Multi-output Functional Decomposition with Exploitation of Don't Cares

Christoph Scholl

Institute of Computer Science, Albert-Ludwigs-University
79110 Freiburg im Breisgau, Germany
email: scholl@informatik.uni-freiburg.de

Abstract

Functional decomposition is an important technique in logic synthesis, especially for the design of lookup table based FPGA *architectures.*

We present a method for functional decomposition with a novel concept for the exploitation of don't cares thereby combining two essential goals: the minimization of the number of decomposition functions in the current decomposition step and the extraction of common subfunctions for multi-output Boolean functions.

The exploitation of symmetries of Boolean functions plays an important role in our algorithm as a means to minimize the number of decomposition functions not only for the current decomposition step but also for the (recursive) decomposition algorithm as a whole.

Experimental results prove the effectiveness of our approach.

1 Introduction

Functional decomposition was introduced by Ashenhurst [1], Curtis [4], Roth and Karp [17, 8]. During recent years functional decomposition attracted a lot of interest especially in connection with the synthesis of lookup table based FPGA architectures (see e.g. [15, 9, 10, 11, 19, 25, 22]). Efficient functional decomposition methods based on Binary Decision Diagrams (BDDs) were proposed; there were improvements on the basic decomposition techniques with respect to the extraction of common sublogic in the decomposition of multi-output Boolean functions [12, 25, 22]. In [3, 2] a BDD based method was presented which computes extensions of incompletely specified single-output functions with a minimal number of decomposition functions in the current decomposition step.

The exploitation of don't cares is an important step in functional decomposition even for completely specified functions, since decomposition is applied recursively and at least at higher levels of the recursion we usually obtain incompletely specified functions.

In our decomposition procedure we use an improved method to exploit don't cares which does not only minimize communication complexity in the current decomposition step but has also an effect on the (recursive) decomposition of the decomposition functions in our procedure. We assign don't cares to maximize the number of symmetries to achieve this `global' effect (see also [21]).

In addition we developed a new method for don't care assignment with respect to the computation of common decomposition functions of multi-output functions. This method is intended to increase the potential of sharing decomposition functions between several single-output functions.

Finally we apply the method of Chang and Marek-Sadowska [3, 2] to minimize the number of decomposition functions for single-output functions in the current decomposition step.

The crucial point in our don't care assignment concept is the fact that all steps in this procedure are compatible in the sense that one step does not destroy the results of the previous one.

Recently, Eckl et al. [5] developed a method for multi-output functional decomposition exploiting don't cares independently of us. However their approach has only a local effect to the current decomposition step and does not take into account global effects of the don't care assignment.

The paper is organized as follows: In Section 2 we define basic notations. In Section 3 we briefly review our method to compute common decomposition functions for multi-output functions [22]. The role of symmetries in logic synthesis is discussed in Section 4 and our concept for don't care assignment is given in Section 5. In Section 6 experimental results are given and Section 7 concludes the paper.

2 Preliminaries

We restate some well–known definitions for decomposition of Boolean functions.

A single-output Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$

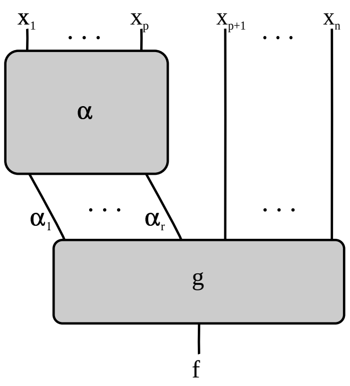

Figure 1: Decomposition of $f : \{0,1\}^n \rightarrow \{0,1\}$ with respect to $\{x_1, \ldots, x_p\}$.

with input variables $x_1, \ldots, x_n$ is decomposed with respect to a subset $\{x_1, \ldots, x_p\}$ $(1 < p < n)$ of the input variables according to Figure 1. The functions $\alpha_i : \{0,1\}^p \rightarrow \{0,1\}$ $(1 \leq i \leq r)$ are called *decomposition functions* and the function $g : \{0,1\}^{r+q} \rightarrow \{0,1\}$ is called *composition function*. $\{x_1, \ldots, x_p\}$ is called *bound set* and $\{x_{p+1}, \ldots, x_n\}$ is called *free set*. f is said to be decomposable with respect to $\{x_1, \ldots, x_p\}$, if there is a decomposition with $r < p$ decomposition functions.

If the Boolean function is to be realized by an FPGA with n_{LUT}-input lookup tables and if $p \leq n_{LUT}$, $\alpha = (\alpha_1, \ldots, \alpha_r)$ can be realized by r lookup tables (similarly for g). If the number of inputs of α or g is still too large, decomposition has to be applied recursively to α and g.

Given the subset $\{x_1, \ldots, x_p\}$ of the input variables, the minimum number r of decomposition functions has to be computed. To do so the notion of *equivalent* bound set vertices is used. Two bound set vertices $\epsilon^{(1)}$ and $\epsilon^{(2)} \in \{0,1\}^p$ are called *equivalent* ($\epsilon^{(1)} \equiv \epsilon^{(2)}$), iff $\forall \delta \in \{0,1\}^{n-p}$: $f(\epsilon^{(1)}, \delta) = f(\epsilon^{(2)}, \delta)$. For completely specified functions equivalence of bound set vertices forms an equivalence relation, which partitions $\{0,1\}^p$ into equivalence classes. The number of different equivalence classes of $\{0,1\}^p/_\equiv$ is denoted by $nec(f, \{x_1, \ldots, x_p\})$. It is easy to see that a decomposition with decomposition function $\alpha = (\alpha_1, \ldots, \alpha_r)$ exists, iff $\forall \epsilon^{(1)}, \epsilon^{(2)} \in \{0,1\}^p$:

$$\epsilon^{(1)} \not\equiv \epsilon^{(2)} \Longrightarrow \alpha(\epsilon^{(1)}) \neq \alpha(\epsilon^{(2)}).$$

Thus, the minimum number of decomposition functions in the decomposition of f with respect to $\{x_1, \ldots, x_p\}$ is $\lceil \log_2(nec(f, \{x_1, \ldots, x_p\})) \rceil$.

It is well-known that the number of equivalence classes $nec(f, \{x_1, \ldots, x_p\})$ can be easily determined based on BDD representations, if the variables of the bound set $\{x_1, \ldots, x_p\}$ are located before the variables of the free set in the BDD variable order [10].

3 Multi-output Decomposition

In this section we will briefly review our method to decompose multi-output functions [22].

Given an m-output function $f = (f_1, \ldots, f_m) : \{0,1\}^n \rightarrow \{0,1\}^m$ we have to compute m decompositions with respect to bound set $\{x_1, \ldots, x_p\}$

$$f_i(x_1, \ldots, x_n) = g_i(\alpha_1^{(i)}(x_1, \ldots, x_p), \ldots, \alpha_{r_i}^{(i)}(x_1, \ldots, x_p), x_{p+1}, \ldots, x_n).$$

Unlike [11] we choose decompositions with minimal numbers of decomposition functions $r_i = \lceil \log_2(nec(f_i, \{x_1, \ldots, x_p\})) \rceil$ for each f_i $(1 \leq i \leq m)$, since our goal is to minimize the number of inputs of decomposition functions *and* composition functions, such that they can be realized by one LUT as soon as possible. (In [11] the *total* number of decomposition functions for $f_1, \ldots, f_m$ is minimized, but the number of inputs of g_i can be (much) larger than $n - p + \lceil \log_2(nec(f_i, \{x_1, \ldots, x_p\})) \rceil$.)

Thus, under the condition

$$r_i = \lceil \log_2(nec(f_i, \{x_1, \ldots, x_p\})) \rceil$$

we minimize the number of decomposition functions $|\bigcup_{i=1}^m \{\alpha_1^{(i)}, \ldots, \alpha_{r_i}^{(i)}\}|$.

This is done by a BDD based computation of common decomposition functions for subsets $\{f_{i_1}, \ldots, f_{i_k}\}$ [22]. The computation is significantly sped up by a restriction of the search space to the so-called *strict* decomposition functions. A decomposition function $\alpha_j^{(i)}$ is called *strict*, iff $\forall \epsilon^{(1)}, \epsilon^{(2)} \in \{0,1\}^p$:

$$\epsilon^{(1)} \equiv_i \epsilon^{(2)} \Longrightarrow \alpha_j^{(i)}(\epsilon^{(1)}) = \alpha_j^{(i)}(\epsilon^{(2)}).$$

($\equiv_i$ is the equivalence relation for f_i.)

However, the restriction to strict decomposition functions has not only this `technical' reason. It can be shown, that strict decomposition functions preserve structural properties of the functions f_i, which is crucial for our decomposition algorithm (see Section 4).

4 Symmetries

Logic synthesis can take advantage of symmetries of Boolean functions. In the decomposition approach, e.g., symmetries in the set of bound variables lead to smaller numbers of decomposition functions. For the extreme case of f being symmetric in the bound set $\{x_1, \ldots, x_p\}$ (i.e. f does not change, if any pair of variables from $\{x_1, \ldots, x_p\}$ is exchanged), it is easy to see that the number of decomposition functions needed in a decomposition with respect to $\{x_1, \ldots, x_p\}$ can not be larger than $\lceil \log(p+1) \rceil$. Analogous results hold when f is symmetric in not *all* pairs of variables from $\{x_1, \ldots, x_p\}$.

Strict decomposition functions have the property that they preserve symmetry properties: If f is symmetric in a pair x_i, x_j of variables from $\{x_1, \dots, x_p\}$, then all strict decomposition functions of f are symmetric in x_i and x_j. This fact is also true for more general types of symmetry like G–symmetries in the bound set [1] [7].

5 Incompletely Specified Functions

If in a decomposition

$$f(x_1, \dots, x_n) = \\ g(\alpha_1(x_1, \dots, x_p), \dots, \alpha_r(x_1, \dots, x_p), x_{p+1}, \dots, x_n)$$

some codes $(a_1, \dots, a_r)$ do not occur in the image of α, then $(a_1, \dots, a_r, \delta_1, \dots, \delta_{n-p})$ is a don't care of g for all $(\delta_1, \dots, \delta_{n-p}) \in \{0,1\}^{n-p}$. Because of that we have to deal with incompletely specified functions during the recursive decomposition procedure, even if we start with completely specified functions.

Our concept to assign values to don't cares consists of three steps:

Step 1 First of all, we assign don't cares in order to obtain as many symmetries as possible for the resulting function. As mentioned in the previous section this will lead to a reduction of the number of decomposition functions. There is not only an effect on the current decomposition step, but also on later (recursive) decompositions.

Don't care assignment to obtain symmetries can be done *before* the selection of a bound set for the decomposition. Then we use symmetric sifting [13, 16] to determine a starting point of our gradual improvement heuristic to search for good candidates for bound sets. During the search for a good bound set we exchange groups of symmetric variables.

The difficulty in the don't care assignment consists of the fact, that an assignment to obtain symmetry in a pair (x_i, x_j) can destroy symmetry in another pair (x_j, x_k). Theory and an algorithm to solve this don't care assignment problem heuristically are given in more detail in [21]. We consider both nonequivalence symmetry and equivalence symmetry [6].

Step 2 In general, the functions to be decomposed still have don't cares after step 1. These remaining don't cares are assigned with respect to logic sharing. The don't care assignment takes into account that for a multi-output function $f = (f_1, \dots, f_m)$ the functions $f_1, \dots, f_m$ are decomposed with computation of common decomposition functions. We minimize not only the number of decomposition functions for the single-output functions f_i, but also the total number of decomposition functions to obtain as much logic sharing as possible.

To achieve this goal we propose to minimize a lower bound on the total number of decomposition functions for $f_1, \dots, f_m$.

If $f_1, \dots, f_m$ are completely specified, a lower bound on the total number of decomposition functions is computed as follows:
Now we call two bound set vertices $\epsilon^{(1)}$ and $\epsilon^{(2)} \in \{0,1\}^p$ equivalent ($\epsilon^{(1)} \equiv \epsilon^{(2)}$), iff $\forall \delta \in \{0,1\}^{n-p}$:

$$f_i(\epsilon^{(1)}, \delta) = f_i(\epsilon^{(2)}, \delta) \textit{ for all } 1 \le i \le m.$$

If $nec(f, \{x_1, \dots, x_p\})$ is the number of different equivalence classes according to *this* definition of equivalence, a lower bound on the total number of decomposition functions for $f_1, \dots, f_m$ is given by $\lceil \log_2(nec(f, \{x_1, \dots, x_p\})) \rceil$. Let $\{\alpha_1^{(i)}, \dots, \alpha_{r_i}^{(i)}\}$ be the set of decomposition functions for f_i ($r_i = \log_2(nec(f_i, \{x_1, \dots, x_p\}))\rceil$ as defined in Section 2) and let $r = |\bigcup_{i=1}^m \{\alpha_1^{(i)}, \dots, \alpha_{r_i}^{(i)}\}|$ be the total number of decomposition functions in the decomposition of $f = (f_1, \dots, f_m)$. Then we have

$$\lceil \log_2(nec(f, \{x_1, \dots, x_p\})) \rceil \le r \le \\ \le \sum_{i=1}^m \lceil \log_2(nec(f_i, \{x_1, \dots, x_p\})) \rceil = \sum_{i=1}^m r_i.$$

Thus, $\lceil \log_2(nec(f, \{x_1, \dots, x_p\})) \rceil$ is not only a lower bound on $\sum_{i=1}^m r_i$, but it also provides an estimation to the extent we can expect to find common decomposition functions in the decomposition of $f_1, \dots, f_m$. If $\lceil \log_2(nec(f, \{x_1, \dots, x_p\})) \rceil$ is small and $\sum_{i=1}^m r_i$ is large, then we can hope that there is a large potential to share decomposition functions in the decomposition of the single-output functions f_i.

If $f_1, \dots, f_m$ are incompletely specified functions, we assign values to don't cares to compute extensions $f_1', \dots, f_m'$ of $f_1, \dots, f_m$, such that for $f' = (f_1', \dots, f_m')$ $nec(f', \{x_1, \dots, x_p\})$ is minimal. This is done in order to minimize the lower bound $\lceil \log_2(nec(f', \{x_1, \dots, x_p\})) \rceil$ on the total number of decomposition functions for $f_1', \dots, f_m'$.

For incompletely specified functions we have to distinguish between *equivalent* and *compatible* bound set vertices:

Two bound set vertices $\epsilon^{(1)}$ and $\epsilon^{(2)}$ are called *equivalent* ($\epsilon^{(1)} \equiv \epsilon^{(2)}$), iff $\forall 1 \le i \le m$ and $\forall \delta \in \{0,1\}^{n-p}$:
1. $(\epsilon^{(1)}, \delta)$ *and* $(\epsilon^{(2)}, \delta)$ are in the don't care set of f_i
or
2. $f_i(\epsilon^{(1)}, \delta) = f_i(\epsilon^{(2)}, \delta)$.

Two bound set vertices $\epsilon^{(1)}$ and $\epsilon^{(2)}$ are called *compatible* ($\epsilon^{(1)} \sim \epsilon^{(2)}$), iff $\forall 1 \le i \le m$ and $\forall \delta \in \{0,1\}^{n-p}$:

[1] G–symmetries in the bound set $\{x_1, \dots, x_p\}$ consist of all possible combinations of exchanges and negations of variables from $\{x_1, \dots, x_p\}$. Various types of symmetries can be expressed as G–symmetries: Equivalence symmetry [6] in x_i and x_j, e.g., means that f does not change under application of the following sequence: negation of x_i, exchange of x_i and x_j, negation of x_i.

1. $(\epsilon^{(1)}, \delta)$ *or* $(\epsilon^{(2)}, \delta)$ are in the don't care set of f_i
or
2. $f_i(\epsilon^{(1)}, \delta) = f_i(\epsilon^{(2)}, \delta)$.

Thus, if $\epsilon^{(1)}$ and $\epsilon^{(2)}$ are compatible wrt. f, then there is an extension f' of f, such that $\epsilon^{(1)}$ and $\epsilon^{(2)}$ are equivalent, i.e., if $\epsilon^{(1)}$ and $\epsilon^{(2)}$ are compatible wrt. f, then they can be made equivalent by don't care assignments.

To find an extension f' of f with a minimal $nec(f', \{x_1, \ldots, x_p\})$ we have to assign values to don't cares such that many *compatible* bound set vertices are made *equivalent*.

This optimization problem can be reduced to a clique cover problem for a graph G (or equivalently to a coloring problem for the inverse graph $\overline{G}$).

The number of nodes of this graph G is equal to the number of equivalence classes $nec(f, \{x_1, \ldots, x_p\}) := |\{0,1\}^p/_\equiv|$ for the equivalence relation $\equiv$ of f as defined above.

G can be easily computed based on BDD representations of the incompletely specified functions f_i [20].

For reasons of efficiency – in contrast to [5] – we do not compute and represent *all* possible solutions of coloring problems with $nec(f, \{x_1, \ldots, x_p\})$ nodes.

Step 3 Finally we exploit remaining don't cares after step 2 to further minimize the number of decomposition functions for single-output functions f_i using the method of Chang and Marek-Sadowska [3].

We can prove that the don't care assignment of step 3 can not increase the lower bound from step 2. Moreover we can prove that the procedure does not destroy symmetries, if each group of symmetric variables is completely contained in the bound set or in the free set [20].

6 Experimental Results

6.1 Arithmetic Functions

First of all we demonstrate that our automatic logic synthesis tool is able to produce competitive designs even for arithmetic functions which were already studied intensively using human intelligence.

Applied to adders of various operand lengths our tool automatically produces realizations which are very similar to the well-known conditional-sum adder [23]. Figure 2 shows the example of a two-input gate realization of an 8-bit adder generated by our tool. Differences in details even lead to a smaller number of gates for our realization (in the example of Figure 2, 49 two-input gates compared to 90 two-input gates for the conditional-sum adder.)

We also applied our synthesis tool to partial multipliers, i.e. functions $pm_n : \{0,1\}^{n^2} \rightarrow \{0,1\}^{2n}$ where the inputs are given by the bits of the n partial products[2] and the outputs are given by the $2n$ product bits.

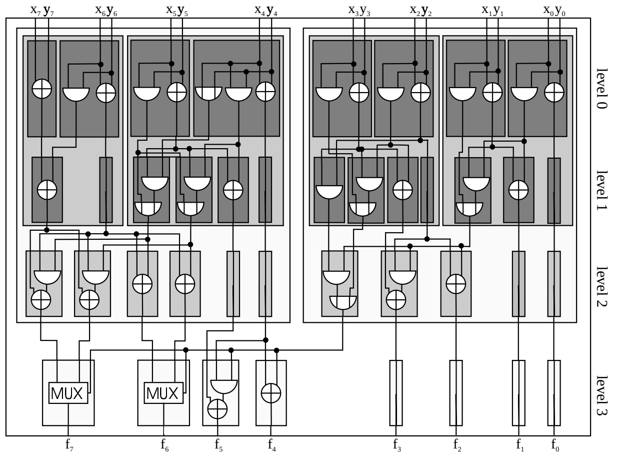

Figure 2: Automatically generated circuit for a 8-bit adder.

Figure 3 shows the result for pm_4. The result can be interpreted as a new multiplier scheme with a `column-wise' addition of the bits of the multiplication matrix. Note that the don't care assignment concept from Section 5 is essential for these results. A realization without this don't care assignment leads to a circuit with 75% more gates for pm_4.

A generalization of the principle to various operand lengths leads to a multiplier with $8\frac{1}{3} \cdot n^2 + O(n \log^2 n)$ two-input gates and depth $5.13 \cdot \log n + O(\log^\star n \log \log n)$ (compared to $10n^2 - 20n$ gates for the Wallace tree multiplier [24] with depth $5 \log n - 5$).[3]

6.2 Benchmark Circuits

We applied the decomposition procedure described above to several MCNC and ISCAS benchmarks to compute FPGA realizations for Xilinx XC3000 device (where the number of inputs of the lookup tables is $n_{LUT} = 5$).

We compared the numbers of CLBs for our new algorithm *mulop-dc* to the results of *mulopII* [22], where we didn't use any don't care assignment procedure[4].

The results of Table 1 show a considerable reduction of CLB counts for our new algorithm. There are reductions of CLB counts of up to 35% for alu2 and the overall reduction is more than 10%. Note that the benchmark functions are all completely specified functions and don't cares occur only at higher levels of the recursion. For this reason it

[2] i.e. conjunctions $p_{i,j} = a_i b_j$ of bits of the operands $(a_1, \ldots, a_n)$ and $(b_1, \ldots, b_n)$

[3] $\log^\star n := \min\{m \mid \log^{(m)}(n) \leq 1\}$ with $\log^{(0)}(n) := n$ and $\log^{(i)}(n) := \log(\log^{(i-1)}(n))$ for $i \in \mathbf{N}$

[4] All don't cares were assigned to 0.

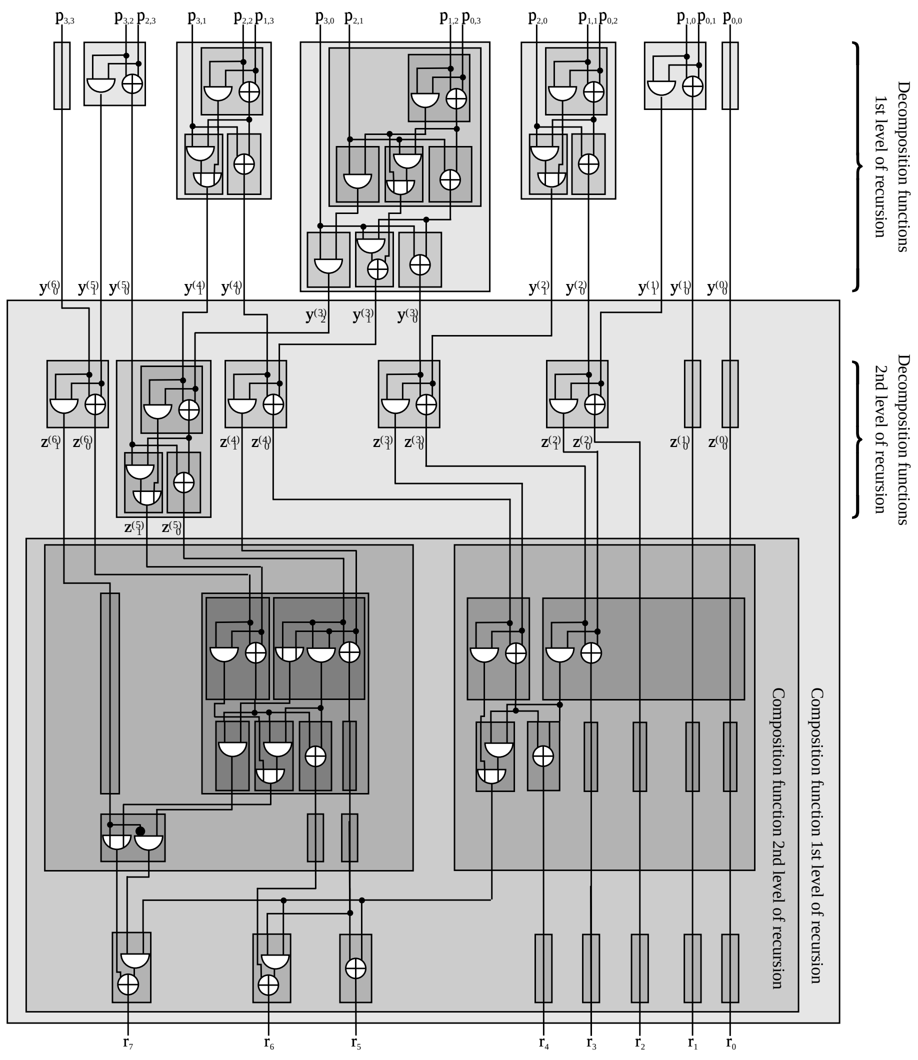
$p_{3,3}$ $p_{3,2}$ $p_{2,3}$ $p_{3,1}$ $p_{2,2}$ $p_{1,3}$ $p_{3,0}$ $p_{2,1}$ $p_{1,2}$ $p_{0,3}$ $p_{2,0}$ $p_{1,1}$ $p_{0,2}$ $p_{1,0}$ $p_{0,1}$ $p_{0,0}$
Decomposition functions 1st level of recursion
$y_0^{(6)}$ $y_1^{(5)}$ $y_0^{(5)}$ $y_1^{(4)}$ $y_0^{(4)}$ $y_2^{(3)}$ $y_1^{(3)}$ $y_0^{(3)}$ $y_1^{(2)}$ $y_0^{(2)}$ $y_1^{(1)}$ $y_0^{(1)}$ $y_0^{(0)}$
Decomposition functions 2nd level of recursion
$z_1^{(6)}$ $z_0^{(6)}$ $z_1^{(4)}$ $z_0^{(4)}$ $z_1^{(3)}$ $z_0^{(3)}$ $z_1^{(2)}$ $z_0^{(2)}$ $z_0^{(1)}$ $z_0^{(0)}$
$z_1^{(5)}$ $z_0^{(5)}$
Composition function 1st level of recursion
Composition function 2nd level of recursion
r_7 r_6 r_5 r_4 r_3 r_2 r_1 r_0

Figure 3: Automatically generated circuit for a partial 4-bit multiplier.

is clear that improvements can be obtained only for larger benchmarks.

Finally, Table 2 shows a comparison between our tool *mulop-dc*, *FGMap* [9], *mis-pga(new)* [15, 18] and *IMODEC* [25] proving the advantages of our procedure.

7 Conclusions

We presented a method for functional decomposition which combines the exploitation of don't cares with the exploitation of symmetries of Boolean functions and the extraction of common subfunctions for multi-output Boolean functions.

Applied to FPGA synthesis, our methods to exploit don't cares lead to considerable reductions of CLB counts even for completely specified benchmark functions, since incompletely specified functions arise during the recursive application of the decomposition procedure.

References

[1] R.L. Ashenhurst. The decomposition of switching functions. In *Int'l Symp. on Theory Switching Funct.*, pages 74–116, 1959.

[2] S. Chang, D. Cheng, and M. Marek-Sadowska. Minimizing ROBDD size of incompletely specified multiple output functions. In *European Design & Test Conf.*, pages 620–624, 1994.

[3] S.-C. Chang and M. Marek-Sadowska. BDD representation of incompletely specified functions. In *Int'l Workshop on Logic Synth.*, pages P6c:1–6, 1993.

Circuit	i	o	Number of CLBs *mulopII*	Number of CLBs *mulop-dc*
5xp1	7	10	9	9
9sym	9	1	7	7
alu2	10	6	51	33
apex7	49	37	45	41
b9	41	21	30	28
C499	41	32	65	50
C880	60	26	87	71
clip	9	5	14	13
count	35	16	26	26
duke2	22	29	114	108
e64	65	65	55	55
f51m	8	8	8	8
misex1	8	7	9	8
misex2	25	18	24	24
rd73	7	3	5	5
rd84	8	4	8	8
rot	135	107	146	135
sao2	10	4	20	18
vg2	25	8	18	18
z4ml	7	4	4	4
$\sum$ (total)			745	669

Table 1: Comparison of CLB counts for XC3000 device without and with don' t care exploitation.

Circuit	Number of CLBs mulop-dcII[4]	FGMap	mis-pga(new)	IMODEC
5xp1	9	15	13	9
9sym	7	7	7	7
alu2	33	53	96	46
apex7	39	47	43	41
b9	28	27	32	-
C499	50	49	66	50
C880	71	74	72	81
clip	13	20	23	12
count	24	24	30	26
duke2	101	178	94	122
e64	50	55	56	55
f51m	8	11	15	8
misex1	8	8	9	9
misex2	23	21	25	21
rd73	5	7	5	5
rd84	8	12	9	8
rot	123	194	143	127
sao2	18	27	28	17
vg2	18	23	18	19
z4ml	4	5	4	4
$\sum$ (subtot.)	612	830	756	667
$\sum$ (total)	640	857	788	-

Table 2: Comparison of CLB counts for XC3000 device between *mulop-dcII*, *FGMap*, *mis-pga(new)* and *IMODEC*

[4] H.A. Curtis. A generalized tree circuit. *Journal of the ACM*, 8:484–496, 1961.

[5] K. Eckl, C. Legl, and B. Wurth. An implicit approach to functional decomposition of incompletely specified boolean functions. In *Int'l Workshop on Logic Synth.*, 1997.

[6] C.R. Edwards and S.L. Hurst. A digital synthesis procedure under function symmetries and mapping methods. *IEEE Trans. on Comp.*, 27:985–997, 1978.

[7] G. Hotz. *Schaltkreistheorie.* Walter de Gruyter, 1974.

[8] R.M. Karp. Functional decomposition and switching circuit design. *J. Soc. Indust. Appl. Math.*, 11(2):291–335, 1963.

[9] Y.-T. Lai, K.-R. Pan, M. Pedram, and S. Sastry. FGMap: A technology mapping algorithm for look-up table type FPGAs based on function graphs. In *Int'l Workshop on Logic Synth.*, pages 9b1–9b4, 1993.

[10] Y.-T. Lai, M. Pedram, and S.B.K. Vrudhula. BDD based decomposition of logic functions with application to FPGA synthesis. In *Design Automation Conf.*, pages 642–647, 1993.

[11] Y.-T. Lai, M. Pedram, and S.B.K. Vrudhula. EVBDD-based algorithms for integer linear programming, spectral transformation, and function decomposition. *IEEE Trans. on CAD*, 13(8), 1994.

[12] P. Molitor and C. Scholl. Communication Based Multilevel Synthesis for Multi-Output Boolean Functions. In *Great Lakes Symp. VLSI*, pages 101–104, 1994.

[13] D. Möller, P. Molitor, and R. Drechsler. Symmetry based variable ordering for ROBDDs. *IFIP Workshop on Logic and Architecture Synthesis, Grenoble*, pages 47–53, 1994.

[14] R. Murgai, Y. Nishizaki, N. Shenoy, R.K. Brayton, and A. Sangiovanni-Vincentelli. Logic synthesis for programmable gate arrays. In *Design Automation Conf.*, pages 620–625, 1990.

[15] R. Murgai, N. Shenoy, R.K. Brayton, and A. Sangiovanni-Vincentelli. Improved logic synthesis algorithms for table look up architectures. In *Int'l Conf. on CAD*, pages 564–567, 1991.

[16] S. Panda, F. Somenzi, and B.F. Plessier. Symmetry detection and dynamic variable ordering of decision diagrams. In *Int'l Conf. on CAD*, pages 628–631, 1994.

[17] J.P. Roth and R.M. Karp. Minimization over Boolean graphs. *IBM J. Res. and Develop.*, 6(2):227–238, 1962.

[18] A. Sangiovanni-Vincentelli, A.E. Gamal, and J. Rose. Synthesis methods for field programmable gate arrays. *Proc. of the IEEE*, 81:1057–1083, 1993.

[19] T. Sasao. *Logic Synthesis and Optimization.* Kluwer Academic Publisher, 1993.

[20] C. Scholl. *Mehrstufige Logiksynthese unter Ausnutzung funktionaler Eigenschaften.* PhD thesis, Universität des Saarlandes, 1996.

[21] C. Scholl, S. Melchior, G. Hotz, and P. Molitor. Minimizing ROBDD sizes of incompletely specified functions by exploiting strong symmetries. In *European Design & Test Conf.*, pages 229–234, 1997.

[22] C. Scholl and P. Molitor. Communication based FPGA synthesis for multi-output Boolean functions. In *ASP Design Automation Conf.*, pages 279–287, 1995.

[23] J. Slansky. Conditional-sum addition logic. *IEEE Trans. on Electronic Comp.*, 9:226–231, 1960.

[24] C.S. Wallace. A suggestion for a fast multiplier. *IEEE Trans. on Comp.*, 13:14–17, 1964.

[25] B. Wurth, K. Eckl, and K. Antreich. Functional multiple-output decomposition: Theory and implicit algorithm. In *Design Automation Conf.*, pages 54–59, 1995.

[4] CLB counts of *mulop-dcII* in Table 2 differ from CLB counts of *mulop-dc* in Table 1 because of a changed procedure to merge LUTs into CLBs (the merging problem is formulated as a maximum cardinality matching problem, as proposed in [14]).