Approaches to the Input-Output Encoding Problem in Boolean Decomposition

Marek Perkowski, Michael Burns, Ralph Almeria, Nick Iliev

Portland State University, Dept. of Electrical Engineering
Portland, Oregon 97207
Tel: 503-725-5411, Fax: 503-725-4882
e-mail: mperkows@ee.pdx.edu

January 9, 1996

Abstract

So far, very little has been published in the literature on the encoding problem that occurs in functional (Curtis, Roth-Karp) decomposition. In particular, the encoding techniques used to optimize the decomposition of functions for this problem have tried to optimize only the predecessor or only the successor logic, but not both concurrently.

This report presents several approaches to the input-output "Symbol Encoding Problem" that occurs in the decomposition of Boolean functions. We show new methods for encoding, that attempt to optimize concurrently both the predecessor logic and the successor logic.

A new combined approach to the column minimization and encoding in functional decomposition is also presented, which means, these two problems that have been solved separately until now, are being solved together. We call this a "concurrent approach" to column minimization and column encoding, to distinguish it from "symbol encoding". In this approach the column reduction process is combined in a single search process with the column encoding to find the really minimum decomposition. We call it then also the "column encoding problem". This is similar to the problem of concurrent state minimization and assignment of internal states in a Finite State Machine.

Contents

List of Figures

1 INTRODUCTION

The problem of symbol encoding occurs in many problems related to the design of logic circuits:

1. Assignment of internal states.
2. Assignment of input states.
3. Assignment of output states.
4. Assignment of codes in microprogrammed units [9, 10, 11].
5. Assignment of values to symbolic variables in generalized Sasao decomposition [49].
6. Assignment of codes to the output of encoder in the decomposed realization of FSMs, realized with the two-dimensional minimization [61].
7. Assignment of signals to symbolic variables while designing iterative circuits.
8. Decomposition of logic functions [49, 60, 57, 56].

Encoding is the assignment of binary vectors to different values of some symbolic (i.e. multiple-valued) variable. This variable may be an input state variable, output state variable, or internal state variable. Below is an example of an encoding for a symbolic input to an Arithmetic Logic Unit (ALU). The ALU has two inputs for the operands, one input for the opcode (operator), and an output for the result. The issue here is to encode the opcodes in such a way that the logic is minimized. An example of encoding for this ALU is shown in Fig. 1.

```
SYMBOLIC    BINARY ENCODING

   add          000
   sub          001
   mul          100
   div          101
```

Figure 1: *Representations for Symbolic and Binary opcodes.*

Why do we care about Encoding? It has been determined that the selection of the encoding (the mapping to the binary vectors) makes a difference in the minimization or decomposition of the function in question. For instance, the encoding shown above in Fig. 1 for the states could be changed to the following encoding from Fig. 2.

Notice that the codes have been changed. These codes or other similar ones could yield a simpler function implementation (i.e. fewer logic gates or a smaller value of DFC, see [1, 46] for the definition of DFC). This is the problem that we are trying to solve - "how to come up with an optimal encoding of the symbolic values".

What type of encoding are we interested in here? For this project, we are interested in the *output-input encoding problem.* This problem is slightly unusual in that it is much like the classical *internal*

SYMBOLIC	BINARY
add	100
sub	011
mul	000
div	001

Figure 2: *Representations for Symbolic and Binary opcodes.*

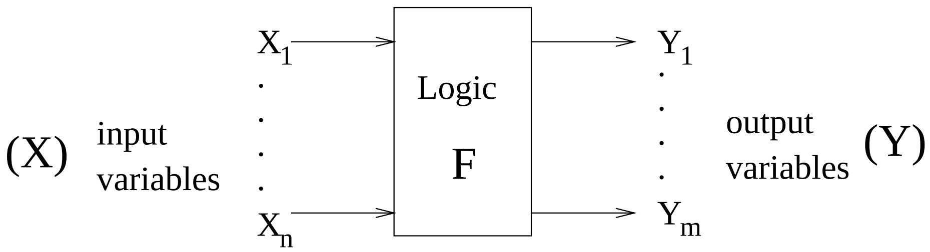


Figure 3: *Logic function before input variable partitioning.*

state encoding (or, state assignment problem (i.e. one of input-output encoding), but reversed. To fully understand this problem we will have to explain what is meant by the output-input encoding problem.

In decomposition, we are trying to reduce the complexity of an implementation of a logic function F by dividing it into two logic blocks: block G, called the *predecessor block*, and H, called the *successor block*. In order to accomplish this, the input variables to the function block F (see Fig. 3) must be divided into a *free set* of H and a *bound set* of G.

The free set of variables are those that are unmodified and sent directly to the new logic block H. The bound set is the group of variables which will be passed through the logic block G in order to reduce the size and complexity of the implementation. The logic block G has outputs v_i which serve as *intermediate variables* (see Fig. 4). Since these are new variables, and we don't really care what values these intermediate variables take on, we need, therefore, to find out what bit vectors these variables should be assigned with. This is our encoding problem. One way of formulating the encoding problem is by imposing certain constraints to be satisfied. We must find a way of simultaneously satisfying the

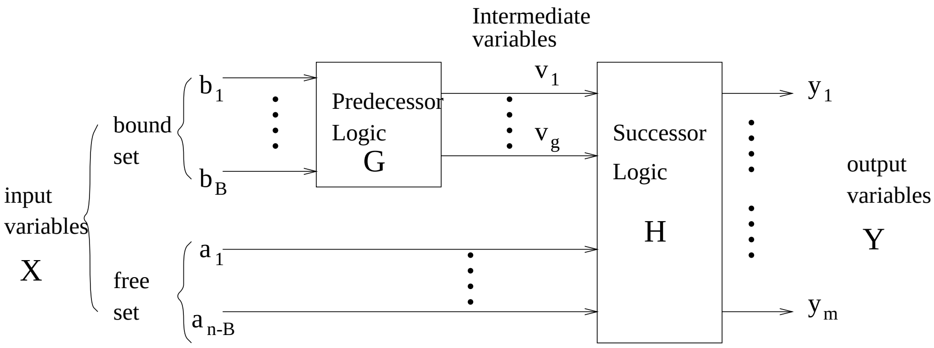


Figure 4: *Logic function after partitioning of input variables.*

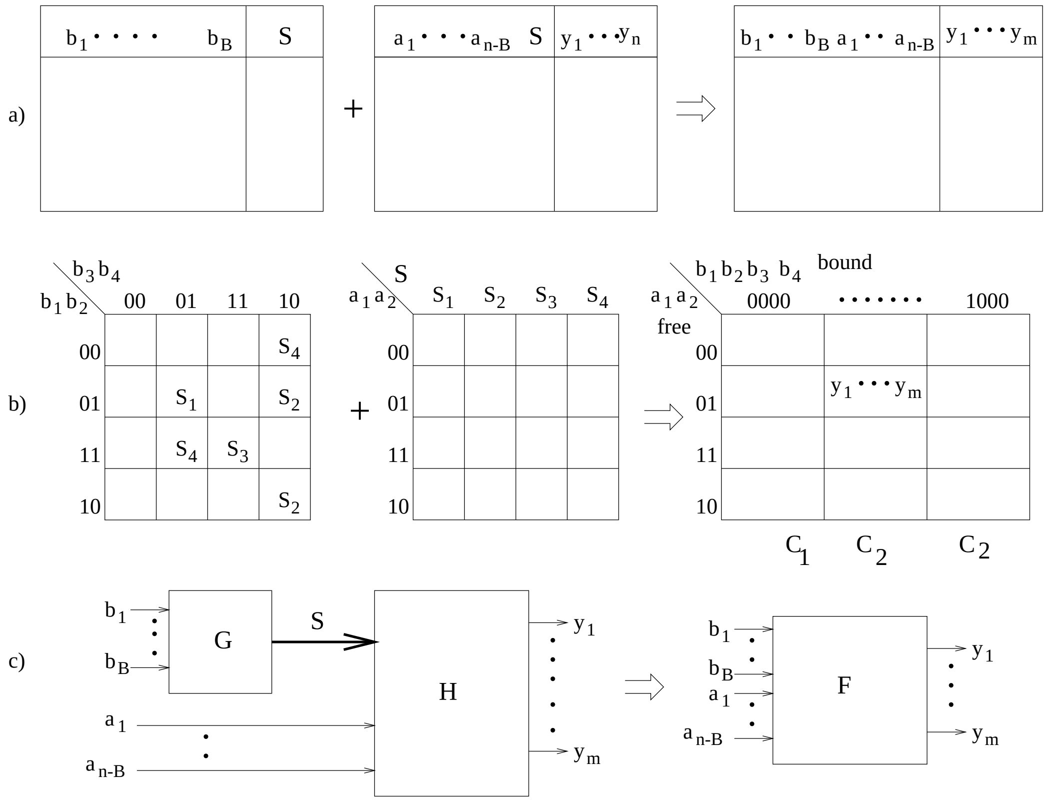


Figure 5: *The Decomposition Process.*

output and input encoding constraints to optimize both G and H.

It is noted that optimizing both the predecessor and successor logic will undoubtedly make one or both of the logic blocks less optimized. But this is the price for taking the simplification of both logic blocks into account. Our goal is to optimize the total complexity.

Figure 5 illustrates the process of the decomposition and the data generated in this process. Original logic function F from Figure 3a is decomposed to a predecessor block G and the successor block H from Figure 3b. In these figures, $b_1, ..., b_B$ are *bound variables*, $a_1, ..., a_{n-B}$ are *free variables*, and $y_1, ..., y_m$ are *output variables.*

The notation for sets of variables is:

X - the set of input variables,
A - the set of free variables,
B - the set of bound variables,
Y - the set of output variables,
V - the set of intermediate variables.

If sets A and B are disjoint, we will say that the decomposition is disjoint. $\mathbf{S} = \{S_1, ..., S_\mu\}$ is a *set of symbols* between the predecessor and successor blocks. The number of symbols is equal to the multiplicity index μ for given Bound Set $\{b_1, ..., b_B\}$. The symbols can be also treated as the values of a multi-valued variable S. After encoding, these symbols are encoded with *intermediate variables* $v_1, ..., v_g$ (Figure 4c). In case that a minimum number of intermediate variables (partitions) is taken, it holds: $g = log_2(\mu)$. The functions realized by these blocks can be represented by tables or array of cubes. We will illustrate them by Karnaugh maps for dydactic reasons. Figure 5b (right part) presents the "initial table" with free variables as rows and bound variables as columns. Columns correspond to the cofactors of the bound set. They are denoted by C_i, $i = 1, ..., t$. In each cell of the "initial table" there is a vector of output variables $y_1{}^{\delta_2}, ..., y_m{}^{\delta_m}$, where $y_j{}^{\delta_j} \in \{0,1,X\}$ denotes a value of output variable y_j. Figure 5a presents the encoding process with a pair of two tables: the first table, called the "predecessor table", corresponds to the predecessor block with encoded output and the bound variables in the input. The output symbols of G become the input symbols (columns) of the table of function H, called the "successor table". Other inputs to H are the free variables that correspond to rows of the table for H. In each cell of the "output table" there is a vector of output variables $y_1{}^{\delta_1}, ..., y_m{}^{\delta_m}$, where $y_j{}^{v_j} \in \{0,1,X\}$ denotes a value of output variable y_i. The figure shows how the data from the subfunctions G and H are composed together to create the initial function F. As we see, selecting good codes for input symbols S_i will simplify the map of the predecessor function G. It will cause also the successor function H to depend in a simpler way on symbols from S and free variables. Thus, having a partition of input variables, one can evaluate its influence on the complexity of G and H simultaneously.

Fig. 5b illustrates Figure 5a on a particular example. Fig. 5c presents the scheme of decomposition into blocks corresponding to Figures 5a,b.

The encoding process is done to concurrently optimize the logic of both G and H. The assignment and encoding methods from the literature were created for either 2-level realization of logic, or for logic realized with algebraic factorization. Our ideal solution here would be to find an encoding method that would optimize exactly the entire set of logic blocks realized with the decomposition method. This would be, however, very difficult, so only approximate solutions will be proposed.

2 RELATED PREVIOUS WORK ON LOGIC ENCODING AND STATE ASSIGNMENT FOR FSMS

Recently, several functional decomposers have been proposed. They include decomposers for Ashenhurst, Steinbach and Curtis (Roth-Karp) decomposition types. It is only the last type that requires the encoding process. The encoding problem is usually solved either by random coding, or by very fast, low quality encoding programs, optimized for the speed of the decomposition process only [8, 48, 57, 56, 50, 49]. Very few of the papers about decomposition discuss the encoding problem and propose efficient approaches to it.

Our functional decomposer, MULTIS, is being created for Machine Learning application, so our interest is particularly in functions with *very many don't cares.* This non-standard type of functions imposes new conditions on the encoding procedure as well. It must be observed, however, that in the decomposition process, when functions of more variables are gradually decomposed into functions of less variables, these smaller functions become more and more specified, since every column minimization process brings reduction in the percent of don't cares. The entire system should then have encoding methods for both: strongly unspecified and slightly unspecified functions.

One of our initial observations was that there is a similarity of the state machine realization problem and the functional decomposition problem. In state machine realization one first minimizes the number of internal states, which in case of incompletely specified machine reduces to the covering/closure problem. Next one finds the state assignment, that is the input-output encoding problem to simplify the excitation logic. Similarly, in functional (Curtis) decomposition one first solves the "column minimization problem" to find the smallest column multiplicity index μ, and next the output-input encoding problem should be solved for the combined compatible columns in order to simplify the logic of the predecessor block and the logic of the successor block. Concluding, many other analogies between the FSM state assignment and encoding for decomposition problems exist, and we hope that they can be used to create good decomposers. We will present some of these analogies in this report.

One of the processes traditionally used to verify and optimize hardware-realized state machines with compatible states and don't cares is *state minimization of FSM.* It was also shown by Pager [37] that the problem of determining the minimal closed table for a compiler which uses Knuth's LR(k) parsing algorithm can be reduced to that of finding the minimal closed cover for an incompletely specified flow table. Such tables have usually high percent of don't cares. We will show, that part of our problem is similar.

The problem of "state assignment of FSM" can be solved during the process of state minimization of an incompletely specified Finite State Machine. One of stages of this process is when the groups of closed and complete compatibles have been already found. As a standard, one just combines states from a compatible group to a single state, selecting randomly when there are many choices as the result of the groups' overlap. However, it can be observed that smart selection of some particular next states from a group of compatible next states can lead to better state assignments at to simpler final machine created using the closed and complete set of compatibles. This *selection problem* was first introduced by Perkowski et al [41]. His method was based on selecting those next states that simplified the state assignment. The states were selected in order to create columns or rows of the same state symbols, equal columns (rows) or some other "simple" areas of the transition table to simplify the assignment of internal states in the subsequent state assignment process. Another approach to this problem, called *mapping*, has been introduced recently by Hachtel et al [21]. This was shown to lead to a very fast minimizer that does not however lead to minimal solutions. Finally, it can be observed that the mapping problem can be reduced to that of minimizing a *Boolean relation* [13], but because such approach seemed inefficient, it has been not yet programmed.

Another idea in FSM design is that of a *concurrent state minimization and state assignment.* The idea was originally proposed for approximate design of asynchronous machines by Hallbauer [22]. It was programmed and was applicable to large machines, but its quality (evaluated on small machines) was poor. The first exact algorithm of this type for synchronous machines was presented in Lee and

Perkowski [31]. Based on partition theory and using the branch-and-bound technique, it minimized the total number of arguments in the excitation functions (each function was separately minimized). Although the program gave high-performance, logic optimized solutions, the computer implementation was limited to 8 $\times$ 8 tables. In the method from [31] the state minimization is a by-product of state assignment of a non-minimal table. The constraint of such approach is that overlapped compatibles cannot be created, which reduces the space of solutions. The state minimization was considered together with state assignment by Kannan and Sarma [26], but they just used separate programs in a sequence. The fastest dedicated method for concurrent state minimization and assignment is due to Hachtel et al [21]. which is based on a special cost function and mapping. In their method the state assignment is a by-product of state minimization in the sense that the mapping problem is solved approximately to anticipate its effects on heuristic encodeability measures. The method is therefore not exact. Another approach to this problem was recently presented by Avedillo and Huertas in [5].

Concluding, it is important to study all concurrent minimization/assignment approaches for FSMs, where both processes will not only be by-products of one another, but the entire space of FSM design would be truly searched in some efficient way. All these methods would lead by analogy to the development of better encoding procedures for functional decomposers as well.

3 BASIC DEFINITIONS

It is assumed that the reader is familiar with the basic terminology from [44].

Definition 3.1 *A group of (multi-output) columns ($C_i, ..., C_j$) of function F is called the* **compatible group** *(CG) (a* **compatible**, *for short), if and only if, under every row r of the Karnaugh map ($1 \leq r \leq na$) the entries $r, i, ..., r, j$ corresponding to all the input columns from this group ($C_i, ..., C_j$), are consistent bit by bit. (Symbol 0 is consistant with symbol 0 and with '-', symbol 1 is consistant with symbol 1 and with '-').*

Based on Definition 3.1 we test the column compatibility conditions. Two input columns (cofactors of bond set) are compatible under the condition that every two corresponding outputs must be either the same or one or both of two bits of outputs are '-'s. This rule must be applied to every two corresponding bits of these two tested outputs.

Definition 3.2 *A set of compatible groups of columns is said to* **overlap** *if the same elements, i.e. columns, appear in different groups of this set.*

One popular idea of *column minimization method* is to minimize the overlap. This retains more don't cares after column minimization, which allows for better results during symbol encoding and further decomposition processes. On the contrary, some other methods use the overlap in order to improve the encoding by the process of symbol selection (section 6).

Definition 3.3 *A set of Compatible Groups of Columns of function F is said to satisfy the* **completeness condition,** *if each column (in a table of the initial function F) is contained in at least one group of this set.*

Definition 3.4 *A* **feasible solution** *to Column Minimization Problem is a set of compatibles that satisfy the completeness condition. In the process, each compatible is uniquely denoted by a symbol S_i (for simplification, we will often use capital letters to denote symbols S_i).*

From now on, by S_i we will denote the symbol of a group CG of compatible columns.

Definition 3.5 *A* **quality** *of a compatible group, CG, is defined by Q, the number of its elements.*

Definition 3.6 *A feasible solution that has the lowest cost (the smallest number of CGs), CF, is called the* **exact solution to Column Minimization Problem.**

Having any feasible solution, one can find a set of two tables (predecessor and successor tables), functionally equivalent to the initial table.

If the solution is exact, the number of symbols, which is also the number of columns of function H, is the minimal. The number of symbols is also the number of output states of function G. Comparing to state machines, the symbol is a counterpart of the internal state of the machine.

4 THE COLUMN MINIMIZATION PROBLEM

Standard approach, applied also as one of variants in MULTIS is to first minimize the number of columns (by set covering, or by set partitioning) and replace the compatibles from the cover (or the partition) with the symbols. Next these symbols are encoded. The standard process is then:

1. Column Minimization to groups of columns called symbols.

2. Encoding of symbols.

Column minization is solved by Set Partitioning or Graph Coloring techniques. These approaches produce disjoint columns in each symbol.

Column Minimization can be also solved by the Set Covering or Maximum Clique Algorithms. These approaches produce non-disjoint sets of columns in a symbol.

The *modified* standard process is to follow the Symbol Encoding with the Column Encoding, while the codes for columns are obtained by modification to the codes of symbols.

The modified standard process is:

1. Column Minimization combines columns to groups of columns called symbols.

2. Encoding of symbols with binary vectors.

3. Encoding of columns based on the encoding of symbols: the codes of the columns corresponding to a symbol are modified.

Finally, the new approach to Column Minimization introduced in this report is to modify the Set Covering or Maximum Clique Algorithms. In the modified algorithms some columns obtain higher probabilities to be merged into final groups of compatibles.

In the following subsections we will briefly present the Set Covering and Clique Partitioning problems. Other approaches to this problem can be found in our Survey paper and our Report paper [44].

4.1 Set Covering

This section will discuss the basic theory of set covering and cliques. It is not the intent of this section to present new research results. Rather, this section will make use of existing algorithms to illustrate the theory of set covering and cliques.

The problem of set covering is an NP-hard one. Oftentimes set covering problems are small in the sense that the covering matrices are small, but they still are difficult to solve because the search space is very large. Thus, set covering applications need to have some smart algorithms and heuristics to limit this search space as much as possible while still obtaining a close-to-optimal solution. The section below will explain the covering problem. The section following, titled *A Set Covering Algorithm*, will demonstrate how set covering has been implemented in algorithms by using the branch-and-bound algorithm as an example.

The set covering problem is based on the following mathematical definitions.

Definition 4.1 *Let X be a set and Y a collection of sets. An element y of Y* **covers** *an element x of X, or x is covered by y, if x is a subset of y. A subset Y' of Y covers a subset X' of X, or X' is covered by Y', if X' is a subset of Y'.*

Definition 4.2 *Let Cost be a cost function defined on Y, i.e., a function from Y into the set of positive real numbers. For any subset Y' of Y, we define $Cost(Y')$ as the summation of the costs of all of the elements in Y'.*

Definition 4.3 Set Covering Problem: *Let X and Y be two sets, and Cost a cost function defined on Y. The set covering problem consists in finding a minimal cost subset of Y that covers X.*

The mathematical definition of set covering, while exact, is not easily understood or visualized. Thus, it is often advantageous to talk about a set covering problem with its covering matrix. The *covering matrix* of a set covering problem is a matrix with rows labeled by elements of X (the *set to be covered*) and columns labeled by elements of Y (the *covering subsets*). A position in the matrix at location $[x,\ y]$, where x is an element of X and y is an element of Y is a 1 if subset y covers x. The set covering problem solved in this form consists of finding the set of columns in the covering matrix that covers all rows in the covering matrix and minimizes the cost. The following example excerpted from [35] will help illustrate the covering matrix approach, see Fig. 6.

				Y		
		y1	y2	y3	y4	y5
	x1	1	1		1	
	x2		1	1		
X	x3				1	
	x4		1			1
	x5	1		1		
	x6			1		1

Figure 6: *A Covering Table*

Assuming that all columns have the same cost, the minimal solutions are $\{y2,\ y3,\ y4\}$ and $\{y3,\ y4,\ y5\}$. Each of these solutions covers every element in X (all rows in the above matrix). Other solutions exist (such as $\{\ y1, y3, y4, y5\}$), but the other solutions have a cost of 4 columns or more while the optimal solutions cover set X using only three columns.

		Y				
		y1	y4	y3	y2	y5
	x3	1				
	x2		1			
X	x5			1		
	x4				1	
	x1					1

Figure 7: *A Partitioned Covering Table 2*

There are a few simple rules that can be used to simplify the covering problem before beginning an in-depth search for the optimal solution. They are: *partitioning, essentiality, dominance,* and *Gimpel's function.*

If the rows and columns of the covering matrix can be permuted in such a fashion that 1's appear only in the diagonal of the covering matrix then the minimal solution of the original problem is the union of all columns that contain a 1. Thus, in Figure 7 the minimal solution is $\{y1, y4, y3, y2, y5\}$.

Essentiality involves identifying the columns that must be present in the minimal solution. This involves identifying any columns that contain an element of x that is not present in any other column. Such columns are essential. They are included in the minimal solution and are removed along with any rows they covered.

A row x is *dominated* if covering another row x' would necessarily cover x. A column is dominated if it covers a subset of the elements x covered by another column y' and the $Cost(y) < Cost(y')$. Dominated rows and columns are removed from the covering matrix. The minimal solution to the original problem can now be found by solving the reduced covering matrix for its minimal solution.

Gimpel's reduction is used when two columns $y1, y2$ (each with the same cost) cover a row x. In this case, column $y1$ and all rows covered by $y1$ and $y2$ are removed. Finally some new rows are added to the covering matrix according to the rules of Gimpel's reduction (not presented here.)

After using the above techniques, the covering matrix has likely been reduced to a simpler matrix than was originally formulated. The effect of the reduced size of the covering matrix in solving for the minimal solution is that the reduced matrix will have a considerably smaller search space. This results in greater efficiency in the calculation-intensive search for the optimal solution.

4.1.1 Branch-and-Bound Set Covering Algorithm

The *branch and bound algorithm* consists of splitting a covering matrix $C(X, Y)$ into two covering matrices C_l and C_r. A single column y is picked through some smart heuristics (not presented here). C_l is the covering matrix that is built assuming that y is in the minimal solution and C_r is the covering matrix that is built assuming that y is not in the minimal solution. Now covering matrices C_l and C_r are recursively split again into two new covering matrices by choosing a new column y. When each split occurs, a new y element is bound and included in a solution (not necessarily the minimum solution). The process for the branch and bound algorithm is illustrated below for two iterations:

Assume a covering matrix has been minimized by partitioning, essentiality, dominance, and Gimpel's function. The reduced covering matrix is assumed to have rows $x1, x2, x3,$ and $x4$ and columns $y1, y2, y3,$ and $y4$.

Branch and bound example: assume through some heuristic column $y2$ was chosen to be bound in

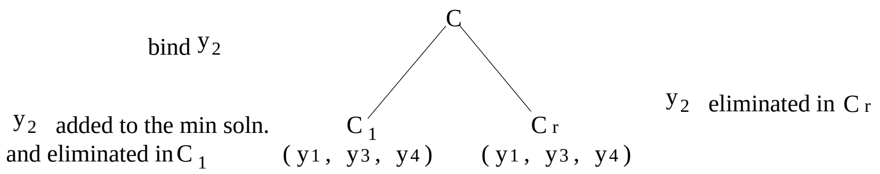


Figure 8: *Creation of new covering matrices*

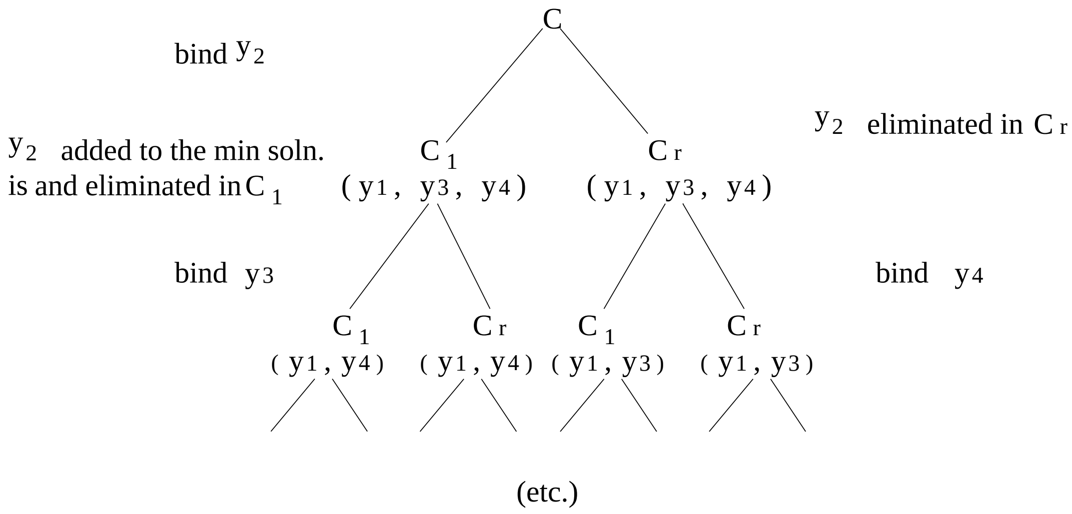


Figure 9: *Tree Search for Covering Table*

the first iteration. Thus, two new covering matrices are created as in Figure 8.

Assume that on the second iteration, y_3 is bound in C_l and y_4 is bound in C_r. The resulting tree looks like in Fig. 9.

The branch and bound algorithm thus searches the entire search space by intelligently picking one column in a covering matrix to bind and then recursively solving the two smaller covering matrices. The effect is that in each iteration one more column is bound and included in the solution. The included column is then eliminated in the subsequent covering matrices making them simpler to solve. This algorithm, like many other problem decomposition algorithms, reduces the complexity of finding the optimal solution by reducing the size of the search space by one variable at a time.

It should be noted that as it was stated above, the branch and bound algorithm is not very efficient. By halting recursion when a solution becomes less optimal than the currently most minimal solution this algorithm can be improved. There are also a number of additional pruning techniques that have been developed to further reduce the necessary search space although they will not be presented here.

4.2 Clique Partitioning and Generalized Clique Partitioning

4.2.1 Definition of Cliques

In the method of cliques, it is desired to express a problem to be solved as a graph. This graph is generated as a number of nodes with edges connecting each node to another node. Each node may be a data point, variable, etc. The edges are formed to represent some sort of relationship between nodes. This relationship between nodes varies by a problem. Take the following graph from Fig. 10 for example:

The nodes in the above graph are represented by the numbers 1, 2, 3, 4, and 5. The edges between nodes represent that those nodes have a relation or interconnection with each other. The edges in the

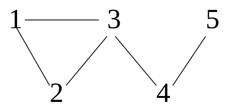


Figure 10: *Graph to illustrate cliques*

above graph are listed below by the numbers of the nodes that they go between:

1,2
1,3
2,3
3,4
4,5

The idea of the clique method is to find a minimum number of subgraphs that entirely describe the relationship between the nodes. This problem is called the *Clique Partitioning Problem* and is mathematically formulated as follows.

Let G be a graph consisting of a finite number of nodes and a set of undirected edges connecting nodes. A nonempty collection C of nodes of G forms a complete subgraph if each node in C is connected to every other node in C. A complete graph C is said to be a clique with respect to G if C is not contained in any other complete graph in G. The clique-partitioning problem is to partition the nodes in G into a number of disjoint clusters such that each node appears in one and only one cluster. Furthermore, each of these clusters itself forms a complete graph [51].

The cliques are found in practice by applying one of a number of algorithms. It should be noted that finding the cliques of a graph G by checking all combinations is an NP-hard problem and thus a smart algorithm is required to efficiently solve the problem. The algorithms often group the nodes according to a cost that is assigned to the edges that connect the nodes.

The generalized clique-partitioning problem is formulated as follows.

- The subgraph in which pairs of nodes have the best profit measure is reduced first.
- Then the pairs of nodes having the next level of profit measure are collected and reduced.
- Repeatedly applying the procedure to the other subgraphs, the process is stopped when a subgraph of a specified category or the original graph G becomes empty, [51].

An example is included in the next subsection that will help clarify the clique method.

4.2.2 Clique Partitioning Algorithm

A basic clique-partitioning algorithm, [51], takes advantage of the neighbor property. Let us consider an undirected graph G with nodes represented by integers. An edge is labelled by the two nodes that it goes between with the lower integer being named first. For example an edge going between nodes 3 and 4 would be called (3,4). Two nodes connected by an edge are called *neighbors*. A common neighbor is a node that is connected by edges to two other nodes which are neighbors. For example in the graph below, Fig. 10, nodes 1 and 3 are neighbors. Node 2 is a common neighbor of the pair of nodes 1 and 3 because both nodes 1 and 3 have an edge connecting them to node 2.

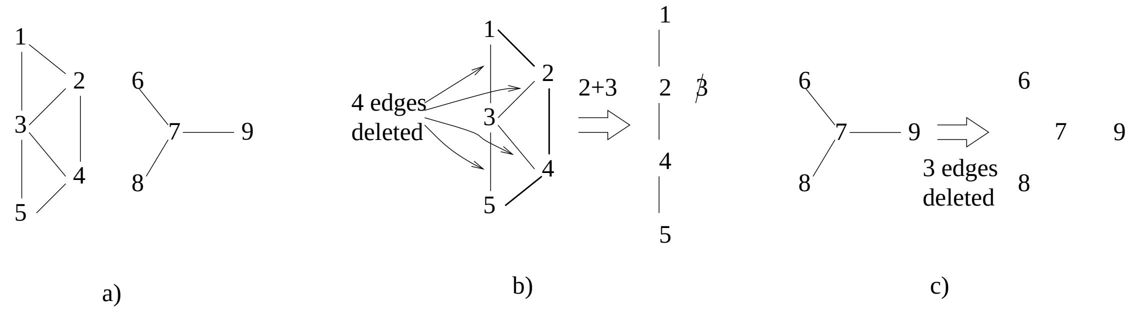


Figure 11: *Graph 2 to illustrate the concepts of the Clique Partitioning Algorithm*

In the neighbor algorithm, two lists are recorded. One is the *Nodes_List* and the other is the *Edges_List*. The list of nodes is a two-dimensional data structure in which each horizontal line represents a clique. At the start of the algorithm, each node occupies its own horizontal line and there are a number of cliques of size 1 equal to the number of nodes. During the algorithm, when nodes are combined to generate cliques of greater size, each horizontal line of the node list will contain more than one node and the number of vertical lines will be reduced as smaller cliques are merged into larger cliques.

The *Edges_List* is a sorted list of all edges in the graph. It is also a two-dimensional data structure. All edges with the same left node are arranged on the same horizontal line with the corresponding right node of each edge sorted in increasing order. For example, in the above graph, the two edges (1,2) and (1,3) exist. In the Edges_List these edges would be recorded as 1 2 3 for left node equal 1 and two right nodes equaling 2 and 3.

Algorithm

1. Scan through the list of edges (*Edges_List*). For each $(i, j), i < j$, compute its number of common neighbors.

2. Pick the edge (p, q) which has the maximum number of common neighbors. Combine the lists of nodes headed by p and q. The smaller one of p and q is used as the head of the resulting clique. Update the list of edges of G (when nodes are combined the edges connecting them are eliminated). If the list of edges is empty, the graph partitioning is completed.

3. Assume p is the head of the resulting clique. Pick an edge which joins node p and other nodes and has the maximum number of common neighbors. Let the edge be (p, r), or (r, p) if r is smaller than p. Save r, or p if r is smaller than p. Update the list of edges in G. If node p (or r if r is smaller than p) no longer appears in *Edges_List* then go to Step 2 and start to collect the next cluster. Otherwise repeat step 3.

For example, consider the following graph from Fig. 11a. The stages of execution of the Algorithm are illustrated in Figure 11b,c.

The sorted edges of this graph along with their corresponding number of common neighbors and the number of edges that must be deleted to combine the nodes in the edge are shown in Table from Figure 12.

It is desired to select the edge that has the most number of common neighbors. If there is a tie, then of those edges, the edge with the fewest number of edges that must be deleted is selected. If this is again a tie then one of the edges is arbitrarily selected. In the above example, edges (2,3) and (3,4) have the same number of common neighbors and the same number of edges that must be deleted. Assume

First Iteration:

Edge	# of Common Neighbors	# of Edges that Must be Deleted to Combine Nodes
(1,2)	1	3
(1,3)	1	4
(2,3)	2	4
(2,4)	1	4
(3,4)	2	4
(3,5)	1	4
(4,5)	1	3
(6,7)	0	3
(7,8)	0	3
(7,9)	0	3

Figure 12: *Table of Edges*

Second Iteration:

Edge	# of Common Neighbors	# of Edges that Must be Deleted to Combine Nodes
(1,2)	0	2
(2,4)	0	3
(4,5)	0	2
(6,7)	0	3
(7,8)	0	3
(7,9)	0	3

Figure 13: *Second Iteration in Table of Edges*

that in the second iteration, nodes 2 and 3 are selected to be combined. This results in the following reduced set of edges from Table from Figure 13.

Notice that the edges that are no longer necessary after nodes 2 and 3 are combined are deleted from the edge list. Repeatedly applying this method to the rest of the nodes in the graph results in the following cliques: $\{1,2,3\}$, $\{4,5\}$, $\{6,7\}$, $\{8\}$, and $\{9\}$.

Concluding, in this section we introduced few methods for set covering and clique partitioning problems. Similar methods have been implemented in MULTIS and we are experimenting with their different variants. The methods differ in their approaches, but basically can be programmed using similar techniques. The set covering algorithm was based on recursively binding progressively more and more sets y until the covering matrices were reduced to a trivial solution (empty). In this algorithm the sets were bound one by one in each iteration in order to generate the minimal solution. In the clique method, the nodes were paired two at a time. This process was continued until the list of edges became empty.

5 THE SYMBOL ENCODING PROBLEM

This section describes a high-performance algorithm for the Symbol Encoding Problem in Decomposition. (The method finds the encodings only for the already column-minimized initial tables.)

This method can find application whenever a very high quality circuit is looked for, in a given realization technology, but when the function is of a rather small size.

The branch-and-bound optimum algorithm based on multi-valued minimization of the successor function and Boolean minimization of the predecessor function will be presented.

5.1 Basic Assumptions of the Symbol Encoding Method

The next design stage after column minimization is the encoding of the symbols. The graph embedding approach [2, 9], the "column-based" approach [16], and the partition theory [54, 23, 28, 52] are the best known state encoding methods that were applied in the past for state machines, and these methods give inspiration to create new algorithms for Symbol Encoding in the case of Functional Decomposition.

It is the conclusion based on the considered by us examples that the classical FSM state encoding assumptions are not directly useful for our encoding problems. Similarly, the recently developed heuristic encoding methods (DeMicheli [11], Mustang of U.C. Berkeley), do not produce the high quality results. It is, therefore, necessary to look for new encoding methods for functional decomposition. The paper [31] had the same basic idea as the method presented in this section, however the proposed algorithms were too slow, mainly because of the inefficient formulation of some search problems. Below we try to improve on the weaknesses of these papers and also, we apply these ideas to the encoding for decomposition.

It is our opinion that the exact minimum solution can be obtained only when the methods based on the partition theory are used. We will, therefore, briefly review some necessary notions. First, it must be mentioned that many of these methods generate the minimum solution assuming the minimization of the number of intermediate variables. In such a case, for K symbols the partitions in a total number of k are selected, where $2^k - 1 \; < \; K \; \leq \; 2^k$. This leads to the concept of *proper partitions,* defined as the two-block partitions in which the number of symbols in the larger block does not exceed 2^{k-1}. The partitions will be, therefore, selected among the set TP of proper partitions. The final family (set) of partitions, TF, is defined as such a family, that the partition product of all partitions from that family is a zero partion. *The optimum family*, TO, is such a final family that minimizes the value of an assumed cost function.

There are two approaches possible here. The first approach is to assume only the minimum number of partitions, and hence the minimum number of intermediate variables. The argument is that less variables minimizes the DFC of the first-level decomposition. The second approach is to take any number of partitions, usually slightly larger than the minimum, in order to simplify the functions of the decomposed blocks. Although the DFC of the first decomposition may suffer, the functions are simpler, so that the next decompositions of both blocks will be able to recover from this temporary DFC increase, and would produce a final DFC with a smaller value for the total decomposed function. Only experiments will confirm which of these ideas will work statistically better on Machine Learning benchmarks. The family TO in the first approach is selected among the proper partitions, and in the second approach among all the two-block partitions. Let us denote the zero partition by $\overline{0}$: $\overline{0} \; = \; \{\overline{1}, \overline{2}, ..., \overline{K}\}$.

In all methods of symbol encoding the partitions are selected to TF in such a way that a unique (one-to-one) encoding is created.

$$\prod_{\tau_i \; \in \; TF} \tau_i \; = \; \overline{0} \tag{1}$$

In both approaches we look for such final family of partitions TF that the cost of encoding is minimized.

The problem of symbol encoding of the column-minimized table can be formulated as follows:

Given are:

- S - the set of symbols of the column-minimized function F; $K = CARD(S)$,
- $T2$ - the set of all proper (first appraoch), or two-block (second appraoch) partitions,
- the inital table of function F.

The set TO to be found is such a subset of the $T2$ set, that both of the following conditions are satisfied:

1.
$$\prod_{\tau_i \in TO} \tau_i = \tau_1 \cdot \tau_2 \cdot \ldots \cdot \tau_s = \overline{0} \tag{2}$$

   That is, the partition product of the selected partitions for intermediate variables v_1, v_2,..., v_s is a zero partition, which means that a different binary sequence is assigned to each symbol.

2.
$$min(CF_r) = min(\sum_{v_i \in \{v_i, \ldots, v_s\} = V^{TO}} q_r(v_i)^{TO} + \sum_{y_j \in \{y_i, \ldots, y_m\}} q_r(y_j(A, V^{TO}))), \tag{3}$$

   where:

   A is the set of free variables,

   V^{TO} is the set of intermediate variables for set of partitions TO,

   $\{y_1, \ldots, y_m\}$ is the set of output signals of function H (the same as for function F).

   $q_r(V)^{TO}$ is the real costs of the realization of the τ_i partition, as one of the TO set. The costs are calculated for the respective circuit realization for corresponding to τ_i intermediate variable v_i.

   $q_r(y_j)$ is the cost of realization of an output signal y_j, calculated assuming the given partitions for TO.

Let us assume that both functions G and H are minimized here assuming two-level logic realizations created for set TO of partitions. This is called the condition of the total cost function minimization.

The encoding methods discussed in sections 5.6 and 6 apply the heuristic search and the branch-and-bound principle [36]. This prevents us from searching the entire solution space, being in this case the set of all subsets of the set of proper partitions (the first approach) or two-block partitions (the second approach). A few heuristics are applied as a means for controlling the search strategy. Among them is the application of quality functions to evaluate partitions. However, in this approach the heuristics do not lead to losses of the minimum solutions (in section 8 the heuristics are described, that increase search efficiency but can omit the optimum). The quality function and the cost function are discussed in sections 5.4 and 5.5.

5.2 Operators and Relations on Cubes

Before we discuss the quality functions (denoted below by QF), used for the symbol encoding and for the joint column minimization and encoding (called also column encoding, see section 7), we need to define a number of operations on an array of cubes.

The names of the operators and relations arised by reason of their application in this report.

a) a_i / b_i

$\bigtriangledown$	0	1
0	0	1
1	1	ϕ

b) a_i / b_i

$\bigtriangleup$	0	1	X
0	0	X	ϕ
1	X	1	ϕ
X	ϕ	ϕ	X

c) a_i / b_i

$\lhd$	0	1
0	0	ε
1	ϕ	1

d)

$\sqsubseteq$	0	1	X
0	ε	ϕ	ε
1	ϕ	ε	ε
X	ϕ	ϕ	ε

Figure 14: *Tables of operators on cubes: (a) column compatibility, (b) row compatibity, (c) column absorption, (d) row absorption*

Definition 5.1 The operator of column compatibility , *denoted by* ∇, *is defined for binary cubes* a *and* b *(here* a *and* b *are n-tuples of elements* $a_i, b_i \in \{0,1\}$*), as follows:*

$$a \nabla b = \begin{cases} \emptyset & if (\exists i)[a_i \nabla b_i = \emptyset] \\ c = (c_1, c_2, .., c_n) & in\ the\ opposite\ case, \end{cases}$$

where
$(\forall i = 1, \ldots, n)[c_i = a_i \nabla b_i]$ *in accordance with the table, Fig. 14a.*

Examples:

$$1100 \nabla 0010 = 1110$$

$$0101 \nabla 0001 = \emptyset \quad \text{because} \quad a_4 \cdot b_4 = \emptyset$$

Definition 5.2 The operator of cube (row) compatibility, *denoted by* $\triangle$, *is defined for ternary cubes* a, b *(ternary cubes are n-tuples of elements* a_i, b_i $\{0, 1, X\}$*), as follows:*

$$a \triangle b = \begin{cases} \emptyset & if (\exists i)[a_i \triangle b_i = \emptyset\ or\ a_i \triangle b_i = X for\ more\ than\ one\ i,\ when\ a_i \neq X], \\ c = (c_1, \ldots, c_n) & otherwise \end{cases}$$

where:
$(\forall i = 1, \ldots, n)$ $[c_i = a_i \triangle b_i]$, *in accordance with table from Fig. 14b.*

Definition 5.3 The relation of column absorption, *denoted by* $\lhd$, *is defined for binary cubes* a *and* b *(*a_i, $b_i \in \{0,1\}$*), as follows:*

$$a \lhd b\ if\ (\forall\ i)[a_i \lhd b_i \in \{0,1,\varepsilon\}] \tag{4}$$

$$a \neg \lhd b \; if \; (\exists \; i)[a_i \;\; \lhd \;\; b_i \;\; = \;\; \emptyset] \tag{5}$$

where the operation $a_i \lhd b_i$ *is defined in accordance with the table from Fig. 14c.*

Examples.
0100 $\lhd$ 0110,
0100 $\neg\lhd$ 0010,

Definition 5.4 The relation of cube (row) absorption, *denoted by* $\sqsubseteq$ *is defined on ternary cubes a and b as follows:*

$$a \sqsubseteq b \; if \; (\forall \; i)[a_i \;\; \subseteq \;\; b_i \;\; = \;\; \varepsilon] \tag{6}$$

$$a \not\sqsubseteq b \; if \; (\exists \; i)[a_i \;\; \subseteq \;\; b_i \;\; = \;\; \emptyset] \tag{7}$$

where the operation $a_i \subseteq b_i$ *is defined in accordance with the table from Fig. 14d.*

5.3 Generation of Prime Implicants of Multi-Valued Functions

In this subsection we will deal with the generation of prime implicants for multi-valued functions.

We will start with the case in which it is assumed that the number of partitions is minimal, and thus only proper partitions are considered. To find the successor function from the successor table the code for each cell of the table is required. The code of each cell is the concatenation of the one-hot code of column and the cube code of row; however, the encoding is yet unknown, and hence so is the column's code. Our goal is then to select such an auxiliary code for columns, that will allow to check all the possibilities of minimization of the output functions (i.e., covering minterms with prime implicants) before finding the final encoding of the symbols of the table. The code will then make it possible to check all possibilities of joining groups of minterms into prime implicants, i.e., applying logic adjacency to minterms from every two columns. Suppose we have encoded the tables according to some proper partition τ_j. Several possibilities of including columns into prime implicants exist now. This results from the fact, that we assume the possibility of arbitrary permutation of the columns, such that the other partitions are assorted in the best possible manner with τ_j, to minimize the *multi-valued function*, described by a table with columns encoded with τ_j (the encoded tables will be called *transition tables*). To formalize such *generalized merging of columns into groups* we must find the respective code for the columns. For K columns of the successor table we select a code of length K, where 1 is written to the i-th column on the i-th position. For example three symbols are encoded as follows:

1 - 100,
2 - 010,
3 - 001.
We call it a *one-hot code*.

Actually, every cell of the successor table (i.e., a 0-cube) can be described by means of the binary string: $c^i \; = \; c^i_\alpha \; \bullet \; c^i_\beta$ where c^i_α is the successor's block free variables inputs cube in Gray code. and c^i_β is the cube of columns in the code described above, The symbol $\bullet$ stands for concatenation. The operations from subsection 5.2 may be utilized to merge the cubes into larger cubes. Both constituents of the string are used. There may be $n_1 \; < \; K$ arbitrary columns in each group after merging. The adopted code is the simpliest one, with which this property can be checked. All the cubes generated by means of joining have the part corresponding to the binary inputs, and the part corresponding to the symbols of columns. The implicants of the multi-valued function will be called *"generalized implicants"* (abbrev. *G-implicants*). The G-implicants which are prime will be called *"generalized* *prime implicants"* (*GP-implicants*). Let ON be the set of minterms and DC be the set of "don't-cares" of the Boolean function, which corresponds to the successor table, whose columns correspond to symbols and are encoded in accordance with the chosen partition τ_j. We shall find the set of all GP-implicants, and next the covering of minterms with the GP-implicants.

Let $F = (y^1, y^2, ..., y^i, ..., y^m)$ be a multioutput multi-valued Boolean function with n inputs and m outputs, where $y_i = y^i(x_1, ..., x_n)$, $i = 1, ..., m$ are the component single output functions. Any multi-valued minimizer such as Espresso-MV [47] can be used to generate good covering with GP-implicants. Another method to find GP-implicants would be to use the method of graph coloring.

5.4 Quality Function for Partitions

Quality function for the set of partitions $\{\tau_1, ..., \tau_r\}$ will be defined as follows:

$$QF(\{\tau_1, ..., \tau_r\}) = \sum_{\tau_i \in \{\tau_1, ..., \tau_r\}} q(\tau_i) \tag{8}$$

where $q(\tau_i)$ is the total cost of realization of the partition τ_i in predecessor and successor blocks. It is calculated as a sum of costs $q(y_i)$ for all output variables and costs $q(v_i)$ for all intermediate variables. Both these costs are calculated for the case of selecting the best additional partitions assorted with partition τ_i.

The cost q depends on the applied method of realizing the decomposed blocks. Below a few methods of estimating the q are presented, q^s stands for these methods, s = 1,2,3,4.

Notations applied:

- NL_i - number of literals in the i-th implicant,
- NL - total number of literals of all implicants NL $= \sum_{i=1}^{i=NT} NL_i$,
- NT - total number of implicants ,

It is the total number of inputs to the AND level gates and the OR level gates.

Calculation of the quality functions for partition τ involves the following steps:

Algorithm 5.1

Calculation of the quality functions for partition τ.
Given are symbolic predecessor and successor functions.
Begin

1. Encode the first block of τ partition with 1 and the second block with 0;
2. Given τ, find the multi-valued minimum cover of the successor function H using GP-implicants. Calculate the binary cost (cost of binary implicants) and the multi-valued cost (cost of GP-implicants). Calculate the weighted total cost of H.
3. Given τ, find the minimum cover of the predecessor function with standard binary implicants.
4. Encode the first block of τ partition with 0 and the second block with 1;
5. Execute steps 2 and 3 with the encoding from step 4.
6. Calculate the minimum of the two costs found in steps 2 and 4 - the cost is computed: $q^{min}(\tau)$.
7. Store in memory the encoding of the partition's blocks to symbols 0 and 1.

End

This algorithm can be also extended to concurrent minimization of functions G and H. We adopt the following notation:

$$\tau_{ij} \;=\; \{\; \overline{i,j},\; \overline{other\ states\ from\ A}\; \}$$

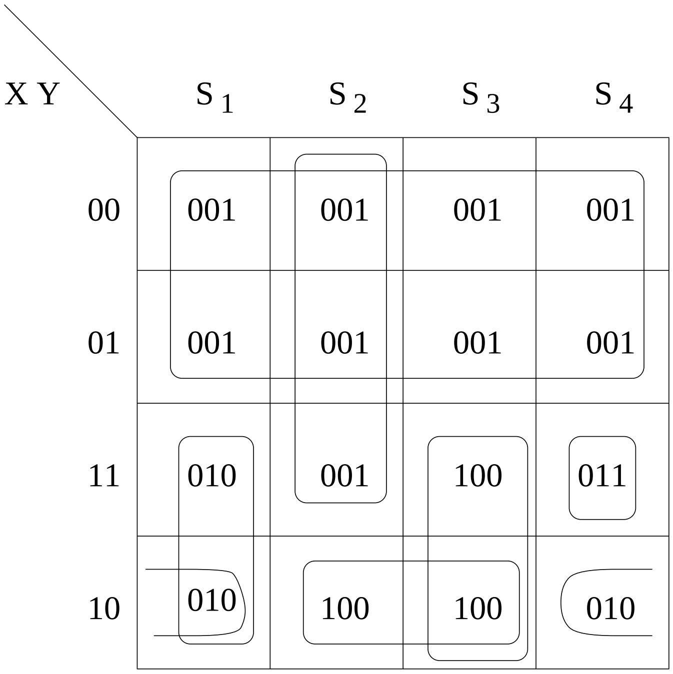


Figure 15: *Function H for Example 5.1.*

Example 5.1

Given is the function from Fig. 15
$T2 = \{\tau_{12},\ \tau_{13},\ \tau_{14},\ \tau_1,\ \tau_2,\ \tau_3,\ \tau_4\}$.

The method of computing $q(y_i)$ for τ_{13} is presented in Fig. 16. For simplification, only assignment of 1's to subscript block of τ is shown. Also, only function H is evaluated. The method can be, however, easily expanded to both block encodings and to the simultaneous evaluation of function G.

Function $y_1 = x\ \bar{y}\ \tau_{23}\ +\ x\ \tau_3$.
Function $y_2 = x\ \bar{y}\ \overline{\tau_{23}}$.

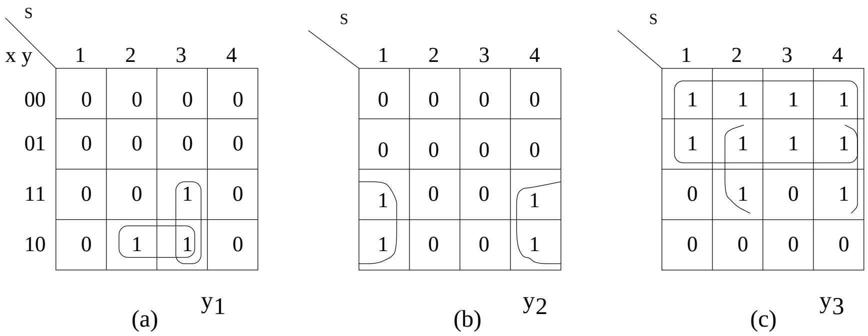


Figure 16: *Calculations of costs of partition τ_{23} for functions y_1, y_2, y_3*

Function $y_3 = \bar{x} + \tau_{24}\ y$.

Depending on the realization method, partition τ_3 can be realized as a separate partition, or as $\tau_{23} \cdot \overline{\tau_{24}}$. The total cost of partition τ_{13} is thus calculated for functions y_1, y_2, y_3.

Similarly, the costs are calculated for all possible partitions from $T2$.

5.5 Cost Functions for Final Families

The cost function for final family of partition T is defined as follows:

$$CF_r(TF) = \sum_{v_i \in V} q_r(v_i) + \sum_{y_j \in Y} q_r(y_i). \tag{9}$$

Functions q_r are computed in the same fashion as the q functions. The difference lies in the methods of minimization of the predecessor functions and the successor functions. In the quality function every partition was evaluated separately, on the basis of the minimum GP-implicant covers, found for the separate functions. Similarly, every function was computed for the best code found for it.

The cost functions CF_r, however, may be computed either separately or jointly. If the cost functions are computed separately for each of the functions and the functions are realized separately, then Algorithm 5.1 (and Algorithm 7.1) guarantee that the minimum solutions are found (see comment 2 to Algorithm 5.4). Cost function CF_r may be also computed for the jointly minimized intermediate and output variables for the given family of partitions. Hence there may be common implicants - there exists then a possibility of getting better solutions than those resulting from the assumptions taken previously. However, for this class of solutions there is no assurance that the optimum solution is found.

In what follows we will apply the following notation for the jointly computed cost functions:

${CF_r}^{2j}$ - corresponds with the number of inputs in AND and OR gates,

${CF_r}^{3j}$ - corresponds with the number of literals in the functions, at the same time in repeated implicants the literals are counted one time only.

5.6 Algorithm of Encoding for Column-Minimized Tables.

5.6.1 Algorithm of Encoding

Encoding of column-minimized tables is performed using the following algorithm. The input data are symbolic functions G and H.

Algorithm 5.2

Begin

E1: Find $T2$ family of all two block partitions using Algorithm 5.3. $T2$ family is stored as a list with name $T2$. This algorithm utilizes breadth-first tree search. Hence it arranges the $T2$ list in such an order that it starts with partitions of blocks with the same sizes of blocks, or with blocks whose lengths differ at most by one, and which terminate with partitions having the longer block of length $K - 1$;

E2: (Option) Sort the $T2$ list according to some criterion of partitions quality. One of the methods is to sort the $T2$ list in such a way, that among the partitions of the same length of the shorter block, the partition with a smaller value of quality function q is placed first. The values of quality functions are calculated by Algorithm 5.1;

E3: Find families (or a single family - in accordance to the value of the user's parameter ALL) of partitions $TO \subseteq T2$ by the means of Algorithm 5.4.

The families must satisfy the following conditions:

a) condition of code uniqueness

$\prod_{\tau_j \in TO} \tau_j = \overline{0}$ product is the zero partition.

b) condition of the cost function minimization:

$$min(CF_r) = min(\sum_{v_j \in V} q_r(v_j) + \sum_{y_i \in Y} q_r(y_i)) \tag{10}$$

End

As a part of this algorithm the Boolean (multi-output) functions of blocks G and H are found.

The algorithm uses a tree search in solution trees. The states (nodes) of the tree are the vectors of coordinates. Each vector for state with number N is denoted as

$$V(N) = (C_1(N), C_2(N), ..., C_r(N)) \tag{11}$$

where C_i, $i = 1, ..., r$ are the coordinates. New states are created from the old ones with use of *operators.*

By NN we will denote the number of the currently created new state of the tree, being the successor of state with number N. By $A \setminus \{b\}$ we denote set (list) A with removed element b. The coordinates have the following meaning:

- $QS(N)$ - part of generated partition,
- $GS(N)$ - set of operators of node N.

The operators correspond to the symbols. Generation of $T2$ family of partitions is done by Algorithm 5.3.

Algorithm 5.3

Begin

1. *Creation of the initial state of the solution tree:*
   $N := 0$;
   $V(0) = (QS(0), GS(0)) := (\{\}, A)$;
   $T2 := \{\}; OPEN := \{V(0)\}$;
2. If $OPEN = \{\}$ then return;
   $V(N) = (QS(N), GS(N)) :=$ first element from list $OPEN$;
3. if $GS(N) = \{\}$ then
   begin delete first element from list $OPEN$; go to 2 end
4. $s :=$ first element from $GS(N)$;
   $QS(NN) := QS(N) \cup \{s\}$;
   $GS(NN) := GS(N) := GS(N) \setminus \{s\}$;
5. add $(QS(NN), A \setminus QS(NN))$ to the beginning of list $T2$.
6. Add new state $NN = (QS(NN), GS(NN))$ to the end of list $OPEN$;
7. Go to 3;

End

Comments.

1. When Algorithm 5.3 terminates the $T2$ list is a $T2$ family.
2. The two-block partition $\tau \in T2$ is represented with the single computer word e. Bits 1 of the word correspond to the elements of the first partition block. Then $\overline{e}$ (the word e after bit by bit negation) corresponds to the second block of the partition. Therefore $\tau = \{B_1, B_2\}$, where $B_1 = \{a_i \mid e[i] = 1\}$, $B_2 = A \setminus B_1$. The partition may also be represented in the transitional form $\{e, \overline{e}\}$. Since we know the value of K then the B_2 block may be easily computed as the bit by bit negation of word e.

The set theory operations are reduced to the operations on the binary words, e.g. product of words, denoted by $\cap$, corresponds to the intersection of sets. Therefore $q(\tau)$ will be identified with $q(e)$. Product of partitions will be denoted by $\cdot p$. In fact Algorithm 5.3 and the following algorithms use words and arrays of binary words. In the expression: $\alpha_i \cdot e, \delta_i$ stands for a word with bit 1 in position i, and with the rest of the bits equal to 0. $GS(N)$ is the set of operators for node N, $QS(N)$ is a set of selected partitions for coding in node N, $PR(N)$ is the product of partitions from $QS(N)$. In this problem the operators are arbitrary partitions from $T2$.

Generation of TO family by means of $T2$ family is done as follows:

Algorithm 5.4

Begin

1. *Creation of the initial state of the tree*

   $(GS(0), QS(0), PR(0), QF(0)) := (T2, \{\ \}, \overline{1}, 0);$

   $CF_{min} := \infty$; SOLUTION $:= \{\}$; initialize list $COSTS$;

2. If $OPEN = \{\}$ then

   begin

   if $ALL = 1$ then print "optimum solutions", $SOLUTION$

   else print "optimum solution", $SOLUTION$;

   return

   end ;

   $V(N) = (GS(N), QS(N), PR(N), QF(N)) :=$ element from $OPEN$ list selected according to some search strategy (for instance the first element for the depth-first strategy);

   If $ALL = 1$

   then

   all minimum solutions are searched - delete from $GS(N)$ all such operators e that $CF_{min} < QF(N) + q(e)$

   else

   only some of the minimum solutions are sought - delete from $GS(N)$ all such operators e that $CF_{min} \leq QF(N) + q(e)$;

3. *Extension of state* $V(N)$

(a) If $GS(N) = \{\}$ then
begin
delete first element from list $OPEN$ and go to 2,
end

(b) e : = first element from $GS(N)$; $GS(N) := GS(N)$ - $\{e\}$;

(c) $PR(NN) := PR(N) \; \cdot p \; \{e, \overline{e}\}$;

(d) $GS(NN) := GS(N) \setminus \{e_i \in GS(N) \,|\{e_i, \; \overline{e}_i\} \; \cdot p \; PR(NN) = PR(NN)\}$;

(e) If $(GS(NN) = \{\})$ and $(PR(NN) \neq \overline{0})$ then go to 3; /* backtrack */

(f) $QS(NN) := QS(N) \cup \{e\}$;

(g) Calculate $q(e)$, add new item to $COSTS$
(this step is executed if $q(e)$ is not found in Algorithm 5.1.

(h) $QF(NN) := QF(N) + q(e)$;

(i) If $QF(NN) > CF_{min}$ then go to 3; /* backtrack */

4. /* *Checking of solution* */
If $PR(NN) = \overline{0}$ then /* *new solution found* */
begin
Calculate CF_r.
/* Cost functions $q_r(\tau_i)$ are computed for the same
assignment of the blocks to 0 and 1, as those, already
found in Algorithm 5.1 for the
best quality function $q(\tau_i)$; */
if $CF_r(NN) = CF_{min}$ then
begin
if all solutions are looked for $(ALL = 1)$ then add $QS(NN)$ to list $SOLUTION$
else $SOLUTION := \{QS(NN)\}$;
go to 3
end ;
if $CF_r(NN) < CF_{min}$
then
begin
$CF_{min} := CF_r(NN)$;
$SOLUTION := \{QS(NN)\}$
go to 3
end ;
end
else
begin
add new node to the tree

add $V(NN) = (GS(NN), QS(NN), PR(NN), QF(NN))$ to the beginning of $OPEN$;

go to 2;

end

end.

5.6.2 Discussion and Examples.

Comments to Algorithm 5.4.

1. If $QS(NN)$ is the TF family (i.e., if $PR(NN) = \overline{0}$) then the minimum realization of the block functions may be found, and then the real cost of function $CF_r(N)$ may be computed. However, while generating the successive states N_i of the solution tree we don't know the codes of the symbols, so we can't calculate the value of $CF_r(N_i)$. Therefore, to be able to apply the branch-and-bound rule we must have some function for each node that will be used for cutting off. It is useful that QF function can be used with that respect.

2. The value of quality function $QF(N_i)$ is the sum of costs of partitions selected from $QS(N_i)$ and the total cost of output signals realizations. (There is the assumption in the method that the block functions are realized and minimized separately). Therefore, for every $N \in S_k$ (i.e., $QS(N)$ fulfilling conditions of the problem) there is: $QF(N) \leq CF_r(N)$. For every node N_i in a branch leading to N: it holds $QF(N_i) \leq QF(N) \leq CF_r(N)$. If the cost of the last solution was CF_{min}, and in the following process of tree extension we find such node N_i, that QF(N_i) $> CF_{min}$, then the further development of every branch that begins with N_i may lead only to solutions QS(N') with values $CF_r(N') \geq QF(N_i) > CF_{min}$, so we can cut it off at node N_i.

3. If all the nodes for which it holds that $QF(N) \leq CF(N)$ have been already looked through, and we encounter the node $N_j \in S_k$, such that $QF(N_j) = CF(N_j)$, then $QS(N_j)$ is the optimum family of partitions TO. Cost $q(\tau_j)$ of every partition is calculated separately, assuming that all the remaining partitions will be chosen in the best way after selection of the partition assumed. From this assumption there results the possibility of arbitrary permutation of rows of the initial table, which in turn leads to the generation of "generalized prime implicants." The partition cost τ_i obtained in this way will then be at most equal to the realization cost of the i-th block function.

4. If functions are realized separately then Algorithm 5.4 gives a guarantee that the minimum solution is found, because in such case it holds for all nodes N in the solution tree that $CF_r(N) \geq QF(N)$, so that the cut-off nodes cannot include the optimum solution. In the case of common minimization the optimum solutions can be cut-off because there may be $CF_r(N) < QF(N)$.

5. For some variants of the method, the value $q(e)$ is calculated for the first call on the basis of Algorithm 5.1, and at the following calls it is found from the incrementally created list of pairs:

$$COSTS = ((e_{i1}\ q(e_{i1})),\ (e_{i2}\ q(e_{i2}))...). \tag{12}$$

6. Such a variant of the method results from the trend to find part of the solutions without generating $q(e)$ for all partitions (there may be plenty of them, and much time would be spend before generating any solution - that would preclude the algorithm from the ability to deal with large functions). In such case list $T2$ is not sorted in step $E2$ of Algorithm 5.2 with respect to the values of q functions, but according to some other quicker method (not involving calculations of q functions), or not sorted at all. $COSTS$ list is initialized as $\{\}$.

6 THE COLUMN GROUP SELECTION PROBLEM

In this section we will present two approaches to improve the quality of the Symbol Encoding method from the previous section at the prize of more complex processing.

The essence of the method is this. Normally only one symbolic code is assigned to a column. Let us call each column covered with more than one group of compatibles the *non-unique column.* It can be observed that for each such column there is a choice of a symbol. It is then reasonable to select a symbol that would simplify the encoding. The method presented here is quite similar to the one from the section 5, but it improves the quality of the solution at the price of slower processing. Algorithm 6.3 can be used as a preprocessor of **every** Symbol Encoding method.

6.1 Selection of Symbols based on Map Simplification

Example 6.1

For dydactic reasons, we will explain first the method on an example.

Fig. 17a presents an *initial function table.* The columns, i.e. the cofactors of the bound set {c,d,e}, are denoted by C_i, $i = 0, ..., 7$. Below the columns, their possible symbols are shown. The symbols are created by multi-coloring or any other method from section 4. For instance, column C_0 can be assigned with symbols A, B, or C. Column C_2 can be assigned with symbols A or C.

Figure 17b shows the enumeration of columns in the initial function map. The corresponding Symbolic Predecessor Relation Map is shown in Figure 17c. For each column of the initial function map (represented by the corresponding cell of the Symbolic Predecessor Map), the symbols that can be used for this column are shown in Figure 17c.

Now, the goal is to select in every cell of the Symbolic Predecessor Relation Map such single symbol that the encoding will be minimized. One can see that selecting A in all cells 000, 010 and 100 will allow to create large groups of A's, so it should simplify the encoding. This choice is shown in Figure 17d.

This new map from Figure 17d is now evaluated using various partitions. In our case we use all two-block partitions, since the table is small. For larger maps only a subset of two-block partitions (or a subset of proper partitions) is tried. (Those partitions are found using a method similar to the one from section 5. These two methods can be combined to evaluate each position based on complexities of both functions G and H.)

Suppose that partition $\tau_1 = \{\overline{AB},\ \overline{C}\}$ is tested and it is assumed that block $\overline{AB}$ is encoded with 1 and block $\overline{C}$ is encoded with 0. This is denoted by {Block1, Block0} $= \{\ \overline{AB}, \overline{C}\ \}$. In this example partition $\tau_1 = \{\overline{AB},\ \overline{C}\}$ has cost 3 - see Figure 17e, partition $\tau_2 = \{\overline{AC},\ \overline{B}\}$ has cost 13 - see Figure 17f, and partition $\tau_3 = \{\overline{BC},\ \overline{A}\}$ has cost 9 - see Figure 17g. Therefore, partitions τ_1 and τ_3 are selected as having the smallest total cost among all pairs of partitions whose product is $\overline{0}$. (it is easy to check that $\tau_1 \cdot \tau_2 = \overline{0}$).

The first method of selecting symbols is as determined by the following Algorithm 6.1:

Algorithm 6.1

1. For every cell of the Symbolic Predecessor Relation Map (with rows and columns corresponding to set B as in Figure) that contains more than one symbol select such symbol that the encoding is simplified - for instance: a row of the same symbols, a row of the same symbols, or a large implicant of the same symbols.

2. Evaluate this map using different partitions, using the method presented in section 5. Similarly to the method from section 5, for every symbolic map of G either all partitions are tested, or only some subset of proper partitions is calculated for speed improvement (for instance those that were good in previous encodings).

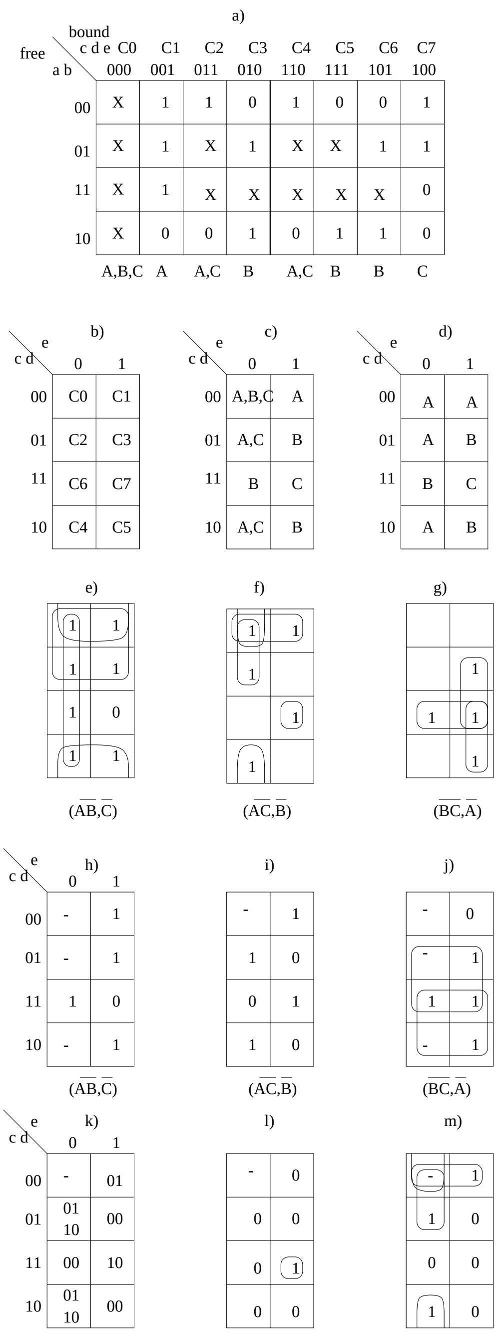


Figure 17: *The Column Group Selection and Encoding process to Examples 6.1 and 6.2.*

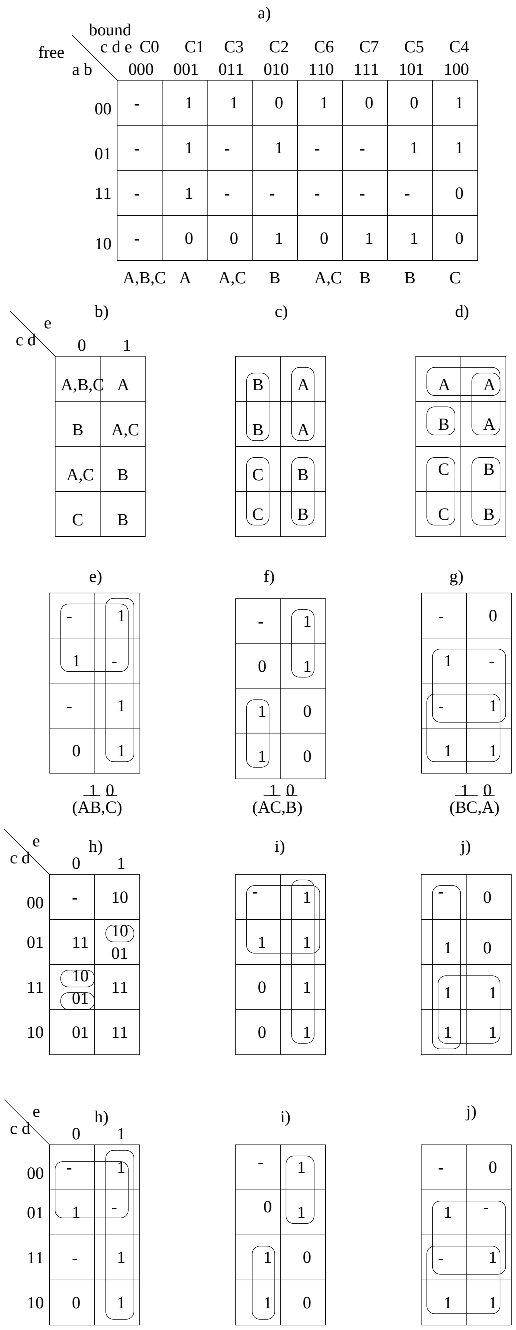


Figure 18: *The Column Group Selection and Encoding process for Example 6.3.*

6.2 Column Encoding Based on Boolean Relation Minimization

The second method is to evaluate the partitions using the symbolic relational map of G before making the mapping choice. The algorithm of this method is as follows:

Algorithm 6.2

1. Take a proper partition. Encode the symbolic relation map with this partition in such a way that all groups which are included in $Block0$ are encoded with 0, all groups which are included in $Block1$ are encoded with 1, all other (those that intersect both blocks of the partition) are encoded with a don't care. Therefore, when a cell of the map includes symbols from both blocks, it is replaced with a don't care. Otherwise, it is assigned the binary symbol associated to this block.
2. The Boolean function created in step 1 is minimized using standard 2-level minimizer Espresso.
3. The steps 1 and 2 are iterated for various proper partitions. Either all partitions are tested, or only some subset of proper partitions is calculated for speed improvement.
4. The partitions with the smallest cost are selected.
5. The Boolean relation is minimized and changed to a Boolean function, to be further processed in the next stages. This is done using Algorithm 6.3 (not presented here in detail; see also section 10.)

Let us observe, that the step 5 of Algorithm 6.2 can be applied to any function G realized as an encoded relation map. This map is found using any encoding algorithm.

Example 6.2

The encoded map created for symbolic relation map from Figure 17c, and partition $\tau_1 = \{\overline{AB},\ \overline{C}\} =$ {B1, B0} is shown in Figure 17h. Using the Symbolic Relation Map from Figure 17c, for partition $\tau_2 = \{\overline{AC},\ \overline{B}\} =$ {B1, B0} - the encoded map is shown in Figure 17i, and for partition $\tau_3 = \{\overline{BC},\ \overline{A}\} =$ {B1, B0} - the encoded map is shown in Figure 17j. The same code as previously is selected.

The Boolean Relation Map is created in Figure 17h. In cell 010 we have two binary vectors - 01 and 10, which means, any of them can be selected, and the selection should be done in a way to simplify the logic. Selecting the first choice, 01 in both cells, will lead to the solution from Figure 17m.

Example 6.3

This is a slightly modified Example 6.2. We just permuted columns in the Initial Map, to demonstrate how such permutation can affect the quality of encoding.

Figure 18a presents an initial function map. As previously, below the columns, their possible colorings from multi-coloring are shown. Two selections to create large groups are shown in Figure 18c and Figure 18d, respectively. They are supposed to simplify the future encoding.

The Symbolic Relation Map from Figure 18b is now evaluated using various partitions. For partition $\tau_1 = \{\overline{AB},\ \overline{C}\}$ with block $\overline{AB}$ encoded with 1 and block $\overline{C}$ encoded with 0, the Boolean Relation from Figure 18e is obtained with cost 2. Similarly, for partition $\tau_2 = \{\overline{AC},\ \overline{B}\}$ the cost is 6 - see Figure 18f, and for partition $\tau_3 = \{\overline{BC},\ \overline{A}\}$ the cost is 2 - see Figure 18g, Therefore, partitions τ_1 and τ_3 are selected as having the smallest total cost among all partitions whose product is $\overline{0}$. The resultant code is: A = 10, B = 11 and C = 01. The Boolean relation map corresponding to this code is shown in Figure 18h. Making the choices encircled in the map, the solution is the Boolean function shown in Figure 18i,j. It can be shown, that the same solution is created by selecting the symbols as in Figure 18c and next encoding with $\tau_1 = \{\overline{AB},\ \overline{C}\}$ and $\tau_3 = \{\overline{AC},\ \overline{B}\}$ - see Figure 18k.

Both Algorithm 6.1 and Algorithm 6.2 can be called for any sets of compatibles. Several heuristics should be added to make the process more efficient. There are several trade-offs among these two approaches. The second method requires an approximate minimizer of Boolean relations. Such program does perhaps exist in U.C. Berkeley tools. Therefore, the second method should be implemented. Yet another variant of this appraoch is present in section 10.

7 COLUMN ENCODING

This section describes a high-performance algorithm for the Column_Compatibility/Encoding Problem of Decomposition. This method can find applications whenever a very high quality circuit is looked for, in a given realization technology, but when the function is of rather small size.

The method combines two, traditionally independent stages of column minimization and symbol encoding, to the single process of concurrent column minimization and encoding. The basic idea of this method is to select such partitions for the symbols of the not necessarily column minimized table (in particular, the columns), that the circuit realization of this table, composed from blocks G and H, minimizes certain cost function. When these partitions are used for symbol encoding, the number of symbols is reduced, as a result of assigning the same codes to the groups of compatible columns C_i. Let us make a point that the number of symbols is reduced but not necessarily totally minimized. Such an approach corresponds to the search for the maps that are equivalent to the initial map and that minimize the realization.

The branch-and-bound optimum algorithm, as well as an approximative variant, based on multivaluated function minimization are described.

7.1 Previous Work

The function decomposition procedure can be either to encode directly the initial map (which leads to having the same codes for many input columns), or first to minimize the number of symbols S_i for columns C_j and follow it with the symbol encoding process. The aim of these methods is, in general, to minimize the semiconductor area in the selected realization technology or some other measure of decomposed function complexity. Various FSM assignment algorithms exist in the literature and can be adopted for these tasks, [2, 9, 16, 54, 23, 29, 27, 38, 52].

Application of the principle of symbol minimization results from the following conviction, that is assumed by most of the systems: "the more the symbols, the more complicated is the realization, hence there are more intermediate variables, and therefore the realization of both block functions is more complicated". Practical examples bear, however, evidence that the map with the minimum number of symbols will not necessarily yield the circuit realization with the minimum total complexity. The symbol minimization is useful, so somehow it must be incorporated to the entire decomposition process, but may be not to the most extent, usually only some reduction of the set of symbols gives the exact optimum solution. Instead of minimizing the number of symbols (compatibles) it is more reasonable to minimize the number of intermediate variables and next, as a secondary criterion, the complexities of functions G and H.

The next design stage is the encoding of the symbols. The graph embedding approach [2, 9], the "column-based" approach [16], and the partition theory [54, 23, 28, 52] are the best known methods applied in the past for state machine assignment and these methods give inspiration to create new algorithms for encoding in decomposition. It is our opinion that the exact minimum solution can be obtained only when the methods based on the partition theory are used. We will, therefore, briefly review these methods. First, it must be mentioned that many of them generate the minimum solution assuming, again, the condition of the minimization of the number of intermediate variables. In such a case for K symbols the partitions of total number k are selected, where $2^k - 1 < K \leq 2^k$. This leads to the concept of *proper partitions,* defined as the two-block partitions in which the number of symbols in the larger block does not exceed 2^{k-1}. The partitions are then selected among the set TP of proper partitions.

Last but not least, the partitions are selected in a way that minimizes the total number of intermediate variables. This condition is expressed by an attempt to select such set TF (called the *final family of partitions*) from set TP of such partitions τ_i, that:

$$\textit{every columns in every block of} \prod_{\tau_i \in TF} \tau_i \textit{ are compatible} \tag{13}$$

It is well-known that the minimum solution doesn't have to produce the minimum number of symbols, but also, the minimum number of intermediate variables.

7.2 The Principle of Joint Minimization and Encoding of Columns

The method described below replaces two stages:

- minimization of the number of symbols (compatible columns),
- encoding of these symbols.

with one stage of joint column minimization and encoding.

In this joint process certain cost function is optimized, which is related to the realization of the predecessor and successor blocks functions in a selected technology. The goal of this approach is to find a new and efficient method of joint column minimization and encoding, which does not have the previously mentioned weaknesses.

The final family (set) of partitions, TF, is defined as such a family, that the partition product of all partitions from that family satisfies Equation (13).

The optimum family, TO, is such a final family that minimizes the value of an assumed cost function.

The family TO in our approach is chosen among all two-block partitions - this gives a possibility of finding better solutions, but is computationally less efficient than using the proper partitions.

The main idea of the method of joint column minimization and encoding proposed here is to select such partitions for the not necessarily column-minimized map, that the two-level AND/OR realizations of the block functions are minimized according to some cost function. Simultaneously, as a byproduct, the number of symbols is reduced (and hence the number of intermediate variables), which results from assigning the same codes to some of the compatible columns (remember that a symbol is a group of columns). This corresponds to the search for such a pair of mappings of the two blocks that their composition is equivalent to the initial mapping of the function and that have the minimum total decomposed function realization.

Separation condition states that each pair of incompatible symbols is separated by at least one partition from TF.

Every set TF that satisfies the separation condition will be referred to as the *final family of partitions.*

A solution to the joint minimization and encoding problem consists of finding the family TO . This family should be such a subset of $T2$, that each of its partitions separates at least one pair of incompatible columns.

7.3 Algorithm of Joint Minimization and Encoding of Columns

Relation COMPATIBLE of symbols is defined as the following set of pairs:

$$COMPATIBLE \underset{===}{\overset{def}{}} \{\{a_i,\ a_j\},\ a_i\ \in\ A,\ a_j\ \in\ A \mid \text{ states } a_1 \text{ and } a_j \text{ are compatible}\}. \tag{14}$$

A dash (-) denotes the don't care column. $COMB{-}SET(A,2)$ denotes the set of all 2-element subsets of set A. Set $COMPATIBLE$ is found with use of an iterative procedure of comparing columns.

Joint minimization and encoding is realized according to Algorithm 7.1.

Algorithm 7.1

Begin

E1: Create set $COMPATIBLE$ of pairs of compatible columns;

E2: Find set of pairs of incompatible columns:
$IS = COMB - SET(A, 2) \setminus COMPATIBLE$;

E3: Assign, under the value of variable $T2$, the family $T2$ generated in Algorithm 5.3;

E4: Find the set of all such partitions from set $T2$, that separate at least one pair of incompatible columns:
(partition $\tau_j = \{B_1, B_2\}$ separates pair

$$p = \{a_1,\ a_2\} \in S, \quad if \quad (a_1 \in B_1 \; and \; a_2 \in B_2) \; or \quad (a_1 \in B_2 \; and \; a_2 \in B_1)) \tag{15}$$

Delete the remaining partitions from $T2$;

E5: (Option) Sort set $T2$ in such a way that among the partitions with the same length of the shorter block the partitions with smaller value of the cost function stand at first. In another variant, sort set TZ according to another criterion;

E6: Using Algorithm 7.2 create the families $TO \in T2$ that satisfy the following conditions:

1. *Condition of separation* of all pairs of incompatible columns

$$(\ \{a_{i1},\ a_{i2}\},\ \in\ IS)(\exists \tau_j \ \in\ TO)[\tau_j \text{ separates the pair } \{a_{i1},\ a_{12}\}] \tag{16}$$

   i.e. for each pair of incompatible columns one of its elements is included in one block of partition τ_j, and the other in the second block.

2. *Condition of the minimization of the total cost function CF_τ.*

End

Together with step $E6$, generated are the block functions. These are next used at the next stages of the decomposition. According to the selected value of parameter ALL the algorithm finds one or all TO families.

Algorithm 7.2

Generation of family (families) TO from family $T2$ by the joint minimization and encoding.

Begin

1. *Creation of initial state of the tree*
   $(GS(0), QS(0), AS(0), PR(0), QF(0)) := (T2,\ \{\},\ IS,\ \bar{1},\ \{\}, 0)$;
   $CF_{min} := \infty$; SOLUTION := {};
   Initialize list $COSTS$;

2. The same as the step 2 in Algorithm 5.4, but now $V(N) = (GS(N), QS(N), AS(N), PR(N), QF(N))$;

3. *Extension of state $V(N)$*

   (a) If $GS(N) = \{\}$ then begin delete first element from list $OPEN$; go to 2 end

(b) $e :=$ first element from $GS(N)$;

(c) $GS(N) := GS(N) \;-\; \{e\}$;

(d) $QS(NN) := QS(N) \;\cup\; \{e\}$;

(e) $AS(NN) := AS(N) \setminus \;\; \{\{a_i,\ a_j\} \;\in\; AS(N) \mid (e \cdot\ \ s\ a_i \;=\; a_i\ and\ \overline{e}\ \cdot s\ a_j \;=\; a_j)$ $or\ \ (e \cdot s\ a_j \;=\; a_j\ and\ \overline{e}\ \cdot s\ a_i \;=\; a_i)\}$;

(f) $PR(NN) := PR(N)\ \cdot \mathrm{p}\ \{e, \overline{e}\}$;

(g) $GS(NN) := GS(N) \setminus \{e_1 \in GS(N) | \,(\nexists \{a_i, a_j\} \in AS(NN))$ [e_1 separates $\{a_i, a_j\}$] } ;

(h) If $(GS(NN) = \{\})$ *and* (PR(NN) $\neq\ \overline{0}$) then go to 3; /* backtrack */

(i) Calculate $q(e)$, add news item to $COSTS$ (this step is executed if $q(e)$ is not found in Algorithm 5.1;

(j) $QF(NN) := QF(N) +\ q(e)$;

(k) If $QF(NN) > CF_{min}$ then go to 3; /* backtrack */

4. 4: *Checking if solution*

5. If $AS(NN) = \{\}$ then new solution found.

6. further steps as in Algorithm 5.4 with the only difference that

   V(NN) = (GS(NN), QS(NN), AS(NN), PR(NN), QF(NN)).

End

Comment.
Set AS is decreased by those pairs of incompatible columns, that were separated by the partition e (i.e. one column from the pair belonged to one block of partition e and the other column to the second block).

8 SYMBOL ENCODING BASED ON DICHOTOMIES

In this section, we will present one more new method for optimizing both the predecessor logic (output encoding) and the successor logic (input encoding).

This method takes some of the ideas from the input encoding techniques that use dichotomies [60], and some ideas from the output encoding using dominance relationships from [15]. These two methods are combined together for our purpose of solving both these problems concurrently.

8.1 The function for Example 8.1

Before explaining the encoding methodology, we will first present an example that will be used in the presentation of our technique.

Consider the following logic function in the initial map, Fig. 19.

ab \ cdef	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
00	0	1	0	0	1	1	1	0	0	0	0	0	0	0	0	0
01	1	0	1	1	0	0	0	0	0	0	1	1	1	1	0	0
11	1	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0
11	0	0	0	0	0	0	0	1	1	0	0	0	1	1	1	1
	A	B	A	A	B	B	B	C	C	C	A	A	D	D	C	C

Figure 19: *Decomposition table for Example 8.1.*

Notice that the columns correspond to the Bound set and the rows correspond to the Free Set, as previously described. In this example, the columns have already been colored, with symbols A, B, C, D. It is these colors (symbols) that we will try to assign the codes to. The presented method is a symbol encoding technique.

8.2 Output Encoding.

The single most important idea of our method for output encoding is the concept of dominance [15].

Definition 8.1 *A binary representation of a symbol S_1* **dominates** *another symbol S_2 if for each bit '1' in S_2 there is a corresponding bit '1' in symbol S_1.*

The following examples illustrate this concept.

$S_1 > S_2$ for $S_1 = 01011$ and $S_2 = 01001$

$S_1 > S_2$ for $S_1 = 0110$ and $S_2 = 0100$

Why do we want to use the idea of dominance in encoding? It is simple. By developing dominance constraints we constrain how the codes can be assigned to the symbols. For instance, having four variables to encode and assuming that $B > A$, $C > B$, and $D > C$, the following encoding would satisfy these constraints:

$$A = 000 \qquad B = 010 \qquad C = 110 \qquad D = 111$$

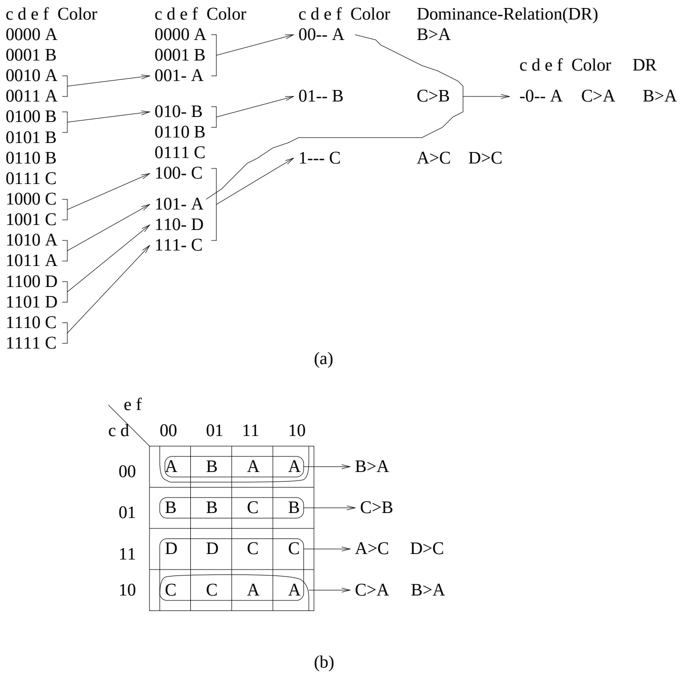


Figure 20: *Boolean minimization of function using dominance relations for Example 8.1: (a) Tabular form of creating implicants and dominance relations, (b) Kmap for creating implicants and dominance relations.*

Let us first consider Figure 19 which shows the cubes for input variables c, d, e and f, and also shows the corresponding colors or intermediate symbols. Output encoding of these cubes and associated outputs will be now performed. To do this, the cubes and their outputs are compared, and the cubes will be merged that have the same outputs. For example, cube 0010 associated with symbol A, and cube 0011 associated with symbol A, can be merged into cube 001- associated with symbol A. This operation can be performed again on the cubes for symbols B, C, and D.

When the preliminary minimization of this cover has been completed, these cubes can be tried to be optimized even further. This is done by merging corresponding cubes with don't cares. Take cube 0000, associated with A, and cube 001 associated with A. Merging these two cubes can only occur by using some other cube as a don't care. The only other cube that can be used to merge these two is 0001 associated with B. Because of this operation, it can be now stated that symbol B dominates symbol A. Similarly, for the other cases, symbol C dominates symbol B, C dominates A, A dominates C, and D dominates C. The graphical interpretation of creating groups and domination relation is shown in Figure 20.

Two approaches are possible with respect to finding the dominance relations. The first approach finds only as many dominance relations that there are no conflicts between the relations. Another approach generates all the possible dominance relations and then constructs a dominance graph showing all the dominance relations.

- For the first approach we would only find enough dominance relations so that they cover all of the symbols and be not in conflict. For example, $B > A$, $C > B$, and $D > C$ would cover the symbols, and looking for other dominance relations would be unnecessary. This is a fast way of generating the dominance graph for output encoding.

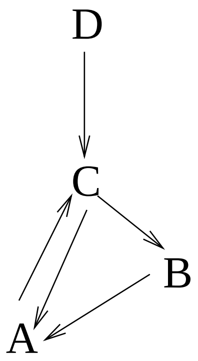


Figure 21: *Dominance graph for Output encoding with all possible relations for Example 8.1.*

- The second approach would be to generate all possible dominance relations, and then to draw the dominance graph for them. Next select from the graph only those relations which are allowed by the following rules. Note that the solution obtained by removing relations from the graph is not necessarily an optimal one. There are many possible solutions for larger graphs. For example, $B > A$, $C > B$, $D > C$ can be one solution, and $B > A$, $B > C$, $A > D$ could be another solution. Obtaining of the final solution using the rules is arbitrary. It may be then a good idea to use the first approach to find the dominance relations instead of finding all possible relations.

We will experiment with these two and also possible approaches to solve this problem.

In case that a decision to generate the complete graph has been done, then the following procedure is observed.

To solve this graph, the conflicting relations have to be removed. This graph can possibly contain conflicting relations, that is, both $A > C$ and $C > A$. Clearly, these cannot both occur simultaneously, because that would imply that the encoding for A and C would be the same. That would be the only way both relations were satisfied, and this is not something that we want.

The dominance graph is shown in Figure 21.

The rules applied in removing some of the relations in order to avoid any conflicts are the following:

1. There should be no direct cycles. That is, $A > B$, $B > C$, and $C > A$ is a cycle and is illegal.

2. For an n-bit encoding length, there cannot be a path created from more than n similarly directed relations. That is, $B > A$, $C > B$, $D > C$ for an encoding length of 2 is illegal.

Using these rules, the graph from Fig. 22 is drawn for our example. Notice that the conflicting relation from A to C for $(A > C)$ has been removed. This would have caused a direct cycle in the graph had it be left there.

Now that the dominance graph has been solved, one can proceed to the input encoding part of the algorithm. One more item needs to be mentioned here. It is possible that the solution that has been obtained from the dominance graph will not fit with the input encoding results, so that it may be necessary to obtain a different solution from the dominance graph in order to obtain a better combination of the two partial encoding methods.

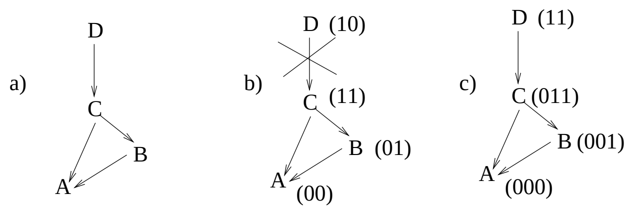


Figure 22: *The Dominance graph after removal of conflicting relations for Example 8.1.*

8.3 Input Encoding.

The input encoding algorithm presented here is based upon dichotomies [60].
We first start out with some definitions.

Definition 8.2 *A* **dichotomy** *is a disjoint 2-block partition (left:right) of a subset of column indexes of the reduced constraint matrix M. The left side of the partition contains the column indexes of the positions with 0's, and the right side contains the column indexes of the first position with a 1 for that dichotomy. Other dichotomies for the same row will get the other 1's in that row.*

Definition 8.3 *A dichotomy $D_i=(l_i{:}r_i)$ is said to* **cover** *dichotomy $D_j=(l_j{:}r_j)$ if l_j is a subset of l_i and r_j is a subset of r_i, or if r_j is a subset of l_i and l_j is a subset of r_i.*

Definition 8.4 irredundant dichotomies *are such dichotomies that are not covered by other dichotomies.*

Definition 8.5 *Two dichotomies $D_1=(l_1{:}r_1)$ and $D_2=(l_2{:}r_2)$, are said to be* **incompatible dichotomies** *if there exists an unordered pair of indexes (j,k), j,k are elements of l1 or j,k are elements of r_1, such that j is an element of l_2 and k is an element of r_2, or if the same is true with the role of D_1 and D_2 reversed. Otherwise the two dichotomies are considered* **compatible**.

Definition 8.6 A union of two dichotomies, *$D_1=(l_1{:}r_1)$ and $D_2=(l_2{:}r_2)$, denoted $D_1 \cup D_2$, is a dichotomy, D=(l:r), such that $l{=}l_1 \cup l_2$ and $r = r_1 \cup r_2$, or $l = l_1 \cup r_2$ and $r = r_1 \cup l_2$.*

Definition 8.7 *A* **prime dichotomy** *is a dichotomy which is not included in any other dichotomy.*

Definition 8.8 *A* **seed dichotomy** *is an elementary dichotomy found in the constraints matrix.*

Definition 8.9 *A* **distinct seed dichotomy** *is a seed dichotomy which is covered by one and only one prime dichotomy.*

Definition 8.10 *An* **essential prime dichotomy** *is a dichotomy which contains at least one distinct seed dichotomy.*

To perform the input encoding, first binary vectors must be assigned to each of the colors in Positional Cube Notation (PCN). (This notation is also called the one-hot notation and is shown to the right of

a b	Color	PCN	output
00	A	1000	0
01	A	1000	1
10	A	1000	1
11	A	1000	0
00	B	0100	1
01	B	0100	0
10	B	0100	0
11	B	0100	0
00	C	0010	0
01	C	0010	0
10	C	0010	0
11	C	0010	1
00	D	0001	0
01	D	0001	1
10	D	0001	0
11	D	0001	1

a b	MPCN	Output
00	1011	0
01	1001	1
10	1000	1
11	1100	0
01	0110	0
10	0111	0
00	0100	1
11	0011	1

Figure 23: *Intermediate variables with assigned positional cube notation, and minimization.*

the corresponding color). Then the multiple-valued minimization of the cover is performed (the binary inputs). This is shown in Fig. 23.

Notice that the table has been reduced by taking rows with similar inputs (a and b) and outputs and merging the cubes for the colors, that is, the positional cube notation. For example, the first rows for colors A, C, and D can be merged to form a row with inputs 00, output 0 and MPCN (merged positional cube notation) 1011. That is the OR of A, C, and D is 1011. This is done repeatedly until all possible mergings have been found. Now, the processing of this cover can be continued.

The cover that we are interested in is for the merged PCN in Fig. 24. According to [11] this cover can be reduced further by deleting the rows with single 1s, with all 1s, duplicated rows, and the meet rows, that is, the rows which are the ORs of two or more rows.

One can notice that the 1000 row can be removed, because it has only one 1. The same can be said of the 0100 row. The 1011 row can be deleted because it is the merging of rows 1001 and 0011. The same can be said of 0111 because it is the merging of rows 0110 and 0011. This produces the minimized cover from Fig. 25.

Now the cubes from the final cover are used to generate dichotomies based on the definitions presented before. For example, 1001 generates the dichotomies (AD:B) and (AD:C) because it has 1's in the first and fourth columns, and '0s in the second a third columns.

The introduced above definition of irredundant seed dichotomies is now used to extract those dichotomies that are not subsets of other dichotomies. These dichotomies would be used to complete the solution. In this particular example all of the dichotomies are irredundant.

MPCN
ABCD
1011
1001
1000
1100
0110
0111
0100
0011

Figure 24: *Initial Cover for Input Encoding.*

	Cover	Dichotomies Generated per Row
ABCD	1001	(AD:B) (AD:C)
	1100	(AB:C) (AB:D)
	0110	(BC:A) (BC:D)
	0011	(CD:A) (CD:B)

Figure 25: *Final Minimized Cover for Input Encoding with Associated Dichotomies.*

Prime Dichotomies	(14:2)	(14:3)	(12:3)	(12:4)	(23:1)	(23:4)	(34:1)	(34:2)
(123:4)				X		X		
(124:3)		X	X					
(12:34)			X	X			X	X
(14:23)	X	X			X	X		

Figure 26: *All Possible Prime Dichotomies and Their Relation to the Seed Dichotomies.*

	V_1	**Color**
A	0	0
B	0	1
C	1	1
D	1	0

Figure 27: *Encoding assigned to intermediate variables based upon input encoding methodology.*

Now all possible prime dichotomies are generated. For this example, they are listed in Fig. 26.

Finally, the crucial step of combining the output and input encoding steps is executed. In order to take into consideration the dominance relationships from the output encoding stage, the possible colorings for the dichotomies in the input encoding are considered.

To find the colorings, the following facts are taken into consideration. There are some seed dichotomies that have only one intersection with a prime dichotomy. This means that these prime dichotomies are essential dichotomies and must be included in any coloring of the dichotomies. For this example, there are two essential prime dichotomies (AB:CD) and (AD:BC). Since these two dichotomies also cover all of the seed dichotomies, then they are also a complete coloring of all of the seed dichotomies. For other examples, there may be more than the one possible coloring, and all these colorings would need to be determined.

Once the colorings (AB:CD) and (AD:BC) have been determined, the solutions are converted to encodings that satisfy the colors A, B, C, and D. The conversion is performed by making the indexes in each dichotomy correspond to the rows of the following table. There is some flexibility in terms of whether one assigns the indexes in the left side of the dichotomy to 0s or 1s. By applying the rules we try to satisfy the dichotomies by selecting the appropriate conversion. In this example, the left side of the dichotomies has been assigned to 0's and the right side has been assigned to 1's (see Fig. 27).

Notice that two of the three dominance relations specified earlier ($B > A$, and $C > A$) have been satisfied. It can be verified that B = 01 dominates A = 00, and C = 11 dominates A = 00. $D > C$ was not satisfied, but this is the price to be paid for concurrently optimizing both the predecessor and the successor logic.

Note that having more colors available (other than the single one that was available), one may have

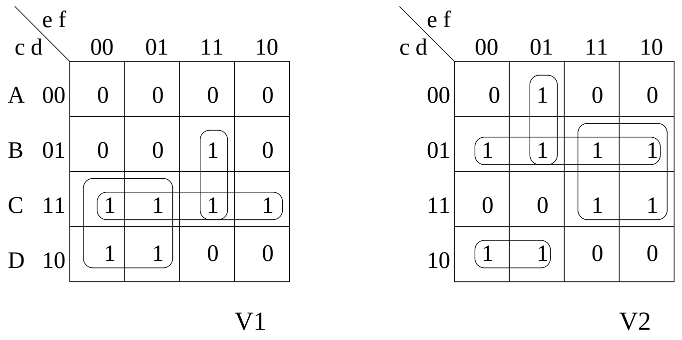


Figure 28: *Karnaugh maps of Intermediate Variables for Example 8.1.*

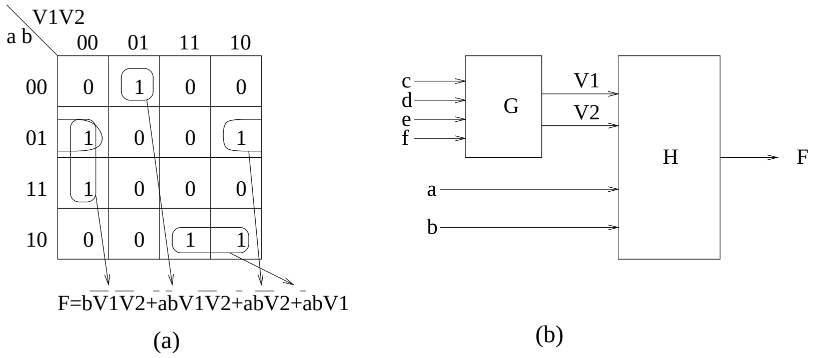


Figure 29: *Example 8.1: (a) Karnaugh Map of the Output function H, (b) the decomposition diagram.*

been able to satisfy also the constraint $D > C$.

Just to reiterate the merging of these two encodings (input and output), the goal is to select the coloring of the dichotomies which satisfies the dominance relations. Remember that to verify that the dominance relations are satisfied, the result of the graph colorings must be converted to actual codes. Only this way will it become obvious that the codes satisfy the dominance relations.

Now the intermediate functions are realized using Karnaugh maps (Fig. 28).
From the maps:

$$V1 = c\overline{e} + cd + d\ e\ f$$

$$V2 = c\overline{d}\,\overline{e} + \overline{c}\ d + \overline{c}\ \overline{e} f \ + \ d\ e$$

Notice that the dominance relation $B > A$ is used to create the group in the first row for V_1. Also, the dominance relation $C > B$ is used to create a group in the second row for V_2.

Finally, we create the Karnaugh map for the output function.
From the map, we obtain:

$$F = b\,\overline{v_1}\,\overline{v_2} \ + \ \overline{a}\ \overline{b}\,\overline{v_1}\,v_2 + \overline{a}\ b\,\overline{v_2} \ + \ a\,\overline{b}\,v_1$$

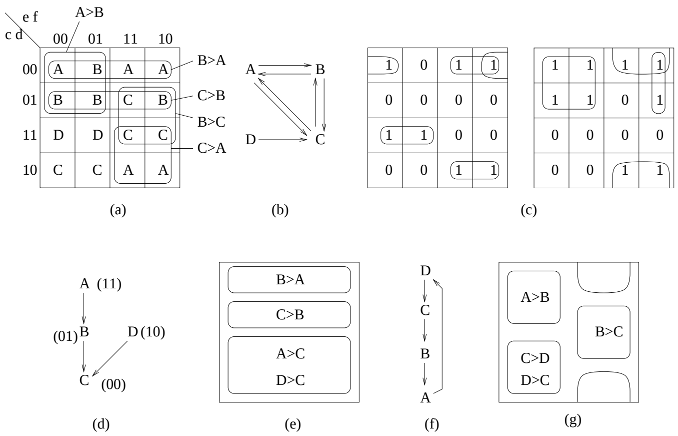


Figure 30: *More examples of finding implicant covering and Dominance Relations for Example 8.1: (a) the Kmap with groups shown, (b) the complete initial Domination Graph, (c) the functions $V1$ and $V2$, (d) - (g) other variants.*

The Kmap for block H and the final decomposition are shown in Figure 29.

More examples of finding implicant covering and Dominance Relations for Example 8.1 are shown in Figure 30.

A systematic method of evaluating partitions, similar to those from previous chapters is shown in Figure 31. Groups AB, AC, AD, etc near arrows correspond to blocks of partitions encoded with one. The costs of these partitions are calculated using Espresso minimization performed on respective Kmaps. The costs of prime implicants are given near the Kmaps. For instance, 223 near the first map means that when A and B are encoded with 1 there are three implicants and their costs are 2,2, and 3. The total cost is 2+2+3 = 7.

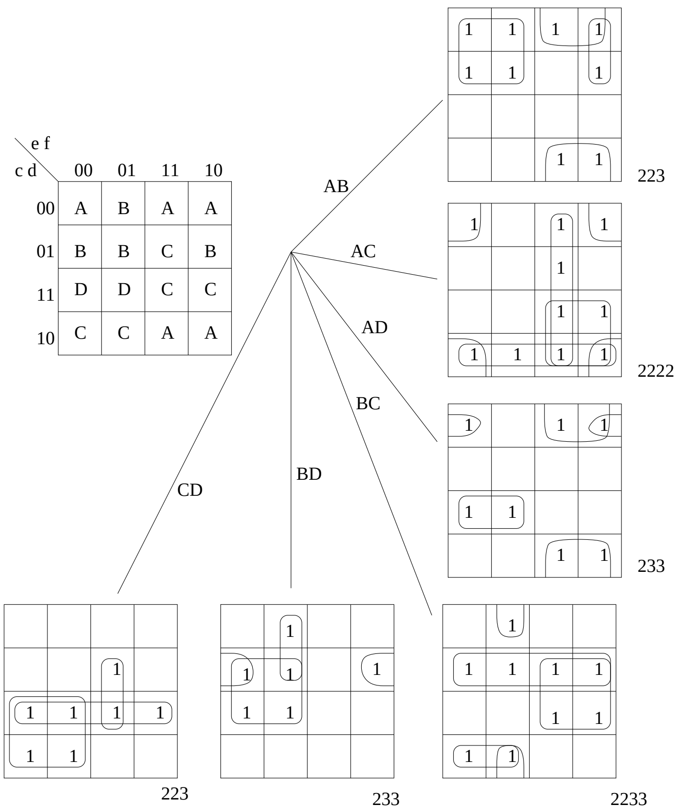


Figure 31: *A systematic method for evaluating partitions.*

8.4 Dominance Relation Construction for G Block

The predecessor and sucessor blocks (G and H blocks) can be synthesized with a concurrent input/output encoding method in which dominance relations among the symbolic outputs (columns C_i) of G are obtained and used to simplify the constrained input encoding problem for the H function. In the following a procedure is proposed to obtain the dominance relations and the associated acyclic graphs introduced in the previous section. (the problems of domination relations are presented in [15]).

Example 8.1

For the map from Example 8.1 one obtains the following specification of function G with symbolic output:

0000 A
0010 A
0011 A
1010 A
1011 A

0001 B
0100 B
0101 B
0110 B

0111 C
1000 C
1001 C
1110 C
1111 C

1100 D
1101 D

The procedure starts with an assignment of colors to the compatible columns of the decomposition chart (cofactors of the bound set variables). In the presented example, the four compatible classes have been colored with A,B,C, and D. The binary enumeration of each column will be treated as a minterm in the 4-valued ON-set of G, i.e. minterms 1100 and 1101 produce value D at the output.

An initial choice for a starting color must be made. Our approach here is based on a heuristic: the groups of minterms (colors) are processed top to bottom, starting with the lower (decimal) valued minterms first and ending with the higher valued ones for the last color D.

Another heurstic may be to select colors that are used in most minterms.

The ON-set of the symbolic output A will be minimized first, with the ON-sets of the following outputs B, C, and D as don't cares. Borrowing of minterms from the ON-sets of B, C, and D is used to achieve a minimal cover for A: eg. borrowing 0001 from the ON-set of B and 1000,1001 from ON-set of C results in a 2-cube cover for value A:

00– A

10– A

Next, the intersection of minterms for D (not 'borrowed' from) and the above cover for A is computed; if it is empty D does not dominate A directly and the corresponding nodes in the dominance graph are not connected. Otherwise, the connection is made as in the Dominance Graph from Fig. 32.

The dominance graph now has 3 of the 4 symbols, with only one cover computed (for A). The covers for B and C are computed next to establish the position of D in the graph.

Borrowing from the minterms of C, the B cover is:

```
  C
 / \
B   |
\  /
 A
```

Figure 32: *The initial Dominance Graph for Example 8.2.*

```
     D
    /
   C
  / |
 /  |
B   |
\  /
 A
```

Figure 33: *The final dominance graph*

0001

01–

and the one for C (borrowing both minterms of D):

0111

100-

11–

Since the minterms of D intersect the cover for C , the final dominance graph is as shown in Fig. 33: The pairwise dominance relations for the following input encoding step of the sucessor are:

D > C

C > A

B > A

As we see, the relation C > B had to be removed to have an encoding with the minimum number of bits.

The above, one at a time, introduction of new variables into the graph prevents the formation of cycles. The top-down (from low-valued to high-valued decimal) construction of the cover facilitates the implementation as well.

9 COLUMN ENCODING BASED ON DYCHOTOMIES

9.1 Input-Output Encoding for Simple Disjoint and Complex Disjoint Serial Decompositions

Similarly to the previous section, the encoding methods proposed in this section will be based on combining the input and output encoding methods.

In brief, the Input-Output Encoding Procedure is performed as in the following Algorithm 9.1

Algorithm 9.1

This algorithm starts from set of columns and corresponding disjoint symbols from graph coloring or clique parititioning. However, symbols of any cliques (also overlapping) can be also used.

1. **Output encoding (G outputs).**
   (a) The output encoding problem for function G is first formulated, and dominance relations are derived for the symbolic variables C_1, C_2,... of G corresponding to the columns of the initial map. (The difference with the method from the previous section is that we are looking here for dominance relations between columns, and not for dominance relations between symbols).
   (b) For each symbolic variable C_i (symbol S_i) find a minimal coer with cubes of the bound set.
   (c) All the input cubes for the column variables following C_i are used as don't cares to produce the above minimal cover. This introduces intersections between the cubes of the cover for C_i and the cubes in covers for other symbolic outputs that follow column C_i.
2. **Binary-multivalued constraint matrix (H inputs).**
   (a) A set of constraints is formulated between the binary variables in the free set and the multivalued variable Column (with values C_1, C_2,...).
   (b) The constraint matrix describes the ON set of the function and since the compatible classes are known at this stage, the don't cares can be assigned only as needed to group columns in a given class, leaving other non-identifying don't cares. Zeroes are assigned for these non-identifying don't cares to increase the number of dichotomies later on. (In other variant we would change all non-identifying don't cares as they are for next states).

   Multivalued minimization of the constraint matrix is not required as in the standard input encoding problem for two-level realization of H.
3. **Seed dichotomy generation.** All pairs of seed dichotomies are formed according to the constraint matrix. Since the multiple-valued minimization is not used, a large number of seed dichotomies is generated, which improves the chance of detecting prime dichotomies without having to add additional 'separating' dichotomies.
4. **Graph of compatible dichotomies.**
   (a) All seed dichotomies which violate the dominance relations for the outputs of G are removed from the list. A dominance relation is not satisfied if any dominating symbols appear in the right part of a dichotomy.
   (b) The graph of all compatible dichotomies is constructed using the following definition of compatibility:

   $(L1; R1)$ is compatible with $(L2; R2)$ iff $L1$ is compatible with $L2$ and $R1$ is compatible with $R2$

de \ abc	C1 000	C2 001	C3 010	C4 011	C5 100	C6 101	C7 110	C8 111
00	X	1	1	0	1	0	0	1
01	X	1	X	1	X	X	1	1
11	X	1	X	X	X	X	X	0
10	X	0	0	1	0	1	1	0
	A	A	A	B	A	B	B	C

Figure 34: *Table 1.*

5. **Identification of prime dichotomies.**

   All prime dichotomies are identified and a subset of them is selected such that all seed dichotomies (meeting the dominance conditions) are covered by the members of the set

   dichotomy $(L1, R1)$ covers $(L2, R2)$ iff $L1$ includes $L2$ and $R1$ includes $R2$.

6. **Final code selection.** The final codes for all columns in the decomposition table are formed from the prime dichotomies in the above subset. Identical code words will result for compatible columns. A don't care can be assigned to a code in case a particular column symbol is missing from the corresponding dichotomy.

The following examples 9.1 and 9.2 will illustrate the above procedure.

Example 9.1

The incompletely specified function from Wan Wei (see [57]) will be used. A simple disjoint serial decomposition exists in this case for the free and bound sets, as shown, and the compatible classes have been identified.

STAGE 1. Finding the dominance relations for columns.
Function F is presented in Table 1 from Figure 34.

The hand calculation of encoding for this function is shown in Figure 35.
Using Clique Partitioning Algorithm from Table 1, we can find all symbols:

Symbol (compatible) A $= \{C_1, C_2, C_3, C_5\}$
Symbol B $= \{C_4, C_6, C_7\}$
Symbol C $= \{C_8\}$

The symbolic relation of G with $C_1, .., C_8$ as outputs is the following:

abc
000 C_1
001 C_1

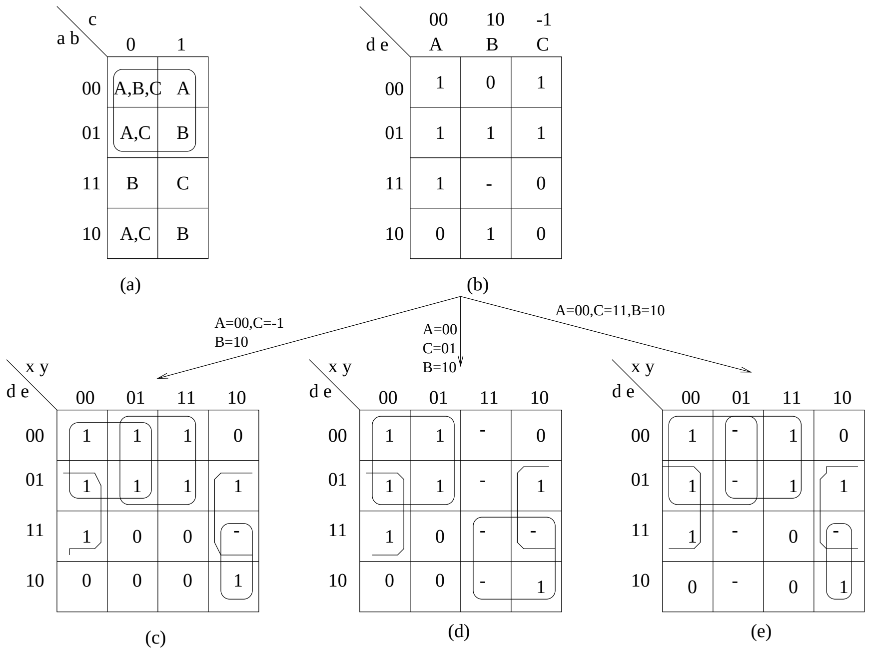


Figure 35: *Realization of function from Example 9.1: (a) Symbolic output relation for function G for Kmap from the previous figure, (b) Symbolic Map of function H with encoding A=00,B=10,C=-1, (c),(d),(e) - Kmaps for three different encodings of map from (b)*

010	C_1
100	C_1
000	C_2
001	C_2
010	C_2
100	C_2
000	C_3
001	C_3
010	C_3
100	C_3
000	C_5
001	C_5
010	C_5
100	C_5
011	C_4
101	C_4
110	C_4
011	C_6
101	C_6
110	C_6
011	C_7
101	C_7
110	C_7
111	C_8

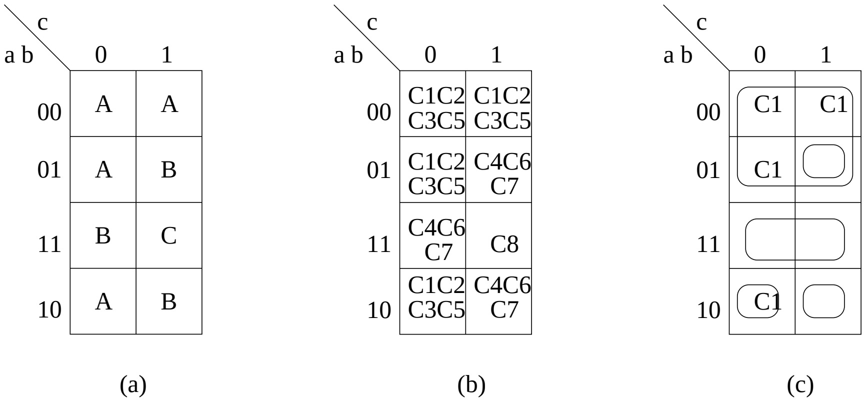


Figure 36: *Graphical illustration of stages of encoding.*

Let us observe that the input cubes are the same in each of groups C_1, C_2, C_3, and C_5. For C_1 the cover is:

0XX
100

Let us observe that to create cube 0XX in class C_1, an input 011 is necessary, which is taken from C_4, C_6, or C_7. We can thus say, that the 0XX cube above is formed by considering the input cubes for columns C_4, C_6, C_7, as don't cares.

The same cover exist also for the input cubes in each compatible class for C_2, C_3, and C_5. For each of the groups, C_4, C_6, and C_7, the cover is the same, and it is:

011
101
11X

The 11X cube above is formed by considering the input cube for column C_8 as don't cares. The above cover intersects the cover for C_1, C_2, C_3, C_5. Thus, the following dominance relation can be found:

$$C_4,\ C_6,\ C_7\ >\ C_1,\ C_2,\ C_3,\ C_5$$

The input cover (cube 111) for C_8 intersects the cover for C_4, C_6, C_7 thus

$$C_8\ >\ C_4,\ C_6,\ C_7$$

These stages are illustrated graphically in Figure 36.

STAGE 2. Finding the dichotomies.

The input constraint matrix for function H (see Fig. 37) is created from symbol compatibles and from TABLE 1.

By comparing the matrix for function H with the original table of the initial function one can verify, that all the "non-identifying don't cares" in class B have been assigned '0' . By "non-identifying don't cares" one means those don't cares that are not replaced with cares during combining compatible column according to some compatible.

Now matrix H is used, row-by-row, to generate seed dichotomies. For instance, from the first row of matrix H, the following two dichotomies are created:

de	C1	C2	C3	C4	C5	C6	C7	C8
00	1	1	1	0	1	0	0	1
01	1	1	1	1	1	1	1	1
11	1	1	1	0	1	0	0	0
10	0	0	0	1	0	1	1	0
	A	A	A	B	A	B	B	C

Figure 37: *The input constraint matrix for function H*

$$P1_a = (C_1C_2C_3C_5C_8;\ \ C_4), \qquad P1_b = (C_4;\ \ C_1C_2C_3C_5C_8)$$

Similarly, the following other seed dichotomy pairs are created from the first row of H:

$$P2_a = (C_1C_2C_3C_5C_8;\ \ C_6) \qquad P2_b = (C_6;\ \ C_1C_2C_3C_5C_8)$$

$$P3_a = (C_1C_2C_3C_5C_8;\ \ C_7) \qquad P3_b = (C_7;\ \ C_1C_2C_3C_5C_8)$$

Similarly, the other seed dichotomy pairs are created from other rows of H:

$$\begin{array}{lll lll}
P4_a & = & (\ C_1\ C_2\ C_3\ C_5\ ;\ C_4\ ) & P4_b & = & (\ C_4\ ;\ C_1\ C_2\ C_3\ C_5\ ) \\
P5_a & = & (\ C_1\ C_2\ C_3\ C_5\ ;\ C_6\ ) & P5_b & = & (\ C_6\ ;\ C_1\ C_2\ C_3\ C_5\ ) \\
P6_a & = & (\ C_1\ C_2\ C_3\ C_5\ ;\ C_7\ ) & P6_b & = & (\ C_7\ ;\ C_1\ C_2\ C_3\ C_5\ ) \\
P7_a & = & (\ C_1\ C_2\ C_3\ C_5\ ;\ C_8\ ) & P7_b & = & (\ C_8\ ;\ C_1\ C_2\ C_3\ C_5\ ) \\
P8_a & = & (\ C_4\ C_6\ C_7\ ;\ C_1\ ) & P8_b & = & (\ C_1\ ;\ C_4\ C_6\ C_7\ ) \\
P9_a & = & (\ C_4\ C_6\ C_7\ ;\ C_2\ ) & P9_b & = & (\ C_2\ ;\ C_4\ C_6\ C_7\ ) \\
P10_a & = & (\ C_4\ C_6\ C_7\ ;\ C_3\ ) & P10_b & = & (\ C_3\ ;\ C_4\ C_6\ C_7\ ) \\
P11_a & = & (\ C_4\ C_6\ C_7\ ;\ C_5\ ) & P11_b & = & (\ C_5\ ;\ C_4\ C_6\ C_7\ ) \\
P12_a & = & (\ C_4\ C_6\ C_7\ ;\ C_8\ ) & P12_b & = & (\ C_8\ ;\ C_4\ C_6\ C_7\ )
\end{array}$$

The dichotomies in a pair are compatible and are included to increase the chance of meeting the dominance relations derived for the outputs of G. After removing the seed dichotomies which do not satisfy the dominance relations from the above list we are left with the following dichotomies:

$$\begin{array}{lll lll}
P4_b & = & (\ C_4\ ;\ C_1\ C_2\ C_3\ C_5\ ) & P5_b & = & (\ C_6\ ;\ C_1\ C_2\ C_3\ C_5\ ) \\
P6_b & = & (\ C_7\ ;\ C_1\ C_2\ C_3\ C_5\ ) & P7_b & = & (\ C_8\ ;\ C_1\ C_2\ C_3\ C_5\ ) \\
P8_a & = & (\ C_4\ C_6\ C_7\ ;\ C_1\ ) & P9_a & = & (\ C_4\ C_6\ C_7\ ;\ C_2\ ) \\
P10_a & = & (C_4\ C_6\ C_7\ ;\ C_3\ ) & P11_a & = & (\ C_4\ C_6\ C_7\ ;\ C_5\ )
\end{array}$$

$$P12_b \quad = \quad (\; C_8 \; ; \; C_4 \; C_6 \; C_7 \;)$$

As we see, the left side is always a dominating column in the dominance relations that were previously found.

The prime dichotomies which cover all seed dichotomies are formed by

$$\{ \; P7_b, \quad P12_b \; \} \quad = \quad (\; C_8 \; ; \quad C_1 \; C_2 \; C_3 \; C_4 \; C_5 \; C_6 \; C_7 \;)$$

$$\{ \; P4_b, \; P5_b, \; P6_b, \; P8_a, \; P9_a, \; P10_a, \; P11_a \; \} \quad = \quad (\; C_4 \; C_6 \; C_7 \; ; \quad C_1 \; C_2 \; C_3 \; C_5 \;)$$

The corresponding code words are:

C_1	00	A
C_2	00	A
C_3	00	A
C_4	10	B
C_5	00	A
C_6	10	B
C_7	10	B
C_8	X1	C

The procedure assigned identical code words to all columns in a compatible class;

G will have 2 outputs $x(c,d,e)$ and $y(c,d,e)$ with $H(x,y)$.

Using the above encoding with $C_8 = 11$ results in the following functions:

$$x = c\overline{d} + c\;e + \overline{c}d\overline{e}, \qquad y = c\overline{d}\;\overline{e}, \qquad H = \overline{a}\;\overline{x} + \overline{a}\;y + b\;\overline{y} + a\;x\;\overline{y}$$

The $C_8 = 01$ code results in the following functions:

$$x \;=\; c\;e + \overline{c}\;d\;\overline{e}$$

$$y \;=\; c\;\overline{d}\;\overline{e}$$

$$H \;=\; b\;\overline{y} + a\;x + \overline{a}\;\overline{x}$$

$$H = \overline{x}\;\overline{d} + e\;\overline{y} + dx$$

Example 9.2

The above input-output encoding procedure can be applied to the complex serial disjoint decomposition problem as well. The following function has complex disjoint decomposition as shown:

$$F(x_1, x_2, x_3, x_4) \;=\; \overline{x_1}\;\overline{x_3}\;\overline{x_4} + x_2\;\overline{x_3}\;\overline{x_4} + \overline{x_1}\;x_3 x_4 + x_2\;x_3\;x_4$$

$$F \;=\; H(g_1(x_1, x_2);\; g_2(x_3, x_4)) \;=\; g_1 \;\cdot\; g_2$$

i.e. H is an AND gate.

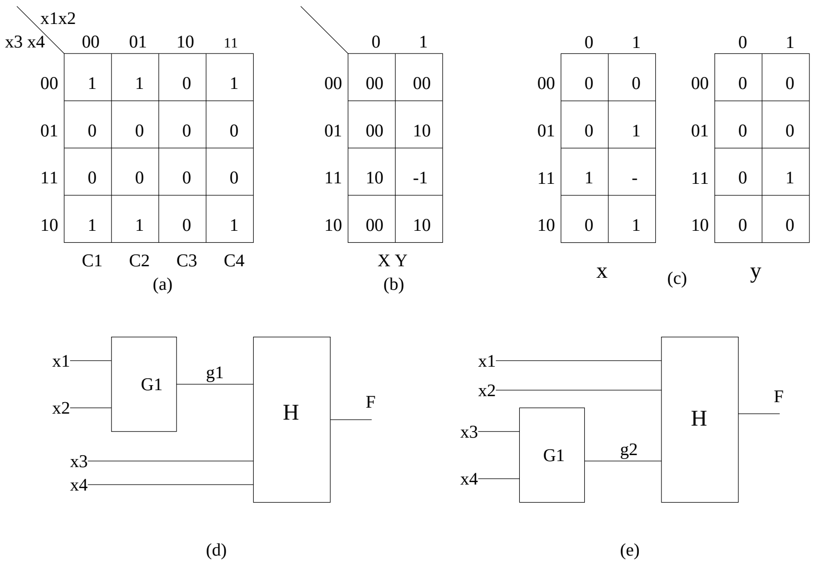


Figure 38: *(a) The decomposition chart for g_1 in Example 9.2, (b) Functions x and y in Example 9.1, (c) realizations of functions x and y, (d) Decomposition with function g_1, (e) Decomposition with function g_2*

x1x2 \ x3 x4	00	01	10	11
00	1	0	0	1
01	1	0	0	1
11	0	0	0	0
10	1	0	0	1
	C5	C6	C7	C8

Figure 39: *The decomposition chart for g_2*

x1 \ x2	0	1
0	C1	C2
1	C3	C4

Figure 40: *A 2 by 2 Kmap.*

X3 X4 \	C1	C2	C3	C4
00	1	1	0	1
01	0	0	0	0
11	0	0	0	0
10	1	1	0	1

Figure 41: *The non-minimized binary/MV constraint matrix for* g_1

The decomposition chart for g_1 is shown in Figure 38a and for g_2 in Figure 39.
Next the cover of the input cubes for g_1 is formed:

cubes 0X and 1X cover C_1, C_2, and C_4

and

cube 10 covers C_3 ; $C_3 > C_1$, C_2, and C_4

x_1x_2	
00	C_1
01	C_1
11	C_1
00	C_2
01	C_2
11	C_2
00	C_4
01	C_4
11	C_4
10	C_3

The above data correspond to the Kmap from Figure 40. The non-minimized binary/MV constraint matrix for g_1 is shown in Figure 41:

The only seed dichotomy meeting the dominance constraint is (C_3 ; C_1 C_2 C_4).

Function g_1 can then be encoded with a single bit, according to:

x1 x2	C5	C6	C7	C8
00	1	0	0	1
01	1	0	0	1
10	0	0	0	0
11	1	0	0	1

Figure 42: *The seed dichotomies*

$$C_3 = 1; \qquad C_1 = C_2 = C_4 = 0$$

Repeating these steps for function g_2 produces:

$0X$ and $1X$ cover C_5, C_8

01 and 10 cover C_6 , C_7 ; $\qquad C_6, C_7 > C_5, C_8$

The seed dichotomies from Figure 42 are:

$$s1_a = (\ C_5\ C_8\ ;\ C_6\ ), \qquad s1_b = (\ C_6\ ;\ C_5\ C_8\ )$$
$$s2_a = (\ C_5\ C_8\ ;\ C_7\ ), \qquad s2_b = (\ C_7\ ;\ C_5\ C_8\ )$$

Dichotomies $s1_b$ and $s2_b$ meet the dominance constraint and assign 0 to $C_5 = C_8 = 0$ and 1 to $C_6 = C_7 = 1$. Using the above encoding for C_1,C_2,C_3, and C_4 results in $g_1 = x_1\ \overline{x_2}$ and the encoding for C_5, C_6, C_7, C_8 results in:

$$g_2 = x_3\ \overline{x_4} + \overline{x_3}\ x_4, \qquad H_1\ (\text{ for } g_1\ ) = \overline{x_3}\ \overline{x_4} + x_3\ x_4 = \overline{g_2}$$
$$H_2\ (\text{ for } g_2\ ) = \overline{x_1} + x_2 = \overline{g_1}$$

Therefore the final $H = \overline{g_1}\ \cdot\ \overline{g_2}$ (see Fig. 43. This is the resulting form for H iff simple disjunctive decomposition exists for both free and bound variable sets (interchanged).

9.2 Encoding for G and H functions in Non-Disjoint decompositions

The above procedure can be used for the synthesis of G and H in a non-disjoint decomposition of the form $F = H(G(B), A)$ where B and A are non-disjoint subsets of the variables of F. The following function can be decomposed in the above non-disjoint form L:

$$F(x_1, x_2, x_3, x_4) = \overline{x_1}\ \overline{x_2}\ \overline{x_3}\ \overline{x_4} + \overline{x_1}\ \overline{x_2}\ x_3\ \overline{x_4} + \overline{x_1}\ x_2\ x_3\ x_4 +$$

$$x_1\ \overline{x_2}\ \overline{x_3}\ \overline{x_4} + x_1\ \overline{x_2}\ x_3\ x_4 + x_1\ x_2\ x_3\ \overline{x_4} = \sum(0, 2, 7, 8, 11, 14)$$

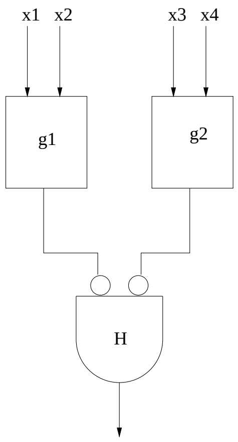


Figure 43: *The final realization of function H*

x1 \ x2 x3 x4	C1	C2	C3	C4	C5	C6	C7	C8
0	1	0	1	0	0	0	0	1
1	1	0	0	1	0	0	1	0

Figure 44: *Function with bound set* $x_2x_3x_4$

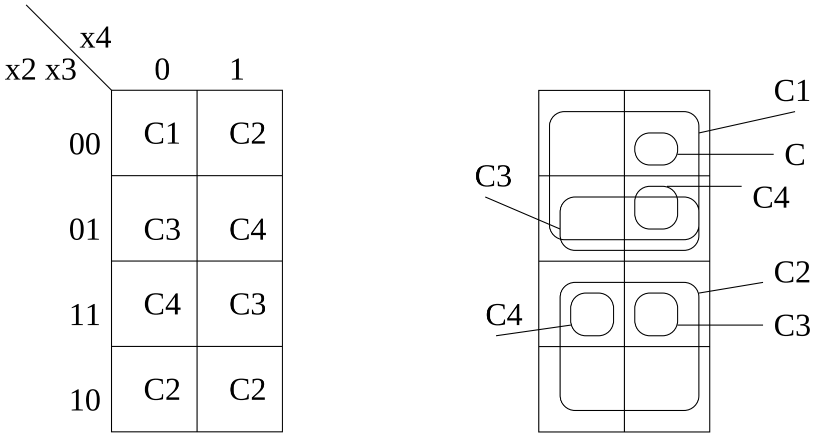


Figure 45: *Notation for columns and groups*

This function with bound set $x_2x_3x_4$ is drawn in Fig. 44:
Functions $G(x_2, x_3, x_4)$ and $H(G(x_2, x_3, x_4), x_1, x_3)$ are chosen for the simple non-disjunctive form above ; as the decomposition chart of F shows, the column multiplicity is 4, but because of the selected non-disjoint bound/free sets, $G(x_2, x_3, x_4)$ will be a single output function:
The input cubes for each column (G output) are:

000	C_1
001	C_2
100	C_2
101	C_2
010	C_3
111	C_3
011	C_4
110	C_4

See, Figure 45.
The covers for input-cube sets are (0XX for C_1 generated by treating 001 cube of C_2, 010 cube of C_3, and 011 cube of C_4 as don't cares, etc.)

0XX	C_1
001	C_2
1XX	C_2
01X	C_3
111	C_3
011	C_4
110	C_4

since covers for C_2, C_3 , and C_4 intersect the cover of C_1, $C_2\ C_3\ C_4\ >\ C_1$.

The covers for C_4, C_3 intersect the cover of C_2 thus $C_4\ C_3\ >\ C_2$. Similarly we have $C_4\ >\ C_3$.
The input constraints matrix for H is given by table from Fig. 46:
The corresponding seed pairs are:

x1 x2	C1	C2	C3	C4
00	1	0	-	-
01	-	-	1	0
11	1	0	-	-
10	-	-	0	1

(a)

x1	C1	C2	C3	C4
0	1	0	1	0
1	1	0	0	1

(b)

Figure 46: *The input constraints matrix for H*

x2 \ x3 x4	00	01	11	10
0	1	0	0	1
1	0	0	1	0

(a)

x1 x3 \ G	0	1
00	1	0
01	1	0
11	0	1
10	1	0

(b)

Figure 47: *(a) The K-map for the predecessor block, (b) The successor* $H(G(x_2, x_3, x_4), x_1, x_3)$

$$s1_a = (C_1 \ ; C_2), \qquad s1_b = (C_2 \ ; C_1),$$

$$s2_a = (C_3 \ ; C_4), \qquad s2_b = (C_4 \ ; C_3).$$

After removing the dichotomies which contradict the dominance relations, we are left with:

$$s1_b \ = \ (C_2; \ \ C_1) \text{ and } s2_b \ = \ (C_4; \ \ C_3) \ \ (s3_b \text{ and } s4_a \text{ repeat})$$

Prime dichotomy $p_1 = (\ C_2\ C_4\ ;\ C_1\ C_3\ )$ covers all of the above seed dichotomies. One bit is therefore required to encode the output of G with $C_2 = C_4 = 1$, and $C_1 = C_3 = 0$. However, since in the seeds for $s1_b$ and $s2_b$ the left and right parts have been interchanged (to satisfy the dominance conditions), one needs to complement the codes. In result: $C_1 = C_3 = 1$, and $C_2 = C_4 = 0$. Thus, the K-map for the predecessor block can be created (Fig. 47a).

$$G = \overline{x_2}\ \overline{x_4} + x_2\ x_3\ x_4$$

The successor $H(G(x_2, x_3, x_4), x_1, x_3)$ is derived from the original decomposition chart for F, as shown in Figure 47b.

Note that the $G = 1$ column corresponds to C_1 and C_3 entries in the original chart (for the corresponding x_1 and x_3 values). Similarly the $G = 0$ column maps the C_2 and C_4 values from the original chart. From the above K-map:

$$H(G(x_2, x_3, x_4), x_1, x_3) \;=\; (\overline{x_1} + \overline{x_3})G + x_1 x_3 \overline{G}$$

The non-disjoint bound/free set selection and encoding used above produced a single output predecessor function and correct image H of F.

10 INPUT-OUTPUT ENCODING COMBINED WITH COLUMN MINIMIZATION

This section addresses some of the well known problems that are dealt with in input and output encoding, and, based on that we develop one more step-by-step method to encode symbols in order to achieve a more simplified total of the predecessor and the successor functions. Techniques used involve the selection of a suitable cover set, heuristics to optimize the quality of encoding at low computational cost, multiple constraint satisfaction using an edge-weighted connection graph, and use of Hamming distances to aid in assigning codes which will result in simpler functions.

The format of this section is as follows: subsection 10.2 introduces some of the basic definitions, in subsection 10.3 the primary encoding phase is covered, in subsection 10.4 the final phase of the encoding approach is presented, in subsection 10.5 an example encoding problem is solved using the encoding approach presented in subsections 10.3 and 10.4, and finally in subsection 10.6 conclusions are presented.

10.1 Basic Definitions

$H(A, G(B, C))$ represents the decomposed function such that $F = H(A, G(B, C))$.

The sets A and B are the free and bound sets respectively where $A \cup B = X$, and $A \cap B = \emptyset$.

The set C is some subset of A when a nondisjoint decomposition is necessary and is an empty set otherwise.

A *Compatible Class* (CC) is a set of columns that are mutually compatible.

Maximum CC (MCC) is a CC that cannot include any additional columns and still maintain compatibility.

Decomposition Function Cardinality (DFC) is a measure used to evaluate the quality or effectiveness of a decomposition. More specifically it is an integer value which is equal to $DFC = \sum_{i=1}^{n} 2^{I_i} \times O_i$ where I_i and Oi are the number of inputs and outputs to block i and n is the total number of blocks.

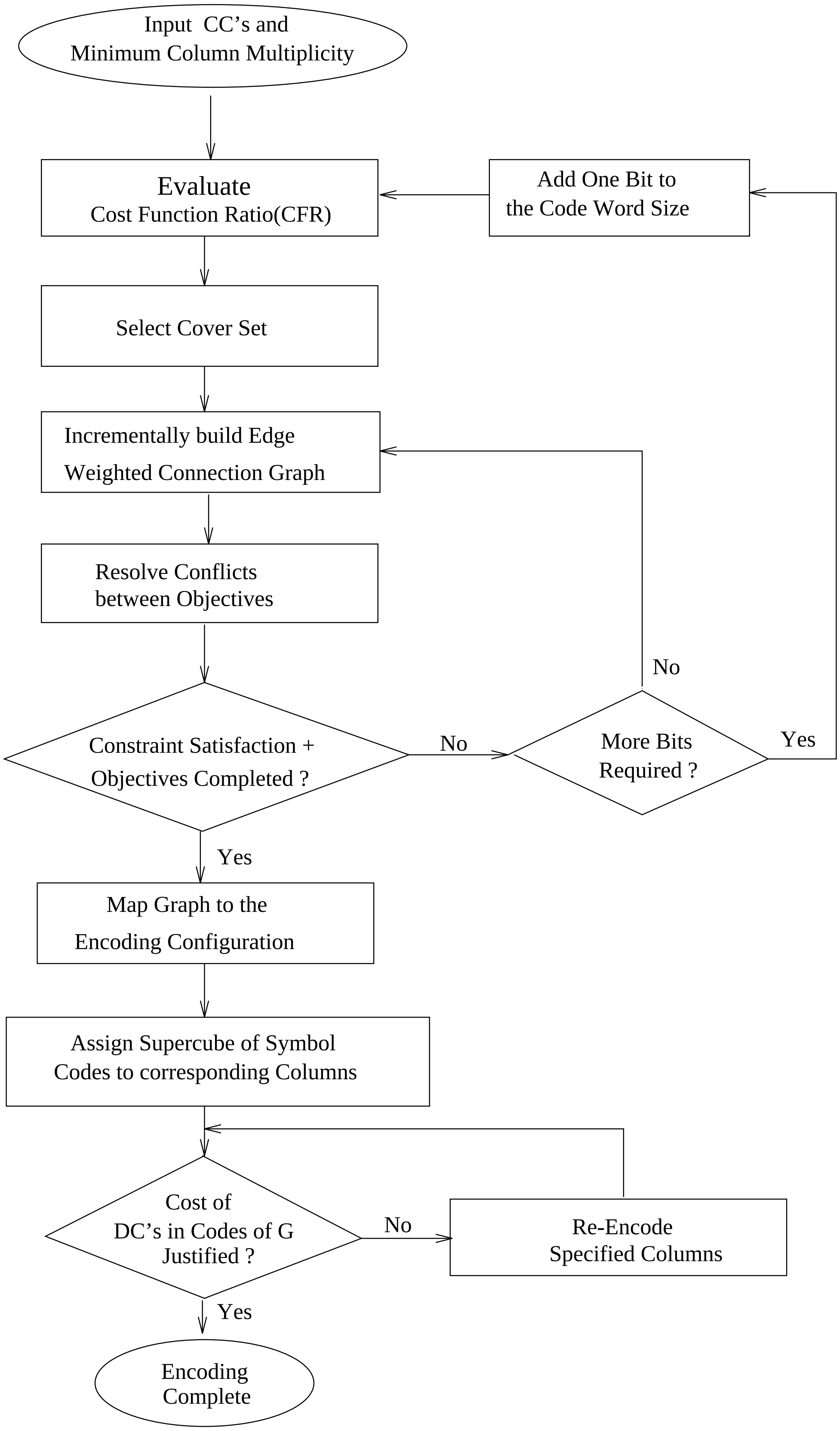
10.2 PHASE I: Primary Encoding Phase

10.2.1 Evaluation of the cost function ratio G_{DFC}/H_{DFC} and selection of a suitable cover set

The purpose of this subsection is to outline a heuristic procedure that is used to give higher priority to the encoding of a sub-function which is much larger than it's counterpart sub-function. It is believed that giving priority will increase the overall number of don't cares in G and H (combined) and that this will ultimately result in a lower DFC, as well as in a reduction of the computation time. The cost function ratio is an approximate measure of the relative sizes of the predecessor and successor functions. It is defined as follows:

Cost Function Ratio $CFR = Cost1 = G_{DFC}/H_{DFC}$

Where G_{DFC} is the DFC of sub-function G and H_{DFC} is the DFC of sub-function H. CFR is used to give a very rough estimate of the ratio of complexity and/or computation time required by each. This in turn is used as criteria for selecting percentages of strategies. These percentages are used to


Input CC's and
Minimum Column Multiplicity
Evaluate
Cost Function Ratio(CFR)
Add One Bit to
the Code Word Size
Select Cover Set
Incrementally build Edge
Weighted Connection Graph
Resolve Conflicts
between Objectives
No
Constraint Satisfaction +
Objectives Completed ?
No
More Bits
Required ?
Yes
Yes
Map Graph to the
Encoding Configuration
Assign Supercube of Symbol
Codes to corresponding Columns
Cost of
DC's in Codes of G
Justified ?
No
Re-Encode
Specified Columns
Yes
Encoding
Complete

determine how a *cover set* should be chosen in order to provide more efficient encoding of the larger sub-function.

The following are the ranges that have been set for the CFR ratio:

Range 1: $CFR \leq .25$
Range 2: $.25 < CFR \leq .5$
Range 3: $.5 < CFR \leq 2$
Range 4: $2 < CFR \leq 4$
Range 5: $CFR > 4$

For Range 1, X = 75 U = 0.0
For Range 2, X = 50 U = .25
For Range 3, X = 33 U = .5
For Range 4, X = 25 U = .75
For Range 5, X = 15 U = 1.0

where X is the percent of don't cares, and U is the parameter to be used in function $FLOOR$ below. The cover set is selected using the following heuristics:

1. Find the top X % of the columns which have the greatest number of specified output values.

2. From the minimal cover of columns with symbols (groups of columns), remove from symbols all redundant occurences of the columns found in step 1). [1] All redundant occurences of other columns should remain in symbols. Keep track of which symbols each column is compatible with.

3. Add the number of $FLOOR(U \times number\_of\_unused\_codes)$ classes to the minimal cover from step 2) to form the new cover. Selection of the additional classes (symbols) is performed as follows: From the minimal cover, select the symbol(s) which have the greatest number of columns that are compatible with $2^k - 1$ symbols, k=1,2,.... Before adding these new symbols to the cover found in step 2), remove ALL occurences of the top X % of columns which were found in step 1).

10.2.2 Edge-Weighted Connection Graph

The *Edge-Weighted Connection Graph (EWG)* is a data structure that will be used by us.

What exactly is the edge-weighted connection graph? The EWCG as defined here is a graph with nodes corresponding to symbols and edges between any two nodes signifying that there exists at least one column which is compatible with both symbols connected by the edge. With few exceptions, each edge has an associated weight (integer value) which is equal to the number of columns which are compatible with the same two symbols. This is somewhat similar to prime implicants covering minterms.

What is the purpose of using an EWCG?

There are two main reasons why an EWCG was used in the encoding approach presented in this section. One of the reasons is to enable multiple types of constraints to be satisfied together in a systematic and controlled manner which might not otherwise be feasible to attempt. The other major

[1] The number of symbols in the minimal cover is equal to the minimum column multiplicity, and the symbols or classes do not necessarily correspond to MCC's.

reason is that once the graph has been constructed following the three rules outlined below, then the graph may be mapped directly to a hypercube of dimension n (where n is the number of encoding bits in the symbolic codes). Also note that this will ensure that all symbols connected by an edge in the graph may be given code words which differ by only one bit.

How is the EWCG built and what are the rules that must be satisfied?

The graph is built by incrementally adding edges between nodes (symbols) in the graph, and systematically checking whether any of the Rules 1 - 3 given below are violated. If no rules are violated, an edge and an associated edge weight is assigned between the two nodes in the graph. The rules that must be satisfied are given below:

Rule 1: No node may have more edges than the number of bits in the code word.

Rule 2: No cycles of odd length.

Rule 3: No two intersecting faces (face A and face B) of size four may have more than two symbols in common.

Procedure for resolving rule violations:

1. If there is a violation in *Rule* 1, then a determination is made as to whether the value of the edge in question is worth the cost of removing an edge(s).
2. If it is not worth the cost ($Cost2 \leq 0$), then no changes are made and the edge in question is removed from the list of edges yet to be considered.
3. If so, then *Rule* 2 is checked.
4. If *Rule* 2 is violated then a similar determination is made as to whether the removal of another edge(s) is worth the cost of assigning the new edge being considered.
5. If it is not worth the cost ($Cost2 \leq 0$), then no changes are made and the edge in question is removed from the list of edges yet to be considered.
6. If so, then *Rule* 3 is checked.
7. If *Rule* 3 is violated, then no changes are made and the edge in question is removed from the list of edges yet to be considered.
8. If there is not a violation in *Rule* 1, then *Rule* 2 is checked.
9. If *Rule* 2 is violated then a determination is made as to whether the removal of an edge(s) is worth the cost of assigning the new edge being considered.
10. If it is not worth the cost ($Cost2 \leq 0$), then no changes are made and the edge in question is removed from the list of edges yet to be considered.
11. If so, then *Rule* 3 is checked.
12. If *Rule* 3 is violated, then no changes are made and the edge in question is removed from the list of edges yet to be considered.
13. If there is not a violation in *Rule* 1 or *Rule* 2, but there is a violation in *Rule* 3, then a determination is made as to whether the removal of a different edge(s) is worth the cost of assigning the new edge being considered.

14. If it is not worth the cost($Cost2 \leq 0$), then no changes are made and the edge in question is removed from the list of edges yet to be considered.

The cost associated with a particular assignment is determined via a cost function which takes into account several factors. The cost function is not called, unless certain rule is violated. The cost function is formulated as follows:

$$Cost2 = N - P \tag{17}$$

$$N = EW_{jk} + .5W_{jk}^{u} \tag{18}$$

$$P = EW_{yz} + .5W_{yz}^{u} \tag{19}$$

$$EW_{xy} = f_{xy} + W_{xy}^{p} + 1 \tag{20}$$

$$W_{xy} = \sum_{i=4}^{m} f_{xy_i} \times w_i \tag{21}$$

Variables defined:

N is a weighted value used for the new edge being considered.

P is a weighted value used for one of the edges being considered for removal from the graph.

m is the size of the largest group of symbols compatible with any column.

EW_{xy} is the edgeweight for the edge between S_x and S_y.

W_{xy}^{u} is a temporary weight given to the pair of symbols x and y corresponding to unplaced groups.

W_{xy}^{p} is a weight given to the pair of symbols x and y corresponding to placed groups.

f_{xy} (frequency) is the number of columns that are compatible with groups of size 3 or less and which include symbols S_x and S_y.

f_{xy_i} is the number of columns compatible with groups of symbols of size i which contain symbols S_x and S_y.

w_i is a weight associated groups of size i and is equal to the number of don't cares that the resulting column code will have if the maximum number of symbols in a group of size i are satisfied in a single face imbedding constraint ($w_i = FLOOR(Log_2 i)$).

The first set of edges to be considered for assignment to the graph are the 1/3 of the symbol pairs S_{ij} that are compatible with the greatest number of columns. As stated before, each of these candidate edges must be checked for rule violations before being assigned. The order of consideration of these edges is in descending order based on the frequencies $f_{xy_{total}}$ for each pair of symbols which are not assigned yet. Where $f_{xy_{total}}$ is the total number of columns which are compatible with symbols S_x and S_y.

The second set of edges to be considered for assignment to the graph are the edges corresponding to symbol pairs that are contained in groups of symbols of size four or larger. The order of consideration of these edges is in descending order based on the weights W_{ij}^{u} for each pair of symbols which are not assigned yet. Hence, the first edge in this set to be considered is the edge with the largest weight W_{ij}^{u}.

The last set of edges to be considered for assignment are all remaining pairs of symbols not considered yet. The order of consideration is in descending order of $f_{xy_{total}}$.

Procedure for Incrementally Building the Graph

i) From the first set of edges to be considered, check the edge having the greatest $f_{xy_{total}}$ for rule violations. If there are rule violations, follow the procedure outlined for resolving rule violations above. If there are no rule violations or the violations were resolved, assign an edge between S_x and S_y and an associated edge weight EW_{xy}.

ii) Remove S_{xy} from the first set of edges to be considered for assignment to the graph. Also, if the edge S_{xy} was assigned, then remove all groups of three symbols which contain both S_x and S_y from the list LGA of groups to be assigned. If the edge corresponding to S_{xy} cannot be assigned, then remove all groups from the list LUG of unassigned groups that can not be satisfied as a result. Then take the edge with the next largest $f_{xy_{total}}$ and repeat step i).

iii) Repeat step i) until there are no more pairs of symbols S_{xy} in the first set to be considered.

iv) Repeat step i) for the second set of edges to be considered for assignment to the graph.

v) Repeat step i) for the third set of edges to be considered for assignment to the graph.

vi) Now that the graph has been constructed, check each of the unsatisfied constraints to see if removing the necessary edge(s) from the graph and adding the unsatisfied constraint will be worth the cost (i.e. $Cost2 < 0$).

   If so, then update the graph accordingly.

vii) Finally, assign any of the *Optional Constraints* that can be satisfied without removing any edges from the graph.

   By "Optional Constraints" we understand constraints that if satisfied will not yield more don't cares in a particular code word, but will provide an optional encoding for certain columns. This can be used to reduce Hamming distances between the output codes in sub-function G.

10.2.3 Mapping the Graph to the Hypercube

This step simply involves assigning nodes in the graph to vertices in the hypercube in such a way that nodes that are connected by an edge in the graph will be assigned to adjacent vertices in the hypercube. Each vertex of the hypercube has a binary codeword which differs in exactly one bit from each of the other vertices that it is connected to by an edge.

The process of mapping to the hypercube is as follows:

1) Start with the largest group(constraint) of 2^k symbols and assign the group to 2^k codes such that the supercube of these codes does not contain any of the codes not in the constraint.

   Example:

   constraint $= (S_1, S_4, S_3, S_7)$

000 S_1

001 S_4

011 S_3

010 S_7

---

0 - - supercube contains only codes contained in the constraint.

2) Find the largest group (constraint) of 2^k symbols which has the largest intersection with the group assigned in 1). Assign this group.

3) Find the largest group (constraint) of 2^k symbols which has the largest intersection with more than one of the previously placed groups(if one exists).

4) Repeat step 3) until no more groups intersect more than one of the previously placed groups.

5) Repeat step 2) until all groups which intersect with the group assigned in step 1) have been placed.

6) If any groups have not been placed yet, then repeat step 1) for the largest of the remaining groups.

7) If any nodes (symbols) in the graph do not have any edges, they may be assigned to any of the remaining codes.

For symbols in the graph which do not have any edges, use of Hamming distances will greatly improve the selection from among the remaining codes. Symbols without edges correspond to classes of columns that do not appear in any other classes (i.e. they are disjoint output classes).

10.3 Assignment of Supercube of Symbol Codes to Corresponding Columns

For many of the columns, the task of assigning them codes is trivial once the symbol codes have been established. However, certain columns may not be assigned to 2^k symbol codes (where k is an integer). In this case, an objective cost function is used to determine which combination of symbol codes to assign to a column being considered. For example, if a column is compatible with five different symbols, then there are five different combinations of four(2^2) symbol codes which can be used to encode the column. Of these five, certain combinations may not be able to be combined because they may include a code of a symbol which is not one of the symbols compatible with the column(i.e. one or more symbols may not satisfy a particular face embedding constraint).

Once the symbol codes have been assigned, a function $ENCODE(C_i)$ is called repeatedly to encode all columns. The trivial and non-trivial cases handled by function $ENCODE(C_i)$ are explained below. There are two conditions that must be checked for each column. If both conditions are satisfied the column is considered a trivial case and is encoded accordingly. If either condition is not satified then the column is treated as a non-trivial case.

Conditions required for trivial encodings:

1) The number of symbols that a column is compatible with is equal to 2^k(where k is an integer).

2) The supercube of 2^k symbol codes which column C_i is compatible with do not include any of the codes not in the constraint.

For the trivial cases, the supercube of 2^k symbol codes may be assigned as the column's code word.
Examples:
000 S1
010 S2
—-
0-0 supercube = column code

110 S4
101 S6
111 S9
100 S3
—-
1- - supercube = column code

For non-trivial cases, the assignment of codes is as follows:

Find all combinations of 2^k symbols which column C_i is compatible with(where $2^k < m < 2^{k+1}$ and m is the total number of symbols that C_i is compatible with). Store the supercube of each combination in a temporary assignment list TAL. Remove any codes from this list which violate the face embedding constraints. Then from the remaining candidate codes determine which code will result in the smallest hamming distances between adjacent codes in sub-function G.

This is done using a cost function which determines the sum of the hamming distances between all codes in G (already assigned) which are adjacent to the code of column C_i. The cost function is as follows:

$$Cost3 = \sum_{j=1}^{n} HAM\_DIST(CODE(C_i), CODE(C_j))$$

where C_i is the column which is being considered for an encoding and C_j is a column which has its corresponding code in G adjacent to the code of C_i. The value "n" is the total number of codes in G which are adjacent to the code of C_i, or more simply, n is equal to the number of variables in the bound set. $CODE(C_i)$ and $CODE(C_j)$ are the corresponding codes of the two columns.

The HAM_DIST function returns the number of bits in two code words which are not compatible (i.e. a "1" in one code word where there is a "0" in the other code word—don't cares are considered compatible with any bit value).

The candidate code with the lowest cost is assigned as the column code. This procedure is repeated for all columns remaining to be encoded from the list $LNTC$ of non-trivial cases.

10.4 PHASE II: Finalizing the encoding

10.4.1 Determining if the elimination of certain Don't Cares in sub-function H is justified

1) For each symbol S_i do the following:

    i) Find the position of all don't cares in S_i that would result if all columns which appear in S_i and at least one other symbol(class) were removed.

    ii) For each don't care found in step 1), find the number of columns (up to 4) which have a care bit in the corresponding position (only consider columns which are found in S_i and at least one other symbol). Add them to a list corresponding to the don't care being considerd and increment a count for each column (this will provide the number of don't cares covered in S_i by each column–this count may be used in the cost function evaluation). If four columns are found which cover the corresponding don't care then take no action.

iii) Repeat step ii) until a don't care is found which is covered by less than four coulmns or until no more don't cares are left to be checked.

iv) After all the lists of columns have been found which cover don't cares in S_i, then for each list which contains only one column evaluate the cost function below:

$$Cost4 \;=\; 1- \mid DC(S_i, C_j) \mid \; + ADJ\_IN\_G(C_j) - ADJ\_IN\_H(S_i, C_j) \qquad (22)$$

where:

$DC(S_i, C_j)$ is the position(s) of a don't care in S_i covered only by column C_j,

$\mid DC(S_i, C_j) \mid$ is the total number of don't cares in S_i covered only by C_j,

$ADJ\_IN\_G(C_j)$ or "Adjacency factor in G" is a measure of how many adjacent output codes in sub-function G will no longer be compatible if the code for symbol S_i is removed from the column code,

$ADJ\_IN\_H(S_i, C_j)$ or "Adjacency factor in H" is equal to the number of outputs in sub-function H (adjacent to the output position(s) in S_i under consideration) which will remain incompatible with the care outputs of C_j if C_j is not removed from symbol S_i.

If Cost4 ≤ 0 then re-encode column C_j and remove it from all the lists generated in ii). Re-encoding of columns is accomplished by function RE_ENCODE(S_i, C_j) which is described in the next section.
Repeat step iv) until all lists have been checked.

2) Repeat step 1) until all symbols have been checked.

3) Make additional passes as needed (it means, repeat step 2) until no more changes are made).

10.4.2 Re-Encoding of Columns which have a low Benefit/Cost ratio

If the value of Cost4 determined in the previous section is less than or equal to zero, then the function RE_ENCODE(S_i, C_j) is called to re-encode column C_j. The process of re-encoding C_j is as follows:

1) Remove S_i from the set of symbols which column C_j can be encoded with.

2) Then call function ENCODE(C_j) to give a new encoding to column C_j.

Note: $ENCODE(C_j)$ is the same function used in the primary encoding phase.

10.5 Encoding Example Solved Step-by-Step

Shown in Figure 48 is the function F which is being decomposed in this example. Refer back to previous sections describing the encoding process as needed for further clarification. Since this example was solved using the minimum number of MCC's prior to completion of the set cover selection procedures, the procedures outlined in section 10.3.1 are not included in this example.

F

ab \ cdef	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
00	0	1	0	0	-	0	0	1	1	0	-	-	0	0	1	-
01	-	-	-	0	-	1	0	-	-	0	0	-	1	-	-	0
11	0	1	-	0	1	0	-	-	1	1	-	-	1	-	-	1
10	-	-	-	0	-	-	0	-	1	-	0	0	-	-	0	0
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	S2 S3	S5 S6	S1 S2 S3 S4	S3	S1 S4 S5 S6	S2	S3 S4	S5 S6	S6	S4	S3 S4 S5	S1 S2 S3 S4 S5	S1	S1 S2 S3 S4	S5	S4 S5

H

ab \ xyz								
	0	0	0	0	1	1	-	-
	1	1	0	0	-	-	-	-
	1	0	0	1	-	1	-	-
	-	-	0	0	0	1	-	-
	S1	S2	S3	S4	S5	S6		

Function F shown with corresponding symbols each column is compatible with.
Also shown is sub-function H with a minimal cover of disjoint classes of columns.

Figure 48: *Function F for Encoding Example*

10.5.1 Building the Edge-Weighted Connection Graph

STEP 1)
Identify the first set of edges to be considered for assignment to the Edge-Weighted Connection Graph (1/3 of symbol pairs which are compatible with most columns). Each of the edges(symbol pairs) in this set is marked with a star in Figure 49(b).
STEP 2)
Begin assigning edges to the graph starting with the edge corresponding to the pair of symbols denoted $S_{ij}(S_i$ and $S_j)$ that has the greatest frequency $f_{ij_{total}}$. $f_{ij_{total}}$ is the total number of columns which are compatible with symbols S_i and S_j. Initially the graph appears as in Figure 3(a).

The first edge to be considered is S_{34}. Since there are no other edges in the graph, no rules are violated and the edge and it's corresponding edge weight are assigned to the graph. The edge weight is calculated as follows:

$$EW_{34} = f_{34} + W^p_{34} + 1 = 2 + 0 + 1 = 3$$

$W^p_{34} = 0$ since no groups of size 4 or greater that have been placed that also contain symbols S_3 and S_4.

The graph now appears as in Figure 50b.

The next edge to be assigned is the one with the next largest $f_{ij_{total}}$. It turns out that there is a three way tie for the next largest frequency. Use the following criteria for breaking ties:

For each S_{ij} that are tied, find the symbol $S_k \in S_{ij}$ which has the most edges. Then compare these values and select the pair which has the lowest number of edges from these(i.e. select the least from the greatest). If there is still a tie select randomly.

In this case there is still a tie so S_{14} is chosen randomly and checked for rule violations. Since there are no violations, it is assigned to the graph.

$$EW_{14} = f_{14} + W^p_{14} + 1 = 0 + 0 + 1 = 1$$

Similarly for the next two edges which had the next largest $f_{ij_{total}}$. No rules were violated so the edges S_{23} and S_{45} are assigned to the graph.

$$EW_{23} = f_{23} + W^p_{23} + 1 = 1 + 0 + 1 = 2$$
$$EW_{45} = f_{45} + W^p_{45} + 1 = 2 + 0 + 1 = 3$$

This concludes the assignment of the first set of edges. The graph now appears as in Figure 50c.

STEP 3)
Determine the second set of edges to be considered. The second set of edges to be considered can be found in the column W_{ij} in Figure 49b which are not equal to zero. An example of how one of these weights is calculated is as follows:

a)

Group Size	C_i-(set of symbols that column C_i is compatible with)
5	C_{11}-(1,2,3,4,5)
4	C_2-(1,2,3,4) C_{13}-(1,2,3,4) C_4-(1,4,5,6)
3	C_{10}-(3,4,5)
2	C_{15}-(4,5) C_6-(3,4) C_7-(5,6) C_1-(5,6) C_0-(2,3)
1	C_3-3 C_5-2 C_8-6 C_9-4 C_{12}-1 C_{14}-5

b)

Pairs of Symbols	$f_{ij_{total}}$	W_{ij}
S_{12}	3	6
S_{13}	3	6
S_{14}	* 4	8
S_{15}	2	4
S_{16}	1	2
S_{23}	* 4	8
S_{24}	3	6
S_{25}	1	2
S_{26}	0	0
S_{34}	* 5	6
S_{35}	2	2
S_{36}	0	0
S_{45}	* 4	4
S_{46}	1	2
S_{56}	2	2

a) Table showing all columns and the corresponding sets of symbols they are compatible with. b) Table showing the total number of columns compatible to each symbol pair(frequency) and the corresponding weight(number of don't cares that can be introduced from groups greater than size 3 if all constraints are satisfied).

Figure 49: *Tables (a) and (b) for the Example*

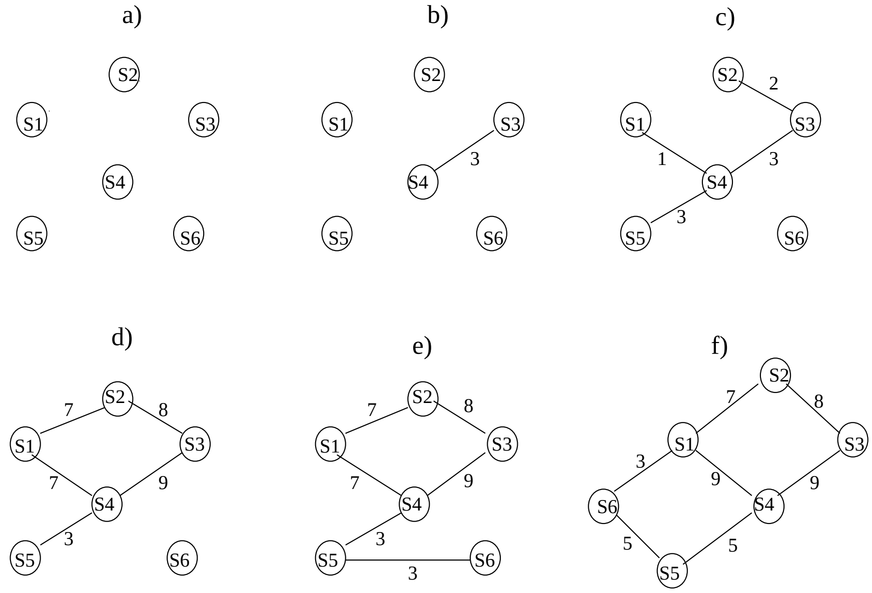


Figure 50: *Construction of the Edge Weighted Connection Graph.*

$$W_{xy} = \sum_{i=4}^{m} f_{xy_i} \times w_i \tag{23}$$

$$W^u_{14} = \sum_{i=4}^{5} f_{14_i} \times w_i = (f_{14_4} \times w_4) + (f_{14_5} \times w_5) = (3x2) + (1x2) = 8 \tag{24}$$

Recall that f_{xy_i} is the number of columns compatible with groups of symbols of size i which contain symbols S_x and S_y

STEP 4)

Assign edges in the second set in descending order according to the values of W^u_{ij}. The first edge to be considered is either S_{23} or S_{14} since they tie for the greatest W^u_{ij}. However, since both have already been placed, simply find the next largest W^u_{ij} and remove S_{23} and S_{14} from the second set of edges to be considered. There is a three way tie for the next edge to be considered(S_{12},S_{13} and S_{24}). Use the same criteria for breaking ties as done for the first set of edges. The tie is broken and S_{12} is selected next. The following was the tie breaking criteria used:

Max edges from S_{12} is $S_1 = S_2 = 1$.
Max edges from S_{13} is $S_3 = 2$.
Max edges from S_{24} is $S_4 = 2$

MIN(MAX(1,2),MAX(1,3),MAX(2,4)) = MIN(1,2,2) = 1.

There are no rule violations so edge S_{12} and it's corresponding edge weight are assigned. This satisfies the first group constraint(or face embedding constraint) of size 4 where the set of symbols in the group constraint is (1,2,3,4).

Once edge weights have been updated, the graph will appear as in Figure 50d. Note that 3 columns are compatible with the group of symbols(1,2,3,4). For each edge in the group constraint, an additional weight of 2(for 2 don't cares) is added to each edge weight for each column that is compatible with the group constraint(i.e. W^p_{ij} = 3 x 2 = 6).

Update edges as follows:

$$EW_{12} \;=\; f_{12} + W^p_{12} + 1 \;=\; 0 + 6 + 1 \;=\; 7 \tag{25}$$

W^p_{12} is calculated as follows:

$$W^p_{12} \;=\; \sum_{i=4}^{5} f_{12_i} \times w_i \;=\; (f_{12_4} \times w_4) + (f_{12_5} \times w_5) \;=\; (2x2) + (1x2) = 6 \tag{26}$$

Similarly, the other edges in the group constraint are updated.

$$EW_{14} \;=\; f_{14} + W^p_{14} + 1 \;=\; 0 + 6 + 1 \;=\; 7 \tag{27}$$

$$EW_{23} \;=\; f_{23} + W^p_{23} + 1 \;=\; 1 + 6 + 1 \;=\; 8 \tag{28}$$

$$EW_{34} \;=\; f_{34} + W^p_{34} + 1 \;=\; 2 + 6 + 1 \;=\; 9 \tag{29}$$

The next edge selected is S_{13}. *Rule* 1 is not violated, but *Rule* 2 is violated, because a cycle of odd length is identified (S_{12}, S_{23}, S_{13}). Therefore the cost function *Cost*2 must be evaluated to determine if adding the new edge to the graph and removing one of the other edges in the cycle will be worth the cost. The following are two evaluations of *Cost*2 if S_{12} or S_{23} are to be removed:

For S_{12}, $Cost2 = N$ - P = $(EW_{13} + .5W^u_{13}) - (EW_{12} + .5W^u_{12})$ = (0 + .5(6)) - (7 + .5(0)) = 3 - 7 = -4

Since Cost2 $\leq$ 0, this change is not worth the cost and edge S_{12} remains in the graph.

For S_{23}, $Cost2 = N - P = (EW_{13} + .5W^u_{13}) - (EW_{23} + .5W^u_{23})$
= (0 + .5(6)) - (8 + .5(0)) = 3 - 8 = -5

Since Cost2 $\leq$ 0, this change is not worth the cost. Therefore no changes are made and edge S_{13} is removed from the list to be considered.

The next edge selected is S_{24}. Again *Rule* 1 is not violated, but *Rule* 2 is violated, because a cycle of odd length is identified (S_{12}, S_{24}, S_{14}). The cost function *Cost*2 is evaluated again for each edge that can break the cycle.

For S_{12}, $Cost2$ = -4

Since $Cost2 \leq 0$, this change is not worth the cost.

For S_{14}, $Cost2$ = -5

Since Cost2 $\leq$ 0, this change is not worth the cost. Therefore no changes are made and edge S_{24} is removed from the list to be considered.

The next edge considered is S_{15}. Again, a cycle of odd length is identified (S_{15}, S_{45}, S_{14}).

For S_{14}, Cost2 = -6
For S_{45}, Cost2 = -3

Since Cost2 $\leq$ 0, no changes are made and edge S_{15} is removed from the list to be considered.

For the next largest W_{ij}^{u} there is a five way tie. Using the procedure outlined for breaking ties results in the following:

Max edges from S_1 or S_6 = 2
Max edges from S_2 or S_5 = 2
Max edges from S_3 or S_5 = 2
Max edges from S_4 or S_6 = 3
Max edges from S_5 or S_6 = 1

MIN(MAX(1,6),MAX(2,5),MAX(3,5),MAX(4,6),MAX(5,6))
= MIN(2,2,2,3,1) = 1.

Therefore S_{56} is the next edge to be considered. No rules are violated so edge S_{56} is added to the graph. The corresponding edge weight is EW_{56} = 3. The graph now appears as in Figure 50e.

Next, there is a four way tie. Randomly, S_{25} was selected from those above which had 2 as the maximum number of edges connected to one of its symbols. *Rule* 1 and *Rule* 2 are satisfied, but *Rule* 3 is violated because this edge created a new face of size 4 which intersects a different face with more than 2 nodes (symbols) in common. Select edges to be evaluated with S_{25} which are not in the same face(i.e. S_{12} and S_{14} are edges in face$(S_{12}, S_{23}, S_{34}, S_{14})$ but not in face $(S_{25}, S_{45}, S_{34}, S_{23})$. If $Cost2 \geq 0$, make changes to the graph accordingly(add new edge and remove the other).

For S_{12}, $Cost2$ = 3 - 7 = -4
For S_{14}, $Cost2$ = 3 - 8 = -5

Since Cost2 $\not\geq$ 0, no changes are made and edge S_{25} is removed from the list to be considered.

The next edge considered is S_{35}. Rule 2 is violated because a cycle of odd length is identified(S_{35}, S_{45}, S_{34}).

For S_{34}, Cost2 = 2 - 9 = -7
For S_{45}, Cost2 = 2 - 4 = -2

Since Cost2 $\leq$ 0, no changes are made and edge S_{35} is removed from the list to be considered.

The next edge considered is S_{35}. There are no rule violations therefore S_{16} is assigned to the graph with its corresponding edge weight. Update the edge weights of those pairs which are elements of the group. Edges updated are:

$EW_{16} = f_{16} + W_{16}^p + 1 = 0 + 2 + 1 = 3$

$EW_{56} = f_{56} + W_{56}^p + 1 = 2 + 2 + 1 = 5$

$EW_{45} = f_{45} + W_{45}^p + 1 = 2 + 2 + 1 = 5$

$EW_{14} = f_{14} + W_{14}^p + 1 = 0 + 8 + 1 = 9$

The graph now appears as in Figure 50f.

This completes the second set of edges to be considered for assignment to the graph. As it turns out, this is the last set of edges for consideration because there are no remaining edges which have not been considered yet.

Therefore this completes the construction of the graph!

10.5.2 Mapping the Graph to the Hypercube

The next step is to map the graph directly to a hypercube. The following mapping of symbols to codes was found using the procedure outlined in section 10.3.3.

First, we select the largest group(constraint) of 2^k symbols and assign it to 2^k codes such that the supercube does not contain any of the codes not in the constraint. This can be ensured by maintaining the connectivity relationship in the hypercube that is found in the graph. It turns out that there is a tie for the largest constraint. The two groups in the graph which are largest are (S_1, S_2, S_3, S_4) and (S_1, S_4, S_5, S_6). Select the group which is compatible with the greatest number of columns. Therefore group (S_1, S_2, S_3, S_4) is selected. Arbitrarily four codes which differ only in one bit are chosen for the constraint

000
001
011
010
—-
0- - supercube does not contain any codes not in the constraint

These codes are assigned so that the connectivity relationship in the graph is maintained.
000 S_2
001 S_3
011 S_4
010 S_1

Next find the largest group (constraint) of 2^k symbols which has the largest intersection with the group already assigned. Assign this group. The group found is (S_1, S_4, S_5, S_6). Since S_1 and S_4 are already assigned, only S_5 and S_6 need to receive codes. The remaining codes available are:

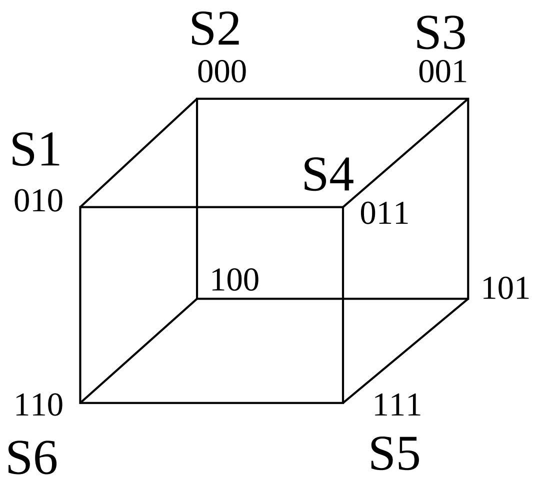


Figure 51: *Assignment of codes resulting from the mapping of the Edge-Weighted Connection Graph to the Hypercube.*

100
101
111
110

S_5 is adjacent to S_4 in the graph and therefore must receive a code which differs by only one bit from the code of S_4. From the availabe codes, the only one which differs from code 011 by one bit is code 111. Therefore S_5 is assigned to code 111. Next, S_6 must be assigned a code which is adjacent to S_1 and S_5. Therefore, it must be given a code from the available codes which differs by only one bit from 010 and 111. Therefore S_6 is coded with 110.

111 S_5
110 S_6

These code assignments and the relationships between the groups in the graph can be seen in the hypercube of Figure 51.

10.5.3 Example Problem—Assignment of Supercube of codes to corresponding columns

Finally, the last part of the primary encoding phase is to assign the supercube of symbol codes to the corresponding columns. Trivial cases correspond to columns which may be assigned to the supercube of codes of all symbols which it is compatible with. There are two types of non-trivial cases. One type of non-trivial case is where a column is compatible with some number of symbols (m) where $2^k < m < 2^{k+1}$ and k is some integer(This implies more than one optional supercube of 2^k codes). The other type of non-trivial case is where the supercube of the codes includes a code which is not in the constraint.

Using the procedure outlined in section 10.3.4, we begin by assigning codes to columns for cases considered trivial. At first all columns which are compatible with 2^k symbols are considered trivial to encode. The supercube is checked to see if any codes are included which are not in the constraint. If so, that column is placed in the list of non-trivial cases $LNTC$ to be coded after all trivial cases. If not, then the column is assigned to the supercube of all codes in the constraint.

For trivial cases the order of assignment of codes does not matter. So, we begin with the columns in order as they appear in Figure 48. The first column to be encoded is column 0 which is compatible with symbols S_2 and S_3. Since there are 2^k symbols compatible with column C_0 (where $k = 1$ in this case), we proceed to check the second condition required for a trivial encoding.

000 S_2
001 S_3
—-
00- supercube

Since the supercube does not contain any codes that are not in the constraint of column C_0, then C_0 is encoded with "00-". Similarly, the remaining columns are coded which do not fall into the category of non-trivial. It turns out that all columns except for C_{10} and C_{11} are encoded as trivial cases (they receive the supercube of all symbols they are compatible with). These can be seen in the Table from Figure 52.

For non-trivial cases we encode columns starting with the largest number of symbols that any one column is compatible with. Therefore column C_{11} is encoded first. The largest group of 2^k symbols that C_{11} is compatible with is 4. There are 5 possible groups of 4 symbols ((1234), (1345),(1245),(1235),(2345)). Of these, only one is possible because the supercube of the others include codes which are not in the constraint. For example, the supercube of symbols (1235) is "$- \ - \ -$". This includes the code of symbol S_6 which is not in the constraint. Therefore C_{11} is encoded with the supercube of symbols (1234) resulting in the code "$0 \ - \ -$".

For the only other non-trivial case(C_{10}), there are 3 possible groups of 2^k symbols codes that C_{10} might be able to be encoded with(k = 1 in this case). The 3 groups are ((34),(35),(45)). Group (35) is not allowed since it's supercube ($- \ - \ 1$) includes the code of an unused code (1 0 1). For the remaining groups (34) and (45) we apply the cost function $Cost3$ to determine which codes(($0 \ - \ 1$) or ($- \ 1 \ 1$)) to select. The following are the columns that are adjacent to column C_{10} in G:

0010 (column C_2)
1000 (column C_8)
1011 (column C_{11})
1110 (column C_{14})

$$Cost3 = \sum_{j=1}^{n} \text{HAM\_DIST}(\text{CODE}(C_i),\text{CODE}(C_j))$$

$$Cost3 = HAM\_DIST(CODE(C_{10}), CODE(C_2)) + HAM\_DIST(CODE(C_{10}),\ CODE(C_8)) +$$
$$HAM\_DIST(CODE(C_{10}),\ CODE(C_{11})) + HAM\_DIST(CODE(C_{10}),\ CODE(C_{14}))$$

For C_{10} encoded with "$0 \ - \ 1$" the cost is:

$$Cost3(C_{10}) = 0 + 2 + 0 + 1 = 3$$

Columns	Symbols	Symbol Codes	Supercube	Acceptable Encodings
0	S2,S3	000 001	00-	000 001 00-
1	S5,S6	111 110	11-	111 110 11-
2	S1,S2,S3,S4	010 000 001 011	0--	010 001 000 011 00- 01- 0-1 0-0 0--
3	S3	000	000	000
4	S1,S4,S5,S6	010 011 111 110	-1-	010 111 011 110 11- 01- -10 -11 -1-
5	S2	000	000	000
6	S3,S4	001 011	0-1	001 011 0-1
7	S5,S6	111 110	11-	111 110 11-
8	S6	110	110	110
9	S4	011	011	011
* 10	S3,S4,S5	001 011 111	--1	001 011 111 0-1 -11
* 11	S1,S2,S3,S4,S5	010 000 001 011 111	---	010 001 000 011 111 00- 01- 0-1 0-0 -11 0--
12	S1	010	010	010
13	S1,S2,S3,S4	010 000 001 011	0--	010 001 000 011 00- 01- 0-1 0-0 0--
14	S5	111	111	111
15	S4,S5	011 111	-11	011 111 -11

Circled codes indicate the column codes determined by the algorithm presented in this paper.

Figure 52: *Application of Supercube Operation in Encoding.*

For C_{10} encoded with " $-$ 1 1" the cost is:

$Cost3(C_{10}) = 0 + 1 + 0 + 0 = 1$

The code resulting in the lowest cost is chosen. Therefore column C_{10} is assigned the code " $-$ 1 1".

Note that the use of the unused codes can improve the quality of the encoding in some cases. The approach in this section does use the unused codes when heuristics in the cover set selection determines that they should be used. However, there is a trade-off and complicated heuristics may be required to ensure that there will be a benefit from particular uses of the unused codes. In the example above one would not benefit "overall" if he would encode C_{10} with the supercube of the three symbols it is compatible with plus an unused code. If one would use the unused code(1 0 1) then the code " $-$ $-$ 1" can be given to column S_{10}. But this forces to assign one of the don't care columns in H to either symbol S_3, S_4 or S_5. If S_5 is chosen, since it has the fewest care bits, then column "1 0 1" in the successor function H will be changed from all don't cares to a partially specified column, as illustrated below:

From To

$-$ 1

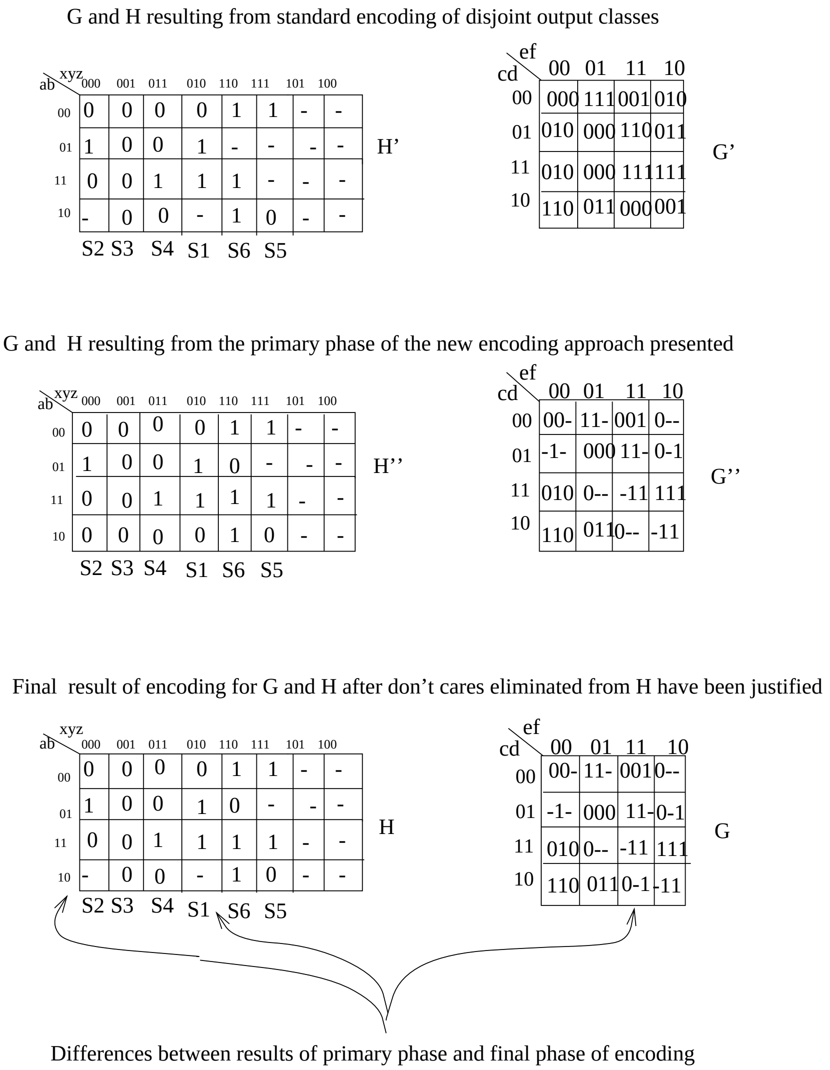


Figure 53: *Subfunctions G and H for the Example.*

– –
– –
– 0

This results in adding one don't care to the predecessor function G and removing two don't cares from the successor function H. In some cases it may be a good trade-off, but in certain cases it may be that far more don't cares would be removed from H (changed to specified values) than would be introduced into G.

10.6 Concluding Remarks

Unlike some approaches which optimize only the input encoding or only the output encoding, the approach presented in this section allows to satisfy multiple constraints and objectives for input and output encodings concurrently in a relatively systematic manner. Though not shown in the example, the method described in the subsection for cover set selection can be used to satisfy the objective to

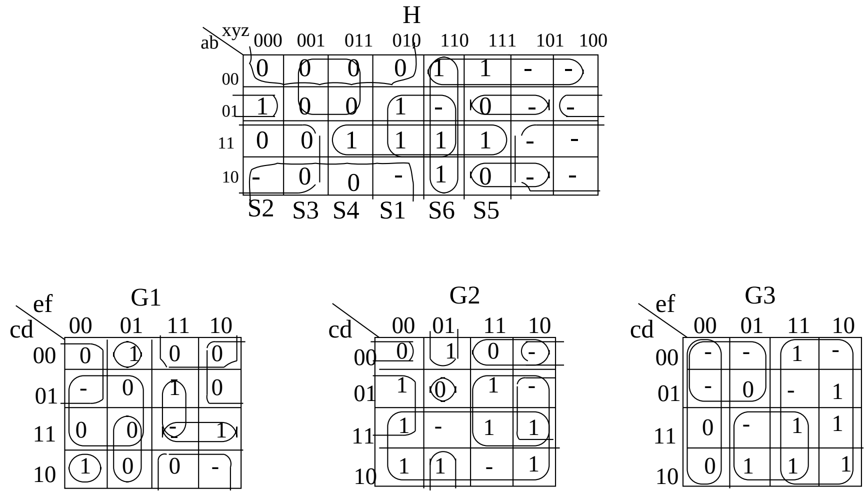


Figure 54: *Minimization of subfunctions G and H for Example*

increase the number of don't cares in the sub-function G while minimizing the loss of don't cares in the sub-function H.

The example of encoding illustrated how a substantial number o don't cares can be introduced into the sub-function G as a result of encoding columns with codes of multiple symbols. Also demonstrated was how multiple constraint satisfaction can be accomplished through a relatively simple set of heuristics applied to an edge-weighted connection graph. Use of minimum Hamming distances between the optional codes and the adjacent codes already assigned illustrated how the complexity of the resulting function could be simplified.

Though the results from small examples calculated by hand are very promising, numerous tests on the implemented algorithm need to be done in order to get an accurate assessment of the potential of this method.

11 CONCLUSION

We have presented several new algoritms:

1. Symbol Encoding by Partitions and Boolean Function Minimization,
2. Column Encoding by Column Selection and Boolean Relation Minimization,
3. Column Encoding by Column Selection and Boolean Function Minimization,
4. Symbol Encoding by Partitions and Multiple-Valued Minimization,
5. Symbol Encoding by Partitions and Multiple-Valued Minimization,
6. Symbol Encoding by Dychotomies and Dominance Relations,
7. Column Encoding by Dychotomies.

The Column Encoding algorithms combine column minimization and symbol encoding methods and take into account the optimization of both the predecessor and successor logic. Most of these methods take some ideas from the existing methods that would perform only the input encoding, the output encoding, or the FSM state assignment. Moreover, some of the proposed methods work on the non-minimized or partially minimized columns, thus essentially enlarging the search space, and potentially improving the results. It is now the task of our group to practically compare these algorithms in computer programs, and test them on one of our decomposers: TRADE, DEMAIN and GUD. The future user of GUD will be able to select either of these approaches.

Since the concept of a Boolean relation is a generalization of the concept of a don't care, the techniques described here can be generalized for column minimization problem with symbolic relations.

12 ACKNOWLEDGMENT

The authors would like to thank Professor Robert Brayton from University of California in Berkeley, Professor Maciej Ciesielski from University in Massachussets in Amherst, and Professor Fabio Somenzi from the University of Colorado in Boulder for helping to find the respective literature. Thanks are also to Prof. Somenzi for the access to his FSM minimizer STAMINA, and to Prof. Ciesielski for the set of his synthesis tools.

13 LITERATURE

This is the selection of only the most important positions in state assignment and encoding. Most of these papers were used in our work.

References

[1] Y.S. Abu-Mostafa, *"Complexity in Information Theory"*, Springer-Verlag, New York, 150pp, ISBN 0-387-96600-5, 1988.

[2] D.B. Armstrong, "A Programmed Algorithm for Assigning Internal Codes to Sequential Machines," *IRE Transactions Electr. Comp.,* Vol. EC-11, pp. 466-472, August 1962.

[3] P. Ashar, S. Devadas, and A.R. Newton, " A Unified Approach to the Decomposition and Redecomposition of Sequential Machines," *Proc. of the 27th A.C.M./I.E.E.E. Design Automation Conference,* pp. 601-606, 1990.

[4] P. Ashar, S. Devadas, and A.R. Newton, "Testability Driven Synthesis of Interacting Finite State Machines," *Proc. of the International Conference on Computer Aided Design,* pp. 273-276, 1990.

[5] M.J. Avedillo, J.M. Quintana, and J.L. Huertas, " State Reduction of Incompletely Specified Finite Sequential Machines," *Proc. Workshop on Logic and Architecture Synthesis,* Paris, May 1990, pp. 107-115.

[6] N.N. Biswas, " State Minimization of Incompletely Specified Sequential Machines," *IEEE Trans. on Computers,* Vol. C-23, pp. 80-84, 1974.

[7] N.N. Biswas, "Implication Trees in the Minimization of Incompletely Specified Sequential Machines," *Int. J. Electron. (GB),* No. 2, pp. 291-298, August 1983.

[8] R. Murgai, R.K. Brayton, and A. Sangiovanni-Vincentelli, "Optimum Functional Decomposition Using Encoding," *Proc. 31st DAC*, pp. 408–414, June 1994.

[9] De Michelli, G., Sangiovanni-Vincentelli, A., Villa, T., "Computer-Aided Synthesis of PLA-Based Finite State Machines," *Proc. IEEE 1983 Intern. Conf. on Computer Aided Design,* pp. 154-156, September 1983.

[10] G. De Micheli, R. Brayton, and A.L. Sangiovanni-Vincentelli, "Optimal State Assignment for Finite State Machines," *IEEE Trans. on Computer-Aided Design,* Vol. CAD-4, pp. 269-285, 1985.

[11] G. De Micheli, and T. Klein, "Algorithms for Synchronous Logic Design," *Proc. of the Intern. Symp. on Circuits and Systems,* pp. 756-761, 1989.

[12] S. Devadas, " Approaches to Multi-Level Sequential Logic Syntesis," *Proc. of the 26th A.C.M./I.E.E.E. Design Automation Conference,* pp. 270-276, 1989.

[13] S. Devadas, and A.R. Newton, "Exact Algorithms for Output Encoding, State Assignment and Four-Level Boolean Minimization," *Proc. Hawaii Intern. Conf. on System Sciences,* pp. 387-396, 1990.

[14] S. Devadas and A. R. Newton, " Exact Algorithms for Output Encoding, State Assignment, and Four-Level Boolean Minimization," *IEEE Trans. on Computer Aided Design,* Jan 1991.

[15] S.Devadas, A.R. Newton, P. Ashar "Sequential Logic Synthesis". pp. 60-64, 1992, *Kluwer Academic Publishers.*

[16] Dolotta, T.A., McCluskey, E.G., "The Coding of Internal States of Sequential Machines," *IEEE Trans. Electr. Comp.,* Vol. EC-13, pp. 549-562, October 1964.

[17] D. Drusinsky, and D. Harel, "State Charts as an Abstract Model for Digital Control Units," *Dept. Applied Math. and Comp. Sci., The Weizmann Institute of Science,* Rehovot, Israel, 1986.

[18] Dunworth, A., Hartog, H.V., "An Efficient State Minimization Algorithm for Some Special Classes of Incompletely Specified Sequential Machines." *IEEE Trans. Comp.,* Vol. C-28, No. 7, p. 531-535, July 1979.

[19] H. Enrich, " A Note on State Minimization of a Special Class of Incomplete Sequential Machines," *IEEE Trans. on Computers,* C-21, pp. 500-502, 1972.

[20] A. Grasselli, and F. Luccio, " A Method for Minimizing the Number of Internal States in Incompletely Specified Sequential Networks, " *IEEE Trans. on Electron. Computers,* EC-14, pp. 350-359, 1965.

[21] G.D. Hachtel, J.K. Rho, F. Somenzi, and R. Jacoby, " Exact and Heuristic Algorithms for the Minimization of Incompletely Specified State Machines," *European Design Automation Conference,* Amsterdam, The Netherlands, pp. 184-191, 1991.

[22] G. Hallbauer, "Procedures of State Reduction and Assignment in One Step in Synthesis of Asynchronous Sequential Circuits," *Proc. Intern. IFAC Symp. on Discrete Systems,* pp. 272-282, Riga, September 30-October 4, 1974.

[23] J. Hartmanis, "On the State Assignment Problem for Sequential Machines, I," *IRE Trans. Electr. Comp.,* Vol. EC-10, p. 157-165, June 1961.

[24] J. Hartmanis, and R.E. Stearns, "Algebraic Structure Theory of Sequential Machines," *Prentice-Hall,* New York, 1966.

[25] R.W. House, and D.W. Stevens, " A New Rule for Reducing CC Tables, " *IEEE Transactions on Computers,* Vol. C-19, pp. 1108-1111, Nov. 1970.

[26] L.N. Kannan, and D. Sarma, " Fast Heuristic Algorithms for Finite State Machine Minimization," *Proc. EDAC 91,* Amsterdam, pp. 192-196, Febr. 1991.

[27] R.M. Karp, "Some Techniques of State Assignment for Synchronous Sequential Machines," *IEEE Trans. Electr. Comp.,* Vol. EC-13 No. 5, pp. 507-518, October 1964.

[28] Z. Kohavi, *Switching and Finite Automata Theory,* New York, McGraw Hill, 1978.

[29] P.K. Lala, "An Algorithm for the State Assignment of Synchronous Sequential Circuits," *Electr. Letters,* Vol. 14, No. 6., 16 March 1978.

[30] B. Lin, and A.R. Newton, " Restructuring State Machines and State Assignment: Relationship to Minimizing Logic Across Latch Boundaries," *Proc. International Workshop on Logic Synthesis,* Research Triangle Park, North Carolina, May 23-26, 1989, paper 5.3.

[31] E.B. Lee, and M.A. Perkowski, " Concurrent Minimization and State Assignment of Finite State Machines," *Proc. of Intern. IEEE Conf. on Systems, Man, and Cybernetics,* Halifax, Nova Scotia, pp. 248-260, 1986.

[32] B. Lin, H.J. Touati, and A.R. Newton, " Don't Care Minimization of Multi-Level Sequential Logic Networks," *Proc. of the Intern. Conf. on Computer Aided Design,* pp. 414-417, 1990.

[33] B. Lin, and F. Somenzi, " Minimization of Symbolic Relations, " *Proc. of the Intern. Conf. on Computer Aided Design,* pp. 88-91, 1990.

[34] F. Luccio, " Reduction of the Number of Columns in Flow Table Minimization, " *IEEE Trans. on Electr. Computers,* Vol. EC-15, pp. 803-805, Oct. 1966.

[35] C. Madre, Coudert, "New Ideas for Set Covering Problems," Proc. DAC 1995.

[36] N.J. Nilsson, "Problem Solving Methods in Artificial Intelligence," *McGraw-Hill,* New York, 1971.

[37] D. Pager, "Conditions for the Existence of Minimal Closed Covers Composed of Maximal Compatibles, " *IEEE Transaction on Computers,* Vol. C-20, pp. 450-452, 1971.

[38] M.C. Paul, and S.H. Unger, " Minimizing the Number of States in Incompletely Specified Sequential Switching Functions, " *IRE Trans. on Electron. Computers,* Vol. EC-8, pp. 356-367, 1959.

[39] P. Paulin, " Horizontal Partitioning of PLA-based Finite State Machines," *Proc. of the A.C.M./I.E.E.E. Design Automation Conference,* pp. 333-338, 1989.

[40] Ch. A. Papachristou and D. Sarma, "An Approach to Sequential Circuit Construction in LSI Programmable Arrays," *IEE Proceedings,* Vol. 130, Pt. E, No. 5, pp. 159-164, September 1983.

[41] M.A. Perkowski, A. Rydzewski, and P. Misiurewicz, "Theory of Logic Circuits. Selected Problems," *Publishers of the Technical University of Warsaw* (book in Polish).

[42] M.A. Perkowski, and N. Nguyen, "Minimization of Finite State Machines in System SuperPeg," *Proc. of the Midwest Symposium on Circuits and Systems,* Luisville, Kentucky, pp. 139-147, 1985.

[43] M.A. Perkowski, and L.B. Nguyen, "An Encoding Program for Concurrent Finite State Machines Realized in Constrained Logic," *Proc. of the Midwest Symp. on Circuits and Systems,* Calgary, Alberta, Canada, 1990.

[44] M. Perkowski, T. Luba, S. Grygiel, M. Kolsteren, R. Lisanke, N. Iliev, P. Burkey, M. Burns, R. Malvi, C. Stanley, Z. Wang, H. Wu, F. Yang, S. Zhou, and J. S. Zhang, "Unified Approach to Functional Decompositions of Switching Functions," *PSU Report*, unpublished.

[45] C.V.S. Rao, and N.N. Biswas, "Minimization of Incompletely Specified Sequential Machines, " *IEEE Trans. on Computers,* Vol. C-24, No. 11, pp. 1089-1100, Nov. 1975.

[46] T. D. Ross, M.J. Noviskey, T.N. Taylor, D.A. Gadd, "Pattern Theory: An Engineering Paradigm for Algorithm Design," *Final Technical Report WL-TR-91-1060,* Wright Laboratories, USAF, WL/AART/WPAFB, OH 45433-6543, August 1991.

[47] R.L. Rudell, and A. Sangiovanni–Vincentelli, "Espresso MV: Algorithms for Multiple-Valued Logic Minimization," *Proc. IEEE Custom Integr. Circ. Conf.,* 1985, pp. 230–234. Rudell

[48] A. Saldanha, paper about encoding.

[49] T. Sasao, "FPGA Design by Generalized Functional Decomposition," in "Logic Synthesis and Optimization, " T. Sasao. (Ed), Kluwer Academic Publishers, pp. 233-258, 1993.

[50] H. Selvaraj, "FPGA-Based Logic Synthesis," *Ph.D. Dissertation,* Warsaw University of Technology, 1994.

[51] Chia-Jeng Tseng, and D.P. Siewiorek, "Facet: A Procedure for the Automated Synthesis of Digital Systems," *Proc. 20-th Design Automation Conference,* p.566.

[52] R.E. Stearns, and J. Hartmanis, "On the State Assignment Problem for Sequential Machines, II," *IRE Trans. Electr. Comp.,* Vol. EC-10, No. 4, pp. 593-603, December 1961.

[53] K.E. Stoffers, "Sequential Algorithm for the Determination of Maximum Compatibles," *IEEE Trans. on Computers,* pp. 95-98, January 1974.

[54] J.R. Story, H.J. Harrison, and E.A. Reinhard, "Optimum State Assignment for Synchronous Sequential Circuits," *IEEE Trans. Comp.,* Vol. C-21, No. 12, p. 1365-1373, December 1972.

[55] S.H. Unger, "Flow Table Simplification - Some Useful Aids," *IEEE Transaction on Electron. Computers,* Vol. EC-14, pp. 472-475, 1965.

[56] W. Wan, M.A. Perkowski, "A New Approach to the Decomposition of Incompletely Specified Functions based on Graph-Coloring and Local Transformations and Its Application to FPGA Mapping", *Proc. of the IEEE EURO-DAC '92, European Design Automation Conference,* Sept. 7-10, Hamburg, 1992, pp. 230 - 235.

[57] Wan Wei, M.S. thesis, PSU Electrical Engineering Department, 1992.

[58] Wang Zhi, Jin Zhang, Workshop on Decomposition, PSU, recent report, 1995. unpublished.

[59] M. Yamamoto, "A Method for Minimizing Incompletely Specified Sequential Machines, " *IEEE Trans. Comp.,* Vol. C-29, No. 8, p. 732-736, August 1980.

[60] S. Yang and M. Ciesielski, "Optimum and Suboptimum Algorithms for Input Encoding and Its Relationship to Logic Minimization," *IEEE Trans. on Computer Aided Design,* Jan 1991.

[61] W. Zhao, "Two-Dimensional State Minimization with Input Encoding," M.S. Thesis, PSU.