AN EFFICIENT AND EFFECTIVE APPROACH TO COLUMN-BASED INPUT/OUTPUT ENCODING IN FUNCTIONAL DECOMPOSITION

Michael Burns, Marek A. Perkowski
Stanislaw Grygiel

Department of Electrical Engineering
Portland State University
Portland, OR 97207
USA

Lech Jozwiak

Faculty of Electronics Engineering
Technical University of Eindhoven
MB 5600 Eindhoven
The Netherlands

Abstract

Encoding in Curtis-style decompositions is the process of assigning codes to groups of compatible columns (or cubes) so that the binary logic descriptions of the predecessor and successor sub-functions can be created for further decomposition. In doing so, the sub-functions created are functionally equivalent to the set of care values specified in the original function. In this paper an input/output encoding algorithm **DC_ENC** *is presented that is designed to achieve the simpliest total complexity of the predecessor and successor sub-functions, and to increase the total number of don't cares for their further utilization in subsequent decomposition steps of these sub-functions.*

1. Introduction. The Encoding Problem in Functional Decomposition.

One of the most promising approaches of modern logic synthesis is general functional decomposition [17, 20, 11]. Together with our collaborators, we developed decomposers TRADE [27], LUTSYN [26] and GUD (all binary), FRED [11, 12] (multi-valued functions), and finally GUD-MV lued) for **multivalued functions and relations** [20, 13, 14]. Observe, that unlikely to TRADE that uses cubes and FRED and most recent decomposers that use BDDs (Pedram et al, Scholl [23], Wurth et al), **GUD and GUD-MV use a new data structure, based on BDD-encoded or bit-set-encoded labeled rough partitions** [13, 14] (Karnaugh maps are used below only for easier explanation). This new data structure allows to realize algorithms tuned to multi-valued relations and don't cares, which would be difficult to implement in BDD-based approaches.

Don't cares have many interesting applications in decomposition, including symmetries and regular structures [21, 9, 11, 12] but in this paper we will **focus on just one aspect** of creating an effective and efficient decomposer: the **Binary Encoding Problem**. We focus our applications on regular VLSI layout for sub-micron techologies and Machine Learning, therefore we **do not** assume the same size of all bound sets. Although our approach accounts for encoding to vectors of arbitrary values v (e.g. as required by the decomposer from [12]), for simplicity it is presented here for $v = 2$ only.

The most important problem of the functional decomposition is finding the sets of input variables for sub-functions that result in high-quality *hierarchical decompositions* (sub-functions are recursively decomposed to next-level sub-functions, $F(A, B) = H(A, G(B))$, where $G_i(B)$ are single-output functions from the predecessor block G, $|G(B)| < |B|$ and B is the bound set). For solving this problem the column incompatibility or compatibility graph is used in the Curtis-style functional decomposition [27, 17, 20]. Each group of compatible columns found based on such graph is then encoded with the same code. Various encodings of function G lead to various realizations of functions G and H, each with various numbers and locations of don't cares. The encoding should reduce some quality measures of both the predecessor and the successor sub-functions, making use of these don't cares. Some **special classes** of functions G_i (symmetric, monotonic, etc.) can be used for encoding [9, 21, 11, 12], but here our interest is on **general functions** used in G. Observe, that the problem is difficult. Firstly, our final goal is not a two-level decomposition, but a multi-level decomposition. Secondly, unlike in the FSM state encoding, the encoding sub-process in functional decomposition is repeated very many times in the entire hierarchical decomposition process. It must be very fast, but also produce high quality results.

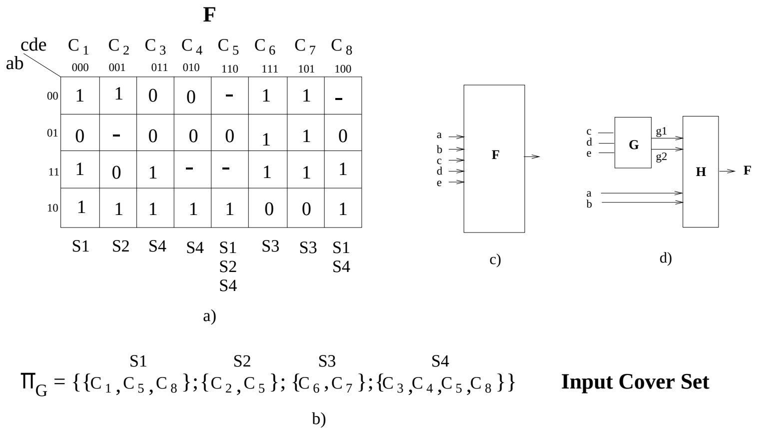

Fig. 1. Function used to Illustrate Encoding of Disjoint vs. Nondisjoint Cover Sets

The "Encoding Problem for Decomposition" can be simplified as the "Input Encoding Problem" or the "Output Encoding Problem". *Input Encoding* is the process of selecting codes for classes of the *cover set* Π_G [17] so that the complexity of the successor sub-function is minimized. The cover set Π_G is a partition of care minterms such that $\Pi_G \cdot \Pi(A) \leq \Pi_F$, and $\Pi_G \geq \Pi(B)$, in decomposition $F = H(A,\ G(B))$, where A is the free set and B is the bound set of input variables. $\Pi(B)$ is the partition of minterms induced by a bound set, and Π_F is the output partition of the function on the set of its care minterms. Examples of some input encoding approaches are found in Wan [27], Murgai [19], and Brayton [4]. *Output Encoding* is the process of selecting codes for classes of the cover set Π_G so that the complexity of the predecessor sub-function is minimized. Example of an output encoding approach can be found in Saldanha [22]. Encoding approaches referred to as combined input/output encoding aim at concurrent simplification of the predecessor and successor sub-functions. Examples of the combined input/output encoding can be found in Devadas [7], Saldanha [22], Volf [26], and Selvaraj [24]. The input/output encoding approaches are typically more difficult to design but have a potential to produce the predecessor and successor functions with much smaller combined complexity.

In Curtis decompositions, the primary goal is typically to minimize, in a number of steps, the total complexity of the hierarchical multi-level realization of a given function. This complexity can be expressed as a *DFC* or Decomposed Function Cardinality, which is the total cost of blocks, where the cost of a (binary) block with n inputs and m outputs is $2^n * m$, [20] (we use DFC because our research is mostly on decompositions for VLSI layout generation and Machine Learning; most authors are interested in FPGAs and use the total number of Look-Up table blocks as the cost function; for instance, XC3000 CLB of Xilinx has a DFC cost of $2^5 = 32$). Thus, one of the main goals of our implementation of generalized Curtis decomposition is to preserve or even increase don't cares for the successive decomposition steps [20].

An important criterion for determining what type of encoding is best for a Curtis decomposition, is the ratio of relative complexities of the predecessor and successor sub-functions (see Fig 1d for the schematic view of Curtis decomposition - G is the predecessor and H is the successor block (sub-function)). Unfortunately, it is difficult to assess the relative complexities of the sub-functions, especially, when they are nearly of the same size (in terms of the numbers of inputs and outputs). However, when the predecessor sub-function is much larger than the successor sub-function, one can reason that the predecessor will have a greater potential for further simplifications. For example, if the predecessor sub-function has 20 inputs and 3 outputs and the successor sub-function has 4 inputs and 3 outputs, then it would be obvious that the predecessor would have a greater potential for further simplification in terms of DFC (i.e. DFC=$(2^{20} * 3)$ vs. DFC=$(2^4 * 3)$) . Conversely, if the successor is much larger than the predecessor, then there is a greater potential for further simplification of the successor sub-function. Therefore, the input encoding approach should be used in this case. The output encoding approach should be applied when the predecessor is much larger than the successor. Finally, when they are roughly of equal size, then the combined input/output encoding should be used. An interesting characteristic of the new **DC_ENC** approach presented in this paper is that it will behave at times like each of the three types of encoders based on a heuristic cost function. Also, the more don't cares - the better the approach works.

In different applications of decomposition, different objectives have to be optimized. However, only three primary encoding objectives are used in the **DC_ENC** encoding approach (**ENC-oding to increase Don't Care sets**) presented in this paper. These three primary encoding objectives are used to achieve the overall desired goal of obtaining a multi-level decomposition which has the minimum total complexity of all logic blocks (or DFC). The *first objective* is to **minimize the Hamming distances between the columns in the Karnaugh map of the successor sub-function for a given bound set B and free set A.** The *second objective* is to **minimize the Hamming distances between the codes assigned to adjacent cells in the Karnaugh map of the predecessor sub-function**. The *third objective* is to **maximize the number of don't cares produced in the predecessor sub-function**. The order of importance of these objectives varies depending on the sizes of the sub-functions. If the successor sub-function is much larger than the predecessor sub-function, then the first objective is more important. If the reverse is true, then the last two objectives are more important. If the predecessor sub-function is much larger than the successor sub-function, then optimizing a combination of the last two objectives will most likely lead to an overall decomposition resulting in a lower DFC. Each of the three objectives mentioned results in a **separate set of encoding constraints.**

The encoding problem that is solved by the **DC_ENC** encoding method is formulated as follows. Given a set of various encoding constraints, use the minimum number of binary variables (bits) to encode all columns in a Karnaugh map, such that at least 75% of the encoding constraints are satisfied. The optimal percentage and types of constraints to satisfy to achieve the overall goal of minimum DFC diff of course from function to function. The goal is to dynamically find the exact or near exact percentage of constraints to satisfy on a function-by-function basis. The value of 75% was selected here for illustration, based on the following rationale. The size of a constraint is the number of symbols in the constraint. Each symbol is assigned a unique code. If the number of bits that differ in the symbol codes is minimal, then more bits are required to assign codes to symbols in large constraints than in small constraints. In each of several example decompositions the number of large constraints was very small in comparison to the number of small constraints. Therefore, 75% was chosen as a heuristic cut-off value.

One of the most important characteristics of the **DC_ENC** algorithm is its ability **to take advantage of the overlap in compatible classes of columns** to "produce" don't cares in the predecessor sub-function. We found on circuit realizations that in many decomposed benchmark functions these don't cares can greatly simplify the complexity of the predecessor sub-function. The **DC_ENC** algorithm involves: **(1)** selection of suitable cover sets, **(2)** heuristics to optimize the quality of the encoding at low computational cost, **(3)** multiple constraint satisfaction using an edge-weighted connection graph, and **(4)** use of Hamming distances to aid in the code assignments which result in simpler functions. Below, we will explain the main ideas of our approach, they can be implemented in various ways in a program. Our program, **DC_ENC** is only one realization of these ideas, and there is no space here to go to technical details. Understanding of state assignment and encoding theories [1, 6, 8, 10, 16, 19, 22] as well as the decomposition theory [11, 12, 13, 14, 20, 25] is very useful to understand our implementation details [3], but we hope that this presentation is sufficiently self-contained to explain the main ideas of our work, even to people who do not work in encoding and decomposition.

2. Definitions, Notations, and Terminology for Decomposition Encoding

Unless otherwise stated, a set which contains column indexes will be considered to be the same as a set which contains columns(i.e., column will be short for column index). Similarly, for sets containing cubes, symbols, classes, etc., the word "index" will be not included. The purpose for this is to avoid the over-use of the word "index". However, where it will be necessary to distinguish between the element's index and the contents of the element, then appropriate clarification will be made. A **cover set** is a set of subsets of columns that cover all columns. A **disjoint cover set** (a partition) is a cover set in which no column is an element of more than one subset. A **nondisjoint cover set** is a cover set which contains some subsets of columns where at least one column is an element of more than one subset. **Hamming Distance** between two code words(or vectors of variables) is defined as the number of positions (variables) in which these code words differ. A **Symbol** is a (multi-value) label representing a set of mutually compatible elements; the elements in each set are either cubes or columns. Symbols are used to denote the individual sets(or classes) within a given cover set. A set of columns corresponding to a particular symbol may be referred to as a **symbol set** or **symbol class** or **symbol group**. For simplicity, a symbol set within the cover set is simply referred to as a symbol. A **Hypercube** of dimension n is an n-dimensional binary sub-space, i.e. a set of 2^n vertices, where each vertex has exactly n edges with n other vertices. No vertex in a hypercube is connected to the same set of edges as any other vertex in the hypercube. A **Supercube** of a certain set of cubes is defined as the smallest cube containing all the 0-dimensional cubes (minterms) contained in this set of cubes.

Example 1: The supercube of cubes 000, 001, and 011 is 0 - -.

A **k-cube** is a supercube of 2^k bit vectors where the number k indicates the number of don't cares in the cube.

Example 2: For the bit vectors 00, 01, 11, and 10, the resulting k-cube (2-cube) is "- -".

A **Face** is a k-dimensional sub-space of the n-dimensional binary space, where $k \leq n$. Typically, a face refers to k-dimensional sub-cube of an n-dimensional cube. A **Column Constraint** (as defined for the purpose of this

paper) is a set(or group) of symbols in the cover set to which a particular column is compatible.
Example 3: The form of a constraint for a certain column C_i is (S_1, S_3, S_7) - see Fig. 1 for an example.

A **Face Embedding Constraint** is a constraint which specifies that a certain set of symbols has to be assigned to one face of a binary n-dimensional cube, without any other symbol sharing the same face. A face embedding constraint is said to be satisfied if all the codes assigned to the symbols in the constraint occupy a single face in an n-dimensional cube and no other symbols are placed in this face. When a face embedding constraint is satisfied, it is possible that some codes of the face remain unused.
Example 4: For a three dimensional cube (- - -) the face embedding constraint for column C_i is (S_1, S_3, S_4). Symbol S_1 encoded with 000, S_3 with 001, S_4 with 011 and no symbol is assigned to 010 (unused code), with the set of symbols in the constraint of column C_i is said to satisfy the face embedding constraint, because the codes assigned to each of the symbols are contained in a single face(i.e., 0 - -) of the given three dimensional cube, and no other symbols are placed in this face.

A **Hypercube Embedding Constraint** is a special face embedding constraint containing exactly 2^k symbols. Like a face embedding constraint, a hypercube embedding constraint is said to be satisfied if all the codes assigned to the symbols in the constraint occupy a single face in an n-dimensional cube and no other symbols are placed in this face but all codes of the face are used.
Example 5: For the hypercube embedding constraint (S_1, S_4, S_3, S_7) for column C_i and the code assignments: $000 = S_1$, $001 = S_3$, $011 = S_4$, $010 = S_7$, thus (0 - -) is the supercube of the codes in the constraint. Remaining symbols, not included in the constraint, are the following: $110 = S_2$, $111 = S_5$, $101 = S_6$. The hypercube embedding constraint is satisfied because the supercube contains only codes contained in the constraint.
Example 6: Given the hypercube embedding constraint (S_1, S_4, S_3, S_7) for column, and the following code assignments: $000 = S_1$, $101 = S_3$, $011 = S_4$, $010 = S_7$, therefore (- - -) is the supercube of the codes in the constraint. Remaining symbols not in the constraint: $110 = S_2$, $111 = S_5$, $001 = S_6$. The hypercube embedding constraint is not satisfied, because the supercube of the codes assigned to the symbols of the constraint includes the codes of symbols which are not contained in the constraint.

The Function Cost Ratio (FCR) is an approximate measure of the relative sizes of the predecessor and successor functions in a Curtis-style decomposition. It is defined as follows: $\mathbf{FCR} = \mathbf{Cost1} = \mathbf{G_{DFC}}/\mathbf{H_{DFC}}$. $\mathbf{G_{DFC}}$ is the DFC of the predecessor sub-function G. $\mathbf{H_{DFC}}$ is the DFC of the successor sub-function H.
Overlap Ratio $\mathbf{R_O}$ is a measure of the "overlap" of columns in symbols(or classes) of Π_G. Columns are said to "overlap" if they are compatible with more than one symbol in the cover set Π_G. R_O is defined as follows: $\mathbf{R_O} = \mathbf{C_O}/\mathbf{C_T}$. $\mathbf{C_O}$ = Number of columns which are compatible with more than one symbol in Π_G. $\mathbf{C_T}$ = Total number of columns(excluding columns of all don't cares).
Example 7. For function from Fig. 1: $C_O = 2$, $C_T = 8$, $R_O = 2/8 = 1/4$.

3. Fundamentals of the Column-Based Encoding

In the Curtis style decomposition, the encoding process follows the column minimization phase [20, 3]. The column minimization process consists in covering all columns with the (quasi) minimum number of cliques of columns. The set of such subsets of columns will be called cover set Π_G. The primary input data to the encoding process are: the function to be decomposed, the cover set Π_G, and the column multiplicity (the cardinality of Π_G). In the column-based encoding, the cover set Π_G is composed of subsets (cliques CCs) containing columns. Though encoding approaches can be diverse and quite complicated, most column-based encoding approaches share the two following primary steps: the assignment of codes to the symbols(or classes) in the cover set Π_G, and the assignment of codes to each column corresponding to the symbol codes of the symbols to which a certain column is compatible.

In Figure 1, the Karnaugh map is shown of an example function and an input cover set is used to illustrate the column-based encoding of disjoint and nondisjoint cover sets. Below each column of the Karnaugh map for function F a list of symbols is given to which a certain column is compatible. The question arises, what is the advantage of the encoding with nondisjoint cover sets over the encoding with disjoint cover sets? The primary advantage is that there are additional optional codes which may be assigned to columns. If there are more codes to choose by the nondisjoint cover sets, then why ever use the disjoint cover sets? The answer is that many encoding approaches (like those in which graph coloring or clique partitioning are used for column minimization [20]) assume that the input cover set is disjoint (a partition), and therefore, they cannot handle the overlap in nondisjoint cover sets. In Section A an example is presented of the column-based encoding for the disjoint cover sets. In Section B an example is presented of the column-based encoding for the nondisjoint cover sets.

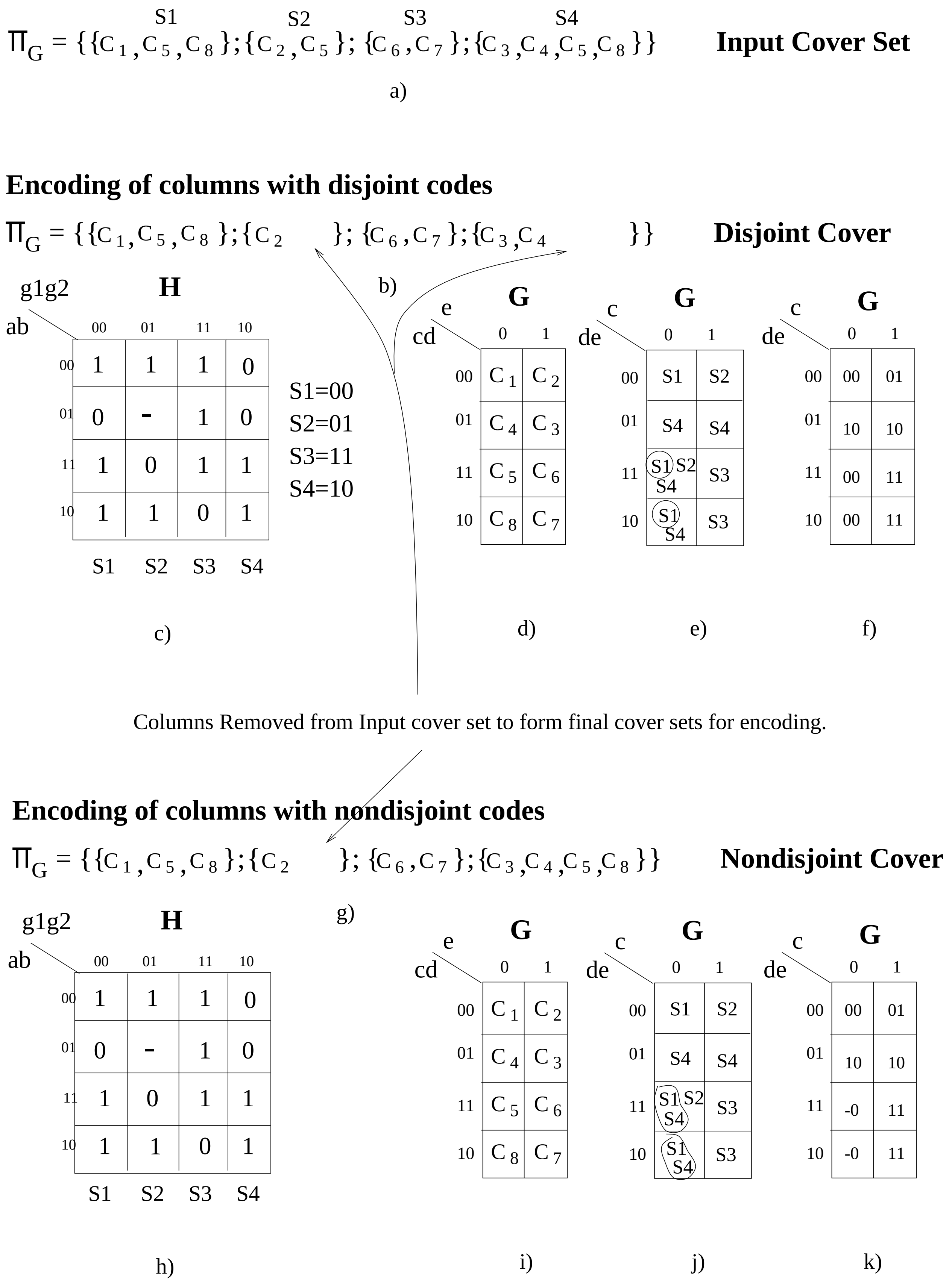

Fig. 2. *Encoding of Disjoint vs. Nondisjoint Cover Sets*

A. Encoding of the Disjoint Cover Sets

To create a disjoint cover set it is necessary to remove all instances of each column from every subset(symbol) of Π_G except for one of them. In general, many various cover sets may result from removing columns from the symbols of Π_G to make it a disjoint cover set. For the input cover set shown in Figure 2a, there are only two columns that are elements of more than one symbol. Column C_5 can be assigned to one of the three symbols of Π_G. Column C_8 can be assigned to one of the two symbols of Π_G. In total, there are six possible disjoint cover sets (i.e., there are six disjoint cover sets which can be produced by removing columns C_5 and C_8 from all but one symbol). For simplicity, columns C_5 and C_8 are arbitrarily removed from all symbols except for one. In Figure 2b the disjoint cover set is shown that is produced by removing columns C_5 and C_8 from all but one symbol of the input cover set.

The cover set selected determines what column types will be in the sub-function H. Each of the column types in the sub-function H is obtained by performing the union on all columns contained in each of the symbols. In Figure 2c, different column types are shown in the sub-function H corresponding to the disjoint cover set selected. The actual position of the column types in the Karnaugh map of sub-function H are not known until the codes($g1g2$) have been assigned to each of the symbols.

In Figure 2d the cells of the sub-function G are shown with column labels corresponding to each vector of input variables of the bound set of function F. In Figure 2e the cells of the sub-function G are shown containing the symbols in Π_G to which a certain column in function F is compatible. Observe that by making the input cover set disjoint, column C_5 may only be assigned to symbol $S1$(shown circled) because it is no longer an element of the subsets corresponding to symbols $S2$ and $S4$. Similarly, column C_8 may only be assigned to symbol $S1$ because it is no longer an element of the subset corresponding to symbol $S4$.

Finally, codes for each symbol have to be assigned. Because the column multiplicity is four, two bits are required to encode each of the four symbols in Π_G. Ideally, the codes should be assigned to simplify both sub-functions simultaneously. However, because the purpose of this section is merely to show the basics of the column-based encoding, the codes for each symbol are chosen arbitrarily. The codes assigned to the symbols are shown in Figure 2c as values of the variables $g1g2$ at the top of the corresponding columns.

The assignment of codes to columns in function F is performed by assigning the code of each symbol to every column that is an element of that symbol. In the cells of the Karnaugh map in Figure 2f the codes for each column are shown. This completes the encoding process for the disjoint cover sets. In Figure 1c and Figure 1d the block diagrams of the original function F and the sub-functions G and H of its Curtis decomposition are presented. Note, that the codes $g1g2$ assigned to columns correspond to the outputs of the sub-functions G and inputs to the sub-functions H.

B. Encoding of the Nondisjoint Cover Sets

The primary difference between the encoding of the nondisjoint vs. disjoint cover sets is that the columns may be now assigned with the combined symbol codes(supercube of codes) in some instances. For example, if the codes assigned to the symbols are the same as they were in the disjoint case (i.e., $S1$=00, $S2$=01, $S3$=11, and $S4$=10), then column C_5 could be assigned codes 0- or -0, as well as 00, 01, or 10.

A column can only be assigned a combined symbol code if the supercube of the symbol codes does not contain any codes of symbols to which the column is not compatible. For example, column C_5 can't use the optional code 0- unless it is an element of the two symbols with symbol codes 00 and 01. Similarly, column C_5 can't use the optional code -0 unless it is an element of the two symbols with symbol codes 00 and 10. Also, we can't assign C_5 with the combined symbol codes of symbols $S2 = 01$ and $S4 = 10$ because they differ by more than one bit. The combined code of symbols $S2$ and $S4$ results in code "- -". This code also includes the code of symbols $S1 = 00$ and $S3 = 11$, and symbol $S3$ is not among the acceptable code assignments for column C_5.

In Figure 2g a nondisjoint cover set is shown which allows columns C_5 and C_8 to receive don't cares in their codes. In Figure 2j combinations of symbols are shown (circled) to which columns C_5 and C_8 were assigned. Assigning columns C_5 and C_8 to the combined codes of symbols $S1$ (i.e., 00) and $S4$ (i.e., 10) results in the combined code -0. The resulting column codes are shown in Figure 2k. Why not give column C_5 the combined code of symbols $S1$=00, $S2$=01, and $S4$=10? The combined code of symbols $S1$, $S2$, and $S4$ is "- -". This code includes the code of symbol $S3$ which is not among the acceptable code assignments for column C_5. For this reason, column C_5 was removed from symbol $S2$ in the initial cover to form the final cover set shown in Figure 2g. This completes the encoding process for the example of the nondisjoint cover sets. The block diagram for the decomposed sub-functions G and H is the same as for the disjoint case. Using the same function, we explained how columns can be encoded using the disjoint and nondisjoint cover sets. We also explained how don't cares can be introduced into the codes of the predecessor sub-function when the encoding method utilizes the overlap in nondisjoint cover sets. Often, these extra don't cares can greatly reduce the complexity of the predecessor sub-function. **The more don't cares, the higher the overlap, the more choices of subsets of symbols, the better the codes and the more don't cares in G and H.**

4. General Outline of Our New Encoding Approach

Though there are many different general selection strategies possible, the strategy described below is simple and effective. Therefore, we adapted it as the **primary strategy** for use in the **DC_ENC** program (there are other strategies [12, 27, 24]). The main purpose of the selection strategy is to evaluate whether some conditions are satisfied for the **DC_ENC** to be effective. If so, the **DC_ENC** encoding program would be called. Otherwise, a different encoding program would be selected.

The selection criterion is based on the evaluation of the Function Cost Ratio (FCR) and the overlap ratio (R_O). The following IF-THEN-ELSE statement is the proposed heuristic, which determines if the **DC_ENC** encoding is used.

If ($R_O > 1/5$ *and* $FCR \geq 1/2$)
then use the **DC_ENC** *encoding approach,*
else
use an alternative encoding approaches.

The above IF-THEN-ELSE statement will be referred to as *Rule#1*. Basically, *Rule#1* states two conditions to be satisfied before the **DC_ENC** encoding approach is used. The first condition specifies that there must be a sufficient overlap of columns in the symbols of Π_G. Why it is so? If there is no overlap, then the **DC_ENC** encoding approach can't utilize overlap to produce don't cares in the codewords. The second condition specifies that the predecessor sub-function must be roughly equal to or larger than the successor sub-function in terms of inputs and outputs(DFC). Why ? Because the **DC_ENC** tends to simplify the predecessor sub-function more than the successor sub-function. For some functions, **DC_ENC** may actually simplify the predecessor sub-function at the cost of making a more complex successor sub-function(i.e., **DC_ENC** produces don't cares in the G block in such a way that Hamming distances may increase in block H). When the two conditions above are satisfied, then the **DC_ENC** program can greatly simplify the complexity of the predecessor block by introducing don't cares in the codes assigned to columns.

Unfortunately, due to the differences in each decomposition, it is not known exactly when the overlap is sufficient for the **DC_ENC** to be effective. However, based on the experience acquired solving numerous examples, our current recommendation is to use a different encoder if $R_O < 1/5$. Future work may include more benchmarks testing to refine this value, and/or to find other criteria to determine if and when the **DC_ENC** is effective.

The function cost ratio (FCR) provides quite rough estimate of the ratio of complexity and/or size of sub-function G relative to sub-function H. Ranges of the FCR ratio are used to make decisions in the general encoding strategy introduced by us. The following set of ranges for the FCR ratio is considered: **Range 1:** $FCR \leq 1/6$, **Range 2:** $1/6 < FCR \leq 1/2$, **Range 3:** $1/2 < FCR \leq 2$, **Range 4:** $2 < FCR \leq 6$, **Range 5:** $FCR > 6$. These ranges are used for two purposes. The first purpose is to determine whether or not one of the two conditions necessary for using the **DC_ENC** is satisfied (IF-THEN-ELSE statement above). The second purpose is to determine values to assign to two parameters, X and U, used in the cover set selection process. The values assigned to X and U are used heuristically to determine how a cover set should be selected in order to provide more efficient encoding for the larger of the two sub-functions in a decomposition. FCR is used to determine values for these parameters and the parameters affect the cover set selection process. Once the **DC_ENC** algorithm has been selected, the cover set selection is performed. The cover set selection is a heuristic process of forming an enhanced cover set according to the value of the cost function ratio FCR.

The next step is the encoding, It belongs to the family of hypercube embedding algorithms (One of the known methods for the hypercube embedding is implemented in $MUSTANG$[8]; later, an improved approach named $JEDI$ was introduced[16]; yet another approach, $MUSE$[10], produced slightly better results than either $MUSTANG$ or $JEDI$). However, our approach differs significantly from them. While the known approaches create an **arbitrary** graph taking into account all constraints, and next embedd (heuristically and non-optimally) this graph to a hypercube, we **construct** the Edge Weighted Connection Graph ($EWCG$) with special properties. The construction process takes possibly only a **subset** of constraints, but guarantees that EWCG is next algorithmically and quickly embeddedable in a hypercube such that all constraints used to create EWCG are satisfied. (This is a general approach to encoding problems that may find also other applications).

Thus, the stages of our algorithm are: **(1)** evaluation of FCR and Overlap Ratio; **(2)** selection of encoding algorithm; **(3)** cover set selection; **(4)** incremental creation of EWCG; **(5)** resolve conflicts between objectives; **(6)** if (constraint satisfaction and objectives completed) go to (7), else if more bits required add one bit to Code Word size and go to (3), else to to (4); **(7)** Embedd EWCG to a hypercube; **(8)** Assign Supercube of Symbol Codes to corresponding Columns.

		cost								
File	i/o	TRADE	MISII	Stei	SCH	LUB	JOZs	JOZh	PSU	[time]
5xp1	7/10	496	384	292	288(9)	288(9)	320(20)	336(21)	236	[11.0]
9sym	9/1	640	984	400	224(7)	160(5)			104	[26.4]
con1	7/2	80	68	60					70	[2.3]
duke2	22/29	6516	2428	2200	3456(108)				2896	[11289.0]
ex5p	8/63		3720	1560					2104	[208.0]
f51m	8/8	372	392	240	256(8)				177	[10.1]
misex1	8/7	472	208	224	256(8)	352(11)	304(19)	288(18)	229	[8.6]
misex2	25/18	548	464	436	768(24)				392	[1086.0]
misex3	14/14	9816	4204	3028					1744	[1316.0]
rd53	5/3	120	96	84					60	[1.8]
rd73	7/3	320	352	256	160(5)	160(5)			113	[13.1]
rd84	8/4	508	672	320	256(8)	224(7)			171	[32.6]
sao2	10/4	1848	516	468	576(18)	576(18)			441	[47.2]
root	8/5					512(16)	736(46)	752(47)	490	[90.0]
alu2	10/3				1056(33)	1632(51)	1856(116)	1824(114)		
alu4	14/8								3455	[1326]
clip	10/3				416(13)	512(16)	1856(116)	1824(114)		
clip	9/5								467	[53]

Table 1
Comparison of Decomposers on selected binary benchmarks

The $EWCG$ is constructed in such a way as to obtain a set of weighted constraints which will be used to maximize the number of don't cares produced in the G sub-function, minimize the Hamming distances between cofactors in the H sub-function, and minimize the Hamming distances between the codes assigned to the cells of the Karnaugh map in the G sub-function. When conflicts occur by trying to satisfy certain sets of constraints concurrently, then some evaluations are made using certain cost function to determine what action should be taken to resolve the conflicts. If 75% of all constraints can not be satisfied together in the $EWCG$, then an additional code bit is added and the process of constructing the $EWCG$ is repeated.

Once the $EWCG$ is constructed by satisfaction of at least 75% of constraints then the $EWCG$ is embedded on a hypercube of dimension n, where n is encoding length. Finally, the assignment of the supercubes of the symbol codes to the columns in the input function is performed. Basically, each column is assigned to the largest combination of 2^k symbol codes to which the column is compatible. For example, if column $C1$ is compatible with codes 00, 01, and 11, then it could be assigned either 00, 01, 11, 0-, or -1. The algorithm would select either column code 0- or -1. However, it could not be assigned the code "- -" because that code includes a symbol code to which $C1$ is not compatible (i.e. code 10).

Table 1 shows the result of comparison of PSU's and other decomposers on some benchmarks (recall that in contrast to others, we do not have a fixed number of inputs to a block). Observe that DFC allows to compare (of course only approximately) decomposers that decompose to various types and sizes of blocks. In Table, **PSU** is the Portland State University Decomposer of multi-valued relations [13, 14, 20]. **Stei** is the binary decomposers from Freiberg (Steinbach). All the functions in the table are binary and are taken from the set of MCNC benchmarks. **TRADE** is a decomposer developed at Portland State University; **MISII** at University of California, Berkeley; **SCH** is the Mulop-dc decomposer from Freiburg (Scholl) [23]; **JOZ** is from Technical Univ. of Eindhoven (Jozwiak) [15] (**JOZs** is systematic and **JOZh** heuristic strategy); and **LUB** is the Demain program from Warsaw/Monash (Luba and Selvaraj) [25]. The final cost value is computed as a sum of the costs of DFCs of single blocks of the result of the decomposition. For our program there is also execution time given (DECstation 5000/240, 64 MB of memory, user time in seconds) to show that the decomposition task can be performed in a reasonable amount of time. Numbers in parentheses are numbers of 5-input CLBs (Jozwiak reports only 4-input CLBs). The underlined results are the best DFC values for a given benchmark. Our version of *clip* is different (9,5), and we cannot find some of the benchmarks used by other authors.

This table, as well as other result, show that further work is needed to create a decomposer that would always find the minimum DFC solution known.

5. Conclusions

Unlike some other encoding approaches which optimize exclusively the input encoding or only the output encoding, the approach presented in this paper allows to satisfy multiple constraints and objectives for input and output encodings concurrently. The method of cover set selection which is used to enhance our encoding approach aims to increase the number of don't cares in the predecessor sub-function G while minimizing the loss of don't cares in the successor sub-function H. This is achieved by selecting for the cover set the compatible classes with large overlap. This encoding approach has the following properties that make it particularly suitable for decomposition approach in Knowledge Discovery and Data Minining applications: **[1]** it minimizes DFC and creates many don't cares, **[2]** it is applicable to relations (not only functions, as it was discussed here), **[3]** it can be easily extended from only binary encodings, to the encodings with multi-valued variables or signals [11, 12].

It can significantly reduce the resulting complexity of the predecessor logic, when $|B|/|A| \geq 1$ and functions are highly unspecified. The condition $|B|/|A| \geq 1$ is by no means an exact cut-off value. It only indicates when the potential for utilizing the overlap of classes is large. The potential for the overlap of classes in the cover set increases as the ratio $|B|/|A|$ increases. It also increases with the number of columns which are compatible with multiple classes in the cover set. Moreover, if the functions are not at least partially unspecified, then no overlap is present in the cover set. No overlap in the cover set means no potential for don't cares in the codes of columns without increasing the number of the encoding bits. For almost completely specified functions or when $|B|/|A| << 1$ the presented encoding approach may result in little or no benefit over some other encoding approaches. **However, it should be stressed that any function (even a completely specified function) can be transformed into a highly unspecified function by performing the *nondisjoint decomposition*, or by *decomposing into relations* [20, 11, 12, 25]. Therefore, the primary criterion for determining whether there are appropriate conditions for application of this encoding approach is:** $|B|/|A| \geq 1$. Benchmark examples of the **DC_ENC** approach illustrate how a significant number of don't cares can be introduced into the sub-function G as a result of the column encoding with codes of multiple symbols. Multiple constraint satisfaction can be accomplished by application of a relatively simple set of heuristics to the edge-weighted connection graph.

References

[1] P. Ashar, S. Devadas, and A. Newton, "Sequential Logic Synthesis," Kluwer Academic Publishers. 1992.

[2] K.A. Bartlett, R.K. Brayton, G.D. Hachtel, R.M. Jacoby, C.R. Morrison, R.L. Rudell, A.L. Sangiovanni-Vincentelli, and A.R. Wang, "Multilevel Logic Minimization Using Don't Cares," *IEEE Tr. CAD,* Vol. 7, pp. 723-740, June 1988.

[3] M. Burns, "Encoding for Functional Decomposition," *ECE PSU Report,* 1998.

[4] R. K. Brayton, G. D. Hachtel, and A. L. Sangiovanni-Vincentelli, "Multilevel Logic Synthesis," *Proc. IEEE,* Vol. 78, No. 2, pp. 264-300, February 1990.

[5] M. J. Ciesielski, S. Yang, and M. Perkowski, "Multiple-Valued Minimization Based on Graph Coloring," *Proc. ICCD'89,* pp. 262 - 265, October 1989.

[6] M. Ciesielski, and J. Shen, "A Unified Approach to Input-Output Encoding for FSM State Assignment," *Proc. DAC,* pp. 176-181, 1991.

[7] S. Devadas, A.R. Wang, A.R. Newton, and A. Sangiovanni-Vincentelli, "Boolean Decomposition in Multi-Level Logic Optimization," *Proc. ICCAD,* pp. 290-293, 1988.

[8] S. Devadas, H-K. T. Ma, A.R. Newton, and A. Sangiovanni-Vincentelli, "MUSTANG: State assignment of finite state machines targeting multi-level logic implementations," *Proc. ICCAD,* Vol. 7, pp. 1290-1300, December 1988.

[9] B.T. Drucker, C.M. Files, M.A. Perkowski and M. Chrzanowska-Jeske, "Polarized Pseudo-Kronecker Symmetry and Synthesis of 2x2 Lattice Diagrams," *Proc. ICCIMA'98,* pp. 745 - 755.

[10] X. Du, G. Hatchtel, B.Lin, and A.R. Newton, "MUSE: A MUltilevel Symbolic Encoding Algorithm for State assignment," *IEEE Tr. CAD,* Vol. 10, pp. 28-38, 1991.

[11] C. Files, R. Drechsler, and M. Perkowski, "Functional Decomposition of MVL Functions using Multi-Valued Decision Diagrams," *Proc. ISMVL'97,* pp. 27 - 32.

[12] C. Files, and M. Perkowski, "An Error Reducing Approach to Machine Learning using Multi-Valued Functional Decomposition," *Proc. ISMVL,* May 1998.

[13] S. Grygiel, M. Perkowski, M. Marek-Sadowska, T. Luba, and L. Jozwiak, "Cube Diagram Bundles: A New Representation of Strongly Unspecified Multiple-Valued Functions and Relations," *Proc. ISMVL'97,* Canada.

[14] S. Grygiel, and M. Perkowski, "New Efficient Representation of Multi-Valued Functions, Relations and Non-Deterministic Finite State Machines," *Proc. ICCD'98.*

[15] Jozwiak, L, "Efficient Logic Synthesis for FPGAs with Functional Decomposition Based on Information Relationship Measures," *Proc. Euro-Micro'98.*

[16] B. Lin, A.R. Newton. "Synthesis of Multiple-Level Logic From Symbolic High-Level Description Languages," *IFIP VLSI,* pp. 187-196, August 1989.

[17] T. Luba, M. Mochocki, and J. Rybnik, "Decomposition of Information Systems Using Decision Tables," *Bull. Polish Acad. Sci., Techn. Sci.,* Vol. 41, No.3, 1993.

[18] R. Murgai, N. Shenoy, R.K. Brayton, and A. Sangiovanni-Vincentelli, "Improved Logic Synthesis Algorithm for Table Look Up Architectures," *Proc. ICCAD,* pp. 564-567, 1991.

[19] R. Murgai, R. Brayton, and A. Sangiovanni-Vincentelli, "Optimum Functional Decomposition Using Encoding," *DAC,* p. 408-414, 1994.

[20] M. Perkowski, M. Marek-Sadowska, L. Jozwiak, T. Luba, S. Grygiel, M. Nowicka, R. Malvi, Z. Wang, and J.S. Zhang, "Decomposition of Multiple-Valued Relations," *Proc. ISMVL'97,* pp. 13-18.

[21] M. Perkowski, M. Chrzanowska-Jeske, and Y. Xu, "Multi-level Programmable Arrays for Sub-Micron Technology Based on Symmetries," *Proc. ICCIMA'98,* pp. 707 - 720.

[22] A. Saldanha, T. Villa, R. Brayton, and A. Vincentelli, "A Framework for Satisfying Input and Output Encoding Constraints," *Proc. DAC,* p. 170-175, 1991.

[23] Ch. Scholl, "Multi-output Functional Decomposition with Exploitation of Don't Cares," *Proc. DATE'98,* France.

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

[25] H.S. Selvaraj, M. Nowicka, T. Luba, "Non-Disjoint Decomposition Strategy in Decomposition-Based Algorithms and Tools," *Proc. ICCIMA'98,* pp. 34 - 42.

[26] F.A.M. Volf, "A Bottom-up Approach to Multiple-Level Logic Synthesis for Look-Up Table Based FPGAs," *Ph.D. Thesis,* Technical University Eindhoven, The Netherlands, 29 September 1997.

[27] W. Wan, and 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. EURO-DAC '92,* Sept. 7-10, Hamburg, 1992, pp. 230 - 235.