A Method to Realize Logic Functions using LUTs and OR Gates

Munehiro Matsuura[1] and Tsutomu Sasao[1,2]

[1]Department of Computer Science and Electronics, Kyushu Institute of Technology
[2]Center for Microelectronic Systems, Kyushu Institute of Technology

Abstract— This paper introduces a new decomposition method called OR-partitioning, where a logic function is realized as an OR of p disjoint sub-functions. This paper also compares three methods to decompose a logic function: The first one is standard functional decomposition; the second one is functional decomposition combined with the Shannon expansion; and the third one is functional decomposition combined with OR-partitioning. The experimental results using many benchmark functions show that functional decomposition combined with OR-partitioning requires smaller amount of memory than functional decomposition combined with the Shannon expansion when LUTs are used to realize functions.

I. Introduction

In this paper, we consider methods to decompose a logic function into several smaller functions. This problem appears in logic design for LUT-type FPGAs [4, 6, 7, 12, 16, 22, 23, 24] as well as in logic simulation [13]. In many cases, the maximum number of inputs for an LUT is limited to k. Such an LUT is denoted by a k-LUT.

When an n-variable logic function is realized by a single LUT, 2^n bits are necessary. However, in many cases, logic functions have special properties, and can be realized more efficiently. Functional decomposition is a standard method to realize logic functions using LUTs. Unfortunately, many functions are undecomposable. For such functions, the Shannon expansion is often used to reduce the number of dependent variables. In this paper, we show a new method to realize undecomposable functions using LUTs and OR gates. This method tries to find a decomposition that produces sub-functions with as small supports as possible. Experimental results show that the new method outperforms the Shannon expansion.

II. Functional Decomposition

Functional decomposition is a standard method to realize a logic function using LUTs [1, 3].

Definition 2.1 *Let $X = (x_1, x_2, \ldots, x_n)$ be the input variables. $\{X\}$ denotes the set of variables in X. Let $\{X_1\}$ and $\{X_2\}$ be subsets of $\{X\}$. If $\{X_1\} \cup \{X_2\} = \{X\}$ and $\{X_1\} \cap \{X_2\} = \phi$, then (X_1, X_2) is a* **partition** *of X. $|X|$ denotes the number of variables in X.*

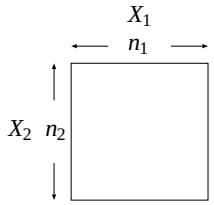

Fig. 2.1. Decomposition chart for $f(X_1, X_2)$.

Definition 2.2 *Let $f(X)$ be a logic function, and let X be partitioned into (X_1, X_2). The* **decomposition chart** *$M(f : X_1, X_2)$ is a table with $2^{|X_1|}$ columns and $2^{|X_2|}$ rows, where each column and row is labeled by a binary number. Each row and column has all the patterns of $|X_1|$-bit and $|X_2|$-bit, respectively, and the corresponding element of the table denotes the value of f.*

Definition 2.3 *Let X be partitioned into (X_1, X_2), where $|X_1| = n_1$ and $|X_2| = n_2$. The number of different column patterns in the decomposition chart $M(f : X_1, X_2)$ is the* **column multiplicity** *for X_1 and is denoted by μ. The variables in X_1 and X_2 are the* **bound variables** *and the* **free variables**, *respectively.*

Theorem 2.1 *Let (X_1, X_2) be a partition of X, and let μ be the column multiplicity for X_1 of the function f. Then, different column patterns in $M(f : X_1, X_2)$ can be encoded by r_1 bits, where $r_1 = \lceil \log_2 \mu \rceil$. The function f can be represented as $f(X_1, X_2) = g(h(X_1), X_2)$, where $g(Y_1, X_2)$ is a function $B^{r_1} \times B^{n_2} \rightarrow B$, and Y_1 is a 2^{r_1}-valued variable. Fig. 2.2 shows a network for $f(X_1, X_2) = g(h(X_1), X_2)$.*

Functional decomposition consists in finding the representation of function f in the form $g(h(X_1), X_2)$. If $n_1 = \lceil \log_2 \mu \rceil$ holds for any bound set X_1 such that $|X_1| \leq k$, then the function is **undecomposable** [18], and we cannot use Theorem 2.1. For such function, we can use the Shannon expansion, and the OR-partitioning introduced below.

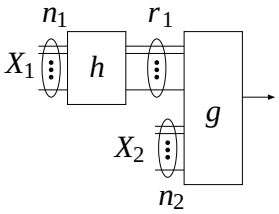

Fig. 2.2. Network realization for functional decomposition $f(X_1, X_2) = g(h(X_1), X_2)$.

III. Realization of Logic Functions using the Shannon expansions

Here, we present design methods using the Shannon expansion.

Definition 3.1 *Let $X = (x_1, x_2, \ldots, x_n)$ be the input variable. A symbol $X^{\vec{a}}$ represents the function*

$$\begin{aligned} X^{\vec{a}} &= 1 \ (\textit{When } X = \vec{a}) \\ &= 0 \ (\textit{When } X \neq \vec{a}), \end{aligned}$$

where $\vec{a} \in B^n$.

Definition 3.2 *$support(f)$ denotes the set of variables on which the function f depends.*

By applying the Shannon expansion iteratively, we have the next theorem.

Theorem 3.1 *An arbitrary n-variable function $f(X)$ can be expanded as*

$$f(X_1, X_2) = \bigvee_{i=0}^{2^r-1} X_2^{\vec{a}_i} f(X_1, \vec{a}_i), \tag{3.1}$$

where $\vec{a}_i$ is a binary vector representing an integer i, and $\vec{a} = (a_1, a_2, \ldots, a_r)$.

The expansion (3.1) is useful to realize undecomposable functions by k-LUTs.

Definition 3.3 *A* **multiplexer** *(MUX) is the selection circuit shown in Fig. 3.1. It has s control inputs $(x_1, \ldots, x_s)$ and 2^s data inputs $(y_0, y_1, \ldots, y_{2^s-1})$. Let $g(x_1, \ldots, x_s, y_0, y_1, \ldots, y_{2^s-1})$ be the output function. Then, $g = y_a$ when the decimal representation of the control input $(x_1, \ldots, x_s)$ is a. That is, when the control input is $(0, 0, \ldots, 0)$, the top data input y_0 is selected. When the the control input is $(0, 0, \ldots, 1)$, the second data input y_1 is selected. Finally, when the control input is $(1, 1, \ldots, 1)$, the last data input y_{2^s-1} is selected. A MUX with s control inputs is denoted by s-***MUX**.*

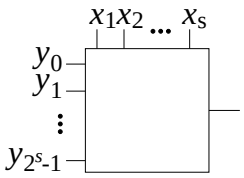

Fig. 3.1. s-MUX.

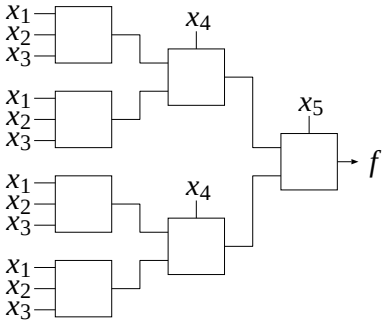

Fig. 3.2. Realization of a 5-variable function using 3-LUTs.

Example 3.1 *Let us realize a 5-variable function $f(X)$ using 3-input LUTs. Let $X = (x_1, x_2, x_3, x_4, x_5)$, $X_1 = (x_1, x_2, x_3)$, and $X_2 = (x_4, x_5)$. By applying the Shannon expansions to $f(X)$ with respect to X_2, we have*

$$\begin{aligned} f(X_1, X_2) &= X_2^{(0,0)} f(X_1, 0, 0) \vee X_2^{(0,1)} f(X_1, 0, 1) \vee \\ &\quad X_2^{(1,0)} f(X_1, 1, 0) \vee X_2^{(1,1)} f(X_1, 1, 1). \end{aligned}$$

Fig. 3.2 shows the realization using 3-LUTs. Three LUTs which are connected to x_4 or x_5 realize a 2-MUX. This realization requires 7 LUTs. (End of Example)

In the practical applications, we can often reduce the size of the support of $f(X_1, a_1, a_2)$ by finding an appropriate set of variable to expand. The advantage of the Shannon expansion is that any function can be decomposed into sub-functions with at most k inputs.

IV. Design Method using OR-Partitioning

In this section, we introduce a new decomposition method. The advantage of OR-partitioning is that it often produces faster and smaller realizations than the Shannon expansion. Unfortunately, some functions do not have an OR-partitioning.

Theorem 4.1 *Let (X_1, X_2) be a partition of $X = (x_1, x_2, \ldots, x_n)$. Then, $f(X_1, X_2)$ can be represented as*

$$f(X_1, X_2) = \bigvee_{i=0}^{p-1} f_i(X_1, X_2), \tag{4.1}$$

x_3x_4 \ x_1x_2	Φ_0 00	Φ_1 01	Φ_2 10	Φ_3 11
00	1			1
01	1	1		
10			1	1
11		1	1	

Fig. 4.1. Decomposition chart for $f(x_1, x_2, x_3, x_4)$.

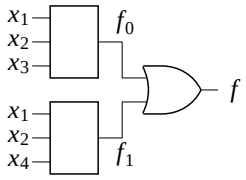

Fig. 4.2. Realization of $f = g_0(x_1, x_2, x_3) \vee g_1(x_1, x_2, x_4)$.

where $f_i(X_1, X_2) = g_i(h(X_1), X_2)$.

In the decomposition $f_i(X_1, X_2) = g_i(h(X_1), X_2)$, we try to reduce the support size of g_i $(i = 0, 1, \ldots, p-1)$. To show the idea of reducing the support of g, we use the following example.

Example 4.1 *Consider the four-variable function*

$$f(x_1, x_2, x_3, x_4) = \bar{x}_1\bar{x}_2\Phi_0 \vee \bar{x}_1x_2\Phi_1 \vee x_1\bar{x}_2\Phi_2 \vee x_1x_2\Phi_3,$$

where $\Phi_0 = \bar{x}_3$, $\Phi_1 = x_4$, $\Phi_2 = x_3$, *and* $\Phi_3 = \bar{x}_4$. *Let* $X_1 = (x_1, x_2)$ *and* $X_2 = (x_3, x_4)$. *As shown in the decomposition chart in Fig. 4.1, all the columns are distinct. So, the function is undecomposable for this partition. Consider the problem of partitioning* $\mathcal{F} = \{\Phi_0, \Phi_1, \Phi_2, \Phi_3\}$ *into two blocks so that the support of both blocks are reduced. Let the partition be* $\Pi = \{[\Phi_0, \Phi_2], [\Phi_1, \Phi_3]\}$. *Note that the block* $[\Phi_0, \Phi_2]$ *only depends on* x_3, *while the block* $[\Phi_1, \Phi_3]$ *only depends on* x_4. *In this case,* f *can be represented as* $f(x_1, x_2, x_3, x_4) = f_0 \vee f_1$, *where* $f_0 = \bar{x}_1\bar{x}_2\Phi_0 \vee x_1\bar{x}_2\Phi_2$ *and* $f_1 = \bar{x}_1x_2\Phi_1 \vee x_1x_2\Phi_3$, *and* f_0 *and* f_1 *can be represented by* $f_0 = g_0(x_1, x_2, x_3)$ *and* $f_1 = g_1(x_1, x_2, x_4)$. *Note that the supports of* f_0 *and* f_1 *are reduced by one, and* f *can be realized by the circuit shown in Fig. 4.2.* *(End of Example)*

Example 4.2 *Let* $\mu = 12$ *be the column multiplicity of the decomposition chart for* $f(X_1, X_2)$. *Let Fig. 4.3 be the decomposition chart, where* Φ_i $(i = 1, 2, \ldots, 12)$ *denote different column patterns of the decomposition chart. In Theorem 4.1 let* $p = 4$, *and let* $f(X_1, X_2)$ *be decomposed as*

$$f(X_1, X_2) = f_0(X_1, X_2) \vee f_1(X_1, X_2) \vee f_2(X_1, X_2) \vee f_3(X_1, X_2), \quad (4.2)$$

where $f_i(X_1, X_2) = g_i(h(X_1), X_2)$, *and* f_0, f_1, f_2, *and* f_3 *realize functions shown in Fig. 4.4(a), (b), (c), (d), respectively. Then,* f *can be realized as shown in Fig. 4.5. In practical applications, we can often reduce the support size of* g_i *by considering the partition of* $\{\Phi_i\}$. *(End of Example)*

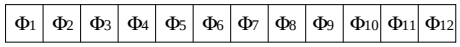

Fig. 4.3. Decomposition chart for function f.

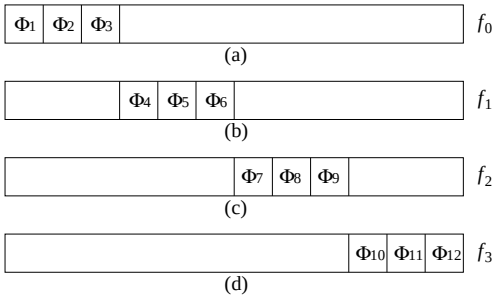

Fig. 4.4. Decomposition charts for f_0, f_1, f_2, and f_3.

For the decomposition introduced in Theorem 4.1, we can formulate the following problem.

Problem 4.1 *Decompose the function* f *into the form (4.1). Let* $n_2(i)$ *be the support size of* $g_i(Y_1, X_2)$, *where*

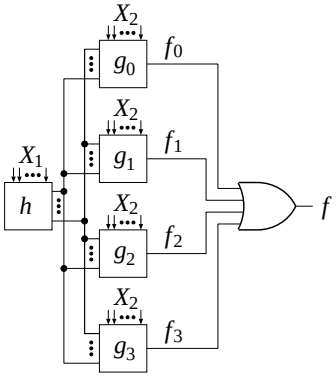

Fig. 4.5. Realization of
$f = f_0(X_1, X_2) \vee f_1(X_1, X_2) \vee f_2(X_1, X_2) \vee f_3(X_1, X_2)$.

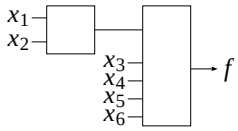

Fig. 4.6. Realization using functional decomposition.

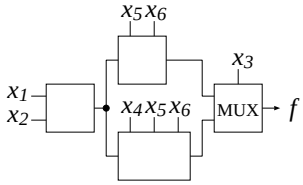

Fig. 4.7. Functional decomposition combined with Shannon expansion.

$f_i(X_1, X_2) = g_i(h(X_1), X_2)$. *Find the partition* (X_1, X_2) *and the partition of* $\mathcal{F} = \{\Phi_0, \Phi_1, \ldots, \Phi_{\mu-1}\}$ *that minimize the cost function defined by* $2^r \cdot u + \sum_{i=0}^{p-1} 2^{n_2(i)}$, *where* $r = |X_1|$ *and* $u = \lceil \log_2 \mu \rceil$.

This problem corresponds to the minimization of the amount of memory needed to realize g_i $(i = 0, 1, \ldots, p-1)$ by LUTs.

Example 4.3 *Consider the six-variable function:*

$$f = (\bar{x}_1\bar{x}_2 \vee x_1x_2)x_3x_4 \vee (\bar{x}_1x_2 \vee x_1\bar{x}_2)x_5x_6.$$

Let $X_1 = (x_1, x_2)$ *and* $X_2 = (x_3, x_4, x_5, x_6)$. *Fig. 4.6 shows the realization of* f *using functional decomposition. It requires a 2-LUT and a 5-LUT. Fig. 4.7 shows the realization of* f *using functional decomposition combined with Shannon expansion. It requires a 2-LUT, a 3-LUT, a 4-LUT, and a 1-MUX. Fig. 4.8 shows the realization of* f *using functional decomposition combined with OR-bi-partitioning. It requires a 2-LUT, two 3-LUTs, and a 2-input OR gate. Different realizations require different number of bits: A single LUT requires* $2^6 = 64$ *bits; the functional decomposition requires* $2^2 + 2^5 = 36$ *bits; the functional decomposition combined with Shannon expansion requires* $2^2 + 2^3 + 2^4 = 28$ *bits; the functional decomposition combined with OR-partitioning requires* $2^2 + 2 \cdot 2^3 = 20$ *bits. When an MUX or an OR gate is realized by an LUT, Shannon expansion requires* $28 + 2^3 = 36$ *bits, while OR-partitioning requires* $20 + 2^2 = 24$ *bits. (End of Example)*

To find an optimal solution for Problem 4.1, we should consider different partitions of the input variables X into X_1 and X_2, and different partitions of columns into p groups. No efficient exact method to solve Problem 4.1 is known, and we have to resort to a heuristic method. In [26], bi-decomposition

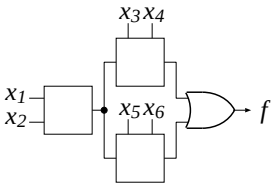

Fig. 4.8. Functional decomposition combined with OR-bi-partitioning.

$f(X_1, X_2) = \alpha(g_1(X_1), g_2(X_2))$ that minimizes $|X_1| + |X_2|$ is considered. It can be considered as a solution to a restricted case of Problem 4.1. The number of different partitions of the input variables is $2^n - 2$. The following lemma shows the complexity of the column partitioning problem.

Lemma 4.1 *[8]: The number of ways of placing* μ *distinct objects into* p *non-distinct cells with no cell left empty is* $S(\mu, p) = \frac{1}{p!}\sum_{i=0}^{p-1}(-1)^i \binom{p}{i}(p-i)^\mu$. $S(\mu, p)$ *is called the* **Stirling number of the second kind**.

For example, when $p = 4$, the values of $S(\mu, p)$ for $\mu = 7$, 8, 9, and 10 are 350, 1701, 7770, and 34105, respectively.

V. Detailed Algorithms

In this section, we show three different algorithms to solve decomposition problems.

A. Ordinary Decomposition Algorithm

The following algorithm finds the best size of X_1 that heuristically minimizes the total amount of memory needed to realize f, measured using an ordinary functional decomposition method.

Algorithm 5.1 *(Disjoint decomposition.)*

1. *Let* $(x_1x_2\cdots x_n)$ *be the BDD variable order that minimizes the size of the BDD.*
2. *For* $r = 2$ *to* $n-1$ *do step 3.*
3. $Size(r) \leftarrow 2^r \cdot u + 2^{n_2}$, *where* $u = \lceil \log_2 \mu \rceil$, μ *is the column multiplicity for the decomposition* $g(h(X_1), X_2)$, $X_1 = (x_1, x_2, \ldots, x_r)$, $X_2 = (x_{r+1}, x_{r+2}, \ldots, x_n)$ *and* $n_2 = |X_2| + u$.
4. *Obtain* r *that makes* $Size(r)$ *minimum, and return* r.

Since the order of the input variables in the BDD is fixed, this algorithm produces sub-optimal solutions.

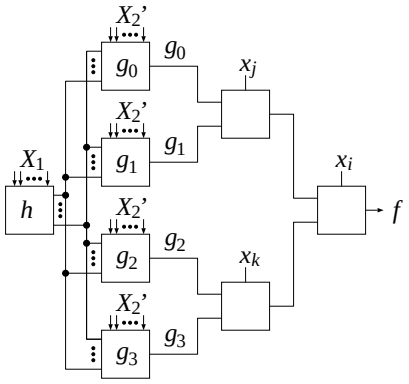

Fig. 5.1. Realization of $f = \bar{x}_i\bar{x}_j g_0(h(X_1), X_2) \vee \bar{x}_i x_j g_1(h(X_1), X_2) \vee x_i\bar{x}_k g_2(h(X_1), X_2) \vee x_i x_k g_3(h(X_1), X_2)$.

B. The Shannon Expansion Algorithm

In this subsection, we show a heuristic algorithm based on the Shannon expansion to design a circuit similar to Fig. 5.1. Note that this circuit realizes the function f in the form

$$f(X_1, X_2) = \bar{x}_i\bar{x}_j g_0(h(X_1), \hat{X}_2) \vee \bar{x}_i x_j g_1(h(X_1), \hat{X}_2) \vee x_i\bar{x}_k g_2(h(X_1), \hat{X}_2) \vee x_i x_k g_3(h(X_1), \hat{X}_2), \quad (5.1)$$

where $\{X_2\} = \{\hat{X}_2\} \cup \{x_i\} \cup \{x_j\} \cup \{x_k\}$. Given the partition (X_1, X_2), the goal is to determine variables such as x_i, x_j, and x_k. First, we show Algorithm 5.2 to partition a function into two sub-functions. Then, we show Algorithm 5.3 to partition a function into p sub-functions. These algorithms try to reduce the support sizes of g_i.

Let $f(X)$ be a function, and (X_1, X_2) be a partition of X. Let $\mathcal{F} = \{\Phi_0, \Phi_1, \ldots, \Phi_{\mu-1}\}$ be the set of sub-functions representing different column patterns in the decomposition chart for (X_1, X_2).

Algorithm 5.2 *(The Shannon bi-partition.)*

1. *Let $\mathcal{F} = \{\Phi_0, \Phi_1, \ldots, \Phi_{\mu-1}\}$, and let $\{X_2\}$ be the support for $\mathcal{F}$.*
2. *For all $x_j \in \{X_2\}$, do step 3.*
3. *Let the partition be $\Pi = (\mathcal{F}_0, \mathcal{F}_1)$, where $\mathcal{F}_0 = \{\Phi_i(x_{r+1}, \ldots, x_{j-1}, 0, x_{j+1}, \ldots, x_n)\}$ and $\mathcal{F}_1 = \{\Phi_i(x_{r+1}, \ldots, x_{j-1}, 1, x_{j+1}, \ldots, x_n)\}$. Let $s_0(j)$ and $s_1(j)$ be the sizes of supports for $\mathcal{F}_0$ and $\mathcal{F}_1$, respectively. Obtain $s_j = 100 \cdot \max(s_0(j), s_1(j)) + s_0(j) + s_1(j)$.*
4. *Find the partition $\Pi = (\mathcal{F}_0, \mathcal{F}_1)$ that makes s_j minimum and return $\Pi = \{\mathcal{F}_0, \mathcal{F}_1\}$.*

If $\min\{support(\mathcal{F}_0), support(\mathcal{F}_1)\} < |X_2|$, then there is a possibility to reduce the amount of memory. In step 3), we find a variable that minimizes the maximum size of the supports for $\mathcal{F}_0$ and $\mathcal{F}_1$. If there are more than two candidates, we select a variable that minimizes the sum of the size of the supports.

Algorithm 5.3 *(The Shannon p-partitioning.)*

1. *Let $\Pi \leftarrow \{[\Phi_0, \Phi_1, \ldots, \Phi_{\mu-1}]\}$ and $q \leftarrow \lceil \log_2 p \rceil$.*
2. *Do step 3 while $q > 0$.*
3. *By using Algorithm 5.2, for each block in Π, bi-partition it into two. Let $q \leftarrow q - 1$.*
4. *Return the final partition $\Pi = \{\mathcal{F}_0, \mathcal{F}_1, \ldots, \mathcal{F}_{p-1}\}$.*

The partitioning algorithm uses a BDD with a fixed variable order to represent a logic function. In this case, the partitioning problem for the input variables X is to determine a point $r \in \{1, \ldots, n\}$ in the given variable order. In this case, we only search the limited number of combinations, so the solution is sub-optimal.

When f is decomposed as (5.1), $Size(r) = 2^r \cdot u + \sum_{i=0}^{p-1} 2^{n_2(i)}$ denotes the required amount of memory, where $r = |X_1|$, $n_2(i) = |support(\mathcal{F}_i)| + u$, and $u = \lceil \log_2 \mu \rceil$. Note that we do not count the cost of the multiplexer. In order to minimize $Size(r)$, we change the size of X_1, i.e., the value of r. The following algorithm finds the value of r that minimizes $Size(r)$. Let n be the number of inputs for f. Our experimental results show that $Size(r)$ is minimum when $n/4 < |X_1| < n/2$. Therefore, to obtain a partition (X_1, X_2) for X, we first search the partitions satisfying $n/4 < |X_1| < n/2$.

Algorithm 5.4 *(Partitioning of the input variables for the Shannon expansion.)*

1. *Let $(x_1 x_2 \cdots x_n)$ be the BDD variable order that minimizes the size of the BDD. Let $r = \lceil \frac{n}{4} \rceil$.*
2. *Do step 3 while $r < \max_{i=0}^{p-1}\{|support(\mathcal{F}_i)|\}$.*
3. *Let $X_1 = (x_1, x_2, \ldots, x_r)$. Obtain μ. Apply Algorithms 5.3 to find the partition $\Pi = \{\mathcal{F}_0, \mathcal{F}_1, \ldots, \mathcal{F}_{p-1}\}$. Let $Size(r) = 2^r \cdot u + \sum_{i=0}^{p-1} 2^{n_2(i)}$, where $n_2(i) = |support(\mathcal{F}_i)| + u$ and $u = \lceil \log_2 \mu \rceil$. Let $r \leftarrow r + 1$.*
4. *Find r that minimizes $Size(r)$, and return the partition $\Pi = \{\mathcal{F}_0, \mathcal{F}_1, \ldots, \mathcal{F}_{p-1}\}$.*

C. OR-Partitioning Algorithm

In this section, we show a heuristic algorithm for OR-partitioning shown in Theorem 4.1. First, we show Algorithm 5.5 that partitions a function into two sub-functions. Then, we show Algorithm 5.6 that partitions a function into p sub-functions. These algorithms attempt to reduce the supports for the sub-functions.

Let the given function be f, and X be the support of f, and (X_1, X_2) be a partition of X. Let $\mathcal{F} = \{\Phi_0, \Phi_1, \ldots, \Phi_{\mu-1}\}$ be the set of sub-functions representing different column patterns in the decomposition chart for (X_1, X_2).

Algorithm 5.5 *(OR-bi-partitioning.)*

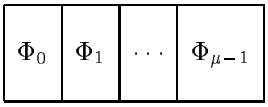

Fig. 5.2. Decomposition chart for f.

1. *Let $\mathcal{F} = \{\Phi_0, \Phi_1, \ldots, \Phi_{\mu-1}\}$. Let $\{X_2\}$ be the support for $\mathcal{F}$. If $|support(\Phi_i)| = |X_2|$ for some i, then f is* **unpartitionable**, *and return.*
2. *For all $x_j \in \{X_2\}$ do step 3.*
3. *Let the partition be $\Pi = (\mathcal{F}_0, \mathcal{F}_1)$, where $\mathcal{F}_0$ is the set of functions which are independent of x_j, and $\mathcal{F}_1$ is the set of functions that depend on x_j. Let $s_0(j)$ and $s_1(j)$ be the sizes of supports for $\mathcal{F}_0$ and $\mathcal{F}_1$, respectively. Obtain $s_j = 100 \cdot \max\{s_0(j), s_1(j)\} + s_0(j) + s_1(j)$.*
4. *Find a partition $\Pi = (\mathcal{F}_0, \mathcal{F}_1)$ that makes s_j minimum.*
5. *If $\max\{s_0(j), s_1(j)\} < |X_2|$, then f is* **partitionable** *else f is* **unpartitionable**, *return $\Pi = (\mathcal{F}_0, \mathcal{F}_1)$.*

Given a partition (X_1, X_2) of X, where $X_1 = (x_1, x_2, \ldots, x_r)$ and $X_2 = (x_{r+1}, x_{r+2}, \ldots, x_n)$, Algorithm 5.5 examines $n - r$ different partitions. If f is partitionable, then there is a possibility to reduce the amount of memory. In this case, we use the following algorithm.

Algorithm 5.6 *(OR-p-partitioning.)*

1. *Let $\Pi \leftarrow \{[\Phi_0, \Phi_1, \ldots, \Phi_{\mu-1}]\}$.*
2. *Do steps 3 to 5 in decreasing order of the support sizes for all $\mathcal{F}_i$ in Π.*
3. *By using Algorithm 5.5, try to partition $\mathcal{F}_i$ into two blocks.*
4. *When $\mathcal{F}_i$ is bi-partitionable, partition $\mathcal{F}_i$ into two blocks $\mathcal{F}_j$ and $\mathcal{F}_k$, $\Pi \leftarrow (\Pi - \mathcal{F}_i) \cup (\mathcal{F}_j \cup \mathcal{F}_k)$.*
5. *When $p = |\Pi|$ or all $\mathcal{F}_i$ in Π are unpartitionable, return $\Pi = \{\mathcal{F}_0, \mathcal{F}_1, \ldots, \mathcal{F}_{p-1}\}$.*

When applying Algorithms 5.5 and 5.6, we may encounter the case, where $\mathcal{F}$ includes many sub-functions with a few support variables. In such a case, we cannot find a good solution, so we apply the following algorithm.

Algorithm 5.7 *(Improved OR-partitioning)*

1. *Let $\Pi = \{\mathcal{F}_0, \mathcal{F}_1, \ldots, \mathcal{F}_{p-1}\}$ be the partition obtained by Algorithm 5.6. Let d_0, d_1, $\ldots$, and d_{p-1} be the sizes of supports for $\mathcal{F}_0$, $\mathcal{F}_1$, $\ldots$, $\mathcal{F}_{p-1}$, respectively. Let $v_{min} \leftarrow \sum_{k=0}^{p-1} 2^{d_k}$.*
2. *Do step 3 in decreasing order of the support sizes for all $\mathcal{F}_i$ in Π.*
3. *Do steps 4 to 5 for all the elements Φ_a in $\mathcal{F}_i$.*
4. *Move Φ_a to all other blocks, and compute the value of $v_i = \sum_{k=0}^{p-1} 2^{d_k}$, and choose v_i with the minimum value.*
5. *If $v_i < v_{min}$, then $v_{min} \leftarrow v_i$ and $\mathcal{F}_i \leftarrow \mathcal{F}_i - \{\Phi_a\}$, otherwise undo the last move.*
6. *If there was a move, then go to step 2, else return $\Pi = \{\mathcal{F}_0, \mathcal{F}_1, \ldots, \mathcal{F}_{p-1}\}$.*

Algorithm 5.7 tries to reduce the amount of memory by moving an element in $\mathcal{F}_i$ to another block, and tries to make the sizes of supports for $\mathcal{F}_i$ equal.

Algorithm 5.8 *(Input variable grouping for OR-partitioning.)*

1. *Let $(x_1 x_2 \cdots x_n)$ be the BDD variable order. Let $r = \lceil \frac{n}{4} \rceil$.*
2. *Do step 3 while $r < \max_{i=0}^{p-1}\{|support(\mathcal{F}_i)|\}$.*
3. *Let $X_1 = (x_1, x_2, \ldots, x_r)$. Obtain μ. Apply Algorithms 5.6 and 5.7 to obtain the partition $\Pi = \{\mathcal{F}_0, \mathcal{F}_1, \ldots, \mathcal{F}_{p-1}\}$. Let $Size(r) = 2^r \cdot u + \sum_{i=0}^{p-1} 2^{n_2(i)}$, where $n_2(i) = |support(\mathcal{F}_i)| + u$ and $u = \lceil \log_2 \mu \rceil$. Let $r \leftarrow r + 1$.*
4. *Find r that minimizes $Size(r)$. Return the partition $\Pi = \{\mathcal{F}_0, \mathcal{F}_1, \ldots, \mathcal{F}_{p-1}\}$ that makes $Size(r)$ minimum.*

VI. Experimental Results

We compared the sizes of memory for the following three realizations; the first one uses conventional disjoint decomposition of the form $f(X_1, X_2) = g(h(X_1), X_2)$; the second one uses the Shannon expansion obtained by Algorithm 5.4; the third one uses OR-partitioning obtained by Algorithm 5.8. Variable orderings for BDDs were obtained using the algorithm in [5, 15]. Multiple-output functions were represented by BDDs for Encoded Characteristic Function for Non-zeros (ECFNs) with the natural encodings of outputs [20]. The auxiliary variables were placed on top of the variables. We decomposed 44 benchmark functions. In Table I, *Name* denotes the function name; *In* denotes the number of the input variables; *Out* denotes the number of the output variables; *ECFN In* denotes the number of the input variables for ECFN; *Mono* denotes the decomposition $f(X_1, X_2) = g(h(X_1), X_2)$ obtained by Algorithm 5.1; *Shannon* denotes the Shannon decomposition obtained by Algorithm 5.4; *OR-partitioning* denotes OR-partitioning obtained by Algorithm 5.8; $|X_1|$ denotes the number of the variables in X_1; $|X_2|$ denotes the number of the variables in X_2; μ denotes the column multiplicity of the decomposition $f(X_1, X_2)$; $\log_2$ denotes the binary logarithm of the required amount of memory; and *CPU* denotes the CPU time for Algorithm 5.8. Note that the memory size for the OR-partitioning and the Shannon expansions does not contain the memory for OR gates nor MUXs. For the Shannon expansions and OR-partitioning, we decomposed the functions with $p = 4$.

For *des*, *C7552*, *C5315*, *apex6*, *i3*, and *i10*, OR-partitioning drastically reduced required amount of memory. For example, *des* is a 256-input 245-output function, but each output depends on at most 19 variables. OR-partitioning reduced the required memory for *des* by $1/2^{65}$. For *C432* and *rckl*, OR-partitioning could not reduce the size of memory, and for *C499*, OR-partitioning increased the memory size.

When the support size of a block decreases in Algorithm 5.4, the Shannon expansion reduces the memory size. In about one quarter of all cases, the Shannon expansion produced smaller circuits than OR-partitioning. However, in such cases, the degree of reduction was relatively small. In general, we observe that OR-partitioning produced smaller networks than the Shannon expansion.

To see the quality of Algorithm 5.6, we also developed an exact algorithm for OR-p-partition, and applied to ex4, vg2, rckl, x1dn, and x9dn. For vg2, the exact method required 163840 bits, while Algorithm 5.6 required 172032 bits. For other four functions, Algorithm 5.6 produced exact solutions.

VII. Conclusion

In this paper, we presented OR-partitioning of logic function, a new method to realize undecomposable logic functions. The merits of this method are:

1) It often produces smaller circuits than the Shannon expansion.
2) It produces faster circuits than the Shannon expansion, because an OR gate is faster than a multiplexer.

Unfortunately, not all functions have an OR-partitioning.

Acknowledgments

This research is partially supported by the Aid for Scientific Research from the Japan Society for the Promotion of Science (JSPS), and a grant from the Takeda Foundation. Discussion with Dr. Alan Mishchenko improved the presentation of the paper.

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] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE TC*, Vol. C-35, No. 8, pp. 677-691, Aug. 1986.

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

[4] Ting-Ting Hwang, R. M. Owens, M. J. Irwin, and Kuo Hua Wang, "Logic synthesis for field-programmable gate arrays," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, Vol. 13, No. 10, pp. 1280-1287, Oct. 1994.

[5] N. Ishiura, H. Sawada, and S. Yajima, "Minimization of binary decision diagrams based on exchange of variables," *ICCAD-91*, pp. 472-475, Nov. 1991.

[6] 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.

[7] 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.

[8] C. L. Liu, *Introduction to Combinatorial Mathematics*, McGraw-Hill, 1968.

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

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

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

[13] R. Murgai, F. Hirose, and M. Fujita, "Logic synthesis for a single large look-up table," *Proc. International Conference on Computer Design*, pp. 415-424, Oct. 1995.

[14] S. Muroga, *Logic Design and Switching Theory*, John Wiley & Sons, 1979, p. 541-549.

[15] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," *Proceedings of the IEEE International Conference on Computer-Aided Design*, pp. 42-47, Santa Clara, CA, November 1993.

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

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

[18] T. Sasao, "Totally undecomposable functions: applications to efficient multiple-valued decompositions," *IEEE International Symposium on Multiple-Valued Logic*, pp. 59-65, Freiburg, Germany, May 20-23, 1999.

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

[20] 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.

[21] S. Hassoun and T. Sasao, *Logic Synthesis and Verification*, Kluwer Publishers, (2001-11).

[22] 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.

[23] 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.

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

[25] S. Yang, *Logic Synthesis and Optimization Benchmark User Guide Version 3.0*, MCNC, Jan. 1991.

[26] S. Yamashita, H. Sawada, and A. Nagoya, "An efficient method to find an optimal bi-decomposition," *IEICE Trans. Fundamentals*, Vol. E81-A, No. 12, pp. 2529-2537, Dec. 1998.

TABLE I
COMPARISON OF AMOUNT OF MEMORY FOR THREE METHODS.

Name	In	Out	ECFN	Mono				Shannon				OR-Partitioning				
			In	$\|X_1\|$	$\|X_2\|$	μ	$\log_2$	$\|X_1\|$	$\|X_2\|$	μ	$\log_2$	$\|X_1\|$	$\|X_2\|$	μ	$\log_2$	CPU
C1908	33	25	38	22	16	553	26.7	22	16	553	26.7	21	17	586	26.2	0.7
C2670	233	140	241	123	118	99	126.5	104	137	41	107.2	108	133	33	112.5	0.8
C3540	50	22	55	32	23	5156	36.9	32	23	5156	36.9	27	28	4250	33.4	18.5
C432	36	7	39	22	17	81	25.5	21	18	94	24.6	21	18	94	25.5	0.1
C499	41	32	46	27	19	1134	31.2	26	20	1358	30.8	26	20	1358	31.4	1.3
C5315	178	123	185	95	90	212	99.0	94	91	221	98.0	49	136	147	56.1	6.3
C7552	207	108	214	71	143	125	150.0	108	106	95	112.1	71	143	125	91.0	1.4
C880	60	26	65	35	30	466	39.6	35	30	466	39.0	24	41	253	34.4	0.5
apex1	45	45	51	28	23	106	31.5	27	24	122	30.5	20	31	193	24.9	0.5
apex3	54	50	60	33	27	123	36.2	23	37	169	28.4	26	34	163	31.7	1.6
apex5	117	88	124	64	60	104	67.9	63	61	106	66.9	30	94	133	46.0	3.9
apex6	135	99	142	73	69	103	76.9	72	70	103	76.2	33	109	108	39.5	3.8
apex7	49	37	55	29	26	51	32.8	27	28	51	30.4	19	36	51	24.7	0.2
b9	41	21	46	24	27	29	27.7	23	23	30	26.8	14	32	31	19.2	0.1
cps	24	109	31	18	13	157	22.0	16	15	157	20.6	15	16	190	20.8	0.1
dalu	75	16	79	41	38	55	44.8	41	38	55	44.8	28	51	95	32.0	2.3
des	256	245	264	135	129	243	138.6	134	130	244	138.0	66	198	316	72.7	51.6
duke2	22	29	27	15	12	50	18.8	14	13	51	17.8	11	16	62	17.3	0.1
e64	65	65	72	38	34	36	41.3	37	35	37	40.3	36	36	38	40.0	0.1
ex4	128	28	133	68	65	52	71.8	54	79	62	57.2	19	114	15	27.6	0.2
exep	30	63	36	20	16	59	23.3	18	18	50	21.4	17	19	52	21.5	0.1
frg2	143	139	151	78	73	185	82.0	75	76	197	79.0	38	113	208	53.8	10.4
i10	257	224	265	137	128	2324	141.3	135	130	2114	139.8	95	170	724	111.0	149.1
i2	201	1	201	101	100	4	103.0	100	101	4	102.0	99	102	4	102.6	0.1
i3	132	6	135	68	67	6	70.8	68	67	6	70.2	20	115	7	34.2	0.1
i8	133	81	140	73	67	277	77.1	69	71	279	73.4	53	87	304	58.8	11.8
ibm	48	17	53	28	25	39	31.8	27	26	49	31.5	21	32	63	27.3	0.2
jbp	36	57	42	24	18	59	26.8	22	20	85	25.7	13	29	89	20.0	0.2
k2	45	45	51	27	24	119	31.5	26	25	123	29.5	20	31	193	24.9	0.5
mainpla	27	54	33	19	14	162	23.0	18	15	173	22.4	15	18	186	21.5	0.2
mark1	20	31	25	14	11	23	17.2	11	14	28	15.1	11	14	28	15.3	0.1
rckl	32	7	35	18	17	14	21.6	17	18	14	20.6	18	17	14	21.6	0.1
rot	135	107	142	75	67	565	78.8	73	69	1018	76.9	55	87	192	60.3	4.9
seq	41	35	47	26	21	107	29.5	25	22	116	28.5	16	31	133	21.7	0.3
shift	19	16	23	12	11	13	15.6	12	11	13	15.6	6	17	16	11.4	0.1
signet	39	8	42	23	19	214	27.6	23	19	214	26.8	19	23	156	25.1	0.3
too_large	38	3	40	21	19	23	24.7	20	20	23	23.7	19	21	23	23.5	0.1
ts10	22	16	26	14	12	25	17.7	14	12	25	17.7	8	18	17	13.2	0.1
vg2	25	8	28	16	12	8	17.8	12	16	8	15.8	13	15	10	17.4	0.1
x1dn	27	6	30	14	16	7	19.1	14	16	7	16.5	15	15	13	18.2	0.1
x2dn	82	56	88	46	42	34	49.3	45	43	36	48.5	23	65	44	28.4	0.3
x6dn	39	5	42	23	19	31	25.8	21	21	39	24.8	12	30	40	18.5	0.1
x9dn	27	7	30	16	14	19	19.7	13	17	7	16.8	14	16	13	17.7	0.1
xparc	41	73	48	26	22	156	30.6	25	23	157	29.6	21	27	156	26.6	0.2

μ: Column multiplicity of decomposition.
$\log_2$: Binary logarithm of required amount of memory.
CPU: CPU time (sec).
Mono: The case of using only conventional decomposition.
Shannon: The Shannon expansion of Algorithm 5.4 into four parts.
OR-partitioning: The OR-partitioning of Algorithm 5.8 into four parts.
IBM PC/AT compatible, PentiumIII 1GHz, 4GBytes main memory.