A New Expansion of Symmetric Functions and Their Application to Non-Disjoint Functional Decompositions for LUT Type FPGAs

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

Abstract— This paper presents a new expansion method for symmetric functions, and shows an application to non-disjoint decompositions. It is useful when the column multiplicity μ satisfies the condition $\mu \neq 2^r$. This paper also shows the realizations of rd73, rd84, and 9sym that require only 4, 6, and 5 Xilinx 3000 CLBs, respectively.

I Introduction

Functions that appears in arithmetic circuits often have symmetries. When logic functions have some symmetries, they have more efficient realizations than the functions without symmetry [14].

This paper presents a new expansion method for symmetric functions. It also shows the application of this expansion to the design of Look up table (LUT) type Field Programmable Gate Array (FPGA) by using non-disjoint decompositions.

An LUT type FPGA consists of many Configurable Logic Blocks (CLBs). We assume that each CLB realizes 1) arbitrary 5 input 1 output variable function, or 2) a pair of arbitrary 4 input functions. To design LUT type FPGAs, functional decomposition is useful [1, 6, 8, 9, 10, 13, 16]. Given a function f, a disjoint decomposition is to represent f in the form $f(X_1, X_2) = g(h_1(X_1), h_2(X_1), \ldots, h_r(X_1), X_2)$, where two variable sets X_1 and X_2 have no common element (Fig. 1.1). We consider the encoding that simplifies function h_r. Specifically, we consider an encoding such that $h_r(X_1) = x_i$; that is, $f(X_1, X_2) = g(h_1(X_1), h_2(X_1), \ldots, h_{r-1}(X_1), x_i, X_2)$. This can be also considered as a non-disjoint decomposition (Fig. 1.2). This type of decomposition is useful for designing LUT-type FPGAs.

As examples, we show realizations of $rd73$, $rd84$, and $9sym$ that require only 4, 6, and 5 CLBs, respectively. As far as we know, these are the smallest CLB realizations ever found.

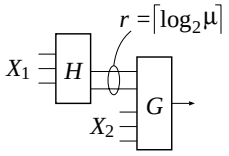

Figure 1.1: Disjoint decomposition.

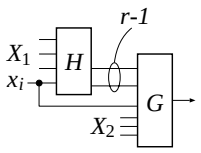

Figure 1.2: Non-disjoint decomposition.

II Symmetric Functions

Definition 2.1 *A function f is* **totally symmetric** *if any permutation of the variables in f does not change the function. A totally symmetric function is also called a* **symmetric function**.

Definition 2.2 *In a function $f(x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n)$, if the function $f(x_1, \ldots, x_j, \ldots, x_i, \ldots, x_n)$ that is obtained by interchanging variables x_i with x_j, is equal to the original function, then f is* **symmetric with respect to x_i and x_j**. *If any permutation of subset S of the variables does not change the function f, then f is* **partially symmetric**.

Definition 2.3 *The* **elementary symmetric functions** *of n variables are*

$S_0^n = \bar{x}_1 \bar{x}_2 \cdots \bar{x}_n$
$S_1^n = x_1 \bar{x}_2 \cdots \bar{x}_n \vee \bar{x}_1 x_2 \bar{x}_3 \cdots \bar{x}_n \vee \cdots \vee \bar{x}_1 \bar{x}_2 \cdots \bar{x}_{n-1} x_n$
$\cdots\cdots\cdots\cdots\cdots$
$S_n^n = x_1 x_2 \cdots x_n$

S_i^n =1 iff exactly i out of n inputs are equal to one. Let $A \subseteq \{0, 1, \ldots, n\}$. A symmetric function S_A^n is defined as follows:

$$S_A^n = \bigvee_{i \in A} S_i^n.$$

Example 2.1 $f(x_1, x_2, x_3) = x_1 x_2 x_3 \vee x_1 \bar{x}_2 \bar{x}_3 \vee \bar{x}_1 x_2 \bar{x}_3 \vee \bar{x}_1 \bar{x}_2 x_3$ *is a totally symmetric function. $f = 1$ when all the variables are one, or when only one variable is one. Thus, f can be written as $S_1^3 \vee S_3^3 = S_{\{1,3\}}^3$.* ∎

The following Lemma is well known [14].

Lemma 2.1 *An arbitrary n-variable symmetric function f is uniquely represented by elementary symmetric functions $S_0^n, S_1^n, \ldots, S_n^n$ as follows:*

$$f = \bigvee_{i \in A} S_i^n = S_A^n, \text{ where } A \subseteq \{0, 1, \ldots, n\}.$$

Definition 2.4 *Let $SB(n,k)$ be the n-variable symmetric function represented by the EXOR sum of all the products consisting of k positive literals:*

$$\begin{aligned}
&SB(n,0)=1\\
&SB(n,1)=\sum\nolimits_{\oplus} x_i\\
&SB(n,2)=\sum_{(i<j)\oplus} x_i x_j\\
&SB(n,3)=\sum_{(i<j<k)\oplus} x_i x_j x_k\\
&\cdots\cdots\cdots\\
&SB(n,n)=x_1 x_2 \cdots x_n
\end{aligned}$$

$SB(n,k)$ has been used as a benchmark function for AND-EXOR logic minimizer [12]. The following two lemmas were been proved by Komamiya [5] and reformulated by the author [14].

Lemma 2.2 *Let $x_1, x_2, \ldots, x_n$ be binary variables and r be an integer defined by $r = x_1 + x_2 + \cdots + x_n$, where + is an ordinary integer addition. Let the binary representation of r be*

$$(y_k, y_{k-1}, \ldots, y_1, y_0)_2,\ y_j \in \{0,1\}\ (j = 0, 1, \ldots, k).$$

In other words,

$$x_1 + x_2 + \cdots + x_n = 2^k y_k + 2^{k-1} y_{k-1} + \cdots + 2y_1 + y_0.$$

Then,

$$y_i = SB(n, 2^i).$$

Lemma 2.3 *Let $0 \le k_1 < k_2 < \cdots < k_s$, and $2^{k_1} + 2^{k_2} + \cdots + 2^{k_s} \le n$. Then,*

$$\bigwedge_{i=1}^{s} SB(n, 2^{k_i}) = SB(n, \sum_{i=1}^{s} 2^{k_i}).$$

Example 2.2

$SB(4,1)SB(4,2) = SB(4,3)$
$SB(6,2)SB(6,4) = SB(6,6)$
$SB(7,1)SB(7,2)SB(7,4) = SB(7,7)$ ∎

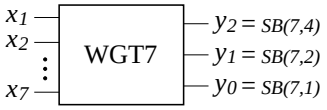

Figure 2.1: WGT7.

Definition 2.5 *WGT n is an n-input $\lceil \log_2(n+1) \rceil$-output function. It counts the number of 1's in the inputs and represents it by a binary number.*

By Lemma 2.2, WGTn produces $SB(n, 2^i)$, $(i = 0, 1, \ldots, \lceil \log_2(n+1) \rceil - 1)$, where $\lceil a \rceil$ denotes the smallest integer greater than or equal to a.

Example 2.3 *WGT7 has $x_1, x_2, \ldots, x_7$ as inputs and y_2, y_1, y_0 as outputs (Fig. 2.1). By Lemma 2.2, we have*

$$\begin{aligned}
y_2&=SB(7,4) = \sum_{i<j<k<l\ \oplus} x_i x_j x_k x_l\\
y_1&=SB(7,2) = \sum_{i<j\ \oplus} x_i x_j\\
y_0&=SB(7,1) = x_1 \oplus x_2 \oplus \cdots \oplus x_7
\end{aligned}$$

WGT7 is also called as rd73. ∎

The following is a new expansion method for symmetric functions using $SB(n,k)$ functions:

Theorem 2.1 *An arbitrary n-variable symmetric function f can be represented by $y_i = SB(n, 2^i)$, $(i = 0, 1, 2, \ldots, t)$ as follows:*

$$f = \bigvee_{(a_0, a_1, \ldots, a_t)} g(a_0, a_1, \ldots, a_t) y_0^{a_0} y_1^{a_1} \cdots y_t^{a_t},$$

where $g(a_0, a_1, \ldots, a_t)$ is 0 or 1, and $t = \lceil \log_2(n+1) \rceil - 1$.

(Proof) A symmetric function f only depends on the number of 1's in the inputs. Since WGTn counts the number of 1's in the input, we can represent f as a function of $y_0, y_1, \ldots, y_t$. □

III Functional Decomposition and Standard Encoding

3.1 Functional Decomposition

Definition 3.1 *Let $f(X_1, X_2)$ be a logic function, and (X_1, X_2) be a partition of X. $|X_1|$ denotes the number of variable in X_1. When $n_1 = |X_1|$ and $n_2 = |X_2|$, an equivalence relation $\sim$ on B^{n_1} is defined as follows: $\boldsymbol{a} \sim \boldsymbol{b} \Longleftrightarrow f(\boldsymbol{a}, X_2) = f(\boldsymbol{b}, X_2)$. Let the equivalence classes of B^{n_1} be Ψ_0, $\Psi_1, \ldots, \Psi_{\mu-1}$. In this case, μ is equal to the column multiplicity for the decomposition table f with the partition (X_1, X_2). Ψ_i is also used to represent the corresponding switching function.*

3.2 Standard Encoding

The minimum-length encoding uses $\lceil \log_2 \mu \rceil$ bits to encode $\Psi_0, \Psi_1, \ldots, \Psi_{\mu-1}$ where $\lceil a \rceil$ denotes the minimum integer not smaller than a. For example, in the case of $\mu = 5$,

Ψ_0 is coded by 000,
Ψ_1 is coded by 001,
Ψ_2 is coded by 010,
Ψ_3 is coded by 011, and
Ψ_4 is coded by 100.

Let the encoding functions be $h_1, h_2, \ldots, h_r$, where $r = \lceil \log_2 \mu \rceil$. In this encoding, "All the vectors in an equivalence class have the same codes," and is *strict*. Minimizing the number of the intermediate variables is preferable, and unused codes can be used as don't cares to simplify G [10].

IV Encoding that Simplifies the Intermediate Variables

In the strict encoding, "all the vectors in an equivalence class have the same codes." However, in general, the coding where, "two vectors in the different classes have different codes" is sufficient. Thus, two vectors in the same equivalence class may have different codes. Such an encoding is *non-strict*, and can simplifies the network H [10].

Many methods exist to encode the equivalence classes; $\Psi_0, \Psi_1, \ldots, \Psi_{\mu-1}$. In this paper, we will use an encoding that simplifies h_r. Assume that $\mu \neq 2^k$. If we can choose an encoding such that $h_r(X_1) = x_i$, then the network for h_r can be deleted.

Example 4.1 *Consider a 7-variable function $f(X_1, X_2)$, where $X_1 = \{x_1, x_2, x_3, x_4\}$ and $X_2 = \{x_5, x_6, x_7\}$, and assume that f is partially symmetric with respect to X_1. In this case, the column multiplicity μ of the decomposition chart for $f(X_1, X_2)$ is at most 5, since it is sufficient to identify if $0, 1, 2, 3$, or 4 of x_1, x_2, x_3, and x_4 are 1. Thus, f can be realized as Fig. 4.1. In the case of $\mu = 5$, we need three intermediate variables: h_1, h_2, and h_3.*

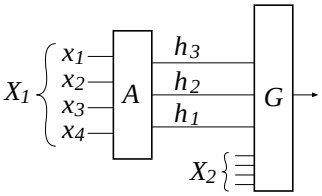

Figure 4.1: Standard encoding.

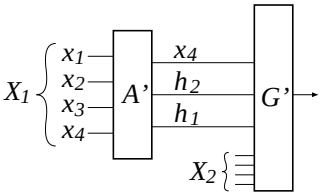

Figure 4.2: Encoding that simplifies h.

a) Standard Encoding
Table 4.1 shows the encoding for h_1, h_2, and h_3. In this case, we have

$$h_3 = SB(4,4) = x_1x_2x_3x_4 = S_4^4$$
$$h_2 = SB(4,2) = x_1x_2 \oplus x_1x_3 \oplus x_1x_4 \oplus x_2x_3 \oplus x_2x_4 \oplus x_3x_4 = S_{\{2,3\}}^4$$
$$h_1 = SB(4,1) = x_1 \oplus x_2 \oplus x_3 \oplus x_4 = S_{\{1,3\}}^4.$$

Note that Table 4.1 represents WGT4, witch is realized by block A. In Fig. 4.1, the network for A requires two CLBs since it produces three non-trivial functions. This encoding uses five code words.

b) An encoding that Simplifies h
For h_3, we use x_4 instead of $SB(4,4)$:

$$h_3 = x_4$$
$$h_2 = SB(4,2)$$
$$h_1 = SB(4,1)$$

Table 4.1: Standard Encoding.

$h_3 h_2 h_1$	*The number of 1's in X_1*
0 0 0	0
0 0 1	1
0 1 0	2
0 1 1	3
1 0 0	4

In this case, (h_3, h_2, h_1) shows the number of 1's in $\{X_1\}$. In other words,

(h_3, h_2, h_1)=(0, 0, 0) *denotes that* X_1 *has no 1*
(h_3, h_2, h_1)=(–, 0, 1) *denotes that* X_1 *has one 1*
(h_3, h_2, h_1)=(–, 1, 0) *denotes that* X_1 *has two 1's*
(h_3, h_2, h_1)=(–, 1, 1) *denotes that* X_1 *has three 1's*
(h_3, h_2, h_1)=(1, 0, 0) *denotes that* X_1 *has four 1's*

Since the function $h_3 = x_4$ is available as an input, we need not realize this function. As shown in Fig. 4.2, the network for A' represents a 4-input 3-output function, and can be realized by using only one CLB. This encoding uses 8 different code words. ∎

Theorem 4.1 *Consider the decomposition:*

$$f(X_1, X_2) = g(h_1(X_1), h_2(X_1), \ldots, h_r(X_1), X_2).$$

The following are sufficient conditions for $h_r(X_1)$ to be represented as $h_r(X_1) = x_i$, where $x_i \in X_1$ are

1) $\mu = 2^{r-1} + 1$ and,
2) There exist two equivalence classes Ψ_A and Ψ_B such that

$$\Psi_A = x_i \psi_A, \text{ and } \Psi_B = \bar{x}_i \psi_B.$$

(Proof) Assume 1) and 2). Then, merge the two equivalence classes as $\Psi_M = \Psi_A \cup \Psi_B$. This will reduce the number of equivalence classes by one. Since $\mu = 2^{r-1} + 1$, we can reduce the outputs for H by one. □

Partially symmetric functions often satisfy the above conditions.

Example 4.2 *Let $X_1 = (x_1, x_2, x_3)$, and the equivalence classes for the decomposition f with the partition (X_1, X_2) be*

$\Psi_0 = \bar{x}_1 \bar{x}_2 \bar{x}_3$
$\Psi_1 = \bar{x}_1 x_2 \vee \bar{x}_2 x_3 \vee \bar{x}_3 x_1$
$\Psi_2 = x_1 x_2 x_3$

In this case $\mu = 3$, but we have only to realize one intermediate variable. Let $\Psi_M = \Psi_0 \vee \Psi_2$, and we have

$\Psi_0 = \bar{x}_1 \Psi_M = \bar{x}_1 \bar{h}_1$
$\Psi_1 = h_1$
$\Psi_2 = x_1 \Psi_M = x_1 \bar{h}_1$

We have only to realize the function $h_1 = \Psi_1$. ∎

Example 4.3 *Let $X_1 = (x_1, x_2, x_3, x_4)$, and the equivalence classes for the decomposition for f with the partition (X_1, X_2) be*

$\Psi_0 = S_0^4(x_1, x_2, x_3, x_4) = \bar{x}_1 \bar{x}_2 \bar{x}_3 \bar{x}_4$
$\Psi_1 = S_1^4(x_1, x_2, x_3, x_4)$
$\Psi_2 = S_2^4(x_1, x_2, x_3, x_4)$
$\Psi_3 = S_3^4(x_1, x_2, x_3, x_4)$
$\Psi_4 = S_4^4(x_1, x_2, x_3, x_4) = x_1 x_2 x_3 x_4$

In this case, $\mu = 5$, but we have only to realize two intermediate variables $y_0 = SB(4, 1)$ and $y_1 = SB(4, 2)$. Note that

$\Psi_0 = \bar{x}_1 \bar{y}_1 \bar{y}_0$
$\Psi_1 = \bar{y}_1 y_0$
$\Psi_2 = y_1 \bar{y}_0$
$\Psi_3 = y_1 y_0$
$\Psi_4 = x_1 \bar{y}_1 \bar{y}_0$ ∎

Corollary 4.1 *Let $f(X_1, X_2)$ be symmetric with respect to $X_1 = \{x_1, x_2, x_3, x_4\}$. Then, f can be represented as $f(X_1, X_2) = g(y_0, y_1, x_1, X_2)$, where*

$y_0 = SB(4, 1) = x_1 \oplus x_2 \oplus x_3 \oplus x_4$, *and*
$y_1 = SB(4, 2) = x_1 x_2 \oplus x_1 x_3 \oplus x_1 x_4 \oplus x_2 x_3 \oplus x_2 x_4 \oplus x_3 x_4$

This Corollary is useful to design symmetric functions.

Example 4.4 *Realize WGT7 (rd73) by using LUTs.*

(Solution) WGT7 counts the number of 1's in the input, and represent it by a binary number (Fig. 2.1).

Let X be partitioned as (X_1, X_2), where $X_1 = (x_1, x_2, x_3, x_4)$ and $X_2 = (x_5, x_6, x_7)$. The column multiplicity of the decomposition chart (X_1, X_2) is five. So, the straightforward realization produces the network shown in Fig. 4.3, where WGT4 is a 4-input bit counting circuit and produces three functions:

$h_3 = SB(4, 4) = x_1 x_2 x_3 x_4$
$h_2 = SB(4, 2) = x_1(x_2 \oplus x_3 \oplus x_4) \oplus x_2(x_3 \oplus x_4) \oplus x_3 x_4$
$h_1 = SB(4, 1) = x_1 \oplus x_2 \oplus x_3 \oplus x_4$

Also, WGT3 is a 3-input bit-counting circuit (i.e., a full adder), and produces two functions:

$h_5 = SB(3, 2) = x_5 x_6 \oplus x_6 x_7 \oplus x_7 x_5$
$h_4 = SB(3, 1) = x_5 \oplus x_6 \oplus x_7$

G_1 adds two 2-bit numbers: (h_2, h_1) and (h_5, h_4) producing the two least significant bits of the sum, and G_2 adds a 2-bit number with a 3-bit number: (h_3, h_2, h_1) and (h_5, h_4) producing the most significant bit of the sum. In Fig. 4.3, WGT4 has three outputs and require two CLBs. Note that

$y_2 = h_3 \oplus h_2 h_5 \oplus h_1 h_4 (h_2 \oplus h_5)$
$y_1 = h_2 \oplus h_5 \oplus h_1 h_4$
$y_0 = h_1 \oplus h_4$

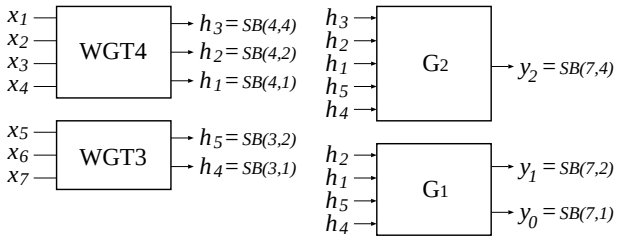

Figure 4.3: Realization of WGT7 with 5 LUTs.

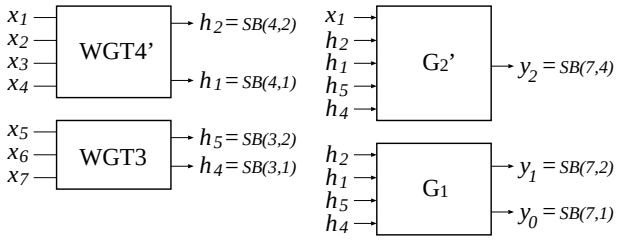

Figure 4.4: Realization of WGT7 with four LUTs.

Since each of G_1 and G_2 in Fig. 4.3 requires one LUT, we need five LUTs in total.

However, if $h_3 = SB(4,4)$ is replaced by $h_3 = x_1$ as shown in Fig. 4.4, we need only four LUTs. In this case, we use the relation $h_3 = x_1\bar{h}_1\bar{h}_2$ and $y_2 = x_1\bar{h}_1\bar{h}_2 \oplus h_2h_5 \oplus h_1h_4(h_2 \oplus h_5)$. ∎

Example 4.5 *Realize WGT8 (rd84) by LUTs.*

(Solution) The WGT8 realizes the four functions $SB(8,8)$, $SB(8,4)$, $SB(8,2)$, and $SB(8,1)$. Let X be partitioned as $X = (X_1, X_2, X_3)$, where $X_1 = (x_1, x_2, x_3, x_4)$, $X_2 = (x_5, x_6)$, and $X_3 = (x_7, x_8)$.

First, realize WGT4 as:

$SB(4,4)=x_1x_2x_3x_4$

$SB(4,2)=x_1(x_2 \oplus x_3 \oplus x_4) \oplus x_2(x_3 \oplus x_4) \oplus x_3x_4$

$SB(4,1)=x_1 \oplus x_2 \oplus x_3 \oplus x_4$

Second, realize WGT6 as:

$SB(6,4)=SB(4,4) \oplus SB(4,3)(x_5 \oplus x_6) \oplus SB(4,2)x_5x_6$

$SB(6,2)=SB(4,2) \oplus SB(4,1)(x_5 \oplus x_6) \oplus x_5x_6$

$SB(6,1)=SB(4,1) \oplus x_5 \oplus x_6$

Here, we use the relation:

$SB(4,3) = SB(4,2)SB(4,1)$

And, finally realize WGT8 as:

$y_3=B(8,8) \ =SB(6,6)x_7x_8$

$y_2=SB(8,4)=SB(6,4) \oplus SB(6,3)(x_7 \oplus x_8) \oplus SB(6,2)x_7x_8$

$y_1=SB(8,2)=SB(6,2) \oplus SB(6,1)(x_7 \oplus x_8) \oplus x_7x_8$

$y_0=SB(8,1)=SB(6,1) \oplus x_7 \oplus x_8$

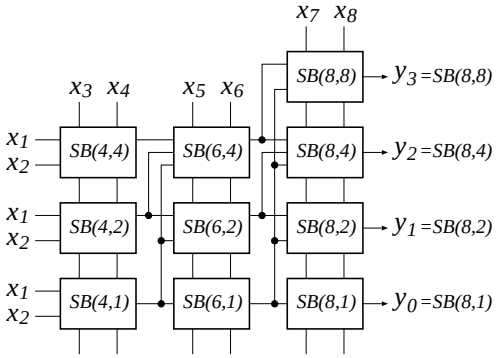

Figure 4.5: Realization of WGT8.

Table 4.2: Truth Table for 9*sym*.

y_2	y_1	y_0	x_8	x_9	f
0	0	0	–	–	0
0	0	1	1	1	1
0	1	0	1	–	1
0	1	0	–	1	1
0	1	1	–	–	1
1	0	0	–	–	1
1	0	1	0	–	1
1	0	1	–	0	1
1	1	0	0	0	1
1	1	1	–	–	0

Here, we use the relations:

$SB(6,6)=SB(6,4)SB(6,2)$

$SB(6,3)=SB(6,2)SB(6,1)$

Thus, WGT8 is realized as Fig. 4.5. However, if we use the relation,

$SB(4,4) = x_1x_2x_3x_4 = x_1\overline{SB(4,2)}\,\overline{SB(4,1)}$,

the LUT for $SB(4,4)$ is absorbed into the CLB for $SB(6,4)$. Also, note that each of the pair of LUTs for $\{SB(4,2), SB(4,1)\}$, $\{SB(6,2), SB(6,1)\}$, and $\{SB(8,2), SB(8,1)\}$ is merged into one CLB. Thus, WGT8 requires only 6 CLBs. ∎

Example 4.6 *Realize the 9-input symmetric function 9sym by LUTs.*

(Solution) 9sym is represented as $f = S^9_{\{3,4,5,6\}}(x_1, x_2, \ldots, x_9)$. $f = 1$ if and only if the number of 1's in the input is 3, 4, 5, or 6. To realize 9sym, we use WGT7. From the definition of 9sym, we have Table 4.2. In Fig. 4.6, the network G for Table 4.2 requires only one CLB. Since WGT7 requires only four CLBs (Example 4.4), 9sym requires only five CLBs. ∎

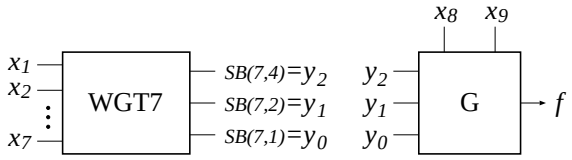

Figure 4.6: Realization for $9sym$.

Table 4.3: Number of XC3000 CLBs.

		IMO DEC	PDD MAP	HYDE	This paper
9sym	SYM9	7	5	6	5
rd73	WGT7	5	–	5	4
rd84	WGT8	8	8	7	6

IMODEC: Wurth-Eckl-Antreich: DAC-95 [7].
PDDMAP: Cong-Hwang: FPGA-97 (XC4000) [2].
HYDE: Jiang-Jou-Huang: DAC-98 [4].

V Conclusion and Comments

In this paper, we have presented a new expansion theorem for an n-variable symmetric function f by using $SB(n, 2^i)$ functions $(i = 0, 1, \ldots, \lceil \log_2(n+1) \rceil - 1)$.

Representation of f by using S_i^n $(i = 0, 1, 2, \ldots, n)$ corresponds to *one-hot encoding*, while representation of f by using $SB(n, 2^i)$ corresponds to *minimum-length strict encoding*. When the number of equivalence class μ satisfies the relation $2^{r-1} < \mu < 2^r$, a strict encoding requires r intermediate variables. However, non-strict encoding often requires a smaller number of intermediate variables. This produces non-disjoint decompositions. By using a new expansion method and non-disjoint decompositions, we obtained the best realizations for well known benchmark functions (rd73, rd84, and 9sym). These realizations were found by inspection rather than by a computer program.

The FPGA design systems such as [2, 4, 7, 11, 15] consider non-disjoint decompositions or encodings that simplify the intermediate variables. Unfortunately, they failed to find the optimum solutions for rd73, rd84, and 9sym (see Table 4.3). To develop the design system that obtains optimum solutions for these functions is challenging.

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. J. T. Butler's comments improved the presentation. Mr. M. Matsuura edited the LaTeX file.

References

[1] 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. , pp. 1226-1236, Oct. 1996.

[2] J. Cong and Y.-Y Hwang, "Partially-dependent functional decomposition with applications in FPGA synthesis and mapping," *Fifth Int. Symp. on Field-Programmable Gate Arrays*, pp. 35-42, Feb. 1997.

[3] Feng Wang and D. L. Dietmeyer, "Exploiting near symmetry in multilevel logic synthesis," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, Vol. 17, No. 9, pp. 772-781, Sept. 1998.

[4] J.-H. R. Jian, J.-Y. Jou, and J.-D. Huang, "Compatible class encoding in hyper-function decomposition for FPGA synthesis," *Design Automation Conference*, pp. 712-717, June 1998.

[5] Y. Komamiya, "Theory of computing networks," *Electrotechnical Laboratory in Japanese Government*, July 10, 1959.

[6] 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.

[7] C. Legl, B. Wurth, and K. Eckl, "Computing support-minimal subfunctions during functional decomposition," *IEEE Trans. VLSI*, Vol. 6, No. 3, pp. 354-363, Sept. 1998.

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

[9] S. Minato and G. De Micheli, "Finding all simple disjunctive decompositions using irredundant sum-of-products forms," *ICCAD-99*, pp. 111-117, Nov. 1998.

[10] R. Murgai, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Optimum functional decomposition using encoding," *31st Design Automation Conference*, pp. 408-414, June 1994.

[11] M. Raski, L. Jozwiak, M. Noricka, and T. Luba, "Non-disjoint decomposition of Boolean functions and its application in FPGA-oriented technology mapping," *Euromicro-97*, pp.24-30.

[12] T. Sasao and Ph. Besslich, "On the complexity of MOD-2 Sum PLA's," *IEEE TC*, Vol. 39, No. 2, pp. 262-266, Feb. 1990.

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

[14] T. Sasao, *Switching Theory for Logic Synthesis*, Kluwer Academic Publishers, 1999.

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

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