Linear Decomposition of Index Generation Functions

Tsutomu Sasao
Kyushu Institute of Technology
Iizuka 820-8502, Japan

Abstract— This paper shows a heuristic method to reduce the number of variables to represent incompletely specified index generation functions using linear decompositions. To find good linear transformations, two measures are introduced: the imbalance measure and the ambiguity measure. Experimental results using m-out-of-n code to binary converters, randomly generated functions, IP address tables, and lists of English words show the usefulness of the approach.

I. Introduction

Let n be the number of the variables, then the size of lookup table (LUT) to implement the function is proportional to 2^n. When n is large, a realization of the function by a single LUT is, in many cases, impractical. **Functional decomposition** [1, 3] is a technique to realize a given function using smaller LUTs by the circuit shown in Fig. 1. Since the advent of FPGAs in 1985, the decomposition technique has been improved: Efficient methods to find decompositions have been developed [2, 6, 10]. However, the basic scheme remains unchanged.

Linear decomposition shown in Fig. 2 is a different type of decomposition. In this case, L realizes a linear function, while G realizes a general (often non-linear) function. Nechiporuk [8] first presented this scheme in 1958. Later Lechner [7] enhanced the theory, and Varma-Trachtenberg [18] proposed a method to find a linear transformation using autocorrelation functions. Unfortunately, the effect of linear decompositions is not so impressive for general functions.

In this paper, we use linear decompositions to realize functions for which only k combinations of inputs are defined. For this class of functions, the circuit sizes can be reduced drastically with linear decompositions. To implement the linear part, we use a special programmable circuit that consists of registers, multiplexers, and EXOR gates (details are shown in Section IV). To implement the general part, we use an LUT. Thus, Fig. 2 can be a reconfigurable circuit.

This scheme is useful for communication applications where frequent changes are necessary [14, 15, 16], but cannot be treated by a conventional functional decomposition. Given an incompletely specified function, the problem of the linear decomposition is to obtain a linear transformation that minimizes the number of variables p for the general part. When a given function is defined for only k input combinations, the number of variables can be reduced to $2\lceil \log_2 k \rceil - 3$, in many cases, and by this, the size of the LUT is reduced drastically.

The rest of the paper is organized as follows: Section 2 defines index generation functions and shows their properties; Section 3 shows a method to minimize the number of variables to represent incompletely specified functions; Section 4 shows a method to reduce the number of variables to represent index generation function using linear transformations; Section 5 introduces m-out-of-n to binary code converters; Section 6 shows experimental results; and Section 7 concludes the paper.

II. Index Generation Function

Definition 1 *[13, 17] Consider a set of k different vectors of n bits. These vectors are* **registered vectors**. *For each registered vector, assign a unique integer from 1 to k. A* **registered vector table** *shows an* **index** *for each reg-*

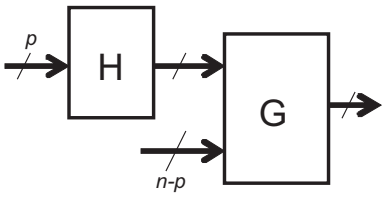

Fig. 1. Conventional Functional Decomposition

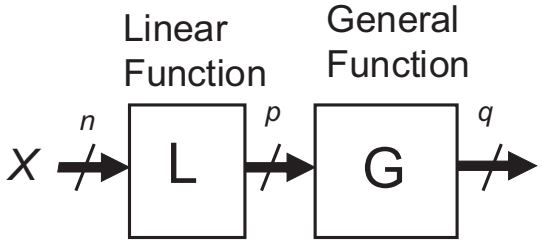

Fig. 2. Linear Decomposition

978-1-4673-0772-7/12/$31.00 ©2012 IEEE

TABLE 1
REGISTERED VECTOR TABLE

Vector							Index
x_7	x_6	x_5	x_4	x_3	x_2	x_1	
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	2
0	0	0	0	1	0	0	3
0	0	0	1	0	0	0	4
0	0	1	0	0	0	0	5
0	1	0	0	0	0	0	6
1	0	0	0	0	0	0	7

istered vector. An **incompletely specified index generation function** *produces a corresponding index when the input vector matches a registered vector. Otherwise, the value of the function is undefined (don't care). In this case, k is the* **weight** *of the function. The incompletely specified index generation function represents a mapping $D \rightarrow \{1, 2, \ldots, k\}$, where $D \subset B^n$ denotes the set of registered vectors.*

Example 1 *Consider the registered vectors shown in Table 1. These vectors show an index generation function with weight $k = 7$.* ■

Index generation functions are useful for address tables for the internet, terminal access controller of local area networks, database, memory patch circuits, text compression, password tables, and code converters.

Table 2 shows the average number of variables p to represent random incompletely specified index generation functions of n variables with weight k. From this, we have the following [16]:

Conjecture 1 *When the number of the input variables is sufficiently large, most incompletely specified index generation functions with weight k (≥ 7) can be represented by $p = 2\lceil \log_2(k+1) \rceil - 3$ variables.*

To derive Table 2, we minimized 30,000 random functions in total. Among them, only 233 functions required one more variables than the bound given by Conjecture 1. In other words, the *exceptions* that do not satisfy the conjecture are only 0.77 percents. For most functions, the necessary number of variables p depends on only k, and is independent of n, the number of variables in the original functions. For example, most index generation functions with weight $k = 7$ can be represented by three variables. Of course, there exist exceptions. An example of the *exceptions* that do not satisfy Conjecture 1 is as follows:

Example 2 *Consider the registered vectors shown in Table 1. It shows an index generation function with weight $k = 7$. To distinguish these seven vectors, 6 variables are necessary.* ■

TABLE 2
THE AVERAGE NUMBER OF VARIABLES p TO REPRESENT INDEX GENERATION FUNCTIONS OF n VARIABLES WITH WEIGHT k.

k	$n = 16$	$n = 20$	$n = 24$	UB
7	3.052	3.018	3.003	3
15	4.980	4.947	4.878	5
31	6.447	6.115	6.003	7
63	8.257	8.007	8.000	9
127	10.304	10.000	9.963	11
255	12.589	11.996	11.896	13
511	14.890	14.019	13.787	15
1023	15.991	16.293	15.874	17
2047	16.000	18.758	17.965	19
4095	16.000	19.992	20.093	21

$UB = 2\lceil \log_2(k+1) \rceil - 3$.

However, for such functions, as shown in the next section, by linear transformations, the number of variables can be reduced. Thus, when we use linear transformations, almost all functions can be realized with the number of variables given by Conjecture 1.

As for the lower bound on the number of variables, we have the following:

Theorem 1 *To represent any incompletely specified index generation function f with weight k, at least $q = \lceil \log_2 k \rceil$ variables are necessary.*

(Proof) The number of different vectors specified with $q-1$ variables is at most $2^{q-1} < k$. Thus, at least q variables are necessary to represent an index generation function with weight k. □

III. A METHOD TO MINIMIZE THE NUMBER OF VARIABLES

To make the paper self-contained, in this section, we review a method to represent an incompletely specified index generation function by using the minimum number of variables [4, 5, 12]. The idea is illustrated by the following:

Example 3 *Let us minimize the number of variables to represent the index generation function whose Karnaugh map is shown in Fig. 3. In the map, blank cells denote don't cares.*

1. *Let the four vectors be $\vec{a}_1 = (1, 0, 0, 1)$, $\vec{a}_2 = (1, 1, 1, 1)$, $\vec{a}_3 = (0, 1, 0, 1)$, and $\vec{a}_4 = (1, 1, 0, 0)$.*
2. *To distinguish $\vec{a}_1$ and $\vec{a}_2$, either x_2 or x_3 is necessary. From this, we have the condition $x_2 \vee x_3 = 1$, where $x_i = 1$ shows that x_i appears in the expression. In other words, $x_2 \vee x_3 = 1$ shows that either x_2 or x_3 is necessary. Similarly, to distinguish $\vec{a}_1$ and $\vec{a}_3$, the condition $x_1 \vee x_2 = 1$ is necessary; to distinguish*

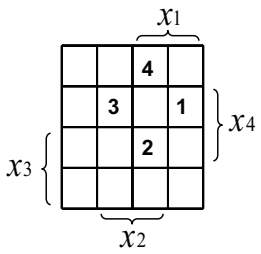

Fig. 3. Index Generation Function of 4 variables.

$\vec{a}_1$ *and* $\vec{a}_4$*, the condition* $x_2 \vee x_4 = 1$ *is necessary; to distinguish* $\vec{a}_2$ *and* $\vec{a}_3$*, the condition* $x_1 \vee x_3 = 1$ *is necessary; to distinguish* $\vec{a}_2$ *and* $\vec{a}_4$*, the condition* $x_3 \vee x_4 = 1$ *is necessary; and to distinguish* $\vec{a}_3$ *and* $\vec{a}_4$*, the condition* $x_1 \vee x_4 = 1$ *is necessary.*

3. *To distinguish all the vectors, all the conditions must hold at the same time. This is represented as* $R = 1$*, where*

$$R = (x_2 \vee x_3)(x_1 \vee x_2)(x_2 \vee x_4)(x_1 \vee x_3)(x_3 \vee x_4)(x_1 \vee x_4).$$

4. *To* R*, apply the distributive and the absorption laws, and obtain the following relation:*

$$R = x_1x_2x_4 \vee x_1x_2x_3 \vee x_2x_3x_4 \vee x_1x_3x_4.$$

5. *Since each product consists of three literals, each corresponds a minimum solution. Thus,* f *can be represented with three variables. Since no variable appears in all products, no variable is essential.*

6. *The expression for the original function is*

$$f = 1 \cdot x_1\bar{x}_2\bar{x}_3x_4 \vee 2 \cdot x_1x_2x_3x_4 \vee 3 \cdot \bar{x}_1x_2\bar{x}_3x_4 \vee 4 \cdot x_1x_2\bar{x}_3\bar{x}_4.$$

To obtain the expression corresponding to the first product $x_1x_2x_4$ *in Step 4, remove the literals for* x_3*:*

$$f = 1 \cdot x_1\bar{x}_2x_4 \vee 2 \cdot x_1x_2x_4 \vee 3 \cdot \bar{x}_1x_2x_4 \vee 4 \cdot x_1x_2\bar{x}_4.$$

■

In principle, the number of variables to represent an incompletely specified index generation function can be minimized by using the following [14, 16]:

Algorithm 1 *(Minimization of Variables)*

1. *Let* D *be the set of vectors* $\vec{a}_i$*, such that* $f(\vec{a}_i) = i$*, where* $i = 1, 2, \ldots, k$

2. *For each pair of vectors* $\vec{a}_i = (a_1, a_2, \ldots, a_n) \in D$ *and* $\vec{b}_j = (b_1, b_2, \ldots, b_n) \in D$*, associate a sum defined by* $s(i,j) = \bigvee_{r=1}^{n} y_r$*, where* $y_r = 0$ *if* $a_r = b_r$ *and* $y_r = x_r$ *if* $a_r \neq b_r$*, where* $r = 1, 2, \ldots, n$*. Note that there are* $k(k-1)/2$ *pairs.*

3. *Define a covering function* $R = \bigwedge_{i<j} s(i,j)$*.*

4. *Represent* R *by a minimum sum-of-products expression.*

5. *The product with the fewest literals corresponds to the minimum solution.*

6. *Derive an expression with the minimum number of variables.*

In Algorithm 1, Steps 4, 5 and 6 compute a minimum covering. By first detecting the essential variables, we can reduce the computational time to derive the covering function.

IV. Reduction of the Number of Variables By Linear Transformations

In the previous section, we showed a standard method to reduce the number of variables for incompletely specified functions. Unfortunately, the minimization ability of such method is limited. In this section, we show that more variables can be reduced by using linear transformations.

Example 4 *By Algorithm 1, the number of variables for the function in Table 1 can be reduced to six: Any one variable can be removed. If we remove* x_7*, then we have:*

$$\begin{aligned} f = 1 \cdot x_1\bar{x}_2\bar{x}_3\bar{x}_4\bar{x}_5\bar{x}_6 \vee 2 \cdot \bar{x}_1x_2\bar{x}_3\bar{x}_4\bar{x}_5\bar{x}_6 \\ \vee 3 \cdot \bar{x}_1\bar{x}_2x_3\bar{x}_4\bar{x}_5\bar{x}_6 \vee 4 \cdot \bar{x}_1\bar{x}_2\bar{x}_3x_4\bar{x}_5\bar{x}_6 \\ \vee 5 \cdot \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4x_5\bar{x}_6 \vee 6 \cdot \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4\bar{x}_5x_6 \\ \vee 7 \cdot \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4\bar{x}_5\bar{x}_6. \end{aligned}$$

■

Definition 2 *A* **compound variable** *has a form* $y = c_1x_1 \oplus c_2x_2 \oplus \cdots \oplus c_nx_n$*, where* $c_i \in \{0, 1\}$*. The* **compound degree** *of* y *is* $\sum_{i=1}^{n} c_i$*, where* c_i *is viewed as an integer and* $\sum$ *denotes an ordinary integer addition. A* **primitive variable** *is one with compound degree one.*

Fig. 4 shows a circuit to generate compound variables with degree two. It performs a linear transformation $y_i = x_i \oplus x_j$ or $y_i = x_i$, where $i \neq j$. It uses a pair of multiplexers for each variable y_i. The upper multiplexers have the inputs $x_1, x_2, \ldots, x_n$. The register with $\lceil \log_2 n \rceil$ bits specifies which variable to select by the multiplexer. The lower multiplexers have the inputs $x_1, x_2, \ldots, x_n$, except for x_i. For the i-th input, the constant input 0 is connected instead of x_i. By setting $y_i = x_i \oplus 0$, we can implement $y_i = x_i$.

We can easily design a circuit for compound variables with a higher degree. A circuit for degree t consists of t modules of $\lceil \log_2 n \rceil$-bit registers, t modules of n-input multiplexers, and a t-input EXOR gate. The cost of the circuit is $O(tn \log n)$. When we fix the value of t, the cost of the linear part is $O(pn \log n)$

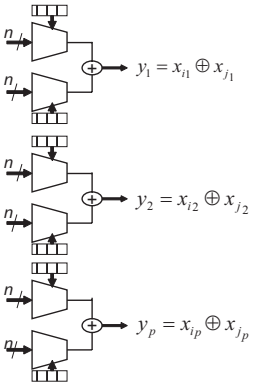

Fig. 4. Circuit for Compound Variables with Degree Two.

TABLE 3
Registered Vectors after Linear Transformation

Vector			Index
y_3	y_2	y_1	
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

Definition 3 *Given an incompletely specified index generation function, a linear transformation that minimizes the number of variables is* **optimum**.

By Theorem 1, if the linear transformation reduce the number of variables to $q = \lceil \log_2 k \rceil$ variables, then it is an optimum. A brute force way to find an optimum transformation is first to construct the compound variables whose degrees are t or less than t. The number of such variables is $\sum_{i=1}^{t} \binom{n}{i}$. Then, apply the method shown in Section III. However, such method takes too much computation time, and is impractical.

Example 5 *To represent the function in Table 1 with fewer variables, consider the linear transformation:*

$$\begin{aligned} y_1 &= x_1 \oplus x_3 \oplus x_5 \oplus x_7, \\ y_2 &= x_2 \oplus x_3 \oplus x_6 \oplus x_7, \\ y_3 &= x_4 \oplus x_5 \oplus x_6 \oplus x_7. \end{aligned}$$

We have the registered vectors shown in Table 3. In this case, three variables (y_3, y_2, y_1) distinguish 7 vectors. ■

As shown in this example, with a linear transformation, we can often reduce the number of variables to represent the function. Why does this linear transformation reduce the number of variables? Fig. 5 shows the decision tree for the function in Table 1, while Fig. 6 shows the decision tree for the function in Table 3. To distinguish 7 vectors,

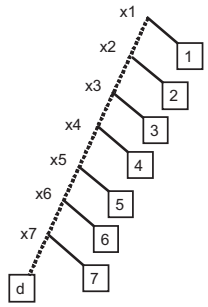

Fig. 5. Unbalanced Decision Tree.

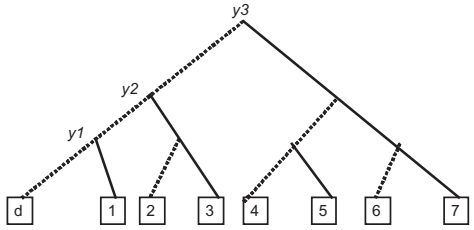

Fig. 6. Balanced Decision Tree.

the tree in Fig. 5 requires 6 variables, while the tree in Fig. 6 requires only three variables. In other words, a more balanced decision tree produces an expression with fewer variables.

To obtain the linear transformation that produces a more balanced decision tree, we define a measure showing the imbalance of the tree.

Definition 4 *In the registered vector table, let $\nu(x_i, 0)$ be the number of vectors with $x_i = 0$, and let $\nu(x_i, 1)$ be the number of vectors with $x_i = 1$. The* **imbalance measure** *of the function with respect to x_i is defined as*

$$\omega(x_i) = \nu(x_i, 0)^2 + \nu(x_i, 1)^2.$$

In the variable x_i, when the numbers of occurrences of 0's and 1's are the same, $\omega(x_i)$ takes its minimum. The larger the difference of the occurrences of 0's and 1's, the larger the imbalance measure. Let k be the number of registered vectors. Then, $\nu(x_i, 0) + \nu(x_i, 1) = k$.

Example 6 *In Table 1, since, for all x_i, $\nu(x_i, 0) = 6$ and $\nu(x_i, 1) = 1$, we have*

$$\omega(x_i) = \nu(x_i, 0)^2 + \nu(x_i, 1)^2 = 6^2 + 1^2 = 37.$$

In Table 3, since $\nu(x_i, 0) = 3$ and $\nu(x_i, 1) = 4$, we have

$$\omega(x_i) = \nu(x_i, 0)^2 + \nu(x_i, 1)^2 = 3^2 + 4^2 = 25.$$

In other words, the linear transformation in Example 5 reduces the imbalance measure, and improves the balance of the decision tree. ■

TABLE 4
INDEX GENERATION FUNCTION

x_1	x_2	x_3	x_4	x_5	f
0	0	0	0	0	1
0	1	0	1	0	2
0	1	1	1	0	3
1	1	1	0	0	4
1	0	0	1	1	5
1	0	1	1	1	6
1	1	1	0	1	7

When the imbalance measure is large, the reduction of variables tends to be difficult. However, if a linear transformation reduces the imbalance measure, then we may reduce more variables.

Definition 5 *Let $f(x_1, x_2, \ldots, x_n)$ be an incompletely specified index generation function with weight $|f|$. Let $\vec{x} = (x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(t)})$ be a vector consisting of a subset of the variables $\{x_1, x_2, \ldots, x_n\}$, where π denotes a permutation of $\{1, 2, \ldots, n\}$. Let $N(f, \vec{x}, \vec{a})$ be the number of registered vectors of f that takes non-zero values, when the values of $\vec{x}$ are set to $\vec{a} = (a_1, a_2, \ldots, a_t)$, $a_i \in \{0, 1\}$. The* **ambiguity** *of f with respect to $\vec{x}$ is defined as*

$$AMB(\vec{x}) = -|f| + \sum_{\vec{a} \in B^t} N(f, \vec{x}, \vec{a})^2.$$

Example 7 *Consider the index generation function shown in Table 4. Assume that the values of (x_1, x_2, x_3) are changed as $(0,0,0)$,$(0,0,1)$,$(0,1,0)$, $(0,1,1)$,$(1,0,0)$,$(1,0,1)$, $(1,1,0)$,$(1,1,1)$, in this order. Then, the values of f change as follows:*

$$[1], [d], [2], [3], [5], [6], [d], [4, 7],$$

where d denotes undefined or don't care. In this case, the ambiguity with respect to (x_1, x_2, x_3) is

$$\begin{aligned} & AMB(x_1, x_2, x_3) \\ = \; & -7 + (1^2 + 0^2 + 1^2 + 1^2 + 1^2 + 1^2 + 0^2 + 2^2) = 2 \end{aligned}$$

When $(x_1, x_2, x_3) = (0, 0, 1)$, the value of f is **undefined**, *while when $(x_1, x_2, x_3) = (1, 1, 1)$, the value of f is* **ambiguous**, *since f can be either 4 or 7.*

Next, let the variable set be (x_1, x_3, x_5). Similarly, the values of f change as follows:

$$[1, 2], [d], [3], [d], [d], [5], [4], [6, 7].$$

In this case, the ambiguity with respect to (x_1, x_3, x_5) is

$$\begin{aligned} & AMB(x_1, x_3, x_5) \\ = \; & -7 + (2^2 + 0^2 + 1^2 + 0^2 + 0^2 + 1^2 + 1^2 + 2^2) = 4 \end{aligned}$$

When $(x_1, x_3, x_5) = (0, 0, 0)$ and $(1, 1, 1)$, the values of f are ambiguous.

Finally, let the variable set be (x_3, x_4, x_5). Similarly, the values of f change as follows:

$$[1], [d], [2], [5], [4], [7], [3], [6].$$

In this case, the ambiguity with respect to (x_3, x_4, x_5) is

$$\begin{aligned} & AMB(x_3, x_4, x_5) \\ = \; & -7 + (1^2 + 0^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2) = 0. \end{aligned}$$

Note that f can be represented with only (x_3, x_4, x_5). ■

Theorem 2 *$AMB(\vec{x}) = 0$ iff $\vec{x}$ can represent f.*

(Proof) Let D be the set of registered vectors for f and let $\tilde{D}$ be the set of vectors consisting of the variables for $\vec{x}$.
($\Rightarrow$) We prove this by contradiction. Assume that $\vec{x}$ cannot represent f. Two cases are possible:

1. f is undefined for some $\vec{a} \in \tilde{D}$. In this case, $N(f, \vec{x}, \vec{a}) = 0$. However, this cannot happen, since $\vec{a}$ is a subset of a registered vector.

2. f is ambiguous for some $\vec{a} \in \tilde{D}$. In this case, $N(f, \vec{x}, \vec{a}) \geq 2$. Since $\sum N(f, \vec{x}, \vec{a})^2 > |f|$, we have $AMB(\vec{a}) > 0$.

From these, for each $\vec{a} \in \tilde{D}$, the value of f is uniquely defined for any $\vec{a} \in B^t$. Thus, f can be represented with $\vec{x}$.
($\Leftarrow$) Assume that f is represented with $\vec{x}$. In this case, the value of f is uniquely defined or undefined. This implies that $N(f, \vec{x}, \vec{a}) = 1$ for all $\vec{a} \in \tilde{D}$. From this, we have $AMB(\vec{x}) = -|f| + \sum_{\vec{a} \in \tilde{D}} 1^2 = 0$, since, $|f| = \sum_{\vec{a} \in \tilde{D}} 1$. □

By using these two measures, we have a heuristic algorithm to reduce the number of variables. In this algorithm, the imbalance measure is used to guide the linear transformation. The compound variable is chosen to minimize the imbalance measure in a greedy manner. Then the ambiguity measure (AMB) is tested. When the AMB is greater than 0, more *resolution* is required to include more compound variables. This process stops when the function resolution is high enough to make AMB=0.

Algorithm 2 *(Heuristic Method to Find a Good Linear Transformation)*

1. *Let the input variables be $x_1, x_2, \ldots, x_n$. Let $t \geq 2$ be the maximal compound degree.*

2. *Generate the compound variables y_i whose compound degrees are t or less than t. The number of such compound variables is $\sum_{i=1}^{t} \binom{n}{i}$. Let T be the set of compound variables.*

3. *Let y_1 be the variable with the smallest imbalance measure. Let $\vec{Y} \leftarrow (y_1)$, $T \leftarrow T - y_1$.*

TABLE 5
REGISTERED VECTOR TABLE FOR 2-OUT-OF-6 CODE TO BINARY CONVERTER.

2-out-of-6 code						Index	After Linear Transformation			
x_6	x_5	x_4	x_3	x_2	x_1		y_4	y_3	y_2	y_1
0	0	0	0	1	1	1	0	0	0	1
0	0	0	1	0	1	2	0	0	1	1
0	0	0	1	1	0	3	0	0	1	0
0	0	1	0	0	1	4	0	1	1	0
0	0	1	0	1	0	5	0	1	1	1
0	0	1	1	0	0	6	0	1	0	1
0	1	0	0	0	1	7	1	1	0	0
0	1	0	0	1	0	8	1	1	0	1
0	1	0	1	0	0	9	1	1	1	1
0	1	1	0	0	0	10	1	0	1	0
1	0	0	0	0	1	11	1	0	0	0
1	0	0	0	1	0	12	1	0	0	1
1	0	0	1	0	0	13	1	0	1	1
1	0	1	0	0	0	14	1	1	1	0
1	1	0	0	0	0	15	0	1	0	0

4. *While* $AMB(\vec{Y}) > 0$, *find the variable* y_j *in* T *that minimizes the value of* $AMB(\vec{Y}, y_j)$. *Let* $\vec{Y} \leftarrow (\vec{Y}, y_j)$, $T \leftarrow T - y_j$.

5. *Stop.*

This algorithm constructs a table with $O(n^t)$ columns and k rows. So, it is impractical when $t > 5$. For small t, this algorithm obtains fairly good solutions in a short time.

V. Code Converters

Here, we consider a class of code converters that can be implemented as index generation functions.

Definition 6 *An* ***m*-out-of-*n* code** *consists of all* $k = \binom{n}{m}$ *binary codes words whose weights are* m.

Definition 7 *An* ***m*-out-of-*n* code to binary converter** *realizes the index generation function with weight* $k = \binom{n}{m}$. *It has* n *inputs and* $\lceil \log_2 \binom{n}{m} + 1 \rceil$ *outputs. An* m*-out-of-*n *code is produced in increasing order of binary numbers. That is, the minimum number is denoted by* $(0, 0, \ldots, 0, 1, 1, \ldots, 1)$, *while the maximum number is denoted by* $(1, 1, \ldots 1, 0, 0, \ldots, 0)$.

m-out-of-n codes are used in asynchronous logic circuits.

Example 8 *Table 1 shows the* 1*-out-of-*7 *code to binary converter.* ■

Example 9 *Consider the* 2*-out-of-*6 *code to binary converter shown in Table 5. Let us reduce the number of variables by using a linear transformation. The imbalance measure of the variable* x_i *is* $10^2 + 5^2 = 125$, *for all* i. *Now, consider the linear transformation:*

$$\begin{aligned} y_4 &= x_6 \oplus x_5, \\ y_3 &= x_5 \oplus x_4, \\ y_2 &= x_4 \oplus x_3, \\ y_1 &= x_3 \oplus x_2. \end{aligned}$$

In the right columns of the Table 5, the variables y_i *are shown. The imbalance measures are reduced to* $8^2 + 7^2 = 113$. *Note that these four-bit patterns show all distinct patterns except for* $(0, 0, 0, 0)$. *This means that* (y_4, y_3, y_2, y_1) *represents the index generation function. Also, by Theorem 1, it is an optimum linear transformation.* ■

TABLE 6
NUMBER OF VARIABLES TO REPRESENT m-OUT-OF-20 CODE TO BINARY CONVERTER.

		Compound Degree t					
m	k	1	2	3	4	5	6
1	20	19	14	10	8	7	6
2	190	19	15	12	10	9	9
3	1140	19	17	14	12	12	**11**
4	4845	19	17	15	15	15	15

VI. Experimental Results

We used two different algorithms according to the value t of the compound degree of the linear transformation. For the realization using primitive variables only, i.e., $t = 1$, we used Algorithm 1. For the realization with $2 \leq t \leq 6$, we used Algorithm 2.

A. *m*-out-of-*n* code

The number of variables to represent an m-out-of-20 code to binary number converter is investigated for different values of t, and for different values of m. Note that the number of registered vectors is $k = \binom{20}{m}$, and by Theorem 1, the function requires at least $q = \lceil \log_2 k \rceil$ variables. Table 6 shows the results of the experiment. When the compound degree is one, all converters required 19 variables. For $m = 1$, $m = 2$ and $m = 3$, with the increase of the compound degree t, the necessary number of variables decreased. For $m = 4$, up to the compound degree three, the necessary number of variables decreased. However, after that, the number of variables could not be reduced. The figure shown in bold face denotes an optimum solution.

B. Random Index Generation Functions

For $n = 24$ and $k = 1023$, we generated 1000 random index generation functions. Fig. 7 shows the results. The vertical axis denotes the number of variables after linear

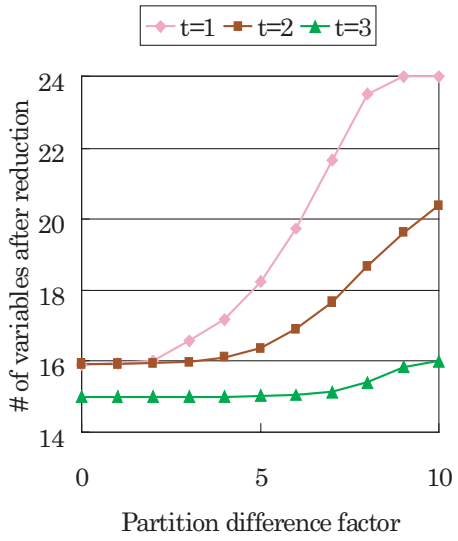

Fig. 7. Number of Variable to Represent Random Index Generation Functions with Weight = 1023 and $n = 24$

transformations. The horizontal axis denotes s, the difference of occurrences of 0's and 1's:

$$s = \frac{|\nu(x_i, 0) - \nu(x_i, 1)|}{64}.$$

Also, t denotes the maximum compound degree of the variables. For $s = 0$, the necessary number of variables to represent function is reduced to 16 when $t = 1$ and $t = 2$. However, for $s = 10$, the necessary number of variables to represent function is 24 when $t = 1$, while it is reduced to 20 when $t = 2$, and it is further reduced to 16 when $t = 3$. Thus, the linear transformations with larger t are more effective when the imbalance measures are large.

C. IP Address Tables

In this experiment, we used distinct IP addresses of computers that accessed our web site over a period of a month. We considered four lists of different values of k. Table 7 shows the results. Note that the original number of variables is 32. The first column shows the number of registered vectors: k. The second column shows the number of variables to represent the function, when only the primitive variables are used (i.e. $t = 1$). The third column shows the number of variables to represent the function, when the variables with compound degrees up to two are used. Other columns show the numbers of variables for different values of t. As shown in Table 7, the LUT size can be reduced when we use compound variables with $t = 3$ or $t = 4$.

TABLE 7
NUMBER OF VARIABLES TO REPRESENT IP ADDRESS TABLE.

	Compound Degree t					
k	1	2	3	4	5	6
1670	18	17	16	16	15	15
3288	20	19	18	17	17	17
4591	21	20	19	18	18	18
7903	23	21	20	20	20	20

TABLE 8
LISTS OF ENGLISH WORDS

		Compound Degree t			
Name	k	1	2	3	4
ListA	1730	31	19	17	16
ListB	3366	31	21	19	17
ListC	4705	37	24	20	19

D. Lists of English Words

To compress English text, we can use a list of frequently used words [9]. We made three lists of English words: *List A*, *List B*, and *List C*. The maximum number of characters in the word lists is 13, but we only consider the first 8 characters. For English words consisting of fewer than 8 letters, we append blanks to make the length of words 8. We represent each alphabetic character by 5 bits. So, in the lists, all the words are represented by 40 bits. The numbers of words in the lists are 1730, 3366, and 4705, respectively. Within each word list, each English word has a unique index, an integer from 1 to k, where $k = 1730$ or 3360 or 4705. The numbers of bits for the indices are 11, 12, and 13, respectively. Table 8 shows the numbers of variables to represent the lists. For these data, imbalance measures are large, and we could reduce many variables by using compound variables with large t.

E. Computation Time

We used a PC with INTEL Core i3, 2.4 MHz, and Windows 7 Professional with 64-bit Operating System. As for the minimization of primitive variables, we used the program of [14], that obtains exact minimum solutions. As for the linear transformations, we used Algorithm 2 that obtains near minimal solutions. For randomly generated functions with $n = 20$ and $k = 127$, the minimization of primitive variables took, on the average, 270 m sec. For the reduction of variables using compound variables, it took, 31, 93, 655, 2839, 12433 m sec, when the maximum compound degrees were $t = 2, 3, 4, 5$, and 6, respectively.

For the 2-out-of-20 code to binary converter, the minimization of primitive variables took 15 ms. For the reduction of variables using compound variables, the CPU times were 46, 202, 873, 3307, 10873 m sec, when the maximum compound degrees were $t = 2, 3, 4, 5$, and 6, respectively.

VII. Concluding Remarks

In this paper, we considered linear decompositions of index generation functions. To find good linear transformations, we introduced two measures: the imbalance measure and the ambiguity measure. We showed a heuristic method to find a linear transformation that reduces the number of variables to represent a given function. When the imbalance measures are large, many variables are necessary to represent the function. However, with linear transformations that reduce imbalance measures of the original variables, we can represent the functions with fewer variables.

As for applications, we showed code converters, random functions, IP address tables, and lists of English words. Most index generation functions are undecomposable by the conventional methods. However, they are decomposed by linear functions to produce efficient memory-based implementations.

The presented design method is useful only for index generation functions whose weights are small. Unfortunately, MCNC benchmark functions do not include such functions. So, the comparison with other method [18] is difficult. Index generation functions are related short-life data such as address tables, password lists, which require frequently updates. Such functions are often implemented by software. However, reconfigurable hardware/ firmware implementations will make them much faster.

Acknowledgments

This research is partly supported by the MEXT Regional Innovation Cluster Program (Global Type, 2nd Stage), and by a Grant in Aid for Scientific Research of the JSPS. The author thanks Prof. Jon T. Butler for discussion and Mr. M. Matsuura for experiments. Comments of reviewers were quite useful to improve the quality of the paper.

References

[1] R. L. Ashenhurst, "The decomposition of switching functions," *International Symposium on the Theory of Switching*, April 1957, pp. 74-116.

[2] V. Bertacco and M. Damiani, "The disjunctive decomposition of logic functions," *ICCAD-97*, Nov. 1997, pp. 78-82.

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

[4] C. Halatsis and N. Gaitanis, "Irredundant normal forms and minimal dependence sets of a Boolean function," *IEEE Transactions on Computers*, Vol. C-27, No. 11, Nov. 1978, pp. 1064-1068.

[5] Y. Kambayashi, "Logic design of programmable logic arrays," *IEEE Trans. on Computers*, Vol. C-28, No. 9, Sept. 1979, pp. 609-617.

[6] Y-T. Lai, M. Pedram and S. B. K. Vrudhula, "BDD based decomposition of logic functions with application to FPGA synthesis," *Design Automation Conference*, June 1993, pp. 642-647.

[7] R. J. Lechner, "Harmonic analysis of switching functions," in A. Mukhopadhyay (ed.), *Recent Developments in Switching Theory*, Academic Press, New York, 1971.

[8] E. I. Nechiporuk, "On the synthesis of networks using linear transformations of variables," *Dokl. AN SSSR*, vol. 123, no. 4, Dec. 1958, pp. 610-612.

[9] D. Salomon, G. Motta, and D. Bryant, *Handbook of Data Compression* (5th edition), Springer, 2009.

[10] T. Sasao, "FPGA design by generalized functional decomposition," In *Logic Synthesis and Optimization*, Sasao ed., Kluwer Academic Publisher, 1993, pp. 233-258.

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

[12] T. Sasao, "On the number of dependent variables for incompletely specified multiple-valued functions," *30th International Symposium on Multiple-Valued Logic*, pp. 91-97, Portland, Oregon, U.S.A., May 23-25, 2000, pp. 91-97.

[13] T. Sasao, "Design methods for multiple-valued input address generators,"(invited paper) *International Symposium on Multiple-Valued Logic* (ISMVL-2006), Singapore, May 2006, pp. 1-10.

[14] T. Sasao, "On the number of variables to represent sparse logic functions," *ICCAD-2008*, San Jose, California, USA, Nov.10-13, 2008, pp. 45-51.

[15] T. Sasao, T. Nakamura, and M. Matsuura, "Representation of incompletely specified index generation functions using minimal number of compound variables," *DSD-2009*, Aug. 2009, pp. 765-772.

[16] T. Sasao, *Memory-Based Logic Synthesis*, Springer, 2011.

[17] T. Sasao, "Index generation functions: Recent developments,"(invited paper) *International Symposium on Multiple-Valued Logic* (ISMVL-2011), Tuusula, Finland, May 23-25, 2011, pp. 1-9.

[18] D. Varma and E. Trachtenberg, "Design automation tools for efficient implementation of logic functions by decomposition," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol.8, No.8, Aug. 1989, pp. 901-916.