

Design Methods for Multi-Rail Cascades

Tsutomu Sasao

Department of Computer Science and Electronics, and

Center for Microelectronics Systems

Kyushu Institute of Technology

Iizuka 820-8502

Japan

email: sasao@cse.kyutech.ac.jp

Abstract

This paper surveys methods to represent logic functions by cascades. First, a design method for multi-rail cascades with redundant inputs is shown. It uses logic minimization of SOPs (sum-of-products expressions) or ESOPs (EXOR sum-of-products expressions) of multiple-valued inputs. Then, a design method for multi-rail cascades with irredundant inputs is shown. It uses functional decompositions using BDDs. In both cases, extensions to multiple-output functions are shown.

1 Introduction

Two of the most crucial problems in modern LSIs are long design time and short life cycles. A solution to these problems may be programmable logic devices (PLDs). They can reduce the hardware development time and cost drastically.

In this paper, we consider a realization of logic functions by PLDs. Various architectures exist for PLDs. Among them, random access memories (RAMs) and programmable logic arrays (PLAs) are simple. However, when the number of input variables n is large, the hardware needed to realize the function by using a single module becomes too large. Thus, field programmable logic arrays (FPGAs), and complex programmable logic devices (CPLDs) are often used. In FPGAs, both logic and interconnections are programmable. Logic is usually implemented using lookup tables (LUTs). In modern FPGAs, the area and delay time associated with interconnections are much larger than for logic. This makes the design of FPGAs fairly complex. This paper considers realizations of logic functions by cascades. The cascade is one of the simplest structure because it is regular. Since interconnections are limited to only the adjacent cells, area and delay for interconnections are minimum. Thus, multi-level cascades may have delay time significantly less than comparable FPGAs with fewer logic levels. This property is quite desirable in deep sub-micron LSI design [3, 13].

Cascade realizations of logic functions were considered in the 1960's [12]. An excellent survey can be found in [14]. Cascades can be classified as 1) single rail and multi-rail, and 2) irredundant and redundant. For example, Fig. 1.1 shows a single-rail cascade with irredundant inputs, Fig. 1.2 shows a two-rail cascade with irredundant inputs, and Fig. 1.3 shows a multi-rail cascade with redundant inputs.

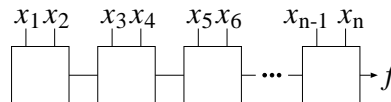


Figure 1.1: Single-rail cascade with irredundant inputs.

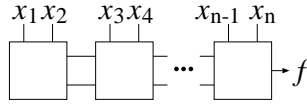


Figure 1.2: Two-rail cascade with irredundant inputs.

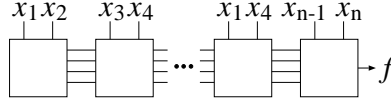


Figure 1.3: Multi-rail cascade with redundant input.

In a single-rail cascade, one wire connects adjacent cells, while in a multi-rail cascade, there are two or more wires. In an irredundant cascade, each variable is connected to only one logic block of the cascade, while in a redundant cascade, some variables are connected to two or more logic blocks.

Irredundant cascades can realize only a fraction of all functions [4, 5]. On the other hand, two-rail redundant cascades of three-input cells can represent any functions of n variables. This is shown in Section 2.

In this paper, we assume that a cell can realize any function of its input variables. Note that a cell is also called a *lookup table* (LUT), where the table's entry corresponds to the (binary) output tuple indexed by the (binary) input tuple. We will consider various methods to realize logic functions using as few cells as possible.

2 Multi-Rail Cascades With Redundant Inputs

2.1 Single-Output Functions

In this section, we show that two-rail cascades with redundant inputs can realize any logic function.

For cascades with redundant inputs, two design methods are known. One is based on decompositions of group functions, and the other is based on sum-of-products expressions (SOPs). The first method requires at most 2^n cells, while the second one requires at most $n2^n$ cells. Since the second one is easy to understand, we will show it.

Theorem 2.1 *An arbitrary n -variable logic function f can be realized as a redundant cascade by using three-input cells shown in Fig. 2.1.*

This can be explained as follows. An arbitrary logic function f can be represented as a canonical sum-

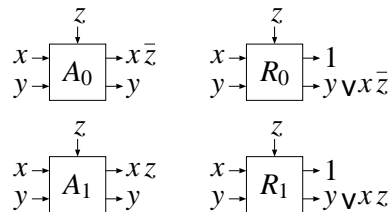


Figure 2.1: Cells for 2-rail cascade.

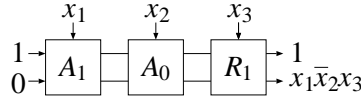


Figure 2.2: Realization of $x_1\bar{x}_2x_3$.

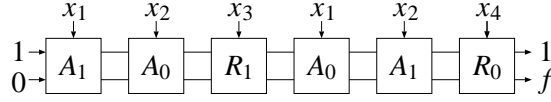


Figure 2.3: Realization of $x_1\bar{x}_2x_3 \vee \bar{x}_1x_2\bar{x}_4$.

of-products expression (SOP)

$$f(X) = \bigvee_{\vec{a} \in A} x_1^{a_1} x_2^{a_2} \cdots x_n^{a_n},$$

where $A \subseteq \{0, 1\}^n$. Suppose that we can use the cells shown in Fig. 2.1. We also assume that the constants can be used as inputs. Then, any product term can be realized as a cascade of cells in Fig. 2.1. For example, the product term $x_1\bar{x}_2x_3$ can be realized as shown in Fig. 2.2. Also, any SOP can be realized as a cascade of cells, as in Fig. 2.1. For example, $f = x_1\bar{x}_2x_3 \vee \bar{x}_1x_2\bar{x}_4$ can be realized as shown in Fig. 2.3. Note that A_0 and A_1 are used to make the AND function, while R_0 and R_1 are used to make both the AND and the OR functions.

Since any function f can be represented by an SOP with at most $n2^{n-1}$ literals, the number of cells is at most $n2^{n-1}$. The design method using Theorem 2.1 produces very long cascades,

A variation of the method is to use ESOPs (EXOR sum-of-products expressions) instead of SOPs. In this case, we use the cells shown in Fig. 2.4 instead of R_0 and R_1 cells. In many cases, ESOPs require fewer literals than SOPs [18, 20, 21].

Theorem 2.1 uses only three-input cells. However, in many applications, we can use cells with more than three inputs. In this case, we can extend Theorem 2.1 to multiple-valued inputs as follows:

Definition 2.1 Let $\vec{a} = (a_1, a_2, \dots, a_n)$ be a constant in B^n , where $B = \{0, 1\}$. $X^{\vec{a}}$ denotes a mapping $X^{\vec{a}} : B^n \rightarrow B$, such that $X^{\vec{a}} = 0$ if $X \neq \vec{a}$ and $X^{\vec{a}} = 1$ if $X = \vec{a}$.

Let $S \subseteq B^n$. X^S denotes the function

$$X^S = \bigvee_{\vec{a}_i \in S} X^{\vec{a}_i}.$$

X^S is a **literal**. A product of distinct literals is a **term**. When $X^S = 1$, X^S is a **trivial literal**, otherwise, it is **non-trivial literal**. A sum of terms is a **sum-of-products expression (SOP)**.

Lemma 2.1 [16, 17] Suppose that the input variables $X = (x_1, x_2, \dots, x_n)$ are partitioned into (X_1, X_2, \dots, X_r) . Then, an arbitrary function $f(X)$ can be represented as

$$f(X_1, X_2, \dots, X_r) = \bigvee_{(S_1, S_2, \dots, S_r)} X_1^{S_1} X_2^{S_2} \cdots X_r^{S_r}.$$

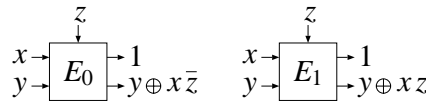


Figure 2.4: Cells for EXOR cascade.

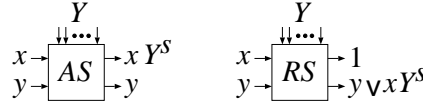


Figure 2.5: Cells for multi-valued SOPs.

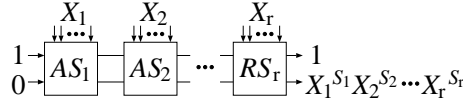


Figure 2.6: Two-rail cascade for multi-valued product.

The number of the products in a minimum SOP is at most $2^{n - \max_i n_i}$, where $S_i \subseteq B^{n_i}$ and $n_i = |X_i|$.

Theorem 2.2 *Let s_1 be the number of occurrences of (perhaps repeated) non-trivial literals in the SOP for $f(X_1, X_2, \dots, X_r)$. Then, f can be realized as a cascade with at most s_1 cells. (A literal that occurs m times contributes m to s_1 .)*

(Proof) Suppose that we can use the cells shown in Fig. 2.5. Any product $X_1^{S_1} X_2^{S_2} \dots X_r^{S_r}$ can be realized as shown in Fig. 2.6. Any SOP can be realized as a cascade of Fig. 2.6. (Q.E.D)

Theorem 2.3 *Suppose that $n = tr$, where t and r are integers. Then, f can be realized by a cascade with at most $r2^{n-t}$ cells.*

(Proof) In lemma 2.1, consider the case where each group of variables X_i consists of t binary variables. Then the number of products is at most 2^{n-t} . Also, each product has at most r non-trivial literals. So, the total number of non-trivial literals is at most $r2^{n-t}$. By Theorem 2.2, we can prove the theorem. (Q.E.D)

Example 2.1 *The benchmark function **sym12** has 12 inputs and 1 output. **sym12** is the symmetric function $S_{\{4,5,6,7,8\}}^{12}$. Its SOP has 495 products and 3960 non-trivial literals. Therefore, when $t = 1$, the cascade requires 3960 cells of 3 inputs. However, when $t = 4$, its SOP has only 15 products and 40 non-trivial literals. Thus, **sym12** can be realized as a cascade consisting 40 cells of 6 inputs. (End of Example)*

2.2 Multiple-Output Functions

Definition 2.2 *Consider a set of m functions f_j ($j = 0, 1, 2, \dots, m-1$) of n variables. The characteristic function for non-zero outputs (CFN) is a mapping $F : B^n \times M \rightarrow B$, where $M = \{0, 1, \dots, m-1\}$, and $F(\vec{a}, j) = 1$ iff $f_j(\vec{a}) = 1$, $j \in M$. The CFN has one auxiliary m -valued variable that represents the output part.*

Example 2.2 *Consider the set of logic functions:*

$$\begin{aligned} f_0 &= x_1 x_2 \vee x_3 x_4 \\ f_1 &= x_3 x_4 \vee x_5 x_6 \\ f_2 &= x_1 x_2 \vee x_5 x_6 \end{aligned}$$

The CFN is $F = x_1 x_2 z^{\{0,2\}} \vee x_3 x_4 z^{\{0,1\}} \vee x_5 x_6 z^{\{1,2\}}$, where z is the auxiliary variable representing the output. (End of Example)

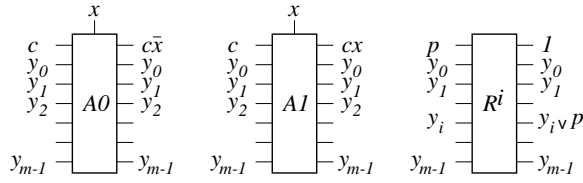


Figure 2.7: Cells for multiple-output functions.

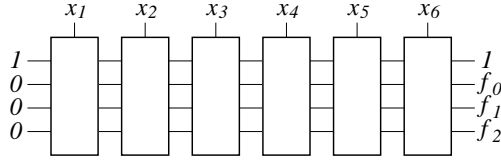


Figure 2.8: Four-rail cascade to realize three-output function.

A sum-of-products expression (SOP) for CFN corresponds to a programmable logic array (PLA) or an AND-OR two-level network. It is extensively used in logic synthesis [22]. This representation can be compressed by using multiple-valued logic minimizers such as MINI and ESPRESSO-MV.

Theorem 2.4 *An arbitrary n -variable m -output function can be realized by a redundant $(m + 1)$ -rail cascade. The number of cells is at most p , where p is the number of occurrences of (perhaps repeated) non-trivial input literals in an SOP for the CFN.*

(Proof) Consider the cells shown in Fig. 2.7. A_0 and A_1 are used to make a product term, while R^i is used to add the product p to y_i . When the product p is an implicant of f_i , then add p to y_i by using an R^i cell. Note that R^i cells can be merged with either an A_0 or an A_1 cell. A product p is an implicant of f_i if the SOP for CFN has the product pz^S , where S contains i . For each non-trivial input literal in an SOP for CFN, we use a cell. In this way, we can realize the cascade for the m -output function. (Q.E.D)

Example 2.3 *Consider the CFN of the three-output function: $F = x_1x_2z^{\{0,2\}} \vee x_3x_4z^{\{0,1\}} \vee x_5x_6z^{\{1,2\}}$. Note that the number of non-trivial input literals is 6. Thus, the three-output function can be realized by the irredundant 4-rail cascade shown in Fig. 2.8. Fig. 2.9 is the corresponding circuit, where \times denotes the OR.* (End of Example)

3 Multi-Rail Cascades with Irredundant Inputs

Since irredundant cascades realize a small fraction of all functions, there has not been much work on such circuits. However, recently an efficient design method has been developed [23], and successfully

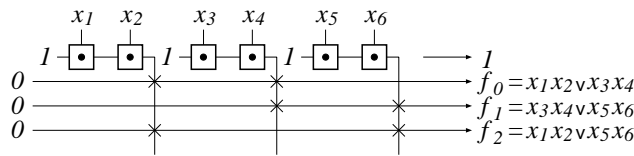


Figure 2.9: Circuit corresponding to Fig. 2.8.

Table 3.1: Decomposition chart.

		$X_1 = (x_1, x_2)$			
		0	0	1	1
$X_2 = (x_3, x_4)$	0	0	0	1	1
	0	1	1	1	1
	1	0	0	1	0
	1	1	0	0	0

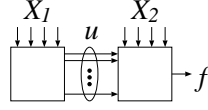


Figure 3.1: Functional Decomposition.

realized many benchmark functions by cascades with LUTs with at most 15 inputs. In this part, we show the outline of the method.

3.1 Single-Output Functions

Definition 3.1 Let $X = (x_1, x_2, \dots, x_n)$ be input variables. A set of variables X is denoted by $\{X\}$. (X_1, X_2) is a partition of X if $\{X_1\} \cup \{X_2\} = \{X\}$ and $\{X_1\} \cap \{X_2\} = \phi$. The number of variables in X is denoted by $|X|$.

Definition 3.2 For a logic function $f(X)$, let (X_1, X_2) be a partition of X . The **decomposition chart** of f , denoted by $M(f : X_1, X_2)$, is the matrix having 2^{n_1} columns and 2^{n_2} rows, where $n_1 = |X_1|$ and $n_2 = |X_2|$. In $M(f : X_1, X_2)$, each row and column has a label with a binary number, and the corresponding element denotes the truth value of f . The columns and rows have all possible patterns of n_1 bits and n_2 bits, respectively. X_1 is the **bound set**, and X_2 is the **free set**.

Example 3.1 Let $f(X)$ be a 4-variable function, and (X_1, X_2) be a partition of X , where $X_1 = (x_1, x_2)$ and $X_2 = (x_3, x_4)$. Table 3.1 is an example of a decomposition chart. (End of Example)

Definition 3.3 The number of different column patterns in a decomposition chart is its **column multiplicity**. The multiplicity of the decomposition chart $M(f : X_1, X_2)$ is denoted by $\mu(f : X_1, X_2)$.

Lemma 3.1 [1, 6] If the column multiplicity of the decomposition chart $M(f : X_1, X_2)$ is μ , then f can be represented as $f(X) = g(h_1(X_1), h_2(X_1), \dots, h_u(X_1), X_2)$, and f can be realized with the network structure shown in Fig. 3.1, where $u = \lceil \log_2 \mu \rceil$.

Lemma 3.2 [9, 19] Let (X_1, X_2) be a partition of X , and let the BDD of f be partitioned into two blocks as shown in Fig. 3.2. Suppose that λ nodes $q_1, q_2, \dots, q_\lambda$ in the lower block are adjacent to the upper block. Then, $\mu(f : X_1, X_2) = \lambda$.

Definition 3.4 Suppose that the ordering of variables in a BDD is (x_1, x_2, \dots, x_n) . Also, assume that $X_1 = (x_1, x_2, \dots, x_i)$ and $X_2 = (x_{i+1}, \dots, x_n)$. The **width of the BDD at level i** is $\mu_i = \mu(f : X_1, X_2)$. The **width of the BDD** is $\mu_{max} = \max\{\mu_i\}$.

Theorem 3.1 Consider the BDD for an n -variable logic function f . Let the width of the BDD be μ_{max} . If $u = \lceil \log_2 \mu_{max} \rceil \leq k - 1$, then f can be realized by a cascade of k -input cells shown in Fig. 3.3. Let

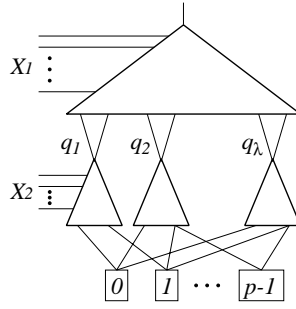


Figure 3.2: Functional decomposition using BDD.

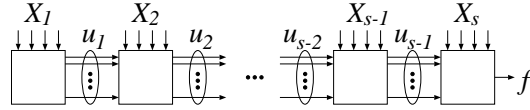


Figure 3.3: Cascade of s cells.

s_2 be the number of levels of the cascade, and N be the number of cells. Then, we have

$$\frac{n+u-2}{k-1} \leq s_2 \leq 1 + \frac{n-2}{k-u}, \text{ and}$$

$$\frac{n+u-2}{k-1} + u - 1 \leq N \leq \frac{n-2}{k-u}u + 1.$$

Example 3.2 For the benchmark function **sym12**, the widths of the BDD at levels 6 and 9 are 7 and 8, respectively. Note that $\lceil \log_2 7 \rceil = \lceil \log_2 8 \rceil = 3$. By Lemma 3.1, **sym12** can be realized with the cascade with only three cells as shown in Fig. 3.4, where there are three lines between cells. (End of Example)

Theorem 3.2 A function f is realized by an irredundant r -rail cascade shown in Fig. 3.5 iff the widths of a BDD for f at the decomposition points are at most 2^r .

(Proof) Suppose that f is realized in the cascade shown in Fig. 3.5. From the structure of the cascade, f can be decomposed as

$$f(X_1, X_2, \dots, X_r) = g(h(X_1, X_2, \dots, X_i), X_{i+1}, \dots, X_r).$$

Since, f can be realized by the structure of Fig. 3.6, and the number of lines between two blocks is r , the width of the BDD at the decomposition point is at most 2^r .

Suppose that the width of the BDD for f at the decomposition points are at most 2^r . Then, f can be realized by a network structure shown in Fig. 3.6. This means that f can be realized by a cascade shown in Fig. 3.5. (Q.E.D)

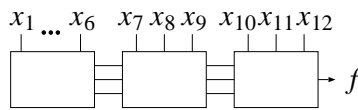


Figure 3.4: Cascade realization of **sym12**.

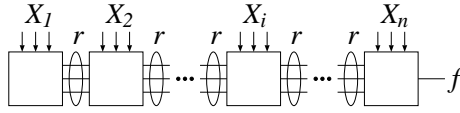


Figure 3.5: An r -rail cascade realization of f .

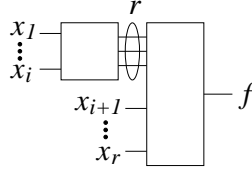


Figure 3.6: Decomposition of the function.

3.2 Multiple-Output Functions

3.2.1 Method using MTBDDs

In this case, the design method is the same as the single-output function except that an MTBDD is used instead of a BDD. When m is large, the size of the MTBDD tends to be very large [24], and the number of inputs of the cells becomes too large. In such a case, we partition the outputs into groups, where each group consists of at most (say) four outputs.

3.2.2 Method using BDD for ECFN

To represent an n -variable m -output function, we can use an encoded characteristic function for non-zero outputs (ECFN) [23]. It is a $(n + u)$ -variable single output logic function, and uses $u = \lceil \log_2 m \rceil$ auxiliary variables to represent the outputs. To evaluate a particular output f_j , we have to set the values of the auxiliary variables to specify f_j . Note that this method evaluates only one output at a time. For any multiple-output function, the BDD for the ECFN is never greater than the corresponding SBDD+, and in many cases, smaller than the SBDD+, where SBDD+ consists of an SBDD and a tree selecting the output [24].

When μ is large, the cascade realization needs cells with many inputs. This may preclude a cascade realization of the function with a reasonable size. For some functions, we cannot construct the BDDs due to excessive memory size. For example, the BDD for the ECFN of a 16-bit multiplier (C6288) is too large to construct. However, for many benchmark functions [2, 26], we can construct BDDs for ECFNs, and the widths of the BDDs are less than 1024. This means that the LUTs with 11 inputs are sufficient to realize many benchmark functions.

4 Concluding Remarks

In this paper, we surveyed cascade realizations logic functions. In the cascades, the interconnections are limited to only the adjacent cells. Thus, the delays for interconnections are minimum.

For a given function, unless we can find an irredundant cascade of a reasonable size, we have to resort to a redundant cascade. One of the most promising approaches is to find a BDD with a limited width, where variables can appear more than once. Such a BDD can be easily mapped into a cascade. Another approach is to use a group function [15] instead of a logic function.

Given a multi-rail cascade, we can reduce the number of rails by converting some intermediate variables into single-variables as shown in Fig. 4.1. This can be efficiently done by considering the output encoding [7, 11].

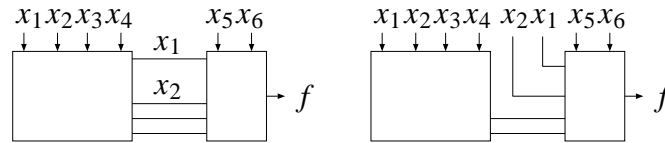


Figure 4.1: Reduction of connections between cells.

Other approach for cascade realizations can be found in [25].

Acknowledgements

This work is partly supported by grants of the Japan Society for the Promotion of Science (JSPS), and the Takeda Foundation. Prof. Jon T. Butler improved English presentation. Discussion with Dr. Alan Mishchenko also improved the paper.

References

- [1] R. L. Ashenurst, "The decomposition of switching functions," *In Proceedings of an International Symposium on the Theory of Switching*, pp. 74-116, April 1957.
- [2] F. Brglez and H. Fujiwara, "A neutral netlist of 10 combinational benchmark circuits and a target translator in FOTRAN," *In 1985 International Symposium on Circuits And Systems*, pp. 663-698, 1985.
- [3] R. K. Brayton, "The future of logic synthesis and verification," in Soha and Sasao (eds.), *Logic Synthesis and Verification*, Kluwer, Nov. 2001.
- [4] J. T. Butler, "On the number of functions realized by cascades and disjunctive networks," *IEEE Trans. Comput.*, Vol. C-24, No. 7, pp. 681-90, July 1975.
- [5] J. T. Butler, "Tandem networks of universal cells," *IEEE Trans. Comput.*, Vol. C-27, No. 9, pp. 785-799, Sept. 1978.
- [6] H. A. Curtis, *A New Approach to The Design of Switching Circuits*, D. Van Nostrand Co., Princeton, NJ, 1962.
- [7] H. Gouji, T. Sasao, and M. Matsuura, "On a method to reduce the number of LUTs in LUT cascades," (in Japanese) *Technical Report of IEICE*, VLD2001-99, Nov. 2001.
- [8] S. Hassoun and T. Sasao (eds.), *Logic Synthesis and Verification*, Kluwer Publishers, Nov. 2001.
- [9] 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, pp. 959-975, Aug. 1994.
- [10] S. Minato, "Minimum-width method of variable ordering for binary decision diagrams," *IEICE Trans. Fundamentals*, Vol. E75-A, No. 3, pp. 392-399, March 1992.
- [11] A. Mishchenko and T. Sasao, "Encoding of Boolean functions and its application to LUT cascade synthesis," *International Workshop on Logic Synthesis (IWLS-2002)*, June 4-7, New Orleans, Louisiana, 2002.
- [12] K. K. Mitra, "Cascade switching networks of two-input flexible cells," *IRE Trans. Electron. Comput.*, EC-11, pp. 136-143, 1962.

- [13] F. Mo and R. K. Brayton, "River PLAs: A regular circuit structure," *Design Automation Conference*, 14.1, New Orleans, June 2002.
- [14] A. Mukhopadhyay (ed.), *Recent Developments in Switching Theory*, Academic Press, New York, 1971.
- [15] T. Sasao and K. Kinoshita, "Conservative logic elements and their universality," *IEEE Trans. on Comput.*, Vol. C-28, No. 9, pp. 682-685, Sept. 1979.
- [16] T. Sasao, "Multiple-valued decomposition of generalized Boolean functions and the complexity of programmable logic arrays," *IEEE Trans. on Comput.*, Vol. C-30, No. 9, pp. 635-643, Sept. 1981.
- [17] T. Sasao, "Input variable assignment and output phase optimization of PLA's," *IEEE Trans. on Comput.*, Vol. C-33, No. 10, pp. 879-894, Oct. 1984.
- [18] T. Sasao, "EXMIN2: A simplification algorithm for exclusive-OR sum-of-products expressions," *IEEE TCAD*, Vol. 12, No. 5, pp. 621-632, May. 1993.
- [19] T. Sasao, "FPGA design by generalized functional decomposition," in (Sasao ed.) *Logic Synthesis and Optimization*, Kluwer Academic Publishers, 1993.
- [20] T. Sasao (ed.), *Logic Synthesis and Optimization*, Kluwer Academic Publishers, Jan. 1993.
- [21] T. Sasao and M. Fujita (eds.), *Representation of Discrete Functions*, Kluwer Academic Publishers, Apr. 1996.
- [22] T. Sasao, *Switching Theory for Logic Synthesis*, Kluwer Academic Publishers, Feb. 1999.
- [23] T. Sasao, M. Matsuura, and Y. Iguchi, "A cascade realization of multiple-output function for reconfigurable hardware," *International Workshop on Logic and Synthesis (IWLS01)*, Lake Tahoe, CA, June 12-15, 2001, pp. 225-230.
- [24] T. Sasao, Y. Iguchi, and M. Matsuura, "Comparison of decision diagrams for multiple-output logic functions," *International Workshop on Logic and Synthesis (IWLS2002)*, New Orleans, LA, June 4-7, 2002, pp. 379-384.
- [25] S. C. Tai, M. W. Du, and R. C. T. Lee, "A transformational approach to synthesizing combinational circuits," *IEEE T. CAD*, Vol. 10, No. 3, pp. 286-295, March 1991.
- [26] S. Yang, *Logic Synthesis and Optimization Benchmark User Guide Version 3.0*, MCNC, Jan. 1991.