Decomposition of Boolean Functions Specified by Cubes. Part II: Practical Results*

M. NOWICKA, T. ŁUBA

Warsaw University of Technology
Institute of Telecommunications
Nowowiejska 15/19, Warsaw, Poland,
luba@tele.pw.edu.pl.

October 17, 1998

Abstract. We describe DEMAIN – a program for the decomposition of multiple-output, incompletely specified Boolean functions represented in the consistent cube notation (*fr* format). We compare the results of DEMAIN with those of other tools, and show that, in most cases, DEMAIN produces better results on standard benchmark circuits and other designs. DEMAIN uses parallel, disjoint serial, and nondisjoint serial decompositions as steps in an iterative algorithm. Heuristic strategies are used to decide which type of decomposition should be used at a given step. This permits us to minimize either the area (number of cells) or delay (number of levels).
Keywords: decomposition, disjoint, FPGA, look-up table, nondisjoint, parallel, serial, strategies.

* This research was supported by a grant awarded by the Dean of the Faculty of Electronics and Information Technology, Warsaw University of Technology, in 1998/99.

1. Introduction

A renewed interest in functional decomposition in recent years was caused by the introduction of field-programmable gate arrays by Xilinx in 1986, and other companies (AT&T, Actel, Altera) in succeeding years. FPGAs are modern logic devices which can be programmed by the users to implement any logic circuit [3]. Because of their short turnaround time compared with that of the standard ASIC process, they have recently become very popular in rapid system prototyping. Many FPGA architectures have been proposed; the architecture based on look-up tables (LUT) is one of the most popular ones. It consists of many identical logic blocks, each of which can implement any Boolean function of k inputs, where k is a small integer. For example, in Xilinx XC3000, k is 5.

The earlier algebraic methods of multilevel synthesis [2] could not be easily adapted to the look-up-table model; consequently researchers switched to decomposition methods based on functional dependencies. Moreover, many improvements were added to classical Ashenhurst and Roth-Karp methods aiming at efficient technology mapping algorithms for FPGAs [11], [12], [16], [17], [18], [25].

Existing FPGA-oriented technology mapping algorithms can be divided roughly into two categories according to their optimization objectives. The algorithms in the first category emphasize minimizing the area (i.e., the number of LUTs). These algorithms include Chortle [8], MIS-pga [15], Trade [25] and TOS [7], [20]. The algorithms in the second category emphasize minimizing the delay (number of levels). These are DAG-Map [5] and FlowMap [6]. Several algorithms have been implemented in both the area- and delay-minimization versions, for example MIS-pga, Chortle, and ASYL [1].

Although many of the above-mentioned algorithms produce good results, they have the common limitation that, for a given design, a simple solution optimized with respect to a certain criterion is produced, while other solutions, preferable under different optimization objectives, are ignored. Usually the logic depth (length of critical path) and the number of LUTs for these solutions vary significantly. For example, the mapping results on the Alu2 benchmark circuit, obtained by area- and delay-minimization versions of the Flow-Map and Chortle are 125 cells/12 levels (area minimization) or 149 cells/8 levels (delay minimization) and 128 cells/13 levels or 227 cells/9 levels, respectively [6]. In general, the area-minimized designs have much larger depth, while delay-minimized designs use many more LUTs. It is very likely that in practice the best design does not come from either of these two extreme solutions. It is important to let the designer have the flexibility to choose from a set of mapping solutions with smooth trade-off between area and depth.

In this paper we propose a balanced multilevel decomposition method to solve the problem of trading-off area and depth in LUT-based FPGA technology mapping.

Specifically, we are interested in obtaining a set of mapping solutions for a given design, which can meet various area and depth requirements.

Unlike the traditional approach of first performing logic minimization and then mapping the design using decomposition and other methods, the proposed method starts the mapping process straight away using different decomposition strategies. An important feature of multilevel decomposition is an attempt to strike a balance between the wider known serial decomposition (described in Part I of the paper) and the so-called parallel decomposition. Such a strategy, on one hand, is capable of detecting and utilizing the fact that a group of outputs depend on the same set of input variables, and, on the other hand, eliminates redundant variables.

The paper is structured as follows. Section 2 presents DEMAIN, a program for the decomposition of multiple-output Boolean functions specified by cubes. The algorithm of DEMAIN is described in terms of its main parallel and serial decomposition procedures. In Section 3 experimental results of DEMAIN are shown and compared with the results of other university and commercial tools. Sections 4 and 5 describe the main decomposition procedures, i.e., parallel and serial decomposition. They are used in Section 6 to present decomposition strategies for cell based technology mapping. Finally, Section 7 summarizes Part II of the paper.

2. DEMAIN

DEMAIN is a program, developed at the Institute of Telecommunications, Warsaw University of Technology, for the decomposition of multiple-output, incompletely specified Boolean functions represented in the consistent cube notation, also known as the *fr* format. It can also be used for the reduction of the number of inputs of a function, when an output depends on only a subset of the inputs. From this point of view DEMAIN is a tool specially dedicated to FPGA-oriented technology mapping. DEMAIN was written in C++.

A characteristic feature of DEMAIN is its interactive mode, in which the user can steer the synthesis process according to the requirements of the chosen type of FPGA. The steering process is based on the idea of intertwining different decomposition strategies: at each step, either a parallel, or a serial disjoint, or serial non-disjoint decomposition is performed (for definitions of these concepts see Sections 4 and 5 below, and also Part I of the paper).

Given a function f with n inputs and m outputs, and a logic cell (LC) with C_{in} inputs and C_{out} outputs, a decomposition process is carried out by the following steps[1].

1) If $n \leq m$, use parallel decomposition. Continue iteratively for each of the obtained components.

[1] For the definition of the parameters of a decomposition, the reader should refer to Fig. 1 in Part I of the paper.

2) If $n > m$ try disjoint serial decomposition, with the number of block g inputs equal to the number C_{in} of LC inputs, and the number of block g outputs equal to the number C_{out} of LC outputs. If such a serial decomposition is found, find the corresponding h. Continue iteratively with $f = h$. Otherwise, try to find g with fewer inputs than C_{in} and/or fewer outputs than C_{out}. If such a g is found, find h. Continue iteratively with $f = h$. In case a g that fits in one cell cannot be found, try a larger g. This step is repeated until a decomposition with a function g larger than the cell exists. Find h and continue with $f = h$. Function g will have to be decomposed later.

The decomposition is carried out until all resulting subfunctions are small enough to fit into logic cells available in the assumed implementation technology. In DEMAIN each logic cell is treated as a universal cell capable of implementing any k-input, l-output Boolean function. Because the number of cell inputs accepted by DEMAIN is arbitrary, its applicability is wider than those of other FPGA-oriented tools.

3. EXPERIMENTAL RESULTS

The multilevel synthesis based on balancing parallel and serial decompositions proved to be very effective for the logic synthesis of circuits implemented with FPGAs. To demonstrate this, in Tables 1 and 2 we show the results obtained from DEMAIN for a sample of MCNC benchmark circuits, and compare them with those obtained using alternative synthesis tools, which aim at minimizing the circuit complexity (Table 1 shows the number of Xilinx XC3000 cells), or maximizing the circuit speed (Table 2 shows the depth of the circuit and the number of Xilinx cells). It is assumed that a Xilinx cell can implement any one-output Boolean function of five variables, or a two-output Boolean function, provided each of the

Table 1: Number of cells in decomposed circuits.

Name	**DEMAIN**	Chortle-Crf	ASYL	MIS	FGSyn	mulop
5xp1	**9**	20	13	17	9	9
Alu2	**47**	83	60	84	55	51
Clip	**10**	–	33	23	18	14
f51m.	**8**	–	14	11	8	8
Misex1	**8**	14	13	9	8	9
Misex2	**24**	–	24	23	22	24
rd73	**5**	–	8	7	5	5
rd84	**7**	53	14	12	8	8
sao2	**16**	–	30	28	25	20
z4ml	**4**	3	4	6	4	4
9sym	**5**	–	8	7	–	7

outputs depends on at most four of the five input variables. The data shown in Tables 1 and 2 are taken from [1], [4] (for BDD), [6], [8], [11], [21]. DEMAIN produced better results than other tools in almost all cases.

Table 2: Number of levels/cells in decomposed circuits.

Name	**DEMAIN**	BDD	Mispga-d	Chortle-d	FlowMap	ASYL	LOG/iC
5xp1	**2/9**	2/19	2/21	4/29	3/25	3/13	2/14
Misex1	**2/8**	2/18	2/17	3/25	2/25	2/13	2/11
Rd84	**2/7**	3/13	3/13	4/61	4/43	3/14	3/11
Z4m1	**2/4**	2/6	2/10	3/20	3/13	2/4	3/7
9sym	**3/5**	3/6	3/7	4/76	5/61	3/8	3/6
Alu2	**5/53**	6/95	6/122	9/227	8/162	5/137	–
Sao2	**3/18**	4/26	5/45	4/58	–	3/36	4/24
Rd73	**2/5**	2/8	2/8	4/52	–	2/8	7/14
Misex2	**3/24**	2/41	3/37	2/52	–	3/33	–
F51m.	**2/10**	3/81	4/23	5/65	–	3/16	–
Clip	**2/16**	–	4/54	–	–	4/46	4/24

Additional experiments were performed to compare the results produced by DEMAIN with those produced by the commercial tools: LOG/iC from ISDATA [7] and ALTERA's MAX+Plus2. In the experiment with LOG/iC, we used two sets of sample circuits. The first group includes typical MCNC benchmark circuits, which are completely, or almost completely, specified Boolean functions. The results are shown in Table 2.

The second group includes examples especially prepared [9] to test synthesis algorithms in the case of strongly unspecified Boolean functions. The results obtained are compared to those of DEMAIN in Table 3. As we can conclude from this comparison, DEMAIN performs better than LOG/iC, and the advantage of DEMAIN grows with the size of a function and with the number of "don't cares" in a function.

Table 3: Comparison with LOG/iC results.

	LOGi/C		DEMAIN	
	XC2000	XC3000	XC2000	XC3000
i20x20a	6/4	4/4	5/3	4/3
i20	6/3	3/2	5/2	2/2
i30	12/4	4/2	5/3	3/2
i40	35/7	16/5	10/5	5/3
i50	50/8	23/6	14/6	6/3

In the experiment with ALTERA software (Table 4), each benchmark circuit was initially compiled using the ALTERA compiler. ALTERA's results, i.e., the number of logic cells for FLEX devices required to implement the benchmark, are listed in the column labelled MAX+Plus2. Then, DEMAIN was used to decompose each benchmark into a netlist of cells, where the cells have four inputs and one output. The decomposed multilevel structure of each benchmark was next fed as new data to the MAX+Plus2 compiler to obtain the FLEX implementation of a given circuit. The final results obtained by MAX+Plus2 software supported by DEMAIN's decomposition procedures are listed in the second column of Table 4. The use of DEMAIN resulted in a significant reduction in the number of FLEX cells.

Table 4: Comparison with MAX+Plus2 results.

Name	DEMAIN	MAX+Plus2
9sym	10	98
Clip	19	404
Rd73	9	98
Root	37	113
Sao2	31	125
Z4	6	88

DEMAIN was also applied in the design a cryptographic device; the design was based on the algorithm specified by the Data Encryption Standard (DES). It was implemented on ALTERA's FLEX EPF8282LC84 device [22].The decomposed structure obtained using DEMAIN consisted of a total of 194 logic cells. Therefore, one piece of EPF8282LC84 was sufficient to realize the entire DES algorithm. On the other hand, the structure proposed by ALTERA's design tool consisted of a total of 610 logic cells, requiring three pieces of EPF8282LC84.

4. Parallel decomposition

Consider a multiple-output function *f*. Assume that *f* has to be decomposed into two components, *g* and *h*, with disjoint sets Y_g and Y_h of output variables. This problem occurs, for example, when we want to implement a large function using components with a limited number of outputs. Note that such a parallel decomposition can also alleviate the problem of an excessive number of inputs of *f*. This is because, for typical functions, most outputs do not depend on all input variables. Therefore, the set X_g of input variables on which the outputs of

Y_g depend, may be smaller than X. Similarly, for the set X_h of input variables on which the outputs of Y_h depend may be smaller than X. As a result, components g and h have not only fewer outputs, but also fewer inputs than f. The exact formulation of the parallel decomposition problem depends on the constraints imposed by the implementation style. One possibility is to find sets Y_g and Y_h, such that the combined cardinality of X_g and X_h is minimal. Partitioning the set of outputs into only two disjoint subsets is not an important limitation of the method, because the procedure can be applied again for components g and h. It is essential, however to require that no logic minimization be performed before the parallel decomposition, because decomposition properties may be destroyed by minimization.

Example 1. Consider the multiple-output function given in Table 5. The minimal sets of input variables on which each output of f depends are

y_1: $\{x_1, x_2, x_6\}$
y_2: $\{x_3, x_4\}$
y_3: $\{x_1, x_2, x_4, x_5, x_9\}$, $\{x_1, x_2, x_4, x_6, x_9\}$
y_4: $\{x_1, x_2, x_3, x_4, x_7\}$
y_5: $\{x_1, x_2, x_4\}$
y_6: $\{x_1, x_2, x_6, x_9\}$

Table 5: Function f.

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	y_1	y_2	y_3	y_4	y_5	y_6
1	0	0	0	1	1	1	0	0	0	0	0	0	0	–	0
2	1	0	1	0	0	0	0	0	0	0	0	–	1	0	1
3	1	0	1	1	1	0	0	0	0	0	1	1	0	1	1
4	1	1	1	1	0	1	0	0	0	0	1	1	1	1	0
5	1	0	1	0	1	0	0	0	0	0	0	0	–	0	1
6	0	0	1	1	1	0	0	0	0	1	1	0	1	0	0
7	1	1	1	0	0	0	0	0	0	1	0	–	0	1	0
8	1	0	1	1	0	1	0	0	0	1	1	0	0	–	1
9	1	0	1	1	0	1	1	0	0	–	1	0	1	–	1
10	1	1	1	0	0	0	0	1	0	1	0	1	0	1	–
11	0	0	0	1	1	1	0	0	1	0	0	1	0	–	1
12	0	0	0	1	1	0	0	0	1	–	–	1	0	0	0

An optimal two-block decomposition, minimizing the card X_g + card X_h (where card X is the cardinality of X), is $Y_g = \{y_2, y_4, y_5\}$ and $Y_h = \{y_1, y_3, y_6\}$, with $X_g = \{x_1, x_2, x_3, x_4, x_7\}$ and $X_h = \{x_1, x_2, x_4, x_6, x_9\}$. The truth tables for components g and h are shown in Table 6.

Table 6a: Function g.

	x_1	x_2	x_3	x_4	x_7	y_2	y_4	y_5
1	0	0	0	1	0	0	0	0
2	1	0	1	0	0	0	1	0
3	1	0	1	1	0	1	0	1
4	1	1	1	1	0	1	1	1
5	0	0	1	1	0	1	1	0
6	1	1	1	0	0	0	0	1
7	1	0	1	1	1	–	1	–

Table 6b: Function h.

	x_1	x_2	x_4	x_6	x_9	y_1	y_3	y_6
1	0	0	1	1	0	0	0	0
2	1	0	0	0	0	0	0	1
3	1	0	1	0	0	0	1	1
4	1	1	1	1	0	0	1	0
5	0	0	1	0	0	1	0	0
6	1	1	0	0	0	1	1	0
7	1	0	1	1	0	1	0	1
8	0	0	1	1	1	0	1	1
9	0	0	1	0	1	–	1	0

5. Serial decomposition [2]

As follows from Part I the main task in constructing a serial decomposition of a function f with given sets U and V is to find a blanket $\beta_g \geq \beta_v$ which also satisfies the second condition of Theorem 1. In DEMAIN we choose to use a set system σ_g to obtain the smallest number of outputs in g. We also use σ_y instead of β_y. As discussed in Part I, Section 9, we find σ_g by merging blocks of β_v as much as possible. The concept of compatibility of β_v blocks is used in the calculation of β_g.

Two blocks B_i and B_j of blanket β_v are compatible, if blanket β_{ij} obtained from β_v by merging B_i and B_j into a single block satisfies the second condition of Theorem 1, that is, if $\beta_u * \beta_{ij} \leq \beta_y$. Otherwise blocks B_i and B_j are incompatible. A subset δ of blocks of the input blanket β_v is a compatibility class of blocks if the blocks in δ are pair-wise compatible. A compatibility class is maximal if it is not contained in any other compatible class.

From the computational point of view, finding maximal compatibility classes is equivalent to finding maximal cliques in a graph $\Gamma = (N, E)$, where the set N of nodes is the set of blocks of β_v, and the set E of edges is formed by compatible pairs.

[2] We refer the reader to Part I for further details concerning serial decomposition

The next step in the calculation of σ_g is the selection of a set of maximal classes, with minimal cardinality, that covers all the blocks of β_v. The minimal cardinality ensures that the number of blocks of σ_g, and hence the number of outputs of the function g, is as small as possible.

In certain heuristic strategies, both procedures (finding maximal compatibility classes and then finding the minimal cover) can be reduced to the graph coloring problem. Thus the task of generating σ_g can be reduced to the graph coloring problem. Blocks of β_v are treated as nodes in N and compatible pairs $e = (B_i, B_j) \in E$ as edges of a graph $\Gamma = (N, E)$.

Although, the problem of finding the minimal chromatic number for a given graph Γ is NP-complete, a number of fast heuristics have been developed for it [24].

Calculating σ_g corresponds to finding the minimal number k of colors for graph $\Gamma = (N, E)$ and the number of colors corresponds to the number of blocks in set system σ_g.

To illustrate this process we continue with Example 3 from Part I, where the blocks of β_v have been numbered as shown:

$$\beta_v = \{\overset{B_1}{\overline{1,2,3}}\,;\overset{B_2}{\overline{3,7}}\,;\overset{B_3}{\overline{1,2}}\,;\overset{B_4}{\overline{4}}\,;\overset{B_5}{\overline{6,7}}\,;\overset{B_6}{\overline{5}}\,;\overset{B_7}{\overline{4,6}}\,\},$$

The following are the incompatible pairs: (B_1, B_4), (B_1, B_6), (B_1, B_7), (B_2, B_4), (B_2, B_6), (B_2, B_7), (B_3, B_6), (B_4, B_5), and (B_5, B_7). Therefore, two colors are needed here: one for nodes B_1, B_2, B_3, B_5 and another for nodes B_4, B_6, B_7. The sets of nodes assigned to different colors form the blocks of σ_g:

$$\sigma_g = \{\,\overline{1,2,3,6,7};\ \overline{4,5,6}\,\},$$

which directly determines function g in serial decomposition of function f.

In procedures of actual version of DEMAIN functions f and h are specified by cubes. Function g, however, because of its small size (sizes of typical FPGA-based cells are limited to only few inputs), is specified by truth table.

6. Decomposition strategies for cell-based technology mapping

The intertwining of serial and parallel decomposition strategies opens up several interesting possibilities in multilevel decomposition. Experimental results show that the right balance between the two strategies and the selection of a set V in serial decomposition severely influence the area and depth of the resultant network. The overall performance can be significantly improved by applying non-disjoint serial decomposition. We now present two approaches to optimize the design for area and depth.

In multilevel decomposition with area optimization, serial decomposition is given more weight.

The following guidelines are applied while optimizing for LUTs:

1) Serial decomposition is applied and a decomposition which gives the least number of g block outputs with minimum number of blocks in σ_g is chosen.

a) A disjoint serial decomposition is calculated.

b) It is verified if a non-disjoint serial decomposition can reduce the number of blocks in the g function set system σ_g, thereby reducing the number of g block outputs. It is worth noting that, if such a decomposition exists, it will reduce the complexity of the h function without increasing the complexity of the g function.

2) During the iterative application of serial decomposition, if an h block is obtained for which the number of inputs is less than or equal to the number its outputs + 1, parallel decomposition is performed and the function is split into two.

Let us illustrate the influence of the number of blocks in the g function output set system. Consider the MCNC benchmark *Sao2* with 10 inputs and 4 outputs. For simplicity, let us consider only disjoint serial decomposition and parallel decomposition.

Example 1. Disjoint serial decomposition with the least number of output set system blocks is chosen at every level. The decomposed structure is shown in Fig.1.

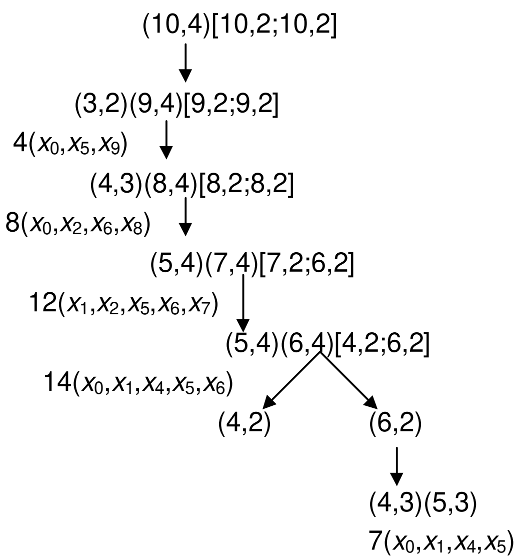

Fig. 1. Sao2 decomposed using disjoint serial decomposition with the least number of σ_g.

The structure can be understood as follows: The top entry (10,4) indicates the number of inputs and outputs of the primary function. Two entries (3,2) and (9,4), which are given below the arrow, show the number of inputs and outputs for functions $g(3,2)$ and $h(9,4)$, respectively. The entry with variables under (3,2) shows the inputs to the g block. An integer number in front of this entry shows the number of blocks in the output set system of the g block. The h function variables are renumbered at every iterative step. In Fig. 1, after extracting $g(x_0,x_5,x_9)$, the inputs to h are renumbered. This means that the primary inputs x_1, x_2, x_3, x_4 ,x_6, x_7, x_8 in the next iteration (it is not a new level in this case), are denoted as x_0,..., x_6, and x_0, x_2, x_3, x_4, x_6 means primary x_1, x_3, x_4, x_6, x_8. Additionally at each step results of a possible parallel decomposition are indicated inside the right hand side square bracket. In this example parallel decomposition is applied to split the $h(6,4)$ into (4,2) and (6,2) components. The whole structure needs 16 Xilinx (XC 3000) cells in 6 levels.

Example 2. In this example "the first found" serial decomposition is selected. The same benchmark function needs 23 Xilinx cells in 5 levels. The structure is shown in Fig. 2. Parallel decomposition is applied to split $h(7,4)$ into two $h(7,2)$ blocks.

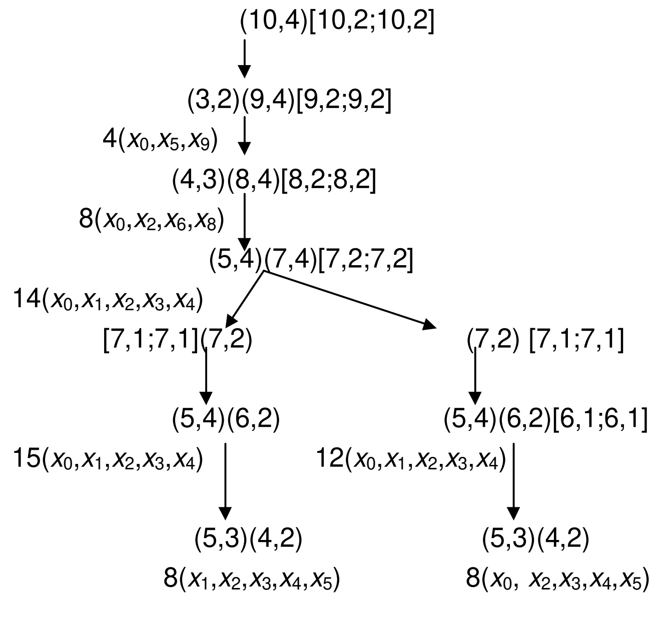

Fig. 2. Sao2 decomposed with the strategy of "the first found" serial decomposition.

The next important issue is non-disjoint serial decomposition. When parallel decomposition is not desirable (number of inputs > number of outputs) and disjoint serial decomposition does not give a satisfactory g function (number of g block outputs is too high), non-disjoint serial decomposition is considered. The introduction of non-disjoint serial decomposition may reduce the number of LUTs at a given level, but may increase the complexity of the h function and increase the total number of LUTs required to implement the function. An heuristic is used to avoid the situation as far as possible. Non-disjoint serial decomposition is chosen only if the number of unused blocks in the g function output encoding is more than that for disjoint serial decomposition. The number of unused blocks in

g function output encoding is the difference between the possible number of codes generated by the g function outputs and the number of their output set system blocks. This is explained in Figs.3 and 4.

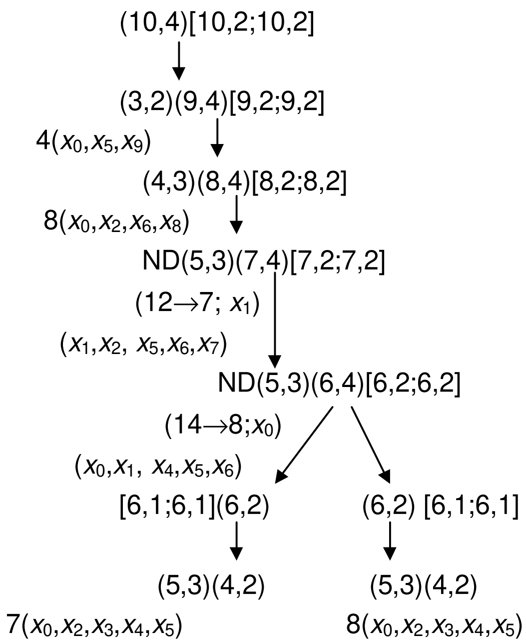

Fig. 3. Sao2 decomposed with non-disjoint serial strategy

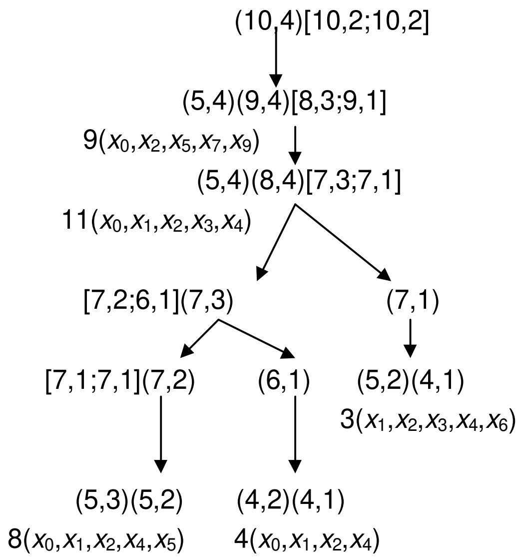

Fig. 4. Sao2 decomposed with special encoding

Let us consider the Sao2 example. The decomposed structure obtained without using the heuristic is shown in Fig 3. Non-disjoint serial decomposition (ND) is applied at the third step. It produces an h block with 7 inputs and 4 outputs using only three Xilinx cells to implement a g block with 5 inputs and 3 outputs. Disjoint serial decomposition (Fig.1) produced an h block with the same number of inputs and outputs but using 4 Xilinx cells to implement a g function with 5 inputs and 4 outputs. Non-disjoint decomposition has reduced the number of Xilinx cells needed at step three. However, for the disjoint serial decomposition, the number of blocks in the g function output set systems was 12. The number of blocks left unused in encoding was 4. For the non-disjoint decomposition, the number of blocks in the g function output set systems is 7. This is shown in Fig. 3 by the arrow mark placed between the numbers 12 and 7 (in the bracket below the ND(5,3) component). In the bracket a variable which is shared in both g and h components is also shown (x_1 in this case). One block is left unused in the encoding process. This has resulted in a more complex h function for further decomposition and the implementation needs 17 Xilinx cells in 6 levels.

The best decomposed structure can obtained using a special encoding of g block outputs. This is shown in Fig. 4. The encoding is applied for the second g(5,4) block; it results in a better parallel decomposition. The parallel decomposition of $h(8,4)$ block results in two, (7,3) and (7,1) components. Therefore, the implementation needs 18 Xilinx cells in 5 levels.

In multilevel decomposition with depth optimization, a set of three guidelines are applied. They are:

1) The set V for a new serial decomposition is selected from the original inputs of function h, if possible;

2) A g function with fewer outputs is selected, even if it requires more LUTs at a given level.

3) Parallel decomposition is given more weight as it does not increase the number of levels.

7. Conclusions

The approaches to decomposition of Boolean functions presented in this paper rely on an effective "balanced" decomposition strategy and appropriate graph coloring for non-disjoint serial decomposition. These methods were already used in decomposition algorithms; however they were never applied together in a technology specific mapper targeted at a look-up table FPGA structure. This paper shows that it is possible to apply the non-disjoint serial decomposition method for the synthesis of FPGA-based circuits directed towards area or delay optimization. The experimental results demonstrate that our decomposition tool DEMAIN performed better than all the other university and commercial tools available for comparison.

The proposed methodology is general, in the sense that other kinds of implementations can be processed. In particular the decomposition algorithms can be also applied to PLA-based implementations.

For a PLA based implementation, the design objective is to decompose the original function into components corresponding to a set of interconnected PLAs, so that the overall area of the resulting logic network is minimized. The silicon area taken by a PLA can be estimated as $S = (2n + m)P$, where n, m and P denote the number of inputs, outputs, and product terms, respectively. To illustrate the effectiveness of the logic synthesis procedure based on functional decomposition, consider one of the MCNC benchmarks, RD84, implementing a data-path logic. The RD84 has 8 inputs, 4 outputs and 256 cubes. Thus, the estimated area of the PLA is: $S = (2\cdot8 + 4)\cdot256 = 5120$. When the circuit is decomposed onto three components g_0, g_1 and h, where both g_0 and g_1 have 4 inputs and 3 outputs, and component h has 6 inputs and 4 outputs, respectively, the total area of 3 the PLAs is only 10% of that of the originally minimized single PLA. It is worth emphasizing that this is better than the best result obtained for this circuit by a tool specially prepared for PLA-based synthesis [26].

References

[1] P. Abouzeid, B. Babba, M. Crastes and G. Saucier, Input-Driven Partitioning Methods and Application to Synthesis on Table-Lookup-Based FPGAs, *IEEE Trans. on CAD*, Vol. 12, No. 7, pp. 913-925, 1993.

[2] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli and A. R. Wang, MIS: A Multiple-Level Logic Optimization System, *IEEE Trans. on CAD*, Vol. 6, No. 6, pp. 1062-1081, 1987.

[3] S. D. Brown, R. J. Francis, J. Rose and Z. G. Vranesic, *Field-Programmable Gate Arrays*. Kluwer Academic Publisher, Dordrecht, 1992.

[4] S. -C. Chang, M. Marek-Sadowska and T. Hwang, Technology Mapping for TLU FPGAs Based on Decomposition of Binary Decision Diagrams. *IEEE Trans. on CAD*, Vol. 15, No. 10, pp. 1226-1236, 1996.

[5] K. C. Chen, J. Cong, Y. Ding, A. B. Kahng and P. Trajmar, DAG-Map: Graph-based FPGA technology mapping for delay optimization. *IEEE Design and Test of Computers*, pp.7-20, Sept. 1992.

[6] J. Cong and Y. Ding, FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs, *IEEE Trans. on CAD*, Vol. 13, No. 1, pp. 1-12, 1994.

[7] A. Ditzinger, Presentation of the LOG/iC2 CPLD and FPGA Design Tool, *Proc. International Workshop on Logic and Architecture Synthesis*, pp. 222-227, Grenoble, 1996.

[8] R. J. Francis, J. Rose and Z. Vranesic, Chortle-crf: Fast Technology Mapping Program for Lookup Table-Based FPGAs, *Proc. 28th Design Automation Conference*, pp. 227-233, San Francisco, 1991.

[9] L. Jóźwiak and P. Konieczny, Input Support Minimization for Efficient PLD and FPGA Synthesis. *Proc. International Workshop on Logic and Architecture Synthesis*, pp. 30-32, Grenoble, 1996.

[10] L. Jóźwiak, General Decomposition and its Use in Digital Circuit Synthesis. *VLSI Design,* Vol. 3, Nos. 3-4, pp. 225-248, 1995.

[11] Y. -T. Lai, K. -R. R. Pan and M. Pedram, OBDD-Based Function Decomposition: Algorithms and Implementation. *IEEE Trans. on CAD*, Vol. 15, No. 8, pp. 977-990, 1996.

[12] T. Łuba, H. Selvaraj, M. Nowicka and A. Kraśniewski, Balanced Multilevel Decomposition and its Applications in FPGA-Based synthesis, in G. Saucier, A. Mignotte (eds.), *Logic and Architecture Synthesis*, Chapman&Hall, 1995.

[13] T. Łuba and H. Selvaraj, A General Approach to Boolean Function Decomposition and its Applications in FPGA-Based Synthesis. *VLSI Design.* Vol. 3, Nos. 3-4, pp. 289-300, 1995.

[14] T. Łuba, Decomposition of Multiple-Valued Functions. *25th International Symposium on Multiple-Valued Logic*. Bloomington, Indiana, pp. 256-261, 1995.

[15] R. Murgai, N. Shenoy, R. K. Brayton and A. Sangiovanni-Vincentelli, Improved logic Synthesis Algorithm for Table Look Up Architectures, *Proc. IEEE International Conference on Computer Aided Design*, pp. 564-567, Santa Clara, 1991.

[16] M. Nowicka, T. Łuba and H. Selvaraj, Multilevel Decomposition Strategies in Decomposition-Based Algorithms and Tools. *Proc. International Workshop on Logic and Architecture Synthesis*, pp. 129-136, Grenoble, 1997.

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

[18] W. -Z. Shen, J. D. Huang and S. M. Chao, Lambda Set Selection in Roth-Karp Decomposition for LUT-Based FPGA Technology.Mapping, *Proc. ACM/IEEE Design Automation Conference*, pp. 65-69, San Francisco, 1995.

[19] H. Shin and C. Kim, Performance-Oriented Technology Mapping for LUT-Based FPGAs, *IEEE Trans. on VLSI Systems*, Vol. 3, No. 2, pp. 323-327, 1995.

[20] U. Schlichtmann, Boolean Matching and Disjoint Decomposition for FPGA Technology Mapping. *Proc. International Workshop on Logic and Architecture Synthesis*, pp. 83-102, Grenoble, 1993.

[21] C. Scholl and P. Molitor, Communication Based FPGA Synthesis for Multiple-Output Boolean Functions, *Proc. of the Asia and South Pacific Design Automation Conference,* pp. 279-287, Makuhari, Chiba, Japan, 1995.

[22] H. Selvaraj, T. Łuba, M. Nowicka and B. Bignall, Multiple-Valued Decomposition and its Applications in Data Compression and Technology Mapping. *Proc. International Conference on Computational Intelligence and Multimedia Applications*, pp. 42-48, Gold Coast, Australia, 1997.

[23] T. Stanion and C. Sechen, A method for Finding Good Ashenhurst Decomposition and Its Application to FPGA Synthesis, *32 Design Automation Conference*, pp. 60-64, San Francisco, 1995.

[24] M. Sysło, N. Deo and J. Kowalik, *Discrete Optimization Algorithms*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1993.

[25] W. Wan, M. A. Perkowski, A New Approach to the Decomposition of Incompletely Specified Multi-Output Function Based on Graph Coloring and Local Transformations and its Application to FPGA Mapping, *Proc. European Design Automation Conference*, pp. 230-235, Hamburg, 1992.

[26] S. Yang and M. J. Ciesielski, PLA Decomposition with Generalized Decoders. *IEEE International Conference on Computer-Aided Design*, pp. 312-315, Santa Clara, 1989.