A New Approach to the Decomposition of Incompletely Specified Multi-Output Functions Based on Graph Coloring and Local Transformations and Its Application to FPGA Mapping

Wei Wan* and Marek A. Perkowski

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

Abstract

The paper presents a new approach to the decomposition of incompletely specified Boolean functions and its application to LUT-based FPGA mapping. Three methods were developed: the fast Graph Coloring to perform a quasi-optimum don't care assignment, the Variable Partitioning to quickly find the "best" partitions, and the Local Transformation to transform a nondecomposable function into several decomposable ones. The presented methods perform global optimization of the input function, while most of the other existing methods recursively perform only local optimizations on some kinds of network-like graphs and few of them can handle incompletely specified functions. A short description of a FPGA mapping program (TRADE) and an evaluation of its results are also provided.

1. Introduction

Decomposition is a general approach for simplifying large problems in logic synthesis applications. Decomposition involves breaking a large logic function, which is difficult to implement, into several smaller ones, which can be implemented with ease. In most existing logic synthesis systems, the don't care (DC) outputs are mistreated as 0's (or 1's) during the process of constructing the Truth-tables (or Boolean equations). This leads to results which are far away from the optimum ones. How to decompose an incompletely specified function and how to assign the DCs as 0 or 1 to simplify the result are the main topics of the paper. As a direct application, the decomposition method is applied to the FPGA mapping, especially to Xilinx's Lookup-Table (LUT) based FPGA architectures [1].

The LUT-based FPGA consists of a matrix of Configurable Logic Blocks (CLBs). A CLB can be programmed to function as one LUT with up to five inputs, or two LUTs with up to four inputs each and under the restriction that three out of these four inputs must be the same. A K-input (K = 5 for Xilinx architecture) LUT is a digital memory with K address lines and a one-bit output. This memory contains 2^K bits and is capable of implementing any Boolean function of up to K input variables.

* This research was supported in part by the NSF Grant MIP-9100772.

2. Current research on FPGA mapping versus our approach

Technology mapping is a process of transforming a technology independent Boolean network into a technology-based circuit. For LUT-based FPGAs, the technology-based circuit is a network of basic logic blocks which can implement any Boolean function of up to five input variables. The traditional library-based technology mapping techniques can not be used because the size of the library increases exponentially with the number of inputs of the components in the library. Several technology mapping approaches for LUT-based FPGAs have been reported.

MIS-PGA(new) [2] applies a variety of decomposition methods, such as *cube packing*, *Roth-Karp decomposition*, *AND-OR decomposition*, *Cofactoring decomposition*, *decomp-d* and *kernel extraction decomposition*, to decompose the input network into a feasible network. Then, it uses a *maxflow algorithm* to generate all possible *supernodes* and solves the *binate covering problem* to minimize the cardinality of the supernode set which covers the entire network. Finally, by solving the *maximum matching problem*, it merges all possible nodes into the FG mode CLBs. *Hydra* [3] uses an approach similar to *MIS-PGA*, but puts more attention on the FG mode CLB. *Chortle-crf* [4] divides the DAG into a *forest of trees*. Then, it carries out technology mapping on each tree to find the *minimum cost circuit*. Several techniques, such as *two_level decomposition*, *multi-level decomposition*, *exploiting reconvergent paths* and *replication of logic at fanout nodes*, are used in the program. *X-map* [5] converts Blif format into an *if-then-else* DAG, then, goes through a *marking* and a *reduction process* to minimize the network. *VISMAP* [6] partitions the input network into several subgraphs of reasonable size and goes through a *pre-processing* and a *main processing* step to determine the *invisible edges* to reduce each subgraph.

The FPGA mapping approaches mentioned above consist, in general, of four major steps: graph construction, decomposition, reduction and packing. In the *graph construction* step, a special kind of network-like graph or a set of subgraphs is created. The graph (or network) can be feasible or infeasible. *Feasible* means that the number of inputs of each node is limited. For Xilinx architecture, this input number is up to five. In the *decomposition* step, a variety of decomposition methods are applied to transform an infeasible network into a feasible one. During the process, the decomposition algorithms try to minimize the number of nodes in the decomposed network as well as the number of input variables per node. In the *Reduction* step, some covering algorithms are applied to try to find a set of minimum number of CLBs which covers the entire network. In the *Packing* step, according to a specific FPGA architecture, some algorithms are used to merge the possible nodes to further decrease the area. Most of the operations used in the four steps above are local operations. The dynamic programming algorithm allows the local operations to traverse across the network. The program is recursively invoked until a satisfactory result is reached.

We developed a FPGA mapping technique which applies global operations and has the following assets:

- The input data to the program is an incompletely specified Boolean function described by sets of ON and OFF cubes. It is the property of this method that the more DC cubes exist, the more efficient the method becomes.
- The decomposition methods are specifically adapted to the LUT-based FPGA architectures.
- A Variable Partitioning method is used to quickly find the "best" partitions, avoiding an exhaustive test of all possible decomposition charts.
- A fast Graph Coloring method is used to perform a quasi-optimum don't care assignment to minimize the column multiplicity.
- A Local Transformation method is used to make the decomposition possible for all kinds of Boolean functions.

3. Basic definitions

Decomposition means breaking a large logic block into several relatively smaller ones.

A *decomposition chart* [7] is similar to a Karnaugh map with the only difference being that the column and row indexes of a decomposition chart are in straight binary order, while those of a Karnaugh map are in Gray code order. Figure 2b shows an example of a decomposition chart. The corresponding Karnaugh map is shown in Figure 2a. Because there is no essential difference between a Karnaugh map and a decomposition chart, Karnaugh map is used instead of decomposition chart later in the paper.

The *bond set* is a set of variables forming the columns of the decomposition chart. The *free set* is a set of variables forming the rows of the decomposition chart. In Figure 2b, {c, d, e} is a bond set, {a, b} is a free set.

The *column multiplicity*, denoted by $\upsilon(B|A)$, is the number of different columns in a decomposition chart. In $\upsilon(B|A)$, B stands for the free set, A stands for the bond set. For example, in Figure 2b, $\upsilon(B|A) = \upsilon(ab|cde) = 3$.

If the horizontally corresponding cells of two columns in the decomposition chart are (0,0), (1,1), (0,x), (1,x), (x,0), (x,1) or (x,x), these two cells are *compatible*. (x stands for DC output.) If all corresponding cells in two columns are compatible, these two columns are *compatible*. Otherwise, they are *incompatible*. In Figure 2b columns 1 and 6 are compatible, while columns 5 and 6 are incompatible. The formula [8] to test the compatibility of two columns (columns i and j) is:

$$\begin{array}{ll} & \{ON(i) \bullet OFF(j)\} = \phi \\ \& & \{ON(j) \bullet OFF(i)\} = \phi \end{array} \quad => \quad \text{i-th and j-th column compatible}$$

ϕ stands for empty set. The formula states that if the Intersection of the ON set of column i and the OFF set of column j is empty, and the Intersection of the ON set of column j and the OFF set of column i is empty as well, these two columns are compatible. Otherwise, they are incompatible.

The *incompatibility graph* is a graph which illustrates the relationship among the columns of a decomposition chart. Each node in the graph represents a column in the decomposition chart. If two columns are incompatible, there is an edge between the corresponding nodes. If they are compatible, there is no edge. Figure 2d shows an incompatibility graph corresponding to the decomposition chart in Figure 2b. In Figure 2d, the number in each node (denoted by a circle) is the column number. The letter beside the circle is the color assigned to the node (column) after Graph Coloring.

4. How to perform decompositions?

In this section, the generalized Boolean decomposition of incompletely specified single and multi-output functions are presented. The basic ideas follow [7] [9] [10] and the general approach based on Graph Coloring is patterned after [8] [11].

4.1. Decomposition of the incompletely specified single output functions

Curtis had described the decomposition of completely specified functions in [9]. Curtis proved the *fundamental theorem*:

$$\upsilon(B\,|\,A) \le 2^k \iff f(A, B) = F[\phi_1(A), \phi_2(A), \cdots, \phi_k(A), B]$$

Which states that if the column multiplicity $\upsilon(B|A)$ is less than or equal to 2^k, then the function f can be decomposed into the F form as given above. The graph representation of this theorem is shown in Figure 1.

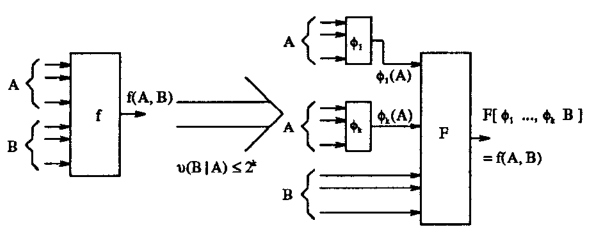

Figure 1. Curtis decomposition.

From Figure 1, we observe that, after decomposition, the big block f is broken into several smaller subblocks $\phi_1, \phi_2, ..., \phi_k$ and F. If we restrict the number of variables in the bond set A to be less than or equal to five, $\phi_1, \phi_2, ..., \phi_k$ could be implemented by CLBs of the Xilinx chip. If we further decompose subblock F until the input variables of each subblocks are less than or equal to five, the function f would be totally realized by Xilinx chip(s).

The generalization of the Ashenhurst decomposition for incompletely specified functions based on proper Graph Coloring was presented in [8]. Perkowski used Graph Coloring to minimize the column multiplicity and used multiplexers to realize the circuit.

The essential problem of the decomposition of incompletely specified function is how to assign DC outputs as 0 or 1 to minimize the column multiplicity. Because the number of colors in a properly colored incompatibility graph is same as the number of different columns (column multiplicity) in a decomposition chart [8], we can transfer the problem of finding the smallest column multiplicity into one of performing Graph Coloring to find the smallest number of colors. We use the following criterion:

Set an integer of value (n), which is the expected number of output variables from the bond set and is less than the variable number of the bond set. If the column multiplicity is equal to or less than 2^k, and k is less than or equal to n, the decomposition is *successful* (or the function is *decomposable*) for that bond set under the expected value of n. Otherwise, the function is *nondecomposable* for that bond set under the expected value of n.

After a successful decomposition, the input variable number of each subfunction (decomposed blocks, like $\phi_1, \phi_2, ..., \phi_k$ and F in Figure 1) is decreased. This will be illustrated with an example.

Figure 2a shows a Karnaugh map of the function f with DC outputs. We intend to decompose the function f into several subfunctions with the number of input variables for each subfunction no more than four. For example, L, M and N as shown in Figure 2c. According to the rules presented above, the incompatibility graph is created as shown in Figure 2d. After Graph Coloring, three colors (therefore, $\upsilon = 3$) which group the nodes as A = {0, 1, 3, 6}, B = {2, 5, 7} and C = {4} are obtained. The columns with the same color are combined horizontally by the rules: $(0, 0) \rightarrow 0, (0, x) \rightarrow 0, (x, 0) \rightarrow 0, (1, 1) \rightarrow 1, (1, x) \rightarrow 1, (x, 1) \rightarrow 1$ and $(x, x) \rightarrow x$. For example, columns 0, 1, 3 and 6 in Figure 2a are combined and replaced by a new vector [1, 1, 1, 0] as shown in Figure 3a.

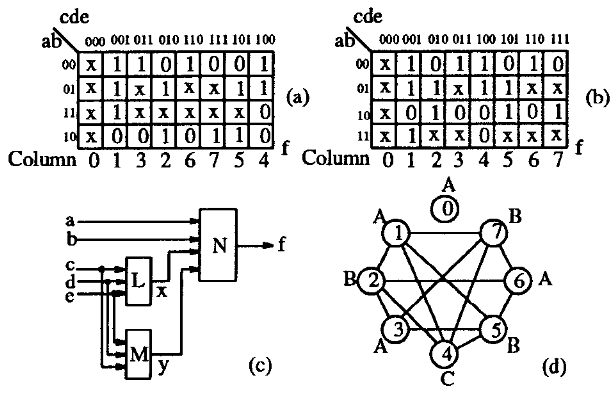

Figure 2. Decomposition example.

In the above example, we have chosen the variables *a* and *b* as the free set and variables *c*, *d* and *e* as the bond set. This partition resulted in a successful decomposition in the sense that the column multiplicity is less than or equal to three. In Figure 2c, *x* and *y* are the encoded outputs of the bond set. We developed a Variable Partition method to quickly find the "best" partitions. The detailed procedure can be found in [12] [13].

There are many methods [14] to produce the decomposed blocks (blocks L, M and N in Figure 2c). The authors developed a Bond Set Encoding algorithm which aims at simplifying the block N. Block L and M will be implemented by CLBs, it doesn't matter then how complex these two blocks are as long as the number of their input variables is no more than four. (we assume that the CLBs have up to four inputs for this example.) The encoding algorithm assigns adjacent codes (Gray code) to similar columns. This will make block N have more large cubes. The encoding results are shown in Figure 3b, 3c and 3d. The detailed procedure can be found in [12] [13].

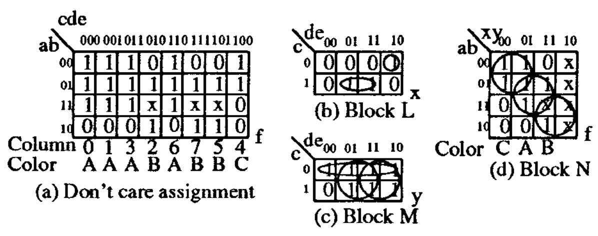

Figure 3. Final results.

Two variables can encode up to four columns ($2^2 = 4$). There are only three columns that need to be encoded in our example. We fill the remaining column (column 10 in Figure 3d) with don't cares (*DC column*). The newly introduced DC column will further simplify the block N. This example illustrates that even if the input function is completely specified, our algorithm may introduce DCs in the middle of the process. These are very useful for the minimization of later stages.

4.2. Decomposition of the incompletely specified multi-output functions

The above techniques can be easily expanded to multi-output functions with only a minor difference in performing the compatibility test of the columns. For a multi-output function:

Two columns are compatible if and only if no ε exists in all bitwise Intersections of the binary output vectors from the corresponding cells of the two columns. (A cell has an m-bit vector for a function with m outputs.)

This statement is consistent with that for the single output case. Figure 4 shows an example of a multi-output function (two outputs). We observe that columns 0 and 3 are compatible, while columns 1 and 3 are incompatible. All other operations for multi-output functions are similar to those of the single output ones.

ab \ cde	000	001	011	010	110	111	101	100
00	1 x	1 x	x 1	1 0	1 x	1 0	x 1	0 1
01	0 1	1 x	x 1	1 1	0 0	1 x	x 0	x 0
11	1 1	1 0	1 1	1 0	1 1	1 0	x 0	0 x
10	0 1	1 1	0 x	1 x	x 1	1 x	1 x	x x
Column	0	1	3	2	6	7	5	4

f

Figure 4. A multi-output function.

5. Two basic speedup approaches

There are three fundamental problems in the efficient implementation of a FPGA mapping program which is based on the Boolean decomposition of incompletely specified functions: (1) How to chose the bond set to minimize the column multiplicity? (2) How to minimize the column multiplicity for a given bond set? and (3) How to transform a nondecomposable function into several decomposable ones? The solution for the first one can be found in [12] [13]. The last two will be presented in more detail here.

5.1. Graph Coloring

Graph Coloring is a procedure in which every two nodes linked by an edge are assigned with different colors. *Minimum Graph Coloring* is one with the minimum number of colors in the final colored graph. There has been a substantial research on Graph Coloring in order to find algorithms for a quasi-optimum solution with the fastest possible speed.

Here the authors present a fast Graph Coloring method which is called the *"Color Influence Method"*. The main idea of this method is to evaluate the influence of the color assignment to a node over the entire graph, and chose the color which results in a minimum influence. The *minimum influence* means that the color assignment to a node will produce a minimum increase in the number of color-in-bar's. The *color-in-bar's* (restrictions) are the colors that the node cannot be assigned with, which are denoted by $\bar{A}$, $\bar{B}$..., $\overline{AB}$,$\overline{AC}$, ... as in Figure 5. After each color assignment to a node, the complexity of the graph is decreased. This is a greedy method with global evaluation. The next example is used to illustrate this method.

Figure 5a shows the graph to be colored. Start from the node with the most number of edges, that is node 2, assign color A to it. This color assignment means that nodes 1, 3, 5 and 6 cannot be assigned with color A. Denote this restriction on those nodes by color-in-bar $\bar{A}$'s, and remove all corresponding edges as shown in Figure 5b. Next, color the node with the most number of color-in-bar's. If there is more than one node with the same number of color-in-bar's, color the node with the most number of edges. If there is still more than one node, evaluate the influence of the color assignment on each node, and assign the node with a color which results in a minimum influence. If a node can be assigned with more than one color, the evaluation of the influence of each color assignment is also required. According to the rules stated above, nodes 5 and 6 are selected because they have the same number of color-in-bar's and the same number of edges. Assigning color B to node 6 will result in a restriction $\overline{AB}$ on node 1 and a restriction $\overline{AB}$ on node 5. While assigning color B to node 5 will result in a restriction $\overline{AB}$ on node 6 and a restriction $\bar{B}$ on node 4. Because one $\overline{AB}$ restriction and one $\bar{B}$ restriction result in less influence than two $\overline{AB}$ restrictions, assigning a color to node 5 produces less influence than assigning a color to node 6. Node 5 is colored with the color B as shown in Figure 5c. The same way, assign color C to node 6 as shown in Figure 5d, color B to node 1 as shown in Figure 5e. Nodes 3 and 4 are in the same condition. If node 3 is assigned with color B, node 4 can be assigned with color A or C. The final color assignment is shown in Figure 5f.

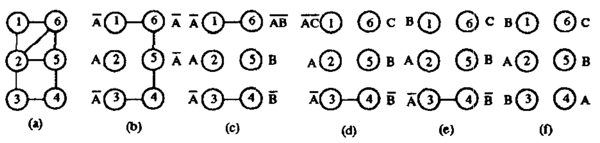

Figure 5. Graph Coloring example.

In summary: first, color the node with the most number of color-in-bar's. If there is more than one node, color the node with the most number of edges. If there is still more than one node, evaluate the influence and color a node with a color which results in a minimum influence. If a node can be assigned with more than one color, the evaluation of the influence of each color assignment is also required.

The above algorithm has been incorporated into a program, named *COLOR*, and was run on a networked SUN 4/370 (a 12.5 mips machine). The program was tested on graphs with different number of nodes (N = 100 → 1000) and different edge percentages (P = 10% → 90%). The maximum number of edges in a graph is N(N - 1)/2. The edge percentage is the percentage of this maximum number of edges. Edges in the graph were randomly generated. Figure 6 shows the results.

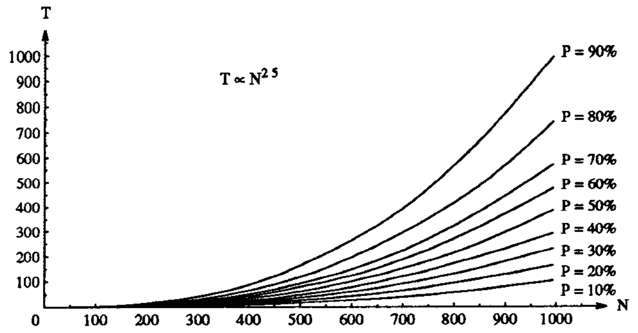

Figure 6. Graph Coloring results.

T is the running time of the program measured by the *time* command of the UNIX system. The units of T are seconds. By statistic analysis, it is found that the time (T) is proportional to the number of nodes (N) in a polynomial form $T \propto N^{2.5}$. Therefore, our algorithm executes in polynomial time (not in exponential time). For small graphs, we are able to verify that the algorithm gives the minimum solutions. Therefore we hope that it will give good results for larger graphs as well. But we were not able to verify this claim since we were not able to access an exact minimal optimizer.

5.2. Local Transformation

The known decomposition methods are passive in the sense that they only test whether a function is decomposable or not. If it is, the decomposition is carried out. But what do we do if the function is not decomposable?

The authors developed a decomposition approach which is called the *Local Transformation Method.* This method can transform a nondecomposable function into several decomposable ones. The basic idea of this method is to modify some columns in the original Karnaugh map to make them identical to some other columns in order to decrease the column multiplicity. Figure 7 shows an example.

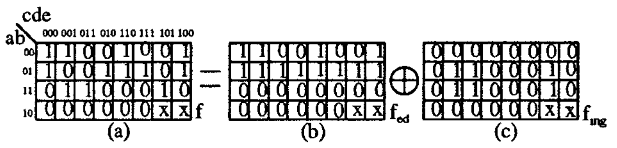

Figure 7. Local Transformation.

Figure 7a shows the original Karnaugh map (f) with a column multiplicity of four. First, the output value of the cube -10-1 and -1101 of the original Karnaugh map in Figure 7a are complemented, which leads to a Karnaugh map (f_{ed}) with a column multiplicity of two as shown in Figure 7b. Next, a compensation is made. Another Karnaugh map (f_{ing}) is created with the positions corresponding to the complemented cubes in the original Karnaugh map set to 1's and the others set to 0's as shown in Figure 7c. Finally, the EXOR operation of f_{ed} and f_{ing} produces the original function. That is:

$$f = f_{ed} \oplus f_{ing}$$

The presented method transforms a nondecomposable function (in sense of the column multiplicity less than or equal to two) into two decomposable functions (with the column multiplicities of both equal to two). The detailed procedure can be found in [12] [13].

6. Program *TRADE* and its evaluations

The techniques presented in the previous sections have been incorporated into a program named *TRADE* (*TRA*nsformation and *DE*composition) which reads in an input file written in *Espresso (.type fr)* format and outputs the result in *Blif* format with the input variables of each node no more than five. Cube Calculus is used in *TRADE* for all operations. We ran *TRADE* on a networked SUN 4/370 (a 12.5 mips machine). The results are listed in Table 1.

Table 1. Comparison table.

	TRADE			*MIS-PGA(phase 1)*			*MIS-PGA(new)*	
E	C	L	T	C	L	T	C	T
alu2	22	3	12.2	122	6	42.6	109	773.8
9sym	6	3	4.9	7	3	15.2	7	339.7
9symml	6	3	4.7	7	3	9.9	7	127.2
rd73	5	2	3.7	8	2	4.4	6	24.0
rd84	8	3	11.6	13	3	9.8	10	73.7
f51m	9	3	2.3	23	4	5.9	17	14.4
5xp1	11	2	4.3	21	2	3.5	18	22.4
z4ml	4	2	2.0	10	2	2.1	5	5.0
sao2	27	3	13.8	45	5	9.5	28	41.9
*bw**	27	1	0.3	28	1	8.3	28	17.3
misex1	14	2	3.4	17	2	1.7	11	2.7
clip	29	4	12.1	54	4	3.7	28	58.4
b9	29	4	28.7	47	3	2.3	39	27.6
misex2	31	4	17.0	37	3	1.4	28	3.4
duke2	159	6	370.7	164	6	16.4	110	203.7
root	21	3	9.8					
*bench**	16	2	1.0					
*fout**	26	2	4.3					
*test1**	66	3	21.1					
t-00	166	5	81.6					
*t-10**	152	5	64.6					
*t-30**	125	5	50.6					
*t-50**	83	5	32.8					
*t-70**	76	4	18.1					
*t-90**	46	3	10.2					

* Incompletely specified function.

All results are verified by the *"verify"* command of *MIS-II* system. The results listed under *MIS-PGA(phase 1)* are from [15]. The results listed under *MIS-PGA(new)* are from [2]. Both of them were run on the DEC 5500 (a 28 mips machine). There is no delay information provided in [2]. The examples "root", "bench", "fout" and "test1" are taken from the Espresso package, and all of them are incompletely specified functions except "root". "t-00" is obtained by changing all DC outputs in "test1" to OFF outputs, therefore, it is a completely specified function. "t-10" is obtained by randomly changing 10 percent of the OFF outputs in "t-00" to DC outputs. The same way, "t-20" to "t-90" is obtained by randomly changing 20 to 90 percent of the OFF outputs in "t-00" to DC outputs, respectively. Because we were not able to access the *MIS_PGA(phase 1)* and *MIS_PGA(new)* programs, we couldn't make comparisons of the incompletely specified functions. In Table 1, E is the name of the example. C is the number of CLBs in the final mapped circuit. T is the running time of the program measured by the *time* command of the UNIX system. The units of T are seconds. L is the longest path (number of CLBs) that a signal must go from the primary input to the primary output in the final mapped circuit.

From Table 1, we observe that if the DC outputs are maintained, the number of CLBs can be greatly decreased. However, even for the completely specified functions, our program found better results than *MIS-PGA(phase 1)* and *MIS-PGA(new)* with respect to both the delay and area minimizations.

7. Conclusions and future work

A new general approach to the decomposition of incompletely specified functions and its application to the FPGA mapping have been presented. Variable Partitioning, Graph Coloring and Local Transformation are the outstanding features of this approach. One of the main advantages of this approach is that it is intended for incompletely specified functions, thus giving for such functions much better results than the other existing methods. Compared with the existing FPGA mapping approach, our method is totally new. We developed a fast Graph Coloring method for the don't care assignment, therefore, the program can accept an incompletely specified function and perform a quasi-optimum assignment to the unspecified part of the function. We developed a high quality heuristic method to chose the "best" partitions, avoiding an exhaustive test of all possible decomposition charts which is impractical when there are many input variables in the input function. We introduced the Local Transformation concept, which can transform a nondecomposable function into several decomposable ones, making it possible to apply decomposition method to the FPGA mapping.

The program has been successfully verified and benchmarked on several MCNC examples and some incompletely specified functions. It is still possible to further improve both its speed and quality of the generated solutions. Currently we are working with two-dimensional Karnaugh maps. A possible extension would be to develop algorithms to operate on three-dimensional or multi-dimensional Karnaugh maps.

References

1. Xilinx, Inc., *Xilinx Programmable Gate Array Data Book*, 1992.
2. R. Murgai, N. Shenoy, R. K. Brayton, and A. Shangiovanni-Vincentelli, "Improved Logic Synthesis Algorithms for Table Look Up Architectures," *ICCAD 1991*, pp. 564-567, Santa Clara, CA, Nov. 1991.
3. D. Filo, J. C. Yang, F. Mailhot, and G. D. Micheli, "Technology Mapping for a Two-Output RAM-based Field-Programmable Gate Array," *European DAC*, pp. 534-538, Feb. 1991.
4. R. Francis, J. Rose, and Z. Vranesic, "Chortle-crf: Fast Technology Mapping for Lookup Table-Based FPGAs," *Proc. 28th ACM/IEEE DAC*, pp. 227-233, San Francisco, CA, June 1991.
5. K. Karplus, "Xmap: A Technology Mapper for Table-lookup Field-Programmable Gate Arrays," *Proc. 28th ACM/IEEE DAC*, pp. 240-243, San Francisco, CA, June 1991.
6. N. S. Woo, "A Heuristic Method for FPGA Technology Mapping Based on the Edge Visibility," *Proc. 28th ACM/IEEE DAC*, pp. 248-251, San Francisco, CA, June 1991.
7. R. L. Ashenhurst, "The Decomposition of Switching Functions," *Proc. Int'l Symp. Theory of Switching Function*, pp. 74-116, 1959.
8. M. A. Perkowski and J. E. Brown, "A Unified Approach to Designs Implemented with Multiplexers and To the Decomposition of Boolean Functions," *Proc. ASEE Annual Conf.*, pp. 1610-1618, 1988.
9. H. A. Curtis, "Generalized Tree Circuit-The Basic Building Block of an Extended Decomposition Theory," *J. ACM*, vol. 10, pp. 562-581, 1963.
10. J. P. Roth and R. M. Karp, "Minimization over Boolean Graphs," *IBM J. of Research and Development*, vol. 6, no. 2, pp. 227-238, April, 1962.
11. L. B. Nguyen, M. A. Perkowski, and N. B. Goldstein, "PALMINI - Fast Boolean Minimizer for Personal Computers," *Proc. 24th ACM/IEEE DAC.*, pp. 615-621, 1987.
12. W. Wan, "A New Approach to the Decomposition of Incompletely Specified Functions Based on Graph Coloring and Local Transformation and Its Application to FPGA Mapping," Thesis, Portland State University, Portland, OR, May 1992.
13. W. Wan and M. A. Perkowski, "TRADE: A Lookup Table FPGA Mapper Based on a Generalized Boolean Decomposition," EE Dept. Report, Portland State University, Portland, OR, April 1992.
14. S. Yang and M. J. Ciesielski, "Optimum and Suboptimum Algorithms for Input Encoding and Its Relationship to Logic Minimization," *IEEE Trans. CAD*, vol. 10, no. 1, pp. 9-12, Jan. 1991.
15. R. Murgai, N. Shenoy, R. K. Brayton, and A. Shangiovanni-Vincentelli, "Performance Directed Synthesis for Table Look Up Programmable Gate Arrays," *ICCAD 1991*, pp. 572-575, Santa Clara, CA, Nov. 1991.