State Assignment Using a New Embedding Method Based On an Intersecting Cube Theory

G. Saucier
C. Duff
INPG/CSI
46, Avenue Félix-Viallet,
38031 Grenoble Cedex, France

F. Poirot
VLSI Technology Inc
Route des Dolines
Sophia Antipolis,
06560 Valbonne, France

Abstract - The controller state assignment methodology proposed here features two improvements over existing methods. First, a larger set of predictive minimizations in the control flowgraph is performed. Secondly, the embedding phase uses a new theory of intersecting cubes in the Boolean lattice. Practical results using the VLSI Technology Logic-Synthesizer on both PLA and multi-level logic demonstrate the effectiveness of the approach.

Introduction

State assignment is an important step in the synthesis of controllers from high-level specifications such as control flowcharts, control flowgraphs and state graphs. Once binary codes have been assigned to the states, next-state and output equations are defined and subsequently minimized with classical logic synthesis tools. The state assignment must be performed in a way that favors simplification of the next-state and output logic (implemented on PLA, standard cells, ROMs,...). The prediction of these down-line minimizations is one of the most difficult aspects of state assignment.

State assignment has been studied intensively in the 1960s and 1970s. These research efforts had two main objectives. The first was the avoidance of critical races for asynchronous sequential circuits. Skillful state assignment could ensure the non-existence of races for these circuits ([UNG63], [ARM62], [LIU63], [SAU72a]). Some proposed solutions consisted in placing, for a given input, the states leading to the same stable state on the same face of the hypercube (excluding the other ones). Universal solutions for any state machine of N states [SAU68] as well as minimized dedicated solutions were proposed. A second objective was the simplification of the next-state and output equations [HAR66]. A point which was clearly brought forth at that time was the compromise between the number of internal variables and the simplicity of the next-state equations [SAU72b]. From a theoretical point of view, two approaches were identified. The first one was based on the partition theory. A partition of the set of states is associated to an internal variable: the states for which this variable is equal to 0 and those for which it is equal to 1. This corresponds to a bisection of the hypercube, from which follows the partition pair theory of Hartmannis [HAR66]. For a given input, any partition of the set of states is mapped into another one through the state table, and therefore gives the key to the next-state (and similarly for the output) equation complexity. In all these approaches, the state assignment is usually performed progressively, variable by variable, choosing the next bisection of the hypercube. This is called column encoding. In parallel, another approach was proposed based on the Boolean lattice theory. A Boolean N-cube is isomorphic to the lattice of the set of parts of a set of N elements. Assigning a binary code to a state consists of assigning a subset of elements to this state. This different view allows for an improved optimization of the state assignment in terms of the number of internal variables for universal assignment as well as for dedicated assignment.

With the advent of silicon compilers, all these approaches were re-investigated to produce efficient state assignment tools for controller synthesis following new achievements in the minimization area. Two steps are clearly identified. The first is the search for situations in the controller specification which will lead to further minimization. An obvious situation for all approaches is the existence of a set of states leading to the same next state. This, of course, leads to an expression which is the sum of the codes of the ancestor states for all internal variables equal to 1 in the next state. Placing all the nodes on a same face of the hypercube reduces this expression to one product term. This situation was recognized in all the proposed tools (KISS [DEM85], ASYL [SAU87], MUSTANG [DEV87], [COP86],..). Based on practical experience, the importance of output logic compared with next-state logic was pointed out in the ASYL system. The practical designs of complex controllers (dedicated μprocessor controllers) showed the important ratio of output logic (4/5 of the area) and the importance of the minimization of the number of internal variables, because of the critical wiring problems between the controller and the datapath and of the decoder size. Therefore, sets of states (or edges) sending identical outputs were recognized in ASYL [SAU87]. This was also done in MUSTANG [DEV87]. As mentioned previously, efficient optimization requirements have led the authors to attempt to recognize more potential simplifications. Therefore, more sophisticated situations are presently recognized (multiple output nodes, edges labelled with same or intersecting inputs). These will be described in detail in this paper. A predictive gain must be associated with all these situations since, during the embedding phase, directed heuristics will attempt to satisfy the situations with the highest predicted gain. The computation of this gain is important as it will be shown that it is based on further minimizations used for the logic.

For the embedding phase, the two classical approaches previously mentioned are exploited. The partition theory approach leads to a column encoding. The nodes belonging to a given situation called adjacency group are placed on a same face defined by fixed values of one or several internal variables; this corresponds to a bisection of the hypercube [DEM85]. The drawback of this approach is the fact that the increase of potential minimizations leads to an increase of the number of internal state variables which is prohibitive for highly optimized controllers. Therefore, progressive embedding (row encoding) in the hypercube with an extendable dimension is preferable. In MUSTANG [DEV87], the nodes of an adjacency group are placed in "close neighborhood" in the hypercube. In ASYL, an early version performed an embedding in the hypercube based on the necessary conditions of a graph to be isomorphic to a subgraph of a hypercube. In practice, the nodes were first assigned to layers of the target hypercube and progressive assignment of relative coordinates to the nodes was then performed [SAU87]. The new version of the embedding algorithm presented here is more sophisticated and more efficient. As the recognized situations have become more numerous, a global treatment of their intersection is realized. An original theory of intersecting cubes is developed. This leads to an embedding of the successive adjacency groups that takes into account the intersections between these groups, and also to the definition of the backtracking possibilities. This provides a near-optimal solution.

As far as logic targets are concerned, a PLA implementation is clearly taken in KISS [DEM85]. Therefore, once the situations are detected in KISS (multiple input node only), a dedicated preparation is performed in order to reach a good global minimization. The PLA target is abandoned in MUSTANG. The placement of nodes in close neighborhood leads to large common factors in the set of codes and eases global minimization through factoring. In ASYL, it has appeared in practice, that large controllers need multiple PLAs or multi-level logic implementation. Both targets are therefore considered. Nodes of an adjacency group are put on a single face (which is the best solution) or in a close neighborhood. The multi-level target is prepared by gain consideration which takes into account the intersections or common product terms between the functions. In [KEU88], multi-level logic is targeted and a method based on kernel finding is proposed. As shown by the authors, they did not improve on a random encoding.

This paper reports on results of a cooperative research between VLSI Technology Inc., and the INPG/CSI laboratory.

The paper is organized as follows: In section 1, a summary of the richest set of situations presently identified for state assignment purposes is described. The associated costs will be discussed. The second section enumerates the different types of embeddings used, with special emphasis on a novel approach that is based on an intersecting cube theory of the Boolean lattice. In the last section, comparative results are presented.

I. Situation recognition and adjacency groups

1.1. Types of situations and adjacency groups

Four basic situations are identified : multiple input nodes, nodes or edges sending the same outputs (Moore or Mealy models), multiple output nodes, and edges labelled with the same input or with intersecting inputs.

Situation 1 : Nodes with multiple input edges

This frequent situation in controller flowgraphs has been widely discussed in the literature. Branching on input variables split controllers into distinct states converging again on key points of the flowgraph. Edges leading to a join node are usually not labelled with any input condition; thus, the product terms produced by this situation are the following: For each internal variable equal to 1 in the join node code, the sum of the ancestor node codes is produced.

The ancestor nodes of the join node constitute an adjacency group. The best solution for any further implementation is to put these nodes on a cube of the hypercube. For multi-level logic, a weaker solution consists of placing these nodes as close as possible in order to increase common subexpressions [DEV87]. A special version for PLA minimization has been widely discussed in [DEM83].

Situation 2 : Nodes or edges sending the same outputs

In Moore (resp. Mealy) controllers, outputs are associated with nodes (resp. edges). For Moore controllers, states sending the same outputs are put in a same adjacency list. If the same set of states send several outputs, the associated weight (for a purpose explained later) is increased. For Mealy controllers, outputs are associated with edges.

Situation 3 : Nodes with multiple output edges

Edges are labelled with exclusive predicates (conditions) on the input variables; the sum of these predicates must be 1. Suppose that a node has 2^k edges labelled with 2^k minterms on the input variables (Figure 1). These input minterms define k partitions of the successor nodes, a partition being defined by an input variable being 0 in one block of the partition and 1 in the other. In Figure 1, the partitions are for instance {(S1, S2), (S3, S4)} for variable "a" and {(S1, S3), (S2, S4)} for variable "b". These partitions define the adjacency relations between the nodes of a 2-cube.

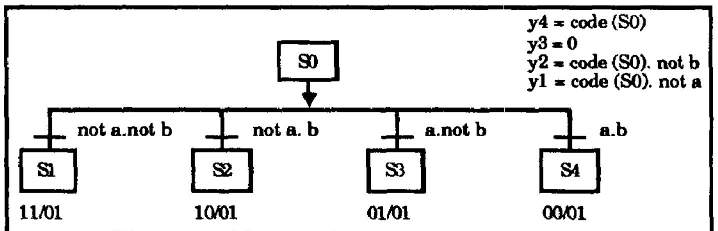

Figure 1 Nodes with multiple output edges

This situation leads to declare {S1, S2, S3, S4} as a constrained adjacency group: The nodes have to be assigned on a 2-cube defined by the partitions of the input minterms. This means that {(S1, S2), (S3, S4)} and {(S1, S3), (S2, S4)} are adjacency relations that have to be respected when constructing the 2-cube associated with the successor nodes. This leads to the generation of the product term {code(S0)}, for all invariant internal variables of the successor nodes and a product term {code (S0). ~Ei} (~Ei is either Ei or not Ei), for the other internal variables.

Situation 4 : Edges labelled with the same inputs or with intersecting inputs

This situation refers to previous work on partition pairs [HAR66]. A given input maps current state on next states. A first step consists in identifying all edges in the flowgraph labelled with the same input or intersecting inputs. In the example given in Figure 2, 4 edges labelled with the same input "I" go from states S1, S2, S3, S4 to states S5, S6, S7, S8.

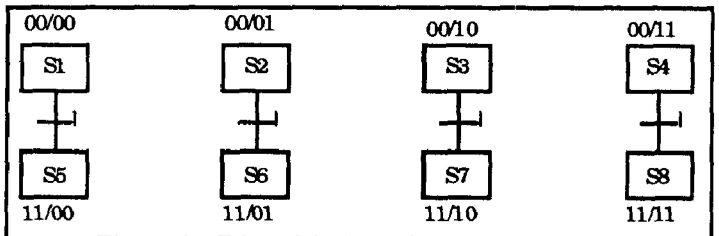

Figure 2 Edges labelled with the same inputs

Suppose we assign the four origin nodes to a 2-cube defined by the two partitions {(S1, S2), (S3, S4)} and {(S1, S3), (S2, S4)}. These partitions are mapped on two other partitions by input "I" (namely {(S5, S6), (S7, S8)} and {(S5, S8), (S6, S7)}). If we now assign the end nodes to a 2-cube defined by the two partitions {(S5, S6), (S7, S8)} and {(S5, S8), (S6, S7)}, we will get a set of product terms well suited to *multi-level logic* :

Y4 = Y3 = not Y4 . not Y3
Y2 = not Y4 . not Y3 . Y2
Y1 = not Y4 . not Y3 . Y1

More precisely, the invariant variables of the codes of the end nodes produce a product term equal to the invariant part of the codes of the start nodes. The other variables Y_i produce a product term equal to the product of the invariant part of the code by Y_i. This property is an extension of a reduced dependency property [HAR66] due to the fact that the internal variables respect the partition pairs associated with input "I". The difference here is that all the states do not appear in the partition.

A simple implementation consists in searching sets of 2^k nodes leading through 2^k edges labelled with the same or intersecting inputs to 2^k successor nodes, and assigning them to 2 k-cubes respecting the mapping of the partitions defining the cubes. An interesting case to point out is a set of loops labelled with the same input condition. This last situation has been used successfully.

Conclusion on the situation recognition phase

The first step of situation recognition ends with the creation of node lists which have to be assigned to a face or cube, or at least put in close neighborhood. For the two last situations, this is somewhat more complicated. The third situation tends to put nodes on a cube with the following constraint : Once one node has been assigned, the assignments of all the other ones are mandatory - there is no more freedom for the code of these nodes. The fourth situation aims at assigning two sets of nodes in a dependent way ; if one set of nodes is assigned to a cube, the structure of the second cube is deduced (the adjacency relations between its nodes are defined).

1.2. Gain associated with a situation

The gains associated with the above-mentioned situations play an important role as they lead to an ordering of the adjacency groups. A local gain is associated with each situation and a global gain takes into account the intersections between adjacency groups. This has the effect of preparing common expressions at any level between different functions (output functions, internal variables, etc.), thus providing further multi-level minimizations. The major advantage of our approach is that the intersections or common subexpressions are preserved during the embedding phase.

1.2.1. Local gain associated with the situations

The local gain is the number of occurrences of variables saved by placing the corresponding adjacency group on a cube. More precisely we take an upper bound of this number equal to the difference between the worst case and the final expression. Let us show this gain for the first two situations.

Nodes with multiple input edges

For this situation with p ancestor nodes ($p = 2^k$) , for each internal variable Y_i equal to 1, the expression is

$$Y_i = \text{(input condition)}.(\text{code } (S_1) + \ldots + \text{code } (S_p)).$$

If N is the number of internal variables and c_j the number of input variables appearing in each of the p conditions, the worst case is (Σc_j+pN) variables in the expression of Y_i. In the best case, if the ancestor nodes are put on a cube, a common factor of (N-k) variables is obtained. The cost is therefore $[(N-k) + (pk + \Sigma c_j)]$ and the gain is equal to (p-1)(N-k) where $k = \lceil \log_2 (p) \rceil$. As the number of internal variables equal to 1 in the next-state code is unknown at this stage, we use an average coefficient of N/2 to weight this gain. The final gain is then N (p-1)(N-k) / 2.

Output situations

The reasoning is very similar to that of the join situation except that the output functions are considered individually, and not "clustered" like the internal variables in the previous situation. The gain associated to an adjacency group is therefore (p-1)(N-k).

1.2.2. Global gain

To cope with multi-level synthesis, special attention is given to intersections between adjacency groups. If respected, these intersections bring potential subexpressions between functions and as the embedding technique has been elaborated for that purpose, this potential is realized. Global gain considerations lead to respect not only a particular adjacency group, but also other adjacency groups included, or partially included in it. It will therefore allow simplification in another function. As such, to the local cost will be added the partial cost of parts of adjacency groups included in this group.

Consider, for instance, two adjacency groups due to join situations {1,2,5,6,10,12} and {5,6,10,13}. The first one has a local gain which will be increased by a partial gain coming from the second one by placing {5,6} on an edge. The cost of the first adjacency group is equal to gain ({1,2,5,6,10,12}) + partial gain ({5,6,10,13}). To handle this easily, a linear approximation is done. Gain(p) as previously indicated is equal to $N(p-1)(N - \lceil \log_2 (p) \rceil)/2$. The global gain considered is therefore gain(8) + gain(4).

II. State assignment

Two state assignments have been implemented for comparison purposes: a simple column encoding assignment with an improved mixed (row/column) assignment inspired by the CADDY system [ROS88], and a sophisticated embedding procedure based on an intersecting cube theory of the Boolean lattice. The latter has been used for practical optimized designs. In all cases, the starting point is an ordered list of adjacency groups.

2.1. The ASYL column encoding and mixed encoding methods

2.1.1. A simple implementation

The adjacency groups are considered in a decreasing gain order. The goal is to assign to each group a set of distinguishing internal variables; their values constitute the identification pattern of the group. These variables define the face of the hypercube dedicated to this group. Internal variables are associated progressively. For the ith adjacency group, the algorithm is the following :

1) Are there already assigned internal variables which take the same value for all the states of this adjacency group ? If yes, they are distinguishing internal variables for this adjacency group: Go to 2. If no, go to 3.

2) Are there states which do not belong to this adjacency group and have the same value of the distinguishing variables of this adjacency group ? If yes, assign a new internal variable distinguishing these states from those of the adjacency group. The value of this variable for all other states is indifferent. If no, go to 4.

3) Assign a new internal variable to distinguish the states of the adjacency group from the other ones. Go to 4.

4) Consider a new adjacency group. Go to 1.

In a final step, final internal variables are added, if necessary, to distinguish all the states.

Let us illustrate this with an example. In a 10-state controller, four adjacency groups have been identified :

(1,2,3,4), (2,3,7), (8,9,10) and (4,6,8). The different steps can be followed in Figure 3.

Step 1 : First adjacency group (1,2,3,4)

Variable Y_1 is assigned to distinguish (1,2,3,4); the identification pattern is [0].

Step 2 : Second adjacency group (2,3,7)

Variable Y_2 is assigned to this group, the identification pattern of which is [-0].

Step 3 : Third adjacency group (8,9,10)

A part of the identification pattern is [11]; a third variable will be necessary to distinguish (8,9,10) from (5,6). Therefore, variable Y_3 is added and the identification pattern becomes [110].

Step 4 : Fourth adjacency group (4,6,8)

A part of the identification pattern is [-1-]; a fourth internal variable is necessary to distinguish (4,6,8) from (1,5,9,10).

Step	Variable	1	2	3	4	5	6	7	8	9	10
1	Y_1	0	0	0	0	1	1	1	1	1	1
2	Y_2	1	0	0	1	1	1	0	1	1	1
3	Y_3	*	*	*	*	1	1	*	0	0	0
4	Y_4	*	*	*	1	0	1	*	1	0	0

Figure 3 Column encoding

Of course, if the number of internal variables is limited, the set of considered adjacency groups may be limited consequently.

2.1.2. Improved column encoding [ROS88]

Inspired by the CADDY system, a simple improvement of the column encoding has been implemented. The adjacency groups are partitioned into clusters such that a state appears only once in a cluster. A minimal set of internal variables are then chosen to distinguish the adjacency groups in each cluster. The states belonging to none of the adjacency groups of the cluster constitute an additional group. Additional variables have to be added to distinguish all the states in the final step.

Consider the following list of adjacency groups obtained for a set of 10 states : (1,2,3,4), (2,4,5), (5,6,7), (1,7), (6,8,9) and (8,9)

The clustering procedure will give :

C_1 = { (1,2,3,4), (5,6,7), (8,9), (10) }

C_2 = { (2,4,5), (1,7), (6,8,9), (3,10) }

Two variables (Y_1,Y_2) are chosen to identify the blocks of C_1 and two variables (Y_3,Y_4) distinguish the blocks of C_2.

Variable	1	2	3	4	5	6	7	8	9	10
Y_1	0	0	0	0	0	0	0	1	1	1
Y_2	0	0	0	0	1	1	1	0	0	1
Y_3	1	0	1	0	0	0	1	0	0	1
Y_4	0	0	1	0	0	1	0	1	1	1

identical codes

Figure 4 Improved column encoding

A single variable is necessary to distinguish 8 and 9 (as well as 2 and 4) as the intersection of all the partitions in C_1 and C_2 is (1)(2,4)(5)(6)(7)(8,9)(10).

2.2. The assignment based on an intersecting cube theory of the Boolean lattice

2.2.1. The hypercube considered as a lattice of the parts of a set of N elements

The following representations given in Figure 5 of a 3-cube are equivalent and illustrate the hypercube seen as a Boolean lattice. The coordinates of Figures 5b and 5c are called relative coordinates. They indicate which bits in the binary code differ from the origin code. Once the code of an element is chosen, all the other codes are defined.

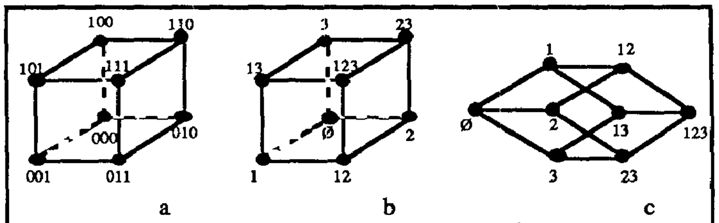

Figure 5 Different representations of a 3-cube

An ordering relation corresponding to the classical inclusion relation is defined among the relative coordinates. For example, we have 12 < 123 and 3 < 123.

Faces in the hypercube

A face can be defined by the couple of the minimal and maximal relative coordinates of its nodes (ε_{min}, ε_{max}) with $\varepsilon_{max} \supseteq \varepsilon_{min}$. If $\varepsilon_{max} = \varepsilon_{min}$, the cube is reduced to a node. In the 3-cube of Figure 5c, [2,123], [1,123] are 2-cubes, [Ø,123] is the 3-cube itself. The set of elements $\varepsilon_C = \varepsilon_{max} - \varepsilon_{min}$ is called the characteristic index set of the cube. The dimension of the cube is given by the cardinality of the set of its characteristic indices. For example {1,3} and {2,3} are respectively the characteristic sets of the two cubes [2,123] and [1,123]. A cube can also be defined either by (ε_{min} , ε_C) or by (ε_C, ε_{max}).

2.2.2. The intersecting cube theory

Definition :

The cube intersection of two cubes (ε_{min}, ε_{max}) and (ε'_{min}, ε'_{max}) is defined by ($\varepsilon_{min} \cup \varepsilon'_{min}$, $\varepsilon_{max} \cap \varepsilon'_{max}$). The intersection is empty if $\varepsilon_{max} \cap \varepsilon'_{max}$ does not include $\varepsilon_{min} \cup \varepsilon'_{min}$.

Properties :

The intersection cube has as characteristic indices the intersection of the sets of characteristic indices of the original cubes.

Two cubes have a non empty intersection if

$$\varepsilon_{max} \supseteq \varepsilon'_{min} \text{ and } \varepsilon'_{max} \supseteq \varepsilon_{min}$$

For example, the two 2-cubes [1,123] and [2,123] intersect at the edge [12,123]. There is no intersection between cubes [2,123] and [24,2345] as {2,4} is not included in {1,2,3}.

Basic procedure used in the embedding phase :

Given a fixed k-cube, find a k'-cube having an intersection with the k-cube, the intersection being a k"-cube.

The first cube is given by its two characteristic sets [ε_{min}, $\varepsilon_{min} \cup \varepsilon_C$]. For the second cube we shall construct its pair of sets of indices [ε'_{min}, ε'_{max}] by distinguishing four subsets of ε'_{max} which can be written as $\varepsilon'_{max1} \cup \varepsilon'_{max2} \cup \varepsilon'_{max3} \cup \varepsilon'_{max4}$ where

* ε'_{max1} is the locally inherited set of indices within this cube and is equal to ε'_{min},

* ε'_{max2} is the set of indices inherited from the first cube and is equal to $\varepsilon_{min} - \varepsilon'_{min}$,

* ε'_{max3} is the set of characteristic indices of the intersection and is therefore composed of k" elements of $\varepsilon_C - \varepsilon'_{min}$,

* ε'_{max4} is the remaining part of ε'_{max} and is made up of $(k'-|\varepsilon'_{max2}|-k")$ indices belonging to $\varepsilon_N - \varepsilon_{max}$ in which ε_N is the whole set of indices of the cube; ε'_{max4} may be empty.

The construction of ε'_{max} is summarized in figure 6:

ε_{max1}	ε_{max2}	ε_{max3}	ε_{max4}
ε'_{min}	$\varepsilon_{min} - \varepsilon'_{min}$	choose k" elements from $\varepsilon_C - \varepsilon'_{min}$	choose k' -| ε_{max2} | - k" elements from $\varepsilon_N - \varepsilon_{max}$

$\longleftarrow$ k' elements $\longrightarrow$

Figure 6 Structure of the intersecting cube

Let us illustrate this with an example: suppose we have a 3-cube [12,12345] and that we look for another 3-cube having a 2-cube intersection with the first one in a 5-cube. Suppose we take {1,5} as ε'_{min}, then the second cube will be defined as indicated in figure 7.

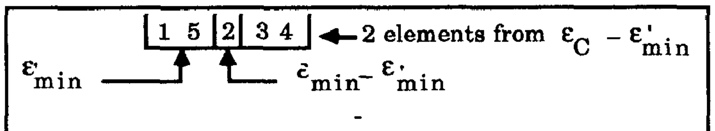
1 5 2 3 4
2 elements from $\varepsilon_C - \varepsilon'_{min}$
ε'_{min}
$\varepsilon_{min} - \varepsilon'_{min}$

Figure 7 Construction of the intersecting cube

ε'_{max4} is empty, the intersection is [125,12345].

ε'_{min} is chosen among ε_{max}, i.e, $\varepsilon_{min} \cup \varepsilon_C$. In ASYL, ε'_{min} is chosen so that the origin node of the second cube is in the leftmost layer. This allows an optimized filling of the hypercube.

2.2.2. The embedding algorithm

The embedding algorithm is very simple. The adjacency groups are processed in a decreasing gain order. Once a group has been assigned, the groups intersecting with it, are processed first according to the basic procedure. The different solutions are stored for backtracking. When all the intersecting groups have been embedded, the next group is considered. If no solution is found, the dimension is increased within an allowed range or a closest neighborhood solution is taken (similar to the MUSTANG approach).

Let us illustrate that with the first example (section 2.1.1). The adjacency groups extended to cubes in a decreasing order gain are (1,2,3,4), (2,3,7,φ), (8,9,10,φ) and (4,6,8,φ). The four groups are embedded in a 4-cube as shown in Figure 8.

adjacency group	cube assignment	intersection assignment
(1,2,3,4)	[ϕ,12]	
(2,3,7,φ)	[ϕ,13]	[ϕ,1] assigned to states (2,3)
(4,6,8,φ)	[2,234]	[2,2] assigned to state 4
(8,9,10,φ)	[4,412]	[24,24] assigned to state 8

Figure 8 Embedding results for the first example

The adjacency group (4,6,8,φ) has been processed before (8,9,10,φ) as it intersects with (1,2,3,4). As an example, let us comment on the assignment of (4,6,8,φ). We look for a 2-cube which intersects through one node with (1,2,3,4) and which does not intersect with (2,3,7,φ). A node which is included in ε_{max} of (1,2,3,4) and not included in ε_{max} of (2,3,7,φ) is chosen; for instance, let's take 2. Then two other indices are chosen. They must not be included in ε_{max} of (1,2,3,4). Otherwise the intersection with [ø,12] will be too large. Therefore (3,4) is chosen.

Let us now consider the second example (section 2.1.2). Without any backtracking the solution shown in the following table (Figure 9) is found.

adjacency group	cube assignment	intersection assignment	
(1,2,3,4)	[ϕ,12]		
(2,4,5,φ)	[ϕ,13]	[ϕ,1] [2,12]	(2,4) (1,3)
(5,6,7,φ)	[3,234]	[3,3]	5
(1,7)	[2,23]	[2,2] [23,23] [12,12]	1 7 3
(6,8,9,φ)	[4,134]	[34,34]	6
(8,9)	[14,134]		

Figure 9 Embedding results for the second example

As shown in Figure 10, this gives a very dense solution on a small portion of the 4-cube. If we take the code 0000 for the state 4 (relative coordinate 0), the number of "1" in the codes is minimized. As an ultimate optimization criterion, the choice of the 0 code is left as an option in the last step. Notice also that the first four adjacency groups can be respected on a 3-cube.

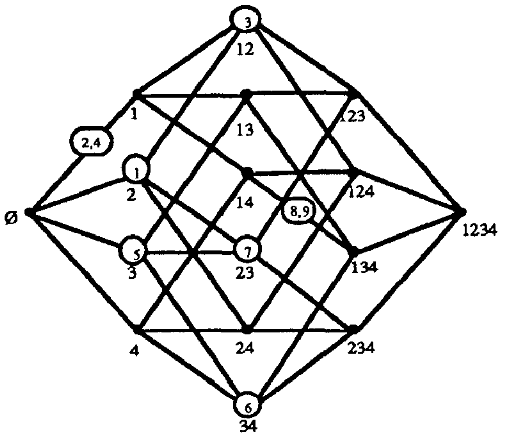

Figure 10 State assignment in the 4-cube

III. Results

A first version of the algorithm presented here has been implemented in cooperation with VLSI Technology Inc.(France). Some features are not yet available (not all the situations are detected and no backtracking for example). But the present results seem very encouraging and, as they are implemented in a real industrial environment, they can be considered as significant.

Let us first comment about state assignment evaluation. State assignment of controllers should be evaluated on a reasonable number of controllers having adequate features. The problem is mainly how to evaluate it. The controller synthesis goes through different steps and, of course, only the state assignment has to be evaluated.

The first and the best manner to evaluate a state assignment algorithm by an absolute information is to summarize the number of situations recognized in the control flowgraph and the percentage of situations or constraints satisfied during the state assignment. Any other estimation tends to compare a combination of two or more algorithms. The number of literals after factoring depends on the final target (area versus time optimization, structure of the library, ...). The number of product terms on PLA depends on the two-level minimizer.

The second manner to evaluate properly state assignment is to do comparisons with an average number of random encodings. The comparison can therefore be done at any level of the synthesis process with the appropriate criteria: number of literals after factoring, number of standard cells, area, speed, etc. Any other comparison does not reflect a physical implementation.

benchmarks characteristics						random, row, column state assignment (PLA)			Multi-level logic	
Name	#i	#o	#s	#svm	#scv	#ptra	#ptrow	#ptcol	#egr	#egv
bbara	4	2	10	4	5	30	27	27	82	62
bbtas	2	2	6	3	4	14	11	13	23	23
donfile	2	1	24	5	18	66	65	56	-	-
keyb	7	2	19	5	5	111	57	51	264	169
mc	3	5	4	2	2	9	8	8	16	12
modulo12	1	1	12	4	4	15	13	13	45	32
planet	7	19	48	6	10	106	106	98	601	564
s1	8	6	20	5	8	101	87	80	367	290
s1a	8	6	20	5	7	96	80	75	280	206
shiftreg	1	1	8	3	5	12	9	10	21	14
tav	4	4	4	2	3	11	11	11	16	17
tbk	6	3	32	5	15	271	182	157	874	515

Notations

#i,#o,#s:	Number of inputs, outputs, states
#svm :	Minimum number of state variables
#svc :	Number of state variables required to satisfy all the recognized situations
#ptcol :	Number of product terms after column encoding using the minimum number of state variables
#ptrow :	Number of product terms using the ASYL row encoding algorithm
#ptra :	Number of product terms after random encodings
#egr :	Number of equivalent gates after multi-level optimization using random encodings
#egv :	Number of equivalent gates using VLSI-adjacency-state-encoding

Figure 11 State assignment results

The first part of the table in figure 11 gives the benchmarks characteristics.

The second part of the table illustrates the superiority of our row encoding with respect to the column encoding for PLA implementation when using a minimal number of internal variables. As mentioned previously, this is mandatory for highly optimized large controllers where the wiring problem is predominant. The ratio (Σ#ptra / Σ#ptrow) is 1.41, compared to the ratio (Σ#ptra / Σ#ptcol) which is equal to 1.28. The number of product terms have been obtained by using the VLSI Technology two-level minimizer.

The third part of the table gives the result of the presented approach in the VLSI Technology multi-level implementation environment. The measure used is based on the gate equivalent notion defined in the VLSI Technology user environment. A gate equivalent represents the number of transistors of a 2-input-CMOS-NAND gate (4 transistors). This number is associated with any gate of the VLSI Technology library. It represents the number of transistors divided by four ([VSC100]). This criterion has been chosen as being properly related to the area. The Ratio (Σ#egr / Σ#egv) = 1.35 indicates a gain of 35% in the number of 'gate equivalent' in the standard cells implementation. This result is more significant than a literal count and proves an effective gain. The computation time is presently negligible compared to the multi-level synthesis time.

The gains (41% for PLA, 35% for multilevel logic) show that the unified package implementing this approach affords good results for both targets. The comparative evaluation after real implementation gives great confidence as no further optimization phase can be suspected to decrease this gain.

Conclusion

The present results confirm the importance of state assignment in the controller synthesis. The method shown above is successful for several targets (PLA or multi-level logic) and relies on a deep understanding of embedding problems. Moreover, this state assignment method simplifies and tends to shorten the work of the multi-level minimization tool, resolving an important practical issue.

References

[ARM62] D.B. Armstrong : "A Programmed Algorithm for Assigning Internal Codes to Sequential Machines", IEEE Trans. on Elec. Comp., pp. 466-472, August 1962

[COP86] A.J. Coppola : "An Implementation of a State Assignment Heuristic", 23rd DAC, pp. 643-649, July 1986

[DEV87] S. Devadas et al. : "MUSTANG : State Assignment of Finite State Machines for Optimal Multi-level Logic Implementations", ICCAD 87, pp. 16-19

[DEM83] G. de Micheli : "Computer-Aided Synthesis of PLA-based Finite State Machines", ICCAD 83, pp. 154-156

[DEM85] G. de Micheli, A. Sangiovani-Vincentelli, R.K. Brayton : "Optimal State Assignment of Finite State Machines", IEEE Trans. on CAD, July 1985, pp. 269-285

[KEU88] K. Keutzer et al. : "A Kernel-Finding State Assignment Algorithm for Multi-Level Logic", 25th DAC, pp. 433-438, June 1988

[HAR66] J. Hartmanis, R.E. Stearns : "Algebraic Structure, Theory of Sequential Machines", Prentice Hall, 1966

[LIU63] C.N. Liu : "A State Variable Assignment Method for Asynchronous Sequential Switching Circuits", ACM, April 63

[ROS88] W. Rosenstiel et al. : "Datapath and Control Synthesis in the CADDY System", International Workshop on logic and architecture synthesis for silicon compilers, Grenoble, France, May 88

[SAU68] G. Saucier : "Encoding of Asynchronous Sequential Networks", IEEE Trans. on E.C., vol 16, pp. 365-369

[SAU72a] G. Saucier : "State Assignment of Asynchronous Sequential Machines using Graph Techniques", IEEE trans. on Comp, March 1972

[SAU72b] G. Saucier : "Next State Equations of Asynchronous Sequential Machines", IEEE trans on Comp., November 1972

[SAU87] G. Saucier et al. : "ASYL : A Rule-Based System for Controller Synthesis", IEEE Trans on CAD, November 1987

[UNG63] S.H. Unger : "A Row Assignment for Delay-Free Realizations of Flow Tables Without Essential Hazards", IEEE Trans. on E.C. Vol 17, pp. 146-151

[VSC100] Portable 1.5μ CMOS Standard Cell Library, V1.1, VLSI Technology Inc., 1988