Cube-Embedding Based State Encoding for Low Power Design

De-Sheng Chen
Electronics Research & Service Organization
Industrial Technology Research Institute
HsinChu, Taiwan
email: dschen@erso.itri.org.tw

Majid Sarrafzadeh
Department of ECE
Northwestern University
Evanston, IL 60208
email: majid@eecs.nwu.edu

Abstract – In this paper we consider the problem of minimizing power consumption of a sequential circuit using low power state encoding. One of the previously published results is based on recursive matching. In general, a matched pair can be considered as a 1-cube being embedded in a hypercube. We generalize this idea of 1-cube embedding and propose a new encoding algorithm based on r-cube embedding. We then present an efficient 2-cube embedding based state encoding approach for low power design. It considers both Hamming distance and the complexity of logic function (by estimation). Experimental results show that this approach is competitive to other existed techniques.

I. Introduction

The increasing demand of personal electronic devices powered by batteries has placed new challenges in digital chip design. For applications such as notebook computers and cellular phones, low-power dissipation is one of the major concerns. The continued push for higher level of integration, device density and operating frequency also prompt power dissipation to become an important design issue.

Power efficiency of a silicon chip has been investigated at various level of design phases. For instance, architectural and algorithmic transformations can trade off throughput, area, and power dissipation; different logic optimization methods can result in different power dissipation of combinational logics; and some conventional layout techniques are modified to improve the power dissipation [2, 4, 9]. In all these works, minimizing switching activity of a design is shown to have great impact on reducing power dissipation. Especially in CMOS circuits, the power dissipation is directly related to the extent of switching activity of the internal nodes in the circuit.

Minimizing power dissipation of a state machine is an important step in reducing overall chip power because in most digital systems, state machines often run at full clock speed. Also, state machine design offers better opportunities for power improvement than the other parts of a digital system, such as data paths, which are difficult to optimize for power without changing the functional specification. Previous work on low power state encoding has been addressed by several researchers [7, 8]. In general, a weighted graph $G_w = (V, E_w)$, where V corresponds to the states of a finite state machine and the edge carries the weight corresponding to the estimated encoding affinity, i.e., switching activity and area cost, between states, is first constructed. Then different encoding methods are applied to G_w to obtain low power encodings. Recently, a new technique that considers the switching activity and area cost interactively during a simulated annealing process was proposed in [11].

In [7], a matching based state encoding algorithm that recursively combines 1-cubes into a bigger cube was presented. Though this approach mostly generates the encodings targeted low power consumption, however, its encoding scheme strongly depends on local relationship between states and, as a result, better encodings might be missed during the process. In this paper, we first generalize the idea of 1-cube embedding and propose a new encoding algorithm based on r-cube embedding. An efficient 2-cube embedding based state encoding approach is then presented for low power design.

II. Preliminaries

A. Power Dissipation and State Machines

The average power consumption by a CMOS gate is given by

$$P_{average} = 0.5 \frac{V_{dd}^2}{T_{cycle}} C_{load} E(switching). \qquad (1)$$

where V_{dd} is the supply voltage, T_{cycle} is the global clock period, C_{load} is the load capacitance, and $E(switching)$ is the expected number of gate output transitions per clock cycle.

In hardware implementation, the architecture of a state machine is shown in Figure 1. Each state is encoded with a binary value represented in the state register. The combinational logic generates the next state encoding and the outputs of the machine from the current state and inputs of the machine. A machine with n states requires a state register with at least $\lceil log_2\ n \rceil$ bits to be properly encoded.

Many researches have been done to find the state encoding of a FSM that targets toward minimizing the area of a

ASP-DAC '97
0-89791-851-7$5.00

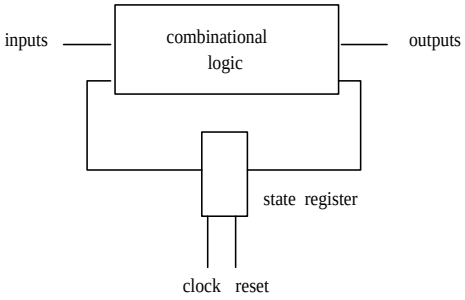


Figure 1: A hardware implementation of a finite state machine.

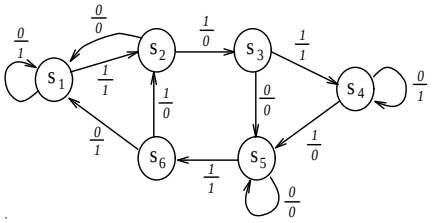


Figure 2: a state transition graph

sequential circuit after logic optimization [5, 12]. Usually, smaller area means less power consumption. However, it is not always true for CMOS design because the power dissipation of a CMOS chip is dominated by the *dynamic power*, which is proportional to the sum of the product of load capacitance and switching activity of each gate. This makes the problem of state encoding for low power harder because we are not only to consider area, but also switching activity.

A state machine is conveniently represented by a state transition graph where the nodes represent the states and a directed edge $\overline{s_i s_j}$, labeled with inputs and outputs, represents the transition from state s_i to s_j. Consider the state transition graph in Figure 2, state s_2 has two outgoing edges to state s_1 and s_3. If s_2 is encoded as '000' and the next state s_3 is encoded as '100', then, when there is a transition from s_2 to s_3, only one flip-flop of the state register will switch from a logic value of 0 to 1. (The number of bit toggles between two encodings is also known as the *Hamming distance* between them, i.e., $h(000, 100) = 1$). However, if s_3 is encoded as '111' instead of '100', then all three flip-flops will transit from 0 to 1. By reducing the number of bit toggles of the state register in each state transition, we possibly reduce the switching activity in the combinational logic which implements the next state and output functions.

The *state transition probability* p_{s_i,s_j}, between two states s_i and s_j, is defined as the probability that a transition occurred from s_i to s_j. The *global state transition probability* between two states s_i and s_j is defined as:

$$P(s_i, s_j) = p_{s_i,s_j} + p_{s_j,s_i}$$

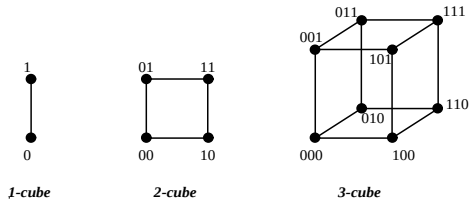


Figure 3: r-cube for $r = 1, 2, 3$.

$P(s_i, s_j)$ is the probability that there is a transition from state s_i to state s_j or vice versa.

In general, the relationship between states is indicated by a weighted graph $G_w = (V, E_w)$, where V corresponds to the states of a state machine and the weight on an edge corresponds to the estimated encoding affinity between states. The weight between two states s_i and s_j can be modeled as a linear combination of switching activity and area cost [3]:

$$w(s_i, s_j) = \lambda P(s_i, s_j) + A(s_i, s_j) \quad (2)$$

$P(s_i, s_j)$ is the global transition probability and is referred as *switching cost*. For those pairs of states that do not have edges connected in the state transition graph, their global transition probabilities are zero. $A(s_i, s_j)$ is the estimated area saving between states s_i and s_j, and is referred as *area cost*. λ is a function to trade off the importance of switching and area cost in determining the low power encoding.

B. Properties of A Hypercube

An r-cube, Q_r, has 2^r nodes and $r2^{r-1}$ edges. Each node is labeled with an r-bit binary number, and two nodes are linked with an edge if and only if their binary numbers differ in only one bit. Therefore, each node has r neighbors, one for each bit position. Figure 3 shows the r-cubes for $r \leq 3$ with appropriate binary numbers.

An important property of a hypercube is that it can be constructed recursively from smaller cubes. For instance, consider two $(r-1)$-cubes whose nodes are numbered from 0 to 2^{r-1}. An r-cube can be constructed from these two $(r-1)$-cubes by connecting every node of one $(r-1)$-cube to the node of the other $(r-1)$-cube having the same number. Thus to number the nodes of the r-cube, we can add 0 to the nodes of one $(r-1)$-cube as $0a_1a_2...a_{r-1}$ and 1 to those of the other $(r-1)$-cube as $1a_1a_2...a_{r-1}$, where $a_1a_2...a_{r-1}$ is a binary number representing the two similar nodes of the $(r-1)$-cubes. An example of constructing a 4-cube from two 3-cubes is shown in Figure 4.

III. Cube-Embedding Based State Encoding

Consider a weighted graph $G_w = (V, E_w)$. We want to find the state encoding such that the states joined by a heavily weighted edge are encoded as close as possible in terms of Hamming distance. The problem can be formulated as a *hypercube embedding* problem, and the goal is to minimize

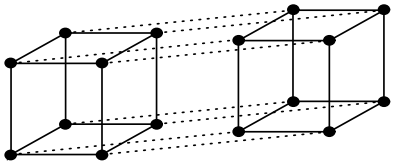
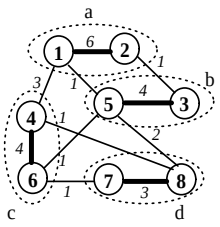
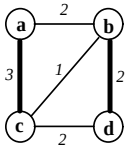


Figure 4: Construction of a 4-cube from two 3-cubes.



(a)

state	encoding	state	encoding
1	0	2	1
6	0	4	1
3	0	5	1
7	0	8	1



(b)

state	encoding	state	encoding
1	00	2	01
6	10	4	11
3	00	5	01
7	10	8	11

Figure 5: An example of 1-cube embedding.

the following cost function:

$$P_{total} = \sum_{(s_i,s_j)\in E_w} w(s_i,s_j)h(enc(s_i),enc(s_j))$$

, where $w(s_i, s_j)$ is the estimated encoding affinity between states s_i and s_j, and $h(enc(s_i), enc(s_j))$ is the Hamming distance between the encodings of these two states.

A. 1-cube Embedding

In [7], a recursive 1-cube embedding approach is presented. At each iteration, they find a maximum weighted matching M^i of a graph G^i_w. Then a "1" is assigned to the i-th bit of the binary number of one of each matched pair, and a "0" to the other. After that, they generate a new weighted graph $G^{i+1}_w = (V^{i+1}, E^{i+1}_w)$, where the new vertices are the pairs of matched states, and the weight of an edge is the sum of the edges' weights between the states which form the new vertices in V^{i+1}.

Example. In Figure 5(a), the edges of maximum matching are marked with bold lines and the encoding of each state is shown at its righthand side. The new generated graph and the associated encodings are shown in Figure 5(b).

After $log|V|$-th matching stage of this recursion, a super vertex corresponding to $|V|$ original states is obtained. A state encoding obtained by the use of this approach is shown in Figure 6(a) and its P_{total} is 34. However, we

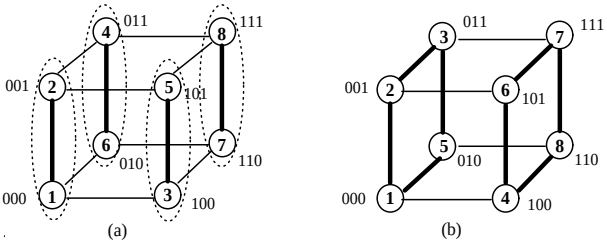


Figure 6: (a) the encoding by 1-cube embedding, and (b) a better encoding.

notice that better results can be obtained if the weights between states are considered more globally. For example in Figure 6(a), if we switch the position of state 4 and state 6, the total cost can be reduced by 2. Furthermore, an even better encoding with P_{total} equals to 30 is shown in Figure 6(b), which is the combination of two 2-cubes,. {1, 2, 3, 5} and {4, 6, 7, 8}. This observation motivates our interest in developing a new r-cube embedding approach that considers the relationship among states more globally.

B. r-cube Embedding

Here we propose an encoding strategy based on r-cube embedding, where $r \geq 2$. The algorithm consists of three major steps. First, a set of heavily weighted r-cubes, S_r, is selected from the graph G_w. The selection is done in a greedy way as follows: we first find the maximal weighted r-cube of the graph. Then remove the r-cube and the associated edges from the graph. Repeat the process on the remaining graph until no more r-cube exists. After the above step, there may exist some vertices which do not belong to S_r. So, the next step is to group these vertices to form a set of clusters, S_c, such that the size of each cluster is at most 2^r. This is done by applying the maximum matching algorithm r times. In the third step, we generate a cluster graph $G^r_w = (V^r, E^r_w)$, where $V^r = \{S_r\} \cup \{S_c\}$, and the weight of an edge is the sum of the edges' weights between V^r. Then apply the maximum weighted matching to the cluster graph, and each pair of matched nodes is collapsed into a single node. The process is repeated until the cluster graph contains only one node. The output from the third step is a *cube-clustering tree*, which it is a rooted binary tree and its leaf nodes are the vertices of G_w.

After the cube-clustering tree is constructed, we need to encode each vertex of G_w. We label the two edges incident from each internal branch nodes with 0 and 1, then the encoding of each leaf node is the sequence of 0, 1 labels of the edges in the path from the root to that leaf.

procedure *Cube-Embedding*(G_w, r)
begin
  $G = G_w$;
  $S_r = \phi$;
  while G contains an r-cube

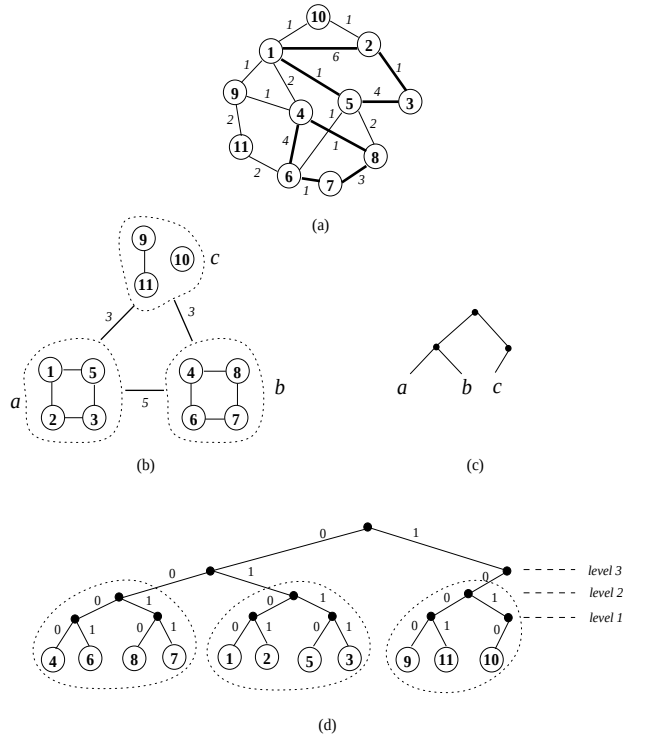


Figure 7: An example to illustrate the algorithm.

begin
  r_{max} = the maximal weighted r-cube of G;
  $S_r = S_r \cup \{r_{max}\}$;
  remove vertices and edges associated with r_{max} from G;
end
if G is not an empty graph, group its vertices to a set of clusters, such that the size of each cluster is at most 2^r;
generate the cluster graph $G_w^r = (V^r, E_w^r)$;
apply maximum matching to G_w^r recursively, output the cube-clustering tree when G_w^r contains only one node;
encode the cube-clustering tree;
end.

Example. In Figure 7(a), two 2-cubes {1, 2, 3, 5} and {4, 6, 7, 8} are detected and marked with bold lines. The cluster graph is shown in Figure 7(b). The resulting cube-clustering tree is shown in Figure 7(c). The ecnoding of the vertices is shown in Figure 7(d).

However, in general, the task of finding all r-cubes of a graph is computationally expensive, even when r is 2. Therefore, in the following we will develop an algorithm to find 2-cubes of a graph and use it in our 2-cube embedding based state encoding approach for low power design.

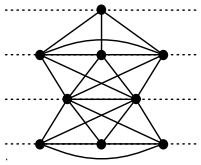


Figure 8: An example of a complete 4-level graph.

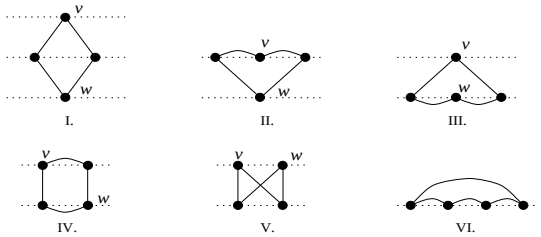


Figure 9: patterns of a 2-cube.

IV. Finding 2-Cube Subgraphs of a Graph

Let G be a level graph with k levels. G is called a *complete k-level graph*, if the vertices in any two adjacent levels form a complete graph. For example, Figure 8 is a complete 4-level graph. In the following, we will propose an algorithm to find all 2-cubes of a complete k-level graph. The algorithm will be used in finding all 2-cubes of a general graph.

If vertices $u_1, u_2, ..., u_l$ ($l \geq 2$) are all adjacent to two common vertices v and w, then any quadruple (v, u_i, w, u_j), $1 \leq i < j \leq l$, forms a 2-cube. However, even in a planar graph, there may exist $O(n^2)$ 2-cubes, where n is the number of nodes of the graph. Thus, instead of listing these 2-cubes individually, we use the representation developed in [1] to list a group of 2-cubes by a triple $(v, w, \{u_1, u_2, ..., u_l\})$.

The patterns of a 2-cube, classified as Type I $\sim$ Type VI, are listed in Figure 9. Now consider V_{i-1}, V_i, and V_{i+1} of a complete k-level graph, where $1 < i < k$ (V_i is the set of vertices at level i). If we pick a vertex v from V_{i-1} and a vertex w from V_{i+1}, then for any two vertices of V_i these four vertices form a Type I 2-cube. These 2-cubes are indicated by a triple (v, w, V_i). Now consider V_{i-1} and V_i only. If we choose a vertex v from V_{i-1} and a vertex w from V_i, then for any two vertices other than v in V_{i-1}, these four vertices form a Type II 2-cube. These 2-cubes are indicated as $(v, w, V_{i-1} - \{v\})$. Type III 2-cubes are obtained similarly.

For a Type IV 2-cube, it has two vertices from V_{i-1} and two vertices from V_i. Here we use a quadruple $(v, w, V_{i-1} - \{v\}, V_i - \{w\})$ to represent the Type IV 2-cubes that are detected by selecting a vertex v from V_{i-1}, a vertex w from V_i, and one vertex from $V_{i-1} - \{v\}$ and $V_i - \{w\}$ each. For a Type V 2-cube, it can be formed by choosing

two vertices v and w from V_{i-1} and two other vertices from V_i. These 2-cubes can be represented by (v, w, V_i). As for the Type VI 2-cube, it can be formed by selecting four vertices from the same level.

The algorithm to find all 2-cubes of a complete k-level graph G is thus as follows:

procedure *Find-2-cube(G)*

begin
  k = levels of G;
  /* report Type I 2-cubes */
  for $i = 2$ **to** $k - 1$ **do**
    for each $v \in V_{i-1}$ and $w \in V_{i+1}$ **do** report (v, w, V_i);
  for $i = 2$ **to** k **do**
  begin
    /* report Type II, III, and IV 2-cubes */
    for each $v \in V_{i-1}$ and $w \in V_i$ **do**
    begin
      report triple $(v, w, V_{i-1} - \{v\})$;
      report triple $(v, w, V_i - \{w\})$;
      report quadruple $(v, w, V_{i-1} - \{v\}, V_i - \{w\})$;
    end;
    /* report Type V 2-cubes */
    for each $u, v \in V_{i-1}$ **do** report triple $(u, v, , V_i)$;
    /* report Type VI 2-cubes */
    for each $v, w \in V_i$ and $v \neq w$ **do** report $(v, w, V_i - \{v, w\})$;
  end;
end.

For a general graph G, we will apply Breadth First Search to it to obtain a new level graph G'. Assume G' has k levels. Since a graph with k levels is not always a complete k-level graph, there exists some missing edges in between or within levels. We have to remove those 2-cubes that contain the missing edges.

V. A Modified 2-cube Embedding Based State Encoding Approach

In our application, the generated weighted graph is a complete graph. Since an edge with light weight means the affinity between these two states is weak, we will remove this kind of edges so as to facilitate the 2-cubes finding process. Therefore, the complete weighted graph will be transformed to a new graph by removing the edges with weights smaller than a value α, for instance, α is equal to the average weight.

After applying the 2-cube embedding algorithm to the new graph, a cube-clustering tree is established, and the encoding of each leaf is the sequence of 0, 1 labels of the edges in the path from the root to that leaf. That is, the encoding of a leaf is determined by its position in the tree. Since in the embedding process, we randomly put one of the matched states or matched clusters to the left and the other to the right, a postprocessing is needed to fine tune the cube-clustering tree to get a better tree.

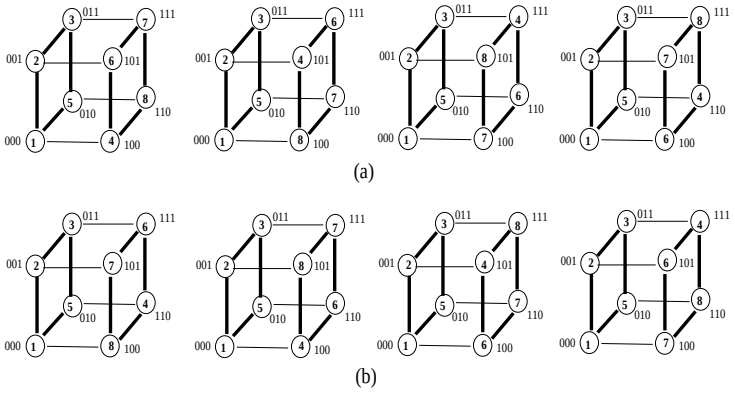


Figure 10: Different combinations of 2-cubes.

A postprocessing to rearrange the positions of some leaves in a cube-clustering tree is as follows: first, for each *level 2* internal node, if it contains less than four leaves, find the optimal tree of them. Then for each *level 3* internal node, find the best tree by rotating and flipping its two subtrees, i.e., two 2-cubes.

Example. In Figure 10, different combination of 2-cubes are shown, by rotating and flipping one of the 2-cube {4, 6, 7, 8}.

After the encoding of a FSM is found, the next step is to perform logic minimization to produce a network implementation. However, the logic minimization approaches, i.e., ESPRESSO and the multi-level logic minimization procedures in SIS [10], usually generate different results, even for two different encodings with the same total cost. Instead of reporting just one encoding at the end of the process, we shall output more encodings (with close total costs) and then choose the one with the lowest power dissipation implementation. The following heuristic is to gererate a reasonable number of encodings with low cost.

For a level 3 internal node of a cube-clustering tree, if we switch its left subtree with its right subtree, the local cost change is zero. Since there are $\lceil n/8 \rceil$ such level 3 internal nodes, where n is the total number of leaves, therefore, we can generate $2^{\lceil n/8 \rceil}$ different encodings by switching two 2-cubes of each level 3 node, respectively.

VI. Experimental Results

We have implemented the modified 2-cube embedding state encoding algorithm and applied it to a number of MCNC FSM benchmarks. For each benchmark, we calculate the state transition probabilities by simulating the FSM over a large number of randomly generated primary inputs. The area cost comes from the fanout-oriented cost function of MUSTANG [5] with normalization. The λ

Ckt	#st	NOVA		λ_w=0.2, λ_b=0.01					
				α=0.5 avg		α=1.0 avg		α=3.0 avg	
		#lit	pow	#lit	pow	#lit	pow	#lit	pow
s386	13	197	978	203	862	172	746	194	771
sse	16	200	927	187	738	186	759	183	743
keyb	19	593	2108	510	1637	416	1373	460	1464
ex1	20	663	2501	631	1916	571	1781	571	1781
dk16	27	401	2363	432	2318	406	2133	449	2259
sand	32	1243	5966	951	3636	925	3525	970	3540
tbk	32	1353	5588	688	2740	677	2608	832	2935
planet	48	1405	9128	1402	4969	1395	4692	1416	4742
s1494	48	1387	6259	1183	3289	1019	2333	1019	2333

Table 1: Comparsion of power consumption and number of literals on benchmark examples with λ_w = 0.2 and λ_b = 0.01.

varable, as seen in Equation (2), is defined as follows [3]:

$$\lambda = (2^{\lceil log_2 n \rceil} - n)\lambda_w + \lambda_b$$

where n is the number of states of a FSM. λ_w is a constant that emphasizes the importance of switching cost as $2^{\lceil log_2 n \rceil} - n$ increases. λ_b is a constant that addresses the switching cost if the size of host hypercube is equal to n.

The power consumption is measured in μW, using the power estimator developed by Ghosh et al [6], assuming a clock frequency of 20MHz and 5V power supply. In the following experiments, the generated encodings are written in *blif* format, synthesized using ESPRESSO, and then calculate the dissipated power. There is no specific library to be used because a unit delay model in calculating the power dissipation is applied.

The first experiment is to compare the encodings obtained by our modified 2-cube embedding algorithm to those obtained by NOVA [12], a state encoding program targeting minimum area. The results are shown in Table 1. The values in columns "#lit" denote the number of literals in factored form. For each benchmark, the best low power consumption encoding is selected and its associated power consumption and the number of literals are reported. Different α values are tested and the parameters, λ_w and λ_b, for calculating λ are set to 0.2 and 0.01, respectively. As can be seen, our algorithm outperforms NOVA in all cases.

The second experiment is to compare the modified 2-cube embedding algorithm to the maximum matching based encoding algorithm (MMBE) by [7] and the LPSA algorithm by [11]. The encodings of MMBE are obtained by our program, assuming the given weighted graph contains no 2-cubes and thus recursively apply maximum matching to it. The encodings of LPSA are provided by the authors of [11]. Table 2 summaries the results. The values in columns "2-cube" are the best results from Table 1. In general the modified 2-cube embedding algorithm produces better results than MMBE, and in four of six cases, it is better than LPSA. The result shows our approach is very competitive to other existed techniques.

Ckt	MMBE	LPSA	2-cube
sse	743	827	738
keyb	1642	1133	1373
ex1	1926	1692	1781
dk16	2540	2200	2133
sand	3992	4543	3525
planet	5826	5699	4692

Table 2: Comparsion of power consumption of MMBE, LPSA, and 2-cube embedding.

References

[1] N. Chiba and T. Nishizeki, "Arboricity and Subgraph Listing Algorithms", *SIAM Journal on Computing*, pp. 210–223, February, 1985.

[2] A.P. Chandrakashan, S. Sheng, and R.W. Brodersen, "Low Power CMOS Digital Design", *IEEE Trans. on Solid State Circuits*, Vol. 27, No. 4, pp. 473–483, April. 1992.

[3] D. S. Chen, M. Sarrafzadeh, and K. H. Yeap, "State Encoding of of Finite State Machines for Low Power Design", *ISCAS95*, May, pp. 2309–2312.

[4] K.Y. Chao and D.F. Wong, "Low Power Considerations in Floorplan Design", *Internationl Workshop on Low Power Design*, pp. 45–50, April, 1994.

[5] S. Devadas, H. Ma, A. R. Newton, and A. Sangiovanni-Vincentelli, "MUSTANG: State Assignment of Finite State Machines Targeting Multilevel Logic Implementations," *IEEE Transactions on Computer-Aided Design*, Vol. 7, No. 12, pp. 1290–1299, Dec. 1988.

[6] A. Ghost, S. Devadas, K. Keutzer, and J. White, "Estimation of Average Switching Activity in Combinational and Sequential Circuits", *Proceedings of the IEEE Design Automation Conference*, pp. 253–259, June, 1992.

[7] G. D. Hachtel, M.H. Rica, A. Pardo, M. Poncino, and F. Somenzi, "Re-encoding Sequential Circuits to Reduce Power Dissipation", *Internationl Workshop on Low Power Design*, pp. 69–74, 1994.

[8] E. Olson and S.M. Kang, "Low-Power State Assignment for Finite State Machines", *Internationl Workshop on Low Power Design*, pp. 63–68, April, 1994.

[9] A. Shen, A. Ghosh, S. Devadas, and K. Keutzer, "On Average Power Dissipation and Random Pattern Testability of CMOS Combinational Logic Networks", *Proceedings of the IEEE International Conference on Computer Aided Design*, pp. 402–407, November, 1992.

[10] E.M. Sentovich, K.J. Singh, C. Moon, H. Savoj, R.K. Brayton, and A. Sangiovanni-Vincentelli, "Sequential Circuit Design Using Synthesis and Optimization", *Proceedings of the International Conference on Computer Design*, pp. 328–333, October, 1992.

[11] C. Tsui, M. Pedram, C. Chen, and A. Despain, "Low Power State Assignment Targeting Two- and Multi-level Logic Implementations", *Proceedings of the IEEE International Conference on Computer Aided Design*, pp. 82–87, Nov., 1994.

[12] T. Villa and A. Sangiovanni-Vincentelli, "NOVA: State Assignment of Finite State Machines for Optimal Two-Level Logic Implementations," *IEEE Transactions on Computer-Aided Design*, Vol. 9, No. 9, pp. 905–924, Sept. 1990.

Acknowledgment

The authors are grateful to Prof. Tsui for the kindness to provide their encodings used for the experiments.