DECOMPOS: An Integrated System for Functional Decomposition

Tsutomu Sasao and Munehiro Matsuura
Department of Computer Science and Electronics
Kyushu Institute of Technology
Iizuka 820-8502, Japan

Abstract

This paper presents a system for disjoint decompositions of logic functions with many inputs. It is a combination of three different methods:

1) Disjoint decompositions with a few bound set variables;
2) Disjoint bi-decompositions; and
3) Decompositions using Jacobian.

1) and 2) are quick, but find only limited classes of decompositions, while 3) finds all disjoint decompositions by spending more time. We show the results of decompositions for more than four thousand functions. We also define a new class of functions: Completely bi-decomposable functions. Experimental results show that many practical logic functions have disjoint decompositions and some are completely bi-decomposable functions.

I Introduction

In general, an n-variable function f requires about $2^n/n$ gates [23]. Suppose that the function f can be decomposed into two networks as shown in Fig. 1.1. Let the numbers of inputs for the network H and G be n_1 and n_2+1, respectively, where $n_1+n_2=n$. Then, H and G can be realized by the networks with at most $2^{n_1}/n_1$ and $2^{n_2+1}/(n_2+1)$ gates, respectively. When n is large, $2^n/n >> 2^{n_1}/n_1 + 2^{n_2+1}/(n_2+1)$. This implies that the decomposed realization requires many fewer gates than the non-decomposed one. Such a design method is a **functional decomposition**. Functional decomposition developed by Ashenhurst [1] has been used for design of contact networks [7], PLAs (programmable logic arrays) [17, 8, 5, 22, 26], FPGAs [18, 16, 21, 6], and multi-level networks [2].

In this paper, we consider the functional decomposition of logic functions with many inputs. We assume that n, the number of inputs, can be more than 30. Direct application of the classical decomposition method has two problems. The first problem is the computation time and the memory requirement. The number of different decompositions is 2^n and the size of the decomposition table is 2^n. Thus, the straightforward implementation of classical method is impractical for the functions with many variables. The second problem is the usefulness of the functional decompositions. Statistically speaking, almost all n-variable functions are undecomposable when n is large [9, 18].

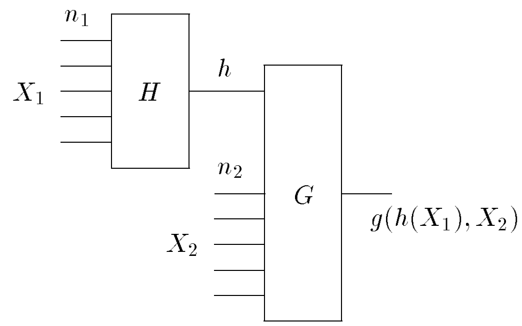

Figure 1.1: Functional decomposition $f(X_1, X_2) = g(h(X_1), X_2)$.

In this paper, we will solve the first problem. It is not a single method, but a combination of three different decomposition methods.

1) Decompose the function into smaller pieces by first finding the decompositions that are easy to detect: Disjoint decompositions with a few bound set variables, and bi-decompositions.
2) For each piece of function which are not decomposed by the above method, find the remaining disjoint decompositions by spending more time. For this purpose, we use an algorithm using Jacobian.

Then, we will demonstrate the usefulness of the functional decompositions by using benchmark functions. Experimental results show that many benchmark functions have disjoint decompositions.

Table 2.1: Example logic function.

x_1	x_2	x_3	x_4	f
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

II Functional Decomposition Theory

2.1 Definitions and Basic Properties

We assume that $f(X)$ is a completely specified non-degenerate function.

Definition 2.1 *Let $X = (x_1, x_2, \ldots, x_n)$ be input variables. The set of variables in X is denoted by $\{X\}$. $(X_1, X_2, \ldots, X_r)$ is a* **partition** *of X when $\{X_1\} \cup \{X_2\} \cup \ldots \cup \{X_r\} = \{X\}$ and $\{X_i\} \cap \{X_j\} = \phi$ $(i \neq j)$. The number of variables in $\{X_i\}$ is denoted by $|X_i| = n_i$.*

Definition 2.2 *Function $f(X)$ has a* **disjoint decomposition** *if f is represented as $f(X) = g(h(X_1), X_2)$. If $1 < |X_1| < n$, then this decomposition is* **non-trivial**, *and f is* **decomposable**. *The variables in X_1 and X_2 are* **bound variables** *and* **free variables**, *respectively.*

Definition 2.3 *Let $f(X)$ be a function, and (X_1, X_2) be a partition of X. Let $n_1 = |X_1|$ and $n_2 = |X_2|$. The* **decomposition table** *of f has 2^{n_1} columns and 2^{n_2} rows, each column has distinct binary label of n_1 bits, each row has distinct binary label of n_2 bits, and the each entry of the table represents the corresponding value of f.*

Example 2.1 *Let $f(X)$ be the function shown in Table 2.1, and (X_1, X_2) be a partition of X, where $X_1 = (x_1, x_2)$ and $X_2 = (x_3, x_4)$. The corresponding decomposition table is shown in Fig. 2.1.*

Definition 2.4 *The number of different column patterns in the decomposition table is the* **column multiplicity** *of the decomposition table and is denoted by μ. The number of different row patterns in the decomposition table is the* **row multiplicity** *of the decomposition table and denoted by ν.*

x_3x_4 \ x_1x_2	00	01	10	11
00	1	0	1	1
01	1	1	1	1
10	0	0	0	0
11	0	1	0	0

Figure 2.1: Decomposition table.

Lemma 2.1 *Let (X_1, X_2) be a partition of X. In the decomposition table for $f(X)$, $\mu \leq 2^\nu$ and $\nu \leq 2^\mu$.*

Corollary 2.1 *$\lceil log_2\mu \rceil \leq \nu \leq 2^\mu$, where $\lceil a \rceil$ denotes the smallest integer not smaller than a.*

Theorem 2.1 *$f(X)$ has the decomposition of the form*

$$f(X) = g(h(X_1), X_2), \qquad (2.1)$$

iff the column multiplicity μ of the decomposition table is $\mu \leq 2$.

Theorem 2.2 *A function $f(X)$ has a non-trivial functional decomposition $f(X) = \lambda(\delta(X_2), X_1)$ iff the column multiplicity μ of the decomposition table (X_1, X_2) is at most four, and there exists non-trivial column ψ, and no column other than ψ, $\bar{\psi}$, 0 (constant zero function), and 1 (constant one function) appear in the table.*

Theorem 2.2 shows that the decompositions $f(X) = g(h(X_1), X_2)$ and $f(X) = \lambda(\delta(X_2), X_1)$ can be detected simultaneously.

2.2 Complex Disjoint Decomposition [1, 7, 10]

Definition 2.5 *Let $(X_1, X_2, \ldots, X_r)$ be a partition of X. The decomposition of the form $f(X) = g(h_1(X_1), h_2(X_2), \ldots, h_r(X_r))$ or $f(X) = g(h_1(X_1), h_2(X_2), \ldots, h_{r-1}(X_{r-1}), X_r)$ is a* **multiple disjoint decomposition**. *The decomposition of the form $f(X) = g(h(\lambda(X_1), X_2), X_3)$ is an* **iterative disjoint decomposition**. *Combinations of these forms such as $f(X) = g(h(\lambda(X_1), X_2), \delta(X_3), X_4)$ is a* **complex disjoint decomposition**.

Lemma 2.2 *Let $f(X)$ have two disjoint decompositions:*

$$f(X) = g(\lambda(X_1), X_2, X_3) = h(X_1, \delta(X_2), X_3).$$

Then, $f(X)$ has a multiple disjoint decomposition:

$$f(X) = \gamma(\lambda(X_1), \delta(X_2), X_3).$$

Lemma 2.3 *Let $f(X)$ have two disjoint decompositions:*

$$f(X) = g(h(X_1, X_2), X_3) = \lambda(\delta(X_1), X_2, X_3).$$

Then, $f(X)$ has an iterative disjoint decomposition:

$$f(X) = g(\gamma(\delta(X_1), X_2), X_3),$$

where $\gamma(\delta(X_1), X_2) = h(X_1, X_2)$.

Lemmas 2.2 and 2.3 show that the complex decomposition can be found recursively. The computation time to find the decomposition of the form $f(X) = g(h(X_1), X_2)$ is proportional to $C(n, n_1)$, where $|X_1| = n_1$. Thus, it is more efficient to find the decomposition with small $|X_1|$ first. If such a decomposition exists, then we will try to find the decomposition of a function $g(u, X_2)$ with $n_2 + 1$ variables.

III Fast Decomposition Methods

3.1 Functional Decomposition with a Few Bound Variables

The size of a decomposition table for an n-variable function is 2^n. Thus, the straightforward method to find a decomposition is impractical for a function with many variables. A method to find decompositions by using ROBDDs (reduced ordered binary decision diagrams) is known [18, 12]. However, this method requires much computation time, since BDD becomes large during the permutation of the input variables.

To compute the column multiplicity of a decomposition, the following algorithm is faster than the method of [18, 12] when $|X_1| = n_1 = 2$. In this case, the number of different partitions to consider is $n(n-1)/2$. Thus, we can detect such decompositions very quickly.

Algorithm 3.1 *(Decomposition with $n_1 = 2$) For a multiple-output function, decompose the function by outputs. Ignore the redundant variables, and decompose each functions recursively.*

For $1 \le i < j \le n$. Let

$$\begin{aligned} & \qquad\qquad\quad\;\; i \qquad\;\; j \\ f_{00} &= f(x_1, \ldots, 0, \ldots, 0, \ldots, x_n), \\ f_{01} &= f(x_1, \ldots, 0, \ldots, 1, \ldots, x_n), \\ f_{10} &= f(x_1, \ldots, 1, \ldots, 0, \ldots, x_n), \\ f_{11} &= f(x_1, \ldots, 1, \ldots, 1, \ldots, x_n). \end{aligned}$$

If the number of the different functions is two, then this function has a decomposition $f = g(h(X_1), X_2)$, where $X_1 = (x_i, x_j)$.

3.2 Bi-decomposition [20]

Definition 3.1 *If $f(X)$ is represented as $f(X) = g(h_1(X_1), h_2(X_2))$, then $f(X)$ has a* **bi-decomposition**.

Bi-decompositions are easy to find from ISOPs (irredundant sum-of-product expressions) and PPRMs (positive polarity Reed-Muller expressions) [20]. ISOPs and PPRMs can be easily generated from BDDs and FDDs, respectively. Experimental results show that many practical logic functions have bi-decompositions [13, 20].

Algorithm 3.2 *For a multiple-output function, decompose the function by outputs. Decompose each function recursively.*

1. *Obtain an ISOP for f. Find the OR type bi-decomposition.*
2. *Obtain an ISOP for $\bar{f}$. Find the AND type bi-decomposition.*
3. *Obtain the PPRM for f. Find the EXOR type bi-decomposition.*

Definition 3.2 *A* **completely bi-decomposable function** *(CBF) is recursively defined as follows:*

1. *Constant functions are CBFs.*
2. *A single variable function is a CBF.*
3. *If $f(X)$ is a CBF, then $\bar{f}(X)$ is also a CBF.*
4. *If $g(X)$ and $h(Y)$ are CBFs, then $f(g(X), h(Y))$ is also a CBF, where X and Y have no common variables, and f is an arbitrary function of two variables.*

If a function is CBF, then f is realized by a tree network with two-input gates.

3.3 Decomposition Method using Jacobian

The decomposition methods in the previous section are very fast. However, they find only limited classes of decompositions. In this section, we will present an algorithm to find all the disjoint decompositions [24, 25]. This method quickly rejects non-decomposable functions. However, it requires a more computation time for the functions with decompositions.

Definition 3.3 *Let f and g be functions, and x_1 and x_2 be variables. The* **Jacobian** *is*

$$J(f, g/x_1, x_2) = \frac{df}{dx_1}\frac{dg}{dx_2} \oplus \frac{df}{dx_2}\frac{dg}{dx_1},$$

where $\frac{df}{dx}$ is a Boolean difference of f with respect to x.

Theorem 3.1 *Let x_i, x_j, and x_k be variables in X and*

$$J(f, \frac{df}{dx_k}/x_i, x_j) \neq 0.$$

If $f(X)$ has a decomposition $f(X) = g(h(X_1), X_2)$ and $\{x_i, x_j\} \subseteq \{X_1\}$, then $x_k \in \{X_1\}$.

Theorem 3.1 is used to reduce the number of candidate bound sets X_1. The computation of all possible Jacobians for $f(X)$ will show the bounds set that cannot produce a decomposition. Thus, only the remaining bound sets must be tested by using Theorem 2.1.

Definition 3.4 *A* **bound set graph** *of a function $f(X)$ is defined as follows:*

1) *It has n nodes. Each node corresponds to a variable in X.*
2) *The edge between nodes x_i and x_j has weight W_{ij}, where*

$$W_{ij} = \left\{ x_k | J(f, \frac{df}{dx_k}/x_i, x_j) \neq 0 \right\}.$$

Definition 3.5 *If $x_i \in W_{ij}$, then any bound set containing x_i and x_j also contains x_k. Thus, any such bound set must also contain W_{ik} and W_{jk}. The process to modify the weight of the bound graph by these conditions is the* **augmentation of the bound set graph**.

Definition 3.6 *Let $E_{ij} = \{x_i x_j\} \cup W_{ij}$. In the augmented bound set graph, if $E_{ij} = \{x_1, x_2, \ldots, x_n\}$, then this bound set is trivial, and the edge $\{x_i, x_j\}$ is deleted from the augmented bound set graph. If $E_{ij} = E_{km}$, then delete the edge $\{k, m\}$. This process to delete trivial bound set is the* **reduction of the bound set graph**.

Algorithm 3.3 *(Decomposition using Jacobian)*

a) *Construct the bound set graph.*
b) *Augment the bound set graph.*
c) *Reduce the bound set graph.*
d) *Derive the candidate sets of bound set.*
e) *Check the decomposability by using Theorem 2.1.*

IV Decomposition System

Algorithms 3.1 and 3.2 are quick, but find only limited classes of decompositions, while Algorithm 3.3 finds all disjoint decompositions by spending more time. Thus, the best strategy is as follows: First, find the decompositions $f(X_1, X_2) = g(h(X_1), X_2)$, where $|X_1| = 2$ by using Algorithm 3.1. Next, find bi-decompositions by using Algorithm 3.2. Finally, find remaining disjoint decompositions by using Algorithm 3.3.

Algorithm 4.1

1. *Find decompositions $f(X_1, X_2) = g(h(X_1), X_2)$, where $|X_1| = 2$ (Algorithm 3.1).*
2. *Find bi-decompositions (Algorithm 3.2).*
3. *Find decompositions using Jacobian (Algorithm 3.3).*

V Experimental Results

To investigate the usefulness of the strategy, we applied decomposition algorithms to about two hundred benchmark functions. Note that most benchmark functions have multiple-output. We decomposed each output separately.

5.1 Decompositions with $n_1 = 2$ (Algorithm 3.1)

1) For the most benchmark functions we tried, Algorithm 3.1 finished computation in reasonable time. It found decompositions for 3516 functions out of 4678 functions.

2) Table 5.1 shows the results for selected functions. In this table, *In* denotes the number of input variables. *Out* denotes the number of output functions. The columns headed by $n_1 = 2$ denote the results of Algorithm 3.1. The column headed by BLK denotes the numbers of blocks after decompositions. For example, apex6 was decomposed into 347 blocks. The column headed by MAX denotes the maximum numbers of inputs after decompositions. For example, the number of inputs for apex6 is 135, but each block depends on at most 22 variables after the application of Algorithm 3.1.

5.2 Bi-decompositions (Algorithm 3.2)

1) When Algorithm 3.2 was applied separately.

   It always obtained the solutions when the BDDs and the FDDs can be constructed. However, it took more computation time than Algorithm 3.1. 3027 functions out of 4338 functions had bi-decompositions. Furthermore 1460 out of 4338 functions are CBFs. Especially, all the outputs of the following benchmark functions are CBFs: mish, misj, rckl,

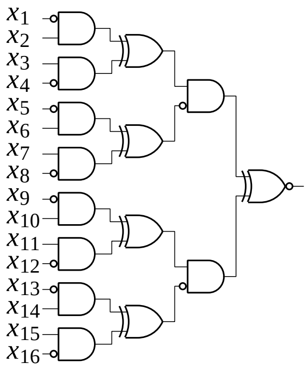

Figure 5.1: Exact minimal multi-level network for t481.

t481, e64, i3, i4, i5, cm42a. Note that t481 is a 16-variable single-output function. It is realized with fifteen 2-input gates. The algorithm obtained an exact minimum multi-level network shown in Fig. 5.1.

2) In Table 5.1, for the benchmark functions without * marks, Algorithm 3.2 were applied to the decomposed results of Algorithm 3.1. The columns headed by BDC denote the results for bi-decompositions. For example, in apex6, the number of blocks became 436 and each block depends on at most 21 variables.

   Benchmark functions with * marks were separately decomposed by Algorithm 3.2. In the current version of the program, when the SOPs for undecomposable blocks are too large, we have to use each decomposition algorithm separately, rather than using Algorithm 4.1.

   As shown in Table 5.1, disjoint bi-decompositions for a function with more than 200 inputs (des) were possible. However, for some functions, we cannot finish the computation due to memory overflow or excessive computation time. The dash (-) denotes that the computation was not finished.

5.3 Decompositions by using Jacobian (Algorithm 3.3)

The columns headed by JAC denote the results of Algorithm 3.3.

1) When Algorithm 3.3 was used separately.

   2975 functions out of 3802 functions had disjoint decompositions. For all the functions up to 25 inputs, the algorithm finished computations within 10 minutes. However, it could not finish the computation for some functions with more variables. It took longer time than Algorithms 3.1 and 3.2.

2) In Table 5.1, for the benchmark functions without * marks, Algorithm 3.3 were applied to the decomposed results of Algorithms 3.1 and 3.2. For example, in apex6, the number of blocks became 488 and each block depends on at most 14 variables. Algorithm 4.1 also successfully decomposed the functions that were unable to decompose by only using Algorithm 3.3. The column headed by DEC denotes the numbers of decomposed outputs. We assumed that the functions up to two variables are decomposable. For the functions with * marks, we may find more decompositions. For these functions, we stopped the algorithm due to excessive size of intermediate results or excessive computation time. Thus, the values may increase after the improvement of the algorithms.

   The column headed by CBF denotes the numbers of completely decomposable functions (CBFs). For example, apex6 has 99 outputs, and all the outputs have disjoint decompositions. Furthermore, 26 output functions are CBFs.

5.4 Comparison with [2]

Bertacco and Damiani presented a fast method to find disjoint decompositions [2]. Although their method is very fast, their method overlooked some decompositions. In fact, our system found more decompositions for C3540, C880, apex1, apex4, apex5, apex7,e64, frc2, k2, pair, rot, vda, and x4. For example, their method found no decompositions for any outputs of C3540, however, Algorithm 3.2 found decompositions for 13 outputs.

VI Conclusions and Comments

In this paper, we presented a system to find disjoint decompositions. It successfully found decompositions for functions with more than 200 inputs. Experimental results show that

1) 3516 out of 4678 functions have decompositions with the form $f(X_1, X_2) = g(h(X_1), X_2)$, where $|X_1| = 2$.
2) 3027 out of 4338 functions have bi-decompositons.
3) 1460 out of 4338 functions are completely bi-decomposable.

Butler has derived the number of n-variable CBFs [4]. He showed that even for moderate n, the fraction of CBFs is extremely small. For example, when

Table 5.1: Results of decompositions.

Name	In	Out	$n_1 = 2$		BDC		JAC		DEC	CBF
			BLK	MAX	BLK	MAX	BLK	MAX		
C3540*	50	22	54	50	–	–	–	–	13	4
C432*	36	7	23	36	23	36	–	–	1	1
C880*	60	26	127	41	92	45	–	–	26	17
accpla*	50	69	595	34	689	32	–	–	69	12
alu4	14	8	14	14	15	14	15	14	4	3
apex1*	45	45	248	33	260	33	–	–	43	12
apex2	39	3	36	16	37	15	37	15	3	2
apex3*	54	50	196	42	211	42	–	–	39	18
apex4	9	19	19	9	23	9	23	9	5	1
apex5	117	88	792	15	870	14	870	14	88	9
apex6	135	99	347	22	436	21	488	14	99	26
apex7	49	37	223	16	261	14	268	9	37	23
b3	32	20	159	28	177	27	183	27	20	6
b4	33	23	101	14	115	14	120	14	23	8
b9	41	21	78	9	88	8	88	8	21	8
cm42a	4	10	30	2	30	2	30	2	10	10
cm85a	11	3	12	9	16	7	20	3	3	1
count	35	16	152	4	168	3	168	3	16	0
des	256	245	1130	15	1186	14	1374	14	245	4
e64	65	65	2081	2	2081	2	2081	2	65	65
ex4	128	28	46	16	60	15	60	15	28	14
exep	30	63	762	15	767	15	767	15	63	57
frg2	143	139	1038	18	1130	17	1227	17	139	40
i1	25	16	46	4	47	3	47	3	16	15
i2	201	1	181	21	187	6	187	6	1	0
i3	132	6	126	2	126	2	126	2	6	6
i4	192	6	186	2	186	2	186	2	6	6
i5	133	66	606	2	606	2	606	2	66	66
i6	138	67	68	5	69	5	69	5	1	0
i7	199	67	72	6	72	6	72	6	3	3
i8	133	81	131	17	131	17	137	17	18	0
i9	88	63	63	13	63	13	63	13	0	0
ibm	48	17	34	16	34	16	34	16	8	0
in3	35	29	182	25	188	25	195	20	27	8
in4	32	20	172	28	191	27	197	27	20	5
in6	33	23	109	14	123	14	128	14	23	8
jbp	36	57	326	11	343	11	352	10	57	31
k2*	45	45	248	33	260	33	–	–	42	12
misex2	25	18	103	8	107	7	107	7	17	12
misg	56	23	43	12	44	11	44	11	23	22
mish	94	43	105	2	105	2	105	2	43	43
misj	35	14	49	2	49	2	49	2	14	14
pair	173	137	1448	30	1547	28	1557	28	137	33
rckl	32	7	216	2	216	2	216	2	7	7
rot*	135	107	508	45	–	–	–	–	104	57
signet	39	8	32	31	32	31	32	31	6	4
soar	83	94	492	14	560	11	560	11	89	47
t481	16	1	15	2	15	2	15	2	1	1
ti	47	72	385	22	431	21	452	21	72	21
vda	17	39	111	17	113	17	113	17	29	7
x1	51	35	234	18	252	17	252	17	35	19
x3	135	99	347	22	436	21	488	14	99	26
x4	94	71	386	9	422	8	443	8	71	35
x2dn	82	56	103	24	105	24	105	24	54	46
x6dn	39	5	28	29	33	28	36	28	5	0
x7dn	66	15	39	25	40	25	43	25	15	0
xparc	41	73	674	30	728	29	744	29	71	11

*: Algorithms were applied separately
BDC: Bi-decomposition
$n_1 = 2$: Decompositions with $n_1 = 2$
JAC: Decompositions using Jacobian
BLK: Number of blocks
MAX: Maximum number of variables for blocks
DEC: Number of decomposed outputs
CBF: Number of completely bi-decomposable functions

$n = 8$, only 10^{-56} percent of the n-variable functions are CBFs. 1460 CBFs out of 4338 functions imply that MCNC benchmark functions have very strong functional properties. We are very surprised with this results.

As far as we know, this paper first demonstrated such decomposability of benchmark functions by extensive experiments. Currently, we are improving the algorithms to make them robust.

Acknowledgments

This work was supported in part by a Grant in Aid for Scientific Research of the Ministry of Education, Science, Culture and Sports of Japan. Mr. Jun-ichi Yamashita and Mr. Yusuke Yauchi developed prototypes for Algorithms 3.2 and 3.3, respectively, while they were students of KIT.

References

[1] R. L. Ashenhurst, "The decomposition of switching functions," *In Proceedings of an international symposium on the theory of switching*, pp. 74-116, April 1957.

[2] V. Bertacco and M. Damiani, "The disjunctive decomposition of logic functions," *Proc. ICCAD*, pp. 78-82, Nov. 1997.

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

[4] J. T. Butler, "On the number of functions realized by cascades and disjunctive networks," *IEEE Trans. Comput.*, Vol. C-24, pp. 681-690, July 1975.

[5] M. J. Ciesielski and Saeyang Yang, "PLADE: A two-stage PLA decomposition," *IEEE Trans. on CAD*, Vol. 11, No. 8, pp. 943-954, Aug. 1992.

[6] S-C. Chang, M. Marek-Sadowska, and T. Hwang, "Technology mapping for LUT FPGA's based on decomposition of binary decision diagrams," *IEEE Trans. on CAD*, Vol. CAD-15, No. 10, pp. 1226-1236, Oct. 1996.

[7] H. A. Curtis, *A New Approach to the Design of Switching Circuits*, D. Van Nostrand Co., Princeton, NJ, 1962.

[8] S. Devadas, A. Wang, A. R. Newton, and A. Sangiovanni-Vincentelli, "Boolean decomposition in multilevel logic optimization," *IEEE Journal of Solid-State Circuits*, Vol. 24, pp. 399-408, April 1989.

[9] M. A. Harrison, *Introduction to Switching and Automata Theory*, McGraw-Hill, 1965.

[10] F. J. Hill and G. R. Peterson, *Computer Aided Logic Design with Emphases on VLSI*, Wiley, 1993.

[11] U. Kebschull, E. Schubert, and W. Rosenstiel, "Multilevel logic synthesis based on functional decision diagrams," *EDAC 92*, pp. 43-47, 1992.

[12] Y-T. Lai, M. Pedram, and S. B. K. Vrudhula, "EVBDD-based algorithm for integer linear programming, spectral transformation, and functional decomposition," *IEEE Trans. CAD*, Vol. 13, No. 8, Aug. 1994, pp. 959-975.

[13] Y. Matsunaga, "An attempt to factor logic functions using exclusive-or decomposition," *SASIMI'96*, pp. 78-83, Nov. 1996.

[14] S. Minato, "Fast generation of prime-irredundant covers from binary decision diagrams," *IEICE Trans. Fundamentals*, Vol. E76-A, No. 6, pp. 976-973, June 1993.

[15] R. Murgai, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Optimum functional decomposition using encoding," *Design Automation Conference*, pp. 408-414, June 1994.

[16] R. Murgai, R. K. Brayton, and A. Sangiovanni-Vincentelli, *Logic Synthesis for Field-Programmable Gate Arrays*, Kluwer Academic Publishers, 1995.

[17] T. Sasao, "Application of multiple-valued logic to a serial decomposition of PLA's," *International Symposium on Multiple-Valued Logic*, Zangzou, China, pp. 264-271, May 1989.

[18] T. Sasao, "FPGA design by generalized functional decomposition," in (Sasao ed.) *Logic Synthesis and Optimization*, Kluwer Academic Publishers, 1993.

[19] T. Sasao and M. Fujita (ed.), *Representation of Discrete Functions*, Kluwer Academic Publishers, 1996.

[20] T. Sasao and J. T. Butler, "On bi-decompositions of logic functions," *ACM/IEEE International Workshop on Logic Synthesis*, Tahoe City, California, May 1997.

[21] H. Sawada, T. Suyama, and A. Nagoya, "Logic synthesis for look-up table based FPGAs using functional decomposition and support minimization," in *ICCAD*, pp. 353-359, Nov. 1995.

[22] C. Scholl and P. Molitor, "Communication based FPGA synthesis for multi-output Boolean functions," in *Asia and South Pacific Design Automation Conference*, pp. 279-287, Aug. 1995.

[23] C. E. Shannon, "The synthesis of two-terminal switching circuits," *Bell Syst. Tech. J.*, Vol. 28, No. 1, pp. 59-98, 1949.

[24] V. Y-S, Shen and A. C. Mckellar, "An algorithm for the disjunctive decomposition of switching functions," *IEEE Trans. on Comput.*, Vol. C-19, No. 3, pp. 239-248, March 1970.

[25] V. Y-S, Shen, A. C. Mckellar and P. Weiner, "A fast algorithm for the disjunctive decomposition of switching functions," *IEEE Trans. on Comput.*, Vol. C-20, No. 3, pp. 304-309, March 1971.

[26] B. Wurth, K. Eckl, and K. Anterich, "Functional multiple-output decomposition: Theory and implicit algorithm," in *Design Automation Conf.*, pp. 54-59, June 1995.

[27] S. Yang, "Logic synthesis and optimization benchmark user guide, version 3.0," *MCNC*, Jan. 1991.