Functional Decompositions Using an Automatic Test Pattern Generator and a Logic Simulator

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

Abstract— This paper presents a method to find disjoint decompositions using an automatic test pattern generator and a logic simulator. Since the method uses netlists rather than binary decision diagrams to represent logic functions, it can decompose larger networks. By using netlists, it efficiently finds decompositions of form $f(X_1, X_2) = g(h(X_1), X_2)$, where $|X_1| \leq k$ or $|X_2| \leq k$.

I Introduction

Functional decompositions are useful in multi-level logic synthesis. In functional decompositions, logic functions are represented by decomposition charts [1], binary decision diagrams (BDDs), sum-of-products expressions (SOPs) [8, 6], or Reed-Muller expressions [8]. Among these representations, BDDs are the most compact for many practical functions.

Recently, efficient methods to find all simple disjoint decompositions for the functions given by BDDs have been developed [2, 5]. Unfortunately, for some functions, BDDs are very large, while their circuits are not so large. For an example, the 16-bit multiplier has the BDD whose size is too large to store in a computer, while its netlist has less than 5000 lines.

In this paper, we will present a method to find disjoint decompositions for the functions represented by netlists. We use an automatic test pattern generator (ATPG) and a logic simulator as basic tools to find decompositions.

II Functional Decomposition

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 the variables in X is denoted by $\{X\}$. $X = (X_1, X_2, \ldots, X_r)$ is a* **partition** *of X when $\{X_1\} \cup \{X_2\} \cup \cdots \cup \{X_r\} = \{X\}$ and $\{X_i\} \cap \{X_j\} = \phi$ $(i \neq j)$. Especially, (X_1, X_2) is a* **bipartition**. *The number of variables in X_i is denoted by $|X_i|$.*

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.*

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.

Definition 2.3 *Let $f(X)$ be a function, and X be a partition of $X = (X_1, X_2)$. 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 corresponding entry of the table is the value of f.*

Example 2.1 *Fig. 2.1 shows an example of a decomposition table for a four-variable function, where $X = (X_1, X_2)$, $X_1 = (x_1, x_2)$, and $X_2 = (x_3, x_4)$.*

Definition 2.4 *The number of different column patterns in the decomposition table (X_1, X_2) is the* **column multiplicity** *and is denoted by $\mu(f : X_1, X_2)$. The number of different row patterns in the decomposition table (X_1, X_2) is the* **row multiplicity** *and is denoted by $\nu(f : X_1, X_2)$.*

Theorem 2.1 *A function $f(X)$ has a non-trivial functional decomposition $f(X) = g(h(X_1), X_2)$ iff $\mu(f : X_1, X_2) \leq 2$.*

Theorem 2.2 *A function $f(X)$ has a non-trivial functional decomposition $f(X) = g(h(X_1), X_2)$ iff $\nu(f : X_1, X_2) \leq 4$ and row patterns of the decomposition table represent constant 0, constant 1, a function h, or its complement $\bar{h}$.*

Example 2.2 *The column multiplicity of the decomposition table in Fig. 2.1 is two. The row multiplicity is four. Thus, this function has a decomposition* $f(X_1, X_2) = g(h(X_1), X_2)$, *where* $X_1 = (x_1, x_2)$ *and* $X_2 = (x_3, x_4)$. *The first column is identical to the third and fourth columns. The first row denotes function* $h = x_1 \vee \bar{x}_2$; *the second row denotes the constant* 1 *function; the third row denotes the constant* 0 *function; and the fourth row denotes function* $\bar{h} = \bar{x}_1 x_2$. *(End of Example)*

Therefore, to find decompositions by using Theorems 2.1 and 2.2, we need to test that

1) Two subfunctions are the same.
2) A subfunction is the constant 0.
3) A subfunction is the constant 1.
4) A subfunction is the complement of an another subfunction.

III ATPG

3.1 ATPG0 and ATPG1

Although an automatic test pattern generator (ATPG) is a software to generate test patterns, in this paper, the ATPG is used to find functional decompositions. To make the argument simple, we will formulate mathematical models of the ATPG.

The ATPG0 finds test vectors for logic networks, while the ATPG1 solves a decision problem.

Definition 3.1 *(ATPG0) Given a netlist for a logic function* $f(x_1, x_2, \ldots, x_n)$, *the ATPG0 finds an assignment* $(a_1, a_2, \ldots, a_n)$ *of input variables that satisfies the following condition:*

$$f(a_1, a_2, \ldots, \overset{i}{0}, \ldots, a_n) \neq f(a_1, a_2, \ldots, \overset{i}{1}, \ldots, a_n).$$

If there is no such assignment, the ATPG0 reports that "f *does not depend on* x_i*"* (x_i *is redundant).*

Definition 3.2 *(ATPG1) Given a netlist for a logic function* $f(x_1, x_2, \ldots, x_n)$, *the ATPG1 solves the following decision problem: Is there any assignment* $(a_1, a_2, \ldots, a_n)$ *of input variables that satisfies*

$$f(a_1, a_2, \ldots, \overset{i}{0}, \ldots, a_n) \neq f(a_1, a_2, \ldots, \overset{i}{1}, \ldots, a_n)?$$

If the answer is yes, then f *depends on* x_i. *Else,* f *does not depend on* x_i (x_i *is redundant).*

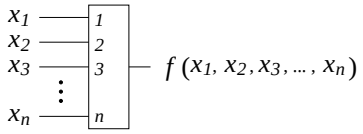

Figure 3.1: Original network.

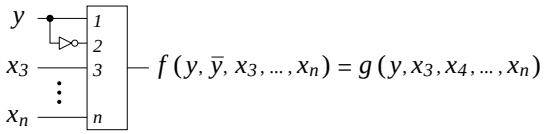

Figure 3.2: Modified network to check $f_{01} = f_{10}$.

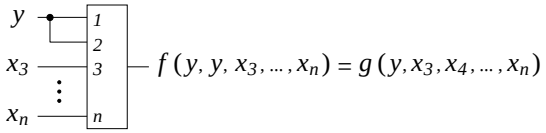

Figure 3.3: Modified network to check $f_{00} = f_{11}$.

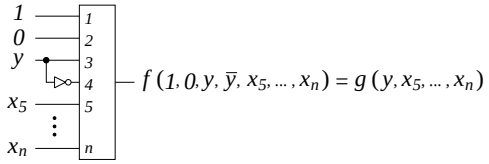

Figure 3.4: Modified network to check $f_{1001} = f_{1010}$.

3.2 Functional Check Using the ATPG1

In this part, we will show methods to check functional relations between subfunctions. These methods will be used to detect functional decompositions.

3.2.1 Equivalence Checking between Two Subfunctions

Let $f_{01} = f(0, 1, x_3, \ldots, x_n)$ and $f_{10} = f(1, 0, x_3, \ldots, x_n)$. If $f_{01} = f_{10}$, then f is partially symmetric with respect to x_1 and x_2. This property can be checked by using the ATPG1 as follows [7]:

By modifying the network in Fig. 3.1 to obtain Fig. 3.2, we can realize the function $g(y, x_3, x_4, \ldots, x_n) = f(y, \bar{y}, x_3, \ldots, x_n)$. If g does not depend on y, then $f(0, 1, x_3, \ldots, x_n) = f(1, 0, x_3, \ldots, x_n)$.

Similarly, by modifying Fig. 3.1 to obtain Fig. 3.3, we can realize $g(y, x_3, x_4, \ldots, x_n) = f(y, y, x_3, \ldots, x_n)$. In this case, if g does not depend on y, then $f(0, 0, x_3, \ldots, x_n) = f(1, 1, x_3, \ldots, x_n)$.

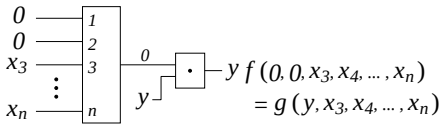

Figure 3.5: Checking for the constant 0.

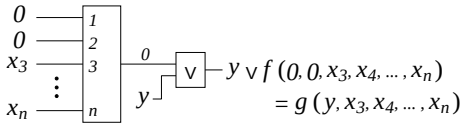

Figure 3.6: Checking for the constant 1.

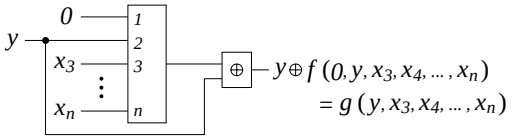

Figure 3.7: Checking for $f_{00} = \bar{f}_{01}$.

By appending constants and inverters, we can check the equivalence of arbitrary subfunctions. For example, in Fig. 3.4, we can check $f(1,0,1,0,x_5,\ldots,x_n) = f(1,0,0,1,x_5,\ldots,x_n)$ by redundancy check of y in $g(y,x_5,\ldots,x_n)$.

3.2.2 Checking for the Constant 0

To check whether $f_{00} = 0$ or not, we can modify Fig. 3.1 to obtain Fig. 3.5. This network realizes the function $g(y,x_3,x_4,\ldots,x_n) = yf(0,0,x_3,x_4,\ldots,x_n)$. If g does not depend on y, then $f(0,0,x_3,x_4,\ldots,x_n) = 0$.

3.2.3 Checking for the Constant 1

To check whether $f_{00} = 1$ or not, we can modify Fig. 3.1 to obtain Fig. 3.6. This network realizes the function $g(y,x_3,x_4,\ldots,x_n) = y \vee f(0,0,x_3,x_4,\ldots,x_n)$. If g does not depend on y, then $f(0,0,x_3,x_4,\ldots,x_n) = 1$.

3.2.4 Checking for $f_{00} = \bar{f}_{01}$

To check whether $f_{00} = \bar{f}_{01}$ or not, we can modify Fig. 3.1 to obtain Fig. 3.7. This network realizes the function $g(y,x_3,x_4,\ldots,x_n) = y \oplus f(0,y,x_3,x_4,\ldots,x_n)$. If g does not depend on y, then $f(0,0,x_3,x_4,\ldots,x_n) = \bar{f}(0,1,x_3,x_4,\ldots,x_n)$.

As shown in this section, we can find functional decompositions by using only ATPG1. Unfortunately, these methods are quite time consuming. In the next section, we will show faster methods.

IV Speedup Using A Logic Simulator

In this section, we will show a method to detect sets of bipartitions (X_1, X_2) for which f has no decomposition $f = g(h(X_1), X_2)$ by using a logic simulator.

4.1 Detection of Undecomposable Bipartitions Using a Logic Simulator

In this part, we will show methods to find bipartitions (X_1, X_2) for which f has no decomposition $f(X_1, X_2) = g(h(X_1), X_2)$.

Example 4.1 *By using Theorem 2.1, we can find the bipartitions (X_1, X_2) for which f has no decomposition $f = g(h(X_1), X_2)$. Assume that $X_1 = (x_1, x_2)$. Let*

$$\begin{aligned} f_{00} &= f(0,0,X_2), \\ f_{01} &= f(0,1,X_2), \\ f_{10} &= f(1,0,X_2), \text{ and} \\ f_{11} &= f(1,1,X_2). \end{aligned}$$

be subfunctions obtained by assigning the constants to x_1 and x_2. If they denote more than two different functions, then f does not have a decomposition for this bipartition. Let apply the random patterns to input X_2, and let $\boldsymbol{a}_{00}$, $\boldsymbol{a}_{01}$, $\boldsymbol{a}_{10}$, and $\boldsymbol{a}_{11}$ be the response vectors of the subfunctions f_{00}, f_{01}, f_{10}, and f_{11}, respectively. If there are more than two different response vectors, then f does not have the decomposition for this bipartition. (End of Example)

Example 4.2 *By using Theorem 2.2, we can find the bipartitions for which f has no decomposition $f = g(h(X_1), X_2)$. Assume that $X_2 = (x_{n-1}, x_n)$. Let*

$$\begin{aligned} f_{00} &= f(X_1,0,0), \\ f_{01} &= f(X_1,0,1), \\ f_{10} &= f(X_1,1,0), \text{ and} \\ f_{11} &= f(X_1,1,1) \end{aligned}$$

be subfunctions obtained by assigning the constants to x_{n-1} and x_n. If f_{00}, f_{01}, f_{10}, and f_{11} denote functions other than 0, 1, h, or $\bar{h}$, then f does not have a decomposition for this bipartition. Let apply the random patterns to input X_1, and let $\boldsymbol{a}_{00}$, $\boldsymbol{a}_{01}$, $\boldsymbol{a}_{10}$, and $\boldsymbol{a}_{11}$, be the response vectors of the subfunctions f_{00}, f_{01}, f_{10}, and f_{11}, respectively. Let $\boldsymbol{a}$ be a non-constant response vector. If one of the response vectors is a non-constant and denotes neither $\boldsymbol{a}$ nor $\bar{\boldsymbol{a}}$, then f does not have the decomposition for this bipartition. (End of Example)

V Experimental Results

Rather than developing dedicated programs for ATPG0 and ATPG1, we modified the program in [4] for functional decompositions. MCNC benchmark functions in BLIF [10] format were used as input data. Experimental results in [8] show that for the benchmark functions, most decompositions are of the form $f(X_1, X_2) = g(h(X_1), X_2)$, where $|X_1| \leq 2$ or $|X_2| \leq 2$. Thus, we used the following:

Algorithm 5.1 *(Decomposition using an ATPG and a Logic Simulator)*

1. *For each output function, do the followings:*
2. *Extract the netlist for the function.*
3. *Detect the decomposition which are easily found from the netlist.*
4. *By using the logic simulator, find the set of candidate bipartitions (X_1, X_2) having decompositions $g(h(X_1), X_2)$, where $|X_1| \leq 3$ or $|X_2| \leq 3$.*
5. *For each candidate bipartition, check if there is a decomposition by using the ATPG. If exist, reform the netlist and repeat this step.*

For the functions whose BDDs are relatively small, we can find simple disjoint decompositions easily [5, 8]. Thus, in this experiment, we considered the functions whose BDDs are very large, but their netlists are small enough to be stored in a computer.

One of such functions is an n-bit multiplier (*mlp n*).

			x_n	x_{n-1}	$\cdots$	x_3	x_2	x_1
		$x)$	y_n	y_{n-1}	$\cdots$	y_3	y_2	y_1
z_{2n}	z_{2n-1}	$\cdots$	z_n	z_{n-1}	$\cdots$	z_3	z_2	z_1

The *mlp n* has $2n$ inputs and $2n$ outputs. Note that z_1, z_2, z_3, and z_{2n} have decompositions:

$$\begin{aligned} z_1 &= x_1 y_1 \\ z_2 &= x_1 y_2 \oplus x_2 y_1 \\ z_3 &= x_2 y_2 \overline{x_1 y_1} \oplus x_3 y_1 \oplus x_1 y_3 \\ z_{2n} &= x_n y_n g(x_{n-1}, \ldots, x_1, y_{n-1}, \ldots, y_1). \end{aligned}$$

However, other outputs z_k $(k = 4, \ldots, 2n-1)$ of *mlp n* are undecomposable. Especially, the size of the BDD for z_n is exponential for any order of the input variables [3]. Note that the benchmark circuit c6288 represents the *mlp 16*, and its BDD is too large to built. Our system found the decompositions for z_1, z_2, z_3, and z_{2n}. Also, for $k = 4, \ldots, 2n-2$, our system successfully proved that they have no decomposition of the form $z_k = g(h(X_1), X_2)$, where $|X_1| \leq 3$ or $|X_2| \leq 3$.

VI Conclusions and Comments

In this paper, we presented a method to find disjoint decompositions for the functions given by netlists, where an ATPG and a logic simulator show equivalence and non-equivalence of subfunctions, respectively. Since our method uses netlist rather than BDDs, it can decompose functions whose BDDs are too large to store in a computer. The presented method can be extended to the case of k-decomposition [9] $f(X_1, X_2) = g(h_1(X_1), h_2(X_2), \ldots, h_k(X_1), X_2)$. It is possible to replace the ATPG-based engine with SAT-based engine.

Functional decompositions on netlists are more time-consuming than ones on BDDs. However, we have to consider the tasks for converting netlists into BDDs. In many cases, they are non-trivial tasks, and often we cannot build BDDs. Decompositions on netlists are also promising to find good ordering of the input variables to build BDDs.

Acknowledgements

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. Major parts of the system were developed by the following person: M. Kusumoto, N. Harada, and Y. Kurata. Mr. Matsuura edited a LaTeX files.

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," *ICCAD-97*, pp. 78-82, Nov. 1997.

[3] R. E. Bryant, "On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication," *IEEE Trans. on Comput.*, Vol. 40, No. 2, pp. 205-213, Feb. 1991.

[4] S. Kajihara, H. Shiba and K. Kinoshita, "Removal of redundancy in logic circuits under classification of undetectable faults," *Proc. 22nd Fault-Tolerant Computer Sympo.*, pp. 263-270, 1992.

[5] Y. Matsunaga, "An exact and efficient algorithm for disjunctive decomposition," *SASIMI'98*, pp. 44-50, Oct. 1998.

[6] S. Minato and G. De Micheli, "Finding all simple disjunctive decompositions using irredundant sum-of-products forms," *ICCAD-98*, pp. 111-117, Nov. 1998.

[7] I. Pomeranz and S. M. Reddy, "On determing symmetries in inputs of logic circuits," *IEEE TCAD*, Vol. 13, No. 11, pp. 1428-1433, Nov. 1994.

[8] T. Sasao and M. Matsuura, "DECOMPOS: An integrated system for functional decomposition," *1998 International Workshop on Logic Synthesis*, Lake Tahoe, June 1998.

[9] T. Sasao, "Totally undecomposable functions: applications to efficient multiple-valued decompositions," *IEEE International Symposium on Multiple-Valued Logic*, Freiburg, Germany, May 20-23, 1999 (to be published).

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