A Fast State Assignment Procedure for Large FSMs*

Shihming Liu, Massoud Pedram and Alvin M. Despain
Department of Electrical Engineering - Systems
University of Southern California, Los Angeles, CA 90089

Abstract

This paper addresses the problem of state assignment for large Finite State Machines (FSM). This is an important problem in the high performance digital system design where added functionality often comes at the expense of a larger (and slower) FSM to control the system. We present a new method to solve the graph embedding problem which is the main step in the state assignment process. The basic idea is to place the state adjacency graph in a two-dimensional grid while minimizing the total wire length. The grid is then mapped into an n-dimensional hypercube while nearly preserving the adjacency relations that is with dilation at most 2. Experimental results are presented and compared with those of NOVA.

1. Introduction

Controller synthesis is an important problem in the high performance digital system design where added functionality often comes at the expense of a larger (and slower) FSM to control the system. State assignment which takes high-level specifications such as control flowgraphs, state transition tables or state transition graphs as inputs and produces binary codes for the states is an important step in the synthesis of controllers. Once binary codes have been assigned to the states, next-state and output equations are defined and subsequently optimized with classical logic minimization tools. State assignment must be therefore performed in a way that favors simplification of the next-state and output logic (implemented by PLA's, standard cells, ROM's, etc.).

With the rapid advances in circuit complexity and chip density, automatic synthesis tools have become a necessity for integrated circuit design. Simply, the FSM synthesis tools must be able to cope with the increasing complexity of the machines (up to 500 states, equations with more than 1000 product terms) [12]. The existing state assignment techniques are however either inefficient or inadequate for handling large finite state machines. This motivated us to develop a very fast state assignment procedure for handling large machines which is comparable in quality to more sophisticated and elaborate techniques. As PLA's are used extensively in the structured design of high-performance controllers, we will focus on two-level implementation as the target.

1.1. Previous Work

Approaches to state assignment can be divided into two broad classes. The first class derives from the classical *structure-theory.* Examples are [20] and [11] that use algebraic methods based on the partition theory and the reduced dependency criterion. The second class is based on the *graph-embedding* formulation since it formulates the problem as a weighted or constrained graph embedding-problem on Boolean hypercubes. This class is further divided into two categories. The first category [1][7] formulates the encoding problem as an *embedding* problem, where an adjacency graph defining adjacency relations between the states is mapped into the hypercube. The second category [6][22] applies symbolic minimization on an unencoded specification followed by extraction of a set of face (input) constraints from the minimized symbolic cover. These constraints are then enforced as much as possible during the encoding.

1.2. A Unified View

The concepts of adjacency graph and face constraints are somewhat similar. Both of them describe the desire to assign similar codes to a group of states. The main differences between the two categories of methods are twofold: (1)The order of logic minimization and encoding is different. (2) In the second category, distance relations are *required* to be satisfied during graph embedding. Of course, it may not be possible to satisfy all of them and some constraints (which are heuristically picked) may be relaxed. The first category is different in sense that the goal is to minimize a cost function rather than attempting to satisfy distance relations. The *partial constraint satisfaction* described in [9] can be considered as a mixed method.

If we consider the state assignment problem as placing states on an *n*-dimensional hypercube (*n*-cube), then we can combine the structure-theory-based and graph-embedding-based methods. From the structure theory point of view, each state variable y_i introduces a partition τ_i on the set of states, such that two states are in the same block of τ_i if and only if they are assigned the same value of y_i. Therefore, we can think of each state variable assignment as a hyperplane which cuts the *n*-cube. This is similar to a line separating cells in VLSI placement which cuts the chip area into two parts. On the other hand, from the graph-embedding point of view, the set of adjacency relations or input constraints act like the nets or clustering constraints in VLSI placement. As we can mix the partitioning and clustering procedures in cell placement [14], we can hybridize the structure-theory-based and graph-embedding-based methods.

The graph embedding approach for state assignment is attractive because it can be easily modified to optimize different objective functions. For example, in addition to the minimum area objective mentioned above, the graph embedding approach has been used in low power applications [17]. Furthermore, the formulation in [8] that requires some state values to be encoded with non-adjacent binary vectors to improve the testability can be easily modified to

*This research was supported in part by ARPA under contract no. DABT-63-93-C-0064 and by NSF's Young Investigator Award under contract no. MIP-9457392.

32nd ACM/IEEE Design Automation Conference ®

adjacency-relations. The state adjacency-graph can be thought of as a physical netlist which must be embedded on a Boolean hypercube so that the cost is minimum. Unfortunately, this problem is NP-complete.

In this paper, we propose a very fast yet effective method for solving this graph embedding problem. The basic idea is to place the adjacency graph in a two-dimensional array (grid) while minimizing the total wire length. The placement solution is then mapped into an *n*-dimensional hypercube while nearly preserving the adjacency relations. To obtain good state assignment results, one has to decide what kind of multi-pin net representation should be adopted during the two-dimensional placement, what kind of objective function should be used, and what kind of hypercube mapping should be applied. Some of the contributions of the present paper are exactly in answering these questions either theoretically or empirically. Indeed, we will show that the straight-forward choices are *not* the right choices for this application. Experimental results of this approach are very promising in terms of both the CPU time and the circuit area.

The rest of this paper is organized as follows. In section 2 an overview of our proposed approach is given. In section 3 the procedure for constructing the adjacency graph is presented. In section 4 the procedure for two-dimensional placement of the graph is described. In section 5 the procedure for mapping the placement solution into a hypercube of given dimensionality is presented. Experimental results and conclusions are given in section 6 and 7.

2. Outline of the Proposed Approach

We are given a weighted graph which describes the adjacency between various states or the desirability for giving a group of states similar codes. The larger the weights, the more desirable it becomes to give them adjacent codes. The basic idea is the following. We translate the problem of finding the best hypercube embedding into the following mathematical one. We define the distance d_{ij} between two vertices *i* and *j* of an *n*-cube to be the minimal number of edges that must be traversed to get from *i* to *j*. Then the *n*-cube can be coded so that the *n*-bit codes assigned to vertices *i* and *j* differ in only d_{ij} bits. We now wish to assign the nodes of the adjacency graph to the vertices of the *n*-cube so as to minimize the function:

$$Cost = \sum_{i,j} w_{ij} \cdot d_{\pi(i),\pi(j)} \qquad \text{(EQ 1)}$$

where w_{ij} is the weight of edge(i, j) in the adjacency graph, and $d_{\pi(i),\pi(j)}$ is the distance of $\pi(i)$ and $\pi(j)$ where $\pi(i)$ and $\pi(j)$ are coordinates of vertices in the hypercube to which *i* and *j* have been assigned. Solving this problem directly is difficult. Our strategy is to solve the problem by finding an embedding on a two-dimensional array and then mapping that solution to a hypercube.

Our approach *Hyper-Place* is composed of three procedures to handle the state assignment problem (Figure 1).

Procedure 1. An adjacency graph is formed based on the controller specification. In this graph, each node represents a state and there exists a weighted edge between two nodes if they should be given adjacent codes (i.e., codes that differ in only one bit).

Procedure 2. The adjacency graph is then placed on a two-dimensional array. The placement procedure is interleaving the global optimization step with a bi-partitioning step in order to minimize the wire length while avoiding congestion on the placement plane.

Procedure 3. The placement solution is mapped to a hypercube. The question of interest is the following: how can we map the nodes of any two-dimensional placement to the nodes of a hypercube, on a one-to-one basis, so that the relative distances between pairs of nodes in the placement solution is intact after the mapping? A promising result is that grid neighbors can be always mapped to hypercube nodes such that the worst case distance between grid-neighbors in the hypercube is 1 or 2 [4]. So, as long as we keep vertices adjacent in grids, we can nearly achieve our original goal which was to keep those vertices adjacent in the hypercube.

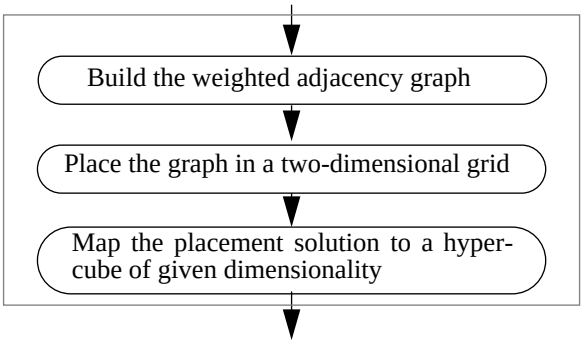


Figure 1: Main steps of Hyper-Place

3. Construction of the Adjacency Graph

An adjacency graph is formed from high-level FSM specification such as control flowgraphs, state transition tables or state transition graphs. In an adjacency graph, each node represents a state. Between any two nodes, there exists a weighted edge if these two nodes (states) want to be adjacent. Two nodes should be made adjacent if that would reduce the circuit area after state assignment and logic minimization [16].

We adopt the following scheme to generate the adjacency relations. We use symbolic logic minimization to group together states that are mapped by some input combination into the same next-state and assert the same output values. States in each group want to have adjacent codes. This kind of grouping is actually the combination of Type-I and Type-III adjacency heuristics in [16] and is also the state grouping which creates face constraints in [6].

3.1. Connection Model of Nodes

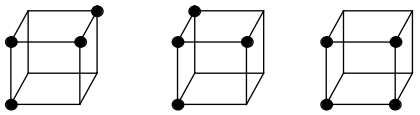
There are two ways to describe the connections among nodes representing states in the same state group. The *hyperedge model* forms one net (hyperedge) connecting all the nodes (states) in each group. That is, the adjacency graph is a hypergraph. The weight of each net is the number of nodes connected by that net. The *clique model* creates a clique on the nodes belonging to each group. In this case, all connections are two-terminal edges. Modern VLSI placement algorithms tend to use the hyperedge model as it more accurately reflects the connection strengths. In the remainder of this section, we will however show that the clique model is better for the state assignment application.

Definition: A*n m-subcube* of a hypercube *H* is an *m*-dimensional hypercube contained in *H*.

Definition: The *supercube* of a graph *G* embedded on a hypercube *H*, is the smallest *m*-subcube in *H* which contains *G*.

Definition: The supercube is *optimal* if *G* has *n* nodes and $m = \lceil \log_2 n \rceil$.

***Theorem* 3.1**: The minimum edge length embedding of an *N*-node clique on a hypercube *H* is always contained in an optimal supercube.
Proof. Omitted.
By Theorem 3.1, the group of nodes connected using a clique can be mapped to an optimal supercube. On the other hand, the group of nodes connected by a hyperedge does not necessarily map to an optimal supercube. That is, the clique model tends to generate bigger *group face*s [6] than the hyperedge model. Figure 2 illustrates the concepts described above. Four nodes belonging to a group are placed in a 3-dimensional hypercube. For the hyperedge model, all configurations (Figure 2(a), (b) and (c)) have the same connection cost. Any one of these configurations may be the placement solution that minimizes the total connection length. For the clique model, Figure 2(c) has the minimum total connection length. We note that Figure 2(c) is the configuration that minimizes the final circuit area after logic minimization.



	(a)	(b)	(c)
Hyperedge model:	3	3	3
Clique model:	10	9	8

Figure 2: Examples of total connection lengths using different models

4. Placement on a Two-Dimensional Grid

To minimize the cost function EQ1, we use a two-phase approach. First, we relax the grid constraints and place the nodes in a continuous plane. Then, this global placement is modified to map the nodes to the grid points [21][14].

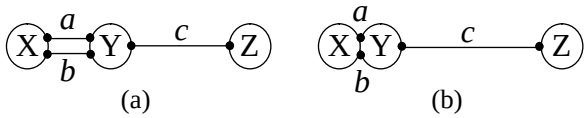
4.1. Global Placement

During global placement the total edge length among nodes is minimized while neglecting slot constraints. Global placement is interleaved with netlist partitioning. The set of nodes is recursively divided into smaller subsets while the placement area is dissected into subregions. The slicing procedure generates constraints for the next global placement step in subregions. These constraints aim at a better distribution of the nodes over the placement.

4.1.1 Influence of Various Distance Measures

The distance function $d_{\pi(i),\pi(j)}$ in EQ1 can be measured in various ways and this in turn affects the final state assignment solution.

Definition: An *l*-norm distance measure is $D(x_i,x_j) := |x_i - x_j|^l$. For $l = 1$, we have the Manhattan distance measure $|x_i - x_j|$. For $l = 2$, we have the Euclidean square distance measure $(x_i - x_j)^2$.
We will describe the impacts of a quadratic and a linear objective function on the placement [19] and discuss how that influences the state assignment. It is difficult to make strong statements about which objective function is better in the context of the state assignment problem. Examples are used to demonstrate the impacts of using different objective functions.
Figure 3 shows a placement scenario: two fixed nodes X, Y and a movable mode Z. They are connected by nets *a*, *b*, *c* with lengths l_a, l_b, l_c, respectively. Minimizing the quadratic objection $\Phi_q = l_a^2 + l_b^2 + l_c^2$ yields the placement in Figure 3(a) with $l_a = l_b = 1/2\ l_c$. The minimization of the linear function $\Phi_l = l_a + l_b + l_c$ results in the placement in Figure 3(b) with $l_a = l_b = 0$.



(a) quadratic objective function (b) linear objective function

Figure 3: Optimal placement for different objectives

It is generally observed that the quadratic objective function tends to make long nets (net *c* in Figure 3) shorter, at the expense of increasing the length of the short nets (nets *a* and *b* in Figure 3). In other words, the standard deviation of the net lengths is smaller for a quadratic objective function.
Modern VLSI cell placement tools tend to use a linear objective function [19]. This is because more tracks as well as more feedthroughs are needed in placement using quadratic objection. For example, wire segments *a*, *b* in Figure 3(a) may cause more tracks or feedthroughs than zero wire segments *a*, *b* in Figure 3(b). However, the costs of tracks and feedthroughs play no role in the state assignment application. On the other hand, the linear objective function tends to change the ratio of distances between one pair of nodes to another. For example, in Figure 3(b), the ratio of distance of X-Y to Y-Z is 0 which overstates the adjacency desirability of X-Y to Y-Z. Therefore, a quadratic objective function seems to reflect the actual adjacency demands more accurately than the linear objective function in our application. The statements made here are experimentally confirmed in section 6.

4.1.2 Quadratic Programming Formulation

The objective function of the global optimization step is then the weighted sum of the squared rubber band lengths of the edges of the given adjacency graph:

$$L = 1/2 \cdot \sum_{v_i v_j\,(i \neq j)} c_{ij} \cdot ((x_i - x_j)^2 + (y_i - y_j)^2)$$

where c_{ij} represents the total number of connections between vertex v_i and v_j. (x_i,y_i) and (x_j,y_j) represent the locations of v_i and v_j. The cost function can then be rewritten using matrix notation as follows [13]:

$$L(x, y) = x^T B x + y^T B y \qquad \text{(EQ 2)}$$

where x is a vector of the x-coordinates of the vertex locations and y is a vector of the y-coordinates. B is a symmetric matrix with $B = D - C$ where $C = [c_{ij}]$ is the connectivity matrix and D is a diagonal matrix with $d_{ii} = \sum c_{ij}$. It has

been shown that if the nodes (vertices) cannot be partitioned into disconnected subsets, then B is positive semi-definite [13]. That means the objective function is a convex function. This fact allows the calculation of a unique global optimum solution and plays an important role in our approach. In addition, B is almost always sparse for practical cases. This enables efficient numerical techniques to be applied to the matrix. Since the coordinate vectors x and y enter separately in the sum of two quadratic forms, we may consider each coordinate independently.

To avoid collapsing all nodes to the center of the placement region, in our implementation, we choose four nodes to be connected to four dummy nodes in the four corners of the placement region. These four dummy nodes will enable us to obtain a non-trivial placement solution. This corresponds to assigning distant codes to these four nodes and thus we pick 4 nodes that are not connected or are weakly connected.

The above scheme leads to a global placement as shown in Figure 4.

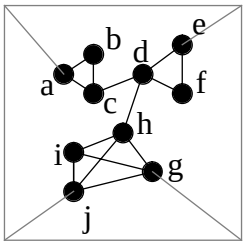


Figure 4: Global placement of adjacency graph

4.2. Mapping to Grid Positions

Since the global placement does not restrict the nodes to be placed on grid, a detailed placement step has to be performed. The goal of this step is to change the global placement as little as possible while mapping the nodes to the grid positions.

This mapping procedure is done by a minimum squared error linear assignment which maps all movable modules from the global placement to the legal positions simultaneously. The error to be minimized is

$$\sum_{i,j=1}^{m} \delta_{ij} [(x_i - m_j)^2 + (y_i - l_j)^2]$$ where x_i, y_i are the coordinates of the ith module and m_j,l_j are the coordinates of the jth legal slot. $\delta_{ij} \in \{0,1\}$ is the selection variable.

Figure 5 shows the final placement for the solution shown in Figure 4.

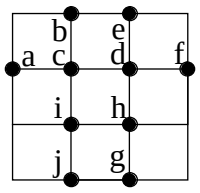


Figure 5: Final Placement

5. Mappings of Grids into Hypercubes

We want to embed the adjacency graph into hypercube. However, the problem of deciding whether a given graph is embeddable into any dimensioned hypercube is NP-complete [15] and the problem of embedding a given graph into a fixed-sized hypercube is also NP-complete [5]. We thus try to achieve the best partial embedding according to the cost function shown in EQ1. Therefore, this section addresses the following graph-mapping problem: given a placement solution on a two-dimensional grid and a hypercube with at least as many nodes as grid points, how can we assign the grid points to hypercube nodes so that the placement cost on the hypercube remains nearly the same as that on the grid? In both cases, the costs are calculated as in EQ1.

In the following discussion, we define the desired properties for the optimal mapping between grids and hypercubes. When these properties are absent we will describe conditions under which those sub-optimal properties can be achieved.

Let G and H denote graph G and hypercube H and d denote the distance function.

Definition: A map f: $G \rightarrow H$ is *distance-preserving* if $\forall a,b \in G$, $d(f(a),f(b)) = d(a,b)$. We also say that G is a *distance-preserving* subgraph of H.

Definition: A map f: $G \rightarrow H$ is *full* if a, b $\in G$ are adjacent if and only if $f(a)$, $f(b) \in H$ are adjacent. We also say that G is a *full* subgraph of H.

If we can make a distance-preserving mapping for grids into hypercubes, then we will have the same placement cost in hypercubes as we have in grids.

***Theorem* 5.1** [10]: If a graph G is a distance-preserving subgraph of some hypercube H, then G must be full.

Definition: The *optimal hypercube* of a two dimensional array is the smallest hypercube with at least as many nodes as the array.

***Theorem* 5.2**: A k-node two-dimensional array (grid) cannot be a distance-preserving subgraph of its optimal hypercube when $\lceil log_2 k \rceil > 4$.

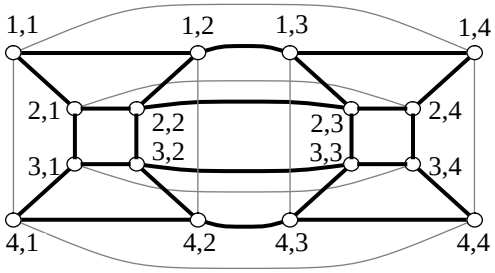
Proof. Suppose G is a k-node two dimensional array and H is a n-dimensional hypercube, where $n = \lceil log_2 k \rceil$, and there exists a map f: $G \rightarrow H$. Because $2^{n-1} < k \leq 2^n$, we can always find a vertex $v_x \in G$ and its corresponding vertex $f(v_x)$ in H where $f(v_x)$ is adjacent to n other vertices which are all mapping nodes of G. On the other hand, vertex v_x in a two dimensional array has at most 4 vertices which are adjacent to v_x. Because $n > 4$, we can always find at least one vertex $f(v_y) \in H$ which is adjacent to $f(v_x)$ but v_y is not adjacent to v_x in G. According to Theorem 1, G is not a distance-preserving subgraph of hypercube H. □

Unfortunately, by Theorem 5.2, we know that most of the two-dimensional grids are not distance-preserving subgraphs of their optimal hypercubes. Specifically, for those FSM's with more than 16 states, we need to find other mapping properties that could be applied to their sizes of grid.

Definition: The *dilation* of a grid-hypercube mapping is the worst case distance of grid-neighbors in the hypercube.

We want the dilation of the grid-hypercube mapping be minimum. For example, if dilation 1 mapping can be achieved, then we will have at most the same placement cost in hypercubes as we have in grid. An example of grid-hypercube mapping with dilation 1 is shown in Figure 6. A

number of researchers have studied this problem in parallel processing domain, with the following results. Over 61 percent of all two-dimensional grids can be embedded into their optimal hypercubes with a dilation 1 by using binary-reflected Gray codes [18]. Recently, Chan introduced an embedding strategy which makes all two-dimensional grids to be embeddable in their optimal hypercubes with at most dilation 2 [4].



Array edges are shown with solid lines, the unused hypercube edges are shown with dashed lines.

Figure 6: Embedding of a 4×4 grid in a 16-node hypercube

Note that we can always get grid-hypercube mappings with dilation 1 if we allow the bigger size of hypercubes.

Definition: The *expansion* of a grid-hypercube mapping is the ratio of the size (in number of nodes) of the embedding hypercube to the size of the optimal hypercube.

***Theorem* 5.3** [2]: The smallest hypercube that can embed a $d_1 \times d_2 \times ... \times d_k$ grid using unit dilation has dimension $\lceil log_2 d_1 \rceil + \lceil log_2 d_2 \rceil + ... + \lceil log_2 d_k \rceil$.

As a direct consequence of Theorem 5.3, we know that all two-dimensional grids can be embedded in hypercubes with dilation 1 using an expansion of at most 2. However, the trade-off is that when we use expansion 2 mappings, we need to add one more bit to encode the states. We adopt the approach in [4] to embed the grid in its optimal hypercube with dilation at most 2.

6. Experimental Results

To increase the flexibility of this assignment method, we distinguish the nodes into two sets, fixed and movable. Fixed nodes correspond to the states whose codings have been decided in advance. For example, in microprocessors some instructions (states) that invoke co-processors are often assigned fixed codes in advance. Their coordinates could be obtained easily by a reverse transformation of their codes. Also, we choose the ratio of the two dimensions of plane (grid) to be 1. This is based on the simple heuristic that the minimum cost of placement solution is often obtained on a square grid. In current implementation, we use GORDIAN [14] to generate the placement solution. Table 1 shows the statistics of examples tested. These examples include all of the large FSM examples (state number > 20 and product number > 200) in the MCNC logic synthesis and optimization benchmarks [23]. We also generated random examples to cover a more complete range of the size of circuits to be tested. In Table 2, we compare our results with NOVA [22]. In both cases, the code lengths were limited to the minimum number of bits and the number of cubes (product terms) after logic minimization [3] were compared. Basically, NOVA has three modes. The *exact* mode which produces the best results. However, its CPU run time is very high. For most of these examples, NOVA *exact* could not produce an answer because the run time (SUN SPARC 1+) was over one week and the process had to be terminated. Therefore, we compare with its *default* and *hybrid* modes.

Table 1: Statistics of benchmark examples

FSM Name	Inputs	Outputs	Products	States
s820	18	19	232	25
s832	18	19	245	25
s1494	8	19	250	48
s1488	8	19	251	48
x3643	3	3	368	64
x4322	4	2	383	32
d5326	5	6	384	32
d5322	5	2	427	32
x4326	4	6	449	32
d5324	5	4	511	32
d5323	5	3	730	32
x5322	5	2	767	32
x5321	5	1	809	32
x5324	5	4	920	32
s298	3	6	1096	218
x5643	5	3	1464	64
x5642	5	2	1535	64
tbk	6	3	1569	32
x5641	5	1	1617	64
x63210	6	10	1649	32
x6326	6	6	1793	32
m6325	6	5	2048	32

Table 2: Comparisons of NOVA and our Hyper-Place

Example	NOVA(default)				NOVA(hybrid)				Hyper-Place	
	cubes	area ratio	CPU time (s)	CPU time ratio	cubes	area ratio	CPU time (s)	CPU time ratio	cubes	CPU time (s)
s820	85	1.13	1.3	0.24	76	1.01	4.2	0.78	75	5.4
s832	71	0.97	1.3	0.25	72	0.99	4.2	0.79	73	5.3
s1494	149	1.14	3.1	0.09	139	1.06	388.8	10.83	131	35.9
s1488	141	1.07	2.7	0.07	133	1.01	416.6	11.23	132	37.1
x3643	271	1.04	7.8	0.17	251	0.97	2560.2	55.90	260	45.8
x4322	261	1.30	2.7	0.18	155	0.77	732.6	50.18	201	14.6
d5326	295	1.15	19.2	1.14	233	0.91	3621.9	215.60	257	16.8
d5322	217	1.10	5.5	0.28	233	1.18	984.4	50.74	197	19.4
x4326	359	1.24	29.7	1.62	266	0.92	3322.0	12.08	290	18.3
d5324	374	1.34	26.1	1.43	248	0.89	2148.0	118.02	279	18.2
d5323	423	1.16	59.7	3.21	330	0.90	3212.3	172.70	365	18.6
x5322	428	1.33	28.3	1.46	232	0.72	1718.4	88.58	323	19.4
x5321	525	1.24	7.1	0.53	378	0.89	2301.0	171.72	423	13.4
x5324	557	1.37	8.9	0.49	388	0.95	2284.8	125.54	408	18.2
s298	723	1.07	58.1	0.65	624	0.92	5460.1	60.67	678	90.0
x5643	750	1.48	75.1	0.65	656	1.09	4020.3	34.84	518	115.4
x5642	840	2.20	129.9	0.95	347	0.91	3070.7	22.50	382	136.5
tbk	176	1.76	17.1	0.85	154	1.54	922.9	45.92	100	20.1
x5641	936	1.37	28.4	0.33	833	1.22	3889.7	45.12	682	86.2
x63210	1072	1.23	21.0	2.66	859	0.99	988.4	125.11	872	7.9
x6326	421	1.49	23.2	1.27	278	0.98	388.9	21.25	283	18.3
m6325	1675	1.19	228.1	11.46	1341	0.95	6619.5	332.64	1406	19.9
Total		1.29		1.005		0.99		62.84		

In Table 2, area ratio is the ratio of the number of cubes of NOVA to that of our Hyper-Place. CPU time ratio is the ratio of the CPU time (SUN SPARC 1+) of NOVA to that of our Hyper-Place. On average, NOVA *default* mode produces results with 29% higher area (number of cubes) and almost the same CPU time compared to Hyper-Place. NOVA *hybrid* mode takes over 62 times the CPU time required by Hyper-Place and produces almost the same quality of results (in terms of the number of cubes).
Table 3 compares the results of state assignment using different objective functions in the placement phase. The approach using the linear objective function costs 6% more cubes than the one using the quadratic objective function which confirms our statements in Section 4.1.1.

Table 3: Comparisons of using linear and quadratic objective functions

Example	Linear obj. fun.		Quadratic obj. fun.
	cubes	area ratio	cubes
s820	80	1.07	75
s832	68	0.93	73
s1494	139	1.06	131
s1488	133	1.01	132
x3643	253	0.97	260
x4322	184	0.92	201
d5326	264	1.03	257
d5322	188	0.95	197
x4326	293	1.01	290
d5324	278	1.00	279
d5323	370	0.99	365
x5322	310	0.96	323
x5321	477	1.13	423
x5324	402	0.99	408
s298	713	1.05	678
x5643	593	1.14	518
x5642	388	1.02	382
tbk	188	1.88	100
x5641	777	1.14	682
x63210	935	1.07	872
x6326	280	0.99	283
m6325	1534	1.09	1406
Total		1.06	

7. Conclusions

In this paper, we presented a new state assignment approach Hyper-Place which runs as fast as the NOVA's *default* mode but produces same quality results as the NOVA's *hybrid* mode. This was made possible by breaking the hypercube embedding problem into two steps: (1) mapping of the adjacency graph to a grid; (2) mapping the solution on the grid to one on a minimum dimensionality hypercube with dilation at most two. Hyper-Place is thus able to handle large FSM's (up to 500 states, equations with more than 1000 product terms) efficiently and robustly.

References

[1] D. B. Armstrong, "A programmed algorithm for assigning internal codes to sequential machines," IRE Trans. Elect. Comp. , vol. EC-11, pp. 466-472, 1962.

[2] J. E. Brandenburg and D. S. Scott, "Embeddings of communication trees and grids into hypercubes, Intel Scientific Computers Report #280182-001, 1985.

[3] R. K. Brayton, G. D. Hatchtel, C. T. McMullen and A. L. Sangiovanni Vincentelli, "Logic Minimization Algorithms for VLSI Synthesis," Kluwer Academic, 1984.

[4] M. Y. Chan, "Dilation-2 embeddings of grids into hypercubes," Proceedings of the 1988 International conference on Parallel Processing, pp. 295-298, 1988.

[5] G. Cybenko, D. W. Krumme and N. Venkataraman, "Fixed hypercube embedding," Information Processing Letters, v.25, pp.35-39, 1987.

[6] G. De Micheli, R. K. Brayton and A. L. Sangiovanni Vincentelli, "Optimal state assignment for finite state machines," IEEE Trans. on CAD, pp.269-284, 1986.

[7] S. Devadas, H. T. Ma, A. R. Newton and A. L. Sangiovanni Vincentelli, "MUSTANG: state assignment of finite state machines for multi-level logic implementations," IEEE Trans. on CAD, pp. 1290-1300, 1988.

[8] S. Devadas, H. T. Ma, A. R. Newton and A. L. Sangiovanni Vincentelli, "A synthesis and optimization procedure for fully and easily testable sequential machines," IEEE Trans. on CAD, pp. 1100-1107, 1989.

[9] S. Devadas, A. R. Wang, A. R. Newton and A. L. Sangiovanni Vincentelli, "Boolean decomposition in multilevel logic optimization," IEEE Journal of Solid-State Circuits, vol. 24, pp. 399-407, 1989.

[10] D. Z. Djokovic, "Distance-preserving subgraphs of hypercubes," J. Combinatorial Theory (B), pp.263-267, 1973.

[11] T. A. Dolotta and E. G. McCluskey, "The coding of internal states of sequential machines," IEEE Trans. Elect. Comput., vol. EC-13, pp.549-562, 1964.

[12] C. Duff and G. Saucier, "State assignment based on the reduced dependency theory and recent experimental results," ICCAD-91, pp. 222-225, 1991.

[13] K. M. Hall, "An r-Dimensional Quadratic Placement Algorithm," Management Science, vol. 17, pp.219-229, 1970.

[14] J. M. Kleinhans, G. Sigl and F. M. Johannes, "GORDIAN: A new global optimization/rectangle dissection method for cell placement," ICCAD-88, pp. 506-509, 1988.

[15] D. W. Krumme, N. Venkataraman and G. Cybenko, "Hypercube embedding is NP-complete," Proc. Hypercube Conf., SIAM, 1985.

[16] D. Lewin, *Computer-aided design of digital systems*, Crane/ Russak, 1977.

[17] K. Roy and S. Prasad, "SYCLOP: synthesis of CMOS logic for low power applications," IEEE International Conference on Computer Design, pp.464-467, 1992.

[18] Y. Saad and M. H. Schultz, "Topological properties of hypercubes," Res. Report 389, Department of Computer Science, Yale univ., 1985

[19] G. Sigl, K. Doll and F. M. Johannes, "Analytical placement: a linear or quadratic objective function?" 28th DAC, pp.427-432, 1991.

[20] R. E. Stearns and J. Hartmanis, "On the state assignment problem for sequential machines II," IRE Trans. Elect. Comput., vol. EC-10, pp.593-603, 1961.

[21] R-S Tsay, E. Kuh and C-P Hsu, "Proud: a sea-of-gates placement algorithm," IEEE Design & Test of Computers, pp.44-56, Dec. 1988.

[22] T. Villa and A. L. Sangiovanni-Vincentelli, "NOVA: State Assignment of Finite State Machines for Optimal Two-Level Logic Implementations," IEEE Trans. on CAD, pp. 905-924, 1990.

[23] S. Yang, "Logic synthesis and optimization benchmarks user guide," Version 3.0, MCNC, 1991.