating period. The *steady-state probability of state matrix* is denoted as $SP = \{sp_i\}^{1\times m}$. Since in the steady-state we may assume that sp_i does not change with time, the following equation holds:

$$SP = SP \times LP \text{ or } (LP - 1)^T \times SP^T = 0$$

The above equation has only $m-1$ linearly independent equations. Thus, to obtain SP, one additional equation is required:

$$\sum_{i=1}^{m} sp_i = 1$$

The probability for state transition from s_i to s_j, denoted as tp_{ij}, is just $sp_i \times lp_{ij}$. The *state transition probability matrix* is defined as $TP = \{tp_{ij}\}^{m\times m}$. For example, the steady-state probability and the state transition probability for the STG shown in Fig. 2 are given in Fig. 3.

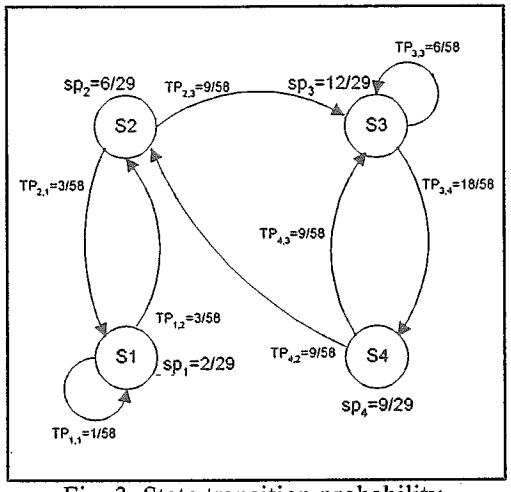

Fig. 3. State transition probability.

2.2.3. Switching Activity

Our goal in this research is to minimize the switching activity due to state transitions in FSMs. Therefore, we must have a way to estimate the amount switching activity. Let the binary encoding of state s_i be denoted as $enc(s_i)$. The Hamming distance between two codewords is the number bit positions that the two codewords differ. The switching activity can be defined as:

$$\sum_{i=1}^{m}\sum_{j=1}^{m} tp_{ij} \times Hamming\_distance(enc(s_i), enc(s_j))$$

The switching activity is the average number of bit changes required for a state transition. An FSM with higher switching activity indicates a higher level of switching rate in the real circuit.

2.3. Bipartite Representation of Graphs

The ultimate goal of code assignment is to assign binary code to states in an FSM. The code space consisting of all 2^n n-nit binary words is usually called a binary n-cube. Since an n-cube is always a bipartite graph, it is useful to study the relationship between these two graphs. In this section we briefly introduce the work by Kang, Wey, and Fisher, which discusses how to represent bipartite graphs and n-cubes with tables [11]. Two types of tables, which are derived from bipartite graphs, are introduced. One is the bipartite adjacency table (BAT), and the other is the bipartite representation table (BRT) for n-cube. Most of the notations in this section are borrowed from [11].

2.3.1. Bipartite Adjacency Table

A bipartite graph $G = (V, E)$ is a connected graph, in which the set of vertices V is divided into two disjoint subsets X_r and X_c (r and c stand for column and row, respectively). No vertices in the same subset are connected. In other words, if one endpoint of an edge $e \in E$ is in subset X_c, then the other endpoint of e must be in subset X_r. Let $|X_r| = x$, and $|X_c| = y$ (i.e., x and y are the cardinalities of X_r and X_c, respectively). To simplify the notation, the vertices in X_c and X_r are represented as S_i and s_j (i.e., $X_c = \{S_1, S_2, \ldots, S_y\}$ and $X_r = \{s_1, s_2, \ldots, s_x\}$. The set of edges in a bipartite graph is thus $E = \{(S_i, s_j) \mid S_i \in X_c \text{ and } s_j \in X_r\}$.

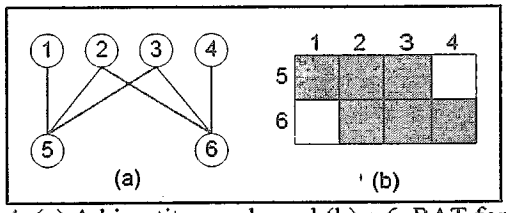

Fig. 4. (a) A bipartite graph, and (b) a 6-BAT for (a).

Fig. 4(a) is an example of bipartite graph and Fig. 4(b) is its corresponding bipartite adjacency table. If the number of vertices in a bipartite graph is m (i.e., $x + y = m$), the corresponding bipartite adjacency table will be called an m-BAT.

2.3.2. Bipartite Representation Table

Since an n-cube must be a bipartite graph, we can represent the connection pattern in an n-cube with a bipartite adjacency table. In Fig. 5 we show a 3-cube and its corresponding 3-BATs. A 3-cube consists of two 2-cubes and the edges connecting them, as shown

in Fig. 5(a). In this figure, vertices $\{a, b, c, d\}$ are elements of one 2-cube, while the other 2-cube consists of vertices $\{e, f, g, h\}$. The two 2-cubes, connected by four edges, form a 3-cube. These edges are $\{(a, e), (b, f), (c, g), (d, h)\}$. The 2-cube consisting of $\{a, b, c, d\}$ is denoted as $(a, c \mid b, d)$. This notation implies that any one of vertices $\{a, c\}$ is adjacent to any vertex in $\{b, d\}$. However, vertices a and c are not adjacent to each other, nor are vertices b and d adjacent to each other. Similarly, the 2-cube formed by $\{e, f, g, h\}$ is denoted as $(f, h \mid e, g)$. In the same way, the 3-cube can be denoted as $((a, c \mid b, d) \mid (f, h \mid e, g))$. This implies that vertices $\{a, c\}$ are not adjacent to vertices $\{f, h\}$, and vertices $\{b, d\}$ are not adjacent to $\{e, g\}$. Therefore, the connection pattern of a 3-cube can be represented by a bipartite representation table, as shown in Fig. 5(b). Fig. 5(c) and Fig. 5(d) are both equivalent to Fig. 5(b). Larger BRTs can be constructed from smaller BRTs recursively.

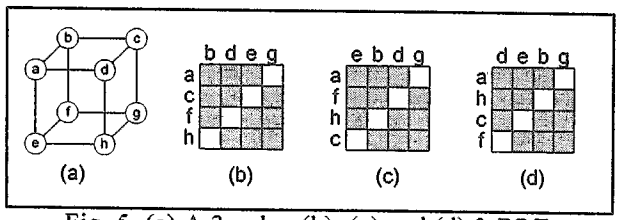

Fig. 5. (a) A 3-cube, (b), (c) and (d) 3-BRT.

3. State Assignment

In this section we present our approach to state assignment in FSM for low power consumption.

3.1. Basic Strategy

In order to reduce power dissipation in a circuit, one may want to reduce the switching rate of a given logic circuit. One way to achieve this goal in FSM design is to minimize the number of bit changes when state transition occurs. The optimal condition is that whenever there is an edge in the STG (except for the self loops), the two states connected by the edge should differ in only one bit position. In this way, each state transition involves at most one bit change. Obviously, this goal is not achievable in most cases. Whenever this happens, we shall try make the codewords of adjacent states as close to each other as possible. This strategy reduces the number of bit transitions associated with state transitions, which in turn reduces the switching activity of memory elements. The reduction of switching activity in the present-state part of the combinational logic (see Fig. 1) can also reduce the internal switching activity of the combinational circuit [12], so that the power dissipation in the combinational circuit can be reduced, too.

The basic strategy of our approach is described as follows. First, we compute the state transition probability matrix TP. A weighted undirected graph $G' = (V, E', W(E'))$ is obtained from the STG $G = (V, E)$ and TP. The set of vertices in G' is the same as that in G. An undirected edge $e=(s_i, s_j)$ exists in E' if at least one of two edges $\{(s_i, s_j), (s_j, s_i)\}$ exists in G and $i \neq j$. The weight on e $w(e)$ is $tp_{ij} + tp_{ji}$. Since weights on self-loops do not affect our state assignment, self-loops are removed in G'. Such a graph is referred to as the *state adjacency graph* henceforth. The state adjacency graph obtained from the STG in Fig. 2 is given in Fig. 6. The goal of the state-assignment algorithm is thus to minimize the following expression.

$$\sum_{i=1}^{m} \sum_{j=i+1}^{m} w(e(s_i, s_j)) \times Hamming\_distance(enc(s_i), enc(s_j))$$

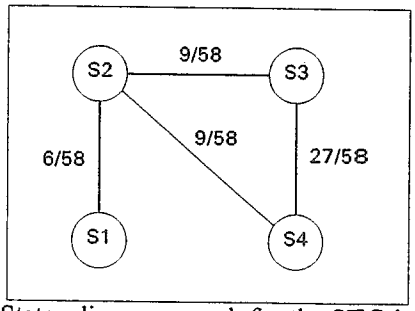

Fig. 6. State adjacency graph for the STG in Fig. 2.

The next step is to embed the resulting graph into a binary n-cube, where $n = \lceil \log_2 m \rceil$ and m is the number of states. However, most state adjacency graphs can not be embedded directly. Whenever G' is not embeddable, the best we can do is to embed a graph $G'' = (V, E'', W(E''))$, which is a subgraph of the state adjacency graph G', such that G'' preserves the maximum sum of weights in G'. The basic strategy of our code assignment can thus be expressed as follows.

Step 1. Obtain the state adjacency graph G'.
Step 2. Generate an embeddable graph G'' from G'.
Step 3. Embed G'' into an n-cube.

3.2. Generation of Bipartite Graph

The following theorem gives the necessary conditions under which a state adjacency graph can be embedded.

Theorem 1: Consider a state adjacency graph in which there are m states. Let $n = \lceil \log_2 m \rceil$. If the graph can be

embedded into an n-cube, then:
1) each state in the graph is adjacent to at most n other states, and
2) if there are cycles in the graph, the length of the cycle must be even.

Proof: Since each node in an n-cube is adjacent to only n other nodes, it is obvious that a state adjacency graph in which at least one state has more than n neighbors cannot be embedded.

To see why (2) is true, remember that each n-cube is a bipartite graph. Let the two sets of unconnected vertices be V_0 and V_1. Let the length of a path be l, l is an odd number, and let the vertices on this path be denoted as v_i, $0 \le i \le l$. Without loss of generality, assume vertex v_0 belongs to V_0. From the definition of bipartite graphs, we have $v_1 \in V_1$, $v_2 \in V_0$, etc. The other endpoint, v_l, thus belongs to set V_1 since l is an odd number. Since v_0 and v_l are not in the same subset, they can not be the same point. Thus this path is not a cycle. ■

By Theorem 1, it is clear that we have to generate a bipartite subgraph from the given state adjacency graph so that it can be embedded into an n-cube.

Whenever a state adjacency graph violates Theorem 1, some edges have to be removed. This is done as follows. First, we check the number of edges connected to any node. Let the number of nodes adjacent to node i be a_i. Whenever $a_i > n$ is true for node i, the $a_i - n$ edges whose weights are smaller are removed. Next we break odd-length cycles. Whenever an odd-length cycle is found, the edge with least weight on the cycle is removed.

During this procedure, we may remove more edges than we really have to. There are two situations in which the recovery of a removed edge is possible. Suppose an odd-length cycle is found and we decide to remove edge (s_i, s_j). Now if in the original state adjacency graph the degree of either s_i or s_j is greater than n, some edges must have been deleted already. Since edge (s_i, s_j) has to be removed, one of the edges that have been removed earlier can be recovered. On the other hand, it is possible that edge (s_i, s_j) is also contained in another odd-length cycle, which has been broken earlier by removing another edge. Since edge (s_i, s_j) must be removed anyway, the edge removed earlier can be recovered, too. This recovery mechanism gives us a better adjacency relation and may potentially lead to a better result.

3.3. Embedding a Bipartite Adjacency Table

Up to this stage we have obtained an m-BAT (the original STG has m states) in which there are no odd-length cycles. However, it is still possible that the bipartite graph can not be embedded into an n-cube. First we have to check whether a BAT can be embedded into an n-BRT. If the BAT cannot be embedded, some extra links are removed. Then the resulting BAT is embedded.

3.3.1. Identification of Mappable BAT

The following theorems are used to check whether an m-BAT is mappable.

Theorem 2 [11]: An m-BAT is unmappable if either
1) each row or column of the m-BAT contains more than n links, or
2) the link set of the m-BAT includes $\{(S_i, s_j) \mid i = 1, 2, 3 \text{ and } j = 1, 2\}$ or $\{(S_i, s_j) \mid i = 1, 2 \text{ and } j = 1, 2, 3\}$.

The first condition has been checked when we generate the bipartite graph, so we only have to check the second condition. The second condition essentially says that any two nodes in the same set (either X_c or X_r) can not have common links to three nodes in the other set. For example, the BAT shown in Fig. 7 is unmappable. Thus we will remove the least weighted edge from the set {(1,7), (1,8), (3,7), (3,8), (4,7), (4,8)}.

	6	7	8	9	10
1	0	1	1	0	0
2	0	0	0	0	1
3	0	1	1	0	0
4	0	1	1	0	0
5	0	0	0	0	1

Fig. 7. An example of unmappable BAT.

The following conditions are applicable to BATs in which there exist more than one 2-cube.

Definition 1 [11]: For the link (S_i, s_t) in a BAT, we define $\text{adj}(S_i, s_t) = \{(S_i, s_k) \mid s_k \in X_r \text{ and } s_k \neq s_t\} \cup \{(S_k, s_t) \mid S_k \in X_c \text{ and } S_k \neq S_i\}$, and $\langle \text{adj}(S_i), \text{adj}(s_t) \rangle = \{(S_m, s_n) \mid (S_m, s_n)$ is a member of the link set, $(S_m, s_n) \in \{\text{adj}(S_i, s_k) \cap \text{adj}(S_k, s_t)\}$, where $\text{adj}(S_i, s_k), \text{adj}(S_k, s_t) \in \text{adj}(S_i, s_t)\}$.

Theorem 3. [11]: Let a BAT contain the following links: (S_i, s_r), (S_i, s_t), (S_j, s_r), (S_j, s_t), (S_j, s_u), (S_k, s_t), and (S_k, s_u). The BAT is unmappable if $\langle \text{adj}(S_i), \text{adj}(s_u) \rangle \neq \emptyset$ or $\langle \text{adj}(S_k), \text{adj}(s_r) \rangle \neq \emptyset$.

The condition given in Theorem 3 is that two 2-cubes are contained in a 3-cube and they share a link (S_j, s_t). In other words, the two 2-cubes share one row and one column in the BRT. This condition is illustrated in Fig. 8, in which the seven circles are the

links listed in Theorem 3. When the condition is satisfied, then the BAT is mappable if both $\langle adj(S_i), adj(s_u)\rangle$ and $\langle adj(S_k), adj(s_r)\rangle$ do not contain any link other than the ones in the 3-cube.

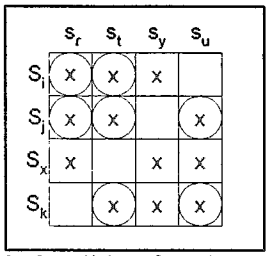

Fig. 8. Condition for Theorem 3.

Our way to check this condition works as follows. After having calculated $\langle adj(S_i), adj(s_u)\rangle$ and $\langle adj(S_k), adj(s_r)\rangle$, we check whether this is a 2-cube contained in a 3-cube. If those links are not contained in the same 3-cube, then the links are removed. Consider Fig. 8. If $\langle adj(S_i), adj(s_u)\rangle$ include (S_x, s_y) and the 2-cube $(S_x, s_k \mid S_y, s_u)$ does exist, then (S_x, s_y) is in the same 3-cube and thus does not have to be removed.

The above theorem deals with the condition in which two 2-cubes share one common row and column. In this paper we extend the concepts in [11]. Consider the situation in which two 2-cubes share only one row or only one column, as illustrated in Fig. 9. Now we have to check four sets $\langle adj(S_a), adj(s_g)\rangle$, $\langle adj(S_b), adj(s_g)\rangle$, $\langle adj(S_c), adj(s_f)\rangle$, and $\langle adj(S_d), adj(s_f)\rangle$. If any one of them is nonempty, the BAT is unmappable and the corresponding links have to be removed. However, if the resulting links become a 2-cube contained in rows S_a, S_b, S_c, S_d, then they are not removed.

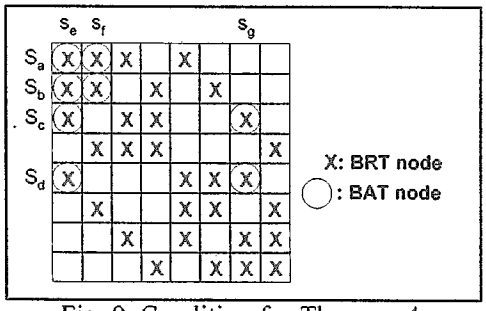

Fig. 9. Condition for Theorem 4.

Theorem 4: Suppose a BAT contains the following links: (S_a, s_e), (S_a, s_f), (S_b, s_e), (S_b, s_f), (S_c, s_e), (S_c, s_g), (S_d, s_e), (S_d, s_g). The BAT is unmappable if $\{S_v \mid (S_v, s_f)$ is in the BAT, and $S_v \neq S_a, S_v \neq S_b\} \cap \{S_w \mid (S_w, s_g)$ is in the BAT, and $S_w \neq S_c, S_w \neq S_d\} \neq \emptyset$. The same situation can be applied to columns in BAT.

Proof: In Fig. 10 we draw a sphere whose center is s_e and the radius is Hamming distance 1. S_a, S_b, S_c, and S_d are four nodes on the sphere. Again we draw four spheres centered at these four nodes, and the radius of each sphere is Hamming distance 1. Now the spheres centered at S_a and S_b intersect at two points s_e and s_f, while the spheres centered at S_c and S_d intersects at two points s_e and s_g. From the figure it is obvious that there are no nodes whose Hamming distance to s_f and s_g are both 1 except for those given in the Theorem. ■

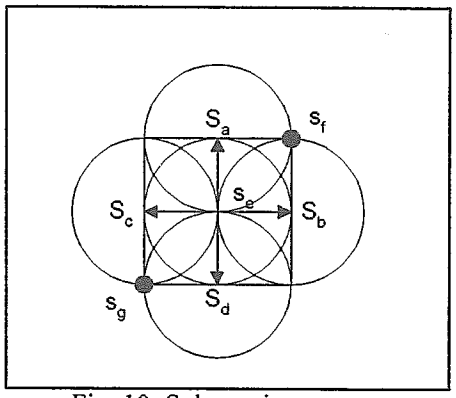

Fig. 10. Spheres in n-space

In most of the cases, the graph obtained so far can readily be embedded into an n-cube. However, since all the conditions we use to check for embeddable bipartite graph are only necessary conditions, it is still possible that the resulting graph can not be embedded. Even if the graph can be embedded, the process can be time-consuming since state embedding (as discussed below) is an NP-complete problem. Thus we employ an optional heuristic in the state assignment algorithm. We can assign a small threshold w_{th}, and all edges in G' whose weights are smaller than w_{th} are removed. A large threshold makes state embedding easier and faster; however, the switching activity of the resulting circuit is usually higher. If w_{th} is 0, we simply get the original exact algorithm.

The following algorithm is used to check whether a given bipartite graph can be embedded into an n-cube. If the graph cannot be embedded, some links are removed.

Algorithm : Get a mappable bipartite graph

```
{
    Remove links whose weights are less than w_th;
    Remove links that violate Theorem 1;
    Remove links that violet Theorem 2;
    Remove links that violate Theorems 3 and 4;
}
```

3.3.2. State Embedding

At this stage we have two tables: an n-BRT which is a representation of an n-cube, and an m-BAT which is a tabular form of the state adjacency graph. The state embedding problem here is to find a subgraph in the n-BRT such that the subgraph is isomorphic the given m-BAT. The subgraph isomorphism problem is known to be an NP-complete problem [13]. Thus we apply only a greedy algorithm to compare the given n-BRT and m-BAT. During the state embedding process, two neighboring states in G'' are given codes whose Hamming distance is one. If edge (s_i, s_j) exists in G' but is removed in G'', we will also try to keep the Hamming distance between $enc(s_i)$ and $enc(s_j)$ as small as possible. This will keep the final switching activity low.

4. Experimental Results

We have implemented the above algorithm into a program called 'STATE', which assigns minimum-length code to states in FSMs targeted for minimum switching power. In order to compare it with the previously published method, we adopt the same set of MCNC FSM benchmarks circuits listed in [6] except the circuit *tbk*. The reason we do not include *tbk* is that, for this particular circuit, the number of cubes given by [6] is only 60% of the circuit synthesized by NOVA[14]. Since NOVA is a state-assignment program targeted for area efficiency, this result seems rather exceptional and thus we exclude it in our comparison.

The experimental results are given in Table 1 and 2. For the fourteen circuits listed in Table 1, our results for NOVA are identical to those in [6] in both the number of cubes and switching activity. Thus, for the method proposed by Hong *et al.* and NOVA, we only list the switching activity for comparison. The number of cubes obtained by these methods can be found in [6]. The lower bound on switching activity listed in the table is obtained by assuming all state transitions, except for self-loops, involving only 1 bit change. Compared with NOVA, STATE gives a 46.1% reduction on switching activity; and there is a 5.0% reduction on switching activity compared with the method by Hong *et al.* Among the 14 circuits, our method gives better results in 8 cases, while in 3 cases the method by *Hong et al.* is better. The results are identical for the other 3 circuits. In Table 2, however, our results for NOVA are different to those given in [6] in terms of switching activity, although the results for cubes are the same. These circuits are either incompletely specified machines, or FSMs with a resetting input. The difference may be caused by the way transition probability being computed. For these five circuits, STATE gives a 46.9% reduction on switching activity compared with NOVA. This result is consistent with those in Table 1.

In terms of size of the synthesized circuit, the results given by STATE are close to those in [6]. For the 19 circuits, our method requires 920 cubes in all, while Hong's method requires 907 cubes. The number obtained by NOVA is 825. Thus our method requires 11.5% of extra cubes while Hong's method increases the number of cubes by 9.9%.

5. Conclusions

In this paper we present a state a state assignment algorithm for FSM targeted for low switching power. This method can reduce more 46% of switching activity associated with state transitions at the cost of about 11.5% extra area. Compared with previous results [6], this method gives lower switching activity and costs a little more area.

REFERENCES

[1] Z. J. Lemnios and K. J. Gabriel, "Low-Power Electronics," *IEEE Design & Test of Computers*, pp. 8-13, Winter 1994.

[2] Z. Chen, J. Scott, J. Burr, and J. D. Plummer, "CMOS technology scaling for low voltage low power applications," *IEEE Symp. om Low Power Electronics*, pp. 56-57, Oct. 1994.

[3] C. Tsui, M. Pedram, and A. Despain, "Technology decomposition and mapping targeting low power dissipation," in *Proc. 30th Design Automation Conf.*, pp. 68-73, 1993.

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

[5] M. Alidina, J. Monteiro, S. Devadas, A. Ghosh, and M. Papaefthymiou, "Precomputation-based sequential logic optimization for low power," *IEEE Trans. VLSI Systems*, vol. 2, no. 4, pp. 426-436, Dec. 1994.

[6] S. K. Hong, I. C. Park, S. H. Hwang and C. M. Kyung, "State asignment in finite state machines for minimal switching power consumption," *Electronics Letter*, vol. 30, no. 8, pp. 627-629, 1994.

[7] G. D. Hachtel *et. al*, "Re-encoding sequential

circuits to reduce power dissipation," in *Proc. Intl. Workshop Low Power Design*, pp. 69-74.

[8] V. Veeramachaneni, A. Tyagi, and S. Rajgopal, "Re-encoding for low power assignment of FSMs," in *Proc. Intl. Symp. Low Power Design*, pp. 173-178, 1995.

[9] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design: A System Perspective*, 2nd Ed., Addison-Wesley, 1992.

[10] F. N. Najm, "Transition density: a new measure of activity in digital circuits," *IEEE Trans. CAD*, vol. 12, no. 2, pp. 310-323, Feb. 1993.

[11] J. W. Kang, C. L. Wey, and P. D. Fisher, "Application of bipartite graphs for achieving race-free state assignments," *IEEE Trans. Comput.*, vol. 44, no. 8, pp. 1002-1011, Aug. 1995.

[12] C. L. Su, C. Y. Tsui, and A. M. Despain, "Saving power in the control path of embedded processor," *IEEE Design & Test of Computers*, pp. 24-30, Winter 1994.

[13] M. R. Garey and D. S. Johnson, *Computer and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, 1979.

[14] T. Villa and Sangiovanni-Vincentelli, "NOVA: state assignment of finite state machines for optimal two-level logic implementation," *IEEE Trans. CAD*, vol. no. 9., pp. 905-924, Sep. 1990.

Table 1. Experimental results (I)

		ST ATE		Hong *et al.*	NOVA	Lower Bound
	n-cube	SA	Cubes	SA	SA	SA
bbara	4	0.279	26	0.295	0.459	0.223
bbsse	4	0.776	31	0.856	1.495	0.673
bbtas	3	0.443	9	0.561	0.809	0.443
cse	4	0.239	48	0.292	0.604	0.228
donfile	5	1.125	45	1.083	1.750	0.750
ex6	3	1.009	30	1.009	1.530	0.803
keyb	5	0.556	58	0.647	1.469	0.549
modulo12	4	0.500	12	0.583	1.000	0.500
planet	6	0.984	103	1.153	2.833	0.960
s1	5	1.175	91	1.131	1.698	0.731
sand	5	0.610	109	0.604	1.083	0.491
shiftreg	3	1.000	10	1.000	1.500	1.000
styr	5	0.553	99	0.578	1.276	0.511
tav	2	1.000	10	1.000	1.500	1.000
Total		10.249	681	10.792	19.006	8.862

Table 2. Experimental results (II)

		ST ATE		NOVA	Lower Bound
	n-cube	SA	Cubes	SA	SA
ex1	5	1.135	47	1.938	0.809
ex4	4	0.957	18	2.261	0.870
opus	4	0.712	17	1.449	0.650
scf	7	0.845	147	1.787	0.750
train11	4	0.714	10	0.786	0.571
Total		4.363	239	8.221	3.650